Indian Institute of Technology, Delhi

Fall 2021

COL333

Introduction to Artificial Intelligence

Assignment 3

Markov Decision Process

Submitted by:

Jitender Kumar Yadav
(2019CS10361)

Rakshita Choudhary
(2019CS10389)

A. Computing Policies

1. Problem Formulation as MDP

The taxi domain as a Markov Decision Process (MDP) is formulated as follows:
For simplicity, instead of representing grid coordinates as (x, y), we represent each position in the grid as a single integer. For a n x n grid, $(x, y) = n * y + x$. The following figure shows the given 5 x 5 grid as per our notation:

00.01|02.03.04
05.06|07.08.09
10.11.12.13.14
15|16.17|18.19
20|21.22|23.24

A state is represented as follows: $(taxi - location, status, passenger - location)$. Here $taxi - location$ is an integer representing the location of taxi in the grid, $status$ is a boolean value indicating whether or not the passenger is inside the taxi, and $passenger - location$ is an integer representing the location of passenger in the grid. $status$ can be true only when $taxi - location$ and $passenger - location$ are the same. Now, $taxi - location$ and $passenger - location$ can take n^2 different values each, making a total of n^4 combinations. $status$ can take two values (true and false). There are n^4 combinations possible for $status$ = false and n^2 combinations possible for when it is true. Therefore, there are a total of $n^4 + n^2$ states possible i.e. for the given 5 x 5 grid, the total number of states is 650. One out of these 650 states is the termination state, it is $(destination - location, False, destination - location)$. This state indicates that the passenger has been delivered to the destination.
The action space consists of the following actions: $East$, $North$, $South$, $West$, $Pickup$, $Putdown$
The transition model contains the transition probabilities of moving from one state to another on taking a certain action. The entries look like $Transition(s_1, a, s_2) = p$, where s_1 is the initial state, a is the action taken in the initial state, s_2 is the state reached after taking this action and p is the corresponding probability. For s_1, there are two possibilities:

1. Passenger is in the taxi ($status$ = True): In this case, any navigation action will lead to a state in which the new $taxi - location$ is an adjacent grid cell to the previous $taxi - location$, and $status$ = True. The transition probability to all other states is 0. $Putdown$ action in such a state leads to a state with the same taxi and passenger locations but $False$ status, with a probability of 1. $Pickup$ action does not make any change in the state, i.e. $s_1 = s_2$ with probability 1.

2. Passenger is not in the taxi ($status$ = False): In this case, any navigation action will lead to a state in which the new $taxi - location$ is an adjacent grid cell to the previous $taxi - location$, new $passenger - location$ is the same as before, and $status$ = False. The transition probabilities to any other states are 0. $Putdown$ action does not make any change in the state, i.e. $s_1 = s_2$ with probability 1. $Pickup$ action succeeds with probability 1, if $passenger - location$ and $taxi - location$ in s_1 are the same. Otherwise, it does not lead to any change in the state.

Note that,

$$\sum_{s \in States} Transition(s_1, a, s) = 1$$

For a 5 x 5 grid, there are a total of $650 * 6 * 650$ entries in the transition model.
The reward model contains the rewards of moving from one state to another on taking a certain action. The entries are similar to that of the transition model, i.e. $(s_1, a, s_2) = r$. The agent gets a reward of +20 for successfully delivering the passenger to destination grid cell i.e. $Reward(s_1, Putdown, destination) = 20$. Further, a reward of -10 is given if the taxi attempts to $Pickup$ or $Putdown$ a passenger when the taxi and the passenger are not located in the same grid cell i.e. $Reward(s_1, Putdown/Pickup, s) = -10$ when $s_1(taxi - location) \neq s_1(passenger - location)$. Rest all other actions lead to a reward of -1.

2. Value Iteration

Considering Gamma (γ) = 0.9, we obtain the optimal policy for Epsilon (ϵ) = 0.001. The number of iterations taken for convergence is 35.

