COL333: Assignment 3

Sachin 2019CS10722
Saurabh Verma 2019CS50129

November 2021

Before implementing any AI learning we created a class called Map, that stores the map and checks for walls between two cells. We also created a class called State that stores the state parameters and has some useful functions required by different components of assignment. Map class stores the map in the form of dictionary d (having walls) and 2 variables height and width. More of them will be explained inside the questions.

1 Part A: Computing Policies

1.1 Formulation of the Taxi Domain as an MDP:

We have created an MDP for the Taxi Domain formulation as described in the problem statement. We take the wall locations, position of depots, and width and height of the grid as input. Then we create an instance of the map class with these values and use in the MDP.

a) **State Space:**
We have created a class for the state. Each state in Taxi Driver MDP has four identifying features: the position of the taxi, position of the passenger,a boolean variable picked denoting whether passenger is picked by the taxi, and the destination. These are the variable describing the state.
Now the x position can take discrete values up to the width, y position can take up to the height. So the total number of values position of passenger and position of taxi can take is height*width. But when the boolean value of picked is true then position of taxi and passenger is same. And destination variable takes values equal to the number of depots.
So that makes the total number of variables to be :-
$w * h * w * h * d + w * h * d$
where,
h = height of map
w = width of map
d = number of depots
So state is $((x_1, y_1), (x_2, y_2), p, d)$ where $(x_1, y_1) = (x_2, y_2)$ if p is True.

b) **Transition Model:**
Now the action in our formulation are (N, S, E, W, PICK, DROP). So the transition model is as follows.

- T(s, DROP, s')=1
- T(s, PICK, s')=1
- This same applies to all the other direction(the x and y values get modified accordingly)
  - T$((t_1, p_1, p, d), N, (t_2, p_2, p, d)) = 0$
    (Given that there exists a wall between two t_1 and t_2 otherwise:-)
  - T$(((x, y), p_1, p, d), N, ((x, y + 1), p_2, p, d)) = 0.85$
  - T$(((x, y), p_1, p, d), N, ((x, y - 1), p_2, p, d)) = 0.05$
  - T$(((x, y), p_1, p, d), N, ((x - 1, y), p_2, p, d)) = 0.05$
  - T$(((x, y), p_1, p, d), N, ((x + 1, y), p_2, p, d)) = 0.05$

  if p is True then $p_2 = t_2$ (taxi position in new state) else $p_2 = p_1$ in all above transitions.

c) **Reward Model:**
The reward structure is as follows:

- R$((d.pos, d.pos, True, d), DROP, (d.pos, d.pos, False, d)) = 20$
- R$(s, DROP, (t_2, p_2, p, d)) =$ -10
  (Given that $t_2 != p_2$)
- R$(s, PICK, (t_2, p_2, p, d)) =$ -10
  (Given that $t_2 != p_2$)
- All other transitions have a reward of -1

1.2 Implementing Value Iteration for the Taxi Domain:

a) We implemented the value iteration in our code inside valueIteration() function in the MDP class. The value iteration took epsilon as the input. Also, the default value of γ is set to be 0.9 but can be changed as specified. For the epsilon value of 0.1, the number of iteration the code took to converge is 15.

b) The plot of Max-norm distance vs Iteration index is given as follow: (The discount factors are given in the range [0.01,0.1,0.5,0.8,0.99] in that order)

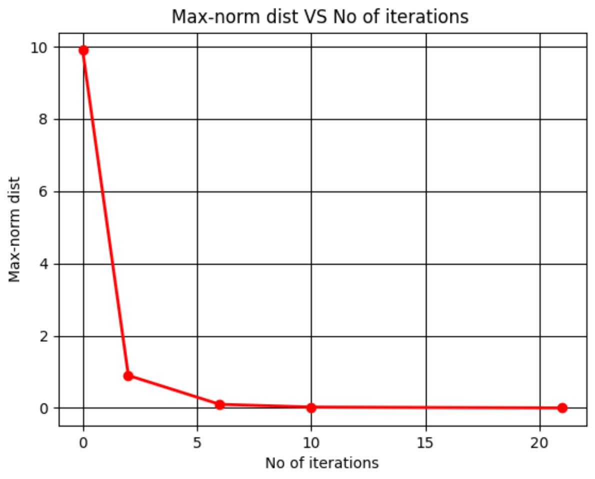


Figure 1: Max Norm distance vs iteration Index

The max-norm distance is the amount of error we can incoperated in the values, that in a sense signifies the maximum difference from the optimal value. So as the number of iteration of convergence is increasing we are coming more and more close to the optimal value. Our estimates are improving further and further as we increase the number of iterations, hence the max-norm distance decreases.
One another way to look at it is that as the max-norm distance decreases, the estimate value must be close to the optimal value itself, hence the number of iteration of convergence is more.

