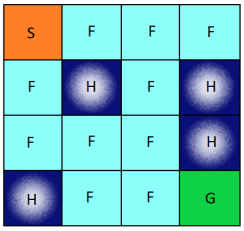
CS7641 Fall 2022 HW4 Markov Decision Processes

Fung Yi Yuen
fyuen3@gatech.edu

***Abstract*— Two Markov Decision Processes (MDPs), Frozen Lake and Forest Management were chosen as the problems to be solved by 3 RL algorithms (value iteration, policy iteration and Q-learning). Two different sizes (one large and one small) of state were solved for each MDP to compare the behavior of the three RL algorithms.**

I. MARKOV DECISION PROCESS (MDP)

1.1. Frozen Lake



Frozen lake is a grid-world problem where the agent starts at the top left-hand corner (the orange box) and its goal is to reach the bottom right-hand corner (the green box). The episode ends when the agent reaches a goal or falls into holes (dark blue boxes). A reward of 1 will be received if the agent reaches the goal otherwise 0 reward if it falls into a hole.

The agent can move LEFT, DOWN, RIGHT, UP (represented by 0, 1, 2, 3 in code respectively). For every action there will be 0.3333 of chances the action is correctly executed, a 0.3333 chance heading to the left of the direction chosen, and a 0.3333 chance heading to the right of the direction chosen. Thus making the problem stochastic.

1.2. Forest Management

This is a non-grid world problem. A forest that can grow trees for wildlife and provide woods for making money. An agent can choose to wait for the forest to grow old or to cut the trees which sets the forest's state back to 0. There is a probability p that a fire burns the forest which also sets the state back to 0.

If the agent chooses to wait at the oldest state, it will have *1-p* chances to stay in the oldest state and *p* chances to go back to state 0. If the agent chooses to cut at the older state, it will go back to state 0.

The transition matrix is written as below:

(0=wait, 1=cut, p=chances of fire occurrence)

```
            | p 1-p 0.......0  |
            | .  0 1-p 0....0  |
P[0,:,:] =  | .  .  0  .       |
            | .  .         .   |
            | .  .          1-p|
            | p  0  0....0 1-p |

            | 1 0..........0 |
            | . .          . |
P[1,:,:] =  | . .          . |
            | . .          . |
            | . .          . |
            | 1 0..........0 |
```

The agent will get the largest rewards (r1 or r2) if it chooses to wait or cut the forest at the oldest state. Or it gets reward 1 for cutting at any non-initial state.

Choose "wait" at the oldest state get reward r1:

```
            |  0  |
            |  .  |
R[:,0] =    |  .  |
            |  .  |
            |  0  |
            |  r1 |
```

Choose "cut" at the oldest state get reward r2:

```
            |  0  |
            |  1  |
R[:,1] =    |  .  |
            |  .  |
            |  1  |
            |  r2 |
```

1.3. Why are the two MDPs interesting?

These two MDP problems are interesting because they possess different characteristics which could be a good comparison when observing the behavior of the 3 iterative algorithms (value iteration, policy iteration and Q-learning). Frozen Lake is a grid problem while Forest Management is a non-grid problem. Their reward calculation and convergence behavior would be different.

Moreover, Frozen Lake is a system where all the components are known and can be modeled through iterations. But Forest Management is a spatially linked system in which the dynamic is partially unknown. These two MDPs serve as a comparison to each other when applying the 3 iterative algorithms.

In this assignment, a small and a large number of states were chosen for each MDP problem to observe the convergence behavior of the 3 algorithms. Thus not only are the problems themselves different but also the number of states in each MDP problem.

II. REINFORCEMENT LEARNING (RL) ALGORITHMS

2.1. Value Iteration (VI)

Value iteration is a way to find optimal policy through the optimal value function. It uses the Bellman Equation as an update function to iteratively calculate the state values. Since the utility is not known at the first place, VI starts with an arbitrary value and the reward of the end point. Then it computes the utility of its closest states and backward propagates all the utility of the whole system from that end reward point. This process is repeatedly run until the state values fall below a certain threshold and the algorithm is said to be converged.

2.2. Policy Iteration (PI)

Policy iteration is a process of evaluating a given policy by

calculating the value function of that given policy. At first, an arbitrary policy was chosen as initialization. If a policy π is improved using value V' to yield a better policy π', the new policy π' is used to compute a better value V' which yields another better policy π''. The new policy π'' is then used to compute a better value V''. These policy evaluation and policy refinement steps run repeatedly until no more improvement in policy. Then the algorithm is said to be converged.

2.3. Q-learning

Q-learning is a model-free RL algorithm in which the agent explores the environment according to what is given at its current state. The Bellman equation is used in this algorithm to tell the maximum future rewards received if an agent enters the current state s and is going to enter the next state s'. This is called the Q-value Q(a, s), which is used to determine how good an action is taken at state s. A learning rate α is used to control how much difference between the previous and the new Q value should the agent consider when exploring the environment. Since the q-learner needs exploration to compute Q values in a table, it is expected the algorithm will take much more time than the previous two algorithms.

III. ANALYSIS

3.1. Frozen Lake (size 4x4)

3.1.1. Value Iteration

In value iteration, the true value of each state in the frozen lake grids is calculated by backward propagating the utility value of each state from the goal to its closest neighbors and then to the rest of the grids. The algorithm converges when the delta in utility falls under a threshold, epsilon.

Different combinations of gamma and epsilon were tested to find the best reward an agent can achieve. The table below shows **high gamma value and low epsilon value** give the greatest reward in the 4x4 grid world MDP.

gamma	epsilon	time	iterations	reward
0.1	1.00E-05	0	4	0.345235
0.1	1.00E-10	0	9	0.345239
0.1	1.00E-15	0	14	0.345239
0.1	1.00E-20	0	19	0.345239
0.3	1.00E-05	0	8	0.375101
0.3	1.00E-10	0	17	0.375103
0.3	1.00E-15	0	26	0.375103
0.3	1.00E-20	0	36	0.375103
0.6	1.00E-05	0	17	0.447647
0.6	1.00E-10	0	37	0.447649
0.6	1.00E-15	0	58	0.447649
0.6	1.00E-20	0	73	0.447649
0.9	1.00E-05	0.01	77	0.639019
0.9	1.00E-10	0.01	162	0.63902
0.9	1.00E-15	0.01	246	0.63902
0.9	1.00E-20	0.02	270	0.63902

Different gamma values with a fixed epsilon(=1e-20) were experimented to see the effect on the reward function:
gamma= [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.999]
epsilons = [1e-20]

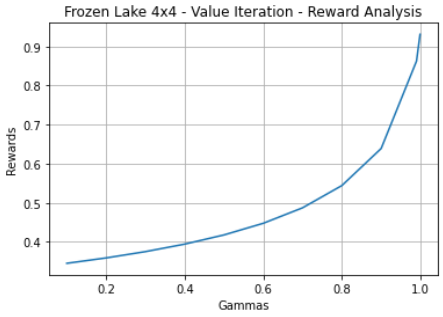


fig.1 Reward vs Gammas (Frozen Lake 4x4)

Fig.1 shows the reward is the highest as the gamma approaches 1. Larger gamma values means more weight is put on long-term gains while smaller gamma values means more weight is put on short-term gains. As the gamma is closer to 1, the optimal policy would be the one that runs on infinite time.

