Markov Decision Processes (MDPs)

By **Sahil Gupta**

I. Abstract

Theoretical Expectations of MDP Algorithms

1. Our goal is to understand how Markov Decision Processes "MDPs" can be applied for sequential decision making to determine actions that focus not just on immediate rewards, but also future rewards. Hence we'll be exploring ***Value Iteration & Policy Iteration (referred to as "VI" & "PI" respectively for brevity)***. For the third algorithm, ***Q Learning "QL"*** is used.
2. Out of curiosity, I also wrote ***custom VI, PI & QL solvers*** to solve the grid world problem using the algorithms from the book "*Reinforcement Learning: An Introduction*"[1] written by Richard S. Sutton & Andrew G. Barto. This book will be heavily referenced throughout this paper. I also leveraged ***Hiive MDP Toolbox*** for the continuous domain problem.
3. In this paper, ***training*** *will imply the process of executing an MDP algorithm to get an optimal policy & optimal value function* given states, actions & rewards. ***Testing*** *will mean using this optimal policy to run a fixed number of episodes and evaluating this policies' performance on the environment.*
4. Let's briefly summarize learnings from lectures & book[1] to compare theoretical expectations with empirical results:
    a. **Policy Iteration**: Since we'll be looking at finite MDPs, PI is *expected "to converge to an optimal policy and optimal value function in a finite number of iterations."*[1] ***PI is expected to train faster than QL but slower that VI in terms of evaluation time since it requires sweeping the entire state space multiple times (at least on harder problems)***. PI *performs iterative Policy Evaluation & Policy Improvement until convergence*. ***PI is expected to take the lowest number of episodic iterations (still slower) to converge***. Another challenge PI is expected to run into is when there are multiple optimal policies, PI can run into infinite loops and needs a way to break ties or terminate early.
    b. **Value Iteration**: ***VI is expected to converge the fastest on harder problems to an optimal policy in a finite number of iterations***. VI focuses on finding the optimal value function iteratively and *requires only one policy extraction*. This is because an optimal value function yields an optimal policy. ***While VI can be faster on harder problems, it's expected to take more iterations than PI to converge***. Moreover, *VI uses the bellman operator & uses a non-linear max operation to determine utility*. Value is calculated using a one-step lookahead.
    c. **A Reinforcement Learning Algorithm - Q-Learning**: QL is a model free approach as the agent only discovers the reward for transitioning from one state to another given an action & upon receiving a reward. Per lectures, "Q learning rule is used to infinitely update where $\hat{Q}$<s, a> approaches Q <s, a>, the solution to Bellman equation." Hence, ***QL is expected to require more significant parameter tuning*** *(e.g. exploration-exploitation trade-off for epsilon, learning factor alpha & discount factor gamma)* ***and it's expected to take much larger iterations to converge***. *A major advantage of QL is that it can be easily applied to more real world scenarios since it learns by trial & error. PI & VI are expected to do better on the grid world than QL* because we know exactly how the rewards and state transitions are set up. On the other hand, QL is expected to perform better on larger state space problems where the agent has no prior knowledge of the world. From the book, "*The action-value function effectively caches the results of all one-step-ahead searches*. In $\hat{Q}$, the agent does not even have to do a one-step-ahead search: for any state s, it can simply find any action that maximizes $\hat{Q}$(s; a)". *Hence, QL is expected to perform better than VI though it takes many more iterations to learn.*

Defining And Understanding the Problems - Environments

Both the problems below have ***stochasticity due to the physics of the environments***, have ***infinite horizons*** since the agent continues to live forever after each iteration ends and have discrete number of iterations (unlike continuous time MDPs).

5. **Problem I: Frozen Lake from Open AI Gym**[2]: This is a classical 2D grid world problem. In frozen lake, an agent controls the movement of a character which starts off at a ***"Safe" starting point*** *S* on a lake but has to navigate through a course of ***obstacles H*** ("holes" in the water) to reach the ***terminal state/"goal"*** *G*. Some ***"frozen" tiles F*** of the grid lake are walkable but **movement direction of the agent is uncertain and only partially (probabilistically) depends on the chosen direction**. The agent is rewarded for finding a walkable path to the goal tile. This problem is particularly interesting because of its similarity to several real world applications such as autonomous mining, driving & lawn mowing, etc. *Another popular example is a Roomba vacuuming a floor* in a home where it has to avoid obstacles to clean the floor and reach it's charging station (terminal state). *In all these real world examples, several stochastic processes affect an agent's path, similar to agent's movement on frozen tiles.*
6. **Problem II: Forest Management from MDP Toolbox**[3]: Forest management is a non grid domain problem where the goal is "to manage a forest stand with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. The forest stand is managed by ***two possible actions: Wait or Cut***. An action is decided after each period t with the first objective to maintain an old forest for wildlife & second to make money selling cut wood. Each year there is a ***probability p that a fire burns the forest***. States S are defined corresponding to the ***age-classes of trees from 0 (youngest) to S-1 (oldest)***." This problem is particularly interesting because of it's *real world application to manage vast forests & national parks that face competing challenges - wildfires, wildlife preservation and logging*. I live in California where forests have a high

probability of wildfires & understanding its effect on wildlife & logging is interesting. Moreover, this problem is a very challenging bandit problem: “To solve this one must decide the best sequential harvest over a finite period of time. This problem becomes very interesting when the management involves a large amount of stands that are spatially linked.”[4]

II. Experimental Approach & Metrics Chosen

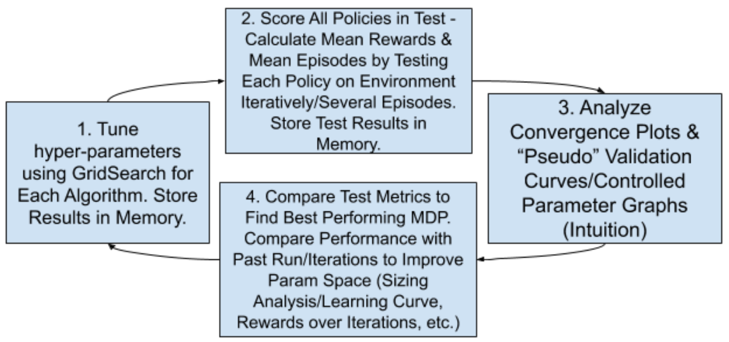
Metrics Used

During the ***training phase, the key metrics analyzed are evaluation time (seconds), number of iterations & number of errors*** (penalties) across all 3 algorithms. ***For PI & VI, their Value/Utility functions are also studied***. During the testing phase, each trained output policy was put to test on the environment for 1000 test episodes. ***Test metrics calculated are average rewards and average episodes***. In the case of Q Learning, the test average penalty is also analyzed.

Experimental Approach

You are encouraged to follow along with the linked code if interested. This contains dozens of experimental-runs with images and each future run is a subsequent improvement from past runs. More than 100 iterative experiments were conducted on a large AWS GPU based instance (16 CPUs & 16 threads) each to truly get best results possible. Some experiments ran for almost 20 hours each! The application is highly multi-threaded & parallelized.

Fig 1. ***Grid search was performed on each algorithm & on each problem of varying sizes during training***. Pseudo Validation curve graphs were used to ***understand the effects of each hyper parameter***. Each policy was then compared after testing this policy on the environment. ***Optimal policy is defined as one that maximizes test average rewards*** (total rewards/number of test episodes) for PI & VI. ***In QL, optimal policy was one that incurred the least penalties/errors & had best train rewards***. The average test rewards metric was not chosen due to the reward structure on environments. Finally, ***learning curves were plotted to understand the effects of problem size on MDP performance***.


1. Tune hyper-parameters using GridSearch for Each Algorithm. Store Results in Memory.
2. Score All Policies in Test - Calculate Mean Rewards & Mean Episodes by Testing Each Policy on Environment Iteratively/Several Episodes. Store Test Results in Memory.
3. Analyze Convergence Plots & “Pseudo” Validation Curves/Controlled Parameter Graphs (Intuition)
4. Compare Test Metrics to Find Best Performing MDP. Compare Performance with Past Run/Iterations to Improve Param Space (Sizing Analysis/Learning Curve, Rewards over Iterations, etc.)

