Assignment 4: Reinforcement Learning

Naziul Talukder
ntalukder6@gatech.edu

***Abstract*—In this paper, I will choose two MDP problems - Frozen Lake and Forest Management and find solutions using model based and model free methods. As model based methods: policy iteration and value iteration will be used. As a model free method: Q learner will be used. I will analyze the differences in the optimal policies obtained by each methods and necessary hyper-parameter tuning. Frozen lake will be treated as a smaller state space problem. The forest management problem will be treated as the larger state space problem. I will also discuss how increasing the state space impacts execution time and performance of the policies.**

I. Frozen Lake

One MDP problem I discussed in this paper is the frozen lake problem. The problem dictates that an agent must cross a frozen lake with $n \times n$ grids. The starting position of the agent is considered **S** state. The agent needs to cross the lake to reach a certain position. This is considered **G** state. The agent needs to move through other grids which are typically frozen. They are considered **F** state. There are possible holes in the frozen lake which is defined by the **H** states. The objective of the agent is to start from the **S** state, walk over the **F** states avoiding the **H** states and reach the **G** state. The agent can move up, down, left or right as it attempts to reach the goal. The action space is 4. In a stochastic world, we assume that the lake is slippery. The agent does not always move to the intended direction due to the slipperiness. The agent will move towards intended direction with probability $\frac{1}{3}$ and will move towards either of the perpendicular directions with equal probability of $\frac{1}{3}$. The observation space or states depend on the size of the grid. There are possible n^2 states (including the start and goal) for an $n \times n$ grid frozen lake problem. Each grid in the system can constitute as a state. It can be the starting, hole, frozen or goal state. The agent receives $+1$ reward when it reaches the goal state. It receives 0 for reaching a frozen state or hole state.

A. Value Iteration

Using value iteration I tried to solve the frozen lake problem obtaining the best policy. Value iteration converges to the best value function and optimal policy when γ is between 0 and 1. Because the state space is small we expect value iteration to converge quickly. I used 0 as the initial value function for each state. For the exit scheme in the iteration, I used $100,000$ as maximum iteration and 10^{-20} as the convergence threshold. The value function is considered to have converged if the new updated value function is within the convergence threshold of the old value function. The reward and transition functions discussed in the previous section was used. Using the Bellman equation and the discounted factor γ I calculated and updated the value function for each states until it converged or reached the maximum number of iterations. Once the optimal value function was obtained I extracted the optimal policy from there as seen in figure 1. The agent is performing well to avoid the holes as the policy shows the agent chooses opposite directions near the hole states. In some states, the agent is driving towards the hole. Due to the floor being slippery the agent sometimes mistakenly chooses to walk toward the hole states. On the right side, the agent chooses to keep moving right instead towards the goal state. In this reward scheme, the agent does not get penalized for taking more steps towards the goal. Due to stochastic nature, the agent can eventually move towards the goal state. This behavior can be improved by introducing negative rewards. If the agent receives negative rewards for each step, then it would try to minimize it and choose the shortest path as the optimal policy. That would improve the current policy.

Figure 2 shows analysis of the value iteration process as γ is varied. Best rewards were obtained when the discount factor was closer to 1. If the discount factor was too small the reward was insignificant. This is apparent due to the structure of the reward function. The agent only receives 1 reward when it reaches the goal state. With lower discount factor, the agent prioritizes immediate rewards. With higher discount factor, the agent prioritizes long term goals. In this case which allows it to travel to the goal state. The figure discusses high γ values. With very high γ the rewards are higher as shown in the graph. The optimal γ value is very close to 1. The figure also shows that execution time for high discount factor (0.999) is much lower compared to execution time of low discount factor (0.900). The range of discount factor discussed here does not impact the number of iterations needed to converge as seen in the figure. It requires 2278 iterations to converge when the threshold of convergence is 10^{-20}. $\gamma = 0.999$ was used to obtain the optimal policy shown in figure 1.

