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".

Problem Introduction

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.

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*** and provides a ***cutting reward***. 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 very 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.

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*** and can be of ***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.

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 an environment 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.

Both VI an PI rely on the Bellman Equation, which can be represented in different ways as seen in **Equation 1**

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

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 to determine error. I will refer to this error threshold as the *E Stop* value. In some discussions it is also confusingly referred to as ϵ, which 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 emphError Threshold values aren't reached for much later. 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 optimal policies more likely to increase cutting. One way to reduce cutting is to increase the

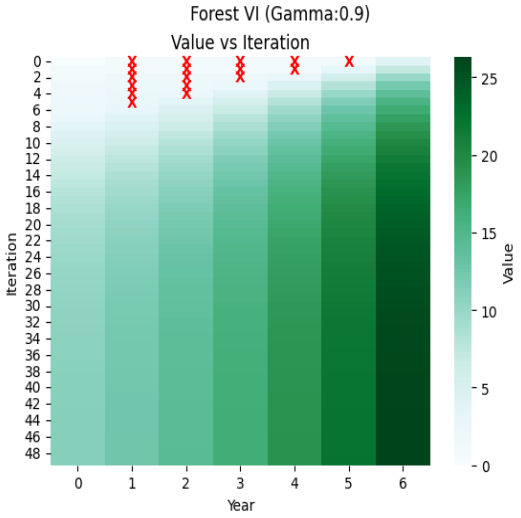
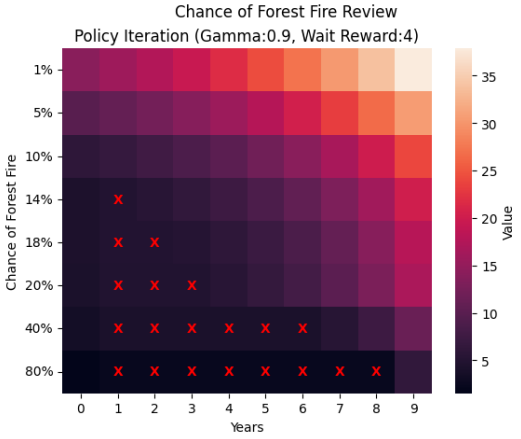
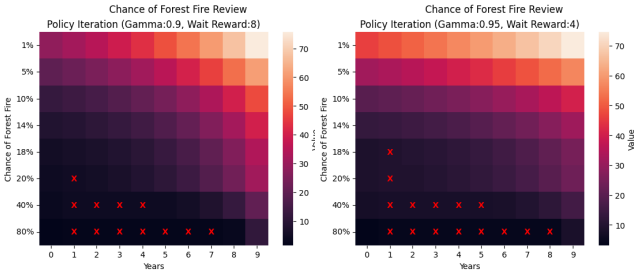


Figure 1: Forest MDP Value Iteration (VI) with *Error Threshold=0.01*. Red X indicates the cut action, otherwise the action is wait.



(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

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 values associ-

ated 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.

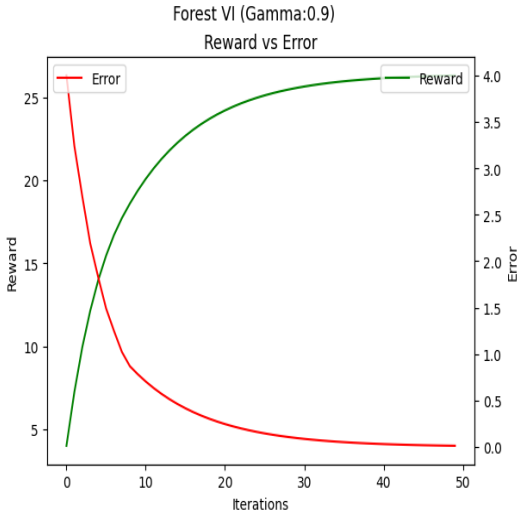
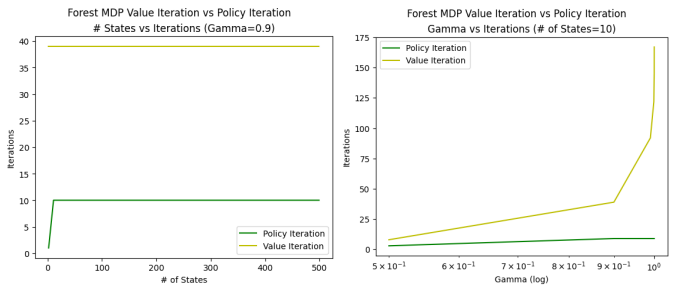


Figure 3: Forest MDP Value Iteration (VI) Error and Reward vs Iterations

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.



