Assignment 4

Joe Kraemer

I. Environments

A. Forest

In this environment, there is a fictitious forest which grows with each timestep in the environment. The state is an integer defined as the age of the tree. In each step, the agent can choose to cut down the forest, or wait. If the agent chooses to wait, the forest has a chance to burn down ($p_{fire} = 0.1$) and returns to the start state s=0.

The agent is rewarded with a small reward if they cut down the forest if it is not in its youngest state (s =1). There is a medium reward if the forest is cut in its oldest possible state (the maximum number of states in the problem) and the agent is given a large reward if it waits until the forest is in its oldest state.

This problem is difficult because the frequent small rewards create local optima at every state other than the start state.

One possible policy would be to never cut and collect the large final reward. This policy would be difficult for model free agents such as Q-Learning because the agent would select many non-greedy actions in a row to arrive at the final state. Another possible policy would be to alternate cutting and waiting. These two policies could be better than one another depending on if the chance of fire is high.

B. Frozen Lake

In this environment, the agent moves around in a 2D grid where the possible actions are move up, down, left, and right. The goal is to arrive at the goal square. The environment has holes in it that will immediately end the episode and return no reward. The environment is stochastic and so each of the actions has a possibility of moving the agent 90 degrees off of the attempted action. For example, moving left would instead have a possibility of moving up or down.

Because the final reward in the base environment is so sparse (there is only one state with an action), I added some reward shaping to speed up the learning process. I added a small negative reward to the holes [1].

Even with the reward shaping, the holes will still be a difficult part for model free agents as they will have to restart the episode after a false step into the hole. This will require many more iterations to observe the state space and discover the only positive reward.

Furthermore, the large state space will be difficult for model free agents because there will be many states to explore and the final goal state is far from the start state.

II. Forest Management

In the Forest Management environment I expect that policy iteration will be very quick to come up with an optimal policy. I expect that value iteration will be slower because it will take longer to propagate the value of each subsequent state. I don't expect that Q-Learning will find the same policies as PI and VI. This is because the exploration-exploitation trade off will make it difficult for the agent to progress through the state space to the final state where the large reward is. Instead it will likely chop the forest down early.

A. Value Iteration

Value Iteration convergence is achieved when the error between the current and previous Value Matrix is sufficiently small. When selecting a value for convergence (ϵ), its important to consider what environment parameters could change the policies. In this state space, there is only two actions, and I hypothesized that the more valuable of the two actions for each state would be clear quite quickly. The most difficult policy to find would be to wait until the final state because this would require the most value iterations. To test this hypothesis, I set the convergence criteria ϵ infinitely small, and set the chance of forest fire to be small as well so that for all state space sizes, the wait forever policy would be the global optima. I checked when the policy arrived at its final form and ceased to change. I tested multiple different state space sizes and I found that the final policy change that happened latest was around 0.1 error for s=100. I set the convergence error to be defined as 0.01 error to add some additional confidence that the policy is indeed stable.

In Figure 1, we can see that the error initially drops incredibly sharply. This is because the previous iterations value is multiplied by the discount rate and then the reward is added. This leads to a logarithmic decay with the limit approaching R(s)/(1-(γ*P(s,a)).

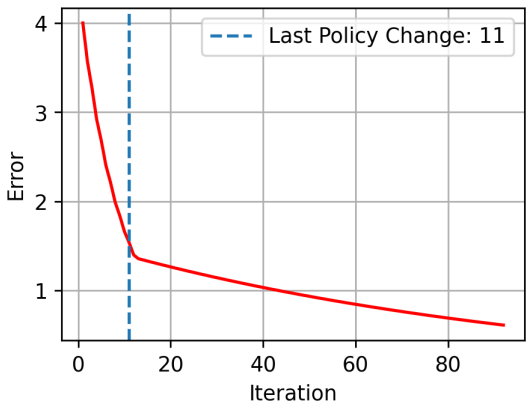

Fig. 1: Error per iteration of Value Iteration ($\gamma = 0.99, \epsilon = 0.01$) algorithm in the Forest Management Environment ($size = 10, p_{fire} = 0.1$).

B. Policy Iteration

Policy Iteration (PI) convergence is achieved when the new policy is identical to the previous policy. In other words, for

every state in the new policy, the old policy matches. This is similar to the Value Iteration definition, except we are not tracking the underlying Values of the state so instead it makes more sense to define convergence based on the policy since this is what we are iteratively updating.

