# ASSIGNMENT 3, COL333

Sarthak Singla 2019CS1039, Lalit Meena 2019CS50439

## PART A: Computing Policies

1. Formulating Taxi Domain problem as an MDP

   a. We formulate the Taxi Domain Problem as the following MDP -

   **Environment**
   Map grid, depot locations and passenger's destination

   **State**
   Taxi position, passenger position, and whether passenger is inside taxi if passenger and taxi are at same position.
   Our state is defined by an object containing values tx, ty, px, py, s. Here (tx,ty) is the taxi position, (px, py) is the passenger position, and (s) = 0 if passenger is outside taxi and 1 if inside taxi.
   States with taxi position and passenger position different cannot have s=1.
   Terminal States are states with passenger outside taxi and at the destination.

   **State Space**
   All possible states with positions inside grid as defined above.
   For an nxn grid:
   Passenger out of taxi: nxn positions for taxi and nxn positions for passenger. So total $n^4$ states.
   Passenger inside taxi: nxn positions for taxi and passenger position is same as taxi. So total $n^2$ states.
   Hence in all $n^4 + n^2$ states for an nxn grid will be there. In our case of 5x5 grid, we have 650 states.

   **Action Space**
   NORTH, SOUTH, EAST, WEST, PICKUP, PUTDOWN actions are possible at each state

**Transition Model T(s,a,s'):**
We assume a stochastic environment. Therefore, the action executed may not be same as action taken. Also, the state reached depends only on the action executed. Transition Model gives the probability of reaching s' after taking action a at state s.

| Action(a) | T(s,a,s') | |
|---|---|---|
| NORTH/SOUTH/EAST/WEST | 0.85 | if taken action executed |
| | 0.05 | if other movement executed |
| PICKUP/PUTDOWN | 1 | if taken action executed |
| | 0 | otherwise |

If s has passenger at destination, and passenger out of taxi, then episode has terminated and no actions are taken thereafter.

**Reward Model R(s,a,s'):**
Reward Model gives the reward got on reaching state s' after taking action a at state s. For feasible combinations of s,a,s' according to the Transition Model:

s= Taxi,Passenger at destination and Passenger inside Taxi
a= PUTDOWN
s'=Taxi, Passenger at destination and Passenger outside Taxi
R(s,a,s') = +20

s= Taxi,Passenger not at same position, passenger not at destination
a= PICKUP/PUTDOWN
s'= s (the only feasible state)
R(s,a,s') = -10

Any other valid combination of (s,a,s')
R(s,a,s') = -1

b.

The simulator has been implemented as
taxiDomainInstance.simulate(state, action)

It takes an instance of Taxi Domain problem, a state and an action as input and returns the next state and the reward obtained.
Here also print_simulate function to simulate on terminal by taking current state and action. So results get printed on terminal

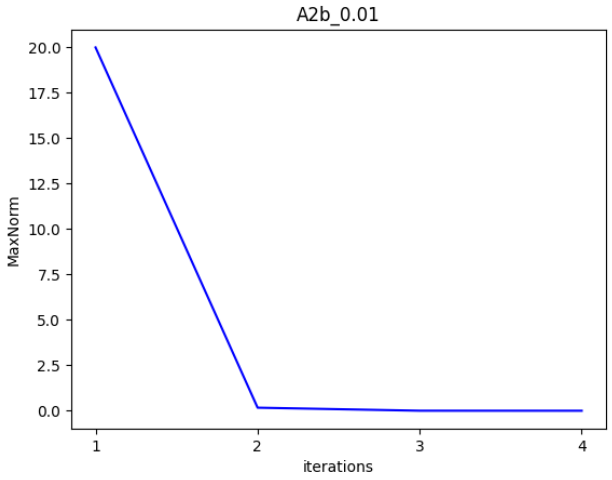
2. Implement Value Iteration for Taxi Domain problem
    a.

    In this problem, we take an instance of the Taxi Domain problem, convergence factor epsilon and discount factor gamma.
    We initialise the values as 0 and on each iteration, we update values of all the states using a 1 step lookahead.

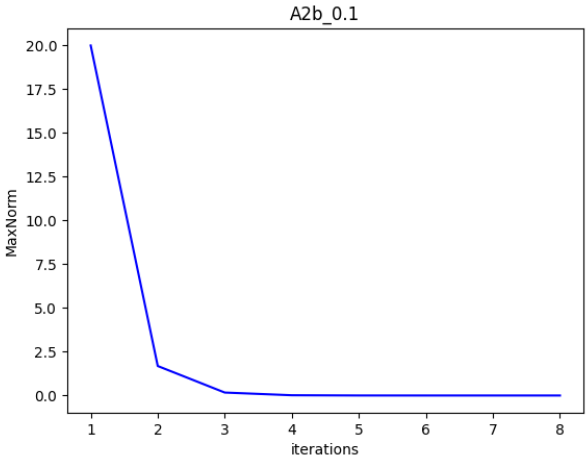
    The convergence criteria used is that MaxNorm distance between optimal values and current values should be less than epsilon. This translates to MaxNorm between successive values is less than epsilon(1-gamma)/gamma.
    For the terminal states, where passenger is outside taxi and at destination, we fix value as 0.