III. Grid World - Frozen Lake

Random frozen maps were generated using `gym.envs.toy_text.frozen_lake.generate_random_map` with 20% holes. As seen in Fig 2, it has the states: a starting point (yellow), frozen tiles (green), holes (red) & goal (green). The agent starts on the starting point & performs actions - `"Left", "Down", "Right", "Up"` to reach terminal state. During each action, the probability of action actually working is only 0.33 i.e. say the action is down, then there is 0.33 chance agent goes down, 0.33 chance it goes left & 0.33 chance it goes right! The reward for reaching the goal is 1 and zero otherwise. ***The problem sizes are varied from [4, 10, 15, 20, 25, 30]. Though, only sizes 4x4 (easy - 16 states) & 30x30 (hard - 900 states) will be discussed***. Due to 30x30 large size, it can’t be visually shown in Figure 2.

Optimal Policy Visualizations

The figure below shows the final optimal policies obtained after running grid search and selecting “best” policy. All these policies below should make sense visually. *When the agent is near a red hole, it’s usually rewarded to not fall in it & hence acts in a direction opposite from going towards the hole*. When on frozen tiles, arrows are usually directing the agent towards the terminal state green. Empirically, PI & VI produced the same optimal policies for all sizes (some not shown for brevity)! Per Sutton & Barto, “*Value iteration effectively combines one sweep of policy evaluation & one sweep of policy improvement. In general, the entire class of* ***PI algorithms can be thought of as a sequence of sweeps. Because max operation in VI is the only difference between these updates***.” Hence, *VI is similar to PI except for these structural differences mentioned.* ***QL had minor differences in optimal policy*** *and reasonably so because it’s a model free learner (transition model & rewards are not known to the agent)* ***learning through trial & error***. *It has to learn each state by observing <<State, Action>, Q>.*

Fig 2a. Optimal Policy Using PI

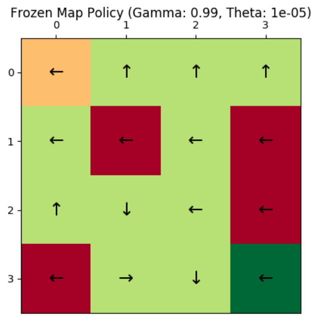


Fig 2b. Optimal Policy Using VI

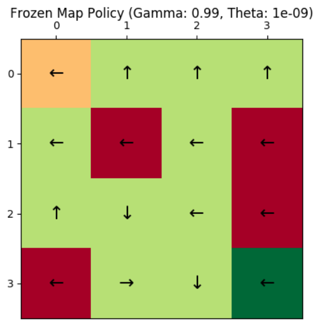


Fig 2c. Optimal Policy Using QL

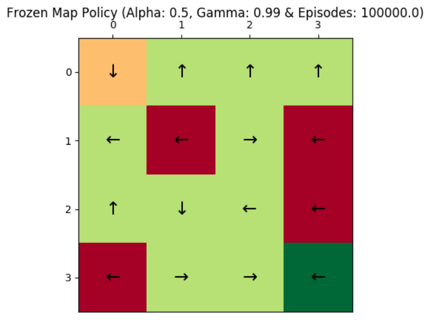


Fig 2d. PI & VI lead to same policies (Both have Gamma = 0.99 & Theta = 1e-7) as:

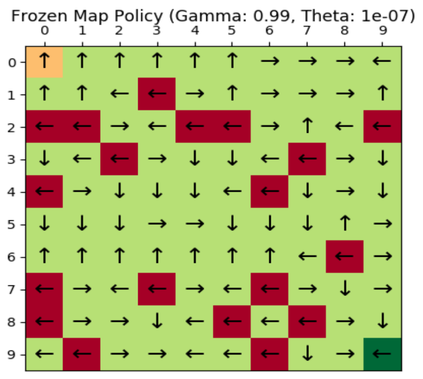


Fig 2e. QL leads to slightly different optimal policy on a 10x10 grid as:

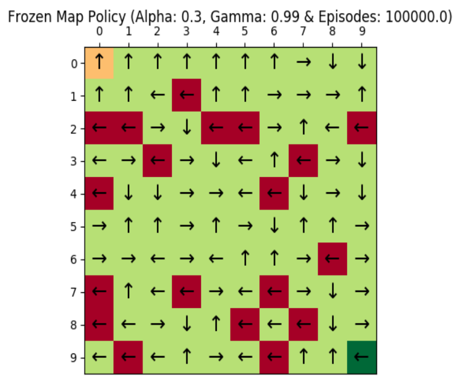
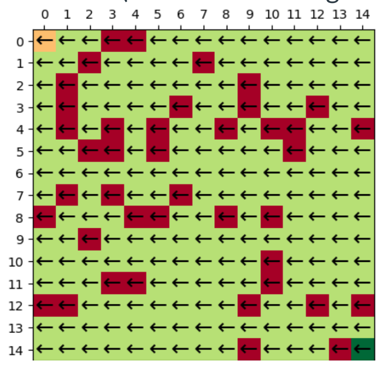

Frozen Map Policy (Alpha: 0.3, Gamma: 0.99 & Episodes: 100000.0)

Fig 2f. QL stops converging for grids including & over 15x15 even after 1e7 iterations (almost 3.5 hours grid search!).



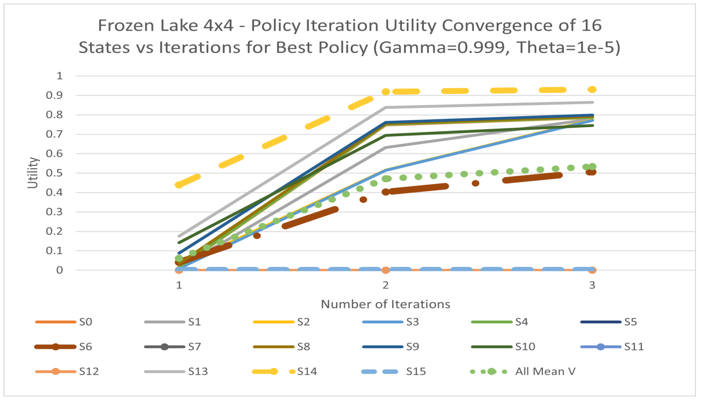
Convergence Plots

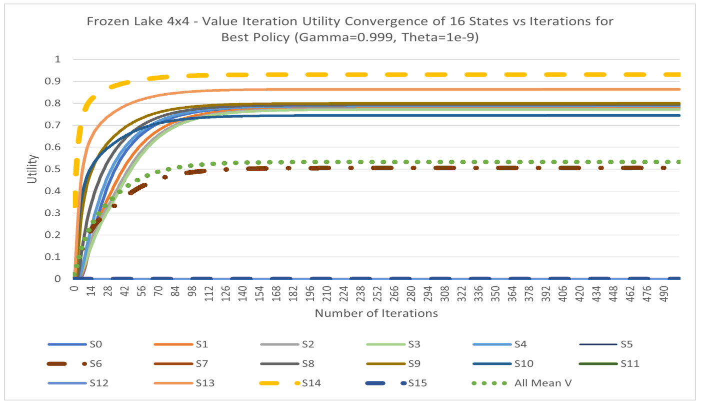
Figure 3 below shows utility or value function convergence plots vs number of iterations for Frozen lake 4x4 as seen in Fig 2 above. As expected, ***we see PI finish in mere 3 iterations. VI takes 500 iterations****, but convergence to optimal utility (or policy) starts to occur around the 150 iteration mark.* ***These empirical results match our expectations!*** Here are some unique findings:

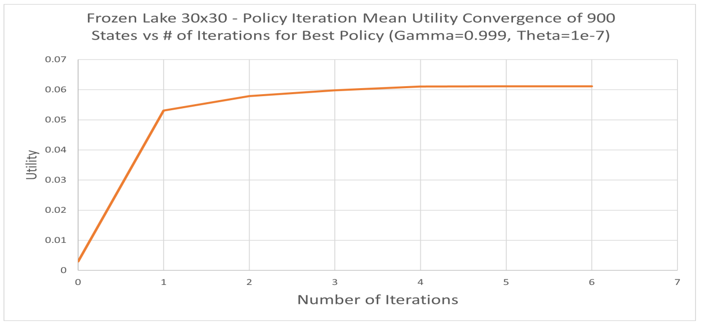
1. See the flat blue dotted overlapping lines on x-axis. These are ***states with red holes (S5, S7, S11, S12) with utility 0***.
2. The *yellow dashed line* ***S14 has the highest utility****! This is because it's the only entry point to the goal state (green).*
3. The green dotted "All Mean V" line represents the combined mean utility of all states. Notice the brown dashed-dotted ***state S6 line that has the lowest non-zero utility****. This is because S6 frozen tile is between two red holes S5 & S7* that increase the likelihood of the agent falling into these holes!

