# OMSCS 7641: Machine Learning

*Markov Decision Processes*

Github Repo: https://github.com/TheOriginalAK47/omscs_cs7641_spring (assignment4 branch)

Andrew Kogler (akogler3@gatech.edu), 4/11/2020

## INTRODUCTION & PROBLEM BACKGROUNDS

In this assignment centered on Markov Decision Processes and Reinforcement Learning to a lesser extent, I focused on two problem domains, one pertaining to a grid-world representation, and one a non grid-world example: The Frozen Lake problem, and the Forest Management problem. Both of these are common examples used in Markov Decision Problems and come as part of the two Python packages I employed to implement Value Iteration, Policy Iteration, and Q-Learning solutions to the two respective problems, those being pymdtoolbox and openai-gym. I opted for the Forest Management problem to be my large state-space problem as the computations for this problem were more manageable on my local machine. For the purpose of introduction I want to give a short description of each problem before proceeding to my solutions and results in implementing these solutions.

The Forest Management problem is a problem representation wherein an agent who hopes to harvest as much lumber as possible needs to devise an optimal strategy/policy for procuring as much lumber as possible, with the likelihood of forest fires, the reward value of cutting trees, the number of states, and waiting some time frame serving as tunable parameters for the purpose of the problem. Producing an optimal strategy is non-trivial as one wants to find a strong balance of cutting down trees with a significant amount of lumber on them given they've been able to grow for a sufficient amount of time, taking into consideration the risk of the forest being burned down in the unlikely event a fire breaks out and burns down all trees present. As defined in the assignment requirements, I applied Policy Iteration, Value Iteration, and Q-Learning to come up with the optimal procedures for procuring as much lumber as possible.

On the other hand, the Frozen Lake problem is one in which an agent attempts to navigate from a starting position to a final position without falling in ice-holes in which taking actions are probabilistic in nature due to the slippery texture of the ice itself. The agent is rewarded for steps made that are closer to the final position and rewarded handsomely for reaching the final state, and punished heavily upon falling in an ice-hole representative of death or failure.

## FOREST MANAGEMENT

As mentioned in the introduction there were three main parameters that could be assigned in the configuration of the specific Forest Management problem representation which were S, r1, r2, and p which reflect:

- S: Number of States
- r1: Reward when forest is in it's oldest state and action 'Wait' is selected
- r2: Reward when forest is in oldest state and action 'Cut' is selected
- p: Probability of a forest fire occurring and returning all trees back to their youngest possible state

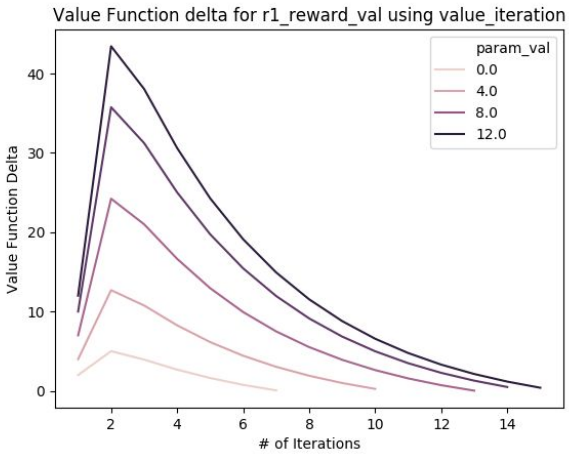
Also, as stated before I ended up using the pymdtoolbox [1] library and the forest() method in particular to speed up construction of this problem representation. I ended up keeping S static across all exploration strategies and opted to set S = 5000 which I thought should be a sufficiently large enough number of states to qualify this implementation as my large state-space problem. Alternatively, I ended up running all three optimization strategies for ranges of parameter values according to the below table:

| Parameter Name | Parameter Value Space |
|---|---|
| r1 | [2, 4, 7, 10, 12] |
| r2 | [1, 2, 4, 7, 10] |
| p | [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7] |

