Assignment 4:
Markov Decision Processes

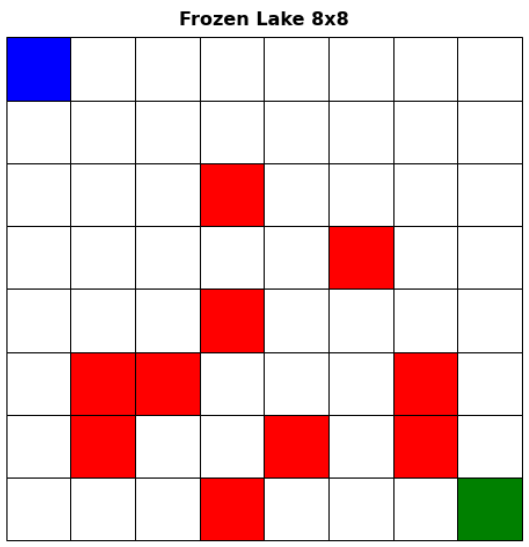
Vipul Koti
vkoti7@gatech.edu

1 INTRODUCTION

This report summarizes the analysis done on two Markov Decision Processes (MDP) problems, In MDP problems, an agent interacts with an environment, At each time step, the agent takes action which leads to change in environment state, and it also leads to some reward/penalty for taking this action. The goal is always to find an optimal policy which maximizes the total value of the rewards for reaching the goal state or for taking actions for certain time frame. I have chosen Frozen Lake (Grid World) and Forest Management (Non-Grid World) as the two MDP problems.These two problems are solved using Policy Iteration, Value Iteration and Q-Learning algorithms. The sizes of these two problems, parameters of the algorithms, are varied and analyzed in this report. I have used Open AI GYM, Markov Decision Process (MDP) Toolbox Python libraries for this analysis.

1.1 Problems Introduction

Frozen Lake - It is a grid world problem, where agent try to go from start position (blue square) to end (green square) navigating through solid squares (white) avoiding holes (red squares) by taking allowed actions (UP, LEFT, DOWN, RIGHT). The analogy for frozen lake, is an agent tries to navigate through the lake to get a lost Frisbee which is on the other side of the lake by walking on frozen squares and avoiding holes in the lake. The environment we choose is stochastic, which makes this problem interesting, so if the agent tries to take a step in one direction, there are 0.33 chance it will move in the correct direction and 0.33 chances each it will move to either side left/right of the current square. I have analyzed this problem for three sizes, 4×4, 8×8 and 32x32.



Forest Management Problem - It is a non-grid world problem, the agent has two actions (Wait(0) and Cut(1)). Action is decided based on two objectives, First to maintain the old forest for wildlife and second is to earn reward by selling woods. It has a stochastic component, where there is a probability (1-p) that a fire burns the forest. The states of this problem are [0, 1 . . . "S" -1], where 0 means the youngest state and "S" and -1 means the oldest. If the agent takes action 1, Cut the forest goes to state 0 and agent get reward r1(Selling woods), if the agent waits for higher reward r2(forest for wildlife), the forest state remains same, but there is a probability p of fire which brings back the forest to 0 (forest goes to state 0). This problem is interesting as it's a real world problem and due to the stochastic nature of the problem. In some states waiting is ideal and in some state cutting for immediate reward.

1.2 Algorithms Introduction

There are two types of algorithms we analyzed in our report, model based algorithms and model free algorithms. In model based learning the agent interacts with the environment and create state transition and reward models, these models are iterated using Value Iteration/Policy Iteration by the agent to find optimal policy. On the other hand, in model free learning, the agent doesn't learn an explicit state transition and rewards model, it interacts with the environment and derives the optimal policy.(Q-Learning, SARSA)
Few function definitions before discussing algorithms -
V(s), Value Function, It is equal to the expected total reward for an agent starting is state s. It represents how good is an environment state for the agent. The function with the highest reward is Optimal $V^{\hat{*}}$(s).
$Q^{\hat{*}}$(s,a), Q Function, is a function to get total reward for an agent in state s for taking action a. Or in other words, how good is it for agent to pick action a when in state s.
Bellman Equation, defines a recursive way to calculate optimal Q-function (dynamic programming paradigm).
$\pi^{\hat{*}}$(s), optimal policy, which gives the optimal action for state s with maximum total reward.

Value Iteration (VI)- The VI algorithm iteratively improves V(s) and compute the optimal state value. First we initialize V(s) to an arbitrary value, then update Q(s,a) and V(s) values until they converge. VI is guaranteed to converge to the optimal policy π^*.

Policy Iteration (PI)- In contrast to VI, in PI instead of repeatedly improving the V(s) estimate, it re-defines the policy π at each step and compute the V(s) according to this new policy π until the policy converges. As, sometime the optimal policy converges before the value function, it often takes fewer iterations than VI. PI is also guaranteed to converge to the optimal policy π^*.

Q- Learning (QL)- It is an example of model free learning, Agent has no state transition and reward model, it discovers the optimal policy by trial and error. (Exploration and Exploitation). In QL, we basically approximate Q function using the Q(s,a) samples, that we observed previously. The approach is called Time-Difference Learning. Three Greek letter hyperparameters. Alpha, which is learning rate, it tells how much we should trust our observed samples to estimate Q function. Gamma, defines how important is the future reward and epsilon, which is used to balance explore and exploit, for some random chance less than epsilon, we take random actions. For q-Learning we decay both alpha and Epsilon.

