Machine Learning - CS 7641 Assignment 4

Anthony Menninger
Georgia Tech OMSCS Program
amenninger3
tmenninger@gatech.edu

Abstract

This paper explores Markov Decision Processes through the use of Value Iteration, Policy Iteration and Q Learning. It looks at a simple "Forest" MDP and a larger Frozen Lake "Grid World".

Introduction

The key toolset I used was the hiive fork [3] of the MDP Toolbox for Python [2], which is derived from the MDP Toolbox [1]. I also used the OpenAI Gym Frozen Lake environment [4] for the larger Frozen Lake MDP.

Forest MDP

Forest MDP is a simple MDP from the hiive MDP Toolbox [2] that models the value of a forest with respect to two actions that can be performed each year: ***wait*** or ***cut***. There is a stochastic element of forest fires that occur with probability ***p***. Each year is a state, with a max of ***S*** years / states. Cutting deterministically transitions to the initial state, ***state=0***. Cutting provides a ***cutting reward*** of 0 in the initial state, ***cutting reward*** in the final state and a reward of 1 in all other states. Waiting transitions to the next year, or if in the max state, remains there. Forest fires provide a stochastic element, with a fire transitioning back to the initial state. A ***waiting reward*** only occurs in the max state, ***state=S***. This is a continual MDP, with no terminal or absorbing states.

The base setup for ***Forest*** is seven years (***S=7***), a 10% chance of forest fire (***p=0.1***), a (***cutting reward=2***) in any state and a (***waiting reward = 4***) only in the final state.

In the context of MDP's, each state has an action that maximizes expected value and the set of maximizing actions for all states is called a ***policy***. This policy can be examined to determine what rational actors are likely going to do.

There are several interesting aspects to this problem. ***Forest MDP*** models a key choice being made today around ecology and the environment. What are the rewards needed to keep forests around? How do maximizing actions (***policy***) change as the chance of forest fires increase? How does the length of the considered horizon (***Gamma discount***) influence expected outcomes.

This also highlights a key challenge in MDP's and Reinforcement Learning, which is understanding rewards and setting them appropriately. ***Forest MDP*** allows modeling of different situations to inform the construction of public policy for governments, NGO's and corporations. The ***cutting reward*** might be clearly modeled using expected timber value, but other types of reward may want to be considered in the ***waiting reward***, such as carbon reduction and maintenance of ecological diversity. This MDP can be used to model economic rewards and penalties to compel desired outcomes, such as a carbon tax that would reduce the value of the ***cutting reward***.

As described, this MDP is more likely to be "solved" using Value Iteration (VI) or Policy Iteration (PI) in order to use it for comparing different tradeoffs and outcomes.

A key element in both Markov Decision Processes (MDPs) is the idea of a discount, which is often referred to as ***Gamma (γ)***. This discount factor, between 1 and 0, allows solutions for infinite MDPs by discounting future value by ***Gamma*** for each step into the future. In an infinite activity, this ends up valuing future value at $\frac{1}{1-\gamma}$. Gamma close to 0 creates a very close horizon where only immediate rewards are maximized, while Gamma close to 1 creates a long horizon, where maximizing long term reward is emphasized. As will be seen, MDPs are very sensitive to this and small changes can create very different solutions.

Lake MDP

Lake MDP is a larger MDP that uses the OpenAI gym Frozen Lake [4,5] to create its environment. It is a 2D grid with four actions to move ***up, down, left, right***. It can be any ***Size***, which is the number of grids on one size of the square, creating ***Size x Size*** states. The lake can be set to not slippery, where the action is deterministic, or slippery, where the action is stochastic. When slippery, the action has a 1/3 chance of moving in the desired direction, a 1/3 to the left of the desired direction and 1/3 to the right of the desired direction, which is fairly stochastic. This is an episodic MDP where the player starts in the first state - (0,0) upper left corner. The goal state (size, size) is in the bottom right corner and has a reward of 1. There are also random holes which are absorbing states with a reward of 0.

I created Small (***Size=4***), Medium (***Size=8***), and Large (***Size=16***) worlds to explore the impacts of different size worlds. All experiments used the stochastic slippery setting.

This MDP is interesting because it embodies a core Reinforcement Learning challenge of exploring an environment

and learning what to do. It describes a common, everyday task of exploring a world and learning where the rewards and traps are. Robots are often in a situation where they need to move about and find out what they should be doing. In addition, it creates a much more interesting set of transitions between the different states, as compared with the ***Forest MDP***, where transitions are either next state or back to beginning. Also, ***Lake MDP*** is episodic vs ***Forest MDP*** which is continual.

As described, this MDP is more likely to be "learned" in a setting using Q Learning to explore how agents use reinforcement learning to learn to navigate and behave in an unknown environment.

Forest MDP Value Iteration (VI) and Policy Iteration (PI)

We start with the ***Forest MDP***. The VI and PI implementations used for this experiment started with the hiive MDP Toolbox [3], which was then "forked" to allow for modifications for this analysis. Both VI and PI are model based algorithms, which means that the complete MDP is known and used for solving the problem. This matches the ***Forest MDP*** because it likely to be used to model different situations to see the impacts of different parameters.

$$V(s) = R(s) + \gamma \, max_{a'} \sum_{s'} T(s, a, s')V(s') \qquad (1)$$

Both VI an PI rely on the Bellman Equation in **Equation 1**. With Value Iteration, a sweep is performed of all states to update the Values using **Equation 1**. If this is run to infinity, it will converge to the exact solution. Practically, a setting of acceptable error is created to determine convergence, with each iteration comparing the old values and new values, with error as the maximum state value difference between iterations. VI is considered converged when the error is less than an error threshold ϵ, which is sometimes referred to as *E Stop* value in these charts. Confusingly ϵ also is the same designation for exploration in Q Learning.

Figure 1 shows the VI solution for the default setup of the ***Lake MDP*** outlined in the introduction with the default VI setting of *Error Threshold=0.01*. First, we can see that it takes 49 iterations to solve. Initially, the values for all states are low, but over time the values increase and shift from the high value end state towards the initial state. As the values increase, the policy changes from cutting in almost all states to waiting in all states. The final policy is reached in 7 iterations, but the final values aren't reached for much later after ϵ error threshold convergence. Looking at the value of the final state, even though we only get a reward of 4 for waiting, our expected value in that state is almost 27 because with this policy we will stay there until a forest fire forces a transition.

