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

Group 10: Rebecca Cheng, Eddie Wang

October 9, 2018

1 Problem Representation

1.1 Representation Description

We represented states using a *State* class, with two City properties *city* and *taskCity* (details of implementation will be explained in the following section). The possible actions from each state are to get all neighbours from the city and calculate Q-Values using value iteration. Lastly, for rewards and probability transitions, we simply calculated them using the Task Distribution API. There was no need for them to be put in a table since the methods were very well defined, allowing for optimized use of space in our application.

1.2 Implementation Details

For the implementation of the State class, within each State object are two properties: a *city* field to represent the current city that the agent is at as well as a *taskCity* field to represent the city that the agent has a task to deliver to. Note that *taskCity* may be null, signalling that there is no task at that city when the agent arrives to that State.

As for possible actions, they were represented by State classes using neighboring cities of the current State. Values for each action were stored using a HashMap that mapped every State to a double value. Once the value iteration algorithm converges, all of the best actions are stored in an action table, which is also a HashMap that maps every State to a the next City that the agent will go to from that State.

Lastly, for the reinforcement learning algorithm, we follow the value iteration algorithm as described in the lecture slides. The stopping criteria is reached when the Q-Value calculated for a current state is equal to the max Q-Value, meaning that further iterations of the value iteration algorithm will yield the same maximum Q-Values. Also, when the stopping criteria is reached, the Q-Value table/HashMap and Action table/HashMap is updated to have the final best values and actions respectively.

2 Results

2.1 Experiment 1: Discount factor

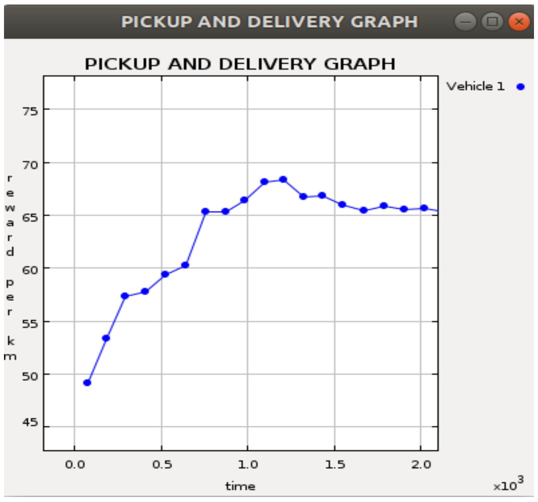
2.1.1 Setting

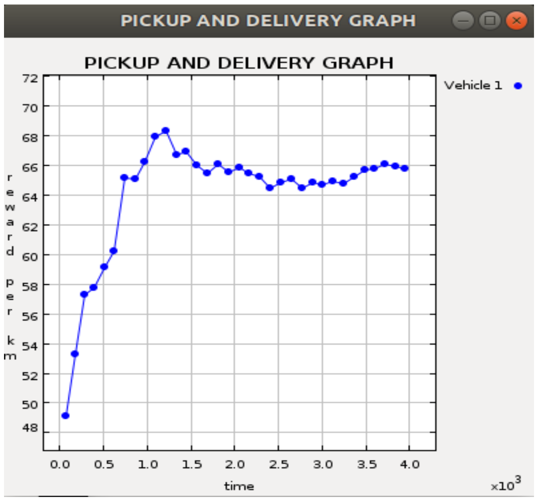
In this experiment, we test the agent with discount values of 0.85 (default), 0.3, and 0.999 (in series) using the France topology. These values are set in the *agents.xml* file. Each experiment runs for 4000 time ticks with a random seed of 1539064511990 to ensure consistent tests.