Frozen lake 30x30 has a much smoother convergence graph for PI & here PI takes 7 iterations while VI takes ~440 iterations. *VI's & PI's test average reward is 0.834 i.e. 83.4% chance & 0.838 i.e. 83.8% chance* respectively that the agent reaches the goal. ***Their performance in test is comparable though PI does slightly better***. QLearning convergence graph can be seen below in Figure 4 for frozen lake 4x4. *We can see* ***convergence around ~35000 iterations***. ***On 30x30, QL never converges*** *due to the reward structure setup i.e. episodic reward of 0 or 1 isn't ideal for QLearning. It* ***performs the worst on hard problems*** *and never converges even after extensive tuning. A key thing to note is that* ***QLearning surprisingly performs the best on small problem 4x4 with average test reward of 0.88 i.e. it 88% chance agent reaches the goal***. Also, *QLearning suffered the most from stochasticity - its results & performance varied significantly across experiments.*

Fig 3. Utility vs Iterations for PI & VI for 4x4 & 30x30 frozen lakes. 4x4 is broken down per state to show detailed insights.







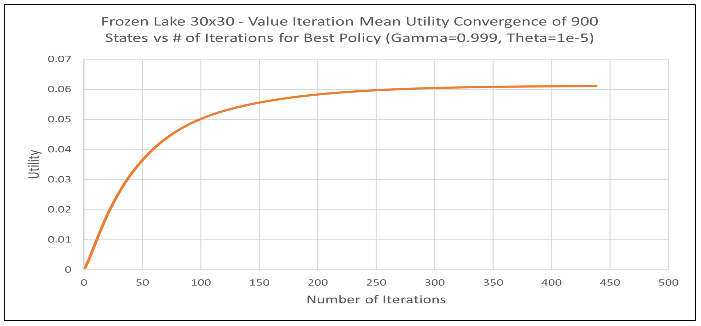
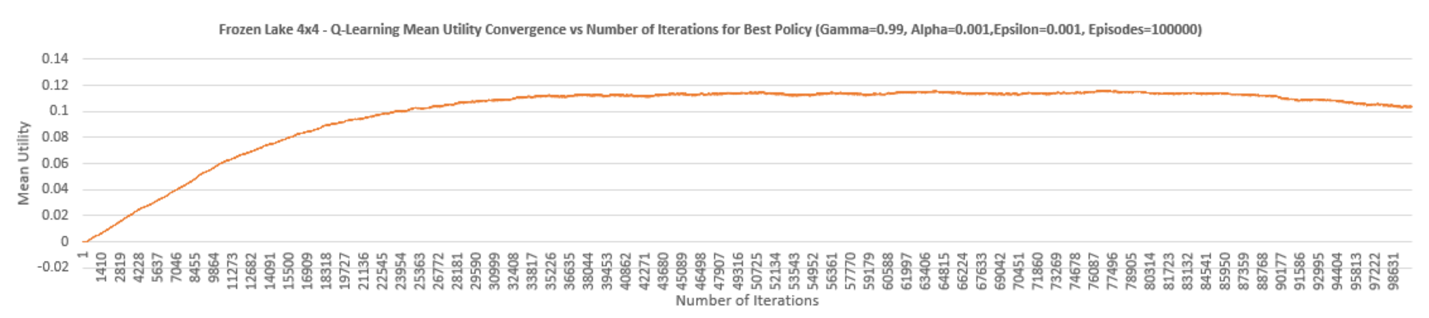


Fig 4. Mean Utility vs Number of Iterations for QL for frozen lake 4x4. 30x30 is not shown because it was unable to converge and doesn't show anything meaningful except poor performance. *Reasons for no convergence with Q-learning are discussed in Part V in great detail.*



Parameter Tuning - Grid Search & "Validation Curves"

To understand how we arrived at the results, it's important to see the extensive tuning performed on each algorithm and effect of each parameter on results. We'll be discussing exploration-exploitation strategies and how they affect the results.

```
"iterative_gamma_discount_factors":
[0.5,0.7,0.9,0.95, 0.99,0.999],
"thetas": [1e-3, 1e-5, 1e-7, 1e-9]
```

```
"q_learning_gamma_discount_factors": [0.7,0.9,0.95,0.99,0.999],
"alpha_learning_rates": [1e-6, 1e-3, 1e-2, 0.1, 0.3],
"epsilons_exploration_exploitation": [1e-3, 1e-2, 1e-1],
"epsilon_decay_rate": [1e-6, 1e-2],
"total_episodes": [1e4, 1e5, 1e6, 1e7]
```

Above are the ranges of parameters explored via the Grid search and validation curves. Figures 5a, 5b, 5c below talk about PI & VI parameters and their effect on learner performance.

Fig 5a. Mean Test Rewards vs Discount Rate (Gamma) can be seen. Both PI & VI behave similarly (PI is shown due to brevity). *Larger discount factor means larger horizons & long term focus. Hence, on frozen lake -* ***long term rewards lead to better test rewards than agent focusing on short term horizon***.

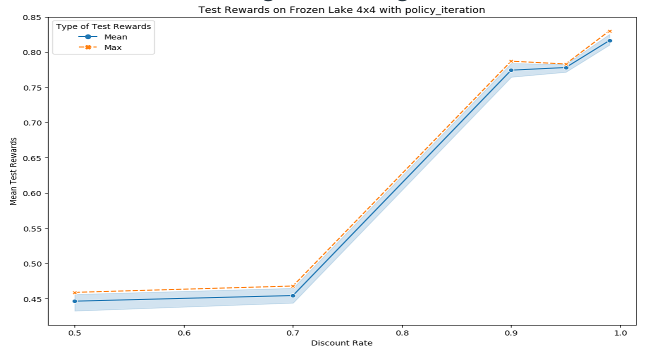


Fig 5b. Mean Train Evaluation vs Log(Theta) can be seen. PI & VI behave similarly (VI shown due to brevity). ***Theta is a small threshold used to determine the accuracy of estimation***. *Hence,* ***smaller theta takes significantly longer to train*** *because more iterations are needed to reach larger accuracy*.

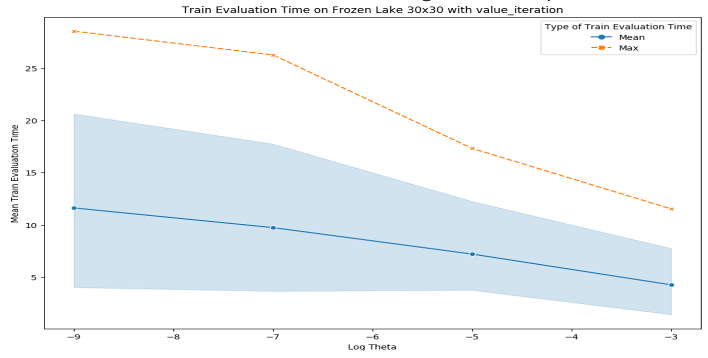


Fig 5c. ***Large discount factors produce best policies but take very long time to train***. E.g. Gamma=0.9 meant PI took 400s mean & 800s max! VI (not shown) also saw an exponential increase with Gamma>0.9. E.g. 25s with Gamma=0.999.

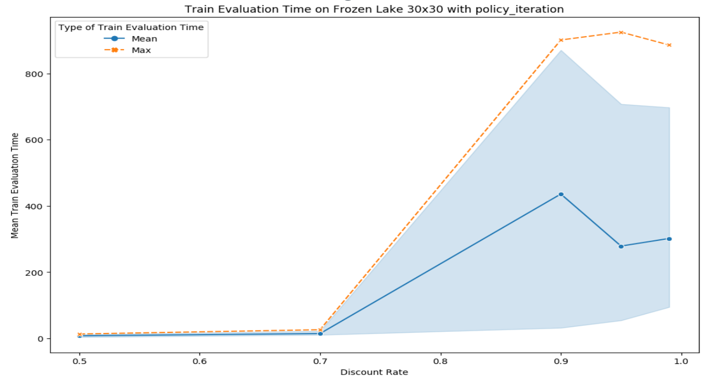


Fig 5d. Q-Learning Tuning for Learning Rate (alpha) on 4x4. Per lectures, "***alpha=0 means no learning and alpha=1 means full learning*** *i.e. we forget everything that we knew and jumped to new value.*" ***It's clear that smaller alpha values are preferred***.