Since the maximum number of iterations is set as 100000 for this 4x4 grid world, it looks to the agent that there are infinite steps for it to achieve the greatest rewards. Thus it makes sense that the larger the gamma, the higher the rewards in fig.1.

Different epsilon values with a fixed gamma were also experimented to see the effect on the reward function:
gammas = [0.9]
epsilons = [1e-20, 1e-15, 1e-10, 1e-5]

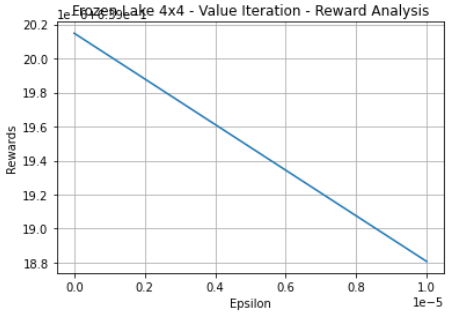


fig.2 Rewards vs epsilon (Frozen Lake 4x4)

Fig.2 shows the reward is the highest as the epsilon approaches 0. It is because epsilon is the threshold to define the convergence criteria. Once the delta (aka the difference) between the current true value and the previous true value falls below a threshold (epsilon), the algorithm has converged and the reward is the greatest. Thus when the epsilon is set closer to zero, it pushes the threshold boundary towards zero which in turn gives higher rewards. However, decreasing epsilon from 1e-5 to 1e-20 only improves rewards by 1e-5 which is insignificant. It means a small epsilon e.g. 1e-15 is already enough to get a good policy.

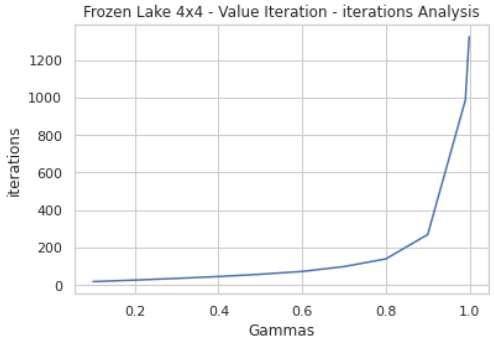


fig.3 Iteration vs Gammas (Frozen Lake 4x4)

Fig.3 shows as gamma becomes higher, more iterations are

needed for convergence. Notice there is a sharp increase in the number of iterations after gamma=0.9. If gamma is 1, 1325 iterations are needed for convergence. But if gamma is 0.9, only 270 iterations are enough for convergence. It means that when future rewards are treated as the same weight as the current reward, the agent can use unlimited steps to solve the grid problem and iterations can become infinite. It can gradually achieve the greatest reward over infinite iterations. Thus the reward increases with the number of iterations (fig.4).

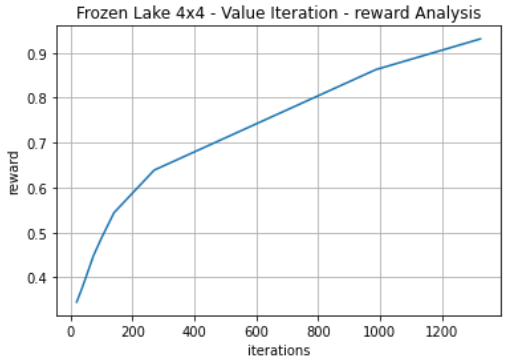


fig.4 Reward vs iteration (Frozen Lake 4x4)

However, in a practical world time is limited to solve the MDP problem (i.e. number of steps is limited). The agent should consider short-term rewards more than the long-term rewards. Thus gamma will not be perfectly 1. Gamma is the discount factor to tell the agent to finish the grid problem within a finite number of steps. Thus gamma=0.9 would be a better choice here since it only uses 270 iterations to converge and the reward is still high.

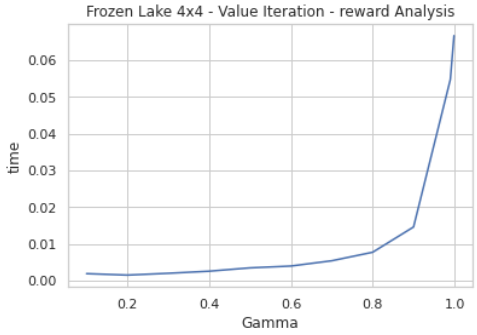
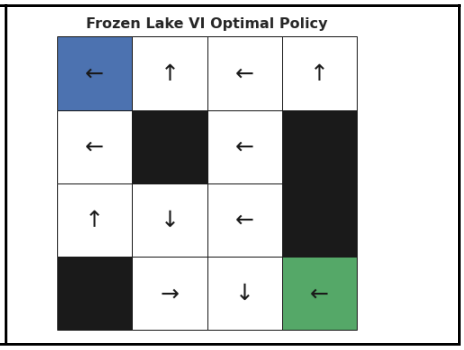


fig.5 Time vs gamma (Frozen Lake 4x4)

Fig.5 shows the time for convergence for different values of gammas. Although all of them use less than 0.1 second to converge, the time for convergence is significantly higher when gamma=1 was used.

The optimal policy for VI is shown as below:

Success = 77.50
Gamma = 0.90
Epsilon= 1E-15



3.1.2. Policy Iteration

In policy iteration, the utility of the current policy is evaluated iteratively to find improvement until all values fall below a threshold (epsilon). The epsilon is set to be 1e-5 which is the convergence criteria of this algorithm.

gamma	epsilon	time	iter	reward
0.1	1e-5	0.003	4	0.345239
0.2	1e-5	0.001	4	0.358992
0.3	1e-5	0.001	4	0.375103
0.4	1e-5	0.001	4	0.394332
0.5	1e-5	0.002	4	0.417861
0.6	1e-5	0.002	4	0.447649
0.7	1e-5	0.001	4	0.487267
0.8	1e-5	0.002	5	0.544196
0.9	1e-5	0.002	6	0.63902

Similar to value iteration, in policy iteration the higher the gamma value the higher the reward (fig. 6)

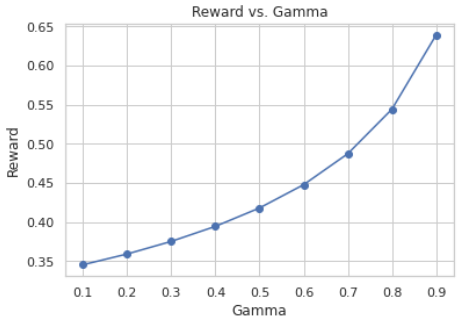


fig.6 Reward vs iteration (Frozen Lake 4x4)

Different from value iteration, policy iteration uses a few number of iterations to converge. All the gamma values tested here use 4 to 6 iterations to reach convergence but in value iteration at least hundreds of iterations are needed for larger gamma with high reward value.

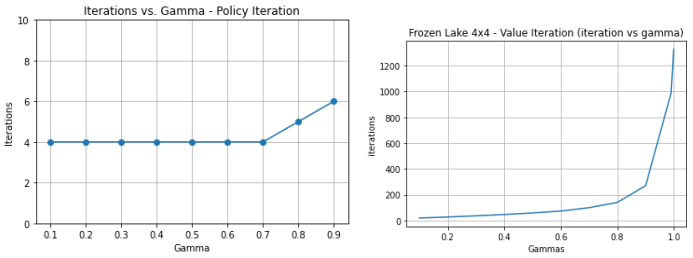


fig.7 Iteration vs gamma (Frozen Lake 4x4)

Similar to VI, PI's optimal policy also uses 30 to 40 steps to solve the 4x4 grid problems (fig.8).

