Markov Decision Processes

Mehmet Oguz Kazkayasi
CS7641 – Assignment 4
mkazkayasi3@gatech.edu

Abstract

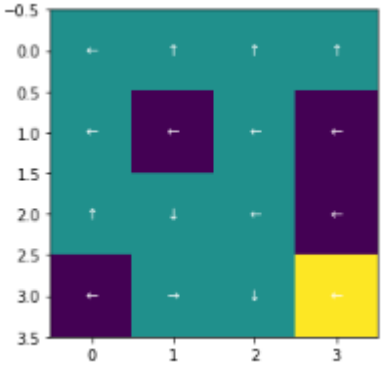
Two different MDP problems (Frozen Lake and Forest Management) in different sizes are solved using Value Iteration, Policy Iteration, and Q-Learning algorithms and the parameters of these algorithms are optimized for the best result in the given environment. The results and the behaviors of the algorithms in different environments are observed and analyzed.

1. Introduction to Environments

1.1. Frozen Lake

Frozen Lake [1] is a grid-world environment, where the agent tries to go from the left-up grid to the right-down grid. The environment has Holes, where the episode ends with 0 rewards and if the agent successfully goes to the right-down grid, the episode ends with 1 reward.

The purple color indicates holes and yellow color indicates the goal.



The agent can take actions (UP, LEFT, DOWN, RIGHT) but since the environment is stochastic, the resulting grid changes with probabilities. The agent ends up in the action's direction with 0.33 probability and ends up either side of itself w.p. 0.33 each. For example, if the agent takes action UP, it goes UP w.p. 0.33, and LEFT w.p. 0.33 and RIGHT w.p. 0.33.

I have used two different sizes for this environment to observe the effect of the state size on different algorithms. The first one is 4x4, where there are 16 states. And the second one is 16x16, where there are 256 states.

1.2. Forest Management

The Forest Management environment is a non-grid world environment, where there's a forest that grows to a maximum. In each step, the agent has options CUT and WAIT, if CUT it gets 1 reward and forest goes to state-0, where it can't be CUT. If the agent uses action WAIT, it passes to the next-state with a probability p, and goes back to state-0 with (1-p) because of a forest fire.

So, the agent has to select if it wants to risk going to the final state to get the larger rewards or get 1 reward by cutting the forest at any given state.

In the final state, if the agent selects WAIT, it earns r2 reward and w.p. p, it stays in that state. But it goes to state-0 w.p. (1-p). On the other hand, if the agent selects CUT, it earns r1 reward and definitely goes back to state-0.

2. Frozen Lake 4x4

2.1. Value Iteration

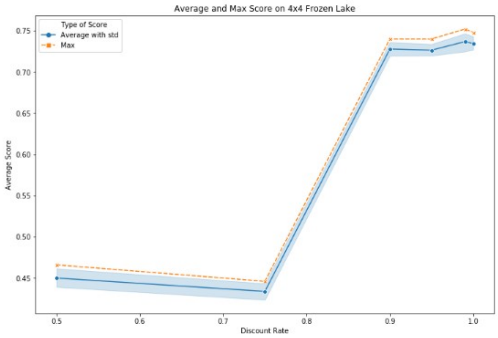
I have used Value Iteration with different discount-rates, which affects the final values of non-terminal episodes as well as the epsilon value that is used to decide on the convergence of the algorithm.

For different discount-rates, I have used 0.5, 0.75, 0.9, 0.95, 0.99, 0.9999 values and for different epsilon values, I have used 1e-3, 1e-6, 1e-9, 1e-12, 1e-15 values. After convergence, I have tested the resulting

policies in 1000 episodes and get the average reward they collect.

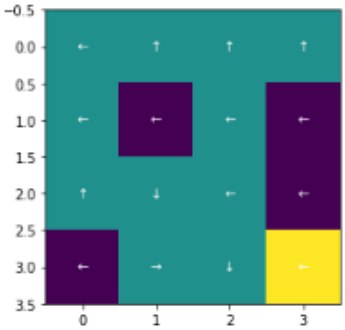
2.1.1 Discount-Rate and Its Effects on Algorithm and Policy

The first thing I've realized is the discount-rate has a very significant effect on the reward. Higher discount-rates results in better policies with better rewards.



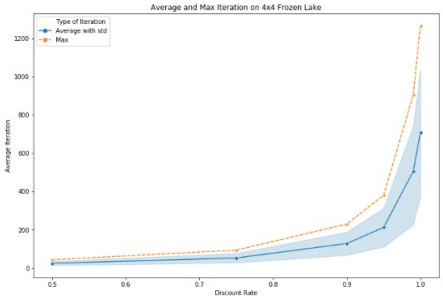
The standard deviation is caused by different epsilon values, and the orange line shows the max of those.