Gamma (γ)	Number of Iterations	Epsilon (ϵ)
0.99	41	0.001
0.8	28	0.001
0.5	14	0.0001
0.1	6	0.0001
0.01	4	0.0001

Table 1: Effect of Discount Rate on Value Iteration

We see that the discount rate has significant impact on the number of iterations and smoothness of convergence for Value Iteration. At a very low discount rate, the number of iterations needed for convergence are very low. The policy converges faster since the impact of rewards at a later stage diminishes. However, this policy will not be a good policy since the taxi agent will not chase a higher reward at the end, instead will keep on staying where he was as the impact of high rewards at a later stage discounts to zero. This will be seen in the next method.

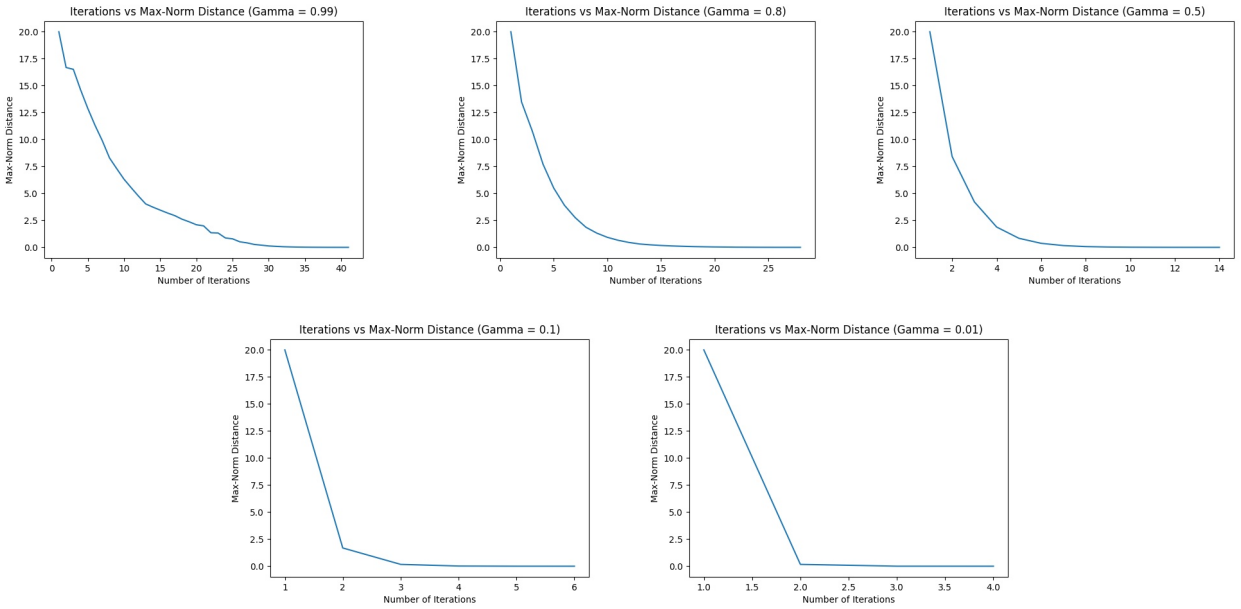

Figure 1: Convergence of Value Iteration versus Discount Rate

As the discount rate rises, the number of iterations needed for convergence also rises proportionally. This is because at a higher discount rate, rewards at a later stage will have significant impact on the policy. The rewards are discounted slowly, hence the agent will chase high rewards even at a later stage and iteration will go on till a sufficiently optimal policy has been reached.

Policy Simulation

We simulated the Value Iteration Algorithm for Markov Decision Process for a given destination and three different sets of initial passenger and taxi location. The state and actions taken were considered for at least first 20 states.

Case#1

Passenger initial location = Y $\equiv$ 20 $\equiv$ (0,0)
Passenger destination location = G $\equiv$ 4 $\equiv$ (4,4)
Taxi initial location = R $\equiv$ 0 $\equiv$ (0,4)

Explanation: The agent performs optimally in case of $\gamma = 0.9$. The taxi starts at location R and goes to passenger at location Y using the shortest path. It then picks up the passenger and goes to the destination and successfully completes the episode. We can also observe that even on choosing action $North$ in state $(8, True, 8)$