An important concern today is the increase in forest fires due to climate change. **Figure 2 a** shows how increases in the chance of forest fires dramatically reduces the values associated with waiting and creates optimal policies more likely to increase cutting. One way to reduce cutting is to increase the reward to wait. **Figure 2 b** shows this, with the reward increased to 8. We can see that cutting has been reduced vs the base, as well as dramatically increasing the

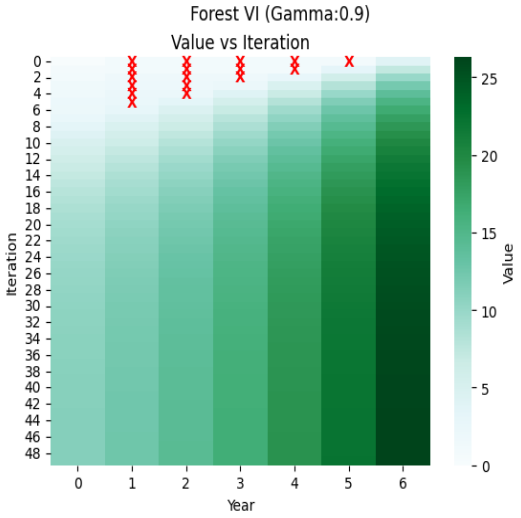
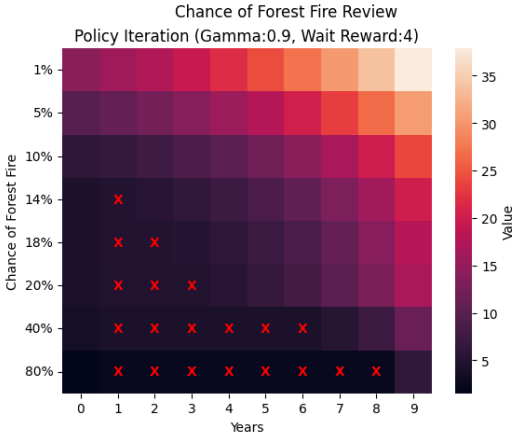
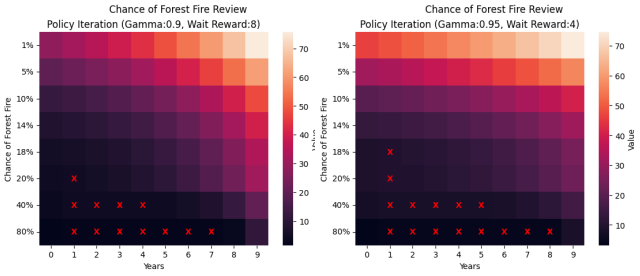


Figure 1: Forest MDP Value Iteration (VI) with *Error Threshold=0.01*. Red X indicates the cut action, otherwise the action is wait.



(a) Discount of 0.9 and a wait reward of 4.



(b) A wait reward of 8. (c) Discount of 0.95

Figure 2: Forest Values and Policy vs Chance of Forest Fire for a 10 year MDP

values associated with states. We can also increase the horizon of our valuation by increasing Gamma, the discount factor. Increasing Gamma to 0.95 from 0.9, in **Figure 2 c**, also dramatically increases the value and reduces cutting. This demonstrates the power of an MDP to explore different parameters to model outcomes and inform decision making.

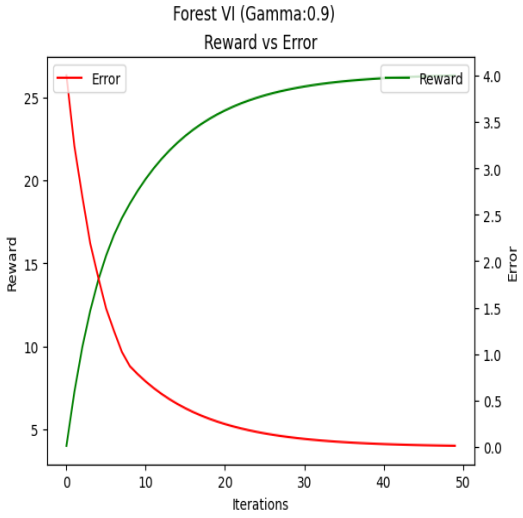
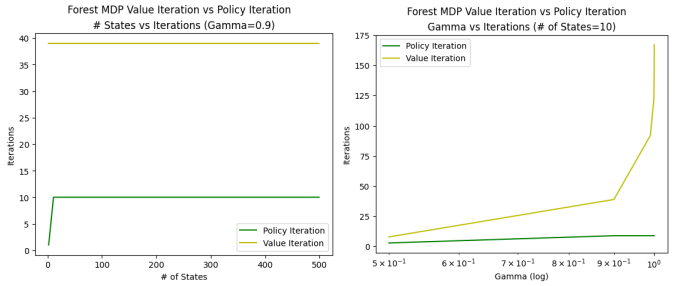


Figure 3: Forest MDP Value Iteration (VI) Error and Reward vs Iterations

From a technical standpoint, we are often interested in how quickly and accurately an algorithm performs. **Figure 3** shows the Reward and Error per iteration of VI, with smoothly increasing value and decreasing error. Error is the maximum difference between an iterations values and the previous iterations values for each state.



(a) Iterations vs # of States (b) Iterations vs Gamma Values

Figure 4: Forest MDP Iterations till Convergence

An important question is what impacts how many iterations it takes to converge? **Figure 4 a** shows the number of iterations needed to solve the MDP as a function of the number of states. When I first saw this, I thought it must be wrong! The number of states in the MDP does not impact the number of iterations to solve for both VI and PI. This triggered a lot of review and it turns out that the VI algorithm is using a very specific way to determine convergence.

$$K = \left[\frac{log\left[\frac{2R_{max}}{\epsilon(1-\gamma)}\right]}{log(\frac{1}{gamma})} \right]$$
$$if ||V - V^*|| < \epsilon \ \ then \ \ ||V^{\pi^i} - V^*||_\infty < 2\epsilon \frac{\gamma}{(1-\gamma)} \quad (2)$$

The top of **Equation 2** shows how the number of iterations, K, can be solved for depending on the maximum reward, the error threshold ϵ and the discount factor. It does not include any reference to state! The bottom part can be solved along with K to determine the number of iterations to produce a policy that is ϵ accurate, which is the effective convergence in the MDP VI algorithm.