As it can be seen, the discount-rates over 0.9 results in over 0.7 average reward. It can be seen that since the goal far from some of the states, lower discount rates can't assign the optimal Values for those. Over 0.9 sigma, the Value Iteration does a good job of assigning optimal values and creating good policy.

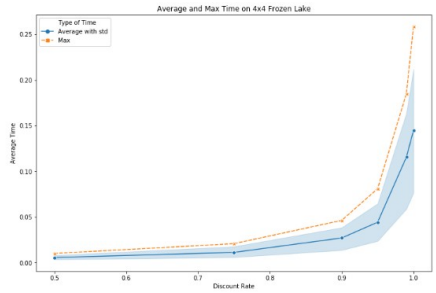


As can be seen, in this stochastic environment, the policy always tries to stay away from the holes and manages to arrive the goal at 75% of the time.

When comparing the time and iterations spent with different discount-rates



Iteration vs Discount Rate

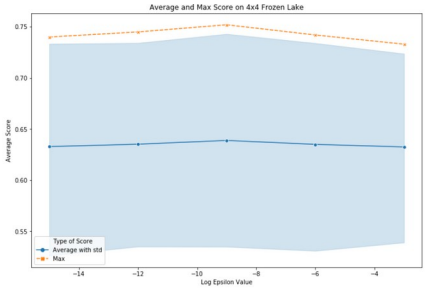


Time-Spent vs Discount Rate

As it can be seen, the increasing discount-rate affects the iteration and time spent to create the policy exponentially. Considering the positive effect on rewards on this particular experiment, it seems viable to use a higher discount-rate for better results. Also, the maximum time spent is still 0.25 seconds with around 1200 iterations. So, for a 16 state environment, it makes sense to use a high discount rate.

2.1.2 Epsilon Value and Its Effects on Algorithm and Policy

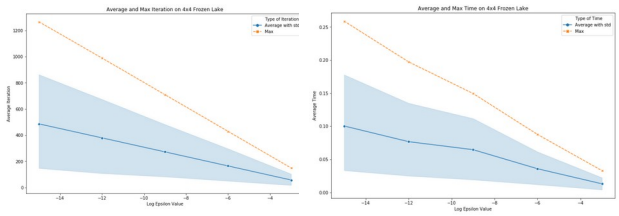
After observing the effects of the epsilon-value, it can be seen that for Value Iteration, discount-values dominates the reward of testing so that the effect of Epsilon value is difficult to observe.



At x-axis, log-values are used for epsilon

As it can be seen, different discount-rate causes a large standard deviation and 1e-0 epsilon value gets

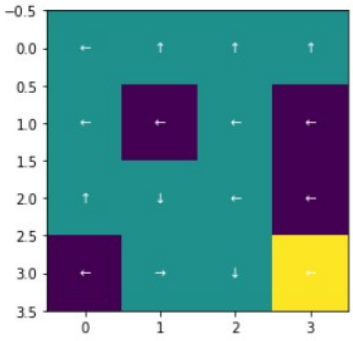
the max (seems the effect of randomization) reward. So, it can be observed that if a small enough epsilon value is used, it will be enough to get a good policy.



As it can be seen from iterations and time spent graphs for convergence in different epsilon values, the log(epsilon) value affects the time significantly. 1e-9, which gets the max reward, has a mean time close to 0.5, whereas 1e-15 takes twice as long time. For iterations, it is a similar case. Even though these time values (0.5sec vs 0.1sec) are so low to discuss. This would make difference in larger MDPs.

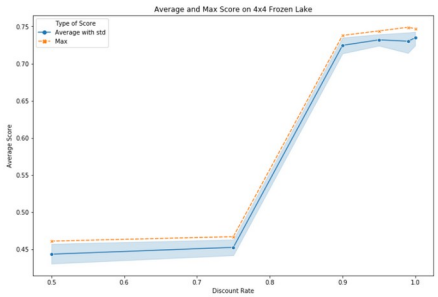
2.2. Policy Iteration

I have used the Policy Iteration from Sutton's Reinforcement Learning [2] and I have used different epsilon values for the 2nd step (Policy Iteration). So, I had two different hyper-parameter: Discount-rate and Epsilon. I have used the same values as I used in Value Iteration and as a result, I get the exact same policy in the better end of hyper-parameters.