Step	Gamma = 0.9		Gamma = 0.1	
	State	Action	State	Action
0	(0, False, 20)	South	(0, False, 20)	East
1	(5, False, 20)	South	(1, False, 20)	South
2	(10, False, 20)	South	(6, False, 20)	East
3	(15, False, 20)	South	(6, False, 20)	East
4	(15, False, 20)	South	(6, False, 20)	East
5	(20, False, 20)	Pickup	(6, False, 20)	East
6	(20, True, 20)	North	(6, False, 20)	East
7	(15, True, 15)	North	(6, False, 20)	East
8	(10, True, 10)	East	(6, False, 20)	East
9	(11, True, 11)	East	(6, False, 20)	East
10	(12, True, 12)	North	(5, False, 20)	East
11	(7, True, 7)	East	(6, False, 20)	East
12	(8, True, 8)	North	(6, False, 20)	East
13	(13, True, 13)	North	(6, False, 20)	East
14	(12, True, 12)	North	(6, False, 20)	East
15	(7, True, 7)	East	(6, False, 20)	East
16	(8, True, 8)	North	(6, False, 20)	East
17	(3, True, 3)	East	(6, False, 20)	East
18	(4, True, 4)	Putdown	(6, False, 20)	East
19	(4, False, 4)	-	(6, False, 20)	East

Table 2: States and Actions taken for start state (0, False, 20)

in step 12, the agent ends up in $(13, True, 13)$ instead of $(3, True, 3)$. This occurs because of the stochastic nature of transitions. As we know the navigation transitions succeed with a probability 0.85 only, here we see one such case where the transition did not go as expected.
The agent does not perform well in case of $\gamma = 0.1$. It fails to even pickup the passenger and never manages to complete the episode. The policy learned here is not optimal. Due to very low discount factor, the agent becomes extremely myopic and learns about actions that produce an immediate reward. The rewards at a later stage cause negligible changes in the utility since the term $\gamma^k \to 0$ as the number of steps, k becomes even slightly larger say $k = 4$. Thus, convergence is delayed at larger discount rate but the policy becomes optimal.

Case#2
Passenger initial location = 22 ≡ (2,4)
Passenger destination location = G ≡ 4 ≡ (4,4)
Taxi initial location = 4 ≡ (4,0)

Step	Gamma = 0.9		Gamma = 0.1	
	State	Action	State	Action
0	(4, False, 22)	West	(4, False, 22)	South
1	(3, False, 22)	South	(4, False, 22)	South
2	(8, False, 22)	West	(9, False, 22)	East
3	(7, False, 22)	South	(9, False, 22)	East
4	(12, False, 22)	South	(8, False, 22)	East
5	(17, False, 22)	South	(9, False, 22)	East
6	(22, False, 22)	Pickup	(9, False, 22)	East
7	(22, True, 22)	North	(9, False, 22)	East
8	(17, True, 17)	North	(9, False, 22)	East
9	(12, True, 12)	North	(4, False, 22)	South
10	(7, True, 7)	East	(9, False, 22)	East
11	(8, True, 8)	North	(9, False, 22)	East
12	(9, True, 9)	North	(9, False, 22)	East
13	(4, True, 4)	Putdown	(9, False, 22)	East
14	(4, False, 4)	-	(9, False, 22)	East

Table 3: States and Actions taken for start state (4, False, 22)

Case#3
Passenger initial location = 6 ≡ (1,1)
Passenger destination location = G ≡ 4 ≡ (4,4)
Taxi initial location = 15 ≡ (0,3)

Step	Gamma = 0.9		Gamma = 0.1	
	State	Action	State	Action
0	(15, False, 6)	North	(15, False, 6)	North
1	(10, False, 6)	North	(10, False, 6)	East
2	(5, False, 6)	East	(11, False, 6)	East
3	(6, False, 6)	Pickup	(12, False, 6)	East
4	(6, True, 6)	South	(13, False, 6)	East
5	(11, True, 11)	East	(14, False, 6)	East
6	(12, True, 12)	North	(9, False, 6)	East
7	(7, True, 7)	East	(9, False, 6)	East
8	(8, True, 8)	North	(9, False, 6)	East
9	(3, True, 3)	East	(9, False, 6)	East
10	(4, True, 4)	Putdown	(9, False, 6)	East
11	(4, False, 4)	-	(9, False, 6)	East

Table 4: States and Actions taken for start state (15, False, 6)

3. Policy Iteration

We implemented policy evaluation using the following two methods:

1. **Algebraic Method:** Consider the following *Bellman* equation (given a policy π) for all n states.