    Epsilon used -> 1e-6
    Discount given -> 0.9
    Number of iterations -> 153

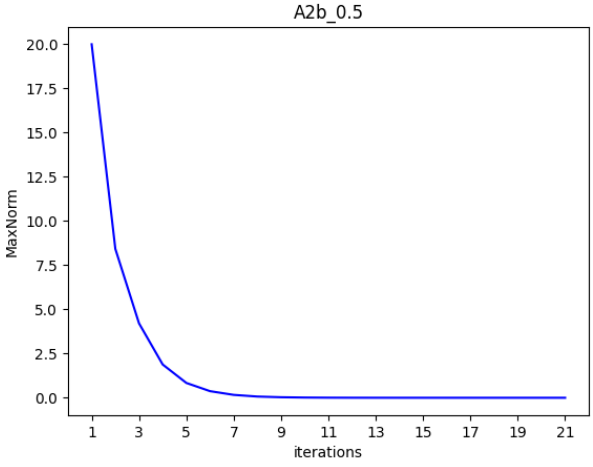
    b.



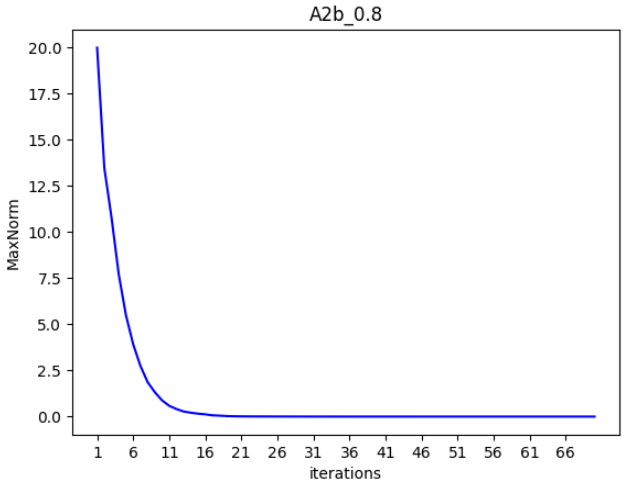

gamma = 0.01



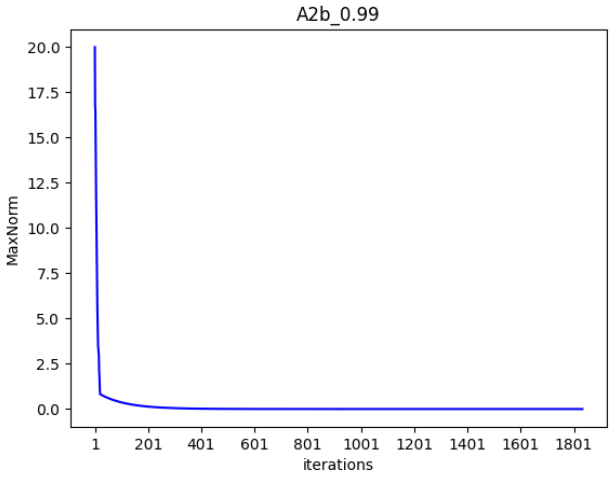

gamma = 0.1




gamma = 0.5




gamma = 0.8




gamma = 0.99

From the plots above, we see that the MaxNorm vs Iterations curve decreases quickly and then slowly.

We also observe that the rate of convergence decreases with an increase in discount factor(gamma).

| **Discount Factor** | **Iterations** |
|---|---|
| 0.01 | 4 |
| 0.1 | 8 |
| 0.5 | 21 |
| 0.8 | 70 |
| 0.99 | 1833 |

c.

In this question, we fixed a destination for passenger, and picked 3 pairs of start states for the passenger and taxi. On each of these, we obtained the policy for discount factors 0.1 and 0.99 and simulated the first 20 state action pairs, using the function defined for part A1b repeatedly, with current state and action obtained from policy.

The execution traces for discount 0.1 showed the taxi almost always taking action NORTH(constant direction implementation dependent), maybe because the immediate rewards are most favourable. Therefore, in 20 state-actions, it doesn't even pickup the passenger. Only on decreasing epsilon a lot, discount 0.1 policy would actually try doing optimal actions.

The execution trace for discount 0.99 showed the taxi taking optimal actions. Within 20 state-action pairs, it is able to pickup the passenger and putdown at the destination.

3. Implement Policy Iteration for the problem.

a.

In this problem, we take an instance of Taxi Domain Problem, the discount factor and which method of Policy Evaluation is to be used.
Policy Iteration was implemented by first taking an arbitrary policy(all actions NORTH). Until convergence, first, we get values of states using current policy by iterative or linear algebra method. Then, we do one step lookahead at the values obtained to get a new policy. Convergence occurs when two successive policies are the same.

**Policy Evaluation - Iterative Method**

We use the the current policy and iteratively find values of states using the equation - $V^{\pi}_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s'))$

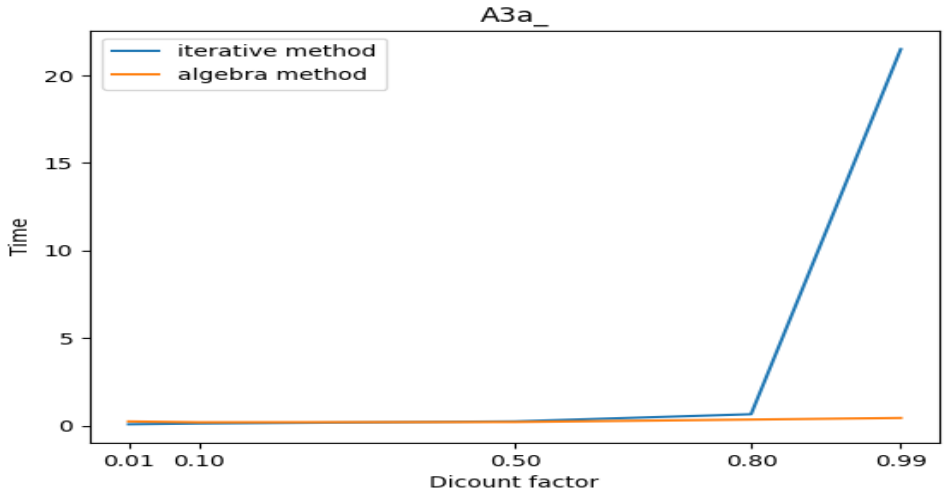
**Policy Evaluation - Linear Algebra Method**