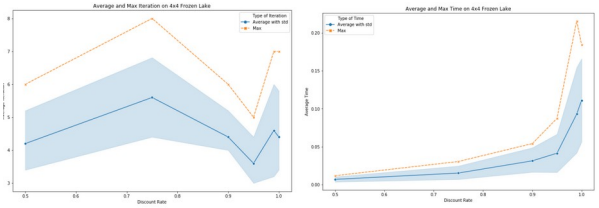


2.2.1 Discount-Rate and Its Effects on Algorithm and Policy

As the resulting policy is the same, the reward is similar to Value Iteration.



Over 0.9 discount-rate gets over 0.7 average reward, whereas under 0.75 is around 0.45. I have checked the time spent and iterations to compare with VI.



As it can be seen in the left-graph, the number of iterations for PI is much lower comparing to VI. And the discount-rate doesn't affect it significantly. The max of iterations is 8, whereas VI converges over 1200 at its max.

On the other hand, it can be seen that discount-rate affects the time spent exponentially. But the time spent in general is below (close to it) VI, too. The max of VI takes 0.25 sec, and max of PI takes around 0.22. In general, PI takes close but a little less time than VI.

2.2.2 Epsilon Value and Its Effects on Algorithm and Policy

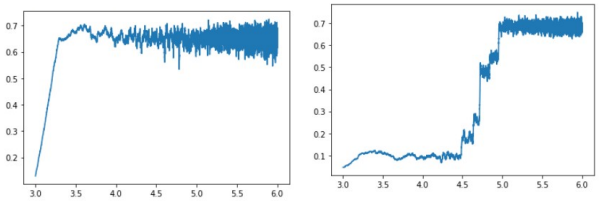
The epsilon value, similar to VI, has less effect on the policy, but some more effect on time-spent. As they don't indicate anything but randomness caused by discount-rate (and for space-efficiency), I am not adding the graphs for epsilon value. As long as a small enough epsilon value is used, it can be said that the only significant effect of epsilon value is on the time spent. Lower epsilon values make PI longer to converge, but this doesn't affect the iteration number (because the iteration is count when policy is improved) or the average reward the resulting policy gets.

2.3. Q-Learning

At the Q-Learning algorithm, I have explored some of the hyper-parameters and used constant values for others after observing they are less effective. I have used different values for training episodes, discount-rate, and learning-rate.

I have used a linear decay with a minimum on epsilon value for epsilon-greedy action selection. I have used only 1 discount-rate (0.9999) to tune these variables at 4x4 Frozen Lake. Also, I have used two different constant learning rate to see its effect on convergence.

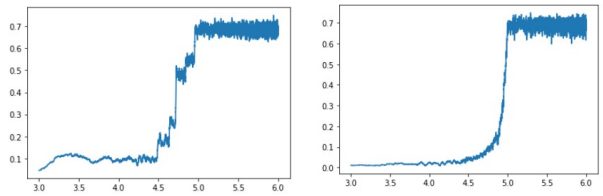
I have drawn the rolling mean of reward in training phase to see the convergence. To analyze the effect of learning rate, I have drawn two graphs: 0.1 learning-rate on left and 0.01 learning-rate on right.



(x-axis is log normalized 6.0 means 1,000,000)

As it can be seen from the graphs, higher learning rate helped convergence faster but started to oscillate more in the end because each actions keep affecting the Q-table significantly. On the other hand, a lower learning rate caused a late convergence but we see little oscillation. So, using a decay on the learning-rate (like we use on epsilon) makes perfect sense to make a fast convergence with little oscillation.

Next, I have analyzed the effect of decay-rate on epsilon-value. I have used two different decay-rates with linear decay on epsilon. I have used the learning rate 0.01 and two different decay-rates (1e-3 and 1e-5) to see the difference. On the left graph, 1e-3 and the right-graph: 1e-5 decay-rate is used.



Since the x-axis is log10 normalized, we can't see it in great detail but, when checked carefully it can be seen that the higher decay rate causes a slightly earlier (50k vs 100k when linearized back) convergence to a better policy. But, they both reach to their optimal, which is equal, at 100k.

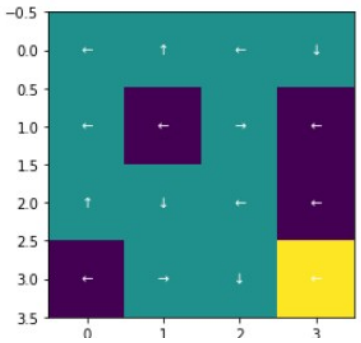
The decay-rate affects the exploration-exploitation balance. In this environment, I think because the stochasticity is too high, the decay-rate didn't make so much difference. But, in other environments, I think it'll be as important as learning rate.