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

   We simplify the above equation for all states to convert them into n linear equations $AV = B$ in n variables (corresponding to the utilities of n states). The matrix equation was solved using the linalg module of the numpy library.

2. **Iterative Method:** This method performs value iteration to solve the *Bellman* equations (given a policy π) for all states until the value variables V converge.

$$V^{\pi}_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')]$$

We executed both policy evaluation algebraically as well as iteratively for many test cases.

```
Case#1
Algebraic Policy Evaluation
Total Policy Iterations = 5

Case#2
Iterative Policy Evaluation
Total Policy Iterations = 6
Total Policy Evaluations = 306
Policy Evaluations per Iteration = 51.0
Policy Evaluations vs Iteration [67, 67, 67, 35, 35, 35]
```

The time required for solving n linear equations in n variables takes place in best $O(n^3)$ time by using eigenvalues of the matrix in consideration. Assuming numpy performs the fastest kind of equation solution, total time for algebraic method is approximately $O(|S|^3)$.
One iteration of Iterative Policy Evaluation involves $O(|S|^2|A|)$ steps. Thus, total time assuming k iterations before convergence is $O(k|A||S|^2)$. Thus, we see that iterative policy evaluation is better than algebraic policy evaluation when the number of iterations before convergence is not too high (that is in most cases). However when the number of iterations is as high as the state space (which is way larger than the actual number of iterations), the algebraic method works better.

Effect of Discount Factor on Policy Iteration

We varied the discount factor, γ and performed policy iterations over the same state space and destination. The policy loss (which is the difference between the optimal policy and the current policy) plotted against the number of iterations to see the effect of discount factor on the convergence of policy iteration.

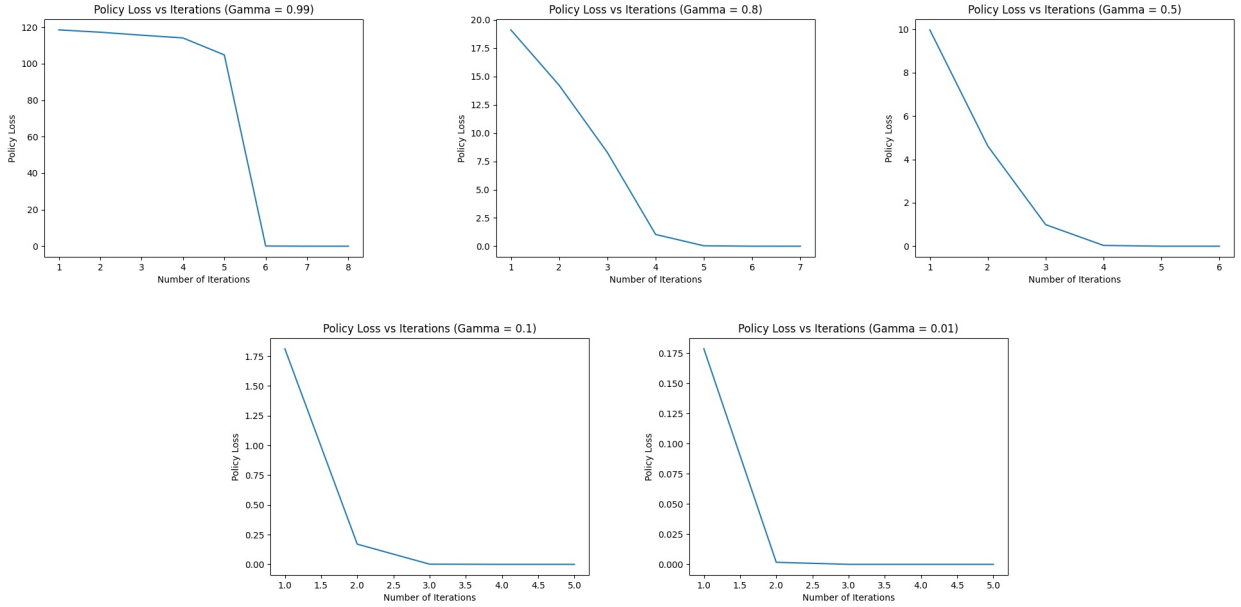

Figure 2: Convergence of Policy Iteration versus Discount Rate

We note that for small discount factor, policy iteration seemingly converges faster while for higher discount factors, policy iteration converges only after quite a few policy improvements. This can be attributed to the fact that for lower discount factor, the impact of rewards which are more than a few steps away from current state becomes negligible and hence policy more or less depends on the rewards just following the state. Thus, convergence is fast but the policy is not optimal. On the other hand, for higher discount rates, policy may converge a while later but is more optimal as it considers rewards at a later stage.