In this method, we solve for V(s) for all s using AX = B in the following equations:

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

Linear Algebra method requires O(nxnxn) time for n states to evaluate the policy. Therefore, if n is small, as in our case, linear algebra method is favourable and faster than iterative method.

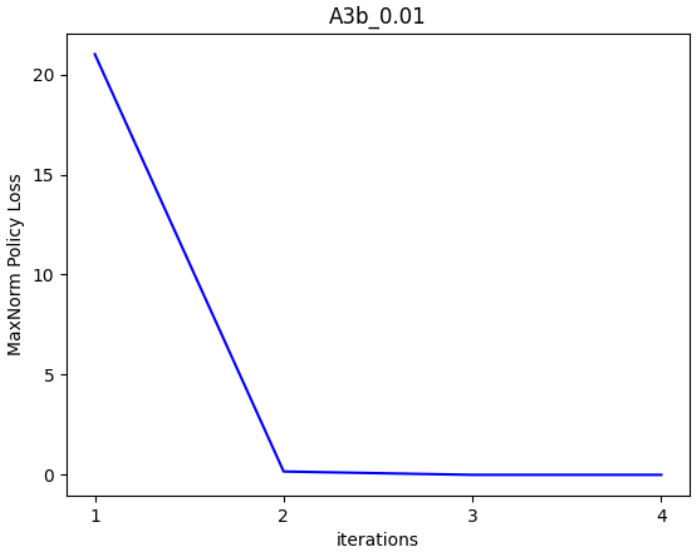
However, in policy evaluation we don't require exact values, a fairly good estimate of values is also OK. In iterative method, we are able to stop at some distance from the actual values after some number of iterations. So, for large state spaces, iterative method doesn't grow in time and is preferred.



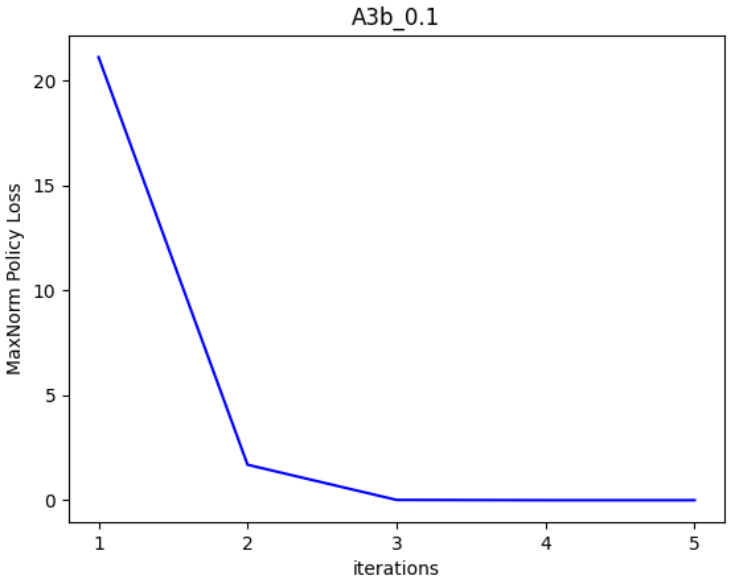

Here we also observe that linear algebra method takes less time than iterative method for larger discount factors.

b.

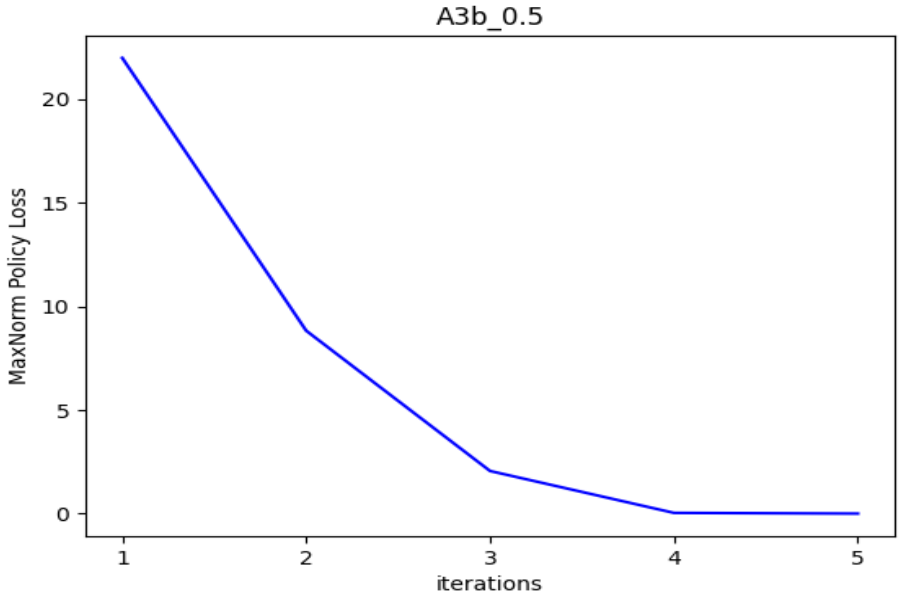
We see that policy loss decreases with an increase in iterations. Also, the rate of decrease of policy loss decreases with increase in iterations. With all discount factors, the number of iterations required is roughly the same.