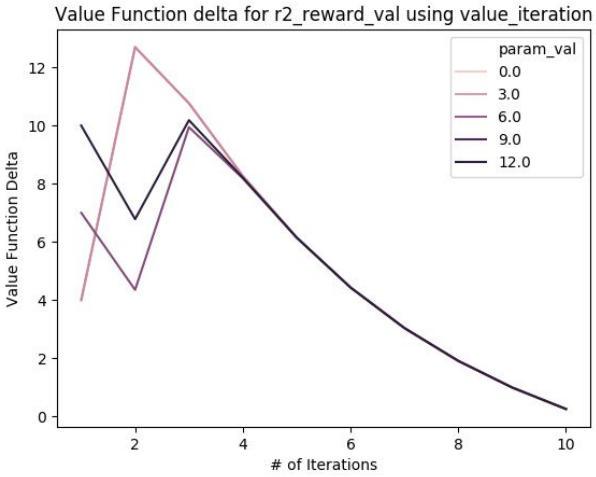
I restricted p at 0.7 as I noticed quite quickly when the probability of forest fire gets exceedingly high, the optimal policies become quite uninteresting where the agent is incentivized to cut down trees as quickly as possible. I then proceeded to plot the deltas in the value functions for these parameter values keeping all other parameters default to isolate the specific parameter's influence and see how it affects the line-plot of the value function deltas, speed of convergence, and the number of iterations to ultimately converge. For Policy Iteration and Value Iteration I decided to define convergence based upon an epsilon value of 0.01 or when a particular iterations change in the value function was less than 0.01 indicating the strategy or policy reaching a state of equilibrium.

## VALUE ITERATION

In this first plot I perturbed the r1 value so as to oscillate the degree to which the agent is rewarded for waiting when the forest is in it's oldest state. Each line in this plot pertains to a different value for the r1 parameter with the legend somewhat mis-mapping as some of the values were not in the parameter space which is an idiosyncratic reflex of seaborn in this case. Focusing on the properties of the plot there are a few conclusions to make. Firstly, all of the curves experience an initial uptick in their first few iterations met by an apex, and followed by a decay until they saturate and reach equilibrium. This is to be expected as the agent likely learns the effect of the parameter values early on and responds accordingly with their successive actions upon the environment. Theoretically, as the reward for waiting decreases, the agent should respond by cutting trees more and more proactively as r1 gets smaller and smaller.

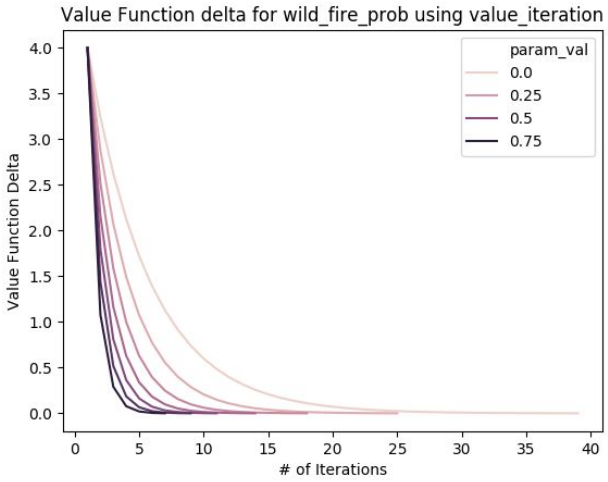


This is easily explained as if the reward for cutting down the tree (default value of 2 for all of these runs) begins to outweigh that of waiting, one would be incentivized to not wait and instead cut down trees and reap the higher reward, waiting as little as possible as r1 approaches 0. This is also corroborated by the shorter and shorter number of iterations to reach equilibrium as the agent notices this trend sooner and sooner as the reward value gets smaller and smaller where there smallest reward correlates to the earliest termination of the Value Iteration technique. The sizes of the peaks in relation to each other in absolute terms is unimportant I think though as that is more a function of the absolute value of the parameter value assigned to r1 at the end of the day.




Continuing onto the second plot where I varied the r2 parameter which represented the reward amount for the agent in which they opted to perform the ‘Cut’ action on the forest. Before immediately addressing the plot, we should expect the agent to chop down trees more immediately as the reward for cutting increases to a point where trees are cut down as soon as they sprout in the extreme situation. If the reward for cutting a tree on the other hand is arbitrarily low in comparison to the ‘Wait’ action (default value of 4 assigned to ‘Wait’), then the agent should otherwise be motivated to be more patient in order to reap a higher reward over the long term and only cutting when the forest is at its oldest state and waiting no longer yields returns.

