Assignment#4, CS7641

Georgia Institute of Technology, Fall 2020

Asif Rehan, arehan7@gatech.edu

1. Introduction

In this assignment, two non-episodic (continuous) tasks are modeled as MDP. To find the optimal policy for these MDPs, two model-based policy finding Reinforcement Learning (RL) algorithms: Policy Iteration (PI), Value Iteration (VI) are contrasted against the model-free Q-Learning algorithm which does not require the explicit models of the Transition and Reward matrix unlike the priors.

2. Background Concepts

2.1. Markov Decision Process (MDP)

MDP is a mathematical framework to model discrete-time stochastic systems. MDP consists of some States (S), Actions (A) to be taken while at each state, Transitions (T) from a state, s, to another state, s', by taking an action with a probability T(s,a,s'). At each state, the agent who is interacting with the system, collects a reward R(s), or R(s,a), R(s,a,s'), depending on the stochasticity of the system.

A MDP is solved when the best policy, $\pi(s)$ is identified for every state. Policy means the action the agent decides to take when at a state so the probability of the total reward in the first equation below is maximized as shown in the second equation below.

$$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s')\right) \qquad \text{Eq (1)}$$

$$\pi(s) := \operatorname{argmax}_a \left\{ \sum_{s'} P(s' \mid s, a) \left(R(s' \mid s, a) + \gamma V(s')\right) \right\} \qquad \text{Eq (2)}$$

2.2. Discount Factor γ

The discount factor γ in Eq(1) and Eq(2), adds the weight on the expected reward after the transition. A discount factor of 1 means the rewards from now and forever is both equally important. It is known as Infinite Horizon.

If the discount factor is 0, then only the reward in current state is considered. So it is a greedy approach. If there is a discount factor $0 < \gamma < 1$, then a finite horizon is considered with a discounted weight on the future returns. It is an important parameter to model a MDP in practice.

2.3. Policy Iteration (PI)

Policy Iteration (PI) algorithm needs the T and R matrices. It starts with a policy to start with and updates the Value function as in Eq(1) and then finds the best policy from Eq(2) until the new and previous policies are the same. This is a straightforward iterative process.

2.4. Value Iteration (VI)

VI also needs T and R matrices. But it can bypass the need to find the policy in each iteration. It initializes a random V(s) matrix with random Bellman function value for each state. Then for each state it finds the best action and updates the V(s) until all the V(s) are stable. This way it can quickly find the best policies for each state that stabilizes the V-matrix, instead of having to explicitly find the policy and then compute V(s) as in PI. It might be faster than PI for same iteration but may often take more iterations.

2.5. Q-Learning (QL)

Unlike PI and VI, QL is realistic that often the actual T and R matrices are not known in a real world problem. Instead an agent can only interact with the environment and try to update its impression about the system by updating its T and R matrix.

$Q[s, a] = Q[s, a] + \alpha*(R[s,a] + \gamma*Max[Q(s', A)] - Q[s, a])$	Eq(3)

In QL, at each state, the agent takes a random action or the best action that maximizes the V(s). This is known as **Exploration Vs Exploitation** controlled by the parameter ε which determines whether the agent chooses to randomly explore a new action or take the best action from what is has seen**.** The agent takes random actions for probability ε and greedy action for probability (1-ε) If it does not explore in the beginning then it keeps taking the same decision based on its first action and gets stuck. So initially, more exploration is useful to make a rich experience of the environment. As the agent explores and takes random actions over and over, it can gradually become confident about its experience and may not need to explore as much. So, the parameter ε can be reduced gradually.

The Learning Rate parameter $0<\alpha \leq 1$ helps update the previous Q(s,a) value with the new value. Lower value makes convergence slow and too high of a alpha value makes it fluctuate the V(s) a lot.

3. Problems

3.1. Forest Management:

Forest Management is a non-episodic (continuous) problem with no termination state. For this `mdptoolbox.example.forest(S=3, r1=4, r2=2, p=0.1)` is used. It generates a MDP example based on a simple forest management scenario.

A forest is managed by two actions: 'Wait' and 'Cut' for each year. Each year there is a probability p that a fire burns the forest.

The model assumes {0, 1,…, s-1 } states of the forest. Here s-1 is the oldest state. Action 'Wait' is 0 and is 'Cut' is 1. After a fire, the forest goes to newest state, i.e. 0.

Here, r1 is the reward when the forest is in its oldest state and action 'Wait' is performed. The default value of 4 is used and r2 is the reward when the action 'Cut' is performed, again the default reward of 2 is used. There is a 10% probability of wild fire occurrence, p. So |A|=2 for FOREST problem. The number of actions is kept a constant for the problem.

For FOREST MDP, the states mean the age of the forest. Since in real world, it is not realistic to wait for more than a few decades before taking a Cut action which will set the age of the forest to the initial state

0, so a small value for the states is meaningful. So the set of number of states that was used is {2,4,8,16,32,64}.

3.2. Random MDP

Random MDP is a non-episodic (continuous) problem with no termination state. For this **`mdptoolbox.example.rand(S, A)`** is used which generates a set of transition and reward matrices for a random MDP problem with more than 1 states and 16 actions i.e. |A|=16. The number of actions is kept a constant

This random MDP is important to contrast the well-defined, interpretable forest management problem. It was of interest how the three RL algorithms perform for a random MDP problem.

For Random MDP, interpretability is not required as in Forest Mgmt. So a higher number of states are considered, i.e. |S| in {64,128,256,512,1024,2048}.