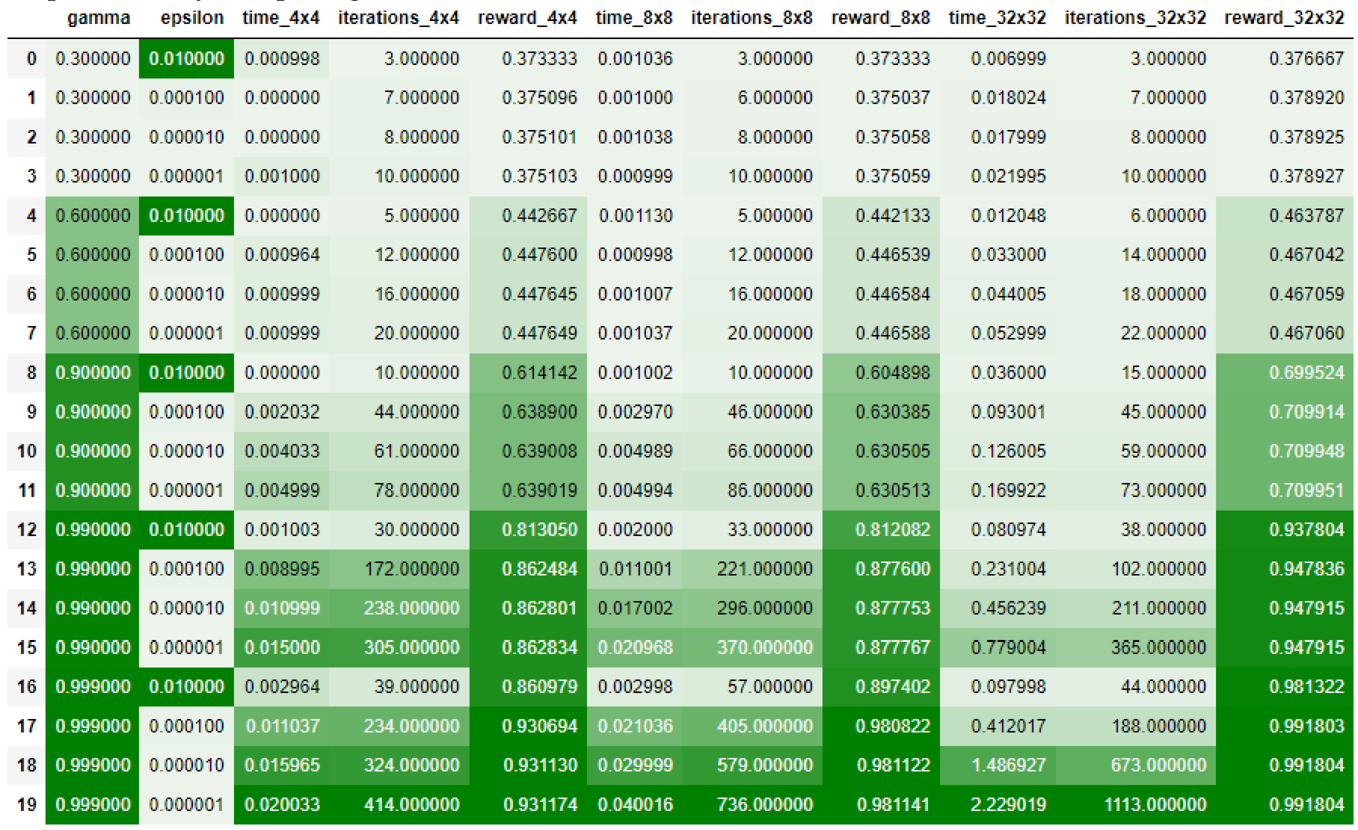
2 GRID WORLD PROBLEM (FROZEN LAKE (FL)) ANALYSIS -

I did my analysis on problem sizes of 4x4 (16 states), 8x8 (64 states) and 32x32 (1024 states). I used Open AI gym to create the environment and then from the environment I computed the Transaction and reward models in (A,S,S) matrix's. These models are then feed into MDPtoolbox Value Iteration method, with combinations of Gamma and epsilon, to see the impact of these hyperparameters. **Note, We are using the stochastic FL and there is no negative reward for each step, reward to reach the goal is 1 for below experiments in the**

table. Also, I have done analysis for each algorithm with negative reward at each step (Impacts Q-learning Convergence time) explained in respective subsections

2.1 Value Iteration (VI) Analysis

For tuning the hyperparameters for VI. Gamma, I have used [0.3, 0.6, 0.9, 0.99, 0.999] and for epsilon I used [1e-2, 1e-4, 1e-5, 1e-6]. Below is the table which shows the impact of gamma and epsilon on VI for all three size problems by comparing time iterations and rewards.



	gamma	epsilon	time_4x4	iterations_4x4	reward_4x4	time_8x8	iterations_8x8	reward_8x8	time_32x32	iterations_32x32	reward_32x32
0	0.300000	0.010000	0.000998	3.000000	0.373333	0.001036	3.000000	0.373333	0.006999	3.000000	0.376667
1	0.300000	0.000100	0.000000	7.000000	0.375096	0.001000	6.000000	0.375037	0.018024	7.000000	0.378920
2	0.300000	0.000010	0.000000	8.000000	0.375101	0.001038	8.000000	0.375058	0.017999	8.000000	0.378925
3	0.300000	0.000001	0.001000	10.000000	0.375103	0.000999	10.000000	0.375059	0.021995	10.000000	0.378927
4	0.600000	0.010000	0.000000	5.000000	0.442667	0.001130	5.000000	0.442133	0.012048	6.000000	0.463787
5	0.600000	0.000100	0.000964	12.000000	0.447600	0.000998	12.000000	0.446539	0.033000	14.000000	0.467042
6	0.600000	0.000010	0.000999	16.000000	0.447645	0.001007	16.000000	0.446584	0.044005	18.000000	0.467059
7	0.600000	0.000001	0.000999	20.000000	0.447649	0.001037	20.000000	0.446588	0.052999	22.000000	0.467060
8	0.900000	0.010000	0.000000	10.000000	0.614142	0.001002	10.000000	0.604898	0.036000	15.000000	0.699524
9	0.900000	0.000100	0.002032	44.000000	0.638900	0.002970	46.000000	0.630385	0.093001	45.000000	0.709914
10	0.900000	0.000010	0.004033	61.000000	0.639008	0.004989	66.000000	0.630505	0.126005	59.000000	0.709948
11	0.900000	0.000001	0.004999	78.000000	0.639019	0.004994	86.000000	0.630513	0.169922	73.000000	0.709951
12	0.990000	0.010000	0.001003	30.000000	0.813050	0.002000	33.000000	0.812082	0.080974	38.000000	0.937804
13	0.990000	0.000100	0.008995	172.000000	0.862484	0.011001	221.000000	0.877600	0.231004	102.000000	0.947836
14	0.990000	0.000010	0.010999	238.000000	0.862801	0.017002	296.000000	0.877753	0.456239	211.000000	0.947915
15	0.990000	0.000001	0.015000	305.000000	0.862834	0.020968	370.000000	0.877767	0.779004	365.000000	0.947915
16	0.999000	0.010000	0.002964	39.000000	0.860979	0.002998	57.000000	0.897402	0.097998	44.000000	0.981322
17	0.999000	0.000100	0.011037	234.000000	0.930694	0.021036	405.000000	0.980822	0.412017	188.000000	0.991803
18	0.999000	0.000010	0.015965	324.000000	0.931130	0.029999	579.000000	0.981122	1.486927	673.000000	0.991804
19	0.999000	0.000001	0.020033	414.000000	0.931174	0.040016	736.000000	0.981141	2.229019	1113.000000	0.991804

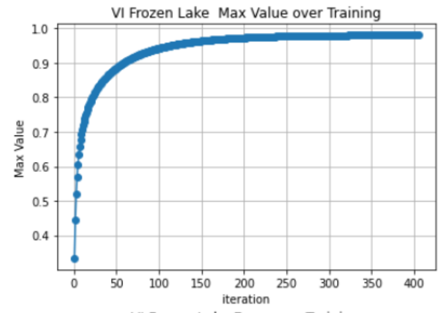
Gamma, discount factor, this is per time step discount factor on future rewards, it varies between 0 and 1, and the higher the gamma, the more you value the future rewards.