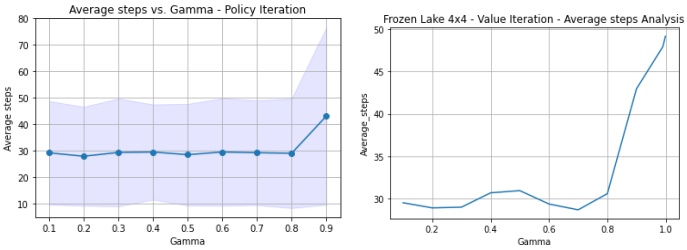


fig.8 Average steps vs iteration (Frozen Lake 4x4)

But the time spent by PI is much faster than VI in general.

The max time for PI to converge only takes 0.006s (fig.9 left graph) while the max time for VI to converge takes 0.25s (fig.9 right graph).

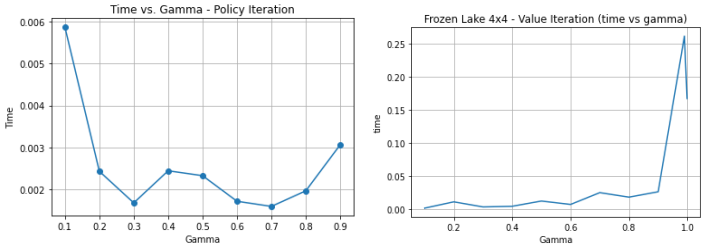


fig.9 Time vs gamma (Frozen Lake 4x4)

Comparing the success rate of VI and PI (fig.10), both have similar success rates for different gammas. For gamma=0.9, PI can reach a success rate of 78.1% while VI can reach 77.5%. PI has a slightly better chance to reach the goal. But when factoring in the time spent and the number of iterations used, PI generally performs better than VI since it converges faster without sacrificing the success rate.

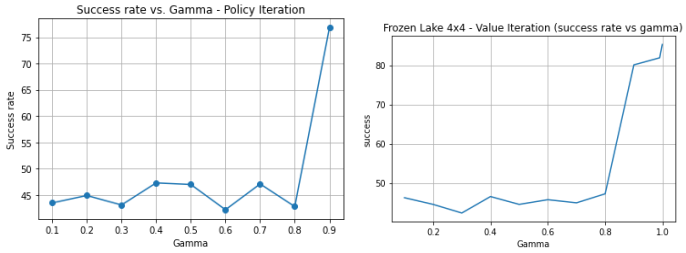
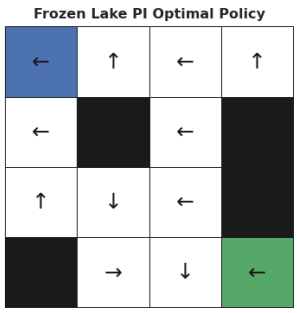


fig.10 Success rate vs iteration (Frozen Lake 4x4)

The optimal policy for PI is shown as below:

Success = 78.10
Gamma = 0.90
Epsilon= 1E-5



PI's optimal policy is the same as VI's one.

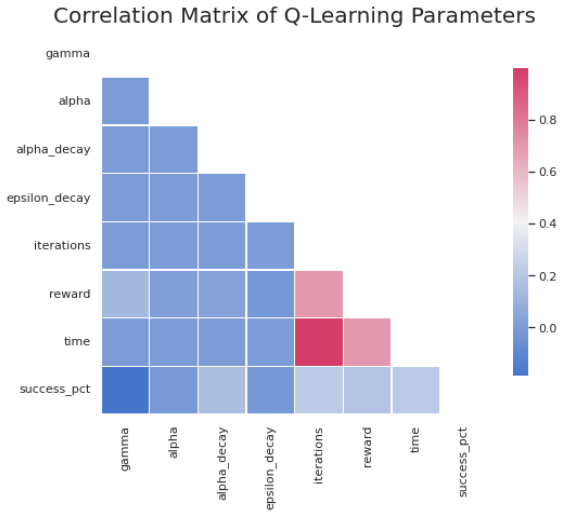
3.1.3. Q-learning

Alpha and epsilon are the two parameters that the Q-learner uses to explore and exploit the environment. Alpha is the learning rate which determines how much weight the learner should assign to the recent state when updating the q table. Epsilon is the probability to explore the environment in an epsilon-greedy policy. The agent first explores different paths by taking random actions to construct a Q-table which stores the Q(s,a). As it learns better, it converges to a stable Q-values and then exploits the best path as the epsilon decays. The q learning algorithm is set to run for a set amount of iterations to reach the optimal policy.

Different hyperparameters e.g. gamma, alpha, alpha decays, epsilon decays and number of iterations were tested to see their effect on the final reward and the time spent. A correlation matrix is plotted to see which hyperparameters affect the final reward the most.

gammas	[0.8, 0.9, 0.99]
alphas	[0.01, 0.1, 0.2]
alpha decays	[0.9, 0.999]
epsilon decays	[0.9, 0.999]
iterations	[1e5, 1e6, 1e7]

Observed from the correlation matrix below, gamma, alpha decay and the number of iterations had greater impact on getting higher reward from an optimal policy.



But the time spent in QL is much longer than VI and PI. It took 4 hours to run the algorithm but VI and PI only take a few seconds at maximum to find the optimal policy. The time spent increases significantly for larger iterations (fig.11).

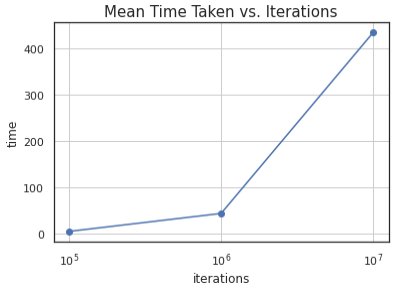


fig.11 Time vs iteration (Frozen Lake 4x4)

Interestingly, more iterations does not guarantee higher reward (fig.12). The reward drops at 10e6 and then rises at 10e7. It could be due to the exploration-exploitation effect in which the agent takes random actions and this causes the reward to fluctuate instead of keep rising.

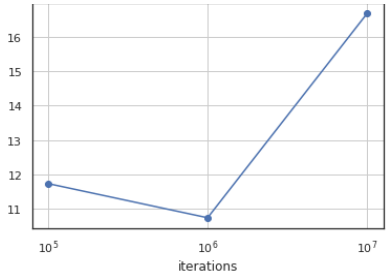


fig.12 Reward vs iteration (Frozen Lake 4x4)

Fig.13 shows the higher the alpha decay, the higher the reward. But the optimal alpha decay value should work in

conjunction with other hyperparameters such as alpha which can affect how fast the QL algorithm converges.