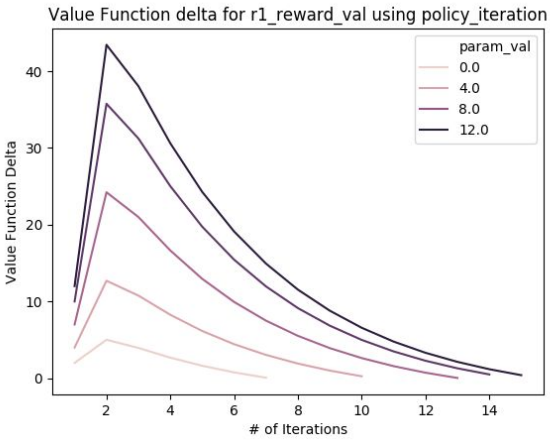
Similar to the behavior exhibited in the varying of the p1 parameter, we also see an early peak in the value function improvement early on before it decays down to a saturation point. Unlike before, we see a more or less merging in the lines though as they appear to function identically after 4 or so iterations at which point the parameter has no effect on the improvement of the value function over time which is interesting. I suspect this is due to the procedure crossing a tipping point wherein the agent predominantly selects one action over the other action due to the disproportionate amount of reward associated with one over the other. This is a learned habit after a small number of iterations where no further learning is achieved in relation to a more sophisticated policy or set of actions and the agent has adopted a greedy gradient-descent like strategy where there’s no incentive to deviate. Also of note is the lesser number of iterations needed for the algorithm to converge upon the optimal strategy which I think could be due to the somewhat stronger influence of this parameter in relation to r1.



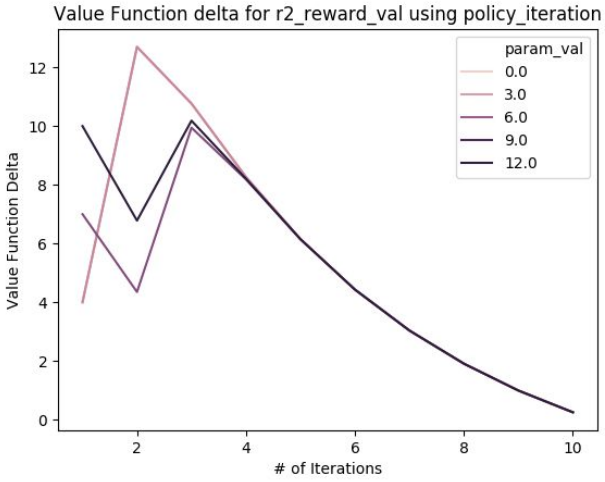

Lastly, I plotted the changes in the value function in relation to the likelihood of a forest fire breaking out in a particular iteration and rendering the forest back to its original state of only freshly sprouted trees. Here we see more distinct behavior with no early uptick but instead near exponential decay towards value function saturation with this occurring later and later based on the likelihood of a forest fire. The rate of the curve decay appears to be proportional to the more and more likely a forest fire is to occur, flattening the entire forest. This is intuitive as the increases in the value function should lessen and lessen if fires become more frequent since the potential reward to be reaped from a young forest is at its minimum. Also, it makes sense that the configuration where p is at its smallest value of 0.1 should take the highest number of iterations to terminate since the environment would need to endure a larger number of sequences to observe enough episodes of forest fires to learn how this impacts optimal policy/value selection.

## POLICY ITERATION

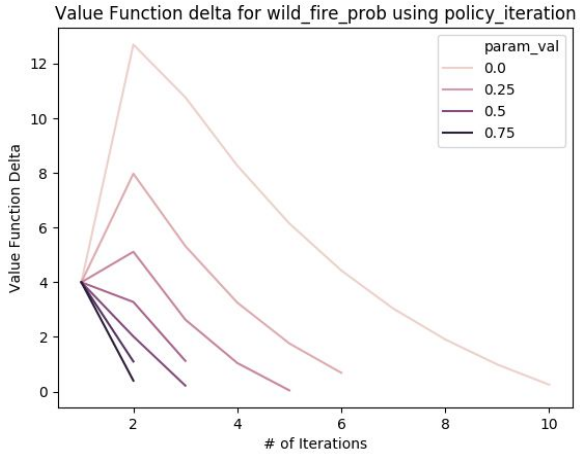
I then continued to re-run the same experiments with the sweeps over the same parameter spaces, instead using the PolicyIterationModified class with an epsilon value of 0.01, which has an identical stopping criterion as ValueIteration for convergence purposes. The plot where r1 was varied, this time using Policy Iteration can be observed below:



Interestingly this plot is virtually identical to the ValueIteration version of the experiment. I'm not entirely certain of why the curves are so similar in fashion. I think this could be due to the types of techniques being irrelevant to how the value functions are maximized or it could be the coincidental case that the initial optimal policy discovered by the ValueIteration version was identical in form to the optimal policy arrived at after policy improvement and final evolution. This could be the case but seems unlikely given the amount of randomness involved in addition to the short timespan with respect to the number of intervals that this occurred in but I don't think it can totally be ruled out of the equation. As these two techniques are simply different approaches to arrive at what could be the same answer/strategy it is technically possible and perhaps even likely in cases where the action/policy space is fairly sparse.