c) In this part we were asked to simulate the policy obtained using the value iteration once with γ 0.1 and then with 0.99 with bound of 20 steps. Following are the results:-

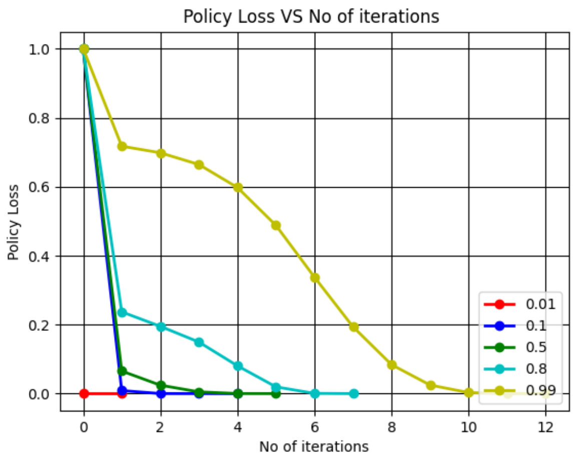
- $\boldsymbol{\gamma = 0.1}$**:** In this case the policy obtained are not optimal and taxi performs very poorely, very rarely it picks and drops the passenger in 20 states bound.
- $\boldsymbol{\gamma = 0.99}$**:** In this case policy obtained is optimal and taxi successfully picks and drops the passenger in required destination within 20 states.

This is because γ denotes the discount factor of algorithm, that is quantifier of importance given to Values of states that are visited later (far off states) so if γ is kept low, then there is less transfer of imformation from one state to another (because there is more attenuation) and hence poor learning.

1.3 Implementing Policy Iteration for the Taxi Domain:

a) To implement the policy evaluation step, we have two options; one is the exact method in which we create linear equations in variables equal to the number of the total state and get the exact utility value of the various states. The other is the approximate method which runs an update on the states until the value converges. The first method mentioned is called Linear Algebra Method, the second one is the Iterative method. We have implemented them both.
The linear algebra method takes O(n^3) time for policy evaluation, where n is the number of states. If the transition model given to us is sparse, meaning that each state transition leads to only a small number of states, then the Linear algebra method is preferred because then the solution is obtained faster. The linear Algebra method is preferred when our state space is small. For larger state space, O(n^3) becomes too much overhead, so iterative method is preferred

b) The graph comes out as following for different values of the discount factor.



2 Part B: Incorporating Learning

2.1 Implementing the approaches to learn the optimal policy:

In this part we had to implement the Q learning and SARSA reinforcement learning techniques with different exploration methods.
For doing that we made a general function that on changing parameter becomes required algorithm. This function takes epsilon and alpha as parameters and learns the optimal Q table. Initially we started with Q table with all values 0 and performed n number of iterations of episodes. In each episode, starting from a random state we keep on moving to next state by performing action depending upon exploration technique and updating Q value depending on algorithm (Q learning or SARSA) by using the reward got upon performing that action untill we reached terminal state.

Learning techniques:-

- **Q learning:** This is an online algorithm, where Q table is updated using the following formula:-
- **SARSA:** This is an offline algorithm, where Q table is updated using the following formula:-

Exploration techniques:-

- **e-greedy:** Any random action with probability e otherwise best action according to Q table.
- **exploration decay:** Any random action with probability e/$\sqrt{i}$ (where i = iteration number) otherwise best action according to Q table.