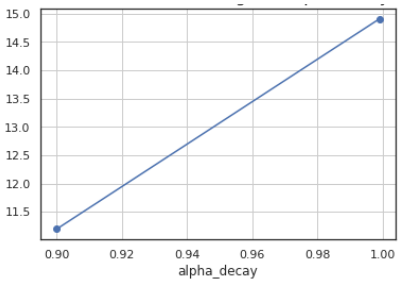


fig.13 Reward vs alpha decay (Frozen Lake 4x4)

Similar to PI and VI, increasing gamma helps improve the success rate (fig.14) but the improvement in QL is not as significant as the ones in PI and VI. It could be due to the stochastic nature of the QL algorithm which makes the success rate not always increase after each episode.

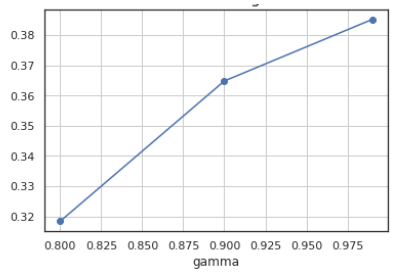
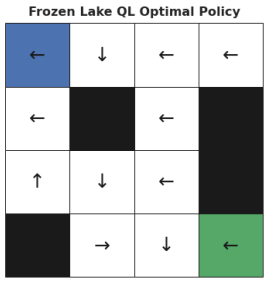


fig.14 Success rate vs gamma (Frozen Lake 4x4)

The optimal policy for QL is shown as below:

Success = 76.90
Gamma = 0.80,
Alpha = 0.01,
Alpha Decay: 0.900,
Epsilon Decay: 0.900,
Iterations: 1.0E+07



3.2. Frozen Lake (size 16x16)

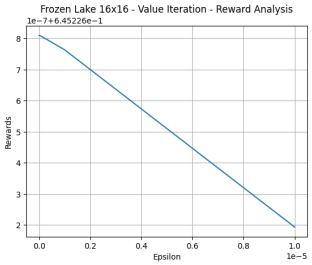
A larger Frozen Lake 16x16 is tested. Since a larger grid world means a longer path to the goal, the reward on average is expected to be lower than the 4x4 Frozen Lake while the number of iterations would be much larger.

3.2.1. Value Iteration

Similar to the previous 4x4 Frozen Lake, different combinations of hyperparameters were tested to find the best reward VI can achieve from optimal policy. Turns out higher gamma and smaller epsilon yields the greatest reward from optimal policy, which is the same as 4x4 grid world.

Gamma	Eps	Time	Iter	Reward
0.1	1.00E-05	0	4	0.345235
0.1	1.00E-10	0.01	9	0.345239
0.1	1.00E-15	0.01	14	0.345239
0.1	1.00E-20	0.02	19	0.345239
0.3	1.00E-05	0	8	0.375102
0.3	1.00E-10	0	17	0.375103
0.3	1.00E-15	0.01	26	0.375103
0.3	1.00E-20	0	36	0.375103
0.6	1.00E-05	0.01	18	0.44773
0.6	1.00E-10	0.02	39	0.447732
0.6	1.00E-15	0.01	60	0.447732
0.6	1.00E-20	0.03	83	0.447732
0.9	1.00E-05	0.09	81	0.645226
0.9	1.00E-10	0.11	172	0.645227
0.9	1.00E-15	0.11	261	0.645227
0.9	1.00E-20	0.14	348	0.645227

The left plot shows a smaller epsilon can improve reward slightly but increase the number of iterations drastically (the right plot). As observed from the previous small grid-world, more iterations means more time to converge. A small epsilon is good enough to find an optimal policy. Epsilon=1e-20 is used here.

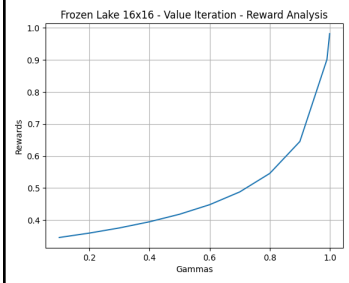


Similar to 4x4 grid-world, in 16x16 grid-world the gamma values increase with rewards. But the number of iterations in 16x16 is much larger than the 4x4 grid-world, which is expected. The number of iterations increases from 270 to 1214 when the grid size changes from 4x4 to 16x16.

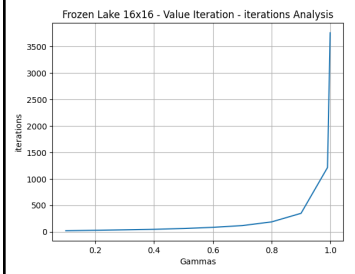
The time spent for convergence takes longer than 4x4. In 4x4, the time for convergence is only 0.03 on average. In 16x16, the time spent is 0.3 on average.

The success rate in 16x16 is lower than 4x4 due to the complexity of a larger grid world. 4x4 grid-world has a success rate of 77.5% from its best policy but 16x16 only has a 10.7% success rate.

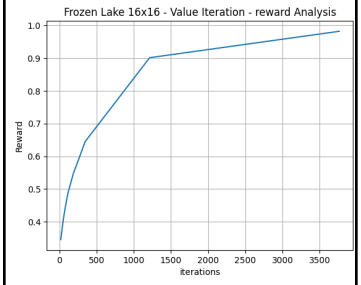
Reward vs Gamma



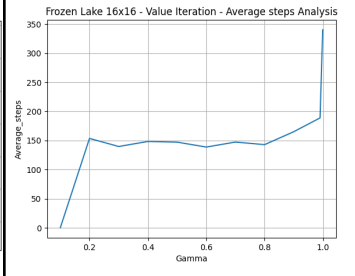
Iterations vs Gamma



Reward vs iterations

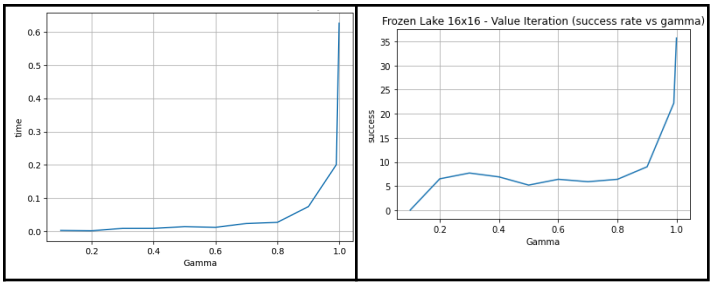


Average step vs Gamma



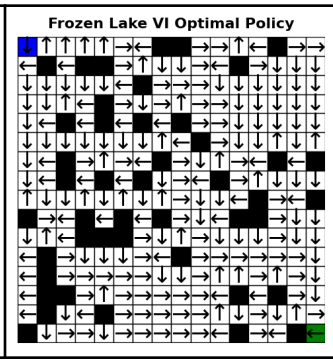
Time vs Gamma

Success rate vs Gamma



The optimal policy for VI is shown as below:

Success = 10.70
Gamma = 0.90
Epsilon= 1E-05



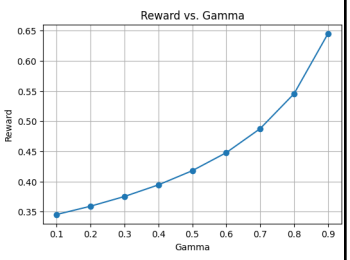
3.2.2. Policy Iteration

gamma	epsilon	time	iter	reward
0.1	1e-5	0.23	17	0.345239
0.2	1e-5	0.1	17	0.358992
0.3	1e-5	0.06	17	0.375103
0.4	1e-5	0.1	17	0.394336
0.5	1e-5	0.05	17	0.41788
0.6	1e-5	0.04	17	0.447732
0.7	1e-5	0.11	17	0.487595
0.8	1e-5	359.88	100000	0.545531
0.9	1e-5	341.45	100000	0.645227