Continuing onto the plot for the r2 parameter we see pretty much the same trend as we just saw for r1. The curves are identical in form, terminate in the same number of iterations, converge after the fourth or so iteration, etc. This is strange to occur for two different parameters at this point. I went through my code even to double-check for errors where I was writing the same output data via DataFrame to seaborn and even re-ran the script multiple times. The two aforementioned plots persisted in character and point to either an underlying bug in my code or the coincidental settling of both techniques on equal policies/sets of actions that resulted in identical responses in the value function.



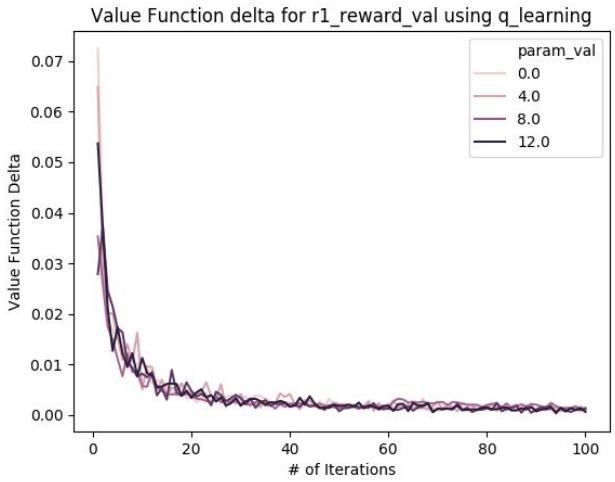
In the last plot above where we vary the p parameter we finally see marked differences in the output as compared to the ValueIteration version of this plot.

Here we can see noticeably different curves than what was the case for in the ValueIteration version of this experiment. Where before we saw lines approximating exponential decay, here we have early increases, followed by the decay property but the rate of decay is much less, and the number of iterations is also much less than before. Also, the gap between curves for the respective p-values are much more discernible and make it more evident how much of a difference this p-value does affect the policy search and evaluation process. Like before, the more likely that a forest fire breaks out per sequence, the less number of iterations we need to simulate as the policy learning procedure is expedited having experienced fires at a faster pace and making policy updates at a higher velocity. At a higher level, we see termination occur much earlier when employing PolicyIteration which could be evident of this procedure latching onto the abstract pattern earlier than in the case of ValueIteration where ValueIteration sort of blindly chases value deltas as opposed to more broad conclusions and strategies that are commonly associated with PolicyIteration.

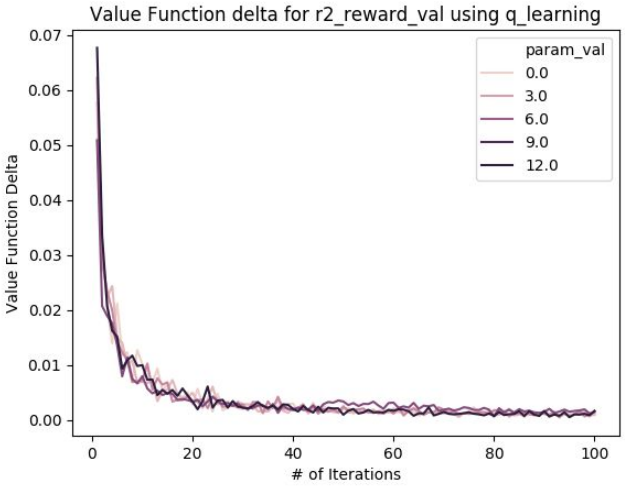
## Q-LEARNING

For the Reinforcement Learning portion of the assignment I opted to use Q-Learning for my algorithm to compare and contrast its performance on the two MDP's with respect to PolicyIteration and ValueIteration. I chose Q-Learning due to my previous exposure to it in both the Artificial Intelligence and Machine Learning for Trading courses that I've taken in the past through the OMSCS program. When it comes to analyzing the performance of the QLearning technique in pymdptoolbox, I used the closest proxy I could to changes in the value function which was the mean_discrepancy property on the QLearning object. This returned vector contains the resultant values for the first 100 iterations so it may actually contain more observations than are relevant to when the algorithm has converged and so some of the tail values may be negligible as part of the appropriate interpretation.