(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.

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

The top of **Equation 2** shows how the number of iterations, K, can be solved for depending on the maximum reward, the error ϵ (not exploration yet) 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.

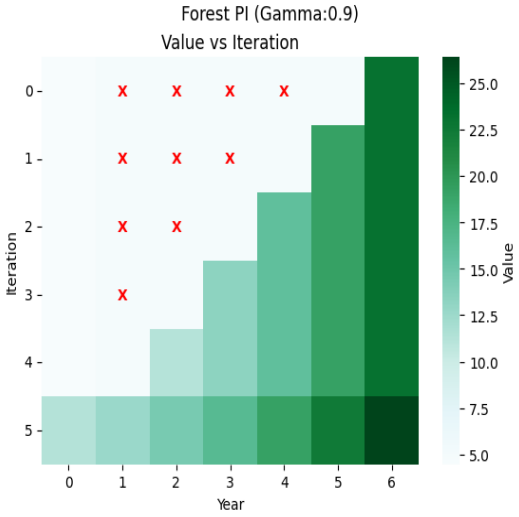


Figure 5: Forest MDP Policy Iteration (PI). Red X indicates the cut action, otherwise the action is wait.

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.

$$Value(s) = max_a Q(s,a)$$
$$Policy(s) = argmax_a Q(s,a) \tag{3}$$

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 using the Bellman Equation **1**. This implementation of PI uses the Q update rule in **Equation ??**, and then derives the policy and values from that as seen in **Equations 3**. It is considered converged when the error is 0.

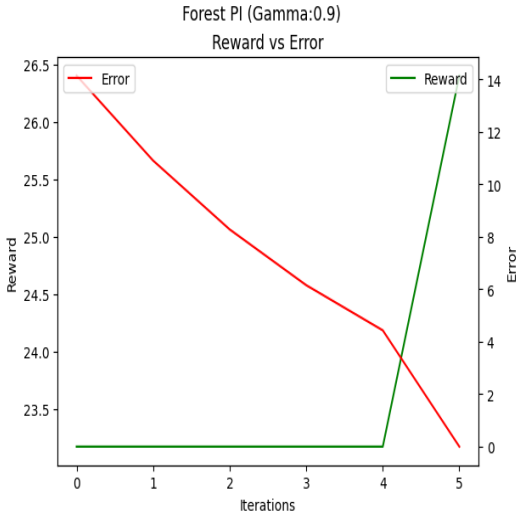


Figure 6: Forest MDP Policy Iteration (PI) Error and Reward vs Iterations

We will see in the ***Lake MDP*** section that this can be problematic. In most cases, it exactly solves the MDP, while VI only approximates it to some tolerance. 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.

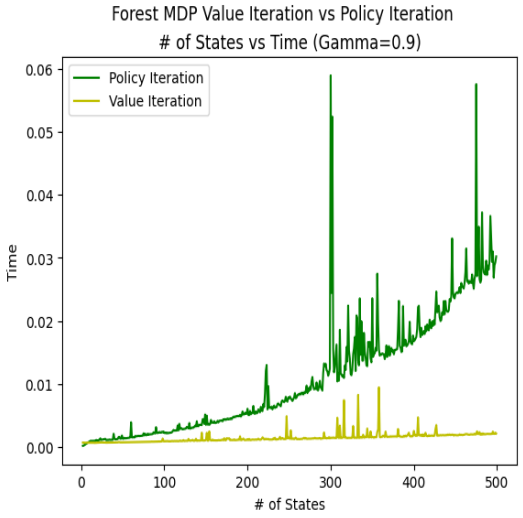


Figure 7: Forest MDP Time to Solve vs Number of States

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*** for the Medium Map. 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 E Stop (ϵ) value. 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 value is an absolute number and the default of 0.01 was much higher than the expected outcomes close the start state, which accounted for the differences. By decreasing E Stop (ϵ), it allowed for more accurate convergence, seen in **Figure 9**, creating only small value differences between VI and PI. 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 difference on the large map between the VI and PI solutions. The value scale shows that the differences are small, but increase farther away from the source of the reward, as well as one policy difference. These differences could be reduced further by lowering E Stop (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.