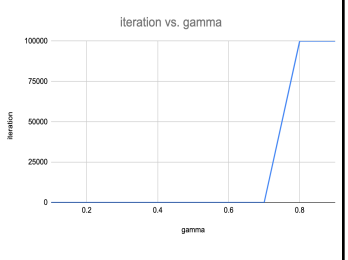
Compared to the 4x4 grid world PI, the 16x16 PI takes more iterations, spends more time to converge and takes more steps to reach the goal. This makes sense because 16x16 is larger and more complex than a 4x4 grid world thus the more iterations and time is needed for convergence. However, the success rate of 16x16 is much lower than 4x4 in general. In 4x4 PI, the success rate is 78.1%. In 16x16 PI, the success rate is only 9.6. This is expected since a larger grid world is a more complex MDP thus it is more difficult to reach the goal.

Compared to the 16x16 VI, PI generally takes much less iterations to converge for gamma<0.8. VI at least takes hundreds of iterations to converge but PI only takes 17 iterations for gamma<0.8. For gamma=0.8 and 0.9, PI takes 100000 iterations to converge which is exceptional. The max number of iterations in the PI algorithm is set to be 100000 and PI takes this amount of iterations to converge. It means when gamma>=0.8, the agent takes infinity steps to find the optimal policy.

Reward vs Gamma

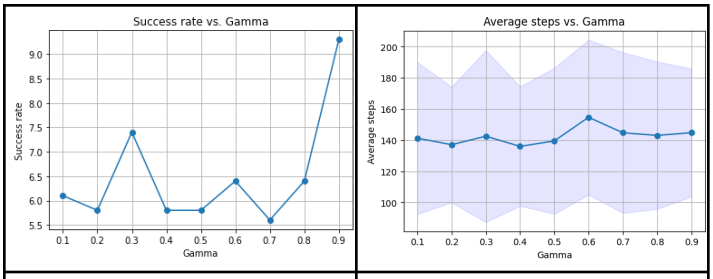


Iteration vs Gamma

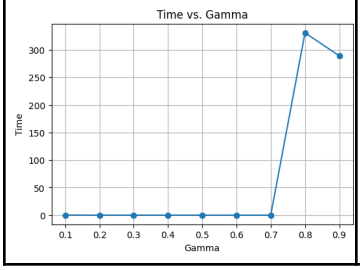


Success rate vs Gamma

Average step vs Gamma

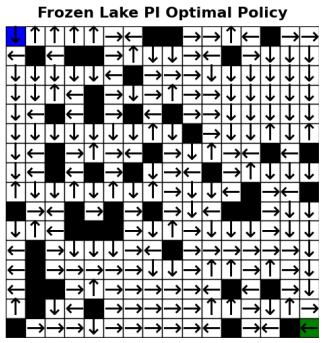


Time vs Gamma



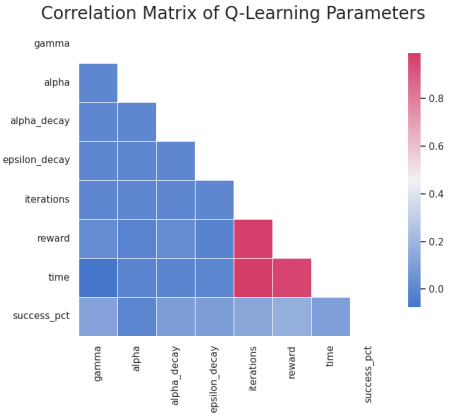
The optimal policy for PI is shown as below:
It is slightly different from the 16x16 VI.

Success = 9.60
Gamma = 0.90
Epsilon = 1E-05



3.2.3. Q-learning

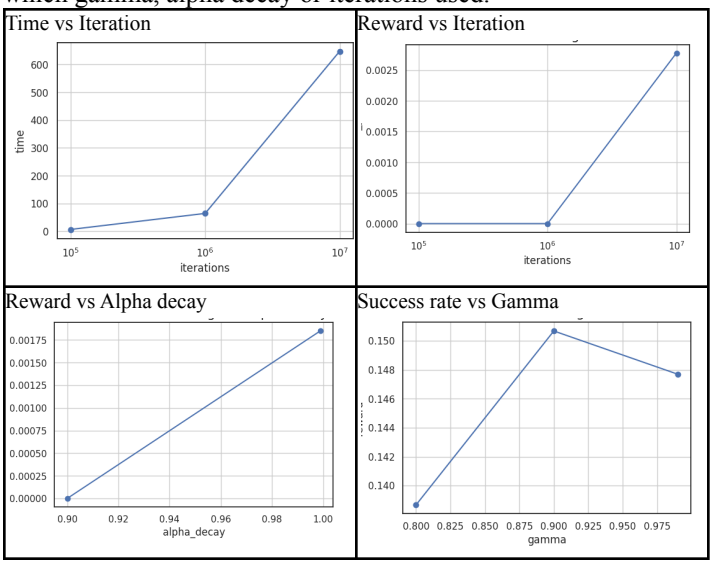
The challenge for QL in this 16x16 grid-world is that 20% of the grids are holes. The chances for an agent to end the game by falling into the holes are very high.



The correlation matrix shows gamma has a relatively larger effect on the reward compared to other hyperparameters. Observed from the plots below, however, the reward and success rate are nearly zero, which are much lower than those in a 4x4 grid world.

In the 4x4 QL algorithm, the average reward is around 13 and the best policy yields a success rate of 76.9. In this 16x16 grid world, the success rate drops to 0.1 which means the agent can hardly reach the goal. The reward is around 0.001 no matter

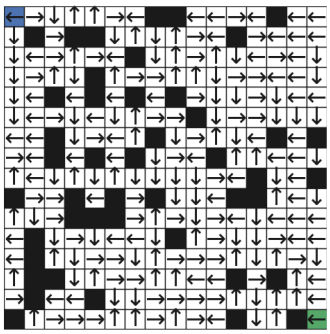
which gamma, alpha decay or iterations used.



QL gives the lowest success rate among the 3 RL algorithms in a 16x16 grid world. It means the agent cannot reach the goal by randomly exploring the environment. One reason for that would be this large grid world has no reward on the non-ending grids which causes the q-table full of zeros. A solution for that would be changing the reward of those non-ending grids to +0.1 (or any positive value<1). This can keep the agent alive and stay away from the holes. A better q-table would be constructed as well.