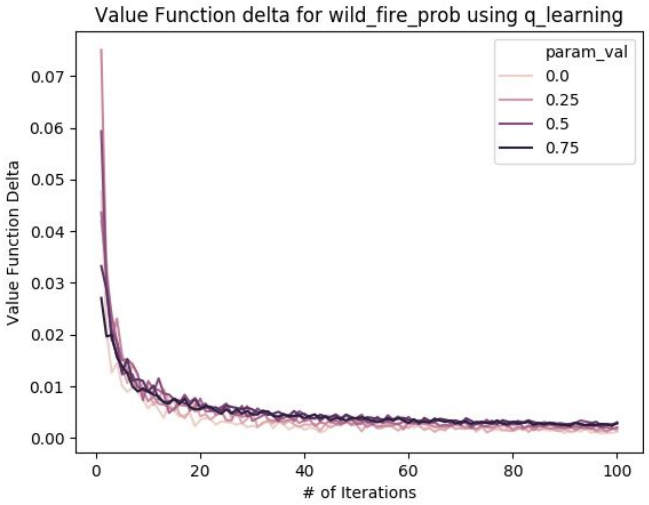
These curves are fairly different from the two examples we've seen previously with the lines possessing much more variance than what we saw before. While the curves exhibit pretty clear patterns they have fairly occasional spikes especially in the earliest iterations although the signal does mostly dampen out near the final iterations. Like before we the overall decaying trend but no initial jump like before. Another discernible difference is the degree to which gaps evaporate in this plot and in fact how closely each of these curves hug each other regardless of the particular parameter value. This could be attributable to the difference in the functional behavior of this property versus that of the value function deltas we were able to capture before. Based on the description in pymdptoolbox I'm led to believe these are functionally equivalent to each other so without looking at the source code my hunch is to treat them similarly. Some of the characteristics of these curves can be explained by a modicum of understanding of how Q-Learning works. For example, the variance in these curves is likely due to the hallucination episodes that the learner experiences and in particular how those hallucinations deviate from the traditional episodes. The relative speed of convergence (it appears to be in less than 10 iterations as nearly all curves are below 0.01 at that point, which is less than the epsilon value I set earlier) is likely also due to the amount of learning that is achieved through these "hallucinations" as well. Overall, it appears that Q-Learning converges at the same speed if not quicker than the previous two techniques which is an important finding.




Now in looking at how the second parameter, r2, affects changes in the mean variations property of the Q-Learning object.




We see largely the same curve as before with some subtle differences. Convergence appears to happen slightly later, at just over two iterations, and also the gaps between curves seem a bit more pronounced than before. The variance of curves that aren't represented by the r2=12 value have more variance than is observable in the first plot which is interesting on a relative note. Like in the previous plot we don't see an initial spike and then a decline which is novel. This is likely due to the hallucinations being a random sample of the action space as opposed to a more guided procedure which would more likely result in a large improvement in the value function. If we decide at random between waiting and cutting without much attention paid to the age of the forest, then there likely won't be a huge impact on the value function. An early spike in this case would be lucky in that the algorithm would've chosen a random action that actually was coincidentally the action that prompted the biggest improvement in the value function.




I then performed the last experiment for the forest management scenario where I applied Q-Learning but where the likelihood of a forest fire, as represented by p was varied for each run. The curves look fairly similar to the first two but with small differences in the most extreme p values displaying as such. First, the curve corresponding to the lowest probability of forest fire sees the biggest change in the value function on the first iteration, with the spike proceeding to be smaller and smaller as the p-value increases. This makes sense, as on the average the amount of reward that can be obtained from the forest on the first iteration is dependent on the likelihood that the forest will even be able to age after the early iterations of the simulation in this environment. When this chance is quite low, we have a significantly higher expected value as is evidenced by the very early data points present in this graph. Also, we see noticeable gaps among the trends later on in the graph which are viewable even after some 60 odd iterations which wasn't as evident in the previously generated plots. This could be motivated by the two main stochastic variables at this point p, and the sampled state spaces that are generated via the learner's hallucination episodes.