B. Incorporating Learning

Q-Learning Learning Algorithms: Convergence

Q-Learning and SARSA were implemented with decaying and constant exploration rates. The following plots show the total rewards earned in a sample episode averaged over 10 runs against the number of episodes learnt so far, for the four algorithms.

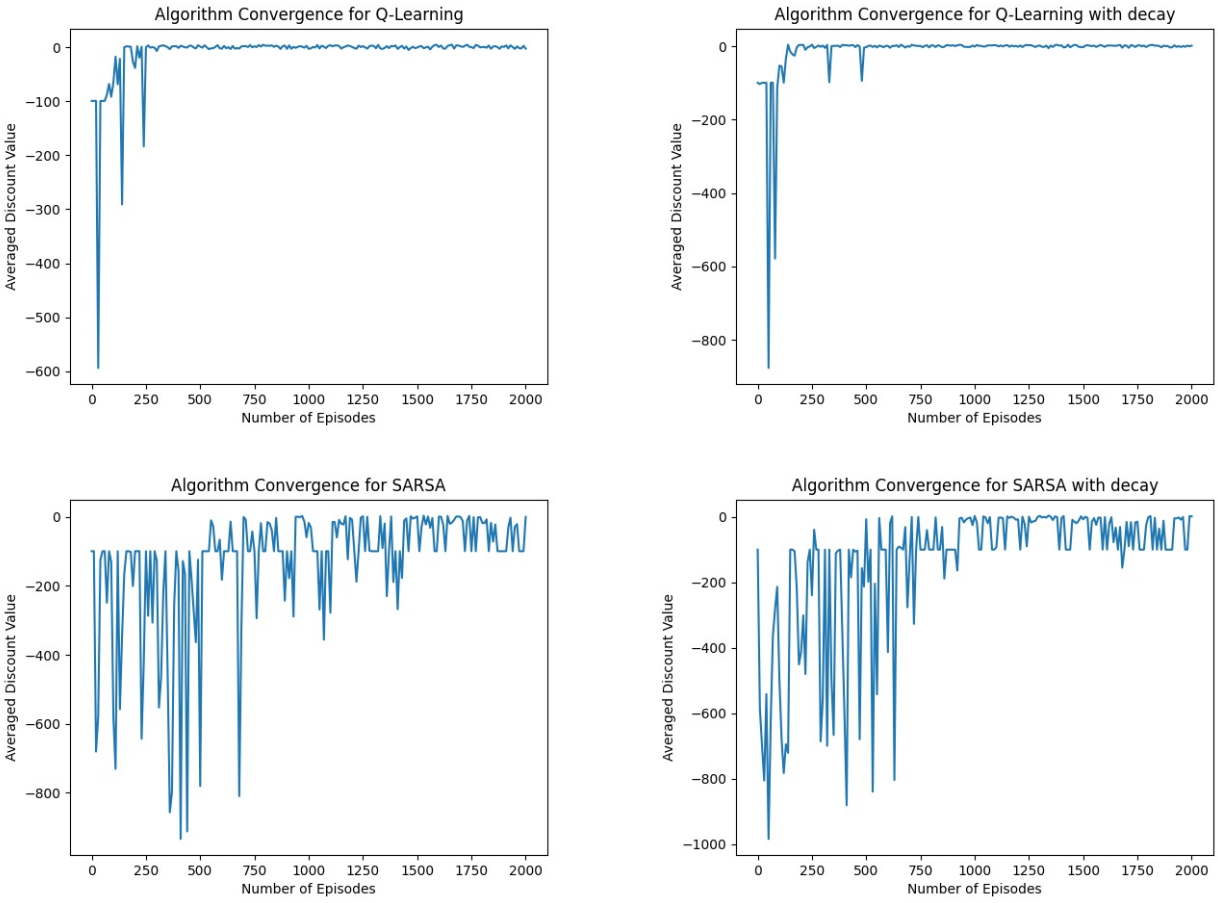

Figure 3: Convergence of Learning Algorithms

It is seen that Q-learning converges fast and smoothly as compared to SARSA. This can be explained by using the fact that Q-Learning updates Q-values with the optimal policy of next state s' i.e.