The optimal policy for QL is shown as below:

Success = 0.10
Gamma = 0.99,
Alpha = 0.10,
Alpha Decay: 0.999,
Epsilon Decay: 0.999,
Iterations: 1.0E+07



3.3. Forest Management (state=25)

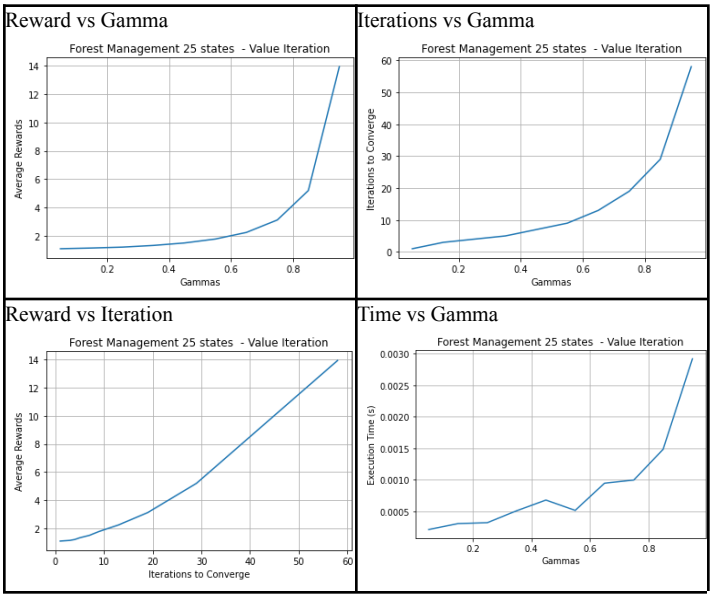
3.3.1. Value Iteration

The Forest Management VI uses the same convergence criteria as the Frozen Lake VI in the previous section, i.e. when the delta in utility falls below the threshold (epsilon).

Different combinations of gamma and epsilon were tested. Turns out the highest gamma (0.9999) and the lowest epsilon (1e-12) yields the best reward.

Gamma	Eps	Time	Iter	Reward
0.1	1.00E-02	0	2	4.36
0.1	1.00E-03	0	3	4.3933
0.1	1.00E-08	0	8	4.396613
0.1	1.00E-12	0	12	4.396613
0.3	1.00E-02	0	4	5.460862
0.3	1.00E-03	0	6	5.489575
0.3	1.00E-08	0	15	5.491933
0.3	1.00E-12	0	22	5.491933
0.6	1.00E-02	0	11	8.797055
0.6	1.00E-03	0	15	8.808703
0.6	1.00E-08	0	33	8.809994
0.6	1.00E-12	0.01	48	8.809994
0.9	1.00E-02	0	39	23.089675
0.9	1.00E-03	0	50	23.147532
0.9	1.00E-08	0.01	105	23.17236
0.9	1.00E-12	0.01	149	23.172433
0.9999	1.00E-02	0.02	210	134.486464
0.9999	1.00E-03	0.02	232	144.907286
0.9999	1.00E-08	0.04	341	196.537389
0.9999	1.00E-12	0.04	429	238.219997

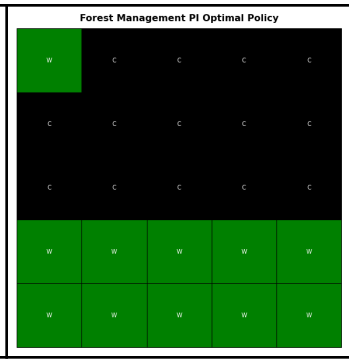
Notice that the change in the value of epsilon doesn't affect much on the reward when the gamma is smaller or equal to 0.9. But as gamma becomes very close to 1, the change in epsilon affects the reward significantly. It is because when the long-term reward weighs the same as the short-term reward (i.e. gamma=1), the agent will always choose 'WAIT' to gain the greatest rewards (assuming max iterations is unlimited). Setting epsilon close to zero tells the algorithm to find the optimal policy until no more difference in the true values. Thus the reward gets improved by iterating hundreds times more. This is also explained why the rewards rise sharply once gamma is larger than 0.9.



Compared to Frozen Lake, this Forest Management is more complicated because there are multiple rewards in the system instead of one final reward at the goal. An agent can choose to "WAIT" in the final state or "CUT" in every non-initial state. There is no termination state thus the agent can gain reward endlessly if the max number of iterations is unlimited. That's why the reward increases almost linearly with the number of iterations instead of being capped by the end goal like Frozen Lake.

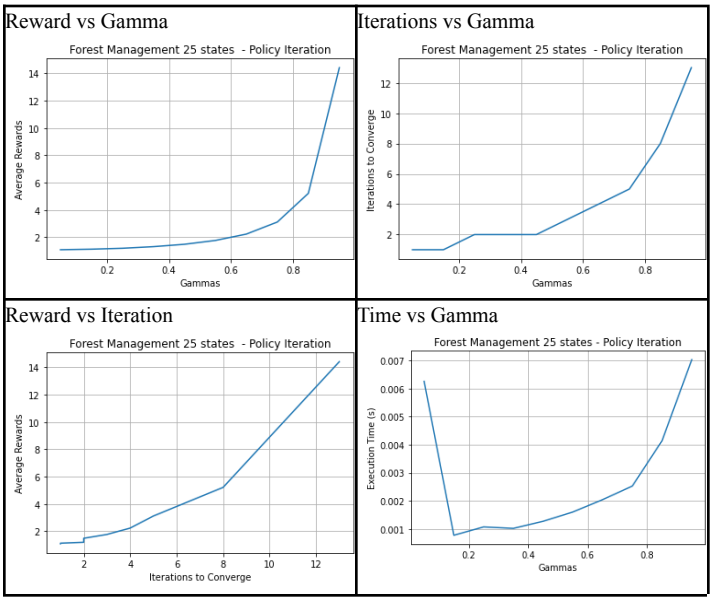
Since the state is 25 in this Forest MDP which is relatively small, the time to converge should be pretty fast. It only costs 0.003s (when gamma=0.9) for VI to converge, as expected.

The optimal policy for VI is shown as below:

Reward = 23.17 Gamma = 0.9 Epsilon= 1E-12	

3.3.2. Policy Iteration

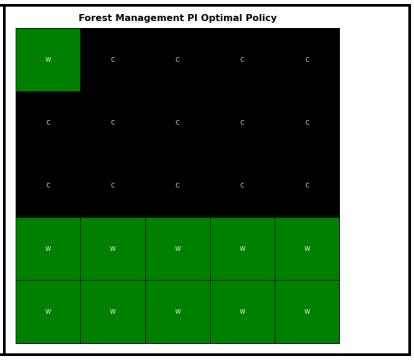
Similar to the Frozen Lake MDP, epsilon is set to be 1e-5 as the convergence criteria for this 25-state Forest Management.



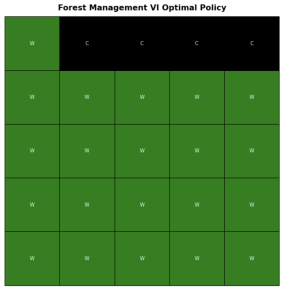
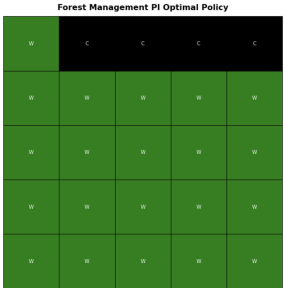
Compared to VI in the previous section, PI uses on average 0.005s to converge. The average time spent to converge for PI is close to the one for VI. The reward gained from the best policy of PI is the same as VI. However, the number of iterations to converge is much less than VI. PI only uses 12 iterations but VI uses 60 iterations (when gamma=0.9).

Compared to the small Frozen Lake PI, the small Forest PI took longer time and more iterations to converge in general. PI in small Frozen Lake only took 0.002s and 6 iterations to converge while PI in small Forest needed 0.007s and 12 iterations to converge. It is expected because Forest Management is a more complex MDP in the sense that it is a spatially-linked environment and not all the states are known in the system like the grid world setting.

