Assignment 4

This report investigates Policy Iteration, Value Iteration and Q-Learning algorithms and applies these three algorithms to Markov Decision Process (MDP) problems: The Taxi game (Grid World Problem) and the Forest Management problem (Non-Grid World Problem).

Introduction

Multiple Reinforcement Learning techniques to make decisions within an environment are investigated explicitly in this assignment. Analysis of two MDPs is performed using Policy and Value Iteration algorithms, and a model free Q Learning algorithm. The behavior of the three algorithms is tested using small and large number of states.

I will start by a brief introduction about Markov Decision Process, Policy and Value Iteration, Q Learning, and finally I will talk about the Taxi game and the Forest Management problem.

After that, I will talk about the observations and analysis of the experiments conducted in this assignment, then I will conclude this assignment in the final section.

Markov Decision Process (MDP)

MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions. More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3,…. At each time step t, the agent receives some representation of the environment's state (S), and on that basis selects an action (A). One time step later, in part as a consequence of its action, the agent receives a numerical reward (R) .

In terms of Reinforcement Learning the Environment is modelled as a markov model and the agent needs to take actions in this environment to maximize the amount of reward.

Policy and Value Iteration Algorithms

In **Policy Iteration** - We randomly choose a policy and find value function corresponding to it, then find a new improved policy compared the previous value function, and so on this will lead to optimal policy. Policy iteration works on principle of "Policy evaluation ➔ Policy improvement".

In **Value Iteration** - Works by applying the **Bellman equation** to evaluate progressively state-by-state until the solution converges. We randomly choose a value function , then find a new improved value function in an iterative process, until reaching the optimal value function, then derive optimal policy from that optimal value function

Value Iteration works on principle of " Optimal value function ➔ optimal policy".

Comparison between Policy and Value Iteration algorithms is shown in the below table

Policy Iteration	Value Iteration
• Starts with random policy • Algorithm is complex • Cheaper to compute • Guaranteed to converge • Fast • Requires less iterations to converge	• Starts with random value function • Algorithm is simple • Expensive to compute • Guaranteed to converge • Slow • Requires more iterations to converge

Q-Learning Algorithm

The model free algorithm works by giving all the states a Q-value. Q is initialized to a random fixed value (selected by the programmer). Then, at every time (t) the agent takes an action (A), observes a reward (R) and enters a new state, and Q is updated based on immediate and delayed rewards. The core of the algorithm is the Bellman equation as a value iteration update, using the weighted average of the old value and the new information.

This is very much like the concept of exploration and exploitation in random optimization. Q-Learning balances between executing immediate rewards and learning to explore delayed rewards. Q-Learning will explore more at the early stage, but as it progresses, it will begin to exploit more.

The Taxi Game

Taxi game is one of the built-in games in Gym. There are four designated locations in the grid world indicated by R(ed), B(lue), G(reen), and Y(ellow). When the episode starts, the taxi starts off at a random square and the passenger is at a random location.

The taxi drive to the passenger's location, pick up the passenger, drive to the passenger's destination (another one of the four specified locations), and then drop off the passenger. Once the passenger is dropped off, the episode ends.

Rewards: There is a reward of -1 for each action and an additional reward of +20 for delivering the passenger. There is a reward of -10 for executing actions "pickup" and "drop-off" illegally.

Rendering: Blue: passenger – Magneta: destination – Yellow: empty taxi – Green: full taxi – Other letters: locations

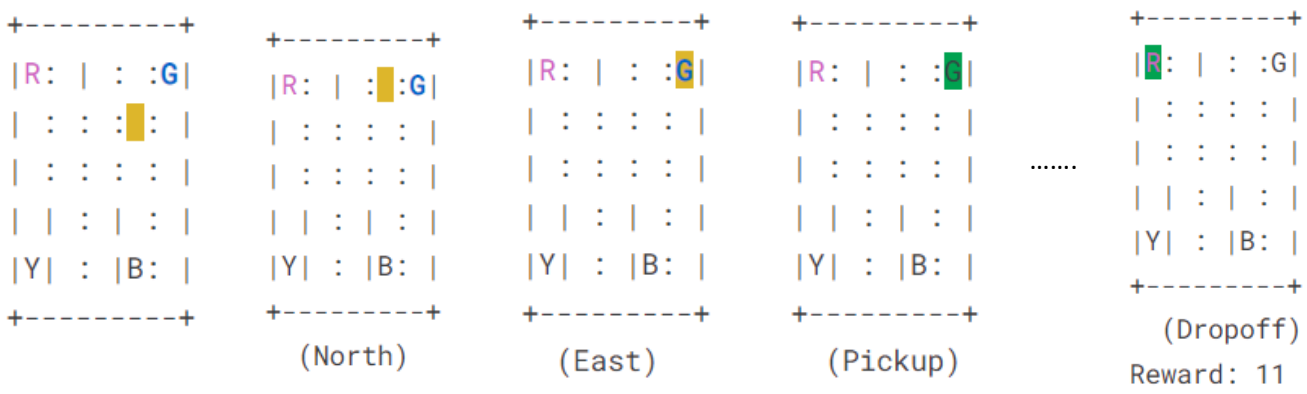
Each timestep, the agent chooses an action, and the environment returns an observation and a reward.

observation, reward, done, info = env.step(action)

- observation: an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game line Taxi.
- reward: amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
- done: whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes and done being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)
- info: ignore, diagnostic information useful for debugging. Official evaluations of your agent are not allowed to use this for learning.

Action Space: (0=South, 1=North, 2=East, 3=West, 4=Pickup Passenger, 5=Drop-off Passenger)

some random steps are shown below in the game, so we see how the game looks like:



Forest Management Problem

A forest is managed by two actions: 'Wait' and 'Cut'. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability P that a fire burns the forest. Here is how the problem is modelled. Let {0, 1 . . . S-1 } be the states of the forest, with S-1 being the oldest. Let 'Wait' be action 0 and 'Cut' be action 1. After a fire, the forest is in the youngest state, that is state 0.

Implementation

1. Gym Taxi game implemented with 500 states (small number of states).
2. Value and Policy Iteration algorithms tested and the algorithms behavior is recorded by plotting number of iterations vs average rewards using different values of Gamma (Discount factor). Also, the convergence time is plotted vs number of iterations.
3. Q-Learning algorithm: A code written from scratch to implement the Q-Learning algorithm, the reason for writing the code from scratch is to investigate the exploration and exploitation features of the algorithm by changing Epsilon value. Code snippets is shown below for the criteria of choosing Epsilon and if we need to explore or exploit. This can be verified in the code file.