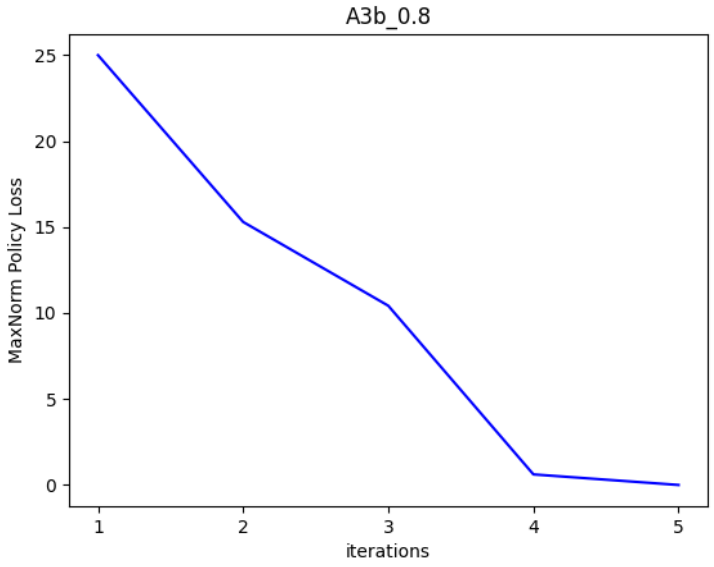

gamma=0.01



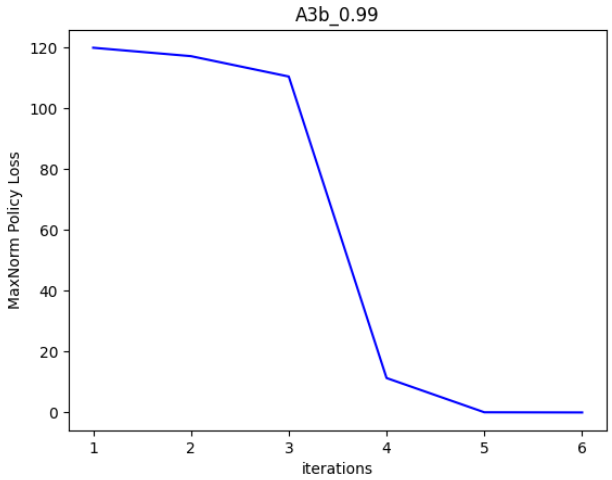

gamma=0.1




gamma=0.5




gamma=0.8




gamma=0.99

# PART B: Incorporating Learning

1. Implemented model-free approaches to learn an optimal policy
   Any algorithm can be called with required parameters.

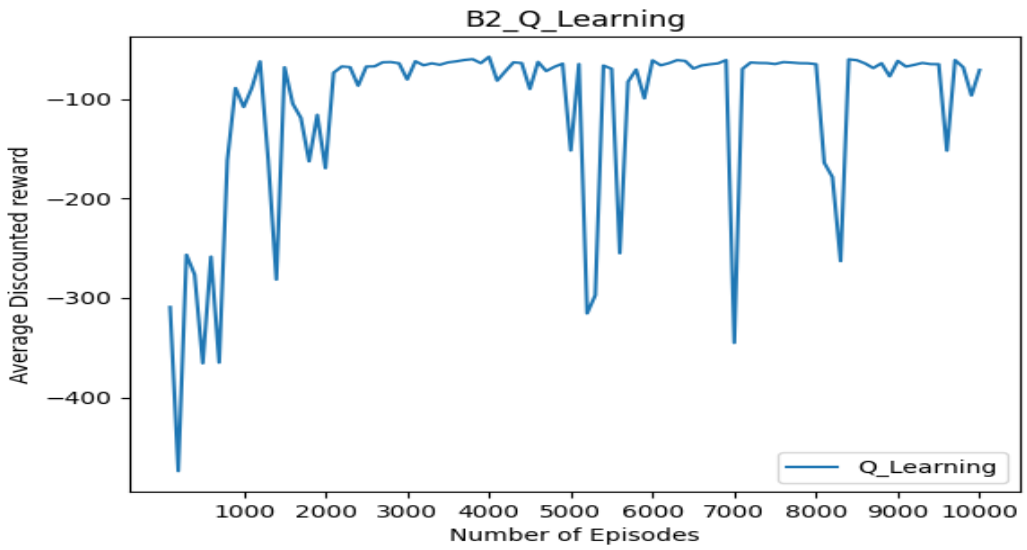
2. In this part we executed Q Learning, Q Learning with decay, SARSA and SARSA with decay for 10000 random episodes (initialised with taxi - anywhere in grid, passenger - any depot except destination). Each episode is run for max 500 steps.

