A Reactive Agent for the Pickup and Delivery Problem

Smail Ait Bouhsain, Maxime Gautier

October 6th, 2018

1 Problem Representation

1.1 Representation Description

We designed the state representation using all the information available to the agent when it needs to make a decision, ergo each state is composed of three values : The current city, whether a task is available or not and the eventual delivery city. The possible actions the agent can take when in each state is to either pick up the task and go to the delivery city or not pick up the task and go to a neighboring city. Actions are thus defined as two values : whether the task is accepted or not, and the destination city if it is refused. This gives us a number of states that is $numberofcities^2$ and a number of actions that is $numberofcities + 1$.

For the reward table, if the agent picks up the task we know the current city and the delivery city therefore the reward is given by the reward function defined in the TaskDistribution class minus the cost of travel. If the agent does not pick up the task or if there is no task, the reward for each city is the cost of travel to that city.

For the transition table, since when picking up a task we know the delivery city, we know what the next state's current city will be, which we will call city j. In the cases where the task is picked up in the next state, the table will contain the probability that there is a task between city j and each other city, which is given by the probability function in the TaskDistribution class. If the task is not picked up or none are available, the table contains the probability of there being no task available in that city given by the same function when the second argument is null. All other cases will have a probability of zero since it is impossible to end up in those cases.

If the agent does not pick up the task or if none are available, let's consider an action where the agent goes to a city k, which is a neighbor of the current city. The only next states in which we are interested are the ones where the current city is city k. If the agent picks up the task in the next state, the probability of going to the city is given by the probability function in the TaskDistribution class. The probability that the agent doesn't pick up the task or if none are available in the next state is given by the same function where the second argument is null. Once again all the others cases have a probability of zero. There is an action for each city the agent can go to and the same process is used every time.

1.2 Implementation Details

The way we defined our states and actions includes actions that are not permitted, such as moving to a non neighbor city without task, or moving to the same city. It also includes impossible cases like choosing to pick up a task when none are available in the current city. This was necessary to create a reward and probability transition table with a fixed size, and those cases are initialized at zero and stay untouched. We also put up conditions so that we do not use those particular cases in the algorithm that finds V(s) and that way don't falsify our results.

Considering how we have defined our states and actions, when picking up a task, the destination in the actiontype class is not set and when there is not available task, the delivery city in the state is not defined.

The cost of travel is calculated with the function costPerKm in the Vehicle class multiplied by the distance in kilometers.

The state where the current city and the delivery city is the same does not exist which justifies our number of states cited earlier.

We used the algorithm from the slides to find the best V(s) which we then used to find our optimal policy. This policy gives the best action depending on the current state and is called every time the agent needs to make a decision.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

We conducted the experiment with three agents with different discount factors. One had a very high discount factor of 0.95 (blue), one a very low one of 0.25 (green), a middle one with a discount factor of 0.65 (red) and lastly a random agent with a discount of 0.85 (yellow).