   Different values for Gamma (Discount factor), Alpha (Learning rate) and Decay rate are tested and the number of episodes vs total reward are plotted as well in the process of tuning the hyperparameters of the Q-Learning algorithm.

```
    ### STEP 2: SECOND option for choosing the initial action - exploit
    #If the random number is larger than epsilon: employing exploitation
    #and selecting best action
    if exp_exp_tradeoff > epsilon:
        action = np.argmax(Q[state,:])

    ### STEP 2: FIRST option for choosing the initial action - explore
    #Otherwise, employing exploration: choosing a random action
    else:
        action = env.action_space.sample()


    ### STEPs 3 & 4: performing the action and getting the reward
    #Taking the action and getting the reward and outcome state
    new_state, reward, done, info = env.step(action)

    ### STEP 5: update the Q-table
    #Updating the Q-table using the Bellman equation
    prev_Q=Q.copy()
    Q[state, action] = Q[state, action] + alpha * (reward + discount_factor * np.max(Q[new_state, :]) - Q[state,
    #Increasing our total reward and updating the state
    total_training_rewards += reward
    state = new_state
```

```
    #Cutting down on exploration by reducing the epsilon
    epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay*episode)

    #Adding the total reward and reduced epsilon values
    training_rewards.append(total_training_rewards)
    epsilons.append(epsilon)
    training_times.append((time.time()-init_time)*1000)


return training_rewards,training_times,Q
```

4. Gym Taxi game implemented with 2520 states (large number of states).
5. Steps 2&3 are repeated for the Taxi game with 1680 states.
6. Forest Management problem implemented with 10 and 20 states.
7. Steps 2&3 are repeated for the Forest Management problem.

Taxi Game 500 States

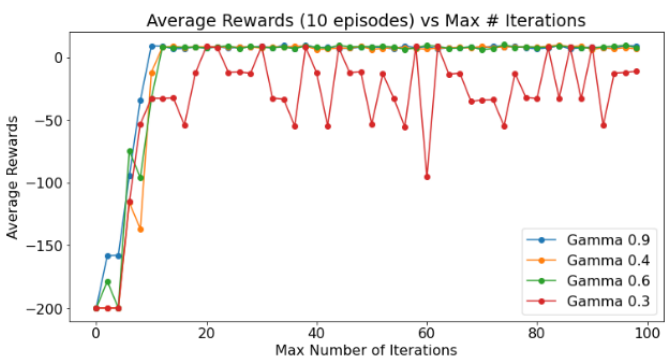
The Taxi game has 500 possible states: 25 squares, 5 locations for the passenger, and four destinations. Therefore, 25*5*4 = 500 states. How the Taxi game works was explained in the previous sections, so I will move forward to the discussion and analysis of the behavior of the 3 reinforcement learning algorithms applied on the Taxi game.

1. Policy and Value Iteration Algorithms

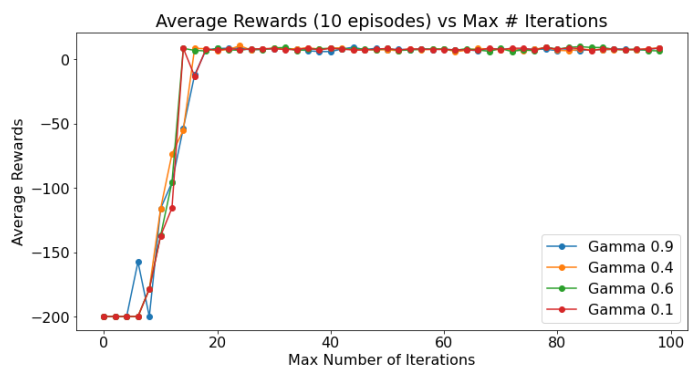
In this experiment, I am using 10 episodes and Gamma values = [0.3, 0.4, 0.6, 0.9]. As shown below, VI and PI have almost the similar behavior in the number of iterations they need to reach convergence. However, we can see that VI used fewer iterations than PI to reach convergence. If we take Gamma = 0.9, we can see that at this discount factor value VI converged at iteration = 6, while PI converged at iteration = 15. This is very interesting result because we should expect the opposite to happen, where PI uses fewer iterations to converge than the VI. More investigation on the initial state assignment is needed to track this behavior. On the other hand, we can see that the time taken per iteration for PI is 50% greater than the time taken per iteration for VI. This behavior is expected due to the complexity of the PI algorithm. Furthermore, it is clear from these plots that PI algorithm is more stable than VI in terms of convergence and average rewards per all Gamma values for this number of states in the Taxi environment.

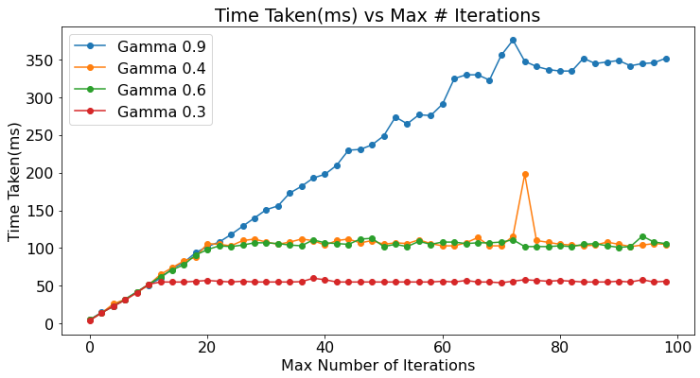
It is important to note that the average reward for all Gamma values in VI algorithm is approx. 7.9 except for Gamma = 0.3, the average reward for discount factor was -19. For PI algorithm, we got average reward value of 7.9 for all Gamma values.

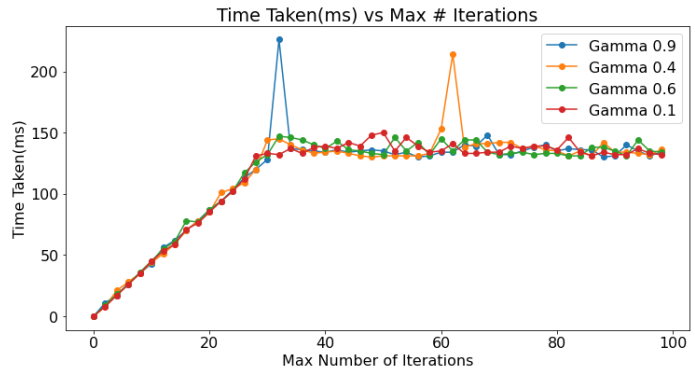
Value Iteration



Policy Iteration







2. Q-Learning Algorithm

We will move to interesting part of this assignment which is the model-free Q-Learning algorithm. Learning rate (Alpha), Discount factor (Gamma) and Epsilon-Decay were the main focus in this part, but first let me give a brief introduction about these 3 hyperparameters of the Q-Learning algorithm:

Learning rate (Alpha): Learning rate is how big we take a step in finding optimal policy. In the terms of Q-Learning it's how much we are updating the Q value with each step.