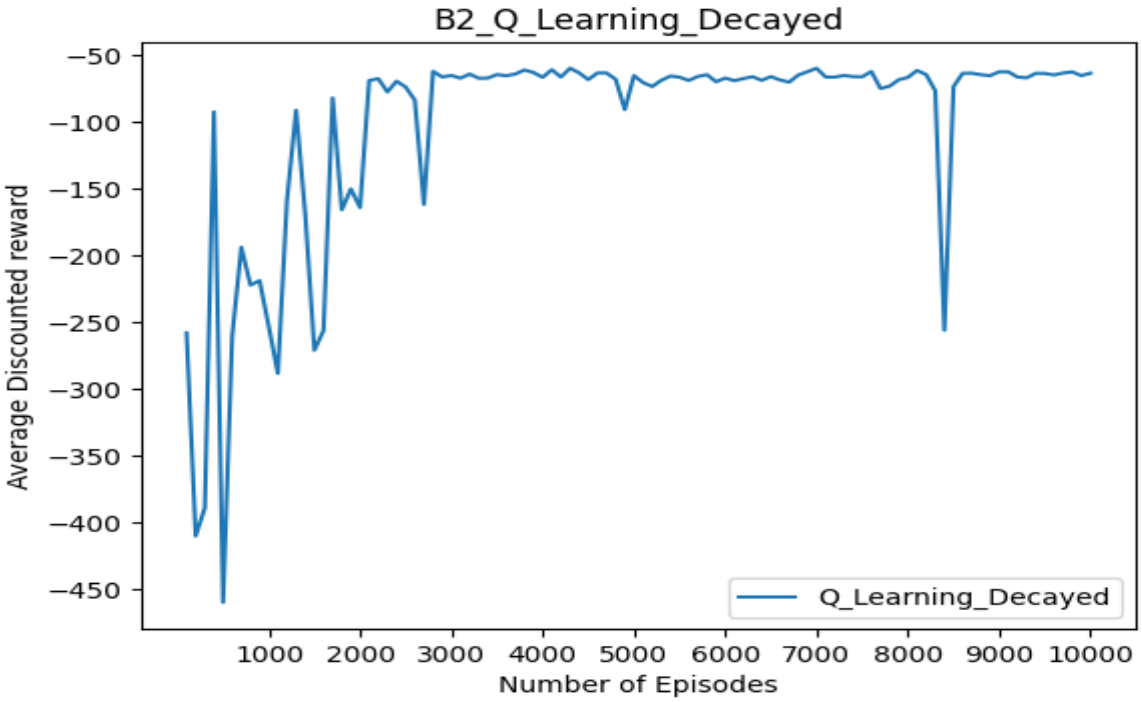
   Discount factor = 0.99
   Alpha = 0.25

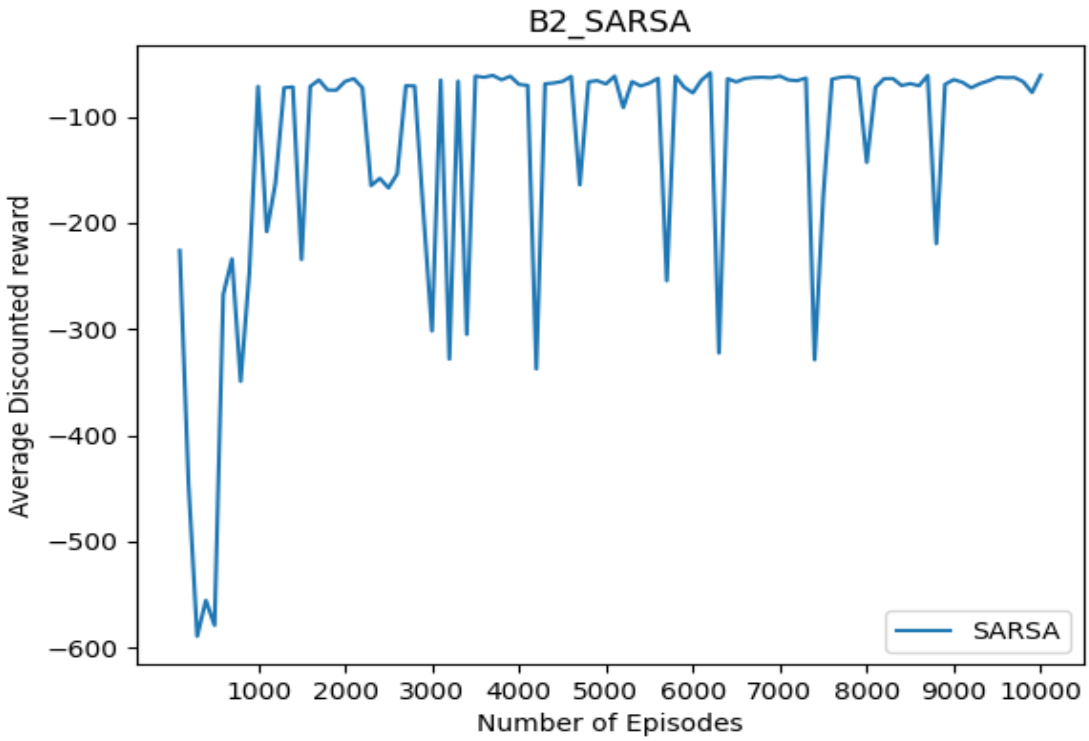
   To plot the discounted rewards versus number of episodes in training, we find the policy at every 100th episode. To evaluate the policy, we use 10 previously generated random episodes, and run each 10 times. We take the average discounted reward over these to plot.

