CS 7641 Assignment 4: Markov Decision Process

Ali Alrasheed
College of Computing
OMSCS

I. Introduction

This paper explores two different Markov Decision Processes (MDPs) which are the frozen Lake from OpenAI gym, and the Forest Management System from the MDP Toolbox module. Model-based algorithms such as Value and Policy Iteration will be used to solve the two MPDs problems. In addition, Q-learning (model-free) will be also used to solve the same MDPs. The performance of these three algorithms will be compared, discussed, and analyzed in this paper.

Before diving deep into the two MDPs problems, the following subsections will briefly discuss some important concepts such as Markov decision Process (MDP), Value iteration (VI), Policy iteration (PI), and Q-learning.

A. Markov Decision Process (MDP)

Markov Decision Process (MDP) is a mathematical technique to describe decision-making processes which are controllable but also scholastic (usually but not necessarily). MDPs are defined by a set of States (S), Actions (A), Transition Probabilities (T), Rewards (R), and discount factors (γ). In addition, MDPs must obey the **Markovian Property** which states that the transition from the current state to the next state is only dependent on the current state regardless of the history (previous states). In Reinforcement Learning (RL), MDPs are very useful since the majority of the RL problems can be modeled as MDPs which help sloving them using common algorithms such as value iteration, policy iteration, and Q-learning.

B. Value Iteration

Value Iteration (VI) is one of the algorithms used to solve MDPs by finding the optimal state value function $V(S)$ for each state in the MDP. VI starts with a random value function and then iteratively updates the value function using Bellman equations shown below.

$$V(s) \leftarrow \max_{a} \sum_{s',a'} T(s,a,s')[r + \gamma V(s')] \tag{1}$$

Equation 1 is updated iteratively until the changes are lower than a threshold. The convergence of Value Iteration is guaranteed since the value function in each iteration can only get better but never not worse. It is also optimal since it always finds the value function that maximizes the discounted future rewards which is used to find the optimal policy.

C. Policy Iteration

Policy Iteration is another technique to solve the MDPs. It starts with a random policy π and then iteratively performs two steps: policy evaluation and policy improvement. The first step involves calculating the new value function using the current best estimate of the policy as it is shown in equation 2. The advantage of updating the value function using equation 2 as opposed to equation 1 is that by getting rid of the maximum term, equation 2 can be solved using linear programming techniques, and thus policy iteration is usually simpler and sometimes faster than Value Iteration.

$$V(s) \leftarrow \sum_{s',a'} T(s,\pi(s),s')[r + \gamma V(s')] \tag{2}$$

The second step involves a one-step look-ahead to update the current policy based on the new value function estimation from equation 2. The policy improvement equation is shown in equation 3.

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s',a'} T(s,\pi(s),s')[r + \gamma V(s')] \tag{3}$$

Similar to Value iteration, Policy Iteration is also guaranteed to be the optimal policy given it runs enough iterations.

D. Q-Learning

So far we introduced two model-based RL algorithms: Value iteration, and Policy iteration. Although VI and PI converge to the optimal policy, they can only be used when the transition function of the MDP is given. However, the transition function of the MDP is not always available, thus, model-free algorithms can be used when the transition function of the MDP is unknown. Q-Learning is only of the well-known model-free RL algorithms that is used to estimate the value function by exploring the environment. In other words, Q-learning relies on exploring and then exploiting the environment to estimate the value of the state for each action $Q(S,A)$. Bellman equation is used to update the Q(S,A) value as it is shown in equation 4.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{\bar{a}} Q(\bar{s},\bar{a}) - Q(s,a)) \tag{4}$$

where α is the learning rate that determines how fast and slow the learning will be (updating the Q values). Another important dilemma that arises with model-free algorithms is exploration vs exploitation. In order for Q-learning to learn

the Q-values of each action given a state, it needs to explore enough in the environment. However, exploiting the environment (using the maximum action given the current estimate of the best policy) is also important to speed up the convergence, especially in sparse environments where rewards are only collected at the end. In this paper, ϵ-greedy approach will be used for choosing the next action for exploration. Typically in ϵ-greedy, the agent starts will by performing random actions and then slowly shifts toward actions that maximize expected return.

II. Frozen Lake

Frozen Lake is a grid-based RL environment from OpenAI gym [1]. It involves walking into a frozen lake until reaching the goal without falling through any hole. The environment is divided by an NxM grid, and each position in the grid can be one of the following types: Start (S), Goal (G), Hole (H), or Free (F). There are also only four actions allowed (moving left, moving right, moving up, and moving down).

What makes this environment interesting is that the rewards are very sparse. This means that the agent can move through the environment for many steps and get no rewards. This makes it difficult for any algorithm to learn the optimal policy. In addition, the environment is very stochastic. For example, when the agent wants to move to the left, the probability for this action will be as follows: P(left) = $\frac{1}{3}$, P(up) = $\frac{1}{3}$ and P(down) = $\frac{1}{3}$, and so on. This makes the environment more challenging to solve. In addition, this environment will be particularly hard for Q-learning since it contains terminating states (Holes and Goals) once that agent reaches them there is no escape. This will affect the exploration strategy as the agent can get stuck in one state for a long time. To solve this, the Q-learning from the MDP toolbox module was modified to reset the states if the agent stayed in the same state more than 5 times.

Fig. 1: 5x5 Grid of frozen Lake Environment

A frozen lake with a 5x5 grid will be used to evaluate the performance of value iteration, Policy iteration, and Q-learning. A visualization of the environment is shown in figure 1.