B. Policy Iteration

Policy iteration is similar to value iteration where instead of initializing and updating the value function, one updates the policy itself. I used a random policy as the initial policy. With the initial policy, the initial value function is calculated. A new policy is enacted from the new value function. Recursively the policy and the value function gets updated until they converge or reaches the maximum number of iterations. $200,000$ was the maximum number of iterations that was used. 10^{-10} was used as the convergence threshold for the value function here. The policy is considered to have converged if older policy and updated new policy is the same policy or the value function of the new policy is within the convergence threshold of the

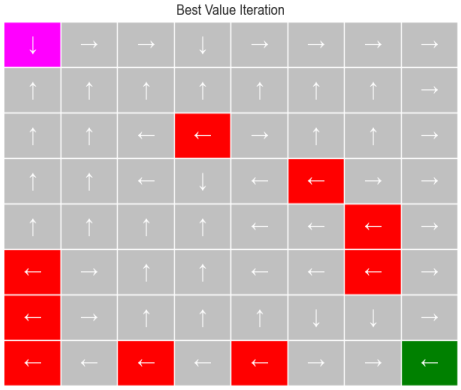

Fig. 1. Optimal Policy (VI)

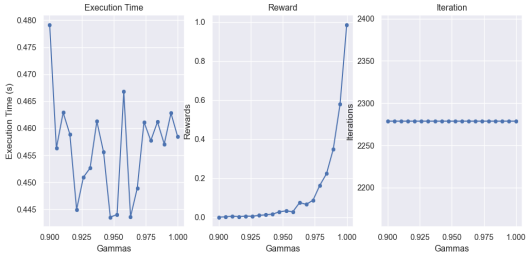

Fig. 2. Analysis (VI)

value function of the old policy. Policy iteration is expected to converge quicker than value iteration. This is due to directly updating the policy. In this method, we are greedily updating the policy based on the current value function. But in value iteration we need multiple iterations to converge and then we can update the policy. Policy iteration is more computationally expensive. The value iteration only updates the policy after convergence. But policy iteration needs to calculate the value function in each iteration. Figure 3 shows the best policy obtained through policy iteration. Most of the times, the agent chooses to avoid the holes and moves towards the goal state as seen in the figure. On the right corner the agent continuously moves to right and due to slippery nature of the floor, the agent eventually reaches the goal state. This could be improved by introducing negative rewards in the rewards scheme to force the agent to find the shortest path to the goal. The policy obtained is similar to the policy obtained by value iteration. Similarly in some states, the agent mistakenly chooses to walk towards the hole states.

Figure 4 shows analysis of policy iteration as γ is varied. Similarly as value iteration, we observed high rewards for high γ values. The best γ value in policy iteration is 0.999 similar to value iteration case. The rewards graph in both 4 and 2 are similar. For policy iteration with $\gamma = 0.999$ the execution time is less than $0.4s$ whereas for value iteration it is $0.46s$. Only 5 iterations are needed to converge for the policy iteration case whereas for value iteration it was significantly higher. Our result signifies the strength of policy iteration and directly updating the policy compared to the value iteration approach.

Fig. 3. Optimal Policy (PI)

C. Comparison: PI vs VI

Figure 5 shows the convergence plot of both policy iteration and value iteration methods when $\gamma = 0.999$. Even with small number of iterations policy iterations outperforms value iterations. But the iterations of policy iteration are computationally more expensive. Value iteration observes high rewards very quickly but it takes a long time to converge to the optimal policy.

The optimal policy obtained through value iteration and policy iteration can be visualized in figure 1 and 3. On the first row the fourth column is different for these two figures. It shows how the optimal policy obtained through these two methods are different. The figure shows that in most of the states the optimal policy obtained through value iteration is similar to the optimal policy obtained through policy iteration.

Due to the difference in approach (focusing on policy vs focusing on value function) in these two methods the optimal policy obtained is different. Due to the stochastic nature of the problem and the reward function it is natural to observe different optimal policies for these two methods. Exploiting the improvement in policies is not the same as exploiting the improvement in the value function for this system. Hence, we are observing two different optimal policy obtained by these methods. Changing the reward function, the hole states (introducing more holes or less holes) and the probability of slipping on the states might change the optimal policy obtained through the methods. It's possible that changing these values we could observe a system where exploiting the improvement in policies correlate with exploiting the improvement in the value functions and vice versa. In that system, we could observe same optimal policy observed by both methods.

D. Q Learning

Q learning is a model free approach compared to value iteration and policy iteration which are model based approach. It uses a quality factor rather than state or policy to obtain the best policy. The quality factor depends on both state and action. The quality factor can be connected to both the value function and the policy of the state. Q learning does not need knowledge of the transition probability and rewards. It can continuously interact with the environment updating the Q

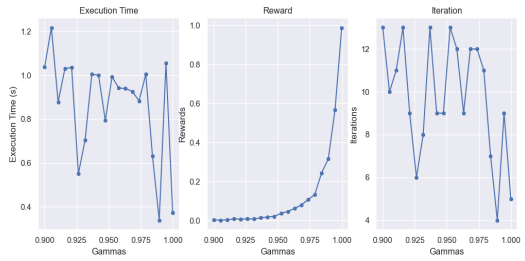

Fig. 4. Analysis (PI)

