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

Group №16: Léopold Bouraux, Vincent Rinaldi

October 8, 2019

1 Problem Representation

1.1 Representation Description

In our model, the *State* is represented by 3 attributes : *currentCity* which is the location where the vehicle is when being in a given state, *hasTask* which is a Boolean informing if a task is available in that location or not, and if it's the case, the attribute *destinationCityOfTask* will be the city where the concerned task has to be delivered, such that this destination can't be the same city as the one where the task is picked up. If we consider that there are N different cities, we can then count up to N^2 different States : N States with no task available, and $N * (N - 1)$ States having a task to be delivered. Then we have indeed $N + N * (N - 1) = N^2$ States. Our *Actions* are also represented by 3 attributes : *fromCity* which is the city from where we start our action, *toCity* which is the city from where we end our action, and *isPickupAction* defining if the action is to pick up a task (delivering it is automatically handled by the platform), or if it is a move.

Concerning the build of the *Reward Table* $R(s, a)$ associating a pair *(State, Action)* to its correct reward amount, we decided to use two other different tables : a first table that links each existing State to all the possible Actions that can be chosen being in this given State, and a second table which links each Action to its actual reward. For the calculation of the reward of a *PickUp* Action, we can first get the reward amount $r(i, j)$ of the delivery from the table r that is already given to us, where i is the start city and j is the destination city j. The actual reward of the *PickUp* Action will then be $r(i, j) - cost$ where $cost$ is equal to $DistanceFrom(i)To(j) * CostPerKmOfAgentVehicle$. Concerning the *Move* Actions, their actual reward will then be $(-cost)$ since we only travel between two neighboring cities without any task to deliver. Finally for the *Probability Transition Table* $T(s, a, s')$, we can retrieve the probability $p(i, j)$ (probability that a task is available at city i and has to be delivered at city j) of each State having an available task directly from the given table p (we will call this probability the *probability of occurrence of this State*). The last thing to do is to find the probability of occurrence of a State that has no available task. This is simply $1 - \sum_j p(i, j)$ such that we have i fixed, being the city from where we start moving of this State, and j being all the other cities such that $j \neq i$. We then obtain the *Probability Transition Table* by saying that the probability $Pr(s'|s, a)$ is equal to the probability of occurrence of the State s'.

1.2 Implementation Details

Concerning the representation of our model, we created two additional classes : *States* to represent our States and *VehicleAction* to represent our Actions. Our agent classes are *ReactiveRLA* (main agent) and *ReactiveRandom* (created dummy agent). They are implemented the same way, but with the only difference residing in the *act* method on the condition for the agent to be able to take into account an available task at its location. In *ReactiveRLA*, the agent will always consider if the *PickUp* action is better than the *Move* one, or not, if a task is available in the city he currently is, whereas, in *ReactiveRandom*, each time the agent goes to a location where a task is available, there is a probability $1 - DiscountFactor$ that he decides to ignore it without even considering *PickUp* that could be the best action to perform at this moment.

The *setup* method is divided in two parts : filling our *HashMap* tables (method *fillHashMaps*) and executing the Reinforcement Learning Algorithm (method *offlineRLAlgo*). Filling our *HashMap* tables will link each existing city to a list of their possible States, each State to a list of their possible Actions, each State to its probability of occurrence, each existing Actions to its actual reward value, each State to their best Action to choose (computed by the RL Algorithm), and each State to its corresponding accumulated value (also computed by the RL Algorithm). In the method *fillHashMaps* we operate on each existing city. For each city, we operate on each possible existing States for a given city. For each State, we will create a new instance of them, and then link them to their probability of occurrence, and their possible Actions (after having also created an instance of them) accordingly, using the appropriate *HashMap* tables. We will also link each of these Actions to their corresponding calculated reward, and link the given State to an initial maximum accumulated value that will be computed later in the RL Algorithm. As a last step we link the given city to its possible States, now that we have created them. About the RL Algorithm, the method *offlineRLAlgo* follows the same procedure : while the execution has not yet converged to a best Action given a State (for all possible States), we will compute the corresponding accumulated value given a pair *State, Action* and keep the highest one recorded for a given fixed State, giving us then the best Action for this State. To compute the corresponding accumulated value, we use the formula $Q(s,a) = R(s,a) + \gamma * \sum_{s'} T(s,a,s') * V(s')$ where $R(s,a)$ is the corresponding reward of the current pair (State, Action) in the *Reward Table*, γ is the discount factor, s' is a potential next State, $V(s')$ is the current corresponding accumulated value of this potential next State, and $T(s,a,s')$ is the probability of occurrence of this next State in the *Probability Transition Table*. At the end, we check if the new calculated accumulated value for the given State s and Action a is greater than the current stored one, and if it's the case, we save the new one instead, and assign the new computed best Action to its corresponding State.