A. Value Iteration and Policy Iteration

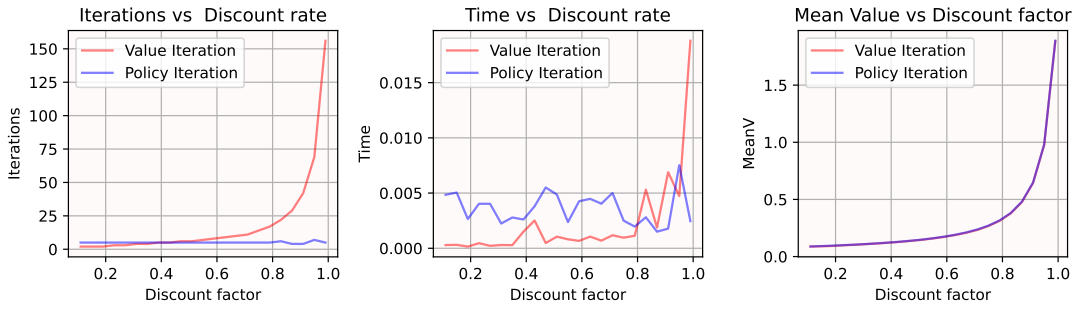
In this section, the performance of VI and PI will be explored using the frozen lake environment shown in figure 1.

There are two important parameters for both VI and PI which are the discount rate and the stopping criterion (ϵ). The discount factor tells us how much we value future rewards. A discount factor of close to zero means that we do not care about future rewards, whereas a value close to one means that we value feature rewards almost as much as current rewards. The discount factor is strictly less than one since it makes the sum of infinite rewards finite which helps guarantee the convergence of PI and VI. The second hyper-parameters is the stopping criterion (ϵ) which is used to decide convergence for VI and PI. Detailed experiments varying both above-mentioned hyper-parameters can be seen in figure 2.

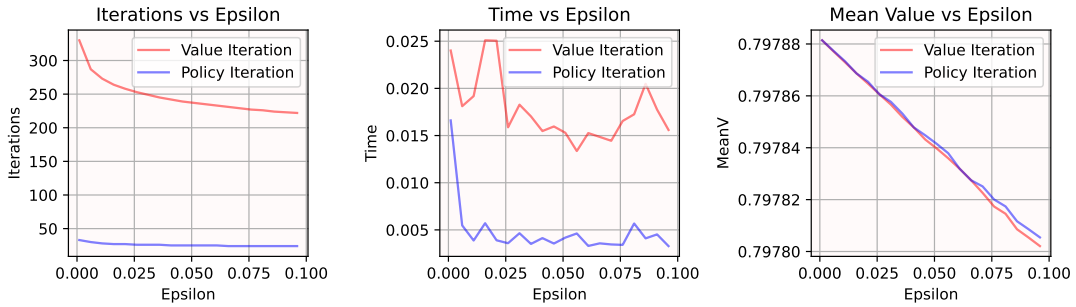
The effect of changing the discount factor is shown in the plots of figure 2a. It is important to point out that both VI and PI reach the same mean value function. In addition, It can be observed from figure 2a the performance (mean value function) is very poor with a low discount factor. Since the environment is very sparse (collecting rewards only when reaching the goal), having a low discount value means that reaching the goal is not important since the collected rewards will be very small. In addition, this will cause both VI and PI to converge faster since the changes in the value is already small because of the low discount factor. On the other hand, a higher discount factor will allow VI and PI to emphasize the rewards obtained from reaching the goal and back-propagate it through the grid. This results in a higher mean value function compared to the low discount rate case. However, since there is no free lunch, the time complexity required for convergence also increases especially for VI. It is also interesting to observe that the number of iterations and time complexity of VI was very sensitive to changing the discount rate, unlike the PI. This is because PI converges whenever the policy stops changing, whereas value iteration converges when the value function changes are less than a threshold.

Furthermore, the effect of changing the stopping criterion (ϵ) is shown in the plots of figure 2b. It is obvious from the figure that the mean value function does not increase much with a very low (ϵ) value (lower than 0.1). However, the time complexity and the number of iterations increase significantly with very lower (ϵ). Thus, it is important to balance the trade-off between the stopping criteria and finding the optimal policy.

Based on the result in figure 2, a discount factor of 0.9999 and ϵ of 0.1 will be used for both VI and PI. The convergence plot of VI and PI is shown in figure 3. Figure 3 shows that both algorithms reach the same mean Value. This is expected since both algorithms are guaranteed to converge to the optimal policy. However, it is interesting to see that PI is capable of converging to the optimal values much faster than VI. This is due to the fact that PI is a less complex algorithm that can be solved with linear programming. In addition, VI takes



(a) Discount factor



(b) Stopping Criterion (epsilon)

Fig. 2: PI and VI hyper-parameters

significantly more iterations to converge with a higher number of discount factors since the propagation of the value from the goal will be more significant. In addition, PI converges faster since its stopping criterion is when the policy stop changing, unlike VI which only converges with the change in values is less than ϵ.

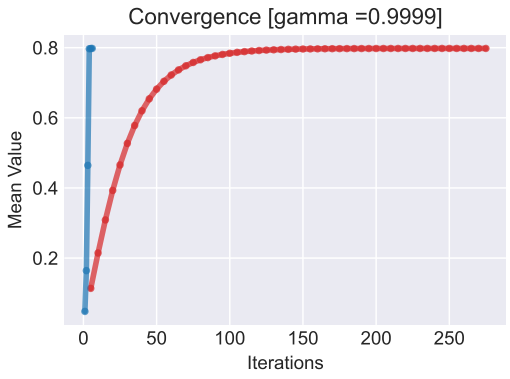


Fig. 3: Convergence graph for VI and PI

Figure 3 shows that VI needs significantly more iterations to converge than PI. This is also reflected in the time complexity of convergence. Figure 4 shows that PI converges significantly less time to converge than VI which also emphasizes the advantage of using PI. However, it is not always the case that PI is faster than VI. Figure 2a shows that with a low-value discount factor, VI tends to be faster than PI.