Figure 4 b shows VI's sensitivity to $gamma$, with increasing values requiring more iterations to solve. Intuitively this makes sense with larger $gamma$ having a longer horizon, requiring many passes to reach the long term value. The amount of time needed for increasing $gamma$ closely follows this chart and is not reproduced.

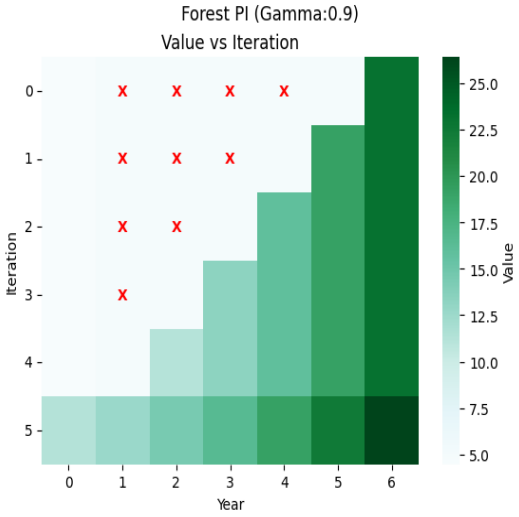


Figure 5: Forest MDP Policy Iteration (PI). Red X indicates the cut action, otherwise the action is wait.

Policy Iteration (PI) usually requires many fewer iterations than VI. PI's **Figure 5** shows this, with it converging in 5 iterations, which is many fewer than VI. **Figure 6** shows the reward and error curves per iteration, which are not smooth.

$$Value(s) = max_a Q(s,a)$$
$$Policy(s) = argmax_a Q(s,a) \quad (3)$$

PI starts with an initial settings, in this case with all state values set to 0. It then solves for the policy using a linear solver and the initial Values. With this new policy, it then recalculates all V values. The MDP toolbox implementation of PI actually solves for Q and then derives the policy and values from that as seen in **Equations 3**. It is considered converged when the error is 0. We will see in the ***Lake MDP***

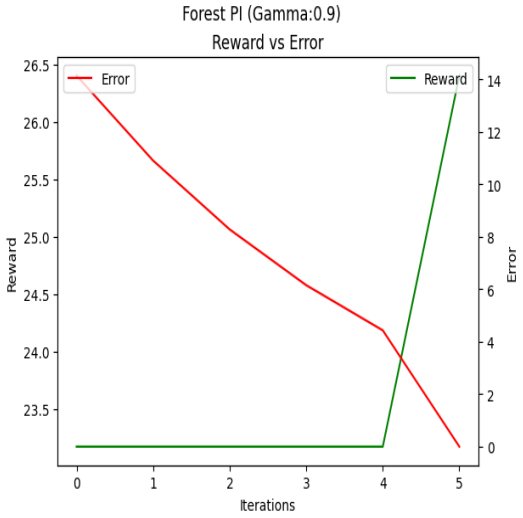


Figure 6: Forest MDP Policy Iteration (PI) Error and Reward vs Iterations

section that this can be problematic. In most cases, it exactly solves the MDP, while VI only approximates it to some error tolerance ϵ. VI and PI produced the same policies, but there were small Value differences, with these differences almost two orders of magnitude below the correct PI values.

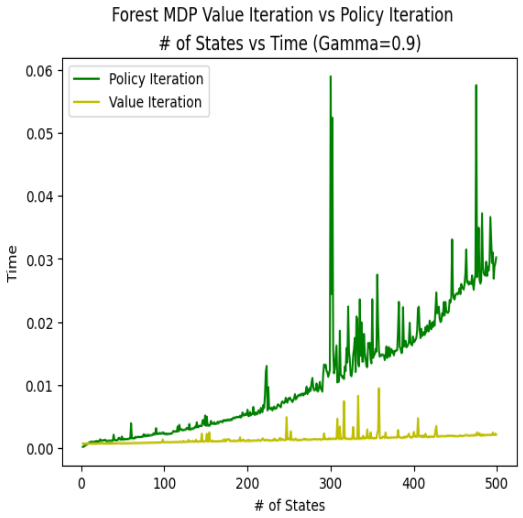


Figure 7: Forest MDP Time to Solve vs Number of States

As seen in **Figure 4**, PI requires fewer iterations to converge than VI and the number of states does not impact the number of iterations needed. It also is only slightly impacted by increasing Gamma (γ), while VI was strongly impacted. It looks like a clear winner. However, **Figure 7** shows the challenge with PI. Even though it takes fewer iterations, solving the linear equations is time consuming and very sensitive to the number of states.

Lake MDP Value Iteration (VI) and Policy Iteration (PI)

Figure 8 shows the ***Lake MDP*** solution evolution for the Medium Map for VI. This is a little clearer than the Large map for highlighting key elements. Like with the Forest, we can see as that with iterations, value spreads from its sources, in this case the Goal state in the bottom right corner. As the value spreads, the arrows which indicate the policies action direction change. The empty value spots are holes, with value 0 and policy directions point away from them where possible.

A key thing to note is the error threshold (ϵ) value (called E Stop on charts). When I initially compared the results of the VI vs PI for the ***Lake MDP***, I found a significant number of differences in value and policy. After digging in, I realized that the maximum value of this MDP was around .7. The E Stop (ϵ) value is an absolute number and the default of 0.01 was much higher than the expected values close to the start state, which accounted for the differences. By decreasing E Stop (ϵ), it allowed for more accurate convergence, seen best in the Large Lake MDP in **Figure 9**. Once again, small differences in parameters produce very different outcomes.

Figure 10 a is another take on how VI works, comparing Iteration 15 to 16 and highlighting the Value and Policy differences. The value changes slowly as it spreads from the reward state, changing policy along its change edge, but as it continues to change, we can see a second wave of policy changes. This is in contrast to PI in **b**, which moves dramatically with each iteration. **Figure 11** shows the Large Lake PI solution, which is the solved solution.