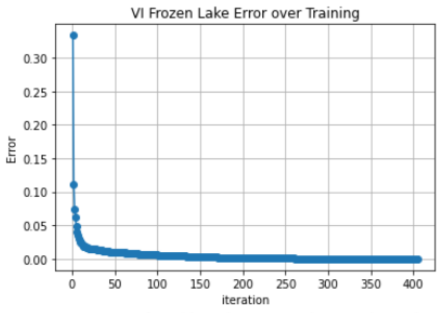
Gamma impact on Iterations, Time, and Reward- From the table we can see, as we increase the gamma, the number of iterations increases. This is because the reward to reach the goal takes more iterations to decay close to 0 (when jumping in the hole is equivalent to future reward). Interesting observations, when we see iterations for different size problems for different Gamma size, for small gamma size 4x4, 8x8, 32x32 takes equal iterations but for higher value like 0.99 and 0.999, we see it takes more time to iterate as time is directly proportional to the size of the problem, the reason being the future reward is discounted by a very low factor, and as we increase the size of the problem the number of holes increases, complexity increases, so there is more variation in the value function which takes more iterations to converge. Iterations are directly proportional to time. Also, as Gamma, discounts the reward on each step, gamma value is directly proportional to the reward.

Epsilon, Stopping Criteria, On each iteration the change is maximum V(s) is compared against epsilon, if the change falls below the epsilon value the V(s) is considered converged to $V^{\hat{*}}(s)$

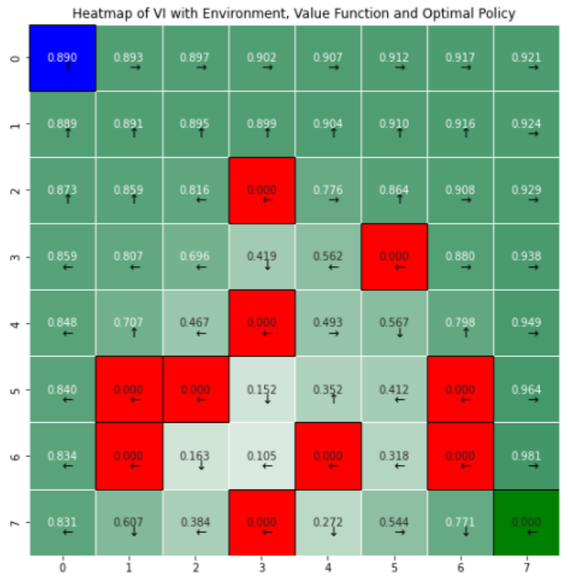
Epsilon impact on Iterations, Time, and Reward- Epsilon is inversely proportional to time iterations and reward for all three size problems. This is because epsilon is stopping criteria, if the epsilon value is high the value functions converges early, if the epsilon value is small, it runs for more iterations to improve the values function and reduce the difference between max V(s) at each iteration. Interestingly, for Epsilon value 0.01 and 0.0001 and gamma value 0.999, we see PI takes fewer transactions for complex problem 32x32 in comparison to 4x4 and 8x8, this is because, PI runs VI at each step, for 32x32 problem policy converges faster for high epsilon value.

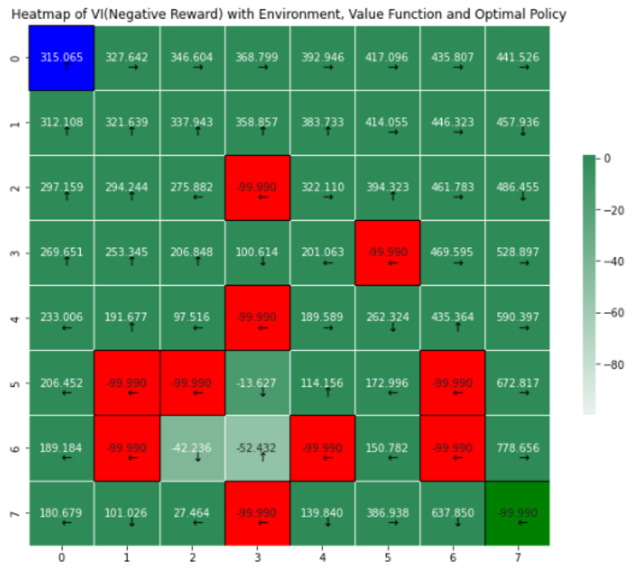
Final chosen Gamma and Epsilon for VI, We will use Gamma value of 0.999 and epsilon value of 0.0001 reason being, PI iterative algorithm uses epsilon of 0.0001(MDPToolbox), equal comparison.





Frozen Lake 8x8, VI Convergence plot and policy Analysis - From the plots of left and table above, we can see FL 8x8, with gamma 0.999 and epsilon 0.0001 converges in 405 iterations of VI. We can see in the plots the Max Value function and Error plateaus and converges. The converged policy is in the below image on the left, As our environment is stochastic, we can all the arrows in the row 1 pointed up and column 7 pointed right so that there is no chance for the agents to go into the red holes. Also, all the squares surround the red holes moves outward. To explore different Strategy, I made -1 reward for each stage and 1000 reward goal state Green (7,7). The resulted policy is the policy on the right. This policy took more than double iterations (918) and 5 times more time.(0.5 compared to 0.1). The show's negative rewards strategy negatively impacts the VI algorithm.

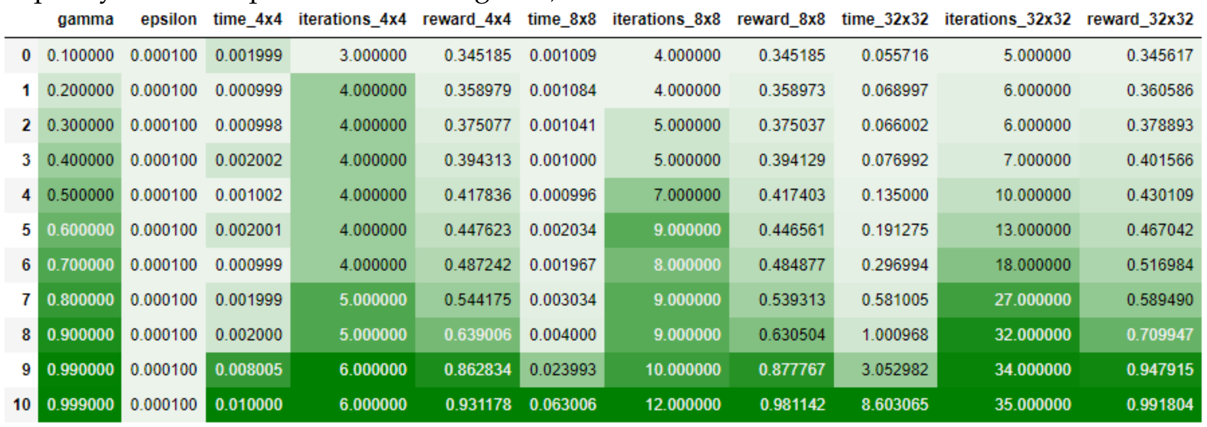




By comparing both the policies side by side, we can see the policy with -1 reward safely more directed towards the reward. Some arrows in row 1 and column 7 are pointed towards the reward. Also, we can see square (6,3) chooses to go up rather than away from the reward with equal chance of falling in the hole. Value function is negative in the squares surrounded by holes.