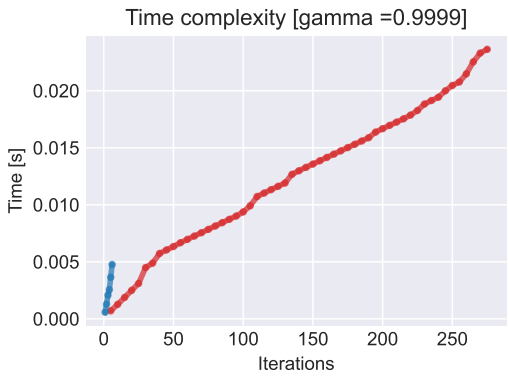


Fig. 4: Convergence Time for VI and PI

B. Q-Learning

There are some essential hyper-parameters to be explored in the Q-learning algorithms such as the number of iterations, exploration decay factor (epsilon decay), learning rate, and learning rate decay.

An essential parameter of the Q-learning algorithm is the epsilon decay rate which determines the exploration vs exploitation strategy. Q-learning typically starts ϵ close to 1 meaning that the agent mostly performs random actions to explore the environment. However, the exploration coefficient ϵ typically decay as the number of iteration increases to allow the agent to exploit what it has learned about the environment and hence accelerate the convergence. Figure 5 shows the sensitivity of Q-learning to different epsilon decay factors.

It is clear from the figure that a value close to 1.0 allow Q-learning to converge faster. This is because, with a high exploration rate, the agent will have a higher chance to explore the environment and hence higher chance to find the goal as well. It is also interesting to observe that even high decay rate (0.8), Q-learning will be able to converge to the optimal solution found by both PI and VI. However, this is because the environment is relatively small with only 25 states. In the case of a larger and more complex environment, a high exploration decay rate such as 0.8 will lead to poor learning as the agent will not have enough steps to explore the environment.

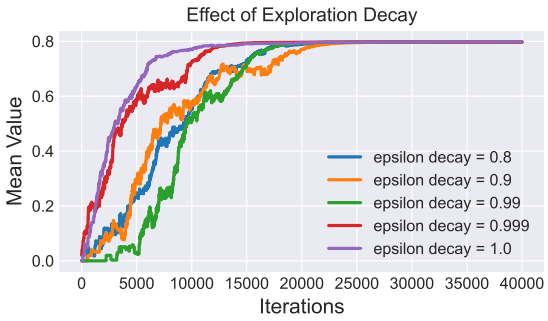


Fig. 5: Effect of ϵ Decay on Mean Value function

Another important parameter of Q-learning is the learning rate which determines how much the Q values should be updated per step. A very high could lead to instability in the Q value as well as an overly optimistic value of Q. Thus, it is important to select an appropriate value of the learning rate in order for the Q-learning to converge to the optimal Q values. Figure 6 shows the effect of the learning rate on the function of the mean value. It can be seen from the figure that, the higher the learning rate the faster the learning and convergence of Q-learning. However, if the value is too high such as alpha =2.0, it will lead to very bad learning and high instability.

Fig. 6: Effect of Learning Rate on Q-learning Performance

In the previous figure 6, the learning rate was assumed to be constant for all the steps. However, it is also common to apply some decay to the learning which iteratively reduces the magnitude of changes applied to the Q values, and hence improves the stability of the learning. Figure 7 shows the effect of decaying the learning rate at different rates. It is clear from the figure that it is possible to have stable and fast learning with a constant learning rate (alpha decay of 1.0). However, this is will be dependent on the initial learning being not too high, in this case, it is 0.5. Figure 7 also shows that learning rate decay close to 1.0 produce good results as well.

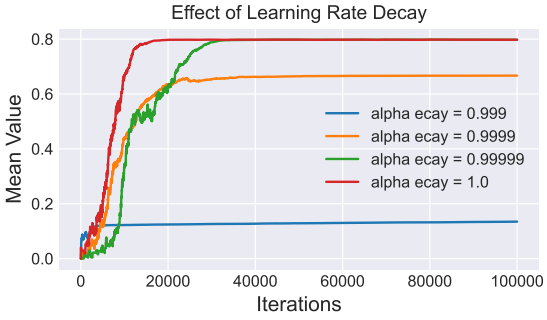


Fig. 7: Effect of Decaying the Learning Rate

Finally, it was clear from figure 5, 7, and 7 that Q-learning needs significantly more iterations than both VI and PI since Q-learning is model-free which means that it needs significant exploration in the environment to produce good results. However, the time complexity is linearly proportional to the number of iterations as it is shown in figure 8. This is will be a huge drawback for Q-learning with large states as it will need a significant number of iterations to converge to the optimal Q value which is not always feasible due to time and computational constraints.