In Figure 2, we can see that the error does not have the same taper as in VI. This is partially due to the fact that we don't continue to reduce the error in the value function after the policy has stopped changing. The other reason is that Policy Iteration is able to propagate changes faster because many states are updated instead of just neighboring states like in Value Iteration.

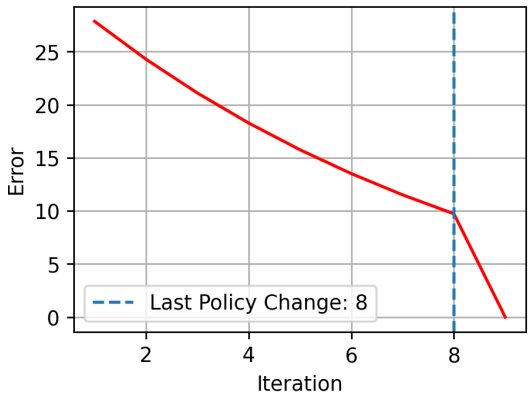

Fig. 2: Error per iteration of Policy Iteration ($\gamma = 0.99$) algorithm in the Forest Management Environment ($size = 10, p_{fire} = 0.1$).

We can also see that VI takes more iterations than PI to reach the final policy, but continues for many more iterations. Despite the higher number of iterations, Value Iteration finished in less wall clock time than Policy Iteration (Table II). This is likely because solving the large set of linear equations is actually quite costly. This highlights an advantage of PI which is that it takes much less iterations. However, if the cost of each iteration is low, then Value Iteration makes more sense. In the Forest Management case, both PI and VI have the true transition and reward functions. If the agents did not have the transition and reward functions, and instead were required to use an environment that is costly to interact with to gather experience, then the advantage of lower number of iterations for convergence of Policy Iteration would be much more important.

In Table I, we can see that Value Iteration and Policy Iteration both converged to an identical policy. This policy is to never chop down the forest and instead wait the entire time. Both VI and PI were able to identify the globally optimal policy despite there being local minima. We expected VI and PI to converge to the same policy despite taking differing iterations.

C. Q-Learning

Q-Learning convergence is defined in a similar way to Value Iteration where convergence is achieved when the error between the current and previous Value Matrix is sufficiently small. I chose this value by observing several different error plots for several different state space sizes and several different $\epsilon - decay$ values and looking to see where the last policy change was. I noticed that most policy changes were complete by the time the error was around 0.02. I ended up choosing 0.01 to have some additional buffer room.

Algorithm	Policy
Value Iteration	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Policy Iteration	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
QLearning	[0, 1, 1, 1, 1, 1, 0, 1, 1, 1]

TABLE I: Table of Final Policies for various Reinforcement Learning algorithms in the Forest Management environment. The policy is a list where each number represents the action for a given state where the state is the index in the list.

I expect that Q-Learning will struggle to reach the global optima in this state space. Given that this agent is using an epsilon greedy exploration method, it will be difficult for it to ignore the small reward for chopping the forest while it is too young. To find the global optima would require that the random action selected is to wait n-1 times in a row.

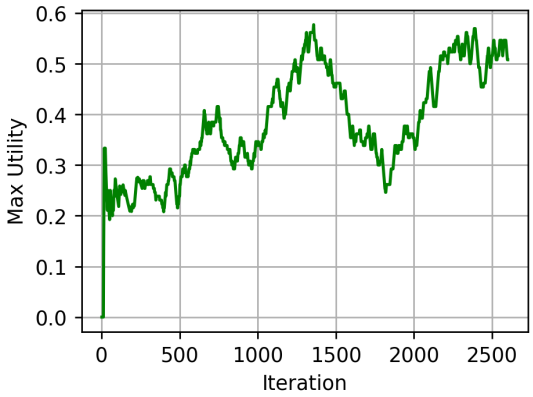

Fig. 3: Maximum Utility of the Q function table of Q-Learning ($\gamma = 0.99, \epsilon - decay = 0.999$) algorithm in the Forest Management Environment ($size = 10, p_{fire} = 0.1$).

As I hypothesized, the Q-Learning agent is never able to achieve a reward higher than the small reward for chopping the young forest as we can see the maximum utility of the Q function never goes above 1 (Figure 3). This is largely due to the difficulty of making so many consecutive non-greedy actions.

In Figure 4, we can see that the initial error is small. This is because the Q function is not changing because the agent hasn't had experiences with positive rewards to update the function. This is in contrast to the VI and PI which were able to immediately update their utility functions.