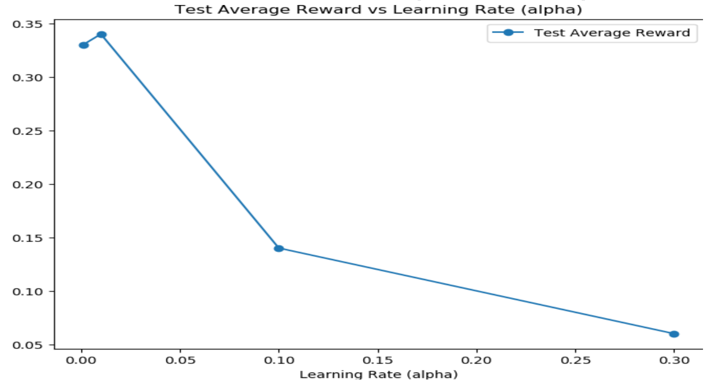


Fig 5e. Q-Learning Tuning for Discount Rate (gamma) on 4x4. Same as VI & PI, higher gamma leads to better performance but consumes more evaluation time (not shown). *Though, we see an interesting pattern here of reduced returns as gamma is increased further (inverted U shape).*

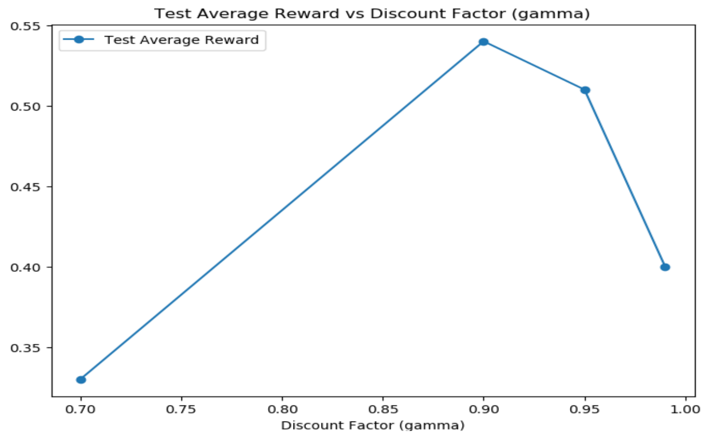
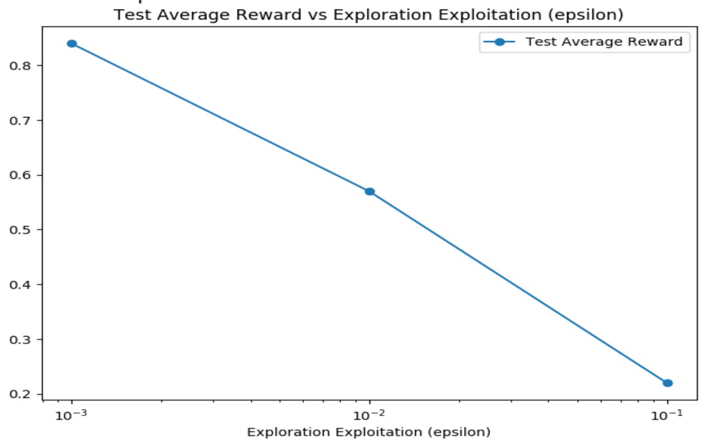


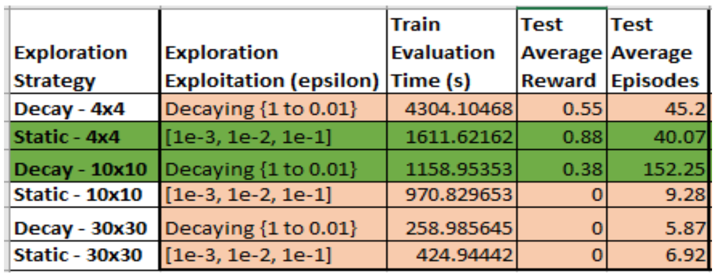
Fig 5f. Q-Learning Tuning for Epsilon on 4x4. Per lectures, "***epsilon is probability of taking a random action (exploration) instead of best action (exploitation)***." It's evident empirically that ***smaller epsilon values are preferred by the agent on the frozen lake*** example as 1E-3 leads to a much better policy than 1E-1 epsilon.



QLearning Exploration & Exploitation Strategies & Trade-off

Per lectures, "In exploration-exploitation, ***exploitation is about using what you know i.e.*** $\hat{\Pi}$***->***Π***(pi-hat approaches pi i.e. optimal policy) & exploration is about getting the data that you need so that you learn i.e.*** $\hat{Q}$***->Q (Q-hat approaches Q)***. Using GLIE i.e. greedy in the limit with infinite exploration i.e. ***we decay our epsilon greedily***." Here, epsilon is decayed from epsilon=1 to epsilon=0.01.

Fig 6. Table summarizes the performance of 2 strategies on Frozen lake. One where ***epsilon is decayed greedily and one where epsilon was kept the same per training & testing round but tested for different epsilons - [1e-3, 1e-2, 1e-1]*** (as seen above in Fig 5f). It's clear that a *decay strategy will do better for harder problems but static epsilon (no decay) can perform well on easier problems depending on parameter selection & tuning.*



Exploration Strategy	Exploration Exploitation (epsilon)	Train Evaluation Time (s)	Test Average Reward	Test Average Episodes
Decay - 4x4	Decaying {1 to 0.01}	4304.10468	0.55	45.2
Static - 4x4	[1e-3, 1e-2, 1e-1]	1611.62162	0.88	40.07
Decay - 10x10	Decaying {1 to 0.01}	1158.95353	0.38	152.25
Static - 10x10	[1e-3, 1e-2, 1e-1]	970.829653	0	9.28
Decay - 30x30	Decaying {1 to 0.01}	258.985645	0	5.87
Static - 30x30	[1e-3, 1e-2, 1e-1]	424.94442	0	6.92

IV. Continuous Domain - Forest Management

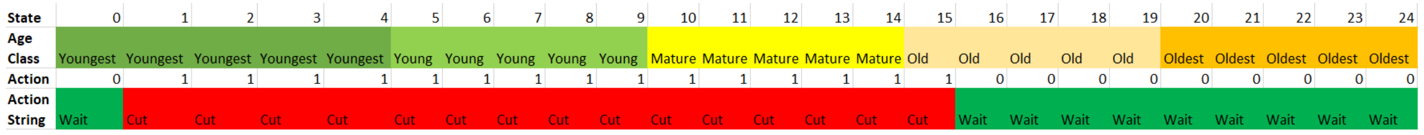
Random Forest Management maps were generated using `hiive.mdptoolbox.example.forest(S=num_states, r1=oldest_wait_reward, r2=oldest_cut_reward, p=wildfire_probability)` with varying number of states. The reward when the forest is in its oldest state and action is 'Wait' was set to 4 & the reward when the forest is in its oldest state and action is 'Cut' was set to 2. The stochasticity here is due to the probability of forest fires which is set to 0.1 i.e. there is 10% chance that a wildfire occurs and leaves the forest stand at the youngest age class or state. ***The problem sizes are varied from [25, 100, 225, 400, 625]. Though, only sizes with 20 states (easy) & 625 states (hard) will be discussed***. In this MDP, the environment provides a transition probability array P (A x S x S') & reward array R (S x A).

Optimal Policy Visualizations

Since this is a non-grid world problem, it's not straightforward to visualize these in a 2-D picture. Below is a heat-map which shows the output policies here by each algorithm for 25 & 625 state forest stands & the output actions proposed by these policies. Optimal output policies were selected using the same experimental grid search approach discussed in Part II. Young trees are coded green & older trees are encoded yellow-orange. Actions proposed - wait=0 are encoded green and cut=1 are encoded red. *Same as the frozen lake grid world,* ***we can see that VI & PI produce the same (similar) policies***. ***QL had several differences in optimal policy as before***. *We can see that* ***for younger states, PI, VI & QL all recommend "Cut" but for older states, QL recommends action as "Cut" while VI & PI say "Wait"***. *Moreover, on the harder problem, there is even more difference between policy generated by QL vs PI & VI. This is unsurprising because again QLearning reinforcement learning technique is model free and using trial & error. It may or may not have converged at forest stand size=625. Let's investigate this in the next section in greater detail.*

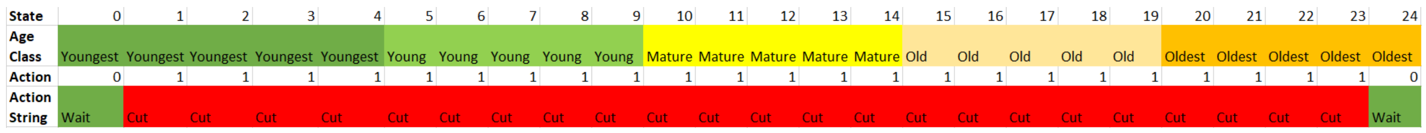
Fig 7. Optimal Policy visualizations that maximized test average rewards for # of states 25 (first two) & 625 (last two).