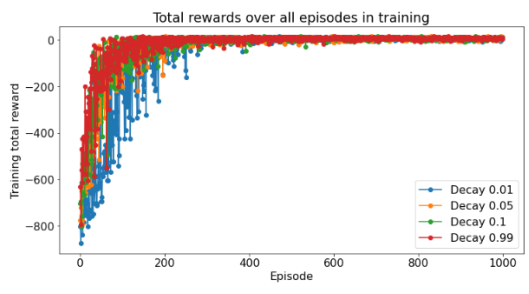
Epsilon-Decay: Epsilon is used when the agent chooses certain actions based on the Q values. For example, if we choose the greediest method (Epsilon = 0), then we are choosing the highest Q value in the Q table for a particular state. Epsilon = 0 means ZERO exploration and highest exploitation. Epsilon values should be between 0 and 1. Intuitively, we need to start with high Epsilon value (Epsilon = 0.7 for example) to let the algorithm explore and learn more and we can assign Epsilon_Decay value, also between 0 and 1, to make the algorithm exploit more as it progresses and learns.

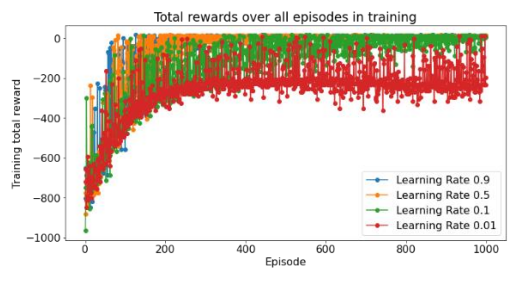
Now let's move to the analysis and the results of the Q-Learning algorithm to train an agent to solve the Taxi game. As shown below, 3 plots for each Q-Learning hyperparameter with different values assigned to these hyperparameters to test the algorithm behavior and sensitivity towards these parameters. By looking at the 3 plots, the behaviors looks consistent with the parameter values.

In the first plot on the left-hand side, it is expected to see higher fluctuations in the reward values corresponds to Gamma = 0.3 because the algorithm is biased towards immediate rewards rather than the long-term rewards. By increasing the discount factor, we can see that these reward fluctuations started to decrease because we are driving the model to be biased towards the long-term rewards.

In the middle plot, we assess the Epsilon decay rate and how it affects the model. Decay rate controls the exploration and exploitation concept of the algorithm. When we chose a Decay rate = 0.01, we are giving the algorithm the opportunity to explore and learn more than being greedy since we start with Epsilon = 0.99 as a default value and that explains the fluctuations in the reward values in the first 300 iterations because the algorithm is exploring new actions and its expected rewards. As we increase the Decay rate, we can see a decrease in this behavior because the model is exploiting more than exploring new actions.

In the last plot on the right-hand side, we assess the learning rate (Alpha), which is a value between 0 and 1. As we can see from the plot, low learning rate value = 0.01 means the Q values are never updated and the model is not learning. On the other hand, when the Alpha value increased, the model started to learn and converged faster.



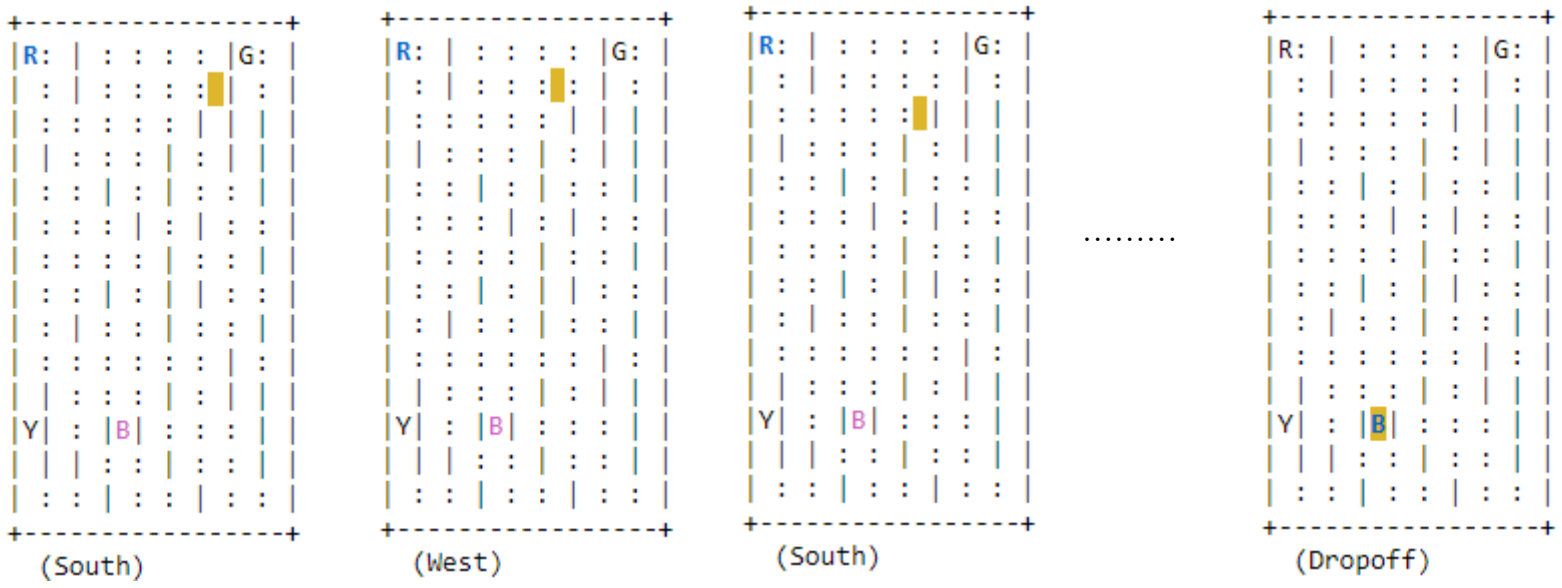


Some Interesting Observations for Q-Learning:

1. Maximum average reward is 3.8, while the average reward for VI and PI was 7.9.
2. Significant drops in the reward values in the exploration phase which make sense because Q-Learning is a model free algorithm, unlike VI and PI algorithms.
3. Q-Learning converges more slowly than VI and PI. It needed at least 220 Iterations to converge.
4. Time taken per iteration in Q-Learning is much less than the time taken per iteration in VI and PI due to the model free nature of the Q-Learning algorithm.
5. Q-Learning is considered a very good algorithm since it doesn't settle on the policy seen in VI and PI but keeps learning new policies, and that explains the lower average reward it has compared to VI and PI algorithms.