Since there are multiple hyper-parameters in Q-Learning, I used this table instead of graphs to analyze their effects.

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.747	8.363291
1	0.9999	10000.0	0.10	0.00001	0.726	2.830679
2	0.9999	10000.0	0.01	0.00100	0.108	2.436700
3	0.9999	10000.0	0.01	0.00001	0.290	2.924997
4	0.9999	100000.0	0.10	0.00100	0.591	104.188003
5	0.9999	100000.0	0.10	0.00001	0.546	53.979627
6	0.9999	100000.0	0.01	0.00100	0.205	46.713976
7	0.9999	100000.0	0.01	0.00001	0.747	48.768106
8	0.9999	1000000.0	0.10	0.00100	0.747	1120.504023
9	0.9999	1000000.0	0.10	0.00001	0.641	1013.372449
10	0.9999	1000000.0	0.01	0.00100	0.747	920.123316
11	0.9999	1000000.0	0.01	0.00001	0.731	988.899270

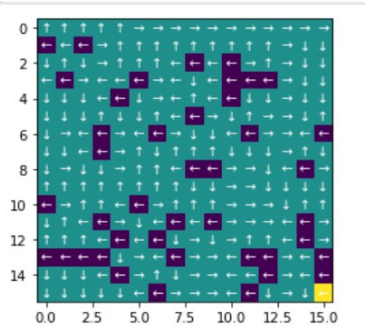
As it can be seen, time-wise the training episode has the highest effect and comparing to PI/VI, Q-Learning takes a lot more time. This is because it cannot use the model that constructs the environment and it uses an agent to explore the environment and understand the underlying model. When we see the mean-reward the policies get, we see that more episodes help a lot but there are optimal policies that get over 0.74 reward at fewer episodes, too. This is with help of higher learning rate of course but there is a stochastic nature inside q-learning algorithm, as well. It is possible to reach the best policy with a non-optimal setup and vice-versa.

When checked the policy created by Q-Learning, we see that it is the almost same with PI/VI algorithms and its testing value is same with PI/VI so it manages to reach the optimal in 4x4 world.



3. Frozen Lake 16x16

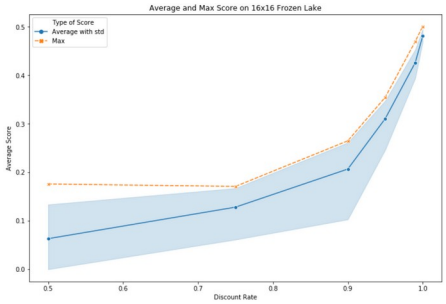
Since the effects of different hyper-parameters are discussed in 2nd section, the differences in the larger grid-world environment will be analyzed in this section.



First of all, it can be said that, because a longer path should be taken by the agent, it is expected to have a lower reward on average.

3.1. Value Iteration

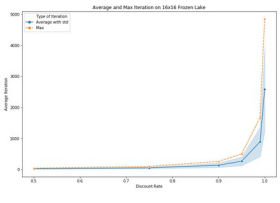
The first metric, I have analyzed is the average reward, which indicates how good the policy produced is.

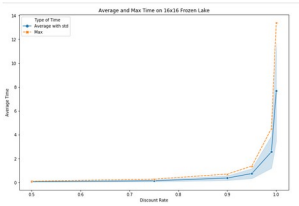


As it can be seen, the average reward is now around 0.5 at max. A significant difference between 4x4 and 16x16 is that now 0.9 discount-rate is not among the optimum values. In 4x4, over 0.9 discount-rate was enough to get the best policy. Now, it can be seen that 0.9999 makes a big difference comparing to 0.99.

This proves that larger discount-rates results in better Values for states and as the environment gets larger, we need larger values.

As the time-spent and iterations for 16x16, it can be seen that higher discount-rate affects more than it does 4x4.