2.1.2 Observations

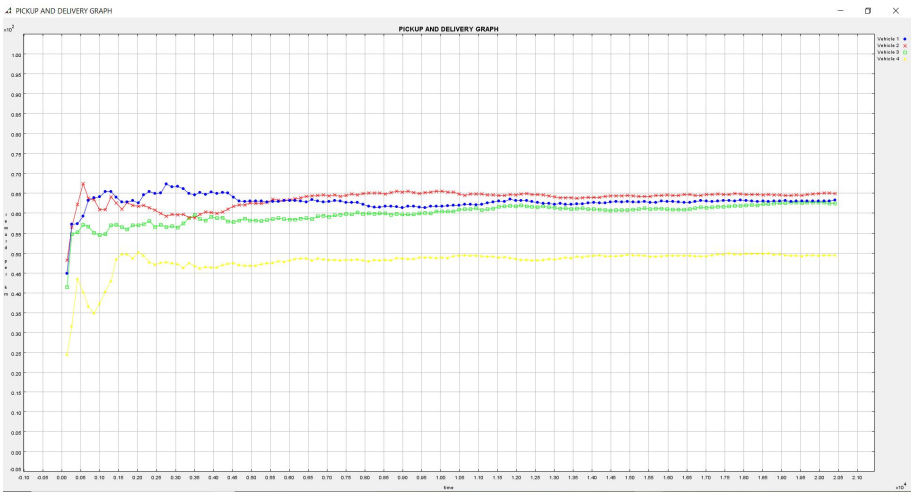

Figure 1: Graph Observations - Experience 1

The discount factor in the Markov Decision Process allows the user to control the importance of future rewards compared to the immediate reward. The higher the discount, the more future rewards are important. We can see that the maximal discount factor doesn't give you the maximum average, which could be explained by the fact that the agent might ignore a lot of tasks in the hope of better future rewards. The agent with the lowest discount factor doesn't perform any better, as it will not consider the potential future rewards as important enough. The best discount factor is therefore in the middle range, so that the agent puts importance in future rewards without overdoing it. In any case, all reactive agents have a much higher average reward than the random agent, despite different discount factors.

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

We compared the performance of three different agents : our reactive agent, a random agent and a dummy we created. The dummy agent will always pick up a task if one is available, and will move to a

random neighbor city if not. Both the random agent and the reactive agent have a discount factor of .85 and the dummy we created does not have a discount factor. The simulation speed used was 10.

2.2.2 Observations

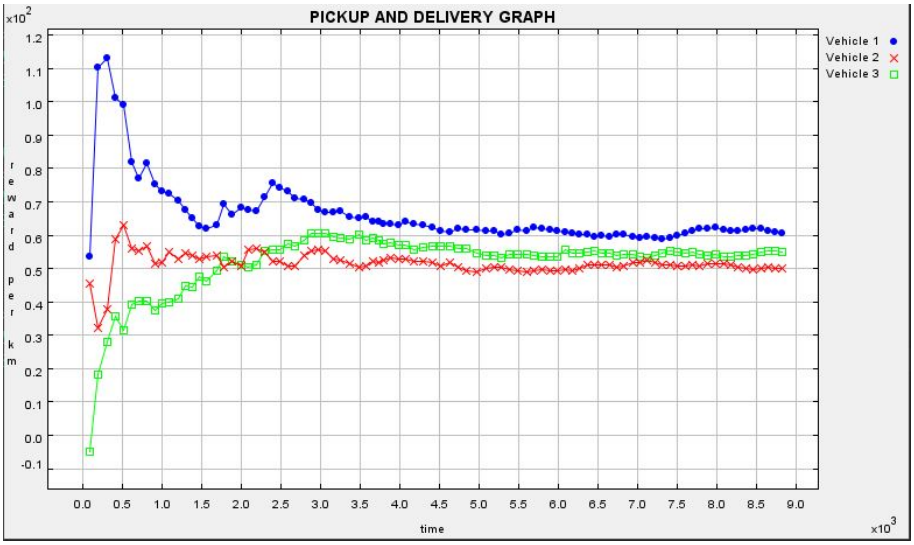

Figure 2: Graph Observations - Experience 2

The blue vehicle is our reactive agent, the green the dummy we created and the red the random agent. We can see that our agent surpasses the other two in terms of reward per kilometer and that the agent that never refuses a task performed better than the random agent. Of course this chart plots the reward per kilometer and not the average reward, but we can see when looking at the console that the average reward follows the same pattern.

```
Reactive Agent : The total profit after 409 actions is 16524175 (average profit: 40401.40586797066)
Random Dummy : The total profit after 382 actions is 11178079 (average profit: 29261.986910994765)
Always Takes Dummy : The total profit after 376 actions is 13383869 (average profit: 35595.39627659575)
```

Figure 3: Console Observations - Experience 2

This is a logical result since an intelligent agent should perform better than dummies as it always acts to maximize the reward. Furthermore, a dummy that always takes a task will have more rewards than one who sometimes refuses them for no reason and it is logical than its average reward is higher.