2.1.2 Observations

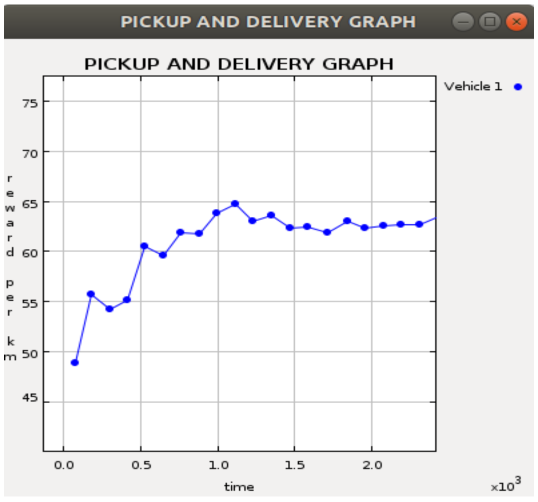
Discount values of 0.85, 0.3, and 0.999 took 118, 20, and 14965 iterations to converge respectively. Note that for below graphs, top left is the starting 2000 time ticks for the simulation and top right is the graph for 4000 time ticks.

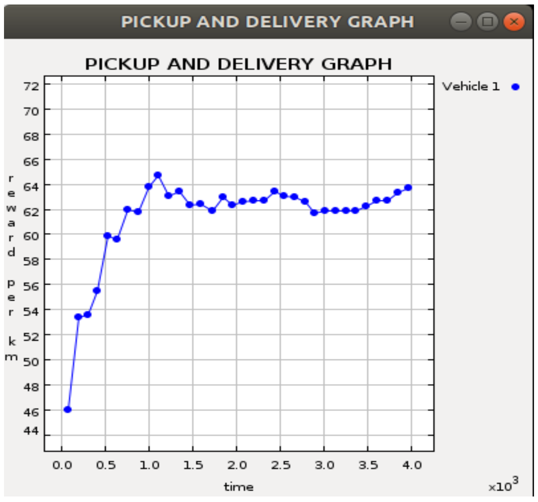
For discount value of 0.85:



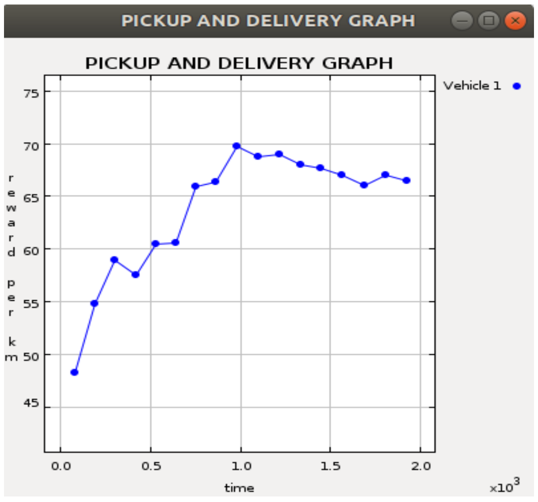


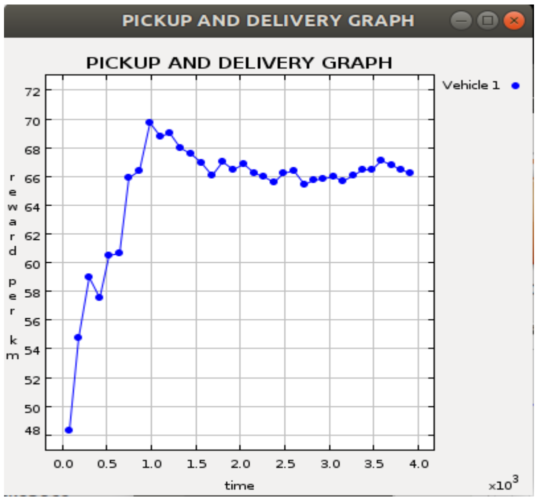
For discount value of 0.3:





For discount value of 0.999:





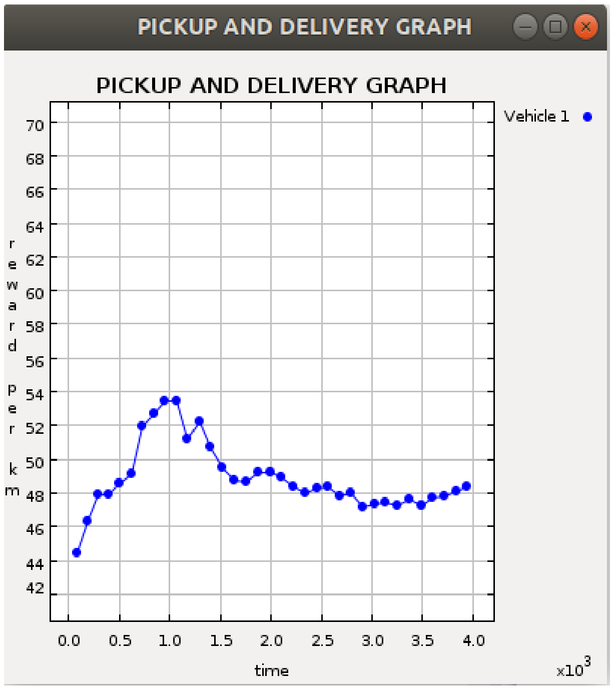
We can see from the above graphs that the highest discount factor of 0.999 performs the best after 4000 time ticks (reward/km > 66), whereas a lower discount factor of 0.3 performs the worst (reward/km < 64). It is also worth noting that 0.999 oscillates around an average profit of 38870 reward per action whereas 0.85 and 0.3 have around 38810 and 38405 reward per action around the 4000 tick mark. This may be because a higher discount value means that the model is more interested in long term reward versus short term reward. This inference is further supported by examining the second tick on the right-side graphs. The 0.3 graph has the highest reward/km of > 55 whereas the 0.85 and 0.999 graphs have values <= 55. This is because for a lower discount rate, the agent tries to maximize short term gain, which explains the higher short term reward for 0.3 (at the second tick) compared to the other rates.

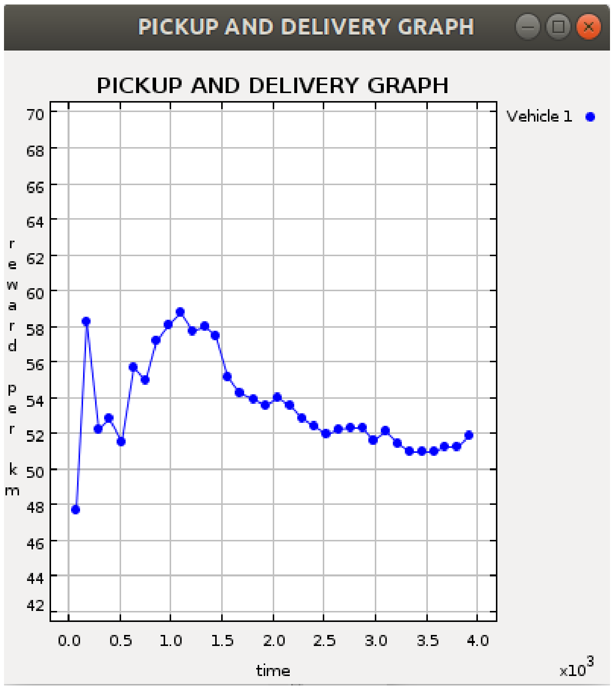
2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

The first dummy agent is simply the random agent given in the starter files. The second dummy agent is configured to always deliver if there is a task present at a city. The dummy agent codes are located in the *RandomAgent.java, DummyAgent.java* files respectively. The *agent.xml* file was configured to run these java files for the experiments, with a discount factor of 0.85. Note that the dummy agent graphs are also stopped after 4000 time ticks, set with seed 1539064511990, and use the France topology to ensure consistency with experiments conducted with varying discount rates in Experiment 1.

2.2.2 Observations





The left graph represents the reward/km performance of the random agent and the right graph represents the reward/km performance of the dummy agent (which always delivers). Compared to the graphs observed in Experiment 1, we see that trained models perform much better than the random and dummy agents, as expected. We see that for the random graph reaches around 48 reward/km and the dummy graph reaches around 52 reward/km, which are significantly worse than the reward/km from the trained models.