The optimal policy for PI is shown as below:

Reward = 23.17 Gamma = 0.9 Epsilon= 1E-5	

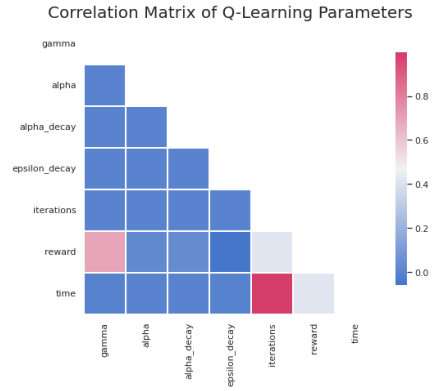
Turns out both PI and VI come up with the same optimal policy. Both algorithms choose to "CUT" every year in the beginning to gain immediate reward and then "WAIT" near the final states to maximize its final rewards. That means if time is unlimited and infinite steps are allowed, the agent will always choose "WAIT" to gain the maximum amount of reward. Out of curiosity, I've tried gamma=0.999, epsilon=1e-99 and max iteration is super high which mimics infinity steps for an agent to run VI and PI. Turns out the agent often chooses "WAIT" in both VI and PI optimal policies (visualized below).

reward =227	reward =238
	

3.3.3. Q-learning

Similar to Frozen Lake MDP, different hyperparameters were tested to see their effect on the final reward and the time spent. A correlation matrix is plotted to see which hyperparameters affect the final reward the most.

gammas	[0.8, 0.9, 0.99]
alphas	[0.01, 0.1, 0.2]
alpha decays	[0.9, 0.999]
epsilon decays	[0.9, 0.999]
iterations	[1e5, 1e6, 1e7]

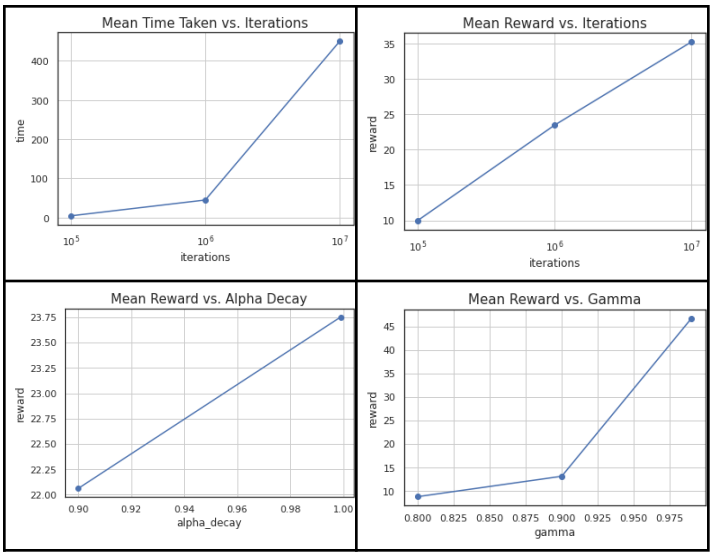


Observed from the correlation matrix, gamma and iterations had a larger impact on gaining reward from QL. Unlike previous VI and PI which converge within 1 second, QL takes 4 hours to

converge and used up the max number of iterations set (which is 10,000,000)

Similar to the small Frozen Lake QL, increasing gamma also improves the reward in this Forest QL. But the reward increases significantly here with a slight increase of gamma values, which is different from the Frozen Lake QL. In Frozen Lake QL, increasing gamma from 0.9 to 0.9999 only improves rewards by 1% but in Forest QL the reward is improved by at least a triple. It could be due to the chained states setup of the system in which an agent always "WAIT" to maximize its reward when gamma=0.9999.

Unlike Frozen Lake QL where the reward only increases slightly with the number of iterations, the reward in Forest QL increases greatly and almost linearly with the number of iterations. This is expected because there are multiple rewards in the system where an agent can always gain reward no matter if it cuts or waits.



The optimal policy for OL is shown as below:

Reward = 79.76
Gamma = 0.99,
Alpha = 0.10,
Alpha Decay: 0.999,
Epsilon Decay: 0.900,
Iterations: 1.0E+07

With gamma=0.99, QL can achieve a reward of 79 from the optimal policy. However, both VI and PI can achieve even higher rewards with gamma>=0.99. In the previous session, VI and PI's best policy yielded a maximum reward around 230 which is much higher than QL. Given that QL needs 4 hours to converge and the reward is lower than VI and PI, using QL is not very effective in solving the Forest Management problem.

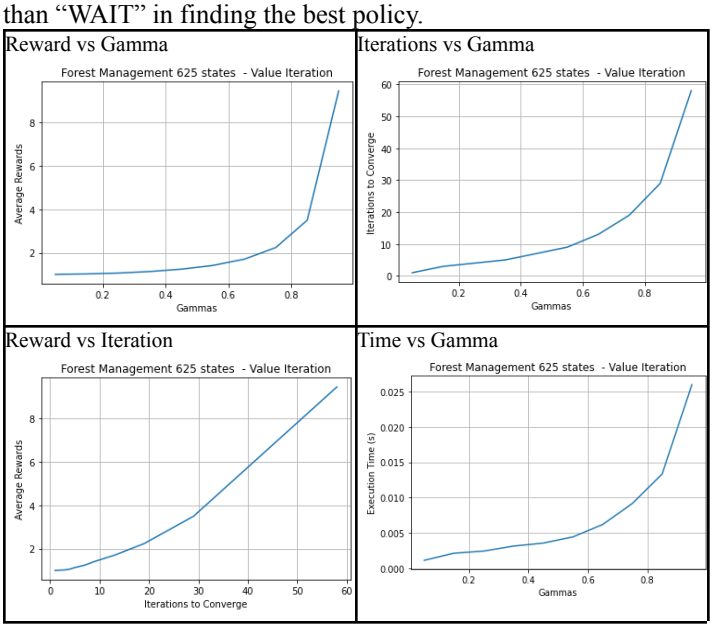
3.4. Forest Management (state=625)

Larger number of states is used to run the 3 RL. The number of iterations and time to converge are expected to be larger than the small Forest since the environment becomes more complex.

3.4.1. Value Iteration

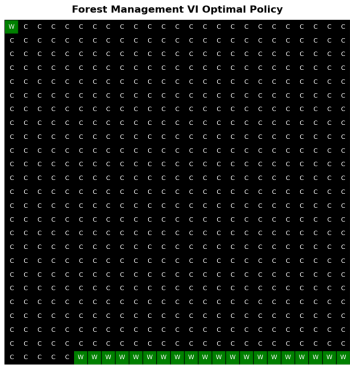
Similar to the small Forest, a combination of large gamma and small epsilon yields the greatest rewards from its optimal policy. As expected, the time to converge is generally longer than the VI in small Forest. The large forest VI took 0.3s to converge but the small forest VI took 0.01s to converge given gamma=0.9. The number of iterations also doubled compared to the small forest.

However, the reward in this larger forest is slightly lower than the small forest. It means the agent chooses "CUT" more often than "WAIT" in finding the best policy.