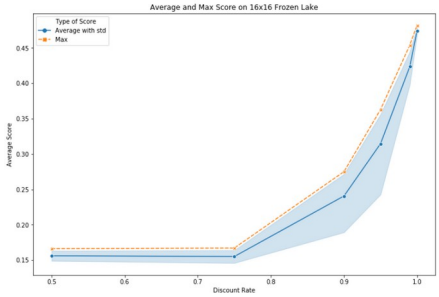




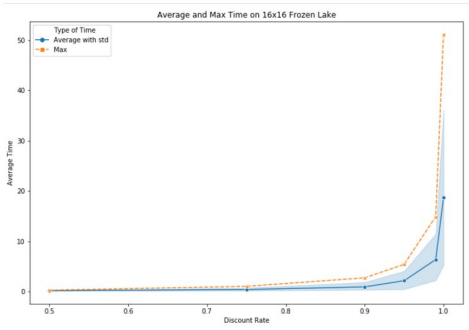
As it can be seen, even though the graphs have similar shapes, the values are quite different. The max time spent for 16x16 is 14 seconds, where it was 0.25 seconds for 4x4. This shows that larger environments needs much larger convergence time. Also, the iterations needed increases from 1200 to 5000 (at max) as well.

3.2. Policy Iteration

Even though I expected a different policy from PI this time because of the larger environment, PI came up with the same policy for 0.9999 discount-rate and 1e-15 epsilon-value for both VI and PI.



As it can be seen in the Discount-Rate vs Average-Score graph, PI has slightly better results for lower discount-rates comparing to VI. But, it is obvious that larger environments need higher discount-rate for PI, as well.



On the other hand, the time-spent for convergence has shown a different result. It takes PI much longer to converge on a larger environment even though the iteration numbers are lower again. That's because it

has so many states to evaluate in each iterations. And evaluation takes multiple iterations.

In the end, both VI and PI came up with similar rewards and same policy. But, PI spent much longer time comparing to VI on a the larger environment.

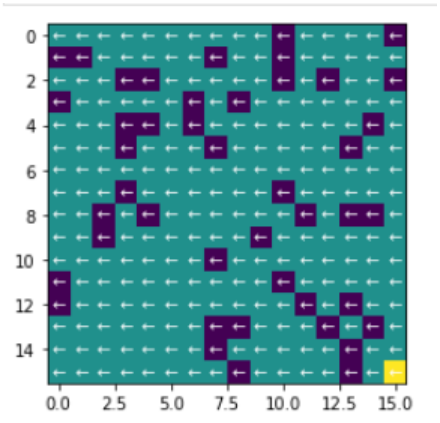
3.3. Q-Learning

For Q-Learning, exploring the 16x16 world will be a challenge since the episode will end up in any Hole (20% of grid).

I have used the same setup with up to 1M training episodes and as a result, I got this table from different hyper-parameters.

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.0	0.710954
1	0.9999	10000.0	0.10	0.00001	0.0	1.609952
2	0.9999	10000.0	0.01	0.00100	0.0	0.851520
3	0.9999	10000.0	0.01	0.00001	0.0	1.773063
4	0.9999	100000.0	0.10	0.00100	0.0	6.428837
5	0.9999	100000.0	0.10	0.00001	0.0	12.025417
6	0.9999	100000.0	0.01	0.00100	0.0	7.981473
7	0.9999	100000.0	0.01	0.00001	0.0	11.076562
8	0.9999	1000000.0	0.10	0.00100	0.0	62.562974
9	0.9999	1000000.0	0.10	0.00001	0.0	67.106997
10	0.9999	1000000.0	0.01	0.00100	0.0	66.184676
11	0.9999	1000000.0	0.01	0.00001	0.0	73.556381

The first thing to realize is that there is no reward over 0. And the time-spent is much lower comparing to Q-Leaning on 4x4 world. When checked the q-tables created by the algorithms, I saw that there's no value over 0. This means that in any of the training events above, there is not a single agent arrived in the Final episode to end the game with reward 1. All the agents that tried to explore the field ended up in holes.



When checked the environment, it can be seen that it is pretty difficult to reach the end using random actions. Since there's no agent reached before, the q-table is always 0 and all actions are random. This is a pretty big disadvantage comparing to PI/VI, which doesn't need to end-up in the Goal with random actions.

So, we can say that using Q-Learning in a large environment with little rewards around (there's only 1 reward in this environment) is not effective. And as the environment has a model, using model-based algorithms such as VI/PI is a better choice.

One thing to solve this problem would be to change the rewards such that staying alive has a very little point, so that Q-Learning would try to stay away from the Holes and randomness would take it to the Goal finally. And the q-table would get a much better shape. Or, we could add -1 reward to all of the Holes, so Q-learning would learn to stay away from them but since this would be changing the environment, I stayed away from this approach.

Also, another solution would be starting the agent from random non-terminal grids instead of starting at state-0. This way, more states would get positive points by reaching the Goal state and this would help solving the environment.