Looking at Table II, we can see that the Q-Learner took substantially more time to train, but still failed to find the global optima. Additionally, the Q-Learner took thousands of iterations while VI and PI took less than 100 iterations.

An important part of using model free algorithms is balancing exploration vs exploitation. Model free algorithms need to have varied interactions with the environment to build a

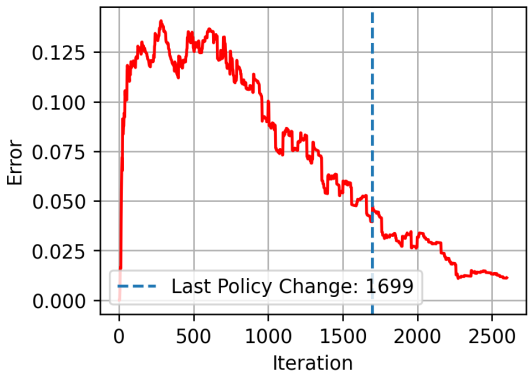

Fig. 4: Error per iteration of Q-Learning ($\gamma = 0.99, \epsilon - decay = 0.999$) algorithm in the Forest Management Environment ($size = 10, p_{fire} = 0.1$).

utility function, but at the same time they need to take optimal actions to generate higher rewards. In Figure 5, we can see that the lower values of of epsilon decay lead to higher average rewards, but more volatile curves. The agent is able to initially wait instead of taking the optimal policy. The higher epsilon decay values mean that the agent is taking too many random actions and is not optimizing the policy fast enough before the alpha decays too much. This shows that there is a large interdependence on exploration parameters.

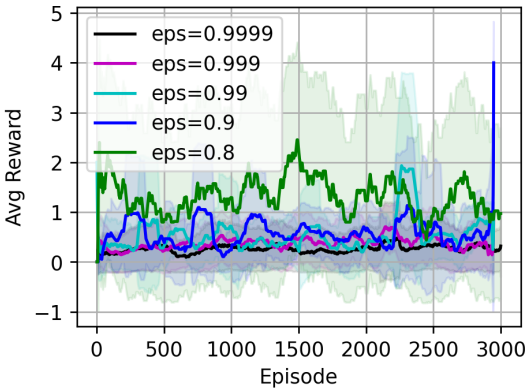

Fig. 5: Average Reward per iteration of Q-Learning ($\gamma = 0.99$) algorithm with various $\epsilon - decay$ values in the Forest Management Environment ($size = 10, p_{fire} = 0.1$).

D. Comparative Analysis

In this problem set, Q-Learning stands far behind. Given the long wall clock time and high iterations, it has a large penalty with environment interactions. Additionally, given that it was not able to converge to the global optima, this algorithm is the worst of the three.

Looking at VI and PI, I think that they both have strong points. In this environment, Value Iteration might have a slight edge because of the lower wall clock time, but this is almost negligible. PI is likely a better choice given that it can complete in less iterations.

Looking at II, we can see that the large state space causes Policy iteration and Q-learning to take much longer to compute. This is a huge disadvantage for PI and if an application is sensitive to wall clock time, then VI would be preferred.

III. Frozen Lake

A. Value Iteration

As previously mentioned, Value Iteration convergence is achieved when the error between the current and previous Value Matrix is sufficiently small. I went through a similar procedure of looking at the error curves with the annotated final policy change line of multiple state space sizes and ended up deciding on a convergence delta of 0.001.

In Figure 7, we can see that the error initially decreases quickly but then decays. This curve is much more aggressive than the curve of the Forest Management. This is because there are many more states and so it requires more iterations to propagate the error throughout the value matrix.

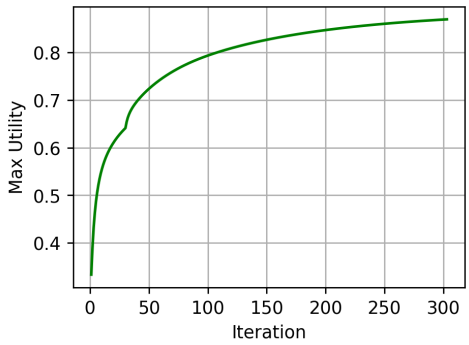

Fig. 6: Maximum Utility of the Value function table of Value Iteration ($\gamma = 0.999$) algorithm in the Frozen Lake Environment ($size = 16$)