Taxi Game 2520 States

This version of the Taxi game has 2520 possible states: 126 squares, 5 locations for the passenger, and four destinations. Therefore, 126*5*4 = 2520 states. some random steps are shown below in the game, so we see how this version looks like:



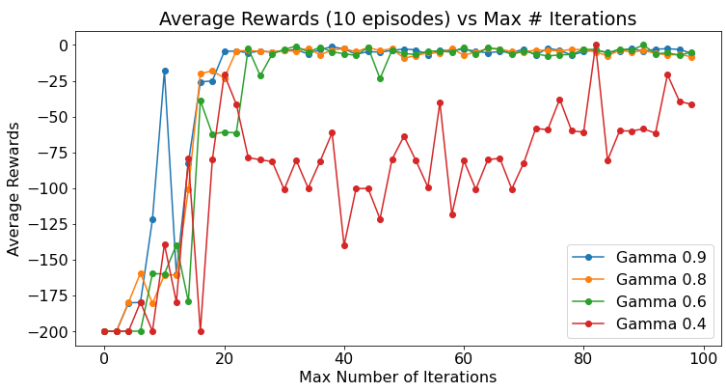
1. Policy and Value Iteration Algorithms

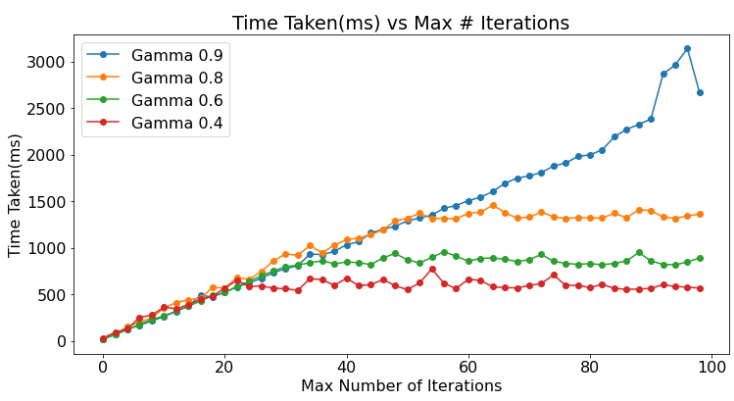
In this experiment, I am using 10 episodes and Gamma values = [0.4, 0.6, 0.8, 0.9]. As shown below, VI and PI have almost the similar behavior in the number of iterations they need to reach convergence. However, we can see that VI used fewer iterations than PI to reach convergence. If we take Gamma = 0.9, we can see that at this discount factor value VI converged at iteration = 18, while PI converged at iteration = 35. This is very interesting result because we should expect the opposite to happen, where PI uses fewer iterations to converge than the VI. More investigation on the initial state assignment is needed to track this behavior. On the other hand, we can see that the time taken per iteration for PI is almost equal to the time taken per iteration for VI, except for Gamma = 0.9 where it needs more than 100 iterations to stop updating the value. This behavior is expected due to the complexity of the PI algorithm. Furthermore, it is clear from these plots that PI algorithm is more stable than VI in terms of convergence and average rewards per all Gamma values for this number of states in the Taxi environment.

It is important to note that the average reward for all Gamma values in VI algorithm is approx. -4.1 except for Gamma = 0.4, the average reward for discount factor was -78.5. For PI algorithm, we got average reward value of -4.1 for all Gamma values.

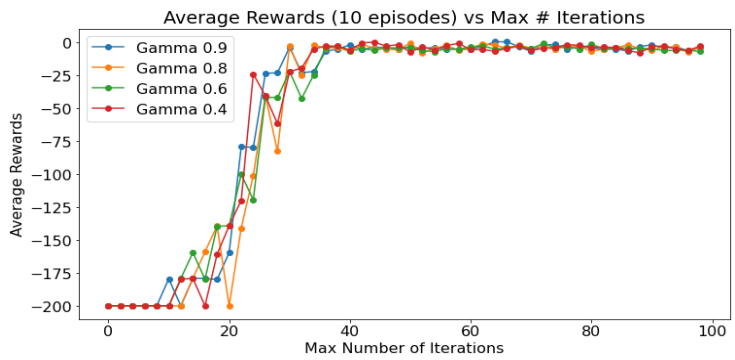
I would say there is no difference in VI and PI behaviors between the 500 states environment and the 2520 states environment.

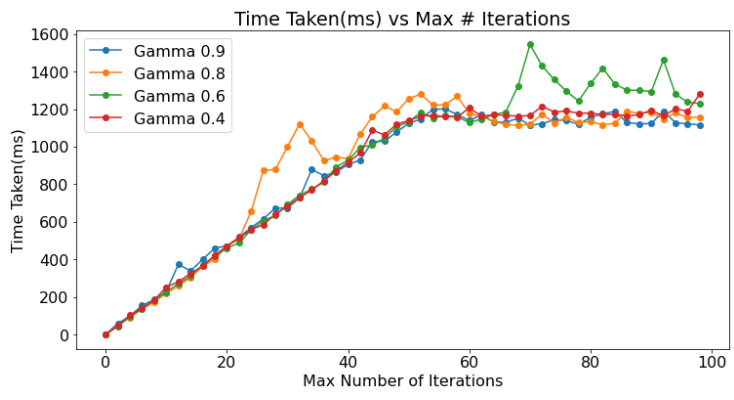
Value Iteration





Policy Iteration





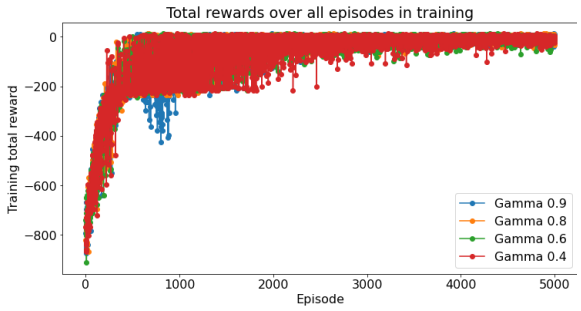
2. Q-Learning Algorithm

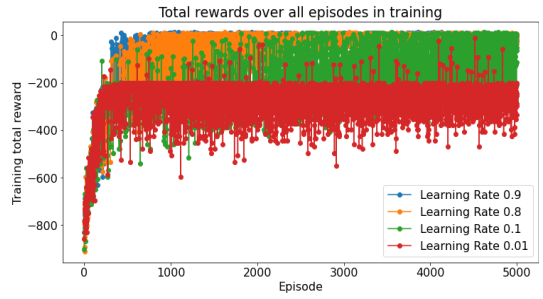
I used Gamma = [0.4, 0.6, 0.8, 0.9], Learning rate (Alpha) = [0.01, 0.1, 0.8, 0.9] and Epsilon-decay = [0.01, 0.99]. I am following the same process done in the small state environment. The Learning rate and Epsilon-decay was explained explicitly in the Q-Learning section of the 500 states Taxi environment. I will move forward to the results and analysis directly for this part.