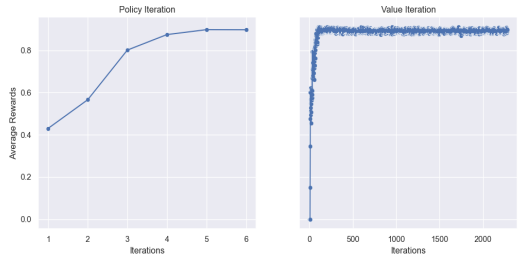

Fig. 5. Convergence Plot VI and PI

Fig. 6. Q Learning Policy

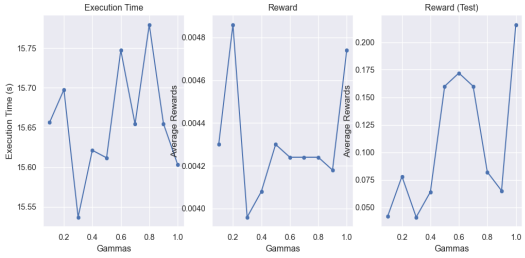

Fig. 7. Q Learning Discount Factor Tuning

value towards the best rewards. Through trial and error while interacting with the environment the optimal Q values (once it has converged) can be obtained. The optimal policy can be retrieved from the optimal Q value.

In the Q learning we need to balance the exploration and exploitation. I used a greedy epsilon approach with a decay schedule. It determines if I will choose the best action for a specific state or a random action to update the Q value. During each iteration, there is ϵ probability that a random action is taken. During the first iteration $\epsilon = 1$. After each iteration, the it decreases 10% until it reaches 0.01. During early stages, the agent frequently explores the action spaces choosing random actions. At the later stages, it is more selective and likely to choose the best actions.

At first I initialized 0 as Q value for all the possible states and actions. Beginning from the start state, the agent interacts with the world updating the Q values based on the action, next state and reward.The Q values were updated following this equation:

$$Q(s,a) <= (1-\alpha)Q(s,a) + \alpha[R + \gamma max(Q(s',a'))]$$

The learning rate α in Q learning determines the impact of new Q value. A high learning rate implies the new Q value determines the updated value. A lower learning rate ensures older Q value persist through the updates. With higher sub-optimal learning rate the agent can be unstable and not converge at the goal state. With lower sub-optimal learning rate the agent will not take the new information into account properly. It might result it slow convergence or no convergence.

I varied the discount factor as seen in the figure 7. The training phase (middle graph) shows the rewards as the optimal policy is learned. The testing phase (third graph) shows the rewards as the optimal policy is used over 1000 iterations. $\gamma = 0.2$ provides great rewards as the agent learns the policy but in the test phase the rewards returned is poor. This indicates that $\gamma = 0.2$ would not be the best choice. The highest γ value provided the best test rewards and less execution time indicating faster convergence compared to other values. It reflects our observation from the policy iteration and value iteration where higher γ value provided better convergence.

Figure 8 shows how the average rewards change with varying learning parameter. As expected, the high α parameter makes the agent unstable leading to higher execution time and low average rewards. Best average reward and lowest execution time are observed in the learning phase for $\alpha = 0.234$.

I used $\alpha = 0.234$ as the learning rate and $\gamma = 0.999$ as the discount factor. The iteration stops when the agent reaches the goal state or the maximum number of iterations is obtained. I used $50,000$ as the maximum number of iterations. Figure 6 shows the optimal policy obtained. This policy differs from the ones obtained by policy iteration and value iteration. The states above the goal state, chooses to move towards the goal state in this policy. For other policies the agent chose to move towards right. The agent was more focused on avoiding the hole states. But Q learner policy chooses to move towards the goal state. Also for the two states above the holes on the last row of the lake, the Q learner policy moves right: towards the goal state. Whereas in the other policies, the agent chooses to move up. The agent is more active in avoiding the holes in the policy and value iteration methods. In the Q learner method, the agent is more active in reaching the goal state.