$$Q(s,a) \leftarrow (1-\alpha) \times Q(s,a) + \alpha \times [R(s,a,s') + \gamma \times max_{a'} Q(s',a')]$$

On the other hand SARSA updates Q-Value considering exploration as well, thus leading to sub-optimal Q-Value updating as in,

$$Q(s,a) \leftarrow (1-\alpha) \times Q(s,a) + \alpha \times [R(s,a,s') + \gamma \times Q(s',a')]$$

where $a' = argmax_{a'} Q(s',a'))$ with probability $1-\epsilon$ and $a' = random(Actions(s'))$ with probability ϵ. Thus, SARSA converges slow as compared to Q-Learning.

We see that learning without decay converges little faster than learning with decay, since decaying exploration rates make it difficult for the algorithm to choose a new possible action at a later stage which could be better than the action the policy suggests. However, this difference is very small because the policy has converged to a large extent and does not need much exploration at a later stage. Decaying exploration makes prevents the learner from taking random actions at later stage which could cause a little deviation from the converging value,thus preventing random sharp troughs in the graph.

Effect of Initial Locations on Reward Values using Q-Learning

We studied the effect of depots on Q-learning reward values using the same destination. We did not notice large amount of deviation in rewards as is clear from the convergence algorithms.

```
Part B3: Destination (3,0)
Case#1
Taxi Start Location: (0,4)
Passenger Pickup Location: (4,4)
Averaged Rewards Earned = 1.0825217887731808
Optimal Deterministic Reward: 4.249
Case#2
Taxi Start Location: (4,4)
Passenger Pickup Location: (0,4)
Averaged Rewards Earned = -2.90037583100555
Optimal Deterministic Reward: 2.175
Case#3
Taxi Start Location: (4,4)
Passenger Pickup Location: (0,0)
Averaged Rewards Earned = -2.3072481198153443
Optimal Deterministic Reward: 2.175
Case#4
Taxi Start Location: (0,0)
Passenger Pickup Location: (0,4)
Averaged Rewards Earned = 3.7502066757746313
Optimal Deterministic Reward: 6.366
Case#5
Taxi Start Location: (0,4)
Passenger Pickup Location: (4,4)
Averaged Rewards Earned = 0.30214314693622574
Optimal Deterministic Reward: 4.249
```

We compare the obtained values with the optimal discounted values that would be rewarded if the taxi driver knew the entire map and all actions were deterministic. We see that the obtained rewards are proportional to the maximum obtainable rewards which is expected behaviour considering the stochastic nature of actions. Thus, we see that the learned policy performs quite well in various cases.

Effect of Exploration Rates on Reinforcement Learning

We studied the convergence of Q-Learning algorithms with varying exploration rate ϵ.