In the first plot on the left-hand side, it is expected to see higher fluctuations in the reward values corresponds to Gamma = 0.4 because the algorithm is biased towards immediate rewards rather than the long-term rewards. By increasing the discount factor, we can see that these reward fluctuations started to decrease because we are driving the model to be biased towards the long-term rewards.

In the middle plot, we assess the Epsilon decay rate and how it affects the model. Decay rate controls the exploration and exploitation concept of the algorithm. When we chose a Decay rate = 0.01, we are giving the algorithm the opportunity to explore and learn more than being greedy since we start with Epsilon = 0.99 as a default value and that explains the fluctuations in the reward values in the first 300 iterations because the algorithm is exploring new actions and its expected rewards. Now the behavior of the model at a Decay rate = 0.99 is UNEXPECTED and needs further investigation because at this decay rate value, the fluctuation at the reward value should be decreased significantly.

In the last plot on the right-hand side, we assess the learning rate (Alpha), which is a value between 0 and 1. As we can see from the plot, low learning rate value = 0.01 means the Q values are never updated and the model is not learning. On the other hand, when the Alpha value increased, the model started to learn and converged faster.





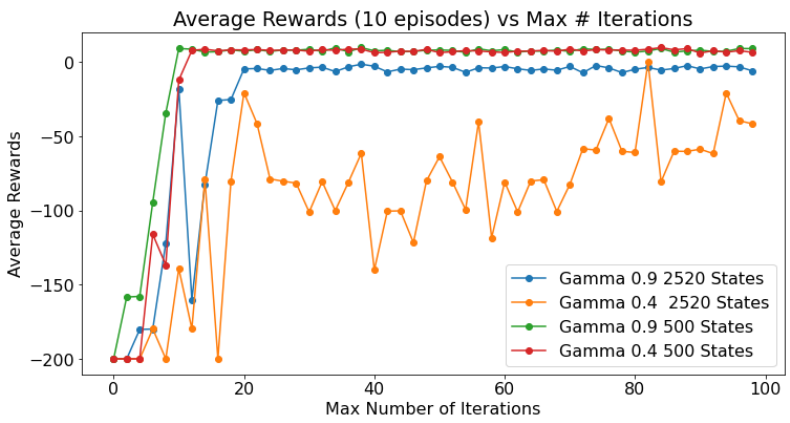
3. Taxi Game Small VS Large states

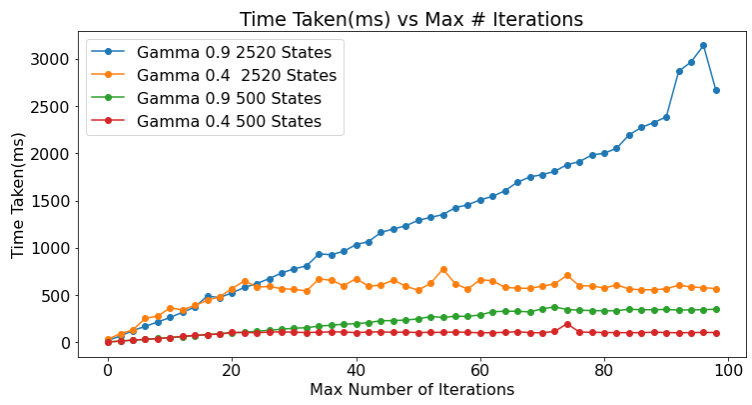
In this section, we will compare the behaviors of the VI, PI and Q-Learning algorithms on the small and large states Taxi game using the below plots for more visualization. Analysis and observations are summarized in the below table:

Number of States	Value Iteration	Policy Iteration	Q-Learning
500	• @ Gamma = 0.4; Average Rewards = 7.9 and number of iterations for convergence = 18. Time taken for 18 iterations to be completed = 50 ms. • @ Gamma = 0.9; Average Rewards = 7.9 and number of iterations for convergence = 15. Time taken for 15 iterations to be completed = 50 ms.	• @ Gamma = 0.4; Average Rewards = 7.9 and number of iterations for convergence = 16. Time taken for 16 iterations to be completed = 50 ms. • @ Gamma = 0.9; Average Rewards = 7.9 and number of iterations for convergence = 18. Time taken for 18 iterations to be completed = 50 ms.	• @ Decay rate = 0.1; number of iterations for convergence = 500 iterations. Time taken for 500 iterations = 1000 ms. • @ Learning rate = 0.8; number of iterations for convergence = 650 iterations. Time taken fir 650 iterations = 1080 ms. • @ Gamma = 0.9; number of iterations for convergence = 500 iterations. Time taken for 500 iterations = 1000 ms.
2520	• @ Gamma = 0.4; Average Rewards = -78.5 and number of iterations for convergence = 120. Time taken for 120 iterations to be completed = 600 ms. • @ Gamma = 0.9; Average Rewards = -4.1 and number of iterations for convergence = 20. Time taken for 20 iterations to be completed = 500 ms.	• @ Gamma = 0.4; Average Rewards = -4.1 and number of iterations for convergence = 40. Time taken for 40 iterations to be completed = 900 ms. • @ Gamma = 0.9; Average Rewards = -4.1 and number of iterations for convergence = 40. Time taken for 900 iterations to be completed = 500 ms.	• @ Decay rate = 0.1; number of iterations for convergence = 3000 iterations. Time taken for 3000 iterations = 7000 ms. • @ Learning rate = 0.8; number of iterations for convergence = 3500 iterations. Time taken fir 3500 iterations = 7500 ms.