Figure 25 a in the summary section shows the difference on the large map between the VI and PI solutions. The value scale shows that the differences are small and increase farther away from the source of the reward, as well as one policy difference. This makes sense, because each iteration sweep is propagating the source reward outward, so the source reward becomes more attenuated farther away. These differences could be reduced further by lowering the error threshold (ϵ), but would require more iterations for VI.

One interesting thing was found regarding convergence for PI on the Medium Lake Map. When looking at number of iterations, it would use up to the maximum number of iterations (100), essentially not converging. **Figure 12** shows what was happening. The linear solver was solving very slightly different values, bouncing back and forth between two states, never reaching an error of 0. Notice the extremely small scale of change - 10^{-17}, which is a product of floating point math. A check could be added to this algorithm that the error doesn't need to be zero, but below some extreme amount, like 10^{-15}.

The VI and PI error and reward curves for the ***Lake MDP*** were similar in form to the ***Forest MDP*** and not reproduced in the paper.

Q Learning

VI and PI are model based algorithms which use the fully defined model for solving. This is not practical in many situations in the real world, where we are exploring an envi-

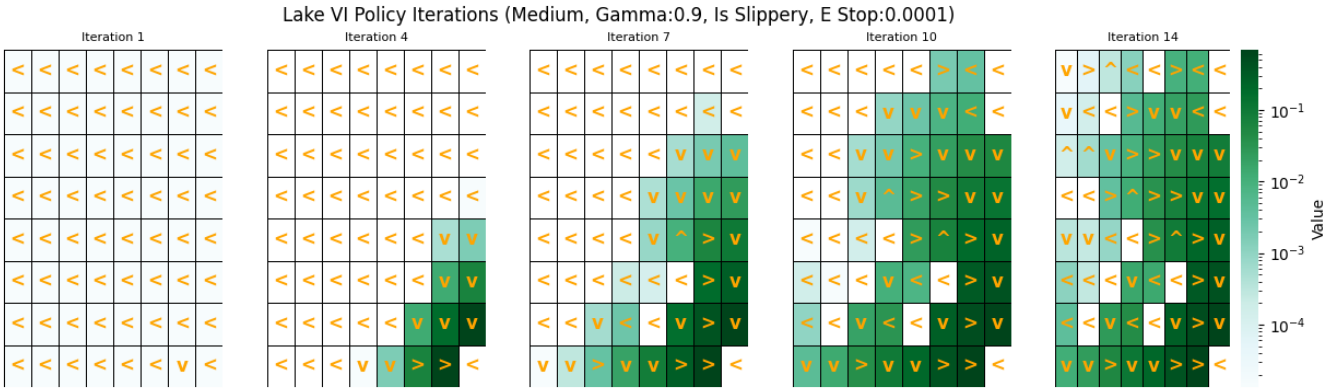


Figure 8: Lake MDP Value Iteration

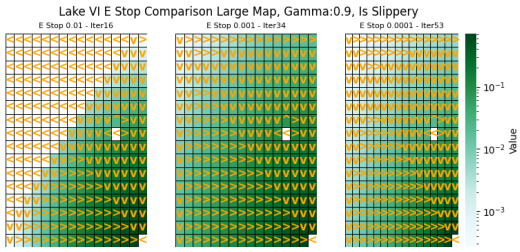
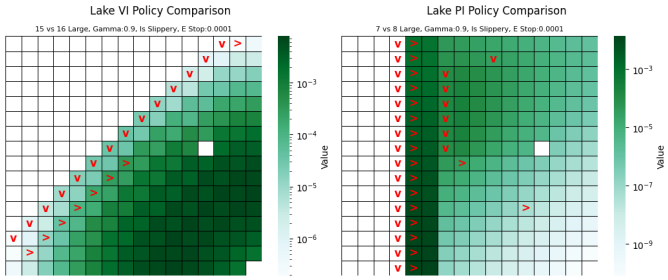


Figure 9: Lake MDP Different E Stop ϵ values



(a) Value Iteration (b) Policy Iteration

Figure 10: Lake MDP Comparing Iterations. Red Arrows indicate where policy changes and the green highlights how much the value has changed.

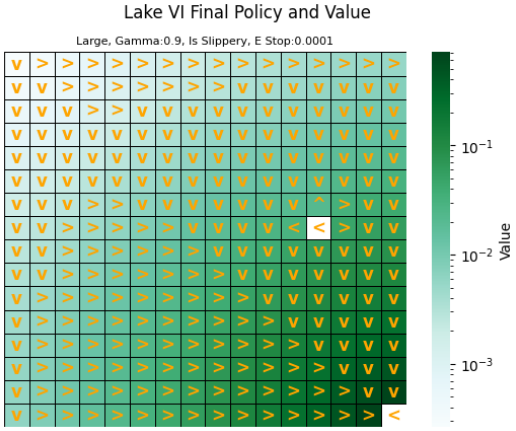


Figure 11: Large Lake PI solution

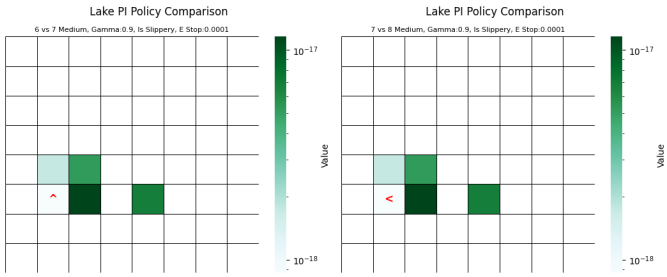


Figure 12: Lake MDP Policy Iteration Oscillating Non Convergence

ronment. Q Learning is a model free algorithm, which will learn a policy by interacting with an environment. This uses the Q form of the Bellman Equation, shown in **Equation 4**. One of the first key differences is that instead of sweeping all states in each iteration, now each iteration is one state, action, reward, next state step. We can expect more, shorter iterations. While ***Forest MDP*** is continual, we will also review episodes as another key concept in the ***Lake MDP*** section.

$$\hat{Q}(s,a) \xleftarrow{\alpha_t} R(s) + \gamma \, max_{a'} Q(s',a') \tag{4}$$

The next key concept is that of exploration vs exploitation. There is a tension between maximizing reward using what is know versus accepting a lower immediate reward in order to find better new rewards. A common model is to explore with chance ϵ or maximize with chance (1-ϵ). This is called an ϵ **Greedy** policy. By lowering ϵ over time by some decay factor, an environment can be explored and then gradually maximized. The decay used by the MDP algorithm is geometric, which decays by a certain percentage each step, subject to a minimum floor.