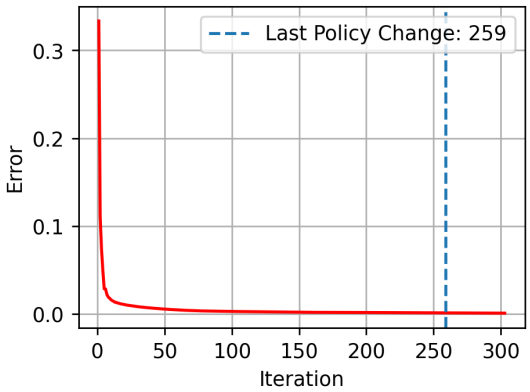

Fig. 7: Error per iteration of Value Iteration ($\gamma = 0.999$) algorithm in the Frozen Lake Environment ($size = 16$)

In Figure 8, we can see the final policy for value iteration. We can see that algorithm chooses to follow the left wall until the first hole, then it moves by slipping its way through the holes so that it does not accidentally fall into the hole. This takes more iterations because it has to get lucky, but its a safer

option than moving straight through the gap which could lead it to slip into the hole.

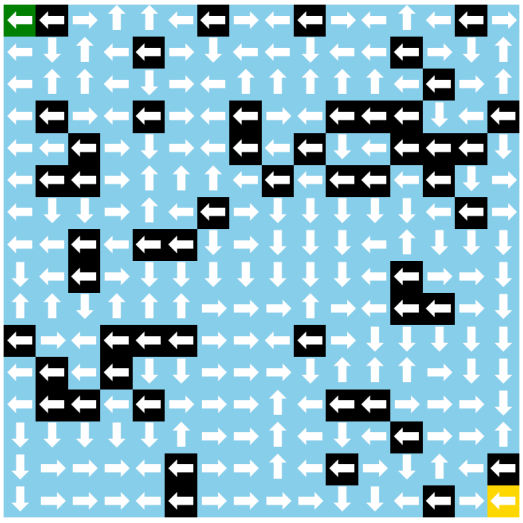

Fig. 8: Final Policy for Value Iteration in the Frozen Lake environment. Black squares represent holes, the yellow square represents the goal state, blue squares are the frozen lake and the green square is the starting state.

B. Policy Iteration

When setting the convergence behavior of Policy Iteration in the Frozen Lake environment, I initially used the same criteria as Forest Management where convergence is achieved when the new policy is identical to the previous policy. Initially I was getting error plots like Figure 9.

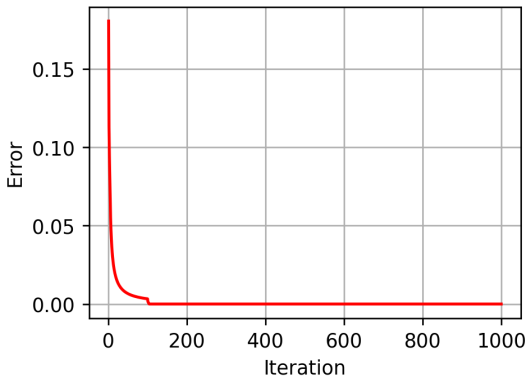

Fig. 9: Error of Policy Iteration ($\gamma = 0.99, \epsilon - decay = 0.999$) algorithm with simple convergence criteria in the Frozen Lake Environment ($size = 16$)

The flat line behavior implies that the convergence criteria was set incorrectly. Upon closer investigation, I noticed that the algorithm was updating only a few low value states. Furthermore, these states were dependent on each other and so they would keep changing in an endless loop. I added an additional convergence criteria that would stop training if the average maximum utility delta was within 0.01 for 50 iterations. With this additional parameter I achieved faster convergence as seen in Figure **??**.

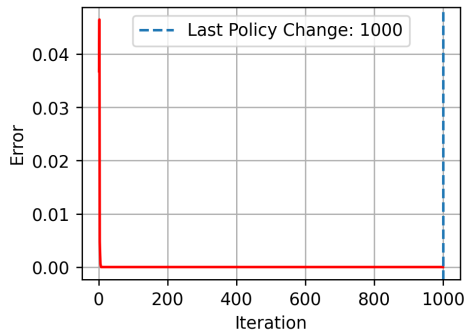

Fig. 10: Error of Policy Iteration ($\gamma = 0.999$) algorithm with updated convergence criteria in the Frozen Lake Environment ($size = 16$)