E. Comparison: Model Based vs Model Free

On the frozen lake problem, the model based approaches like policy iteration and value iteration converged much faster compared to the model free approach like Q learning. Figure 4 and 2 shows the execution time for best discount factor value

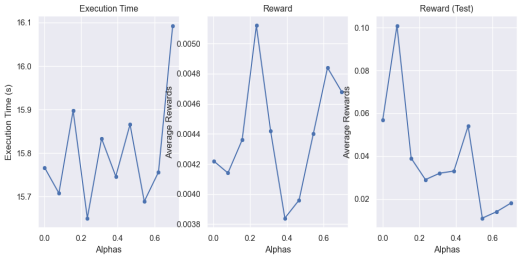

Fig. 8. Q Learning Learning Rate Tuning

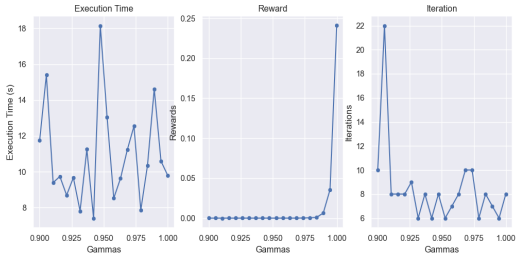

Fig. 9. Analysis Higher State Space (PI)

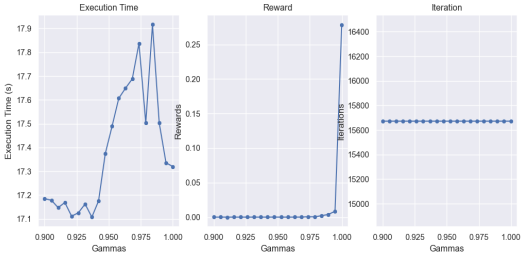

Fig. 10. Analysis Higher State Space (VI)

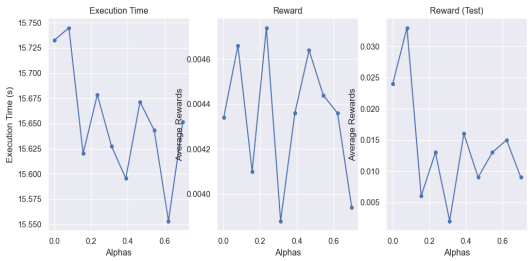

Fig. 11. Q Learning Learning Rate Tuning 400 states

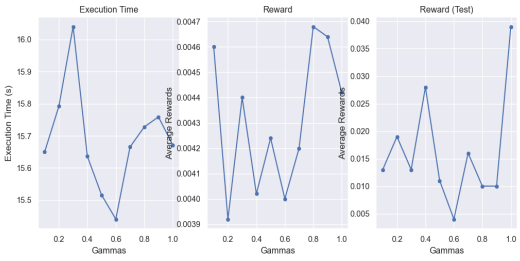

Fig. 12. Q Learning Discount Factor Tuning 400 states

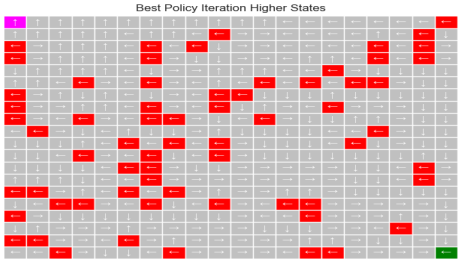

Fig. 13. Policy 400 states (VI)

$\gamma = 0.999$ was close to $0.4s$. Whereas figure 7 and 8 shows that for the best discount factor and learning rate the execution time is a little more than $15s$. The average reward for properly tuned agents using value iteration or policy iteration is close to 1. Whereas the average reward for Q learner is close to 0.2 as seen in figure 7. For this problem, the model based methods tended to outperform the model free method Q learning.

From the visualization of the policies in figure 1, figure 3 and 6 the policy learned by the Q learner seems the most accurate as it correctly identifies the right actions for the states neighboring the goal. The performance of the algorithms could be improved using different rewards in each state. Modifying the rewards to -10 for holes, -1 for all frozen tiles and 1 for the goal state would impact the performance of the algorithms. With introduction of negative rewards, the convergence will be faster compared to the current system. This would ensure the agent prefers policy with fewer steps.

Figure 9, 10, 12 and 11 shows how a similar analysis on each of the algorithms can be performed on a problem where the lake is a 20×20 grid. The state space for this more complex problem would be 400. The execution time for both policy iteration and value iteration is similar. The execution time for policy iteration at the best configuration is close to $8s$. Execution time for value iteration at the best configuration is close to $17s$. Due to more expensive iterations in higher state space, with less iterations: policy iteration outperforms value iteration in higher state space. The rewards is significantly lower for both policies in this higher state space problem. It implies the reward scheme in place is not compatible for higher state space. Q learning's execution time is similar to what we observed in the lower state space problem. But the rewards observed in the testing phase is significantly lower now. This implies Q learning is not effective with this rewards scheme in higher state space. The average rewards for Q learning at best configuration in lower state space is 0.2 and in the higher state space is 0.04.