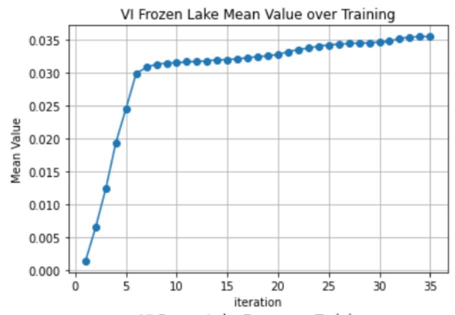
2.2 Policy Iteration (VI) Analysis

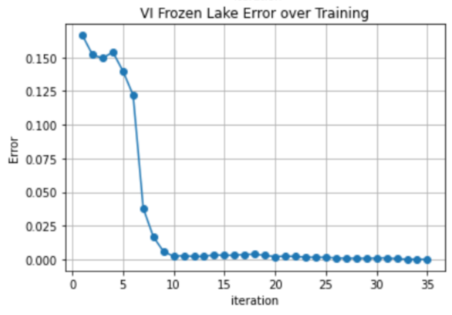
For tuning the hyperparameters for PI, I have use 11 different values of Gamma [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.999], Epsilon value for policy iteration method in MDPtoolbox is 0.0001 (VI at each step using the policy use this epsilon for convergence).



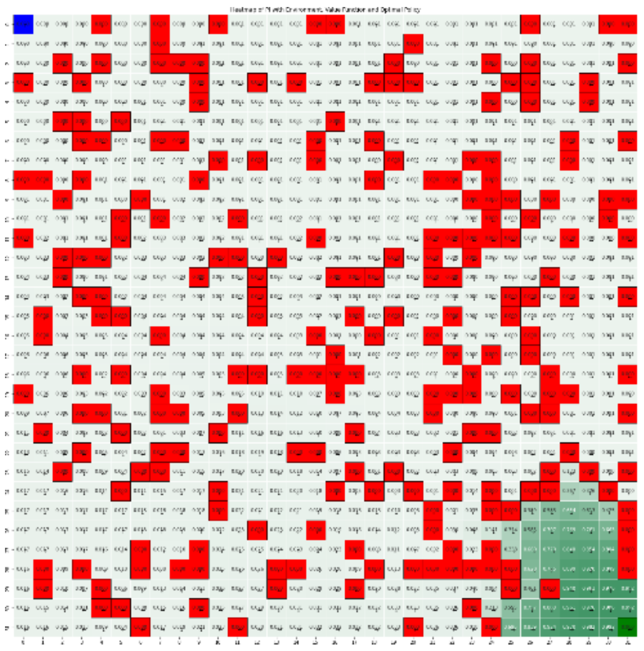
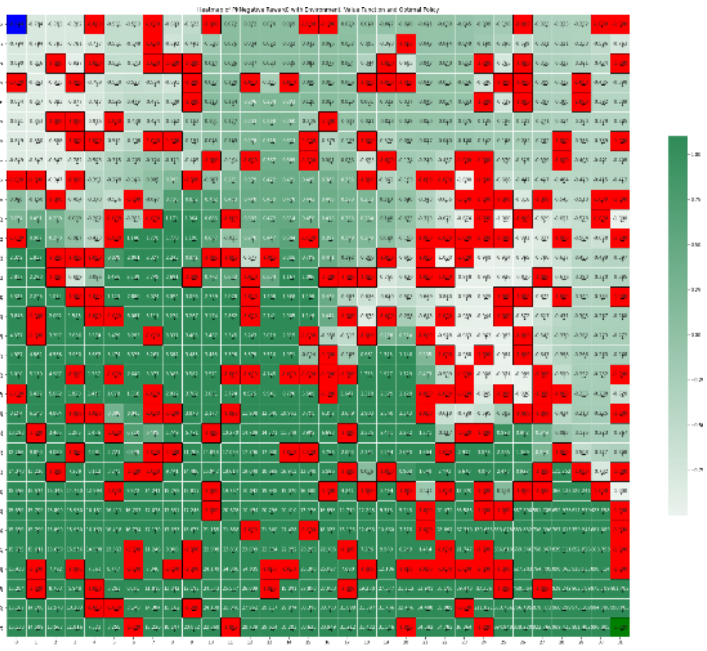
	gamma	epsilon	time_4x4	iterations_4x4	reward_4x4	time_8x8	iterations_8x8	reward_8x8	time_32x32	iterations_32x32	reward_32x32
0	0.100000	0.000100	0.001999	3.000000	0.345185	0.001009	4.000000	0.345185	0.055716	5.000000	0.345617
1	0.200000	0.000100	0.000999	4.000000	0.358979	0.001084	4.000000	0.358973	0.068997	6.000000	0.360586
2	0.300000	0.000100	0.000998	4.000000	0.375077	0.001041	5.000000	0.375037	0.066002	6.000000	0.378893
3	0.400000	0.000100	0.002002	4.000000	0.394313	0.001000	5.000000	0.394129	0.076992	7.000000	0.401566
4	0.500000	0.000100	0.001002	4.000000	0.417836	0.000996	7.000000	0.417403	0.135000	10.000000	0.430109
5	0.600000	0.000100	0.002001	4.000000	0.447623	0.002034	9.000000	0.446561	0.191275	13.000000	0.467042
6	0.700000	0.000100	0.000999	4.000000	0.487242	0.001967	8.000000	0.484877	0.296994	18.000000	0.516984
7	0.800000	0.000100	0.001999	5.000000	0.544175	0.003034	9.000000	0.539313	0.581005	27.000000	0.589490
8	0.900000	0.000100	0.002000	5.000000	0.639006	0.004000	9.000000	0.630504	1.000968	32.000000	0.709947
9	0.990000	0.000100	0.008005	6.000000	0.862834	0.023993	10.000000	0.877767	3.052982	34.000000	0.947915
10	0.999000	0.000100	0.010000	6.000000	0.931178	0.063006	12.000000	0.981142	8.603065	35.000000	0.991804

Gamma (discount factor) impact on Iterations, Time, and Reward- Gamma basically allow the algorithm to see the future. If the Gamma is high, means we value the future more, the reward (Max V) will be high. We see the same behavior in the table above, as we increase gamma, all the time, iteration and rewards increases. For 4x4 and 8x8 we see iteration count doesn't change much for different gammas, but for 32x32 it varies from 5 to 35, this is because the environment has more holes, its more state to explore and the iterations increases with the stochastic complex nature of the problem. The reward (Max V) is always high for 32x32 problem as it depends on how many times the VI runs, and it is increasing with the iterations.