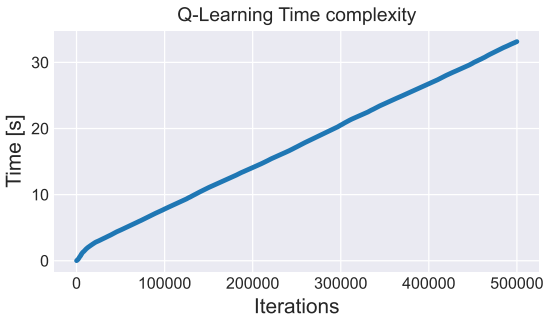
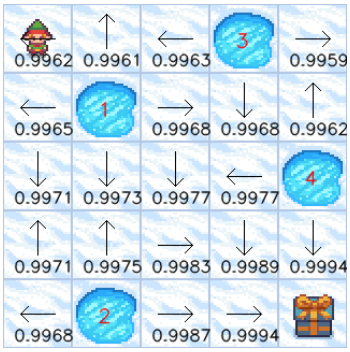


Fig. 8: Time Complexity of Q-learning

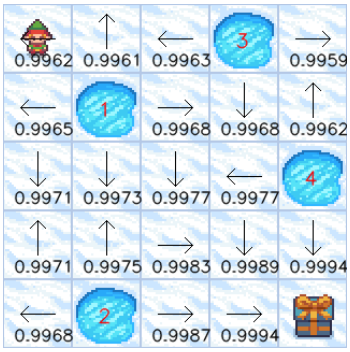
C. Performance Analysis - Small Grid

Figure 9 shows the visualization of the policy of each algorithm for the small 5x5 grid (25 states). All algorithms were able to produce the same policy as can be observed in figure 9. Since PI and VI are guaranteed to find the optimal policy, then the Q-learning also produced an optimal policy for the small frozen lake. There are a couple of interesting behaviors that can be observed in the optimal policy. For example, the optimal policy in figure 9 indicates that the best way to start is moving toward the wall instead of going directly down. This is because moving directly down is risky since

(a) VI Policy



(b) PI Policy



(c) Q-Learning Policy

Fig. 9: Policy Visualization

there is a chance of falling through hole 1 while moving toward the wall might increase the number of steps to reach the goal but eliminate the risk of actually falling through a hole. This also applies to the left and upward cells of holes 4 and 2 respectively. Another observation is that any grid cell that can lead to one of the holes has an opposite direction from the hole (to move away from the holes).

Table I shows the summary of the small frozen lake experiment. Although all algorithms produced almost the same mean value function, the table clearly shows the superiority of PI in terms of time and number of iterations compared to both VI and Q-learning. It must be noted that VI is not always slower than PI but it depends on the chosen discount factor as was shown in figure 2a. However, in this particular problem, it makes sense that we have a discount factor near 1 since getting to the goal is the only ultimate aim of the environment. In addition, Q-learning takes significantly more time to arrive at the same as VI and PI but this is expected since Q-learning is a model-free algorithm meaning it has to figure out how the transition function and Q values directly from experience that requires many exploration in the environment and hence the high number of iterations.

TABLE I: Summary of Small Frozen Lake

Algorithm	Mean V	Number of Iteration	Time [s]
VI	0.7979	275	0.02
PI	0.7979	6	0.006
Q-learning	0.7968	20000	1.48

D. Large Frozen Lake Environment

This section will focus on the performance of the VI, PI, and Q-learning when the environment becomes significantly larger. We go from a 5x5 gird (25 states) to a 40x40 grid (1600 states). Visualization of large frozen lake can be seen in 10.

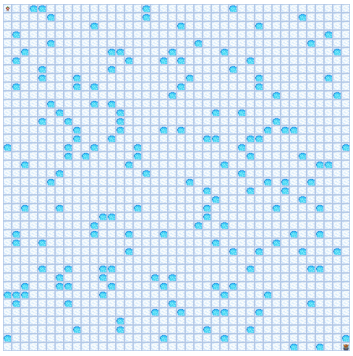


Fig. 10: Large Froze Lake Environment

Figure 11 shows the performance of PI and VI in the large environment. Since PI and VI are model-based algorithms, they are guaranteed to find an optimal policy. Since this is a much larger MDP than the previous one (1600 states compared to only 25 states), both PI and VI took more iterations to converge.

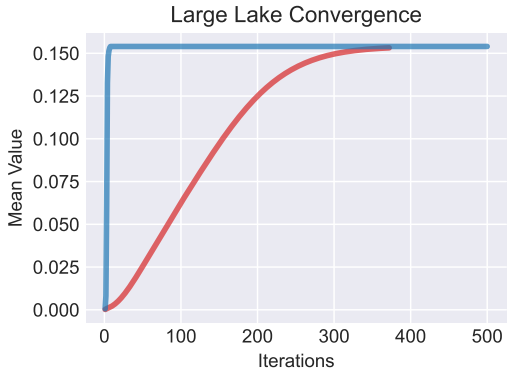


Fig. 11: Convergence Graph for PI and VI in the large frozen Lake

However, PI exhibited an interesting behavior. Although it seems that It is supposed to stop at around ten iterations, it kept going until it reached the maximum number of iterations. That is because of the nature of frozen lake which could

have multiple optimal policies, and hence PI keeps alternating between these optimal policies forever. A modified version of PI will be used to eliminate this issue by also checking if the value function changes. This is reflected in table II which shows that the modified PI takes significantly fewer iterations than what is shown in figure 11

In addition, such a large MDP (1600 states) will be a huge challenge for model-free algorithms. In addition, the frozen lake environment also makes it significantly more challenging by having holes that agent could be stuck there forever as well as the high stochasticity of the actions. In addition, the rewards function of the frozen lake is very sparse as the only reward the agent gets is when reaching the goal, and hence it is possible that the agent preforms many random actions and learns nothing since it never was able to figure out where is the goal. To solve the issue of getting stuck forever in one of the holes, the Q-learning algorithms from the MDP toolbox were modified to reset their states if it gets stuck in the same state more than 5 times. This will significantly improve the learning of the agent. Figure 12 shows the performance of Q-learning with large size frozen lake. As expected, the Q-learning algorithms failed to produce a similar mean value function as the VI and PI in figure 11. This is because Q-learning will need a significantly large number of steps to explore the environment and hence estimate the Q values which needs significant computational power and a long time, especially with sparse rewards function.