Figure 14 and 13 shows the policies obtained policy iteration and value iteration. Shaping the rewards system and introducing negative rewards will provide more incentive to the agent to adopt the path with least number of steps. Modifying the rewards would improve the performance, execution time of policy iteration, value iteration and Q learner.

II. Forest Management

The second MDP problem we will discuss is a non-grid world problem called Forest Management. I considered a

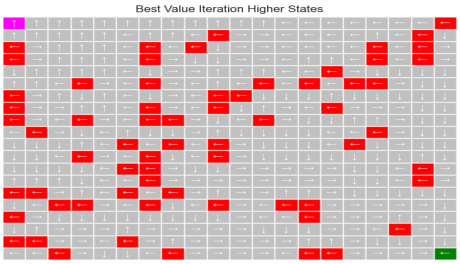

Fig. 14. Policy 400 states (PI)

20×20 grid forest for this problem. There is a probability of wildfire in the forest, $p = 0.1$. The objective is to maximize profit by cutting down trees and nurturing trees. In each iteration if a tree is cut then the agent receives 10 rewards. But if the agent waits and let the tree grow then it receives 5 rewards. The goal of the agent is to maximize the reward. When a tree catches fire the reward value from that tree is 0. The agent needs to learn to balance between risk and immediate reward and future reward. The action space is 2 in this problem. The agent can either cut a tree or wait and let it grow. The state space for this problem depends on the size of the forest. In this case there are 400 possible states. Each state can either have a tree, or not have a tree (due to fire or cutting).

In the frozen lake problem: the action space was 4 as the agent's movement was the action. For this problem it is only 2. For frozen lake the objective was to reach the goal state, whereas here the objective is to find a balance between immediate reward and future rewards as we cut trees in the forest. As the action space is lower for this problem, we expect lower execution time compared to the frozen lake problem (if the grid is considered of same size). Because lower action space implies less expensive transition matrix and value function and policy function calculations.

A. VI, PI and Comparison

I carried out a similar value iteration and policy iteration for this problem like the frozen lake problem. For value iteration maximum $100,000$ iterations were used similar to the frozen lake implementation. I varied the convergence threshold ϵ and found the best rewards for value iteration using $\epsilon = 0.001$ for this reward scheme. I considered ϵ values ranging from 100 to 0.0000001. The value function is considered to have converged if the new updated value function is within the convergence threshold ϵ of the old value function or maximum number of iterations have been reached.

Figure 18 shows the optimal policy obtained with varying discount factor γ through value iteration when $\epsilon = 0.001$. In the figure the grey portion indicates trees the agent chooses to cut and the green portion indicates trees the agent chooses to wait on. The reward scheme provides higher reward for cutting the tree. When discount factor is very low, the agent only focuses on immediate rewards. Hence, it decides cutting the trees to avoid wild fire and obtain the immediate rewards. The policy shows more trees being cut down at this stage. When the discount factor is higher, the agent prioritizes future rewards and waits cutting the tree. Our expectation is verified in figure 18.

The policy iteration implementation is similar to the method described for the frozen lake problem. An initial policy was randomly chosen and iteratively improved to obtain the best policy. The policy is considered to have converged if the new updated policy is same as the older policy or the maximum number of iterations have been reached. The maximum number of iterations was $100,000$ during policy iteration.

We expected policy iteration to be more computationally expensive compared to the value iteration method. As the state space is larger for the forest management problem compared to the frozen lake problem the difference in execution time is expected to be more apparent (comparing with the 8×8 grid lake). As the policy iteration method updates the policy and calculates the value function: I expected the execution time to be higher for this method. I varied the discount factor γ to observe the optimal policy. Highest reward was obtained using $\gamma = 0.999$.

Figure 16 shows how the policy changes with varying the discount factor γ. The grey portion shows the trees the agent chooses to cut. The green portion shows the trees the agent chooses to wait on. The lack of green portion for smaller γ values indicate that the agent chose to cut all the trees. Similarly as the value iteration problem: with lower γ the agent focuses on immediate rewards. As cutting the tree is more rewarding than waiting: the agent chooses to cut the tree. With increasing γ the agent focuses more on the future rewards. Hence, we should expect the agent waiting more for higher γ values. Figure 16 verifies this assumption. As seen in the frozen lake problem, the policy obtained by both methods are very similar but they are not exactly the same. The policies observed in figure 16 and 18 are not exactly the same: but in naked eye they seem very close to each other. Figure 15 shows that policy iteration converges to the optimal policy for 22 iterations and the average rewards observed is much higher compared to value iteration from figure 17.