PI & VI (25)



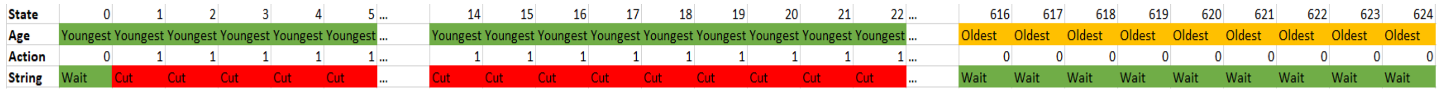
State	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Age Class	Youngest	Youngest	Youngest	Youngest	Youngest	Young	Young	Young	Young	Young	Mature	Mature	Mature	Mature	Mature	Old	Old	Old	Old	Old	Oldest	Oldest	Oldest	Oldest	Oldest
Action	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Action String	Wait	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Wait

QL (25)



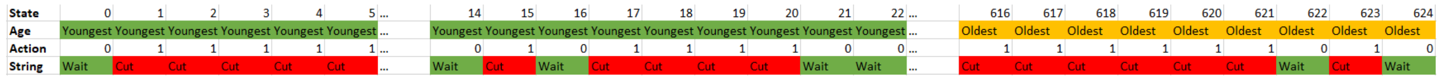
State	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Age Class	Youngest	Youngest	Youngest	Youngest	Youngest	Young	Young	Young	Young	Young	Mature	Mature	Mature	Mature	Mature	Old	Old	Old	Old	Old	Oldest	Oldest	Oldest	Oldest	Oldest
Action	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Action String	Wait	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Wait

PI & VI (625)



State	0	1	2	3	4	5	...	14	15	16	17	18	19	20	21	22	...	616	617	618	619	620	621	622	623	624
Age	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	...	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	...	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest
Action	0	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0
String	Wait	Cut	Cut	Cut	Cut	Cut	...	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	Cut	...	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Wait

QL (625)



State	0	1	2	3	4	5	...	14	15	16	17	18	19	20	21	22	...	616	617	618	619	620	621	622	623	624
Age	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	...	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	Youngest	...	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest	Oldest
Action	0	1	1	1	1	1	...	0	1	0	1	1	1	1	0	0	...	1	1	1	1	1	1	0	1	0
String	Wait	Cut	Cut	Cut	Cut	Cut	...	Wait	Cut	Wait	Cut	Cut	Cut	Cut	Wait	Wait	...	Cut	Cut	Cut	Cut	Cut	Cut	Wait	Cut	Wait

Convergence Plots

Figure 8 below shows the convergence plots for all 3 algorithms for Mean V (utility) vs Number of Iterations. ***PI & VI converged very quickly as expected because they exploit the known model.*** For forest stand size=25, *VI's & PI's test average reward is 1.1627 & 1.1671* respectively. ***Their performance in test is comparable though PI does slightly better (same as frozen lake).*** PI & VI convergence graphs for forest management stand of size=625 are not shown here because they have very similar graphical patterns to size=25 & surprisingly take the same number of iterations (also omitted due to brevity).

Per Fig 8c., QL took almost 10 million iterations to converge (it took 1.5 hrs to run or 5,400 seconds to evaluate this policy***)***! For single agent QL, forest stands greater than 50 states and above couldn't converge in under 1e7 iterations even after extensive grid search on more than 30 different parameter combinations. This is unsurprising because even with multi-agent QL, INRA's Chades` & Bouteiller[4] conclude the challenges in QL algorithm convergence on forest management problem. Part V below discusses QL time complexity in greater detail. Compare QL convergence graph Fig 8c. for forest stand of size=25 with Fig 8d. with forest stand of size=625. Again, *a key thing to note is that* ***QLearning surprisingly performs the best on small forest stand size=25 with average test reward of1.4683 i.e. almost 1.3 times more than PI & VI****! It's also* ***important to point out that while QL didn't converge at size=625 (Fig 8d.), it still did a lot better than on frozen lake*** *(where it got 0 average test rewards after 1e5 iterations).* ***At size=625, it got 0.819 test average rewards in comparison to 1.005 & 1.006 test average rewards for PI & VI respectively***. This implies that if the number of episodic iterations was bumped to say 1e8 or 10 times more, we may have seen convergence. Of course, this may have taken several days to train (beyond the scope). This is also evidenced in the figures 8e & 8f where "QL Rolling Reward Sum/Iterations vs # of Iterations" has been plotted to explain convergence behavior for QL on easy vs hard problems.

Fig 8a. PI Mean V vs # Iterations (size=25). Notice that ***PI converges in least # of iterations here to the optimal utility function i.e. a mere 8 iterations.***

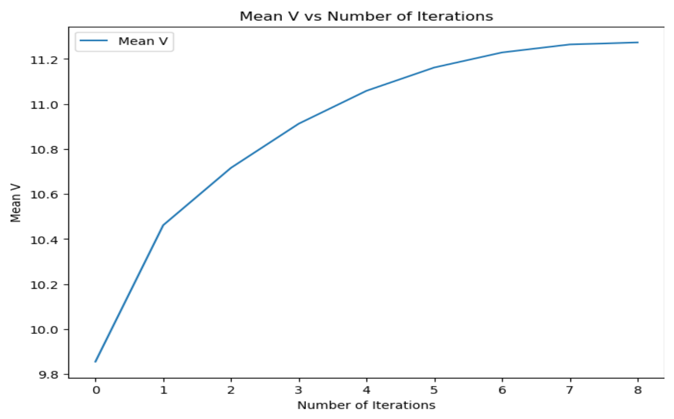


Fig 8b. VI Mean V vs # Iterations (size=25). ***VI which takes ~50 iterations to converge***. Note that Mean V y-axis scales in VI & PI are similar.

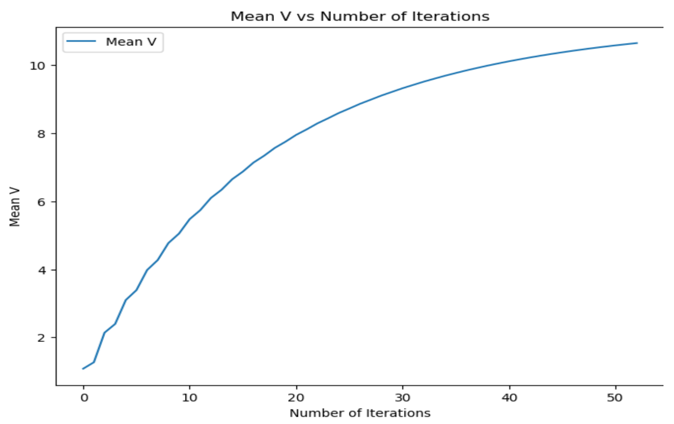


Fig 8c. QL Mean V vs # Iterations (size=25). Note x-axis is actually in 1e3 scale i.e. goes from 1e3 to 1e7 iterations. ***QL takes the longest # of iterations to converge as expected***.