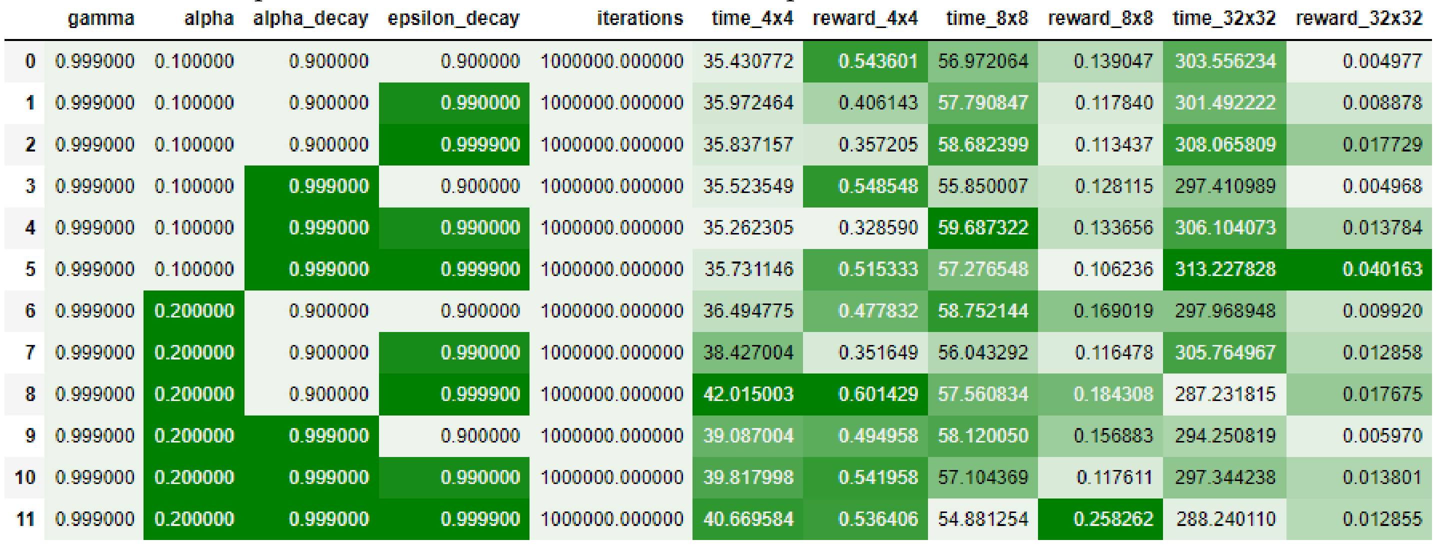
Frozen Lake 32x32, PI Convergence plot and policy Analysis - We selected Gamma 0.999 same as VI and epsilon is 0.0001 also same as VI. From the plots of left and table above, we can see FL 32x32, with gamma 0.999 and epsilon 0.0001 converges in 35 iterations of PI(in comparison to 188 iterations for VI). This is because PI runs VI at every iteration, and the policy converges faster. We can see in the plots the Max Value function and Error plateaus and converges. The converged policy is in the below image on the left, we can see how the reward 1 propagates to the other state, the state space is huge for reward 1, and it is equal for most of the state. To explore a different Strategy, I made -0.01 reward for each stage and 1000 reward goal state. The resulted policy is the policy on the right. This policy took approx 150 iterations in comparison to 33 earlier. This shows for model based learning, negative rewards negatively impacts the policy convergence.

Also, we see the same behavior (top square has less value than bottom squares as top squares are hard to navigate with the holes structure) if we do negative reward on each state, the policy tries to move towards the goal safely (same as VI), it seems the state in the bottom are safer to move with the 0.33 probability stochastic behavior due to which we see good value function on those states.

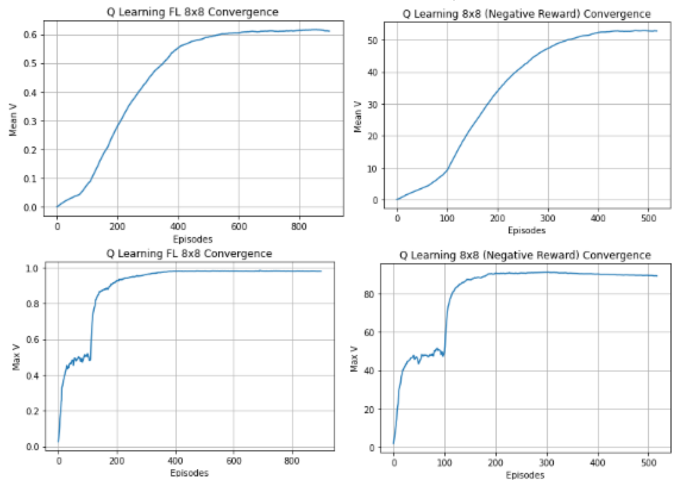
2.3 Q Learning (QL) Analysis

For Q learning Hyperparameter tuning, Gamma had similar effects as we saw in VI and PI. So I used Gamma as 0.999 same as PI and VI. For Alpha, I used [0.1,0.2], Alpha_Decay[0.9,0.999], epislon_decay[0.9,0.99,0.9999] and I used default epsilon, and a million iterations for each problem size as seen in the table below.



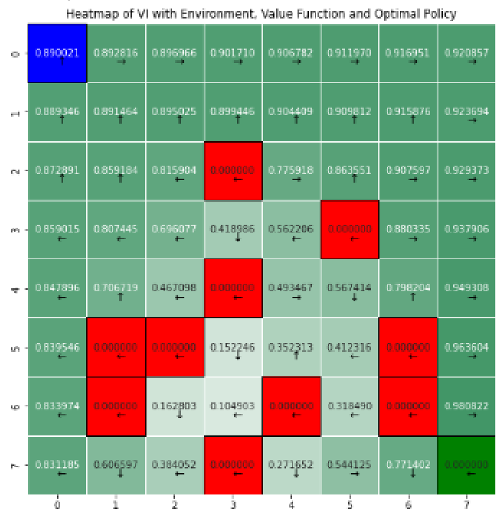
	gamma	alpha	alpha_decay	epsilon_decay	iterations	time_4x4	reward_4x4	time_8x8	reward_8x8	time_32x32	reward_32x32
0	0.999000	0.100000	0.900000	0.900000	1000000.000000	35.430772	0.543601	56.972064	0.139047	303.556234	0.004977
1	0.999000	0.100000	0.900000	0.990000	1000000.000000	35.972464	0.406143	57.790847	0.117840	301.492222	0.008878
2	0.999000	0.100000	0.900000	0.999900	1000000.000000	35.837157	0.357205	58.682399	0.113437	308.065809	0.017729
3	0.999000	0.100000	0.999000	0.900000	1000000.000000	35.523549	0.548548	55.850007	0.128115	297.410989	0.004968
4	0.999000	0.100000	0.999000	0.990000	1000000.000000	35.262305	0.328590	59.687322	0.133656	306.104073	0.013784
5	0.999000	0.100000	0.999000	0.999900	1000000.000000	35.731146	0.515333	57.276548	0.106236	313.227828	0.040163
6	0.999000	0.200000	0.900000	0.900000	1000000.000000	36.494775	0.477832	58.752144	0.169019	297.968948	0.009920
7	0.999000	0.200000	0.900000	0.990000	1000000.000000	38.427004	0.351649	56.043292	0.116478	305.764967	0.012858
8	0.999000	0.200000	0.900000	0.999900	1000000.000000	42.015003	0.601429	57.560834	0.184308	287.231815	0.017675
9	0.999000	0.200000	0.999000	0.900000	1000000.000000	39.087004	0.494958	58.120050	0.156883	294.250819	0.005970
10	0.999000	0.200000	0.999000	0.990000	1000000.000000	39.817998	0.541958	57.104369	0.117611	297.344238	0.013801
11	0.999000	0.200000	0.999000	0.999900	1000000.000000	40.669584	0.536406	54.881254	0.258262	288.240110	0.012855