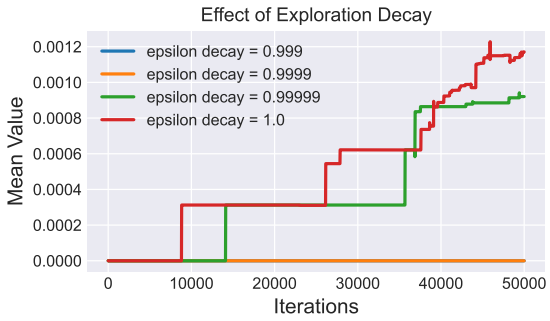


Fig. 12: Q-learning Performance of the Large Frozen Lake

Finally, the summary of the large frozen lake experiment can be found in table II. Once again, although VI and PI have very similar mean value functions, the modified PI algorithms were able to converge faster than VI. As stated earlier, the modified PI involves adding a stopping criterion which is when the value function changes less than a threshold. In addition, the table shows that Q-learning has very poor performance compared to both VI and PI although 50000 iterations were performed. To improve the performance of Q-learning more exploration steps are needed which will depend on the time and computational constraints. In addition, rewards shaping for intermediate actions can be used to help the agent's exploration, and hence improve the result of Q-learning.

TABLE II: Summary of Large Frozen Lake

Algorithm	Mean V	Number of Iteration	Time [s]
VI	0.1532	372	3.14
PI (modified)	0.1539	49	2.13
Q-learning (modified)	0.0023	50000	34.45

III. Forest Management System

The second MDP describes a forest Management system that aims to protect the forest for wildlife but also generate money by cutting some trees. The states of the MDP are defined by the number of years that will be managed. In addition, the MDP consists of two actions "Wait" (action 0) and "Cut" (action 1), and one of these actions can be performed by the end of each year. However, the "Wait" action is not deterministic as there is a probability p that a fire occurs and burns the whole forest which will return the MDP to the first state 0. There are two main rewards to this problem. The first is r1 and it is given when the forest reaches its oldest (last state) and the action is "Wait". The second is r2 and it is given when the forest reaches its oldest but the action is "Cut". The default for r1 and r2 is 4 and 2 respectively. There are also intermediate rewards of 1 for cutting the forest after the first state and before the last state.

To solidify the understanding of this MDP, the transition and rewards metrics will be demonstrated. The transition matrix P is defined in a form of (Action,State,Next State) as follows:

$$P[0,:,:] = \begin{pmatrix} p & 1-p & 0 & \dots & \dots \\ p & 0 & 1-p & 0 & 0 \\ \vdots & \vdots & 0 & \ddots & \vdots \\ p & \vdots & \vdots & \vdots & 1-p \\ p & \vdots & \vdots & \vdots & 1-p \end{pmatrix} \tag{5}$$

$$P[1,:,:] = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix} \tag{6}$$

The matrix 5 defines the transition function for action 0 (waiting). The "Wait" action is stochastic with a probability of p that a fire occurs and burn the forest (return to state 0), and with probability 1-p that the forest transition to the next state (becomes one year older). However, the "Cut" action is deterministic and will always return the forest to its youngest state (state 0) which is demonstrated in matrix 6.

In addition, the rewards matrix comes in from of (State,Action), and it is defined as follows:

$$R[:,0] = \begin{pmatrix} 0 & \cdots & r1 \end{pmatrix}^T \tag{7}$$

$$R[:,1] = \begin{pmatrix} 0 & 1 & \cdots & r2 \end{pmatrix}^T \tag{8}$$

Matrix 7 shows the rewards waiting action which shows that we only get the r1 rewards for "Waiting" if and only if the

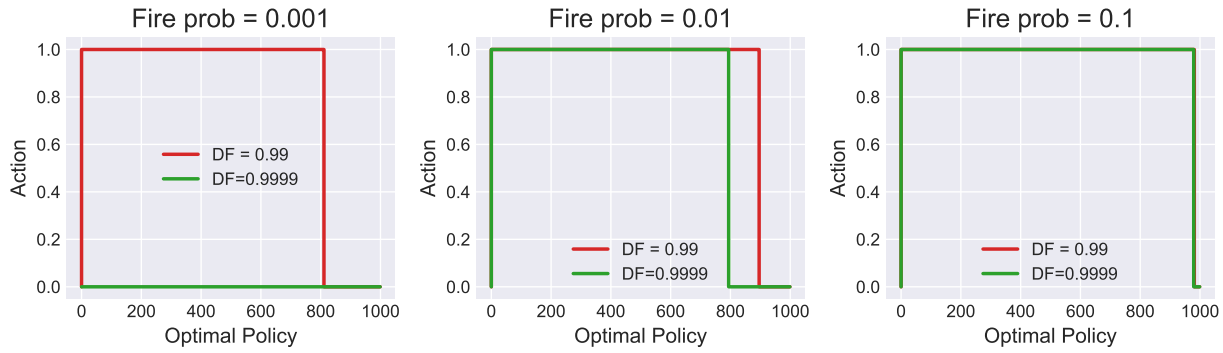


Fig. 13: Optimal Policy Change for Different Fire Prob and Discount rate

forest is maintained to the oldest state (last state), and we get 0 otherwise. Similarly, matrix 8 shows the rewards for "Cut" action which illustrates 1 is rewarded for cutting the forest at any state after 0 and before the oldest state, and r2 is rewarded if and only if the forest is cut at its oldest state.