4. Forest Management (20 states)

In this MDP, the discount-rate has to be a design decision instead of a hyper-parameter for the algorithms. That's because it is not an environment with only one reward, which is a terminal node. In this environment, the agent can get multiple rewards buy not cutting in the final state or may decide to cut depending on the discount-rate. By changing the discount-rate, we will deal with different environments. And the optimal policy will be different. So, I have used 0.9 discount rate for all the algorithms.

The testing of this environment is more complicated comparing to Frozen Lake, too. This is because if we start from state-0 all the time, we will end up in the first CUT decision and can't test the rest of the policy. So, in testing environment, I have started the agent from each state 1000 times and get the average reward in the end. Also, since the discount-rate is set as an environment parameter, I have used the discount-rate for the rewards, too (unlike Frozen Lake). So, in the testing environment, if I keep getting the same reward, the second reward is 0.9 and the third is 0.81 of the first reward.

As rewards, I have used r1=6, which means keeping the forest in the final state has a reward of 6 and r2=10, which means cutting the forest in the final state has a reward of 10.

4.1. Value Iteration

In Value Iteration, I have used different epsilon values and a constant discount-value as explained above. Since there are so many rewards in different state, action pairs in this environment, for 20-state, having a smaller epsilon value didn't affect the Policy as all of the epsilon values tried were small enough to get the optimum policy. I have tried (0.1, 0.01, 1e-6, 1e-9, 1e-12, and 1e-15) epsilon values and get this table in the end.

	Epsilon	Policy	Iteration	Time	Reward
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	31	0.002649	2.154275
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	52	0.002633	2.156900
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	85	0.005243	2.142954
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	118	0.006456	2.152037
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	151	0.007815	2.185030
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	186	0.010476	2.113817

As it can be seen, all the policies returned are the same and the reward/episode they get when started in a random state are close to each other. On the other hand, smaller epsilon values causes more Iterations and thus longer times, as expected. Since the MDP is pretty simple in the Forest Management environment and there are so many rewards, unlike Frozen Lake, VI did a pretty quick job and returned optimal policies at any epsilon value.

4.2. Policy Iteration

In policy iteration, since I have used mdptoolbox's [3] implementation in this problem, I couldn't test for different epsilon values and got only 1 run using the constant discount-rate.

Policy Iteration gave the same policy (so similar mean-reward) with any of the Value Iterations and did this in 12 iterations and 0.03 seconds. As it can be seen, Policy Iteration took the longest time comparing any of the VI above.

For smaller Frozen Lake environment, PI took close (a bit shorter) time comparing to VI. And it solved Frozen Lake in 8 iterations (4x4) at most. But in this environment, it took 12 iterations. That is because in this sort of chain set-up, PI changes a smaller part of the policy each iteration instead of correcting many at the Grid World set-up.

4.3. Q-Learning

Since Q-Learning uses an agent to explore the environment, this type of chained-state environments seem difficult for it. So, I have explored 5 different hyper-parameters for it to converge to the optimum state and checked all their mean-rewards and time to complete the training.

I've explored "number of iterations", "epsilon", "epsilon-decay", "learning-rate", and "learning-rate-decay" hyper-parameters. The results are as in the table below.

]:

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	2.112963	43.618149
1	1000000	0.990	0.0001	10.0	0.990	2.114554	43.677410
2	1000000	0.999	0.0010	10.0	0.990	2.123520	43.546746
3	1000000	0.999	0.0001	10.0	0.990	2.088811	43.096997
4	1000000	0.990	0.0010	10.0	0.999	2.140274	43.282926
5	1000000	0.990	0.0001	10.0	0.999	2.149241	43.344919
6	1000000	0.999	0.0010	10.0	0.999	2.111441	43.098411
7	1000000	0.999	0.0001	10.0	0.999	2.125839	43.276635
8	1000000	0.990	0.0010	1.0	0.990	2.149132	43.359320
9	1000000	0.990	0.0001	1.0	0.990	2.152381	44.149231
10	1000000	0.999	0.0010	1.0	0.990	2.135872	44.541396
11	1000000	0.999	0.0001	1.0	0.990	2.170419	44.624184
12	1000000	0.990	0.0010	1.0	0.999	2.188515	42.555094
13	1000000	0.990	0.0001	1.0	0.999	2.134553	43.361591
14	1000000	0.999	0.0010	1.0	0.999	2.136003	45.035051
15	1000000	0.999	0.0001	1.0	0.999	2.152113	44.596225
16	10000000	0.990	0.0010	10.0	0.990	2.089561	437.497844
17	10000000	0.990	0.0001	10.0	0.990	2.132088	464.760277
18	10000000	0.999	0.0010	10.0	0.990	2.133040	470.177954
19	10000000	0.999	0.0001	10.0	0.990	2.093046	477.440789
20	10000000	0.990	0.0010	10.0	0.999	2.154250	482.488369
21	10000000	0.990	0.0001	10.0	0.999	2.112108	472.839738
22	10000000	0.999	0.0010	10.0	0.999	2.114887	512.676572
23	10000000	0.999	0.0001	10.0	0.999	2.155651	578.868911
24	10000000	0.990	0.0010	1.0	0.990	2.163981	579.066845
25	10000000	0.990	0.0001	1.0	0.990	2.105347	554.939058
26	10000000	0.999	0.0010	1.0	0.990	2.129672	533.451718
27	10000000	0.999	0.0001	1.0	0.990	2.134965	495.792148
28	10000000	0.990	0.0010	1.0	0.999	2.095265	498.717053
29	10000000	0.990	0.0001	1.0	0.999	2.121986	490.025702
30	10000000	0.999	0.0010	1.0	0.999	2.156720	507.424240
31	10000000	0.999	0.0001	1.0	0.999	2.151777	585.571571