2 Results

2.1 Experiment 1: Discount factor

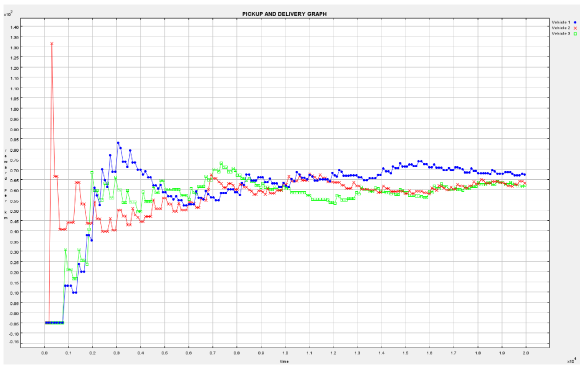

Figure 1

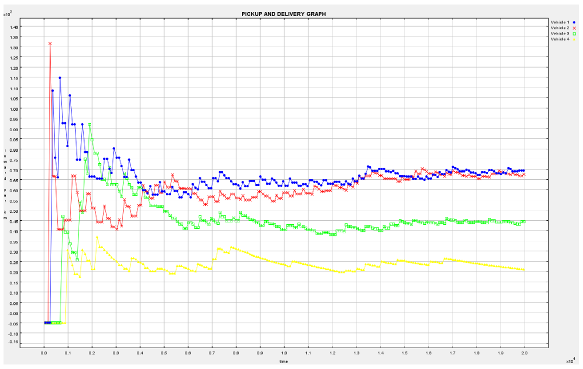

Figure 2

2.1.1 Setting

Figure 1 represents the experiment only with our main agent (*reactive-rla*) with discount factor 0.85 (Blue), 0.50 (Red) and 0.15 (Green). *Figure 2* is an additional experiment with our main agent with discount factor 0.85 (Blue) and our created dummy agent (*reactive-random-enhanced*) with discount factor 0.85 (Red), 0.50 (Green), 0.15 (Yellow). Our main agent with discount factor 0.85 is also our final version. We ran both experiments during 20000 simulation ticks.

2.1.2 Observations

We can deduce from *Figure 1* that the discount factor has no real impact on our main agent on the long term, since that, at the end, the three agents reach almost the same average reward, thanks to an efficient implementation. However we can see, after 700 simulation ticks, that the average reward of the main agent with the highest discount factor tends to linearly increase whereas the average reward of the two other agents still seems to oscillate. With a highest discount factor, the accumulated value, during the execution of the RL Algorithm, tends to deviate way more faster from the initial value *R(s,a)*, making the agent more focus earlier on future states, and stabilizing it more easily. The *Figure 2* shows the discount factor value having an impact on the performance of the agents depending on the way they are implemented, such as our created dummy agent, which sees the convergence of its average reward badly affected if the discount factor is low, since it will ignore more often an available task, and move more recurrently through different cities without any task to deliver.

2.2 Experiment 2: Comparisons with dummy agents

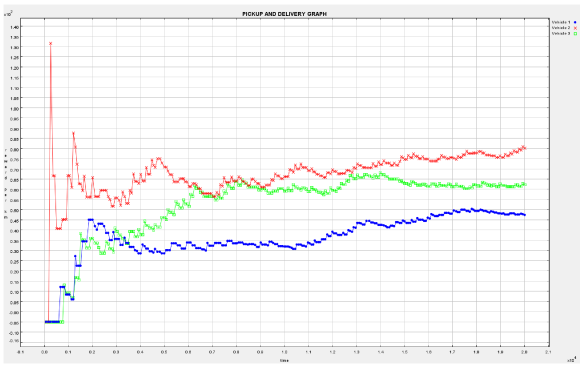

Figure 3

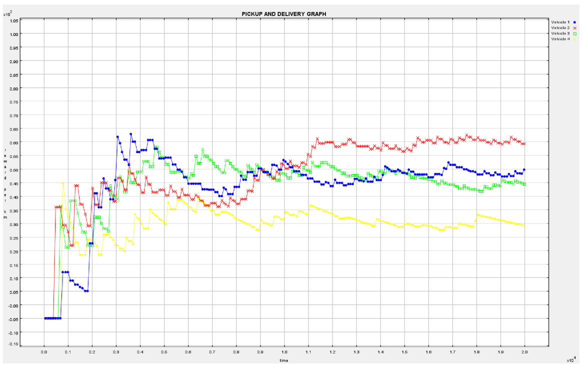

Figure 4

2.2.1 Setting

Figure 3 represents the experiment with the template dummy agent (*reactive-random*) with discount factor 0.85 (Blue), our main agent with discount factor 0.85 (Red), and our created dummy agent with discount factor 0.85 (Green). *Figure 4* is an additional experiment with the template dummy agent with discount factor 0.85 (Blue) and our created dummy agent with discount factor 0.85 (Red), 0.50 (Green), 0.15 (Yellow). While our main agent will always take into consideration the possibility of taking an available task at its location as the best action, our created dummy agent completely ignores each time the available task at a probability $1 - DiscountFactor$. We ran both experiment during 20000 simulation ticks.

2.2.2 Observations

Figure 3 shows that the template dummy agent converges to a lower average reward than the two other agents as expected. By comparing our main agent and our created dummy agent performances, it seems that, by choosing to always consider the possibility to pick up a task when one is available at the same location as our agent, and therefore getting rid of the probability to completely ignore it, we are now able to completely take advantage of the efficiency of the RL Algorithm since that our agent can now always have the opportunity to select the best action depending on the state he is currently in. *Figure 4* is an additional experiment to compare the performance of our created dummy agent depending on the discount factor against the template dummy agent, in order to highlight even more the sensibility of our created dummy agent towards the discount factor, since it performs even worse than the template dummy agent on the long term with a low discount factor.