This MDP is interesting since it demonstrates the dilemma of high-risk high-rewards problems which are present in many real situations. The stochasticity nature of this MDP due to the risk of firing occurring makes the MDP interesting since the longer we wait the higher the rewards will be but also the higher the risk of losing everything due to firs. Thus, solving the MDP involves figuring out when the waiting is no longer beneficial which will be strongly dependent on the number of years (oldest state), the discount factor of future rewards, and the probability of a fire occurring.

Figure 13 shows how the probability of fire and discount factor can significantly alter the optimal policy. Figure 13 shows optimal policy with 0.1%, 1%, and 10% of fire probability with two discount factors 0.99 and 0.9999. The figure clearly shows how reducing the probability of fire resulted in increasing the value of the "Wait" action as waiting is no longer as dangerous. On the other hand, increasing the probability of fire means that waiting does make sense as fire is more likely to occur and burn the forest before reaching its oldest state, and hence get no rewards. The discount factor indicates how the future rewards are valued compared to the present. Increasing the discount factors means that maintaining the forest is very important, and hence we can notice that the Cut action is replaced by the Wait action earlier as the number of discount factor increase which can be clearly seen in the middle plot of figure 13. The transition between Cut to Wait action means that in this state, the value of waiting until the final state of the forest is higher than cutting the forest. In other words, in this state, the expected reward is high enough that it is worth taking the risk of the forest burning.

In this report, the forest management system will be configured to have 1000 states with r1 of 4 and r2 of 2. In addition, the default probability of a fire occurring will be 0.1 unless otherwise specified.

A. *Value Iteration and Policy Iteration*

We will start by investigating the effect of changing the discount factor of future rewards. The discount rate is very crucial in this problem since choosing a low discount factor will result in prioritizing making money since the reward of waiting for the oldest state will be negligible after discounting. On the other hand, a discount factor near 1 will encourage waiting but the waiting will be greatly dependent on the probability of fire and the state size. For example, even with a high discount factor, a high probability of fire will force the forest management system to accelerate the utilization of the forest by cutting the trees before the fire occurs.

As expected, figure 14 shows that with a higher discount factor, the mean value function increased rapidly since future rewards are worth more. Similar to the frozen lake, VI took more iterations that converge compared to PI. However, the computational complexity of PI becomes more obvious in the forest management system as PI constantly took more time to converge than VI but with much less iteration. PI involves solving a set of linear equations in every iteration which will be significant as the number of states increases. Another interesting finding is that VI and PI do not converge to the same value function. However, this is expected since the MDP always for getting infinite discounted rewards by waiting for the last state forever. Thus, the mean value function for each algorithm will greatly depend on their convergence criteria. It is crucial to mention that although the mean value function of VI and PI do not match, both algorithms arrive at the same optimal policy.

Figure 15 shows the result of running PI and VI on the forest management system MDP. The default configuration of the MDP is used with a discount factor of 0.99 as stated at the start of the section. Figure 15 also shows the magnitude of the changes in the value function for each iteration until convergence. The value iteration (VI) uses this error to decide to converge if it is until a certain threshold (T) which can be calculated using the following equation:

$$T\ (VI) = \epsilon * (1 - discount\ factor)/discount\ factor \quad (9)$$

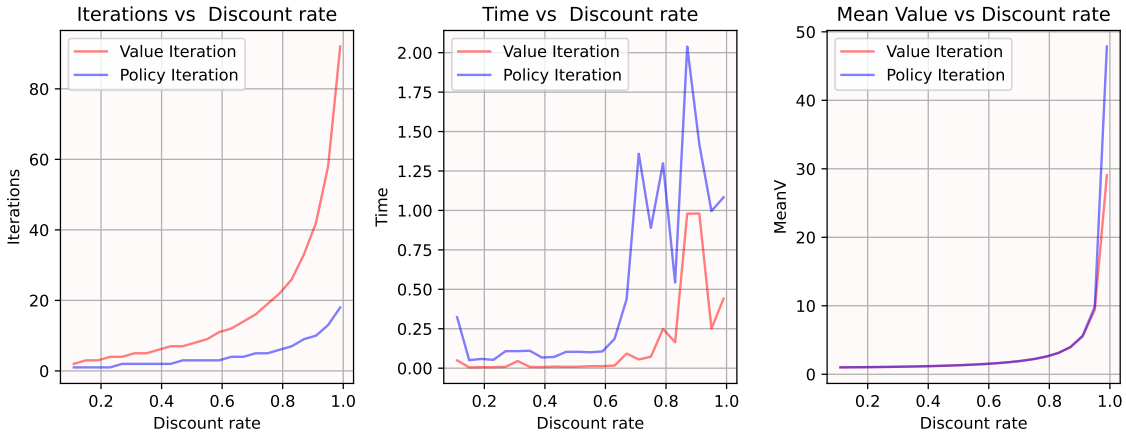


Fig. 14: Forest: Effect of Discount Factor on VI and PI

The default ϵ is 0.01, and the discount factor is 0.99, thus the convergence threshold (T) is 0.0001. This is too small which causes the VI to take more than necessary to converge as it is seen from the figure 15. In addition, the convergence criterion for PI is not based on the change in the value function director but on when the policy stops changing. However, it is intuitive that the policy will converge once the change in the value function is not significant to alter the policy. Overall, figure 15 shows the change in value for both algorithms tends to reduce in every iteration until convergence although the convergence criterion for VI seems a bit too small.