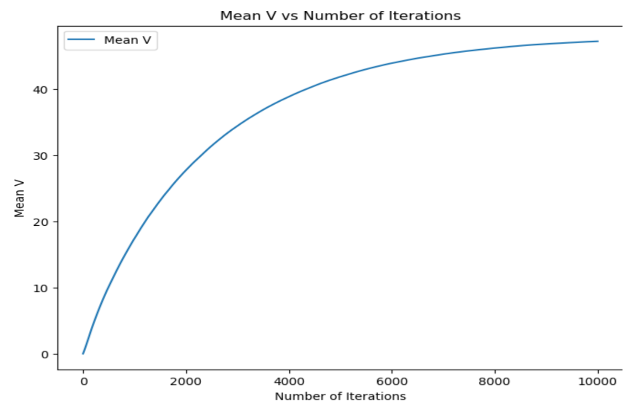


Fig 8d. QL Mean V vs # Iterations (size=625). It shows that *QL never converges here even after 10 million iterations & 1.5 hrs evaluation time.*

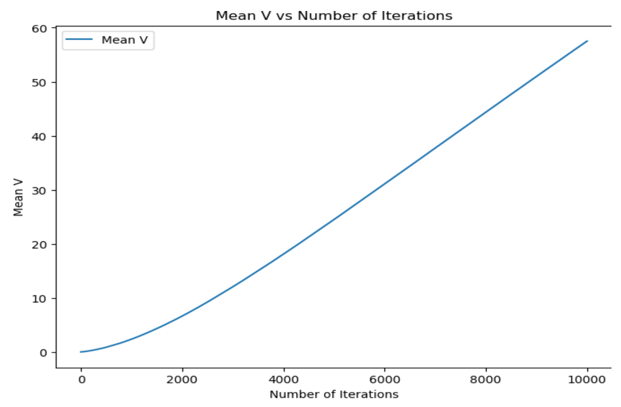


Fig 8e. QL Rolling Reward Sum/Iterations vs # Iterations (size=25) shows that ***within 2e6 iterations, QL achieved most of its rewards (rate of acquiring "rewards/iteration" reduced)***

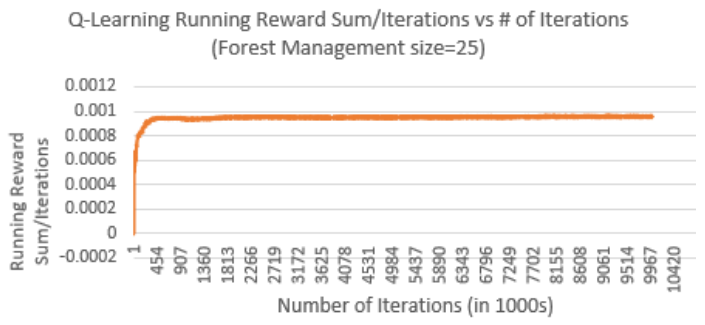
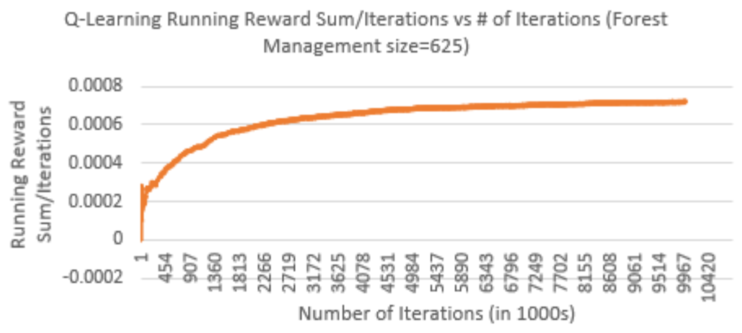


Fig 8f. QL Rolling Reward Sum/Iterations vs # Iterations (size=625) shows that ***even after 1e7 iterations, QL was still learning & getting more rewards.***



Parameter Tuning

Grid Search & "Validation Curves"

Same as frozen lake, extensive parameter tuning was performed on Forest Management. We can see the similarity in tuning between Reinforcement Learning/MDP & Supervised learning tuning processes where comparing training & testing results is crucial for model complexity/validation curve analysis. Note that tuning results for forest stand=625 are not shown due to brevity but similar tuning was performed on it. Below are some interesting insights for forest stand=25:

1. VI/PI Tuning: Compare Figures 9a & 9b which show Rewards vs Iterations in Training & Average Test Rewards vs Discount rate respectively. These showcase Discount Factor tuning results for VI only (PI has similar graphs and hence not shown). We see that during training Gamma=0.999 (brown line) leads to highest rewards over iterations. *Though during testing, mean test rewards are significantly lower for Gamma=0.999*. In fact, Gamma=0.7 led to best performance during testing. Per lectures: "*smaller gamma means shorter horizon i.e. short term focus*. This means that we value short term rewards more and are encouraged to get as many and quick rewards as possible even though long term rewards may be large. ***Hence, Gamma=0.7 shows that short term rewards are favored in Forest Management when compared to Frozen Lake where larger Gamma>0.99 was preferred***. While not shown, similar results were seen for Q-learning too (Gamma=0.7 was chosen for QL).
2. Q-learning Tuning: Compare Fig 9c & 9d which show Rewards vs Iterations in Training & Average Test Rewards vs Discount rate respectively. These showcase Learning Rate (alpha) tuning results for QL only. ***We see that during training & testing Alpha=0.01 led to best performance***. Here, the train & test results are similar.

A Short Note on Q-Learning Exploration & Exploitation Strategies & Trade-off

Compare Fig 9e & 9f which show the 2 exploration-exploitation strategies applied on forest management. We can see that ***a decaying epsilon exploration strategy from epsilon=1 to 0.1 with decay rate=1e-2 performs slightly better than keeping epsilon the same throughout the training round***. Again, epsilon is the probability of taking a random action (exploration). ***Higher epsilon values lead to more exploration***. In Figure 9f, it's clear that higher epsilon values (1e-2, 0.1, 0.3, 0.5) are preferred i.e. more exploration is ideal.

Fig 9a. (VI Tuning Gamma) Training Rewards vs # Iterations.

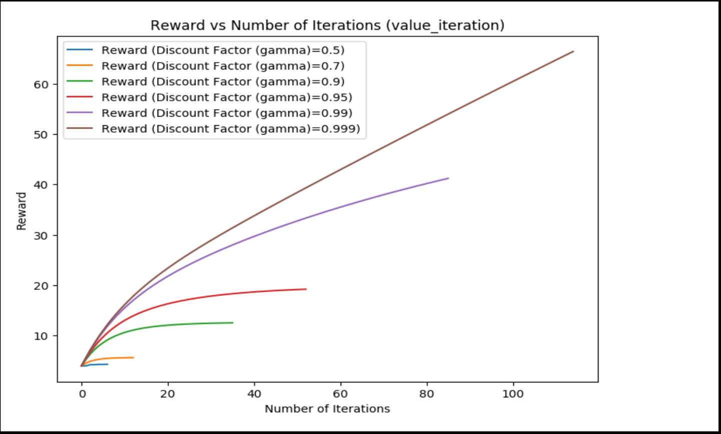


Fig 9b. Average Test Rewards vs Discount Rate for VI.

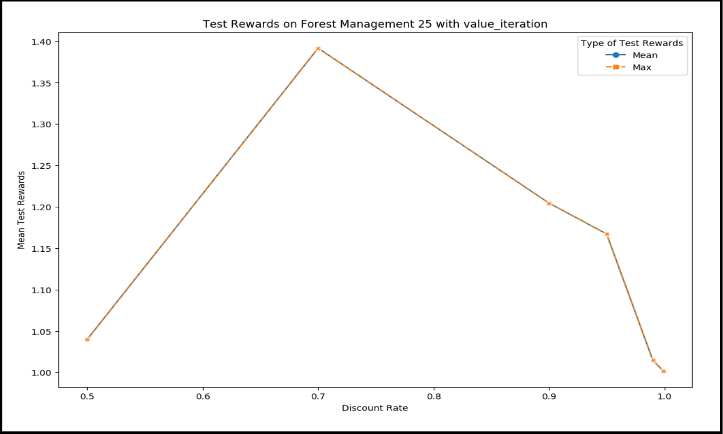


Fig 9c. Training Rewards vs Number of Iterations for QL.

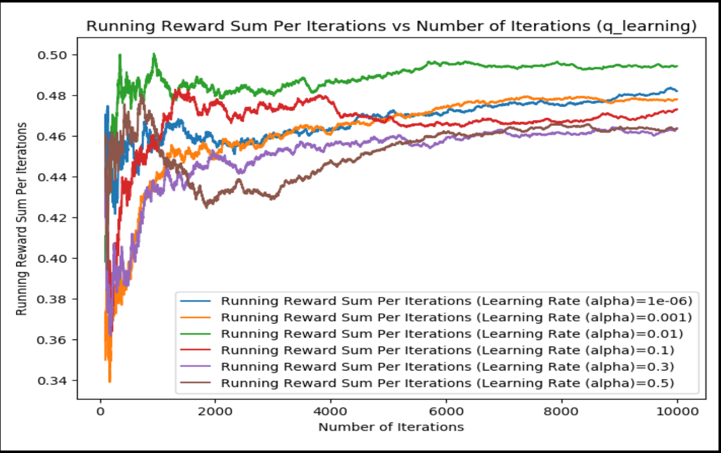


Fig 9d. Average Test Rewards vs Learning Rate for QL.