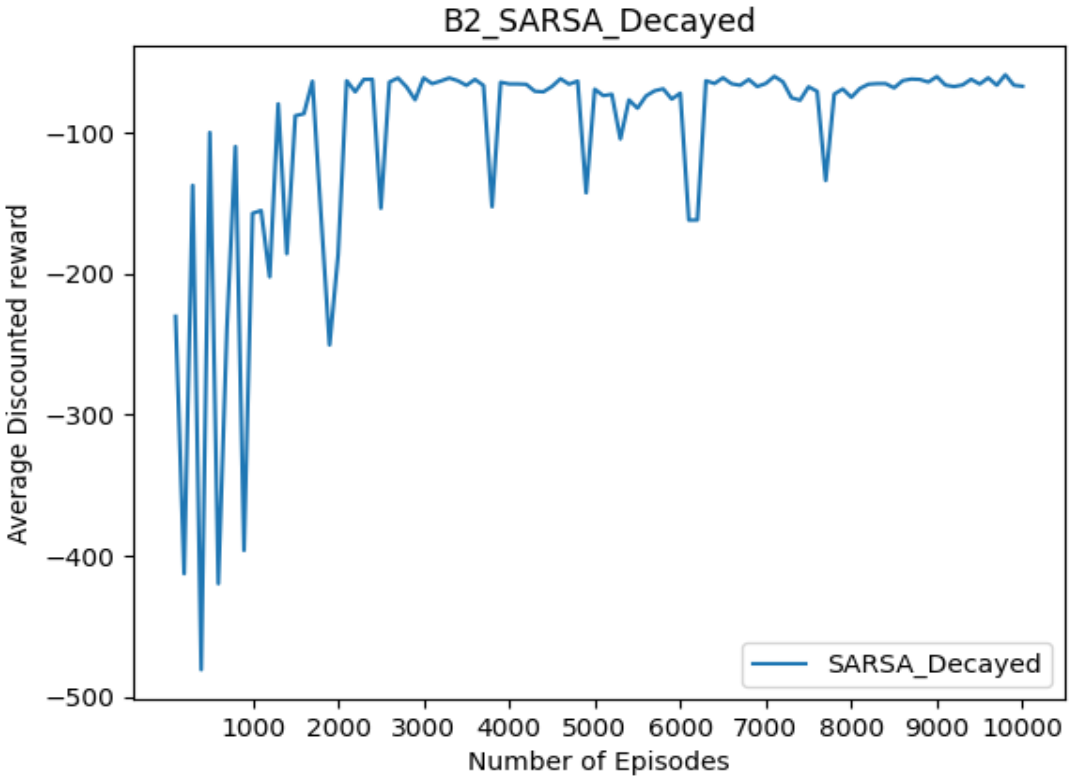
   We observe that the average discounted rewards start converging with increase in number of episodes to around 9000. We also observe that due to decay factor involved, the decaying exploration rate algorithms converged faster. As with increase in episodes the exploration rate is decreased.

**PART B2 graphs for different Algorithms-**




B2_Q_Learning_Decayed
Average Discounted reward
−50
−100
−150
−200
−250
−300
−350
−400
−450
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Number of Episodes
Q_Learning_Decayed


B2_SARSA
Average Discounted reward
−100
−200
−300
−400
−500
−600
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Number of Episodes
SARSA




3.

The best algorithm found by us was Q Learning with decay. It was the one showing the convergence to the highest value in PARTB2. Also, because of the decay, it stopped exploring too much after a number of episodes had already occurred. Q Learning with decay is better than SARSA with decay as Q Learning learns off policy, while SARSA learns on policy and is biased with policy.

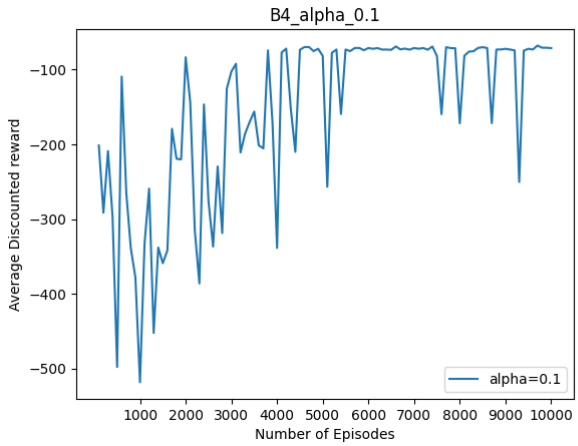
We chose 5 random instances and execute the policy on them.

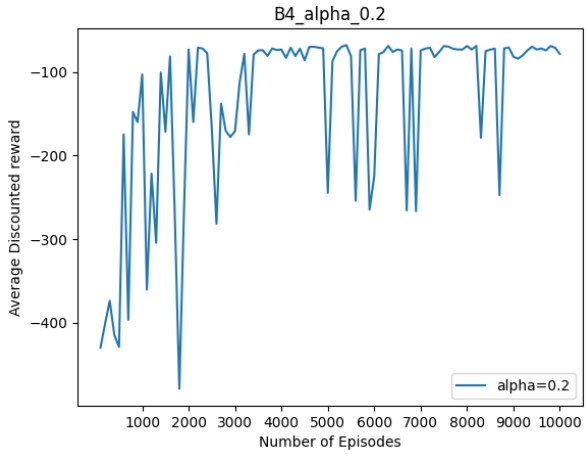
We observe that the expected reward comes out to be near -100 in almost all the cases irrespective of the initial instance of Taxi Domain problem. This is because the expected reward doesn't depend on the initial states of the taxi and passenger much as model depends on destination.
[7.4, -99, -99, -47, -99] -> observed rewards on 5 random instances of the Taxi Domain problem after running Q Learning with decay for 10,000 episodes, epsilon=0.1, alpha=0.25 and gamma=0.99