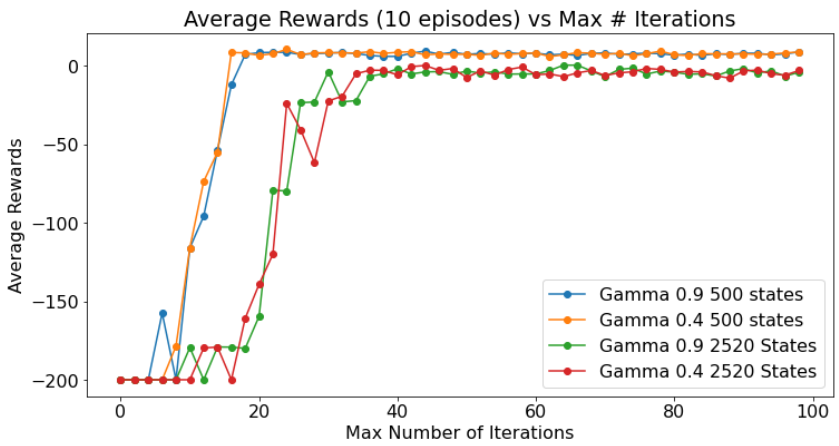
			• @ Gamma = 0.9; number of iterations for convergence = 2500 iterations. Time taken for 2500 iterations = 6500 ms.

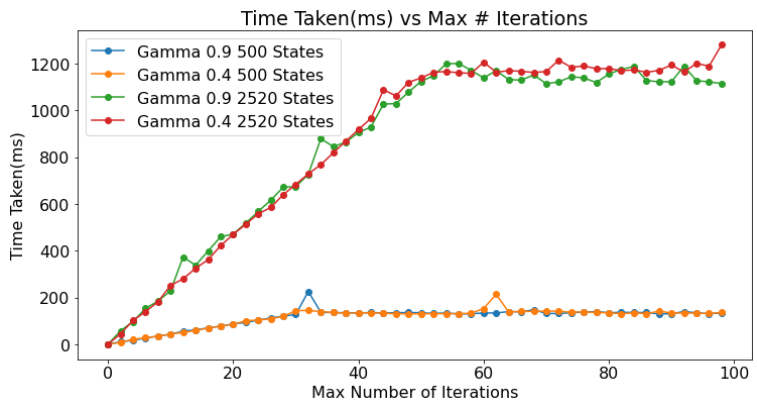
Value Iteration



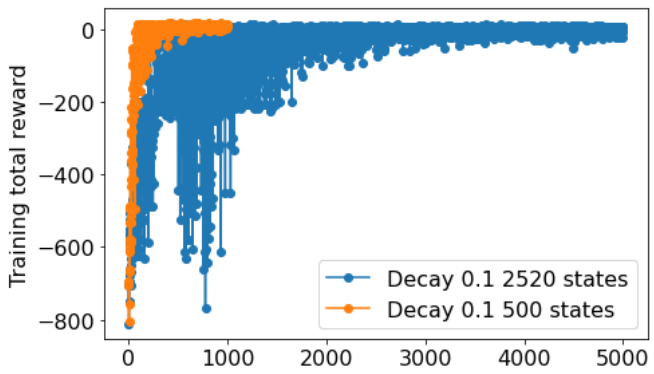


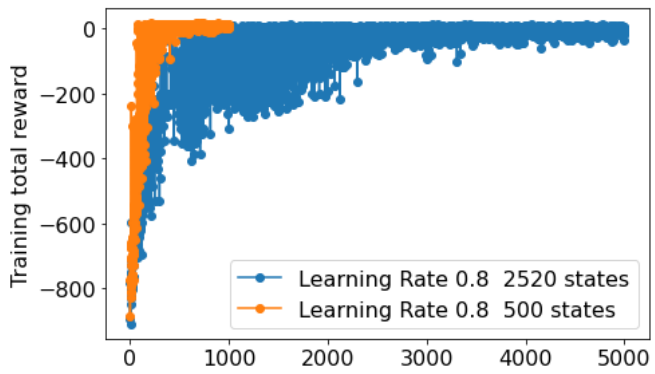
Policy Iteration

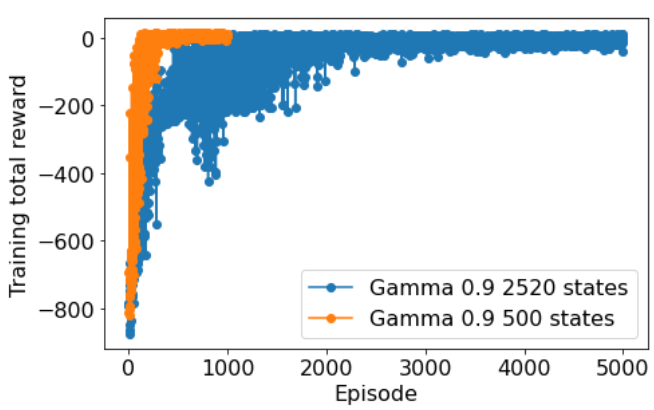


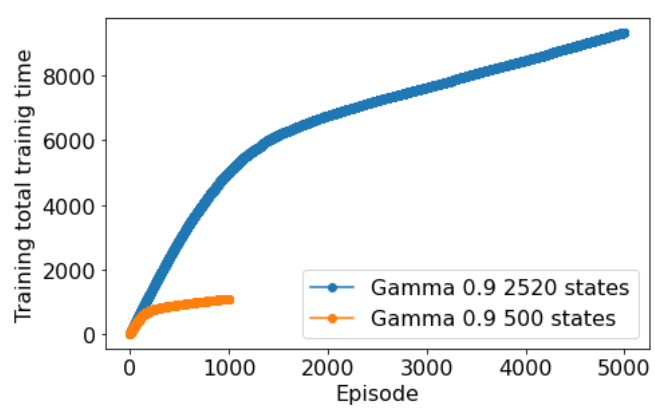


Q-Learning









Forest Management Problem

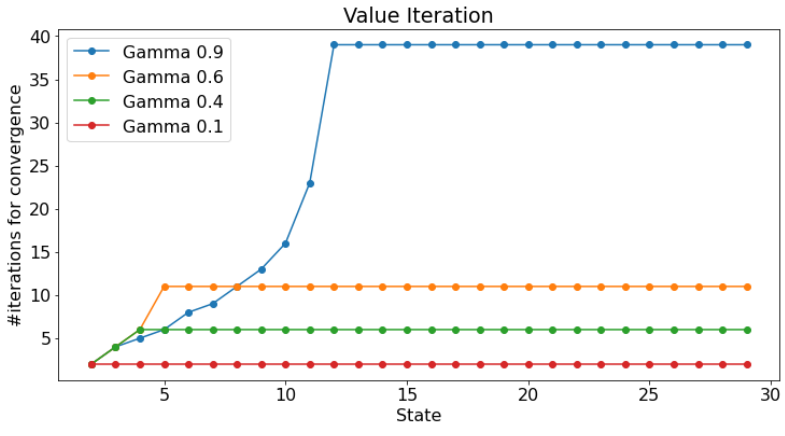
A forest is managed by two actions: ‘Wait’ and ‘Cut’. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. We are using the default rewards in this experiment; 'Wait' = 4 and 'Cut' = 2 Starting by 5 states and up to 30 states are modelled in this experiment. Analysis and results can be seen in the next sections. Examples of optimum policy can be seen in the below code snippets, where 's' is the number of states and 'p' is the probability that wildfire will happen.

