Intelligent Agents - Assignment 2

Reactive Agent

Theytaz Joël Régis and Madjiheurem Sephora

October 14, 2015

1 Introduction

The goal of this second assignment was to implement an offline reinforcement learning algorithm (RLA) to find an optimal strategy to the Pickup and Delivery Problem. This strategy is used by reactive agents to travel through the network. This report describes the model we chose to represent the problem and give details about how we implemented this model. We also discuss our results and compare the performance of agents running different strategies.

2 Our Implementation

2.1 States representation

We chose to represent the states as a pairs of *source* and *destination* cities. In the main class (*ReactiveTemplate*), we have a field called *mStates*. It is a list of all the possible states. For each state the *source* field has to be defined but the *destination* field may be *null* if there is no task to pick up in the current city. If there is a task to pick up, then the *destination* field will be equal to that task delivery city.
A state also has two other important fields called *Vvalue* and *Tvalue*. The latter is the probability that there is a task available for that state. The former is the rewards that can be reached from that state.
A state finally has a field called *bestAction*. It is the best *action* that may be taken from that state.

2.2 Actions representation

An action is simply defined as either moving to a city or delivering a task to a city. To represent an action, we created a class called *ActionContainer*. It contains two fields. The first one is the type of action, that is either *PICK* or *MOVE*. The second one is the destination city. For actions of type *PICK*, it corresponds to the city to which the agent should deliver the taskt to, and if the action is of type *MOVE*, then it corresponds to a neighboring city to which the agent should go to.

2.3 Reward function

To solve the *Pickup and delivery problem* we have to define a *reward function*. For each state and each action we do the following:
We first check what action type it is. If it is a *PICKUP* action, we have to compute the reward that this task will give. To do so, we use the *reward* method of the *TaskDistribution* class. This method has to be given the source city of the current state and the delivery city of the task.
If the action is of type *MOVE* then the reward is *0*. Finally for both kind of actions we have to deduce, from the reward, the cost of the travel. To do so, we just multiply the cost per km of the vehicle with the distance between the source city and the destination city (computed with the *distanceTo* method of the *City* class).

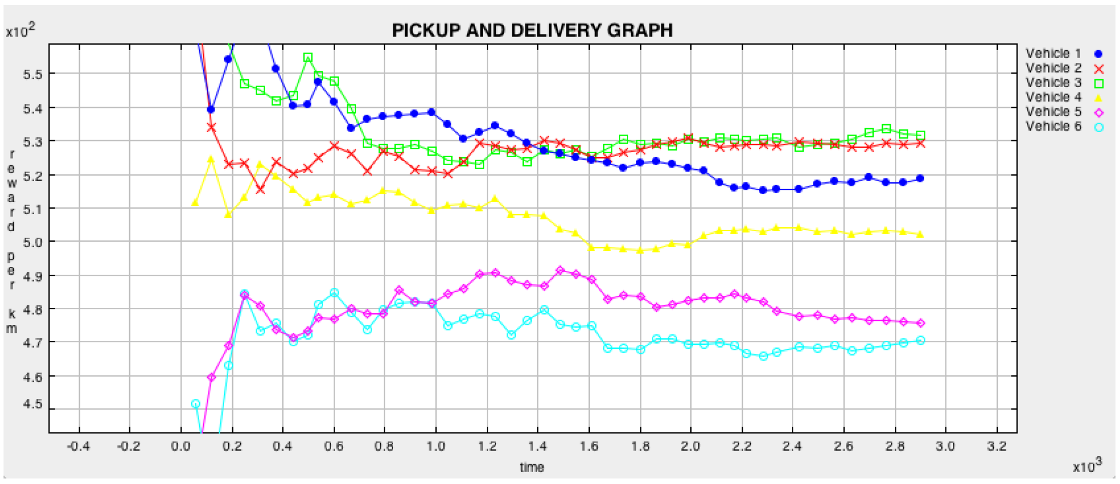

Figure 1: Effect of discount factor γ

2.4 Reinforcement learning

The agents in each state have to choose which action is the best to take. This is computed in our *reinforcementLearning* method. This method is called before the simulation begins. It does the following:
For each state, we compute the actions that may be taken (computed with our *possibleActionsFromState* method). Then for each of those actions, we compute the reward that it gives (as explained in the previous section). Furthermore, we add to the current computed reward value, by value iteration, a "predicted" reward value. This value is multiplied by a discount factor $\gamma \in (0, 1)$ because the future is less valuable than present.

$$\gamma \sum_{s' \in S} T(s, a, s') \cdot V(s')$$

We do this process until the difference between two successive iterations is less or equal than a small value ϵ. We set $\epsilon = 0.01$ in our implementation.