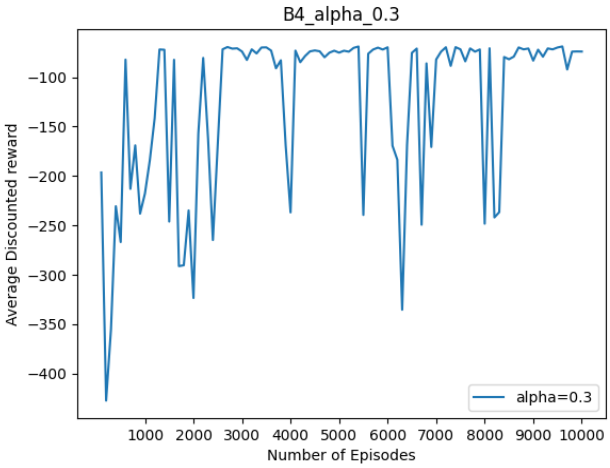
4.

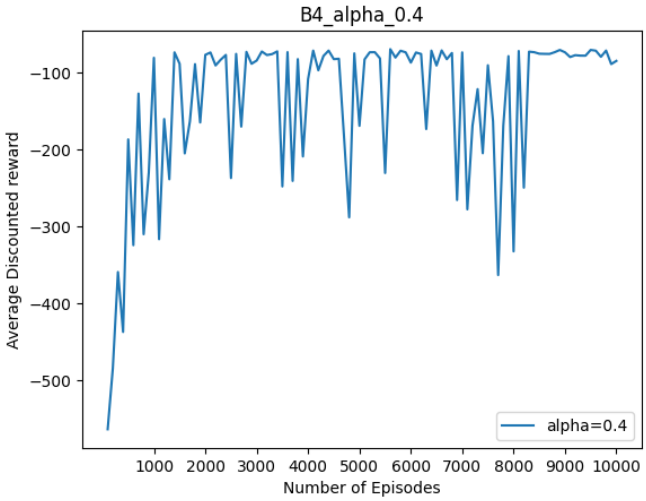
With increasing alpha, we give more weigthage to samples in our Q values than the old value.
From the plots we are able to observe that on increasing alpha the convergence becomes faster, because each sample gets more weightage.
However when alpha is very large, then undue weightage is given to samples and therefore convergence becomes slower. Convergence can be seen from the graphs below.

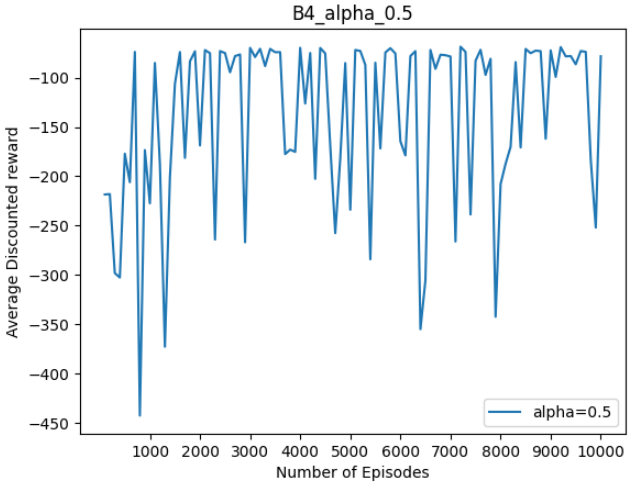
**Epsilon fixed at 0.1**



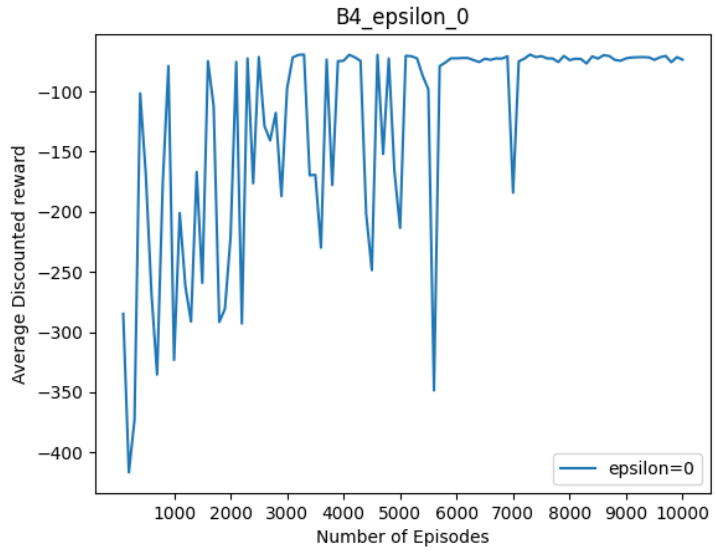

B4_alpha_0.3
Average Discounted reward
−100
−150
−200
−250
−300
−350
−400
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
Number of Episodes
alpha=0.3

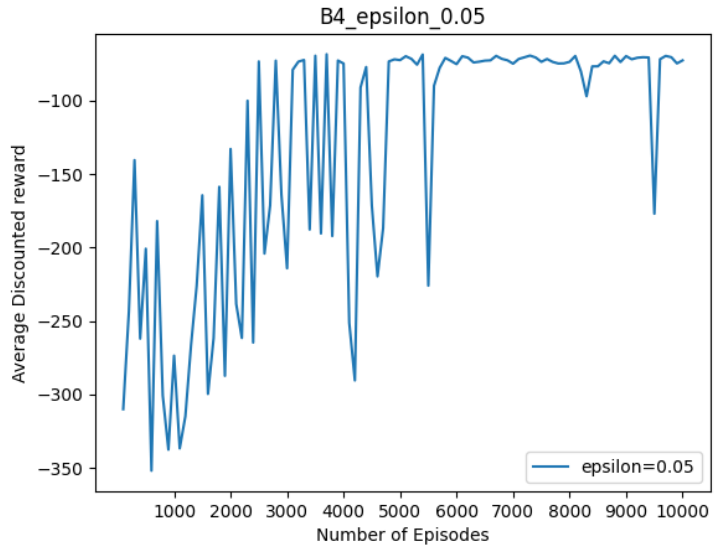

B4_alpha_0.4
Average Discounted reward
−100
−200
−300
−400
−500
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
Number of Episodes
alpha=0.4