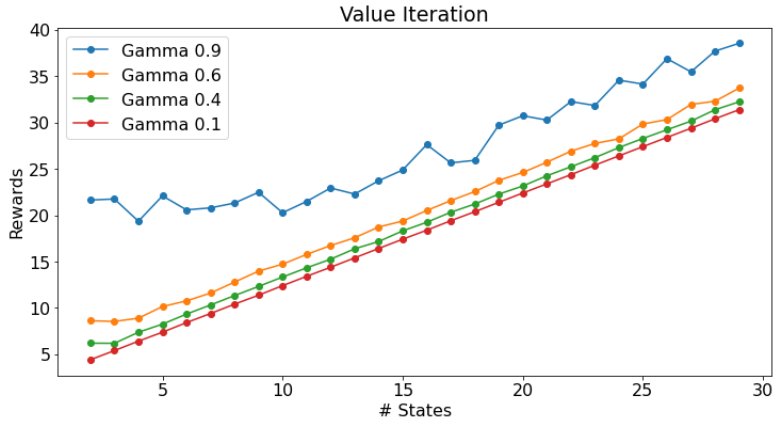
```
In [81]: P, R = mdptoolbox.example.forest(S=5,p=0.3)
         pi = mdptoolbox.mdp.PolicyIteration(P, R, 0.95)
         pi.run()
         pi.policy
Out[81]: (0, 0, 0, 0, 0)

In [82]: P, R = mdptoolbox.example.forest(S=5,p=0.5)
         pi = mdptoolbox.mdp.PolicyIteration(P, R, 0.95)
         pi.run()
         pi.policy
Out[82]: (0, 1, 0, 0, 0)

In [83]: P, R = mdptoolbox.example.forest(S=5,p=0.7)
         pi = mdptoolbox.mdp.PolicyIteration(P, R, 0.95)
         pi.run()
         pi.policy
Out[83]: (0, 1, 1, 0, 0)

In [84]: P, R = mdptoolbox.example.forest(S=5,p=0.9)
         pi = mdptoolbox.mdp.PolicyIteration(P, R, 0.95)
         pi.run()
         pi.policy
Out[84]: (0, 1, 1, 1, 0)
```

1. Value and Policy Iteration Algorithms

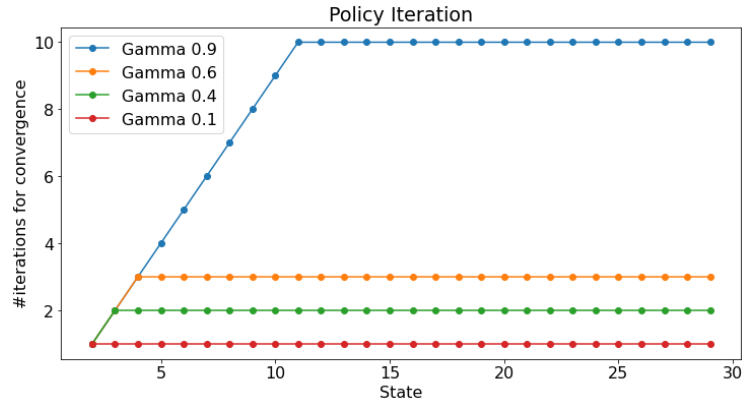
The total rewards and the number of iterations for convergence per number of states is examined in this part of the assignment. From the below plots, when we increase the Gamma value (discount factor), the total reward is maximized which is the main goal for the agent to maximize the total rewards. Also, it is expected that when the number of states increase, the number of iterations needed for convergence also increase. It is expected that VI needs more iterations than PI to converge.

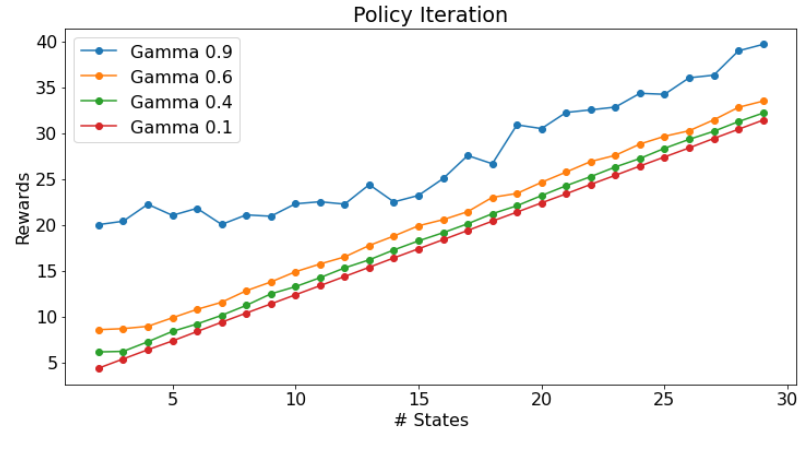
Value Iteration





Policy Iteration



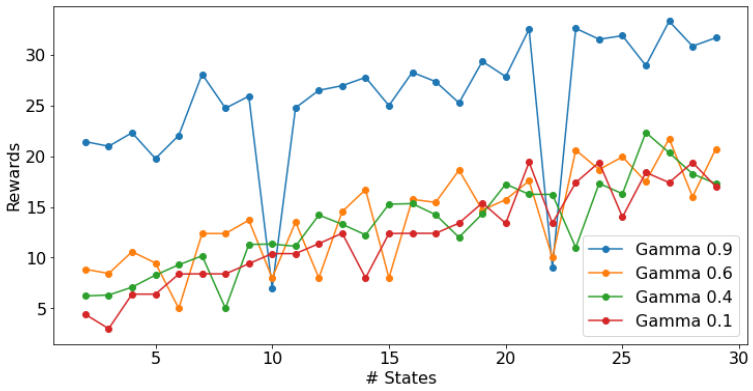


2. Q-Learning Algorithm

In this part, the average rewards per number of states(Up to 30 states) is examined with Gamma values = [0.1, 0.4, 0.6, 0.9] which showed maximum reward = 38 for 27 states. Also, the total rewards in 10,000 iterations are examined for 10 states using Epsilon values = [0.05, 0.15, 0.25, 0.5, 0.75, 0.95] and the convergence time for each value of Epsilon is reported too.