2.2 Number of iterations VS Discounted reward:

In this part we needed to plot Number of iterations VS Max-norm dist graph for all the algo in part 1. For that we first find the Policy by varying number of iterations and then found sum of discounted rewards for 100 random states (for better results) and then average out the reward. And then plotted the graph between number of iteration and discounted reward sum. Same is repeated for all algorithms.
Following are the results:-

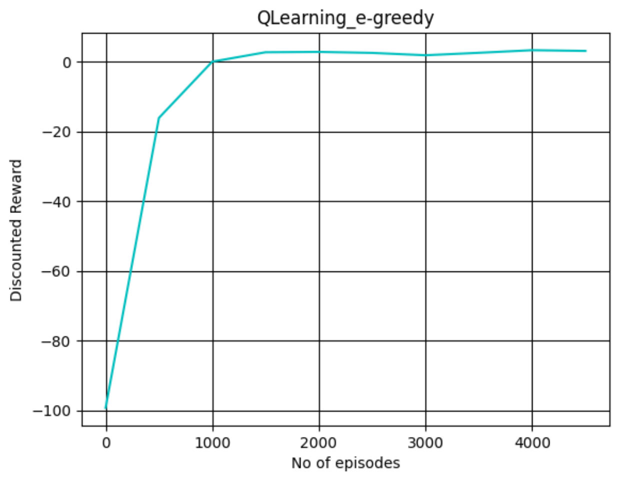


Figure 2: QLearning e greedy

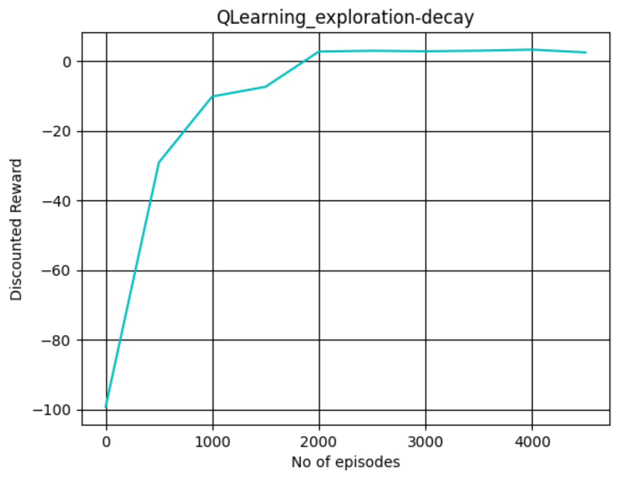


Figure 3: QLearning exploration decay

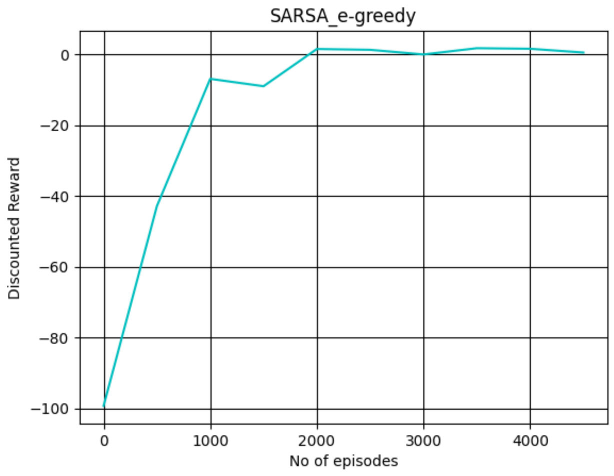


Figure 4: SARSA e greedy

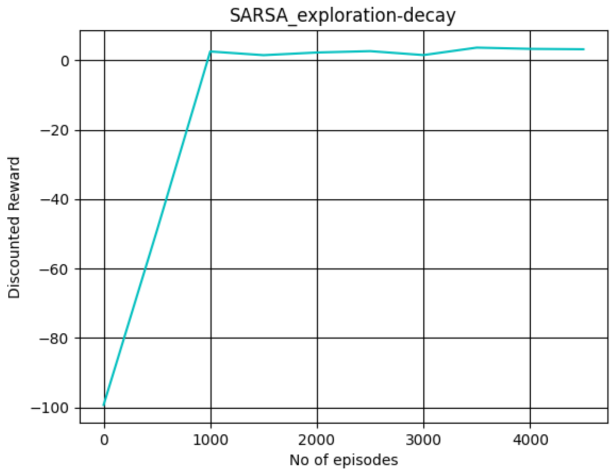


Figure 5: SARSA exploration decay

Analysis:

2.3 Execution using learned policy:

In this we started from any random start state used the policy values learned using Q learning with exploration decay to perform action at any state. So basically we simulated the episode using learned actions. This is done on 5 random start states. Following are the results:-

- The taxi is able to successfully pick the passenger and then drop it at the correct destination in less than the bound set(500) on steps.
- There are stochastic behaviour in next state. Sometimes when taxi takes action North, still it moves in direction other than North.
- But taxi is able to overcome that probabilistic behaviour and able to perform optimally from any given state.