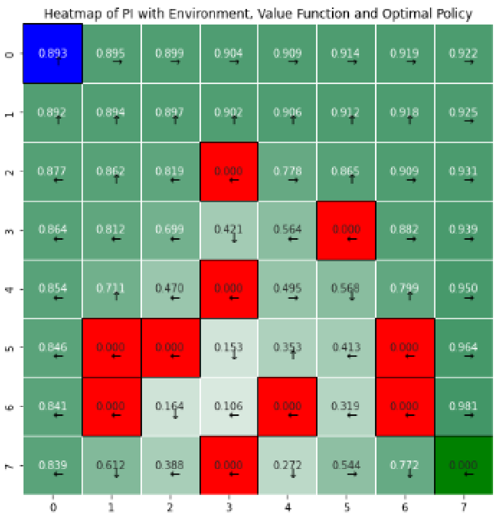
Gamma, discount factor, we discussed in VI section. **Alpha, Learning Rate** Alpha is learning rate, means how much we value the observed Q(s, a). **Epsilon, for random action probability,** so based on this value we explore the environment by taking random actions. **alpha_decay & epsilon_decay** decays the alpha and epsilon over time, so that we can exploit (low alpha, less epsilon randomness) from the observed Q table. For the table we can see, a million iterations are way less to learn this problem for all different sizes. When I use high alpha the learning rate increases, it's good for 4x4 & 8x8 problem, but for a large problem, we might need more iterations to see the effect. Interestingly, the highest reward for 32x32 is for lower alpha. Epsilon decay has the highest impact as this problem has 4 actions plus larger state space like 32x32 and 8x8 shows better reward for 0.9999. So we chose, alpha = 0.1 (default), alpha_decay(0.999, decay alpha slowly and trust observation values), epsilon_decay(0.9999, high exploration). Also we ran the Q learning algorithm for 1000 episodes(upper limit) with 1e5 iterations each, resetting alpha and epsilon at each iteration (Epsilon helped to jump out of local minima) with convergence criteria as, **If 10 consecutive episodes with N_VAR_L10_MeanV(numpy variance of last 10 episodes max mean V for last 10 episode) - P_VAR_L10_MeanV(numpy variance of last 10 episodes max mean V for previous last 10 episodes) is less than 0.0001 and Maximum of Max V is greater than 0.99 (which is optimal policy).**

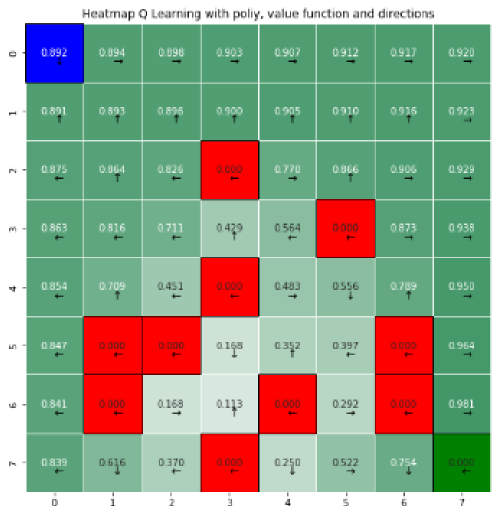


Frozen Lake 8x8, QL Convergence plot and policy Analysis - From the plots of left, we can see FL 8x8, converges in 600 episodes of 1e5 transaction each. This is high because Q learning is a model free algorithm and takes time to explore, I tried two approaches, one with negative rewards(-0.01 reward for each stage and 100 reward goal state) and one where I decrease the epsilon decay to 0.9 instead of 0.9999 after 420 episodes. With negative rewards, the meanV converged around 400 episodes and MaxV around 200 episodes which is comparatively less than 600 and 400 for non-negative rewards (plots on left), **as expected, negative rewards will make Q learning to move faster to the goal.** Changing decay helped to converge early, but produced the policy with suboptimal value function.

Policy Comparison VI, PI and QL (8x8 FL) Conclusion-







When we see the three policies, side by side as we know PI and VI policies are guaranteed to converge to the

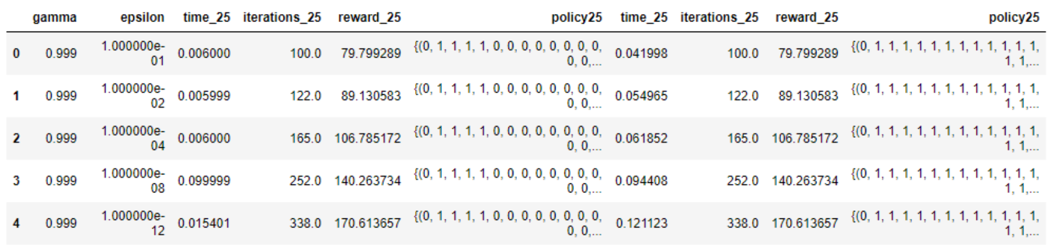
optimal policy, we can QL policy also looks very similar to PI and VI, The max V(s) for the optimal policy is also very close. PI and VI are the same directions for each square, but QL policy as it tries random actions because it is model-less, sqaure (3,3)(6,3) for QL is different, it seems to move towards high rewards.
Conclusion - For FL problem, when we compare model based learning algorithms, VI and PI, PI takes fewer iterations to converge but more time as it does more computation at each iteration. PI and VI converges and give optimal policy as guaranteed. Whereas, model Free learning takes a lot of time (hours) and billions of iterations because the problem is stochastic and has 4 actions, it does find the optimal policy. **Negative reward strategy helps QL converge faster for all sizes. (400 episodes in comparison to 600, 33% faster)**

3 NON-GRID WORLD PROBLEM (FOREST MANAGEMENT (FM)) ANALYSIS -

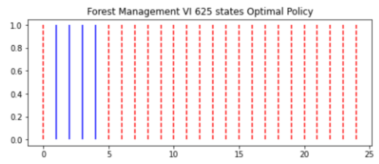
For Forest management, I analyzed two size problems, 25 states and 625 states. I have kept the reward r1, reward r2 and probability to default values(4,2 & 0.1). I have used MDPToolbox to generate and solve these MDP problems. We will discuss both sizes problems side by side for each algorithm.