As it can be seen, all the resulting policies get close results to each other but when checked, I realized that the policies are quite different than each other even though much of them are similar. Also, none of the 32 policies are equal to VI/PI policies (which are same).

On the other hand, the time spent for the Q-learning is much longer. It can take up to 10 minutes, whereas VI/PI takes 0.03 seconds at most.

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.136602	43.697768
10000000	0.9945	0.00055	5.5	0.9945	2.127772	508.858674

After checking the mean values for 2 different iteration value, I saw that the reward didn't increase from 1M to 10M iterations, which means the algorithm already converged in 1M, but time has increased by 10x, as expected.

I have checked the mean values of other hyper-parameters, as well. But, the results are so similar to each other on both reward and time. This means that no matter what hyper-parameter I have used, it has converged to a good enough policy.

5. Forest Management (500 states)

To analyze differences between a small and larger non-grid problems, I have used 500 state Forest Management environment. In this environment, I kept using 0.99 discount-rate but since it is a very long path to reach to final state to collect the rewards, I increased r1=100 (cut at final state) and r2=15 (keep at final state).

5.1. Value Iteration

In Value Iteration, I have used the same epsilon values with 20-state Forest Management problem (0.1, 0.01, 1e-6, 1e-9, 1e-12, and 1e-15).

	Epsilon	Policy	Iteration	Time	Reward
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	79	0.069786	2.744159
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	119	0.037132	2.738842
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	179	0.199475	2.736443
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	239	0.058018	2.726877
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	299	0.387262	2.749473
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	348	0.363754	2.724796

It can be seen that the mean-reward that policies extract from the environment are pretty close to each other in this environment, too. This means that 0.1 epsilon value is small enough to get the best policy. But, the number of iteration is doubled comparing to 20-states, which is pretty reasonable and the time spent is multiplied with 36 for 500-states.

I think these numbers are pretty good from 20 to 500, we can say the time is almost linear with the number of states. That means VI works pretty well in larger environments, too. This was seen in the Frozen Lake environment, as well. VI didn't increase its time so drastically, comparing to PI, with regard to the state size.

5.2. Policy Iteration

The Policy Iteration algorithm created a policy with 2.72 mean-reward, which is so close to what VI creates. But, the chain property increased PI's iteration and time-spent in 500-state environment, as well. It took 46 iterations and 3.1 seconds to converge for PI, which is 10 times longer than what VI took with its most extreme epsilon value. The underlying reason is again the fact that policies are dependent to each other. When next state's policy changes, then it matters more for the current state change or not. This increases the iteration and the time.

As it was seen in Frozen Lake environment, the time and iteration of PI increases much more with regard to the size of environment, comparing to VI. Also, the chained-state environment makes things more difficult for PI. But, in the end it is able to get the optimum policy in 3 seconds in a 500-state environment.

5.3. Q-Learning

Forest-Management with 500 states seems to be a difficult problem for Q-Leaning to solve. But, the biggest advantage here is that the agent gets some reward in almost every state. So that, it will have more guidance.

I have used the same 5 hyper-parameters and got this table from Q-Learning on Forest Management.