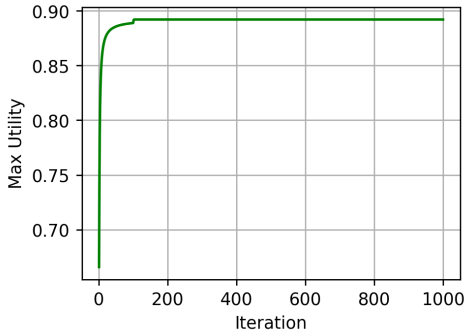

Fig. 11: Maximum Utility of the Q function table of Policy Iteration ($\gamma = 0.999$) algorithm in the Frozen Lake Environment ($size = 16$)

In Figure 12, we can see the final policy for PI. This policy is remarkably similar to VI, except for a few states that are in the the bottom left an top right. These states are unlikely to be encountered and so these two policies are essentially the same. PI exhibits a similar behavior to VI where it attempts to slip through the tunnel of holes on the bottom left. It does this to avoid slipping into the holes.

In Table II, we can see that Value iteration was able to converge much faster than Policy iteration. This is because the state space is now very large where it was much smaller with the forest management. This means that the system of linear equations is much more difficult to solve and thus takes more time.

	Forest Management		Frozen Lake	
Algorithm	Small	Large	Small	Large
Value Iteration	0.009	0.12	0.01	0.047
Policy Iteration	0.012	0.54	0.02	2.102
QLearning	0.309	1.31	3.00	15.4

TABLE II: Table of Training Times (sec) for various Reinforcement Learning algorithms in the Frozen Lake and Forest environments.

C. Q-Learning

Q-Learning convergence is defined exactly how it was defined in the Forest management and I decided on an convergence value by examining the error curve plots of various state spaces. I ended up selecting 0.001.

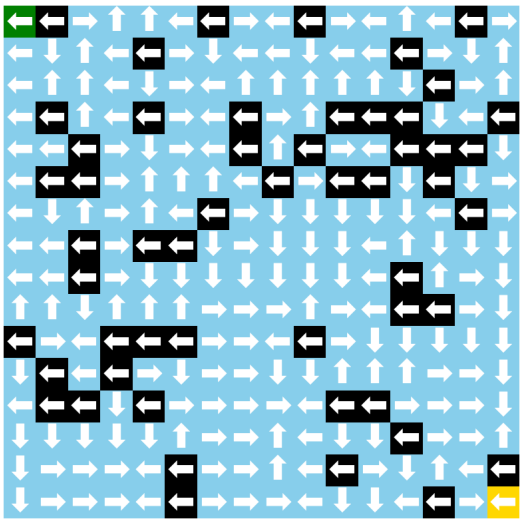

Fig. 12: Final Policy for Policy Iteration in the Frozen Lake environment. Black squares represent holes, the yellow square represents the goal state, blue squares are the frozen lake and the green square is the starting state.

In Figure 14, we can see that there is a very large dip in error before picking back up. This additional error comes from the Q-Learning agent propogating additional negative rewards about the final holes through the Q-Matrix. In a similar light to the PI, these final state changes are not affecting the final obtained reward. In Figure 13, we can see that the average reward is peaking around 60,000 iterations.

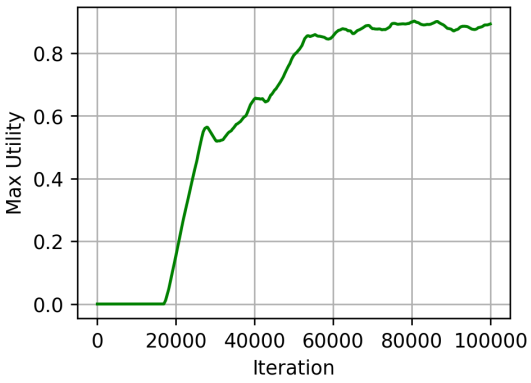

Fig. 13: Maximum Utility of the Q function table of Q-Learning ($\gamma = 0.999$) algorithm in the Frozen Lake Environment ($size = 16$)

In Figure 15 we can see the final policy for the Q-Learning algorithm. This policy is very close to the VI and PI policies that were discussed before. We see the exact same behavior in the critical left side middle hole tunnel where the other two algorithms were using the slip mechanic to guarantee they will pass by the holes. This is a quite novel behaviour and I'm surprised I saw it in all three algorithms.