After reviewing the Q Learning MDP algorithm, I real-

ized that there was a randomizing step that every 100 iterations would restart in a random state. These random restarts significantly sped up exploration. I decided to remove this in my forked version because I thought it was "cheating". When considering an exploration environment, it seemed like a cop-out to magically teleport to any space, cutting out the often difficult exploration to find maximum reward.

The final key concept is that of the learning factor, α, which governs by how much Q is updated on each step. This also decayed over time to take big steps to learn quickly initially, but then taking much smaller steps to not be thrown around by stochastic actions or rewards.

We will see that Q learning requires substantially tuning of hyper parameters to get a good solution. In addition, we will explore convergence conditions using running reward and Error.

Forest MDP

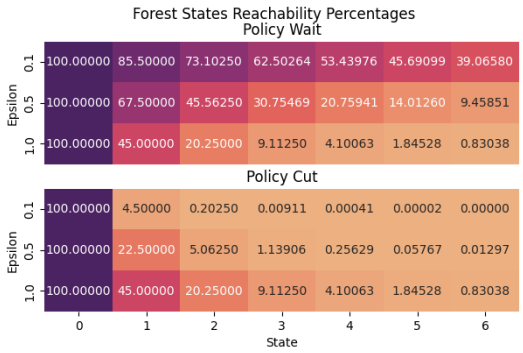


Figure 13: Forest MDP Exploration Reachability Percentages for Q Learning

The first challenge with the ***Forest MDP*** is that the maximum reward state is the far from the starting state at the end of a set of stochastic transitions. **Figure 13** shows the percentage chance of reaching each state for a given policy and ϵ. At the start of Q Learning, ϵ is 1 to explore. Here we see that we have less than a 1% chance of reaching the final state. What this means is that it will take a lot of iterations to eventually get information about the end state. In addition, until the end state is observed, policy will lean towards cutting, as that provides a reward.

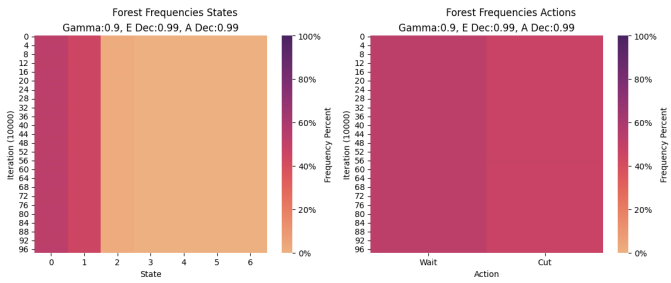


Figure 14: Forest Q, ϵ Decay of .99, α Decay of 0.99

Figure 14 shows the frequencies of actions and of visiting states at different iteration points. The iteration scale is in 10,000 increments, which shows that an increase in the number of iterations of 5 orders of magnitude versus VI and PI. Ouch! The left chart shows state frequency, which shows that all time is spent in states 0 and 1. The right shows that the actions are almost evenly split between cut and wait. The policy learned was to wait in state 0, cut in state 1.

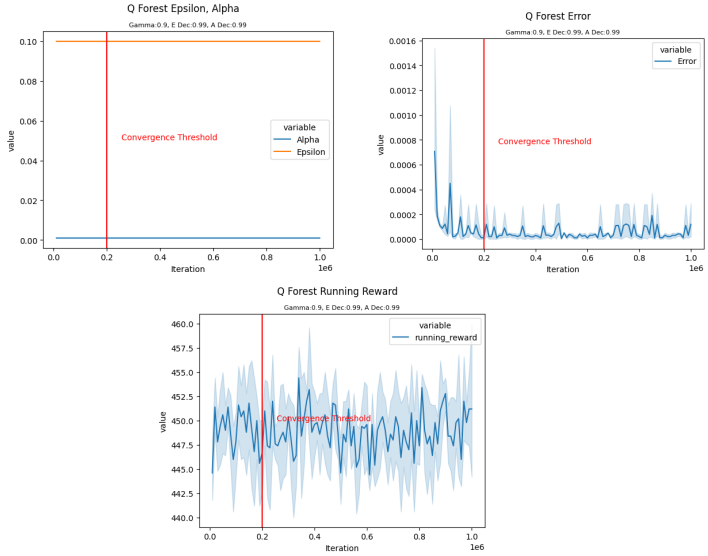


Figure 15: Forest Q, ϵ Decay of .99, α Decay of 0.99

Figure 15 provides insight. The left chart shows ϵ and α essentially unchanged with a decay of 0.99 that declines to its floor quickly. The middle chart shows the error at each given iteration. First, it is very low, which makes sense because only one state is changing per iteration. It is quite noisy, but does have a declining initial pattern. The bottom chart takes the cumulative reward over the last 1000 iterations at each point. This is the most important chart and shows how we are doing. With the initial settings the, the reward is very noisy and makes no improvement. I will describe the convergence threshold line in reviewing **Figure 19**.

