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

Group №: Student 1, Student 2

October 11, 2016

1 Problem Representation

1.1 Representation Description

The state representation: The question for the agent is: "How is the world right now?". For him the world is the city where he is. The agent have one state per city. That's where the agent can pickup a task or move. During his movement the agent can't pick an action. We suggest that the state is the currentCity of the agent.

The possible actions: The agent in a state A can only do two different type of action. The first one is to take the task, we will call this action the PICKUP action. This PICKUP action has for destination state the city of the delivery. The second action is when the agent decide or must move to a neighbor city. We will call this action the MOVE action. This MOVE action has for destination state the chosen neighbor city.

The reward: Every time an agent move from one city to another, this have a cost. The cost is the cost/km * km. Also, if an agent take an PICKUP action, this action will permit him to earn money, if it is a MOVE action, the agent earn nothing. The reward is the substraction of the money earned from the action by the cost of the movement. Everytime an agent will do a move the reward will be negative.

The probability transition table: The uncertainty in the model only comes from the presence of tasks or not in the destination cities. Each city has a probability to have a task for another city. This will be the probability to find a PICKUP action for a destination city in the current city. But we also have the probability to do not find any task in the current city. For this we affirm that the probability to do a MOVE is the probability to do not find task in the city.

1.2 Implementation Details

class State = (currentCity, destinationCity)

class AgentAction = (destinationCity, an action(MOVE or PICKUP))

class TupleActionValue = (action, value): storing the best action and the best value of a state

Data structure:

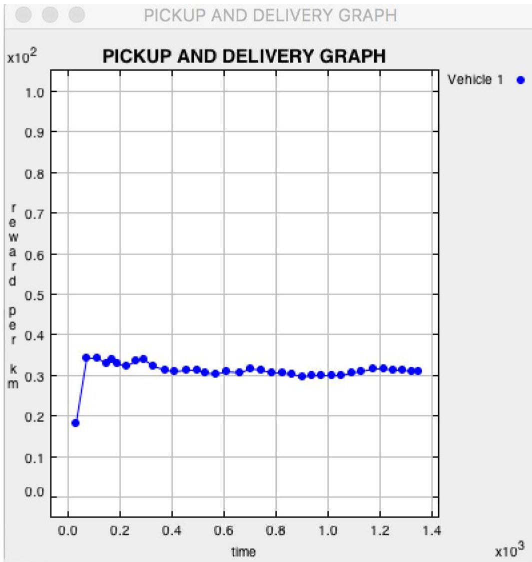
```
private HashMap<State, Double> probaEachState // each state(a->b) has a probability
private HashMap<City, Double> probaMoveAtCity //probability to take a MOVE at each city
private HashMap<AgentAction, Double> actionReward // each action has a reward
private HashMap<State, List<AgentAction>> allActionsOfState // all possible actions of a state
private HashMap<City, List<State>> allStateOfCity // all possible states of a city
private HashMap<State, TupleActionValue> bestChoiceOfState // store the best Action and best
    Value of a state
private ArrayList<City> allCities // all cities
```

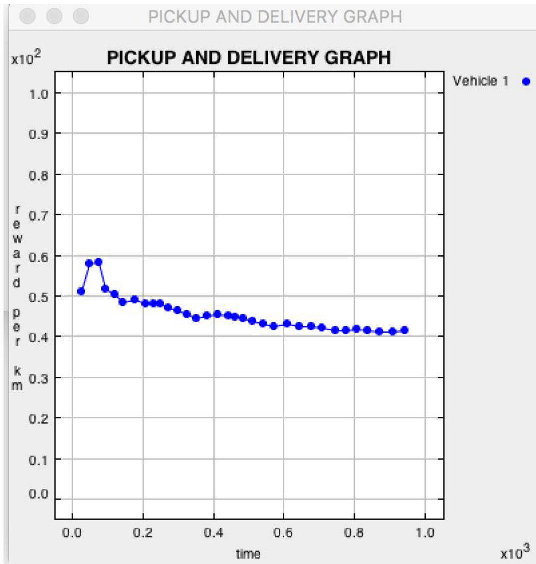
Reinforcement Learning:

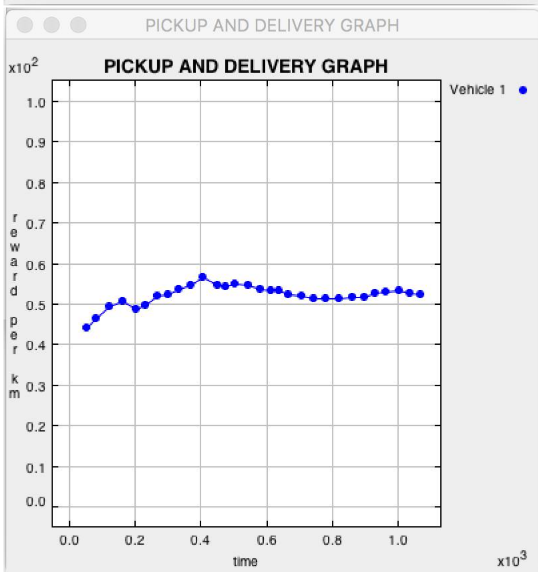
```
  public void reinforcementLearning(){
     boolean goodEnough; // value to determine if that algorithm is converge
     do {
        goodEnough = true;
        for (City city : allCities){
           List<State> allStates = allStateOfCity.get(city); // get all states for learning
           for (State oldState : allStates){
              // There are best Action and best Value of State oldState in this tuple
              TupleActionValue tuple = bestChoiceOfState.get(oldState);
              List<AgentAction> allActions = allActionsOfState.get(oldState); //get all actions
              for (AgentAction action : allActions){ //
                 double qValue = actionReward.get(action); // qValue = Q(s,a)
                 // get all nextState s.t oldState.destinationCity == nextState.currentCity
                 for (State nextState: allStateOfCity.get(action.getDestination())){
                    qValue += (getProba(nextState, action) *
                        bestChoiceOfState.get(nextState).getBestValue());
                 }
                 qValue *= this.pPickup; // time discount-factor
                 if (tuple.getBestValue() < qValue){ // V(S) <- max Q(s,a)
                    tuple.setBestValue(qValue);
                    tuple.setBestAction(action);
                    // There is a change of vector V(S), have to re-calculate all value of V(S)
                    goodEnough = false;
                 }
              }
           }
        }
     } while (! goodEnough);
  }
```

2 Results

2.1 Experiment 1: Discount factor







2.1.1 Setting

From up to bottom, left to right : discount factor = 0.5 ; 0.75; 0.99 respectively

2.1.2 Observations

The result matches our prediction. Generally, more the discount factor is big (up to one), more the reward per km agent will win, and in our model, these three different factor gives us a different result. For discount = 0.5, we have average = 17500. For discount = 0.75, we have average = 26000. For discount = 0.99, we have average = 34000. We can see that different discount leads us to different situation and for discount = 0.99, we have the best gain.