3 Results and Discussion

3.1 Choice of the discount factor γ

For the algorithm to converge to the optimal solution, we have to choose a good value for the discount factor γ. Its value reflects the importance we give to the future. In order to find which value of γ gives the highest reward per kilometers traveled (which is our measure of performance), we ran the simulation using different value for γ. Figure 1 shows the graphs of the reward per kilometers as a function of time of vehicles 1 to 6 having a discount factor of 1, 0.95, 0.8, 0.5, 0.1 and 0 respectively. Observe that the reward per kilometers curve stabilizes in all case after a certain time. From the plots, we deduce that the best value for γ is close to 1, but not exactly 1. For the remainder of the study we used $\gamma = 0.9$

3.2 Performance analysis

To study how well our reactive agent does compared to agents using a different strategy. The first strategy we will use for comparison is the one that was given to us in the initial ReactiveTemplate.java file. Such agents simply choose to pick up a task present in their current city with probability *pPickup = 0.5*, call this the "random strategy". The second strategy is greedy; agents always choose to pickup a task if they have the possibility to (that is, pPickup = 1), call it the "greedy strategy".

We thus ran our simulation with 6 agents: vehicles 1 and 2 are using the optimal strategy computed by the RLA, vehicles 3 and 4 are traveling according to the random strategy and

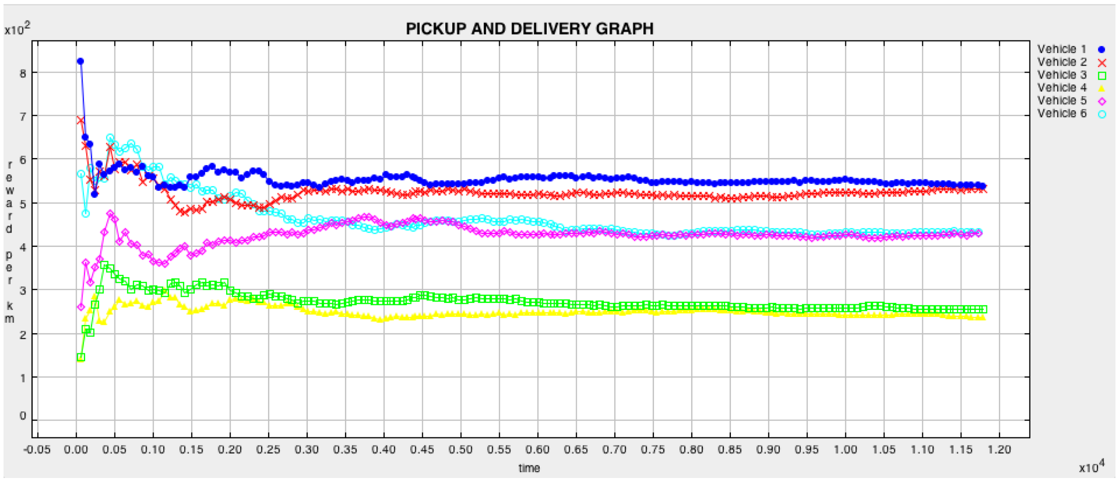

Figure 2: Agents' performance

vehicles 5 and 6 are greedy. The environment in which we are working is the one suggested by the given files in ./config using the dutch cities. More specifically, the probability that there is a task is given by the uniform distribution between 0 and 1, the probability that there is no task is also uniformly distributed but between 0.2 and 0.4, and the value of a reward is a constant between 1000 and 99999, with a higher value for short distance tasks.
Under these settings, we can expect vehicles 1 and 2 to perform the best as they are using a supposedly optimal strategy.

Figure 2 depicts the reward per kilometer as a function of time for each of the 6 agents. Not surprisingly, we can observe that this value converges after some time and vehicles using the same strategy converge to a same value of the reward per km. As we predicted, vehicles 1 and 2 perform better than the other vehicles, so their strategy is probably indeed optimal. Note that the greedy strategy performs the worse, meaning that accepting every task is not a good idea if we want to maximize the reward.

4 Conclusion

In this assignment we have learned to use a reactive agent to solve the Pickup and Delivery problem. We did so by designing a states and actions model that characterized well the problem. We then implemented a reinforcement learning algorithm that computed a strategy offline. Finally, we evaluated the performance of our reactive agent and concluded that the learned strategy was indeed optimal.