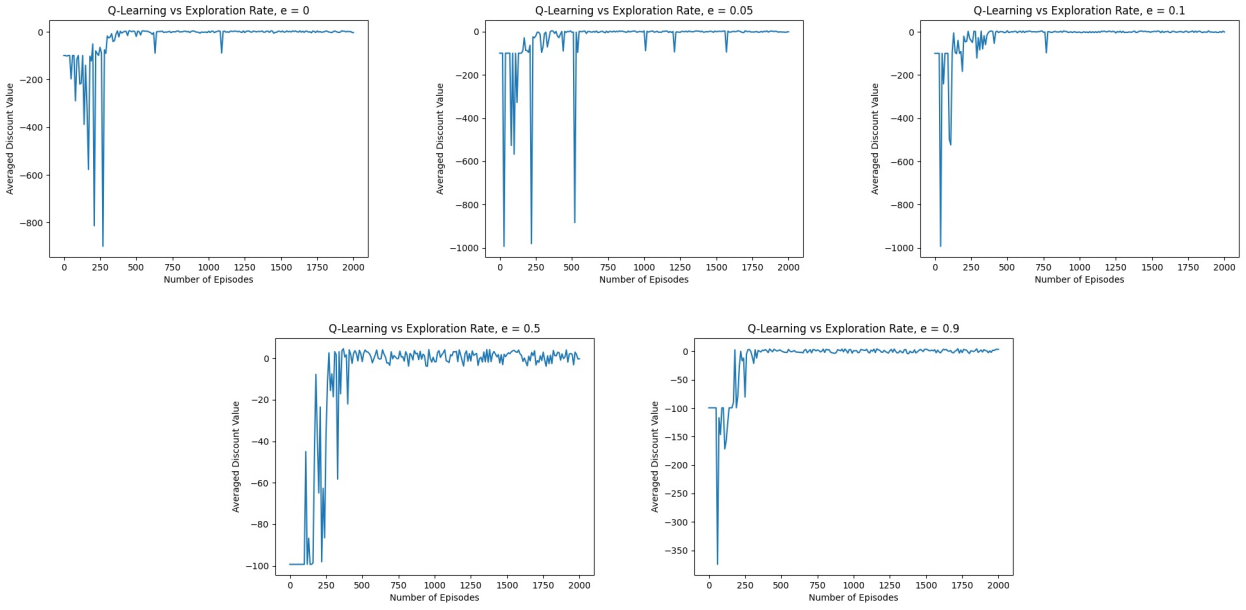

Figure 4: Convergence of Learning versus Exploration Rate

We notice that at very low and very high exploration rates, the algorithm converges faster as compared to exploration rates in the middle. This is due to the fact that at very low exploration rate, the policy only gets optimal as per the current policy and converges without much deviation. Similarly, at a very high exploration rate, the deviations from convergent values are low since the algorithm has already explored enough in the initial phase. Thus, low and high exploration rates converge faster while at intermediate exploration rates, convergence is slightly slow, with a lot of deviation from mean value which decays as the learner learns more.

Effect of Learning Rates on Reinforcement Learning

We studied the convergence of Q-Learning algorithms with varying learning rate α.