As discussed in the previous section: we expect policy iteration to outperform value iteration in terms of wall clock time. I observed, policy iteration converged at $0.2172s$ and value iteration converged at $0.2907561s$. Policy iteration execution time was much faster than value iteration. Similar to the previous section, we expect policy iteration to converge at fewer iterations. As figure 15 shows policy iteration converges at 22 iterations. Whereas figure 17 shows value iteration converges at 210 iterations.

In the frozen lake problem when state space was 400 (20×20 grid) the execution time for policy iteration was close to $8s$ and for value iteration was close to $17s$. For the forest management problem: execution time for policy iteration was less than $0.3s$ and for value iteration was close to $0.3s$. This verifies the assumption that execution time would be significantly lower for forest management problem compared to the frozen lake problem due to having lower action space.

Figure 20 and 19 shows how the execution time varies with number of states. With higher state space the execution time drastically increases. But the overall trend stays the same. With

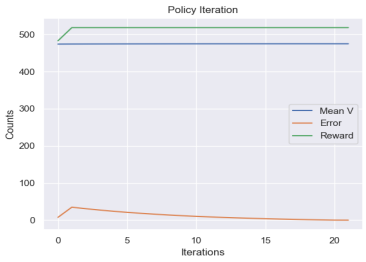

Fig. 15. Policy Iteration Analysis Forest

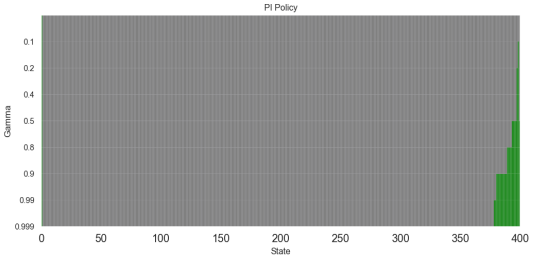

Fig. 16. Visualization of policy varying Gamma (PI)

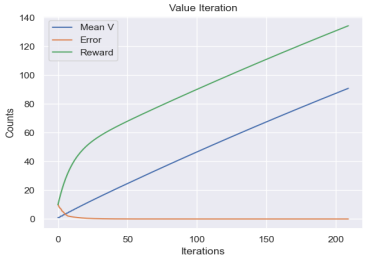

Fig. 17. Value Iteration Analysis Forest

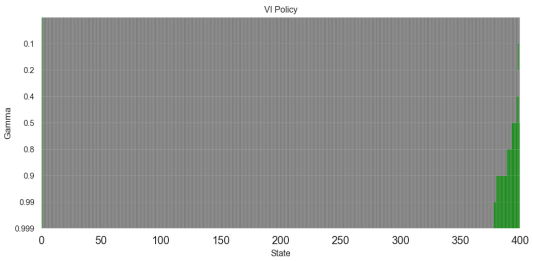

Fig. 18. Visualization of policy varying Gamma (VI)

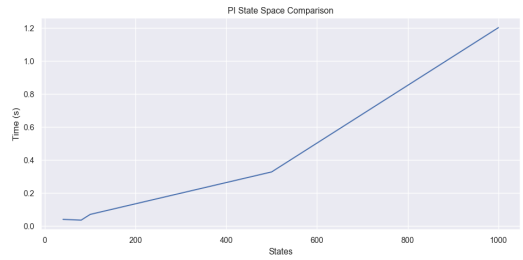

Fig. 19. State space vs Execution Time (PI)

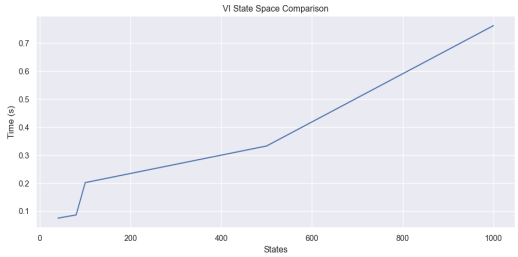

Fig. 20. State Space vs Execution Time (VI)

lower state space and same hyper-parameters the agent tends to cut fewer trees. With higher state space the agent opts for cutting more and more trees in the optimal policy. Optimal policies for both PI and VI looks similar but they are not exactly the same.