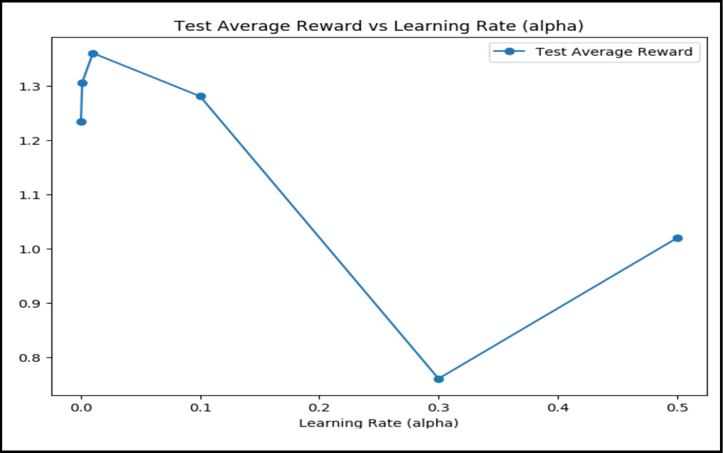


Fig 9e. Q-Learning Test Average Reward vs Epsilon Decay Rate. *Here, the epsilon decay strategy with varying decay rates is applied with max_epsilon=1 and min_epsilon=1 (defaults in MDPToolbox).*

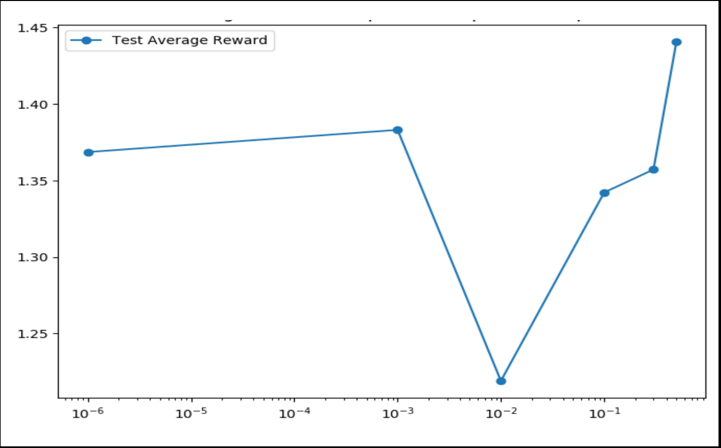
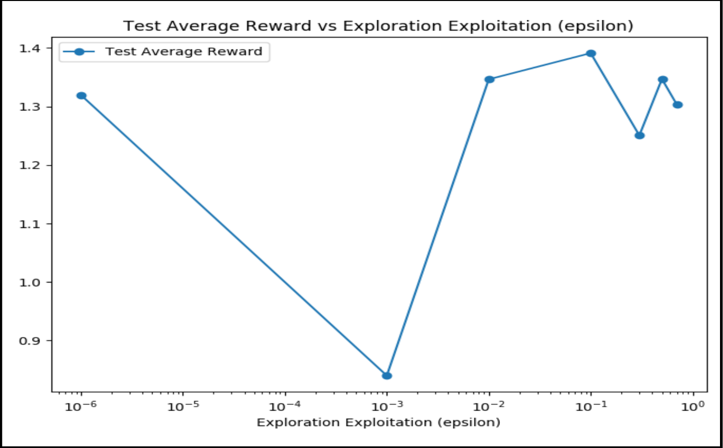


Fig 9f. Q-Learning Test Average Reward vs Epsilon. *Here, the epsilon is kept the same throughout the train & test round per parameter combination (no decay). Several combinations for epsilon are tried: [1e-6, 1e-3, 1e-2, 0.1, 0.3, 0.5].*



V. Deep Dive on Varying Problem Sizes - Pseudo "Learning Curves"

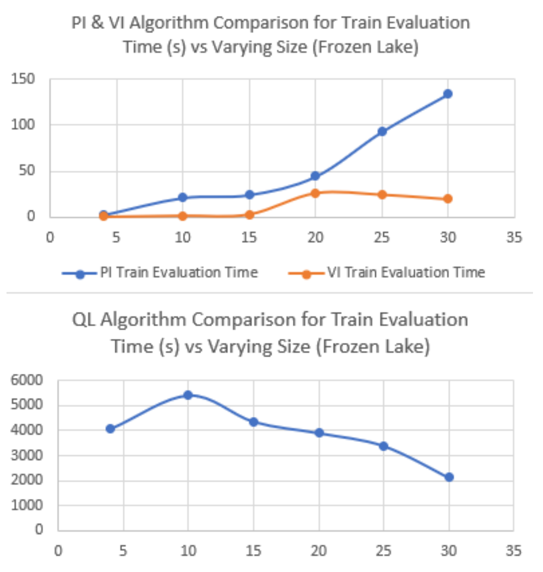
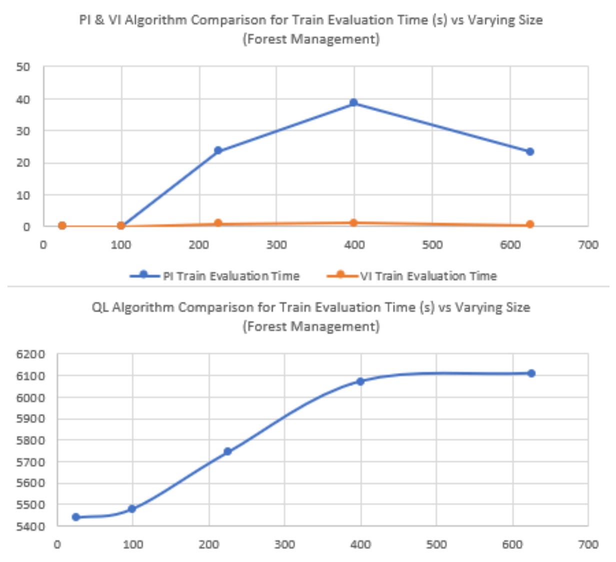
Train Time Complexity

To make sense of training times, let's analyze the time complexity of algorithms in greater detail. ***Worst case time complexity of PI[5] is given by*** $O(|S|^3 + |S|^2|A|)$**[5], *VI[6] is given by*** $O(|S|^2|A|)$***& QL[7] is given by*** $O(n^3)$ *where S is the cardinality of the state set, A is the cardinality of the action set, n is the number of steps needed to reach the goal state (convergence).* These equations imply that:

1. ***For larger state sets, PI will outweigh run times for VI (by $|S|^3$)***. This is exactly the behavior we see in frozen lake (Fig 10a) where PI overtakes VI's run time on harder problems. Same is true for the Forest Management problem (Fig 10b). Frozen lake has 4 actions while forest management below has only 2 actions.
2. ***PI's & VI's growth is linear to the action set's cardinality growth***. Frozen lake has 4 actions while forest management has 2 actions. Not only does PI take significantly longer than VI as size increases, but this growth linearity also explains the difference in curves among the frozen lake (widening curve between PI & VI as state size grows - Fig 10a) & forest management (shortening curve between PI & VI as state size grows - Fig 10b).
3. ***For Q-learning, as the number of episodes/iterations increase, time cost becomes extremely high*** due to cubic n steps.