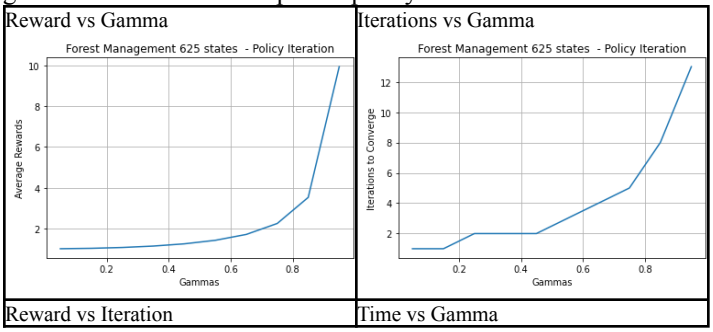
The optimal policy for VI is shown as below:

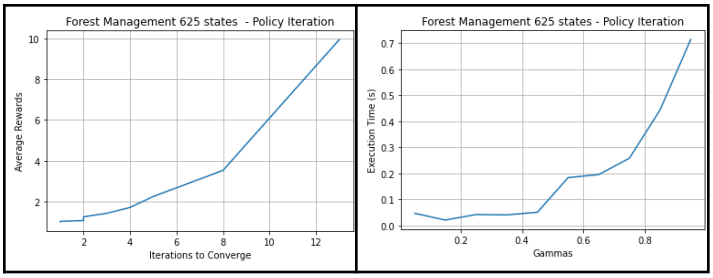
Reward = 238.22
Gamma = 0.9999999
Epsilon= 1E-12



3.4.2. Policy Iteration

Same as the small forest, a constant epsilon was used to define the convergence of PI. Turns out the highest gamma yields the greatest reward from its optimal policy.



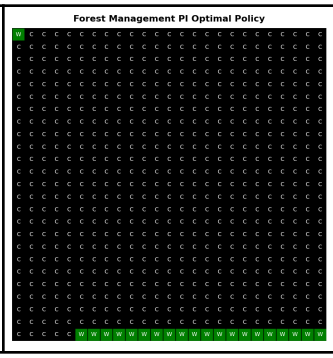


The time taken to converge for the large Forest PI is 0.6s which is 100 times longer than the small forest (0.006s). The number of iterations to converge is also larger than the small Forest PI. It is again due to the larger number of spatially-linked states where PI needs more iterations to reach optimal policy. Thus more time is required.

The reward in this PI is lower than the small forest when gamma<=0.9 is used. It means when the agent focuses more on the short-term rewards, it converges faster to a suboptimal policy which yields a lower total reward.

The optimal policy for PI is shown as below:

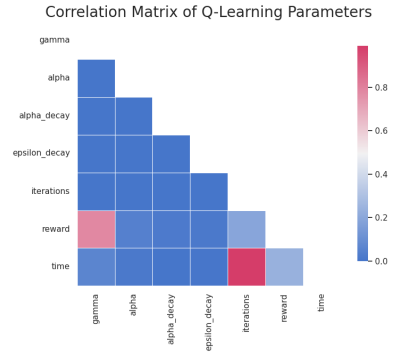
Reward = 508.39
Gamma = 0.999
Epsilon= 1E-5

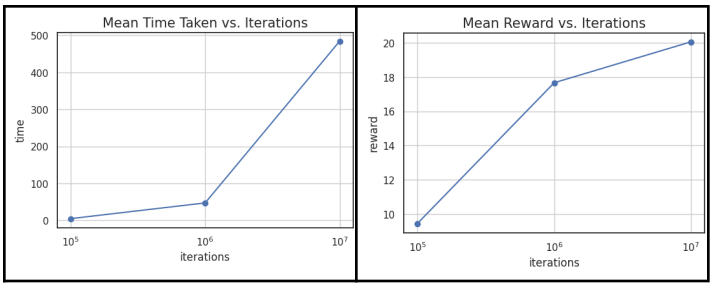


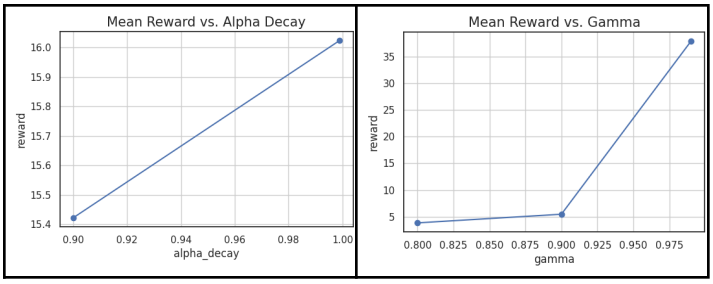
Both VI and PI converge to the same optimal policy, just like the small Forest did. This makes sense because both algorithms have models and know the reward which helps find the optimal policy.

3.4.3. Q-learning

Same as the QL in small Forest, gamma and iterations had larger impacts on the reward from QL's optimal policy.



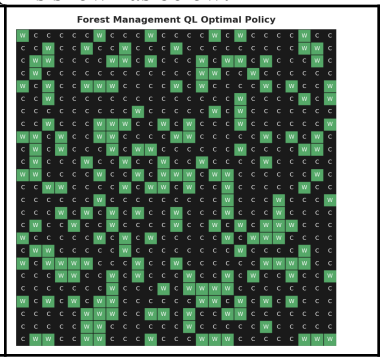




Although the graph looks similar to the small Forest QL in shape, the number of iterations and time spent to converge is greater than the small Forest QL. It takes over 5 hours to converge. The reward is generally lower than the small forest QL as well. This is expected because the q-learner needs to explore the environment before exploiting a best path. When the forest is large (state=625), the learner takes more iterations thus more time than a small forest (state=25) to converge. The q-learner would also have an exploration-exploitation dilemma in a large forest. Thus the resulting policy might not yield high rewards.

The optimal policy for QL is shown as below:

Reward = 47.68
Gamma = 0.99,
Alpha = 0.01,
Alpha Decay: 0.900,
Epsilon Decay: 0.999,
Iterations: 1.0E+07



QL gives the lowest reward among the 3 RL algorithms in a large forest. If time was allowed, giving more iterations and using higher alpha decay would result in a better policy which yields higher reward. However, the time cost would be significant.

IV. CONCLUSION

Both VI and PI can reach the best policies within a short period of time, even in a large grid world or large state non-grid world. Q-learning however, has difficulties in finding the best policy due to its model-free exploring nature. It interacts with the environment to collect states instead of interacting with a model like VI and PI do. If iterations are unlimited, Q-learning may also converge to a policy as good as VI and PI do.

Comparing the grid and non-grid world, previous experiments show that the grid world is relatively easier to converge than the non-grid world for PI since it uses less time to achieve the same reward. Q-learning takes the longest time to converge with a low reward in both grid and non-grid world.

V. REFERENCE

[1]	Convergence of Optimistic and Incremental Q-Learning. Retrieved November 27, 2022, from https://proceedings.neurips.cc/paper/2001/file/6f2688a5fce7d48c8d19762b88c32c3b-Paper.pdf
[2]	Roberts, S. (2021, July 12). Policy and Value Iteration. An Introduction to Reinforcement… | by Steve Roberts. Towards Data Science. Retrieved November 27, 2022, from https://towardsdatascience.com/policy-and-value-iteration-78501afb41d2