Figure 20 shows how with increasing state space the execution time changes for value iteration and figure 19 shows how it changes for policy iteration. It shows that for smaller state space, value iteration converges faster. For larger state space policy iteration converges faster. It implies, for smaller state space, the value function calculation needed for policy iteration acts as a penalty. It is more efficient to opt for value iteration at that stage. But with higher state space, policy iteration outperforms as it requires less iterations to converge. But also, at highest state space: value iteration outperforms policy iteration. It is possible that updating policies can be too complicated and expensive calculation. In that stage, the value iteration outperforms policy iteration in terms of execution time.

B. Q Learning

Like the frozen lake problem, I used a similar implementation of Q learning for this problem. The initial Q value for all states and actions were 0. In this epsilon greedy approach the ϵ and learning rate α decayed with each iteration to balance the exploitation and exploration of the agent. During the hyper-parameter training stage multiple values such as $0.001, 0.01, 0.0001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 0.9, 1.0$ were used as initial learning rate. I used values between 0 and 1 to as the learning decay factor. Minimum learning rate values were chosen to be values such as $0.00001, 0.0001, 0.001, 0.01$. In the decay schedule the initial learning rate is changed based on the exponential of the decay factor until the minimum value is reached. During each episode the learning rate is decayed based on this schedule to observe the optimal policy. The ϵ value is changed in the same method. Initial ϵ values were chosen such as $1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1$ with decay factor as $0.99, 0.9999, 0.999$ and the minimum value as $0.00001, 0.0001, 0.001, 0.01$. $0.1, 0.2, 0.3, 0.4, 0.5, 0.60.7, 0.8, 0.9, 0.99, 0.999, 0.9999$ were used as discount factor γ. 10000 episodes were used to find the best policy. Figure 21 shows a portion of rewards obtained for combination of these values as hyper-parameter. The figure demonstrates how using different decay factor for ϵ and α influences the agent to converge to different

```
#####
Epsilon:  1e-06
 Epsilon Decay:  0.9999
 Epsilon Min:  1e-05
 Alpha:  0.001
 Alpha Decay:  0.99
 Alpha Min:  0.001
 Gamma:  0.1
 Avg Rewards:  19.97
#####
Epsilon:  1e-06
 Epsilon Decay:  0.9999
 Epsilon Min:  1e-05
 Alpha:  0.01
 Alpha Decay:  0.99
 Alpha Min:  0.001
 Gamma:  0.3
 Avg Rewards:  189.865
#####
Epsilon:  1e-06
 Epsilon Decay:  0.9999
 Epsilon Min:  1e-05
 Alpha:  0.01
 Alpha Decay:  1.0
 Alpha Min:  0.01
 Gamma:  0.1
 Avg Rewards:  268.79
#####
Epsilon:  1e-06
 Epsilon Decay:  0.99
 Epsilon Min:  1e-05
 Alpha:  0.001
 Alpha Decay:  1.0
 Alpha Min:  0.01
 Gamma:  0.6
 Avg Rewards:  426.345
#####
Epsilon:  1e-06
 Epsilon Decay:  0.99
 Epsilon Min:  0.0001
 Alpha:  0.001
 Alpha Decay:  0.99
 Alpha Min:  0
 Gamma:  0.1
 Avg Rewards:  426.55
```

Fig. 21. Hyper-parameter tuning of Q learner

policies resulting in difference in the rewards obtained. The best average rewards was close to 426 using Q learner. With various combination of these values the policy providing the best rewards consistently would be chosen as the optimal policy.

The optimal policy was obtained when initial learning rate $\alpha = 0.001$, learning rate decay factor 0.99, minimum learning rate 0, $\epsilon = 0.000001$, decay rate for ϵ 0.99 and minimum ϵ 0.0001, discount factor $\gamma = 0.1$. Figure 22 shows how average rewards change with number of episodes in the Q learner. With increasing episodes the execution time increases linearly as

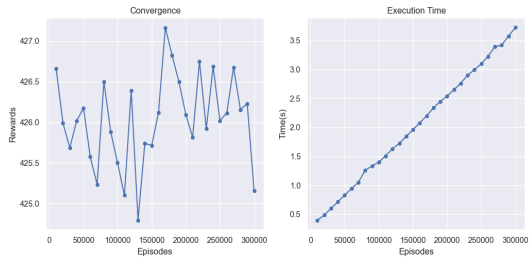

Fig. 22. Convergence of Q learner

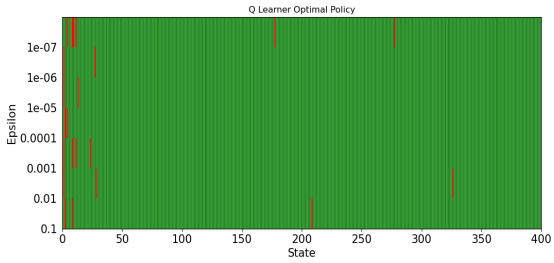

Fig. 23. Visualization of policy (Q Learner)

seen in the figure. But the average rewards does not improve. Hence, I chose $10,000$ as maximum number of episodes for the Q learner.