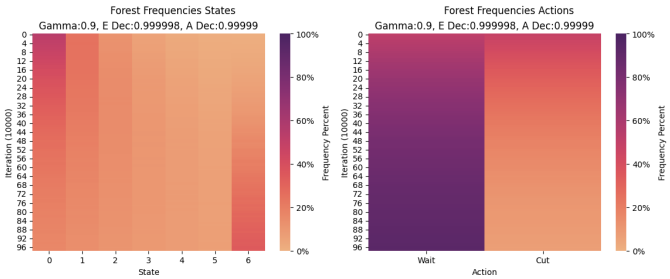


Figure 16: Forest Q, ϵ Decay of .999998, α Decay of 0.99999

Figure 17 shows the results of dramatically increasing the decay factor. The left chart shows that ϵ remains high

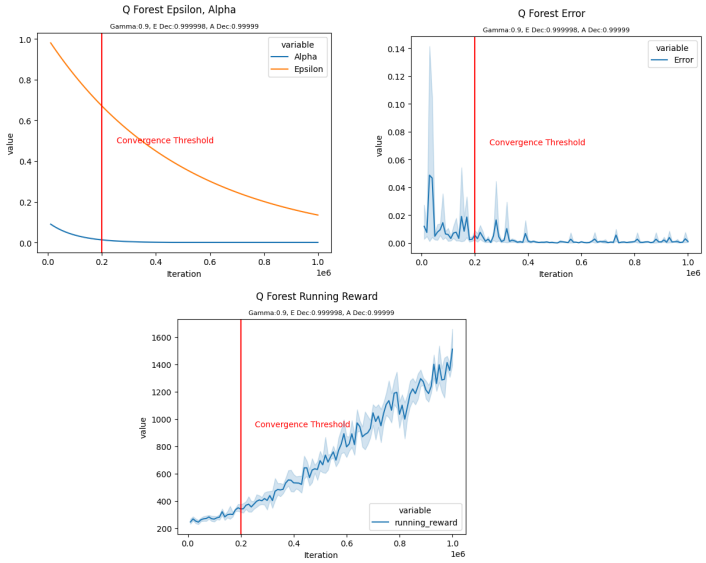


Figure 17: Forest Q, ϵ Decay of .999998, α Decay of 0.99999

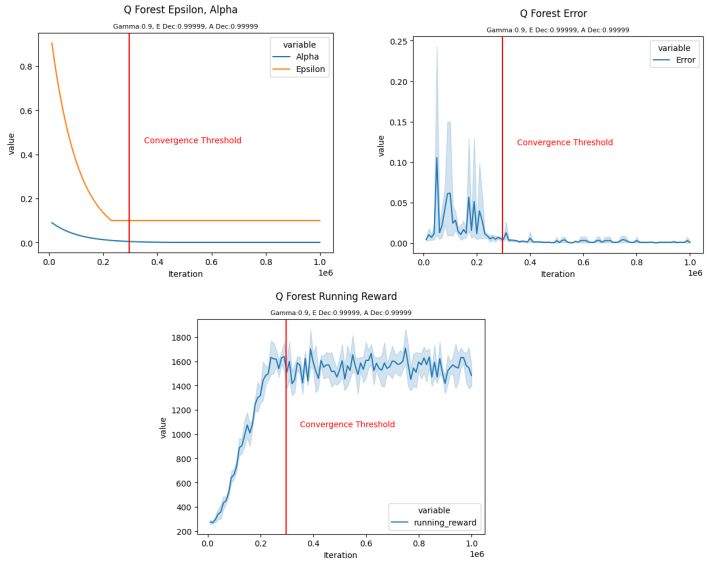


Figure 19: Forest Q, ϵ Decay of .99999, α Decay of 0.99999

throughout the experiment. Error is more dramatic, but shows the same pattern as before. The right running reward chart shows the key difference - a strongly rising running reward. **Figure 16** shows the shift from primarily states 0 and 1 to the final state as Q Learning goes on. Actions also show a strong shift towards waiting with more iterations.

The key observation is that running reward is still increasing, which is primarily governed by the ϵ driven exploration. The action frequency seems to indicate that policy is largely to wait, but exploration is still choosing both actions.

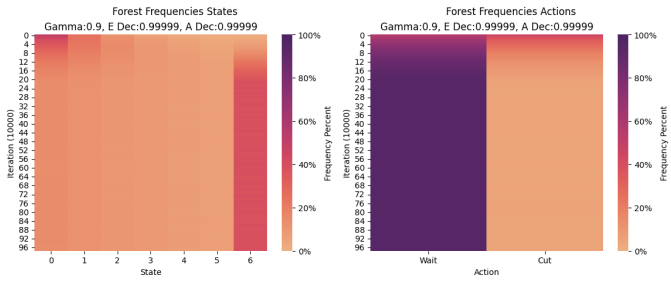


Figure 18: Forest Q, ϵ Decay of .99999, α Decay of 0.99999

Figure 19 shows a good overall solution for Forest by reducing exploration a bit. We see ϵ and α declining over about 300,000 iterations. Error declines as well. The running reward chart is key, showing us reaching a very value quickly, reaching a solved plateau. **Figure 18** frequency charts show action quickly moving to the optimal wait policy.

After looking at the different runs and output, I chose running reward as the convergence criteria, creating a threshold for when the running reward stops increasing. I used large windows and averages to prevent noise from triggering the threshold. I didn't move this convergence threshold back into the MDP toolbox, but it would be possible to do this. The running reward chart serves as the convergence chart for Q Learning.

Lake MDP

I was optimistic that ***Lake MDP*** would be an easy extension of Forest, but the first key issue was that Lake is an episodic MDP. Looking at the data, the state visit frequencies were almost all in the holes and the goal state. This was partially a function of removing the random reset of state every 100 episodes, which would have glossed over the underlying challenge. The challenge was that the transition model from Frozen Lake was that the transitions from a hole or goal was back to itself. The MDP implementation did not have a concept of terminal states. One change I could have made was to change those transition states back to the start, but there would not be a concept of episode. Instead, I created a modified episodic Q learning that takes a list of terminal states and tracks episodes. For Lake Q Learning, I used episodes instead of iterations for charting, which I thought was a better representation for episodic learning, although it makes comparisons between Lake and Forest a bit tricky.

With the episodic changes, the Small and Medium Lake maps were solved using ϵ and α decays from Forest. With the Large Lake map, I was not producing any useful convergence. I moved up to 100 billion iterations and extremely large ϵ and α decays, with no progress. Looking at the frequency charts, all visitation was close to the start. **Figure 13** showed the challenge of reaching end states in the Forest MDP. After consideration, with the highly stochastic actions and holes ending episodes, while I couldn't calculate it, the chance of reaching the end state in the Large Lake without random restarts was effectively zero. Ouch!