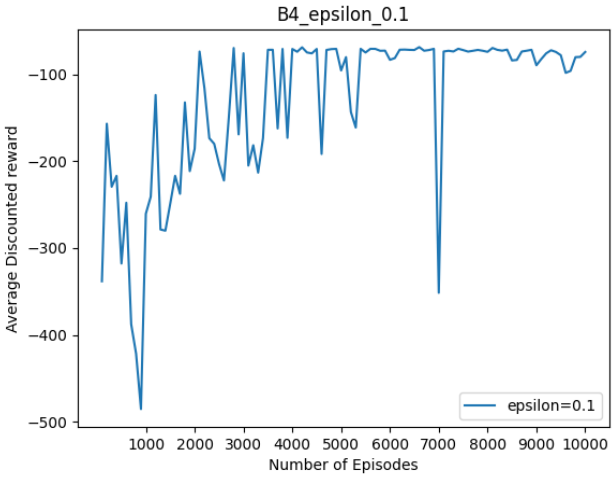

B4_alpha_0.5
Average Discounted reward
−100
−150
−200
−250
−300
−350
−400
−450
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
Number of Episodes
alpha=0.5

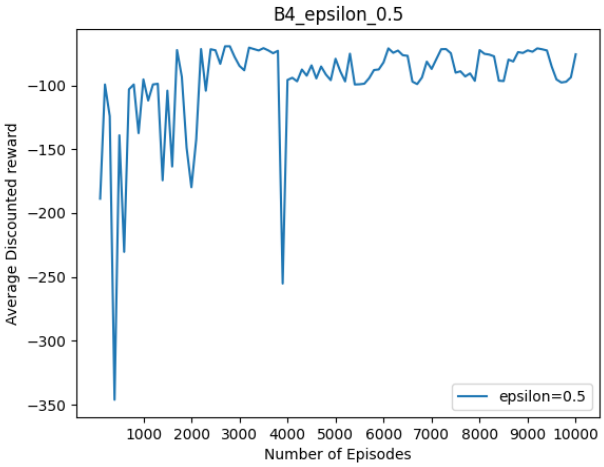
From the plots below, we see that on increasing epsilon, the convergence becomes faster, and with less episodes, the average discounted reward achieves a converged value. This is because, on increasing epsilon, the agent is able to explore more and thus is able to find best policy faster.

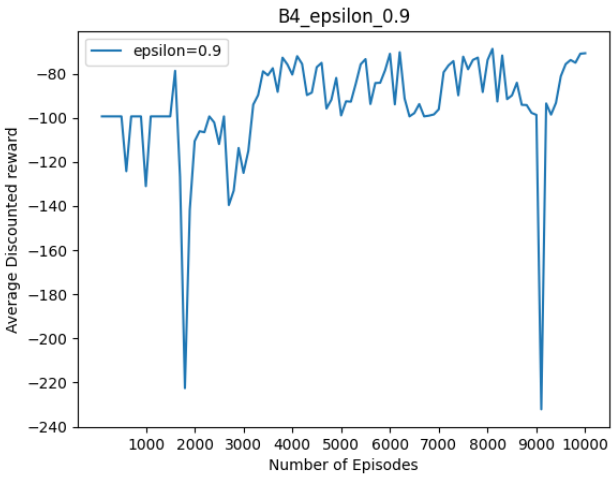
**Alpha fixed at 0.1**




B4_epsilon_0.1
Average Discounted reward
Number of Episodes
epsilon=0.1
−100
−200
−300
−400
−500
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000


B4_epsilon_0.5
Average Discounted reward
Number of Episodes
epsilon=0.5
−100
−150
−200
−250
−300
−350
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000


B4_epsilon_0.9
Average Discounted reward
Number of Episodes
epsilon=0.9
−80
−100
−120
−140
−160
−180
−200
−220
−240
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000

5.

We chose the Q Learning with decay algorithm with decay to run over the 10x10 grid. We run the algorithm for 10,000 episodes and evaluate the policy obtained on 5 random episodes. We repeat the process for 5 instances of Taxi Domain on the 10x10 grid.

**Different destinations for 5 Models:**
[(4, 0), (9, 0), (0, 1), (6, 5), (3, 6)]

**Average discounted rewards for 5 Models:**
[-45.43974914033866, -75.87064176112713, -195.0148910209288, -213.5203864890275, -65.0484004163803]

We see that the average discounted rewards for different destinations converge to different values. The average rewards differs more in this part as compared to part 3. This is because, our model depends on destination only, not on taxi and passenger start position.

**Running script**
We can run parts in question by these simple commands on terminal.

Python3 A3.py part sub_part

Here passed arguments to A3.py script file are-
First passed argument is **part** which can take value - **a or b**
Second passed argument is **sub_part** which takes following values for parts
Part A- 1a,2a,3a..
Part B-1,2,3,4,5