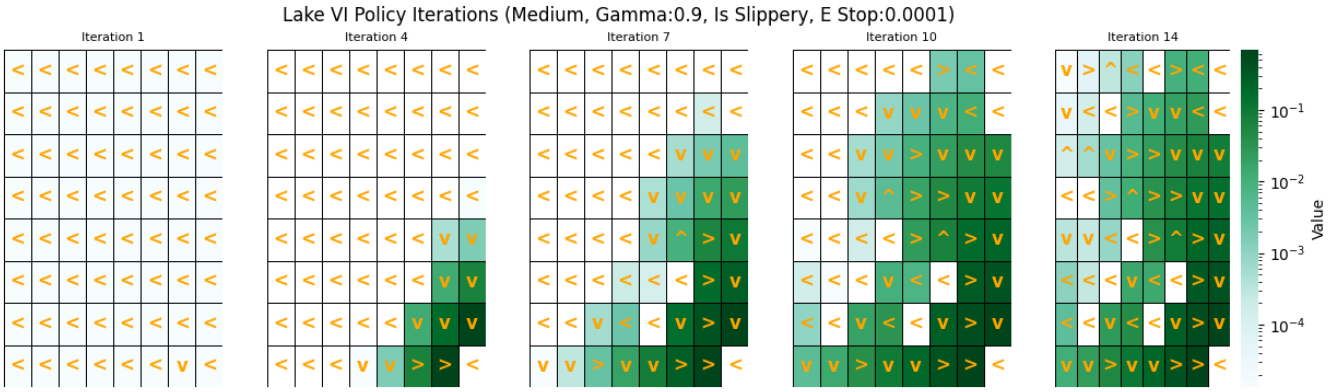


Figure 8: Lake MDP Value Iteration

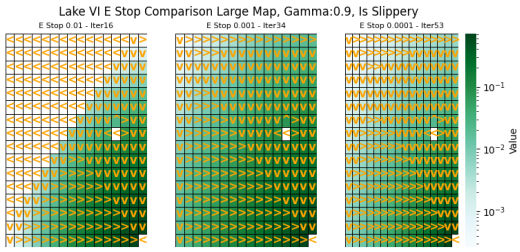
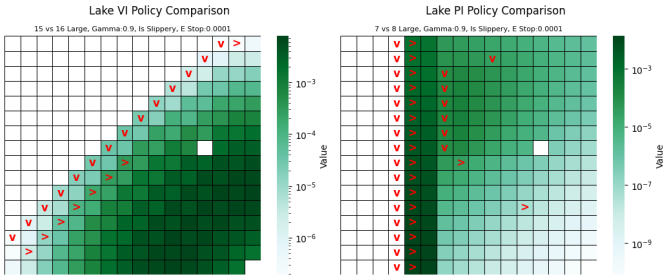


Figure 9: Lake MDP Different E Stop ϵ values



(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.

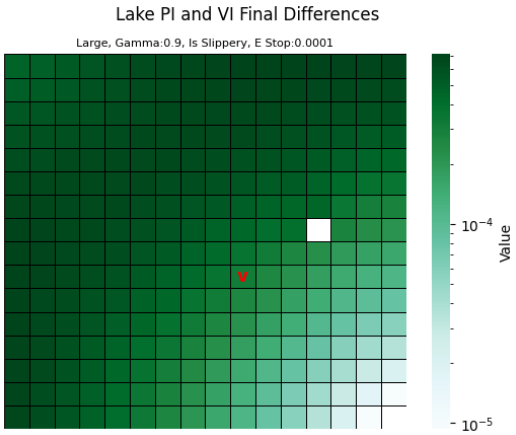


Figure 11: Lake MDP Difference between VI and PI solutions

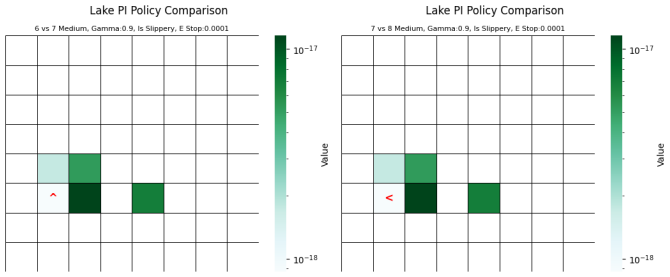


Figure 12: Lake MDP Policy Iteration Oscillating Non Convergence

Forest MDP Q Learning

Lake MDP Q Learning

Summary

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.