Based on this, I reduced the Large map size to 16 X 16. I was still not able to solve, so I started to remove holes until I started to get traction and used this map. I went back and

reran VI and PI using this as the new baseline Large Lake Map (*which is what is used through out the report now*). The Large Lake used 20 million iterations for the experiment runs, which is 6 orders of magnitude greater than VI.

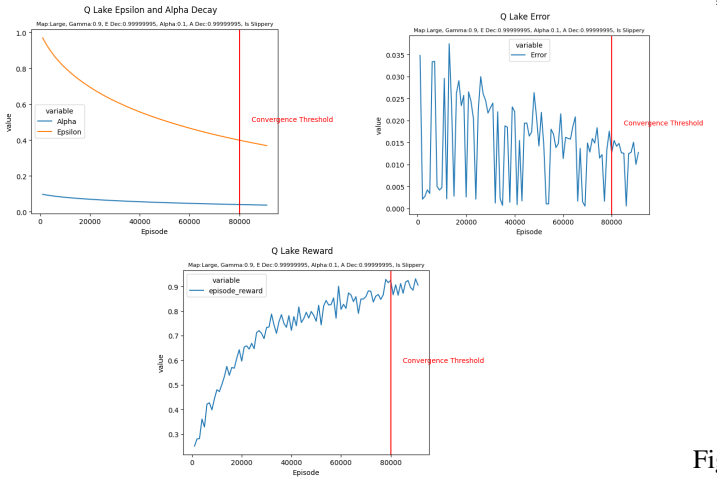
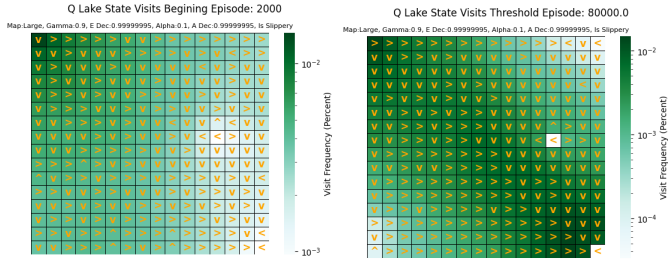


Figure 20: Lake Q, ϵ Decay of 0.99999995, α Decay of 0.99999995



(a) Initial Episodes (b) Threshold Episodes

Figure 21: Lake Q, ϵ Decay of 0.99999995, α Decay of 0.99999995

Figure 20 shows the first Large Lake run with some traction. The left chart shows that exploration is still high at the end, which is reflected in the right running reward chart. Even though it does trigger the threshold criteria, we can see an upward trend, mostly due to the continued exploration. The x axis for these charts is episodes, which is different from iterations used for all charts previously. The length of these episodes decreases over time as there is less exploration and a policy is followed. I will use iterations instead of episodes for comparisons in the summary section.

Figure 21 shows policy and the frequency that each state is visited in the last 1000 episodes. On the left, we see the chart after 2000 episodes. This shows a lot of general exploration originating out of the start location. The right chart shows the pattern at the threshold episode, with a clear pattern of moving across the center of the map. The policies in the top right and bottom left look a little funky and were probably not explored very much.

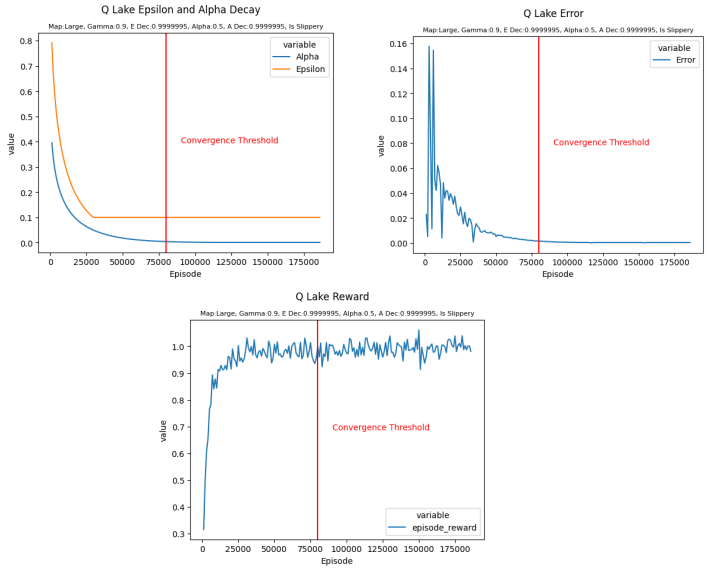
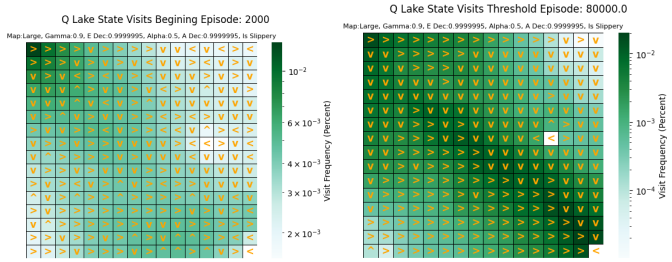


Figure 22: Lake Q, ϵ Decay=0.9999995, α Decay=0.9999995, Initial α=0.5



(a) Initial Episodes (b) Threshold Episodes

Figure 23: Lake Q, ϵ Decay=0.9999995, α Decay=0.9999995, Initial α=0.5

The goal was to reduce the exploration and learning to allow for a true convergence, with several iterations of parameter tuning. **Figure 22** shows the final Large Lake solution. The left shows the quick reduction in exploration. The initial α was also increased, increasing learning in the beginning to more quickly develop the best policy. The right chart shows a quick rise to the maximum reward per episode with a clear, converged plateau. **Figure 23** state visits map on the left shows a fairly established policy with more exploration around the goal state. The converged map on the right shows a much tighter path, with no exploration and a solid policy.

Figure 24 shows the final Large Lap Values and Policy. **Figure 25 b** in the summary section shows the difference on the large map between the Q and PI solutions. The value differences **decrease** farther away from the source of the reward, vs increase as in the comparison between PI vs VI. In addition, the differences are do not have a uniform gradient as in VI, but are patchy. In Q learning, the reward propagation is entwined in the evolving policy through stochastic exploration and learning.