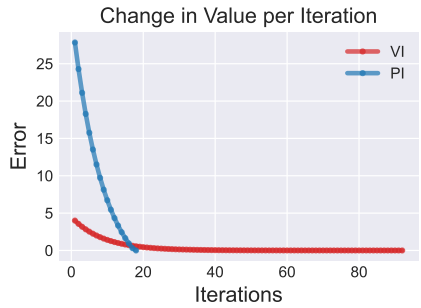


Fig. 15: Change in Value Function per Iteration

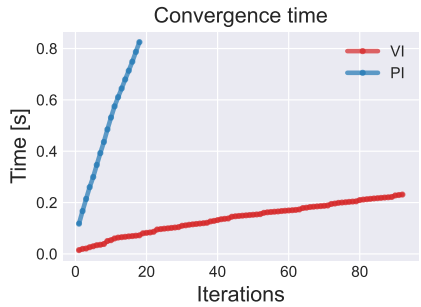


Fig. 16: Convergence Time for VI and PI

The time complexity of the experiment in the figure above 15 is shown in figure 16. Although PI needed much less iteration to converge, it took significantly more time to converge than VI which could be a drawback of PI with very large MDPs.

Finally, figure 17 shows the visualization of the optimal policy derived from both PI and VI. The x-axis shows the states from 0 to 999 (1000 in total), and their corresponding optimal actions (0 or 1). As expected, both PI and VI agreed on the same policy. This is expected since VI and PI are guaranteed to converge to the optimal policy. This is a particular configuration of the MDP (probability of fire is 0.1), the optimal action is to cut the tree immediately (at state 1) since waiting 1000 years with a very high probability of fire is very risky. The probability of fire not occurring at least once in 1000 years is slim to none. However, what is interesting is that optimal policy shows that waiting is recommended near the end of the MDP (last 18 years) since the rewards of waiting will be higher than the risk of losing the forest.

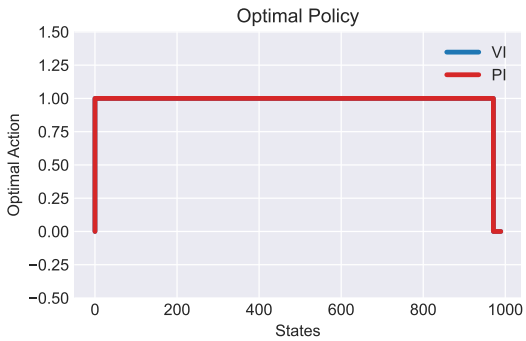


Fig. 17: Optimal policy Derived for VI and PI

B. Q-learning

Q-Learning has been extensively analyzed in the Frozen Lake section. Thus, in this section, the effect of its hyperparameter will be briefly discussed again. We found out from

the frozen lake section, that Q-learning was able to solve the same grid problem with only 25 states but struggled to solve the large grid (1600 states). The reason is that Q-learning is a model-free method that requires exhaustive exploration of the environment and trying many random actions to figure out the optimal policy. Thus, Q-learning is not guaranteed to converge to the optimal policy.

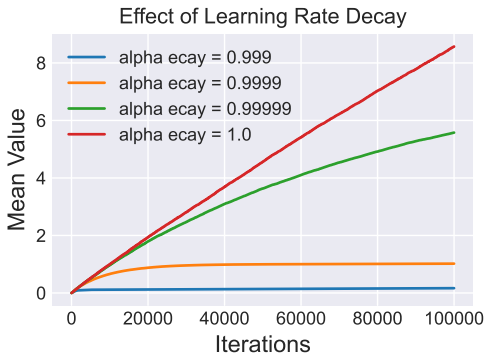


Fig. 18: Effect of Learning Rate Decay on Mean Value Function

Figure 18 shows how decaying the learning rate (alpha) may help Q-learning to converge by slowing the learning over iterations. However, convergence does not mean optimal, thus it might make more sense to give each learning an equal weight (alpha decay=1.0). However, Q-learning with a constant learning rate was not able to converge as the value function kept increasing with more iterations. This suggests that for the Q-learning to figure out the optimal policy for the forest management system with 1000 states, it needs to perform significantly higher number of iterations (exploration).

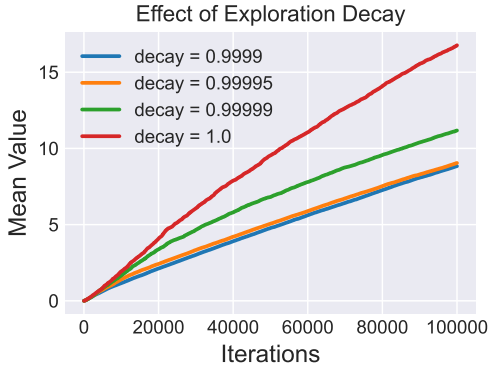


Fig. 19: Effect of Exploration Decay rate on Mean Value Function

The ϵ-greedy algorithm is used to balance between exploring and exploiting the environment. ϵ-greedy starts with full exploration and then the exploration decays as the number of iterations increases. This allows Q-learning to exploit what it has learned previously which may help accelerate the learning. Therefore, it is important to choose the appropriate decay rate as too fast decay will result in a sub-optimal policy. On the other hand, too slow decay will increase the number of iterations needed to reach the optimal policy. Figure 19 shows the effect of the exploration decay rate on the mean value function. The figure shows that Q-learning with no exploration decay (epsilon decay =1) was able to produce the height mean value function among the other decay rate. This suggests the number of iterations is very small to benefit the ϵ-greedy algorithm, and thus more exploration is needed. However, the issue with more exploration is that the time complexity also increases linear with the number of iterations similar to what has been shown in figure 8.