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward
0	1000000	0.990	0.0010	10.0	0.990	0.802000
1	1000000	0.990	0.0001	10.0	0.990	2.577495
2	1000000	0.999	0.0010	10.0	0.990	2.631859
3	1000000	0.999	0.0001	10.0	0.990	2.646176
4	1000000	0.990	0.0010	10.0	0.999	2.539574
5	1000000	0.990	0.0001	10.0	0.999	2.606902
6	1000000	0.999	0.0010	10.0	0.999	2.569366
7	1000000	0.999	0.0001	10.0	0.999	2.637456
8	1000000	0.990	0.0010	1.0	0.990	2.670379
9	1000000	0.990	0.0001	1.0	0.990	2.690222
10	1000000	0.999	0.0010	1.0	0.990	2.619189
11	1000000	0.999	0.0001	1.0	0.990	2.651154
12	1000000	0.990	0.0010	1.0	0.999	0.820000
13	1000000	0.990	0.0001	1.0	0.999	2.679801
14	1000000	0.999	0.0010	1.0	0.999	2.650142
15	1000000	0.999	0.0001	1.0	0.999	2.707013
16	10000000	0.990	0.0010	10.0	0.990	2.723851
17	10000000	0.990	0.0001	10.0	0.990	1.026000
18	10000000	0.999	0.0010	10.0	0.990	2.756234
19	10000000	0.999	0.0001	10.0	0.990	2.818315
20	10000000	0.990	0.0010	10.0	0.999	2.707936
21	10000000	0.990	0.0001	10.0	0.999	2.835022
22	10000000	0.999	0.0010	10.0	0.999	2.739897
23	10000000	0.999	0.0001	10.0	0.999	2.788664
24	10000000	0.990	0.0010	1.0	0.990	2.802476
25	10000000	0.990	0.0001	1.0	0.990	2.865915
26	10000000	0.999	0.0010	1.0	0.990	2.651490
27	10000000	0.999	0.0001	1.0	0.990	2.761586
28	10000000	0.990	0.0010	1.0	0.999	2.762742
29	10000000	0.990	0.0001	1.0	0.999	2.851094
30	10000000	0.999	0.0010	1.0	0.999	2.764619
31	10000000	0.999	0.0001	1.0	0.999	2.812812

In this environment, it can be seen that different hyper-parameters affected the policy and the resulting reward significantly. Some of the policies could get less than 1.0 mean reward, whereas some of them surpassed the VI/PI algorithms. When analyzing independently, grouping by different hyper-parameters:

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.406171	57.976636
10000000	0.9945	0.00055	5.5	0.9945	2.666791	513.044849

It can be seen that 10M iteration gets significantly better result, unlike 20-state Forest Management. So, a larger state space needs more training. This makes sense because Q-learning learns by interacting the environment. From this table, we can say that Q-learning needs more than 1M iterations to converge in this environment.

But, the most significant effect is caused by "learning rate decay" hyper-parameter:

Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0.990	0.00055	5.5	0.9945	2.372588	286.648156
0.999	0.00055	5.5	0.9945	2.700373	284.373330

A higher decay, which causes a slower decrease in the learning rate results in a much better policy. This can be explained by the same fact. It takes longer time to explore an environment with more states. If learning-rate drops too much before the agent reaches to certain states enough times, the q-table can't be updated with a large enough learning rate.

Comparing the q-learning to VI/PI in this environment, we see that it never comes up with the same policy with them. But, it gets good enough policies that get over 2.8 reward, whereas PI/VI can get around 2.74.

6. Conclusion

Two different environment and three different algorithms are used in this analysis. Two of the algorithms were model-based (PI and VI) and they had a big advantage to produce the best policies since they interact with the model directly. This is supported by the results on both mean-reward their policies got, and the time-spent for training.

Q-Learning, a model-free algorithm, has to interact with the environment instead of interacting with the model. So that, it is more difficult to explore the environment for this algorithm. We have witnessed this in Frozen-Lake 16x16, where Q-Learning didn't even reach the Goal state in 10M training episodes. So that, it got no value in the q-table. This can be improved by different techniques as explained in that section but its disadvantage is obvious.

When comparing the grid vs non-grid world environments, we can see that grid world is easier to converge for Policy Iteration because of the Forest Management environment's chained-state nature.

On the other hand, we have seen that increasing the state size doesn't affect the model-based algorithms significantly. They can still solve the problem in reasonable iterations and they reach the optimal. But, this case is not true for Q-learning.

8. References

[1] Gym: A toolkit for developing and comparing reinforcement learning algorithms. (2020). Retrieved 11 April 2020, from https://gym.openai.com/envs/FrozenLake-v0/

[2] Sutton, R., & Barto, A. (2018). Reinforcement learning (2nd ed.). London, England.

[3] Sawcordwell. (2015, April 13). sawcordwell/ pymdptoolbox. Retrieved from https://github.com/ sawcordwell/pymdptoolbox/