2.4 Analysing effect on policy by changing epsilon and alpha:

In this we needed to plot graph of part 3 by changing alpha and epsilon for Q learning with e-greedy. Following are the results:-

(a) **alpha = 0.1 and varying epsilon:**

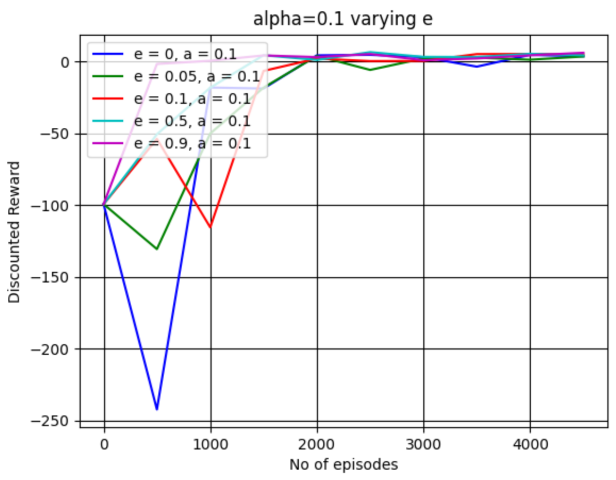


Figure 6: alpha = 0.1 and varying epsilon

It can be seen from the graph that as epsilon is increased, rewards converges in lesser number of iterations. This is quite imperative as epsilon is the probability with which we are exploring states and as more and more states are observed, changes in Q values from far off state travels faster. But there is a tradeoff in time taken per episode. Since we are giving more priority in exploring than exploiting current state, terminal state is reached in larger time.

(b) **epsilon = 0.1 and varying alpha:**

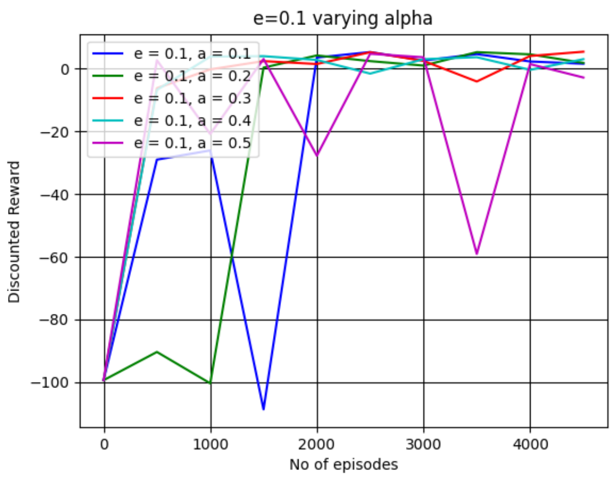


Figure 7: epsilon = 0.1 and varying alpha

It can be seen from the graph that as α is increased the rate of convergence decreases (reward is low and there are large fluctuations). This is because as α is increased, the importance given to next state achieved is increased and importance to the past Q value that is recorded so for is decreased. Hence only nearby neighbours of any state affect it and positive rewards in far off state is not travelled (well, it is travelled but gets attenuated as constant 1-α is multiplied to it that decreases as α is increased). This also explains the reason of large fluctuations, random state that is closer to terminal state will result in positive reward and correct policy but far off state will result is high negative reward.

2.5 Testing on larger grid:

For this part we were required to test out best algorithm on 10*10 grid provided. For that we first created the grid using the method explained in first part, then we calculated the optimal Policies for all 8 destinations using Q learning with exploration decay (10,000 iterations) and stored them. Then we found discounted reward sum on 500 instances of problem and took average of that. The answer was -8.