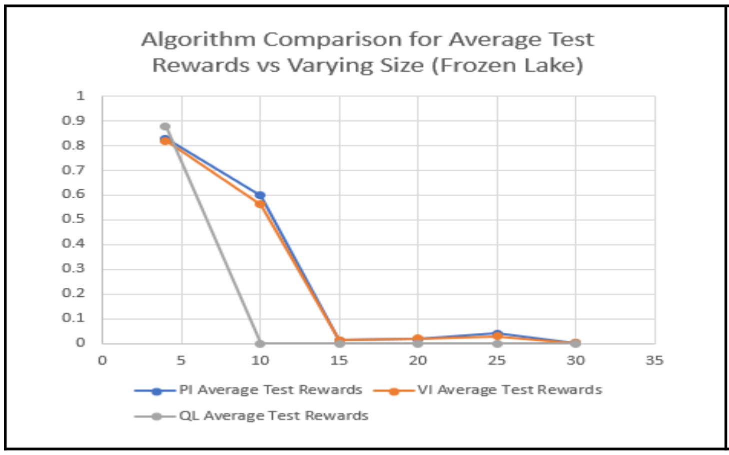
Now, let's look at concrete numbers:

1. *In Fig. 10a, Frozen lake 4x4* - ***PI takes 0.328s to train, VI takes 2.161s & QL takes 1611.62 seconds to train i.e. almost 800 times longer than VI***! On frozen lake 30x30, ***PI takes 149.67s to train, VI takes 52.37s & QL takes 457 seconds to train. We see PI take almost 3 times longer than VI***.
2. *In Fig. 10b, for Forest Management stand=625* - ***PI takes 22.5s while VI takes a mere 0.4s to train 50 iterations i.e. PI takes 55 times longer***. This behavior of PI taking ***significantly longer than VI is the same as frozen lake.*** This is likely because maximization over action set in forest management is costly. Since, VI does this max over a smaller action set in forest management, it's faster than VI in frozen lake when PI is taken as a baseline.
3. QL takes a long time to converge and is shown on different graphs. *For Frozen lake,* ***QL takes shorter time on 30x30 in comparison to 4x4 because there is no learning occurring as it never gets any reward & agent drops into a hole early as size becomes large***. This leads to Q-table on 30x30 set to all 0 (as updates never happen). In Forest Management, size=25 takes 5000s & size=625 takes 6100s. Here, even as size increases, the agent learns & performs well (is learning and hence Q-hat approaches Q).

Fig 10a. Training Time (s) vs Varying Frozen Lake sizes.	**Fig 10b**. Training Time (s) vs Varying Forest Stand sizes.
	

Average Test Rewards Performance

Fig 11a. Average Test Rewards vs Varying Frozen Lake sizes.	**Fig 11b**. Average Test Rewards vs Varying Forest sizes.



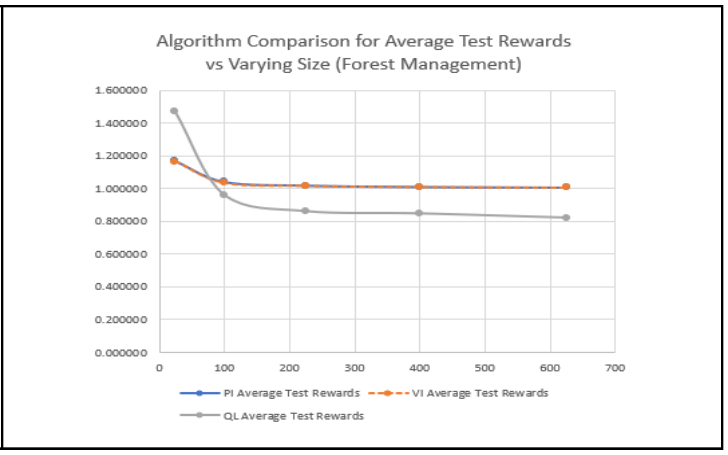


Figure 11a & 11b show performance in test for each algorithm on Frozen lake and Forest management respectively. ***It's clear from both the graphs that VI & PI have comparable performance*** i.e. their output policies are either the same or similar such that running episodic tests on them yields the same results as sizes even as problem size is varied. In Frozen lake (Fig 11a), PI performs marginally better than VI (that might be due to stochasticity) and in Forest Management (Fig 11b), PI & VI deliver the same results across all sizes as evidenced by overlapping orange and blue lines. As we saw before, Q-Learning does really well on easy problems and even better than PI & VI. Unfortunately, on problems with large state set cardinality, QL doesn't perform as well as PI & VI. Though, it's clear that it's not too far behind PI & VI and its performance is within 20% margin.

Q-Learning Performance & Convergence Analysis

Per lectures: "***Q learning rule is used to infinitely update where $\hat{Q}$<s, a> approaches Q <s, a>***". This means that given more (or infinite) computational time, it'll approach true Q function. In larger Frozen lake grids, the agent only gets a reward of 1 when it reaches the goal. ***Hence, as the grid becomes larger and due to significant stochasticity of slipping (only 33% chance of action actually working), it becomes impossible for Q-learning to get rewards in training and test without slipping into a hole because it's model free and doesn't know of state transitions <T, R>.*** Of course, we could've easily customized the environment to have a different reward structure and Q-learning would've converged faster/earlier - say give 1 reward to every action where it stays alive and 0 for everything else. *But, this lack of convergence helped show the stark contrast between Frozen lake and Forest Management problems*! Forest management's reward structure is more conducive for Q-learning to perform well.

VI. Conclusion

It's evident that trade-offs matter when it comes to selecting the right MDP algorithm. The ***heat map below shows a detailed comparison of all 3 algorithms***:

	Train Evaluation Time				Iterations to Converge				Average Test Reward							
Algorithm	Frozen Lake Easy	Frozen Lake Hard	Forest Management Easy	Forest Management Hard	Frozen Lake Easy	Frozen Lake Hard	Forest Management Easy	Forest Management Hard	Frozen Lake Easy	Frozen Lake Hard	Forest Management Easy	Forest Management Hard	Assumes Model is Provided	Time Complexity	Memory/ Space Complexity	Learner Type
Policy Iteration	0.3284	149.6760	0.0062	23.2710	~2 with 3 total	~2 with 3 total	9	9	0.838	0.001	1.1671	1.00583	Yes	O((S^3) + (S^2 * A))	High	Offline
Value Iteration	2.1614	52.3700	0.0255	0.4080	~150 with 503 total	~250 with 438 total	53	53	0.834	0.003	1.1627	1.00658	Yes	O(S^2 * A) or cubic	High	Offline
Qlearning	1611.6200	457.6400	5439.2900	6109.5360	35000	Never	1E+07	Never	0.88	0	1.4683	0.81901	No	O(N^3)	Low	Online

Green highlights superior performance, yellow highlights decent to manageable performance and red implies poor performance. ***Policy Iteration & Value Iteration are model based learners that assume the model is given to them***. Often, in the real world these are unknown and hence, PI & VI can't be applied in those cases. Though, for the 2 MDP problems - grid based Frozen lake & non-grid Forest management, PI & VI perform really well. ***Policy Iteration was a clear winner on several fronts*** - *fastest convergence in terms of # iterations, good test performance on easy problems and hard forest management problem*. ***Value Iteration***, *while, slower than PI to converge in terms of # of iterations, is* ***the fastest to train on hard problems and the best performer in terms of average test rewards on hard problems***.

On the other spectrum, *Q-learning is a model free learner. This reinforcement learning algorithm can be applied even when transition probabilities and rewards are unknown*! In both the examples, we were lucky to be given transition probabilities and rewards. *Surprisingly,* ***Q-Learning with a single agent performed the best on easier problems in terms of test average rewards and beat VI & PI***. Unfortunately, ***on both easy & hard problems it's train evaluation time is very long***. On harder problems, it performs poorly - "model-free algorithms suffer from a higher complexity compared to model-based approaches"[8]. It can be empirically shown that a multi-agent Q-learner can effectively solve harder MDP problems[4] (given more time, this technique would've been explored). Moreover, ***model-free algorithms like Q-Learning "are online, require less space, and, most importantly, are more expressive since specifying the value functions or policies*** is often more flexible than specifying the model for the environment - ***arguably outweigh its cons relative to model-based approaches***."[8]

To conclude, the choice of algorithm depends on the environment given, sample complexity, whether model is known vs unknown, computational complexity & train time availability, etc. to name a few. Finally, it's clear that empirical results closely match all the theoretical expectations that were mentioned in the introduction.

VII. Citations

[1] Richard S. Sutton & Andrew G. Barto, "Reinforcement Learning: An Introduction" [E-Book]
[2] Open AI Gym, "FrozenLake-v0" [URI]
[3] INRA, "Markov Decision Process (MDP) Toolbox" [URI]
[4] Iadine Chades & Bertrand Bouteiller, "Solving Multiagent MDP: A Forest Management Example" [URI]
[5] Yishay Mansour, Satinder Singh, "On the Complexity of Policy Iteration" [URI]
[6] Stack Overflow, "Time Complexity of Value Iteration" [URI]
[7] Sven Koenig and Reid G. Simmons, "Complexity Analysis of Real-Time Reinforcement Learning?" [URI]
[8] Microsoft, UCB, "Is Q-learning Provably Efficient?" [URI]