The Q-learning algorithm was run for 0.5M iterations, and the delta of the value function was recorded and plotted in figure 20. The figure clearly shows that the Q-learning was not able to converge as the change of value function per iteration is still very significant. The figure might suggest that Q-learning might need around 10M iterations to solve such simple MDP which is going to require high computation power and time.

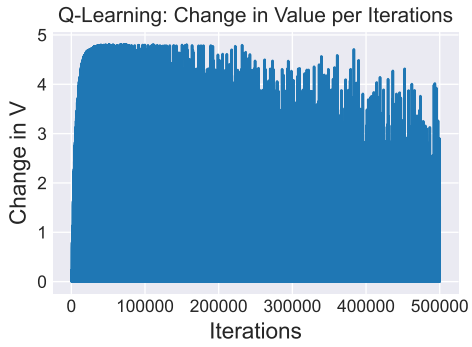


Fig. 20: Q-learning delta convergence

As the Q-learning algorithm was not able to converge within the 0.5M iterations, it is expected that the Q-learning algorithm will produce a sub-optimal policy. This is evidence from figure 21 which shows the policy derived from the Q-learning is different from the optimal policy derived from PI and VI shown in figure 17.

Fig. 21: Q-learning sub-optimal policy

C. Performance Analysis - Large Forest

It has been shown that both Value and Policy iterations were able to solve the forest management MDP and find the same optimal policy. This is not surprising since PI and VI are model-based algorithms, and they are guaranteed to converge to the optimal policy given they run sufficient iterations. On the other hand, Q-learning struggles to converge suggesting that it needs significantly more iterations to be able to solve the MDP. This resulted in Q-learning producing a sub-optimal policy.

The summary of the large forest MDP can be found in table III. The table shows although PI and VI have different mean value functions, they still reach the same optimal policy. The different value function is caused by the fact that the rewards system is configured to give infinite discounted rewards, unlike the frozen lake MDP. In addition, in the frozen lake, it was found out the PI was preferred over the rest of the algorithm since it found the optimal policy in the shortest time. However, this is not the case in the large forest MDP. The table below shows that V-iterations were able to converge in less time although it took more iterations to do so. Thus, in this case, VI is preferred for the forest management system. Furthermore, the table shows that Q-learning was not able to find the optimal policy as it needed significantly more exploration points to find the value of the states relative to each other.

TABLE III: Summary of Small Frozen Lake

Algorithm	Mean V	# of Iteration	Time[s]	Optimal Policy
VI	29.06	92	0.3	Yes
PI	47.85	18	1.09	Yes
Q-learning	30.37	500000	57.2	No

D. Small Forest

Since PI and VI have proven to work in a large forest, this section investigates the performance of Q-learning in the small forest. The forest was reduced from 1000 states to only 25 states.

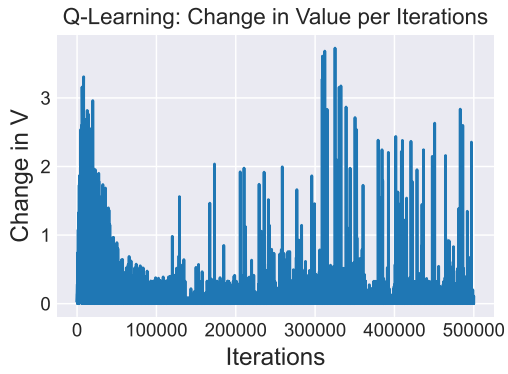


Fig. 22: Q-learning sub-optimal policy

The delta convergence graph of Q-learning can be seen in figure 22. It is clear from the graph that even with 0.5M iterations, Q-learning struggled to converge. Q-learning algorithm seems to struggle with MDPs where there are two conflicting goals since the ϵ-greedy exploration strategy tends to get confused and constantly get stuck on local maximum toward one kind of reward.

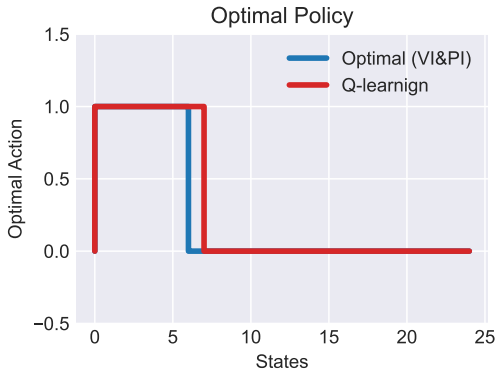


Fig. 23: Q-learning policy vs Optimal Policy

However, although Q-learning was not able to converge, it produce a small policy generated from VI and PI which can be seen in figure 23.

IV. Conclusion

This report discussed the performance of three common reinforcement learning algorithms on two different MDPs. In particular, PI, VI, and Q-learning were applied to frozen lake and forest management system MDPs. It was found that model-based algorithms (VI and PI) were able to find the optimal policy for both MDPs with different sizes. This is not surprising as VI and PI are guaranteed to find the optimal policy given they run sufficient iterations. On the other hand, model-free (Q-learning) was only able to find the solution for the small frozen lake, while failed in all the other cases. However, model-free is still very useful as the transition function of the MDP is not always available which means that model-based algorithms cannot be used.

Finally, it was evident from the frozen lake experiment that PI was the best algorithm to use since it was able to find the optimal policy in the shortest time. However, the forest management system, VI was the fastest, and thus it is preferred over PI.

References

[1] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv: 1606.01540.