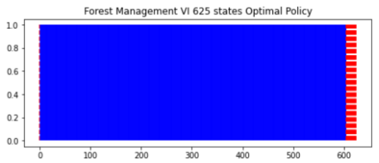
3.1 Value Iteration (VI) for FM 25 & 625 states

From the below table, we can see the policy doesn't change at all (both 25 and 625 states) for the selected gamma (changing gamma changes the policy, as we can look farther by making gamma high). This is because it's a chain state problem (V(s) same for all C states) and FM problem converges to ideal policy with first iteration, and it runs max iterations based on gamma and epsilon selected.



	gamma	epsilon	time_25	iterations_25	reward_25	policy25	time_25	iterations_25	reward_25	policy25
0	0.999	1.000000e-01	0.006000	100.0	79.799289	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	0.041998	100.0	79.799289	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
1	0.999	1.000000e-02	0.005999	122.0	89.130583	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	0.054965	122.0	89.130583	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
2	0.999	1.000000e-04	0.006000	165.0	106.785172	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	0.061852	165.0	106.785172	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
3	0.999	1.000000e-08	0.099999	252.0	140.263734	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	0.094408	252.0	140.263734	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
4	0.999	1.000000e-12	0.015401	338.0	170.613657	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	0.121123	338.0	170.613657	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...



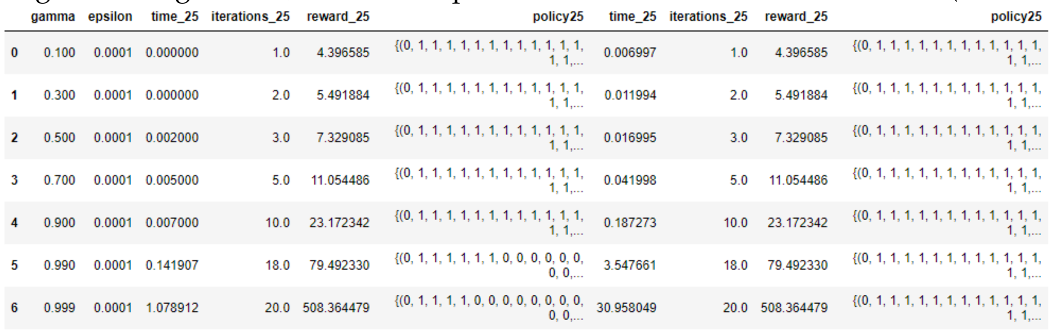


The reason, the rewards are also same for 25 and 625 states is, the wait state and cut state has same reward for both the problems and reward is Max V value with same gamma value 0.999 and epsilon value the highest Max V value converges to same for different size problems. **To check what's going on, First I ran VI with Gamma 0.1 for 25 and 625(r1 4 and r2 2) and noticed the Value function and Policy, only first and last state were W. Because V(s) for W in last is 4+ 0.1 * $(\hat{V}^{*}(s'))$ = 4.39, and for second last state if we do W V(s) = r(s) + 0.1(4) (considering W in last state) which is less than discounted Cut reward at second last state hence Vmax(s) for the state is C. When we increase the Gamma to 0.3, now for second last state v(s) = r(s) + 0.3(4) which is greater than 1 and hence last two states are W. So as we increase the gamma, the W increases from the last state. Because we get waiting reward only by waiting in last state, and it propagates from there, Also the value function for all C state is same, as they all get r2 for Cut and gamma times the reward for state 0. First state is always W as there is no reward for C at state 0.** These reason also proves why 25 states and 625 states and has exactly same W from the last state. (When we selected the same Gamma).

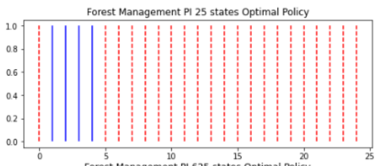
From the policy plots on the right above, we can see (red, means wait and blue means cut) we see first state is always w as s=0 has no reward, and all the Waits again in both 25 and 625 is in the end, rest in between is all the cuts i.e VI takes the cut rewards instead of waiting. Interesting, both policies has one beginning W and 20 W in the end (because of same Gamma, explained earlier), which also affirms why policies for different sizes, converge at same Max V as last state always has the same largest V(s). **(VI convergence plot in PI section)**

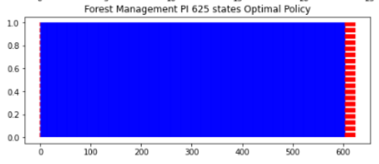
3.2 Policy Iteration (PI) for FM 25 & 625 states

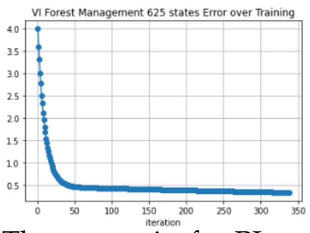
From the below table, we can see the policy changes with the change in gamma, as expected. We use default epsilon 0.0001 (MDP toolbox default) for our PI. The number of iterations is less than VI. But the time for convergence is high on PI, which is expected as PI runs VI at each iteration.(More computation at each iteration)

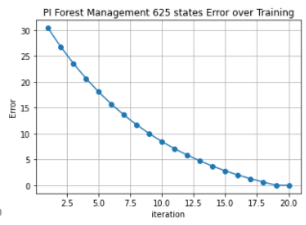


	gamma	epsilon	time_25	iterations_25	reward_25	policy25	time_25	iterations_25	reward_25	policy25
0	0.100	0.0001	0.000000	1.0	4.396585	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	0.006997	1.0	4.396585	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
1	0.300	0.0001	0.000000	2.0	5.491884	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	0.011994	2.0	5.491884	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
2	0.500	0.0001	0.002000	3.0	7.329085	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	0.016995	3.0	7.329085	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
3	0.700	0.0001	0.005000	5.0	11.054486	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	0.041998	5.0	11.054486	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
4	0.900	0.0001	0.007000	10.0	23.172342	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	0.187273	10.0	23.172342	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
5	0.990	0.0001	0.141907	18.0	79.492330	{(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,...	3.547661	18.0	79.492330	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
6	0.999	0.0001	1.078912	20.0	508.364479	{(0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	30.958049	20.0	508.364479	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...



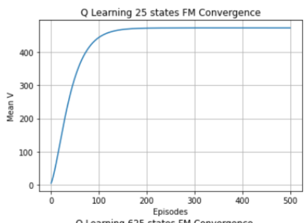


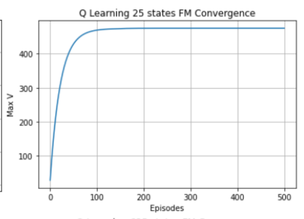


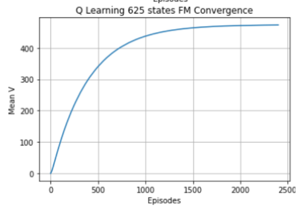


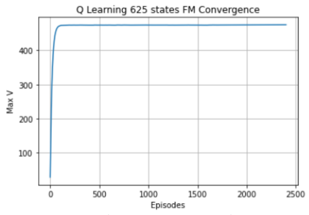
For Gamma 0.999 epsilon 0.0001, VI takes 165 iterations and converges at 165 Max V, whereas PI takes 20 iterations and converges at 508 reward. From the convergence plot of VI and PI error also we can see the same VI converges at error 0.5 and PI converges at 0. (Converged policies are the same.) The reason is, for PI max iteration also it considers old policy, gamma, and epsilon. But PI runs VI at each step, the reward/Max V, error depends on how many times the VI runs. This is also the reason for same iterations for small and big problem. **The reason for same number of W, Max V for 25 and 625 state is same as VI.**