As with Forest Management, I investigated the exploration-exploitation tradeoff. In Figure 16, we can see that given sufficient iterations, all $\epsilon - decay$ values eventually arrived to a similar Average Reward. Policies with smaller $\epsilon - decay$

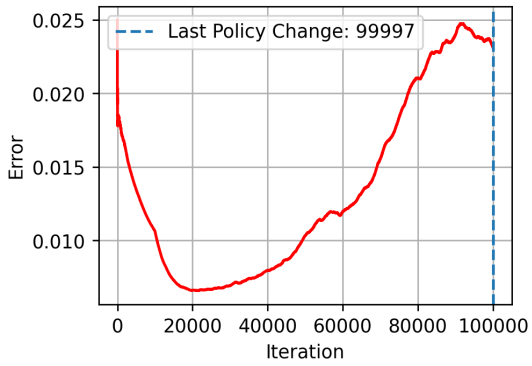

Fig. 14: Error per iteration of Q-Learning ($\gamma = 0.999$) algorithm in the Frozen Lake Environment ($size = 16$)

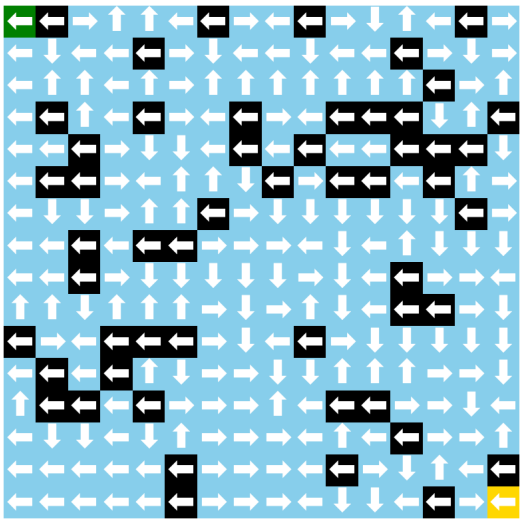

Fig. 15: Final Policy for Q-Learning in the Frozen Lake environment. Black squares represent holes, the yellow square represents the goal state, blue squares are the frozen lake and the green square is the starting state.

values maximize reward faster, and so we see their curves increase earlier. However this can be detrimental if the policy diverges during this maximization period. $\epsilon - decay = 0.8$ value shows a huge fall off after reaching 0.8 average where the policy diverged and took a 1000s of iterations to recover. There is a balance here between exploring viable paths and options before optimizing with one.

If given more time, I would have liked to explore more novel exploration methods such as RE3 which prioritize actions that provide more information about the state space instead of using random chance to determine the next action.

D. Comparative Analysis

If there is a Transition Matrix and Reward Matrix available, Policy Iteration and Value Iteration are easily the best choices. They are very quick to discover optimal policies and are not prone to falling into local optima since from the start, the global optima is known.

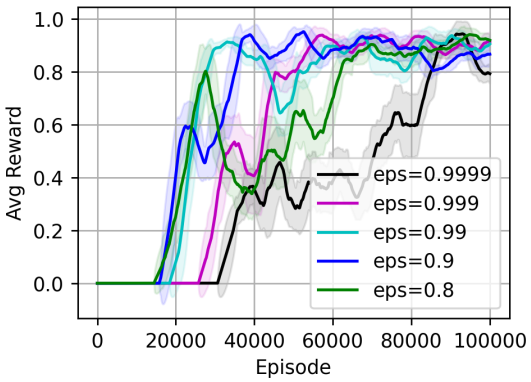

Fig. 16: Average Reward per iteration of Q-Learning ($\gamma = 0.99$) algorithm with various $\epsilon - decay$ values in the Frozen Lake Environment ($size = 16$).

Comparing the Maximum Utility between VI, PI and Q-Learning, they end up converging on remarkable similar values. I think that Q-Learning was able to succeed better in this environment because there were no local optima to get stuck at. The reward shaping helped the agent avoid the holes an make progress through the environment looking for positive rewards.

PI is not as valuable in this environment due to the increased wall clock time of running the linear equation solver in a large state space.

Looking at II, we can see that Value iteration is not as sensitive to changes in state space. This could mean that it is the ideal choice for this environment.

IV. Conclusion

In conclusion, Value Iteration and Policy Iteration are effective tools for determining optimal polices. Policy Iteration can be more useful when interactions with the environment are more costly. We also showed that Q-Learning can find policies but struggles to move past local optimas.

We also showed that changes in state space cause Policy Iteration and Q-Learning to take longer to compute.

V. Appendix

References

[1] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping," 2020. [Online]. Available: https://arxiv.org/abs/2011.02669