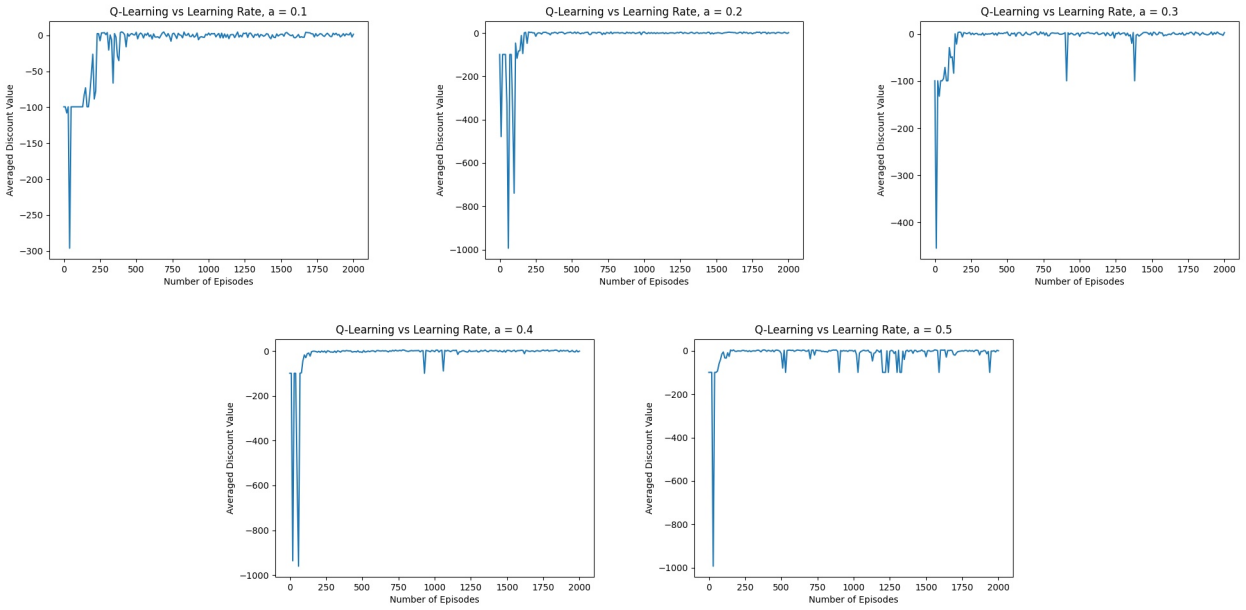

Figure 5: Convergence of Learning versus Exploration Rate

We note that as the learning rate increases, the learning algorithm converges faster. This is because the model learns the impact of rewards obtained and actions taken on the policy learnt at a higher proportion, i.e. α. However, since the factor of the current policy reduces in $1 - \alpha$ reduces, the optimality of the current policy is compromised. Thus, the sharp deviations from the mean or the effective policy increase with increasing learning rate as evident from the frequently occurring sharp troughs or nadirs. The policy will oscillate about the convergent reward if the learning rate is very high.

Q-Learning on a 10X10 Grid

We performed Q-Learning on the 10X10 extension of the taxi domain for 10000 episodes. The effect of source and destination on aggregate reward values was studied for various source and destination locations. We have also depicted the convergence for one case in case of a large grid.
We take a look at the rewards accumulated for particular start and destination locations for the same 10X10 grid. The following is the output corresponding to different start and end states. It denotes the rewards collected after running the learnt policy on given start and end states and averaging over 10 runs.

```
Case#1
Taxi Start Location: (0,1)
Passenger Pickup Location: (3,6)
Passenger Destination: (0,9)
Averaged Rewards Earned = -99.34295169575852

Case#2
Taxi Start Location: (0,1)
Passenger Pickup Location: (4,0)
Passenger Destination: (0,9)
Averaged Rewards Earned = -12.778032793307089
```

```
Case#3
Taxi Start Location: (6,5)
Passenger Pickup Location: (3,6)
Passenger Destination: (0,9)
Averaged Rewards Earned = -4.133990991126522
```

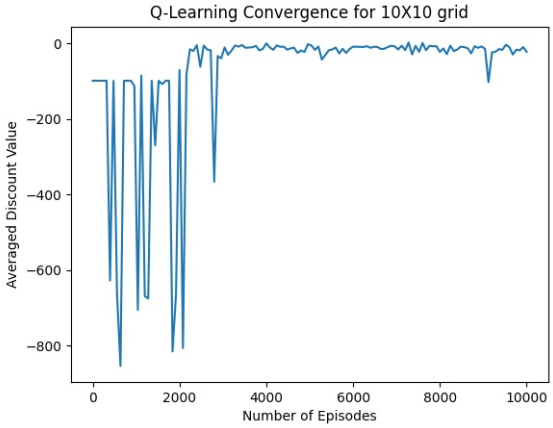

Figure 6: Q-Learning Convergence for 10X10 Grid Case#3

```
Case#4
Taxi Start Location: (0,1)
Passenger Pickup Location: (3,6)
Passenger Destination: (9,0)
Averaged Rewards Earned = -185.81485457477342

Case#5
Taxi Start Location: (0,1)
Passenger Pickup Location: (4,0)
Passenger Destination: (9,0)
Averaged Rewards Earned = -342.06993566837264

Case#6
Taxi Start Location: (6,5)
Passenger Pickup Location: (3,6)
Passenger Destination: (9,0)
Averaged Rewards Earned = -99.34295169575852
```

When we compare the rewards earned by the learnt policy after 10000 episodes with the optimal deterministic discounted rewards, i.e. discounted rewards that would be obtained if the actions were deterministic and were taken optimally, we find that the policy works pretty well taken the stochastic nature of actions and sub-optimality of the policy into account.

Conclusion

In this assignment, we formulated a Markov Decision Process corresponding to given taxi passenger problem in a 2-D grid and solved it using Value and Policy Iteration. We looked at the effect of start and destination locations and discount factor on these methods. Then we performed ϵ-greedy Q-Learning and SARSA learning with constant and decaying exploration rates. We looked at the impact of starting and destination locations, learning rate, exploration rate and the algorithm on the convergence and optimality of the policy. We then extended Q-Learning algorithm to a 10×10 grid as well. Thus, we had a deeper look at Markov Decision Processes and Reinforcement Learning in this assignment.