Figure 23 shows the visualization of the policies obtained by Q learner as ϵ value is varied. The optimal policy is obtained for $\epsilon = 0.000001$. In the figure red color denotes the states where the tree is cut and green where the agent chooses to wait. Figured 16 and 18 shows visualization of optimal policy obtained through policy iteration and value iteration. In those figures grey indicates cutting trees compared to red for the Q learner graph (I chose red here for ease of visualization). The agent chose to cut more trees for policy obtained by policy iteration and value iteration compared to policy obtained by Q learner. The policy of Q learner is drastically different than the policy observed through the model based methods. Q Learner prioritizes future rewards much more compared to policy iteration or value iteration. Hence, it chooses to wait.

Figure 24 shows how the execution time changes as the state space of the forest is varied. Similar to the model based approaches, as seen in figure 19 and 20, with higher state space execution time increases.

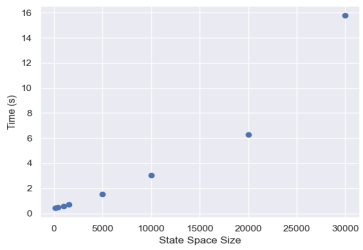

Fig. 24. State Space and Execution Time (Q Learner)

C. Comparison: Model Based vs Model Free

The policy of Q learner is drastically different than the policy observed through the model based methods. Q Learner prioritizes future rewards much more compared to policy iteration or value iteration. Hence, it chooses to wait. Model based methods tend to cut a lot more trees compared to Q learner policy.

Execution time for policy iteration was $0.23s$, value iteration was $0.29s$ and for Q learner was $0.455s$. It shows that policy iteration was the fastest and Q learner was the slowest. Similar to the frozen lake problem, policy iteration converged faster. With increasing state space the execution time for Q learner increased for this problem which wasn't apparent in the frozen lake problem. During the episodes: policy learner observed the highest rewards and value iteration observed the lowest rewards. Q learner obtained high rewards choosing to wait on more trees whereas policy iteration obtained high rewards choosing to cut more trees. Changing the reward scheme and also probability of wildfire could change the policies obtained by these methods. In this case, the probability of wildfire was 0.1. But if the probability of wildfire depended on the number of trees cut directly then it is more likely that optimal policies would be more similar. In that case, choosing to cut trees would impact how often wildfire occurs. This might result in more uniform policy regardless the methods chosen.

III. Conclusion

I have analyzed the frozen lake and forest management problems using policy iteration, value iteration and Q learner. Policy iteration outperformed all the methods in execution time. Q learner had the highest execution time. In the reward scheme for frozen lake problem, there was no negative rewards. Even though the policies converged, a better policy could be obtained changing the reward scheme by introducing negative rewards. This would ensure the shortest path is taken to the goal state. From visualization of the policies Q learner tended to have policy with shortest path leading to the goal state. Q learner focused more on reaching the goal state whereas value iteration and policy iteration focused more on avoiding the hole states. With increasing the state space the execution time increased. In the higher state space policy iteration outperformed value iteration in execution time. For value iteration the execution time increased drastically. The execution time for Q learner did not differ much in the higher state space. This is due to Q learning failing to find a good policy in the higher state space. Q learning's performance degraded drastically as we moved from 64 to 400 state space. For the forest management problem, both value iteration and policy iteration converged to similar policies. But Q learning converged at drastically different policy. Q learner focused on waiting compared to model based methods that focused on cutting the trees. For the forest management problem, changing the rewards scheme and the probability of wildfire might allow the methods to find better policies. Increasing the state space in this problem, increased the execution time for all the methods in a similar trend. Policy iteration obtained more rewards in the episodes compared to Q learner and value iteration. Policy learner also outperformed them in terms of execution time. The success of policy iteration implies, calculating the policies is not too complicated in these problems and it is a helpful shortcut compared to calculating all the value functions like value iterations or interacting with the environment for all the possibilities like Q learner. Comparing both problems: due to higher action space in the frozen lake problem, we observed higher execution time compared to forest management problem.