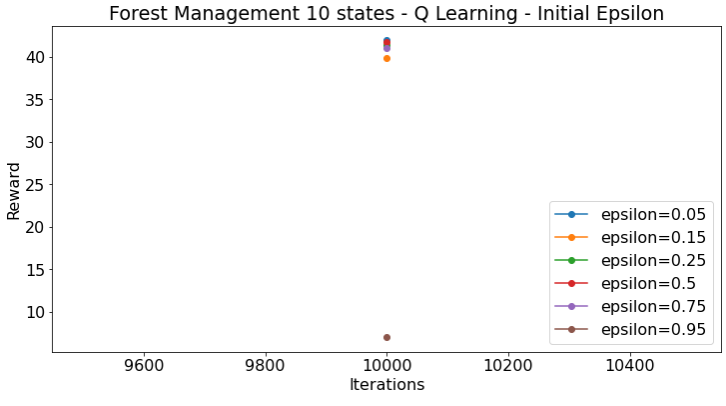
For 20 states, the average rewards in 10,000,000 iterations are examined for 20 states using Epsilon values = [0.05, 0.15, 0.25, 0.5, 0.75, 0.95] and the convergence time for each value of Epsilon is reported too.

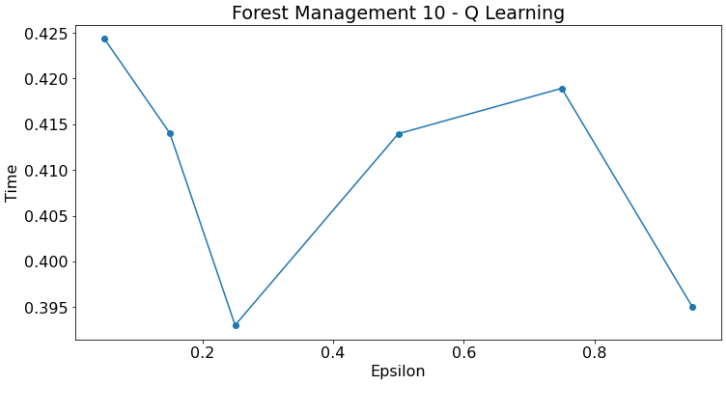
The plots and results can be viewed in the below table:



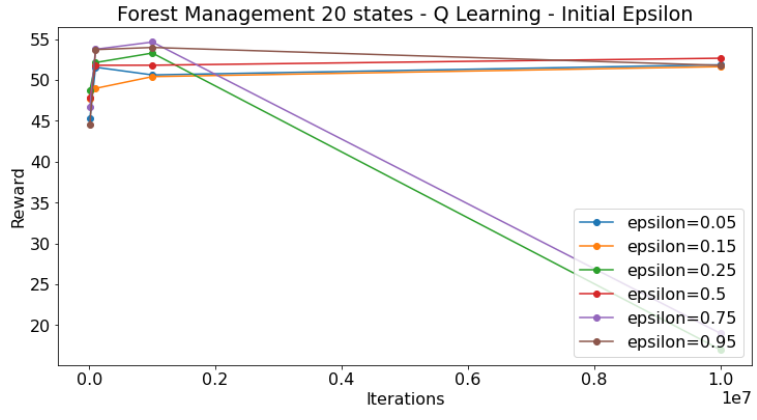
Epsilon Value	10 States – 10,000 Iterations	20 States – 10,000,000 Iterations
0.05	• Reward = 41.9 • Convergence time = 425 ms	• Reward = 51.8 • Convergence time = 412000 ms
0.15	• Reward = 39.9 • Convergence time = 415 ms	• Reward = 51.6 • Convergence time = 383000 ms
0.25	• Reward = 41.4 • Convergence time = 392 ms	• Reward = 17 • Convergence time = 372000 ms
0.5	• Reward = 41.7 • Convergence time = 415 ms	• Reward = 52.6 • Convergence time = 375000 ms
0.75	• Reward = 41 • Convergence time = 420 ms	• Reward = 19 • Convergence time = 380000 ms
0.95	• Reward = 7 • Convergence time = 395 ms	• Reward = 51.8 • Convergence time = 385000 ms

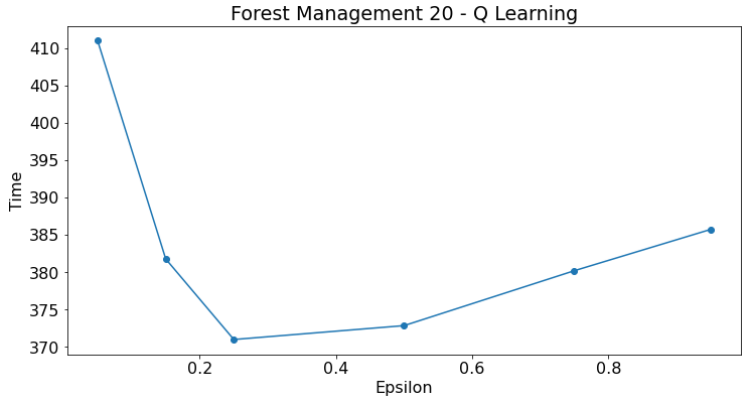
10 States





20 States





Conclusion

In this assignment, three Reinforcement Learning algorithms (Value Iteration, Policy Iteration and Q-Learning) were investigated on two Markov Decision Process problems, the first one was the Taxi game (Grid world problem) and the second was the Forest Management problem (non-Grid world problem).

We learned about discount factor (Gamma) in Value and Policy Iteration algorithms and how it affects the average/total reward values that the agent trying to maximize. For example, in the Taxi game and Forest management problem, we have seen that Gamma = 0.9 increased the average reward values, but on the other hand in the VI algorithm, it needed more iterations to converge.

The results obtained in this assignment suggests that PI provides more stable results than VI algorithm and Q-Learning algorithm is fast because it is a model free algorithm, but on the other hand, Q-Learning algorithm needs too many iterations to learn.

It was very important to discuss the Q-Learning hyperparameters, such as the Learning rate and the Epsilon Decay factor. The results show that increasing the Learning rate hyperparameters gives the agent boost in learning much quicker. Epsilon Decay factor is very significant in the exploration and exploitation perspective. The Q-Learning algorithm needs to explore more at the beginning to learn the environment and how the agent's actions affect the received rewards. The results show Decay rate = 0.1 is sufficient for the agent to explore and learn and as it progresses, it will start exploiting.

References

1. Reinforcement Learning: An introduction, Richard S. Sutton and Andrew G. Barto
2. Mastering Reinforcement Learning with Python by Enes Bilgin
3. https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym
4. Markov Decision Process (MDP) Toolbox: example module — Python Markov Decision Process Toolbox 4.0-b4 documentation (pymdptoolbox.readthedocs.io)