3.3 Q-Learning (QL) for FM 25 & 625 states

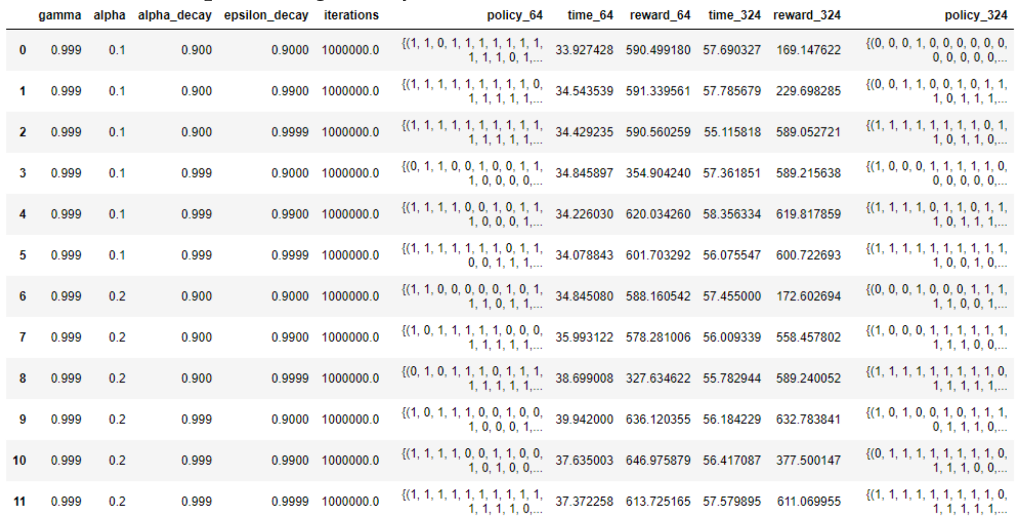




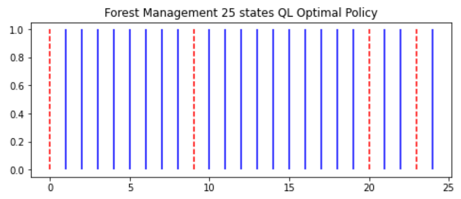


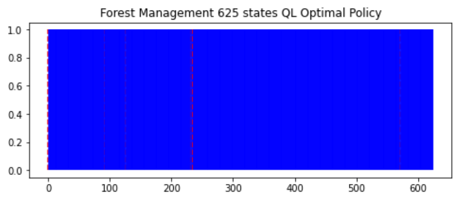


For Q learning, I choose same Gamma 0.999 and varied alpha [0.1, 0.2], alpha_decay[0.9, 0.999], epsilon_decay [0.9, 0.99, 0.999] and epsilon default value as in the table above. I ran QL for both 25 and 625 states, 1 million iterations each. Higher Alpha means high learning rate, so reward moves faster to max V. Epsilon, introduces randomness of actions for exploration. Epsilon decay positively impact reward for 0.9 to 0.99, but it negatively impacts reward with epsilon at 0.9999. This is because of the continuous chain state nature of this problem, where on cutting trees it goes to state 0 and on fire it goes to state 0. So a very high randomness, for 1M impacts negatively on the reward.



	gamma	alpha	alpha_decay	epsilon_decay	iterations	policy_64	time_64	reward_64	time_324	reward_324	policy_324
0	0.999	0.1	0.900	0.9000	1000000.0	{(1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,...	33.927428	590.499180	57.690327	169.147622	{(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
1	0.999	0.1	0.900	0.9900	1000000.0	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,...	34.543539	591.339561	57.785679	229.698285	{(0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,...
2	0.999	0.1	0.900	0.9999	1000000.0	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	34.429235	590.560259	55.115818	589.052721	{(1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,...
3	0.999	0.1	0.999	0.9000	1000000.0	{(0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,...	34.845897	354.904240	57.361851	589.215638	{(1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,...
4	0.999	0.1	0.999	0.9900	1000000.0	{(1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,...	34.226030	620.034260	58.356334	619.817859	{(1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,...
5	0.999	0.1	0.999	0.9999	1000000.0	{(1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,...	34.078843	601.703292	56.075547	600.722693	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,...
6	0.999	0.2	0.900	0.9000	1000000.0	{(1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,...	34.845080	588.160542	57.455000	172.602694	{(0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,...
7	0.999	0.2	0.900	0.9900	1000000.0	{(1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,...	35.993122	578.281006	56.009339	558.457802	{(1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,...
8	0.999	0.2	0.900	0.9999	1000000.0	{(0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,...	38.699008	327.634622	55.782944	589.240052	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,...
9	0.999	0.2	0.999	0.9000	1000000.0	{(1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,...	39.942000	636.120355	56.184229	632.783841	{(1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,...
10	0.999	0.2	0.999	0.9900	1000000.0	{(1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,...	37.635003	646.975879	56.417087	377.500147	{(0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,...
11	0.999	0.2	0.999	0.9999	1000000.0	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,...	37.372258	613.725165	57.579895	611.069955	{(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,...





We can see also from the table, in comparison to PI and VI the Q learning policy changes, also for 25 and 625 state it takes different iterations to converge as it's non-model based learning. From the converged policy on the right policy plots we can see for 25 states there are 4 W and all cuts, whereas with 625 states there are 6 W and all other cuts, which is different from PI and VI where the W were equal for small and big state for selected epsilon gamma values (Reasoned in VI). To plot the convergence for QL, I ran QL for 2400 episodes with 1e5 iterations each, with the same convergence criteria as FL. From the Convergence plots, top 2 (25 states) and bottom 2 (625 states), we can see Mean V and Max V converges at the same value. Max V converges faster, and Mean V takes more episodes to converge.

Conclusion - Forest Management problem is a state chain problem, due to which the number of W depends on Gamma **(detailed explanation in VI section)**, First state is always W, as reward is 0 for C at stage 0 and number of iterations depend on reward epsilon gamma, instead of the size of problem for VI and PI. **For Q-Learning, the state chain problem is even harder, it converges at a suboptimal policy in comparison to PI and VI**, and it takes a lot of iterations and time to converge. Unlike the FL problem, FM problem doesn't find the policy similar to VI and PI. instead, there are Wait states in the middle, this is because the state chain nature of the problem is hard for Q learning.

REFERENCES

[1] Gym is a standard API for reinforcement learning, and a diverse collection of reference environments. Gym Documentation. (n.d.). Retrieved November 26, 2022, from https://www.gymlibrary.dev/

[2] Markov decision process (MDP) toolbox¶. Markov Decision Process (MDP) Toolbox - Python Markov Decision Process Toolbox 4.0-b4 documentation. (n.d.). Retrieved November 26, 2022, from https://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html