## FROZEN LAKE

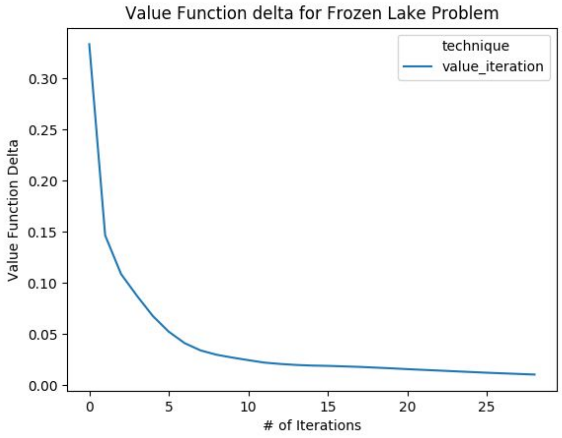
For the purpose of implementing the Frozen Lake problem as my grid-world problem instance I resorted to using the OpenAI Gym [2] library which offers utility methods and classes that help in standing up the Frozen Lake environment quickly. I was able to successfully implement both Policy Iteration and Value Iteration for this environment while I was not able to complete Q-Learning due to time constraints.

In implementing both of these procedures I followed a tutorial as offered by O'Reilly [3] which gave a really good walkthrough of how both these techniques function in general, in addition to specifically when applied to the Frozen Lake problem, as well as general background around the Frozen Lake problem itself.

Since the Frozen Lake problem doesn't have as many free parameters as the Forest Management problem I opted to keep all common parameters constant and worked on the stochastic version of this problem as opposed to the deterministic flavor. I kept epsilon equal to 0.01 like before.

## VALUE ITERATION

I first ran the experiment for the Value Iteration technique on the Frozen Lake problem which yielded the below plot as it relates to changes in the value function. It closely resembles those exponential decay curves we witnessed before with the r1 and and r2 parameters for both PI and VI on the Forest Management problem. While the shapes are very similar, the rate of decay is faster and the number of iterations that were experienced at which point the algorithm achieved convergence was much higher. This indicates that while learning gains are quite rapid, this plateaus quickly and takes a surprising amount of time to finally cross the convergence boundary.

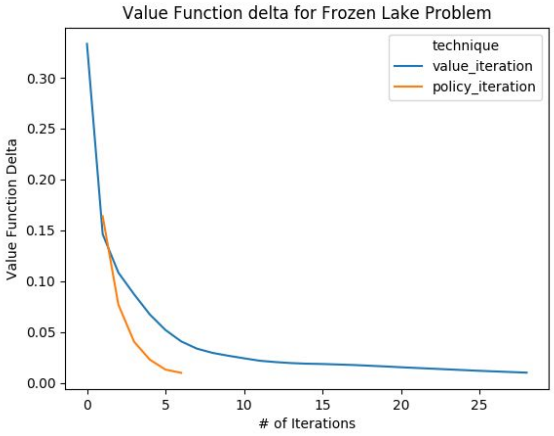



Also of note is the difference in the scale of the y-axis but this is purely due to the reward matrix that is default as part of the Frozen Lake problem so not much can be gleaned from this.

I then proceeded to re-run this problem instead using Policy Iteration.

## POLICY ITERATION

I ended up re-generating the same plot for this problem but with both the VI and PI curves plotted for comparison purposes, building on the previous plot.



As is easily observable, there are significant differences between these two lines where not only does the line representing Policy Iteration terminate at an earlier number of iterations but it's decay outpaces even that of Value Iteration which appeared to be fairly quick already with respect to this problem. Early termination on behalf of both techniques is largely a function of the sparse number of states in this problem (16 versus 5000) which drives the overall complexity of the problem in addition to the Forest Management problem having more free parameters. It's fairly clear from this plot that Policy Iteration really dominates Value Iteration in terms of performance on this specific problem.

## CONCLUSION

Having applied the three techniques to the Forest Management Problem and both Policy Iteration and Value Iteration to the Frozen Lake problem as well, I think there are some general findings we can state having run the requested experiments. In both cases, Policy Iteration performed the best of the three techniques in that it achieved convergence in the fewest number of iterations with the differences in computed Value matrices differing negligibly among the other techniques, which I verified manually but didn't know the best way to capture via the report. I think this is attributable to the nature of the two sample problems where more abstract and generalized strategies are more successful than the more specifically fit solutions as exemplified by Value Iteration and Q-Learning.

This has been a valuable exercise in illuminating the differences in execution and implementation between these different techniques in how they solve MDPs.

## REFERENCES


1. https://pymdptoolbox.readthedocs.io/en/latest/index.html

2. https://github.com/openai/gym

3. https://learning.oreilly.com/library/view/reinforcement-learning-algorithms/9781789131116/ab06aa68-01f9-481e-94ac-4c6748c3b858.xhtml