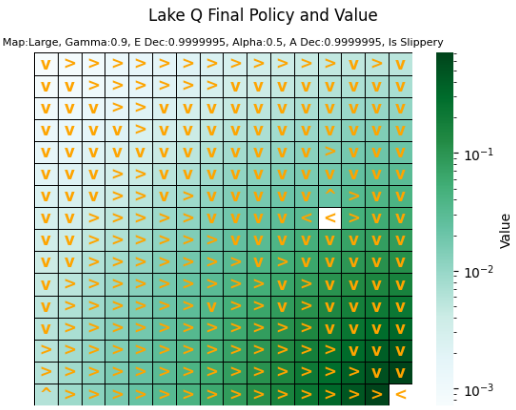


Figure 24: Large Lake Q solution

There are many more policy differences than between PI and VI which highlights that small value differences can create different policies. In general, the policy differences do not make a meaningful difference in outcome. When we look at the differences around the goal in the bottom right corner, down action or right action policy on the diagonal axis won't make a difference as they both move towards the goal. There are many policies in a large stochastic environment which will produce similar outcomes.

Summary

Table 1 reviews the performance for VI, PI and Q Learning for the Forest MDP. PI's time is only slightly better than VI, but iteration count is much lower. Q Learning is more than four orders of magnitude slower. Iterations in VI and PI perform a sweep on all states, while Q Learning iterations only update one state. To provide a more even comparison, the Equalized column normalizes this by dividing Q Learning iterations by 7, the number of state updates in a VI or PI iteration. Q Learning uses more than four orders of magnitude more equalized iterations than PI.

From the performance numbers it would seem to always use PI, but Q learning is likely to more useful in many real world situations. Often, we don't know what the environment is and are truly trying to figure it out and will need a model free algorithm, like Q Learning. Also, in continuous environment, you can't sweep all states. Q can be modified from a table model to a function approximation, often using a deep neural network.

	Time	Iterations	Equalized
PI	0.00078	6	6
VI	0.00083	50	50
Q	3.98869	296,500	42,357

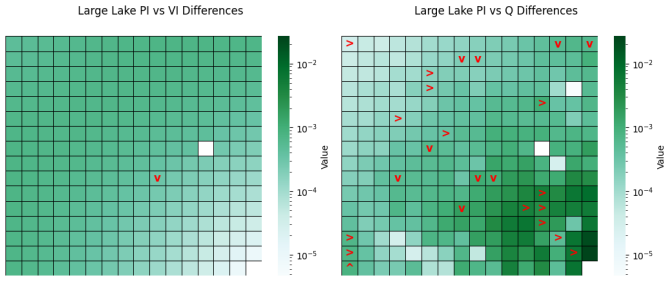
Table 1: Forest Algorithm Comparisons. Equalized Q are iterations divided by 7 for better comparison with VI and PI

Table 2 reviews the Lake MDP. Here we can see that VI uses less time than PI. Earlier, in **Figure 7**, we saw that PI needed more time to solve as more states were added. In the Forest MDP, the small number of states neutralized this issue, while the Large Lake had many more states making VI the faster of the two algorithms. Q learning if 5 orders of magnitude slower but only uses four orders of magnitude more iterations.

	Time	Iterations	Equalized
PI	0.0656	17	17
VI	0.0036	53	53
Q	187.0892	9,603,499	37,513

Table 2: Large Lake Algorithm Comparisons. Equalized Q are iterations divided by 256 for better comparison with VI and PI

As discussed earlier, **Figure 25** shows the differences in PI vs VI and PI vs Q Learning for the Large Lake MDP. PI provides an exact answer as the baseline. Values far from the reward goal state can be different depending on error factor ϵ for VI. Q Learning shows more differences impacted by exploration ϵ decay, initial learning rate α and α decay. However, in both cases, the different policies produce almost equivalent outputs, highlighting that in many MDPs, there can be many policies that are practically optimal.



(a) PI vs VI (b) PI vs Q

Figure 25: Large Lake Value and Policy Comparisons

All MDPs are very sensitive to the time horizon that is being considered, which is governed by the discount factor gamma γ.

Algorithm Review

Policy Iteration (PI) required no tuning and would return the exact answer. The MDP Toolbox iteration is susceptible to oscillation due to floating point math issues, but this could be resolved. PI slows down exponentially with more states, but there are enhanced versions that can minimize this issues through different solving techniques.

Value Iteration (VI) does require some tuning of the error threshold ϵ parameter and returns an approximate answer. The threshold ϵ needs to be thoughtfully set because it can have significant impact on the outcome, as seen initial runs in the Lake MDP.

Q Learning is quite different from VI and PI as it is model free. It returns an approximate answer and requires

careful tuning of several parameters and uses several orders of magnitude more iterations and time. A key issue is the tension between exploration and exploitation. Exploration is needed to discover how the environment works and avoid local maxima and is governed by how fast ϵ is decayed. The Forest MDP demonstrated the need to continue to explore past the immediate cutting rewards. The Lake MDP demonstrated how large state spaces require a lot of iterations and exploration to even find initial rewards. Q Learning also is sensitive to the learning rate α. Q Learning requires significant review to solve with a great deal of interplay between the parameters and environment. Also, unlike VI or PI, Q Learning is also sensitive to how challenging the state space is. VI and PI could solve Large Lake with any amount of holes, while Q Learning would require significantly more time to explore a Lake with a lot of holes.

One thing I should have done with Q Learning was to initialize the MDPs with small random values at the beginning to seed exploration. By not doing this, particularly with the Lake MDP, I created a strong bias away from the goal state, which then needed to be overcome.

References

1 Markov Decision Processes (MDP) Toolbox, https://miat.inrae.fr/MDPtoolbox/ Accessed: 11/1/2022.

2 Markov Decision Process (MDP) Toolbox for Python, https://github.com/sawcordwell/pymdptoolbox: Accessed: 11/1/2022.

3 hiive Fork: Markov Decision Process (MDP) Toolbox for Python, https://github.com/hiive/hiivemdptoolbox: Accessed: 11/1/2022.

4 Brockman, G. et al., 2016. Openai gym.

5 Openai gym Frozen Lake Documentation, https://www.gymlibrary.dev/environments/toy_text/frozen_lake/: Accessed: 11/1/2022.