4. Analysis and Results

4.1. Higher Discount Factor Increases Rewards

First, a range of discount factors are used across a range of states for each of the two problems. For both of these problems, the discount factor values used are in this set = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99}. A discount of 1.0 does not guarantee convergence of VI and PI algorithms. The total rewards for each discount factor is plotted below for each state.

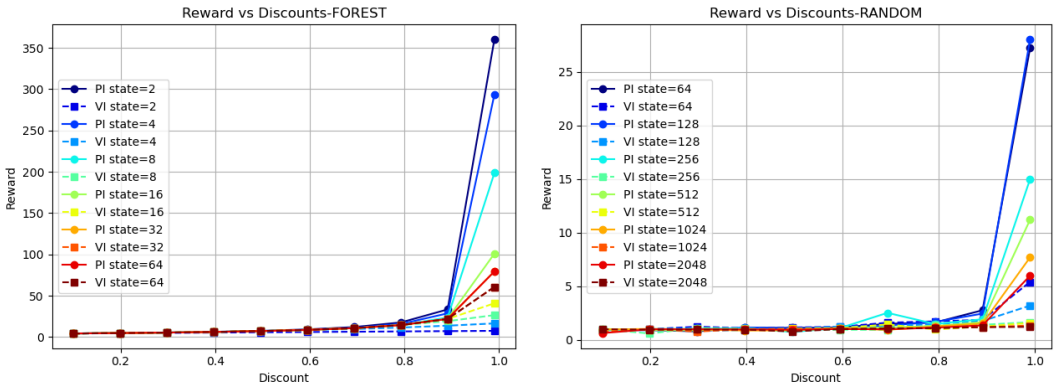

Figure 1: Reward, Discount Factor, Number of States, for PI and VI

As expected from Eq(1), total rewards should be maximum with higher gamma as future rewards have more weights as well as current rewards. The charts above show proof for that, irrespective of the number of states, the reward is maximum at discount=0.99 for both MDPs. Only for FOREST problem the total reward is higher than RANDOM problem because the reward values are higher for FOREST.

4.2. Rewards from PI > Rewards from VI

From the same Fig 1 above, for both problems, FOREST and RANDOM, the rewards for VI is smaller than the reward for PI. This is likely because PI requires both the policy and the Bellman Equation i.e.

V(s) matrix to stabilize. On the other hand, VI does not need to explicitly assume a policy, instead just a V(s) that is stable still finds the same optimal policy.

4.3. Impact of States on Performance in PI and VI:

Fig(2) below shows the runtime and iterations for PI and VI for FOREST and RANDOM problems. Checking the **blue lines** for Time vs State below, for both problems, PI runtime is much higher than VI runtime and PI time complexity increases exponentially. Also, by inspecting **red lines**, PI takes more iterations than VI.

This is because, PI requires searching for a Policy Evaluation and Policy Improvement, and the two steps are repeated until policy stabilizes. Instead, VI repeats until the value function is optimal, then the policy found from it is the optimal policy. So, VI is supposed to be faster than PI.

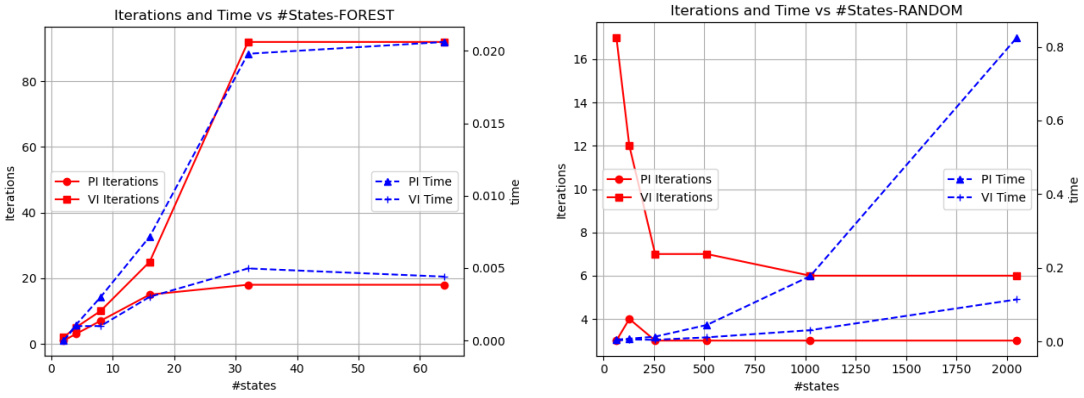

Figure 2: Runtime and Iteration Vs Number of States, for PI and VI

For FOREST, with small state (<=64) and action (2) size, the iterations and runtime increases exponentially with number of states until #States=32 and then it is stable.

For RANDOM, the iterations actually are maximum for #States=64 then decrease for PI and VI takes almost the same few iterations irrespective of the number of states. So the convergence is under question for RANDOM.

4.4. Error Reduction for PI and VI

From Hiive.MDPToolbox source code, the error is defined as the maximum absolute difference in the Bellman Function value in Eq(1) above for any state. There is a parameter to define the tolerance for this error to stop PI or VI, but to keep it simple it was kept constant at the default value.

`error = _np.absolute(next_v - policy_V).max()`	Eq(4)

Plotting this error reduction over iterations below in Fig(3), for FOREST and RANDOM below, we can again see **the slope of error plot for PI in solid lines is higher than for VI**. This is a confirmation of what is presented in 4.3 section above. Here the discount factor is 0.99.

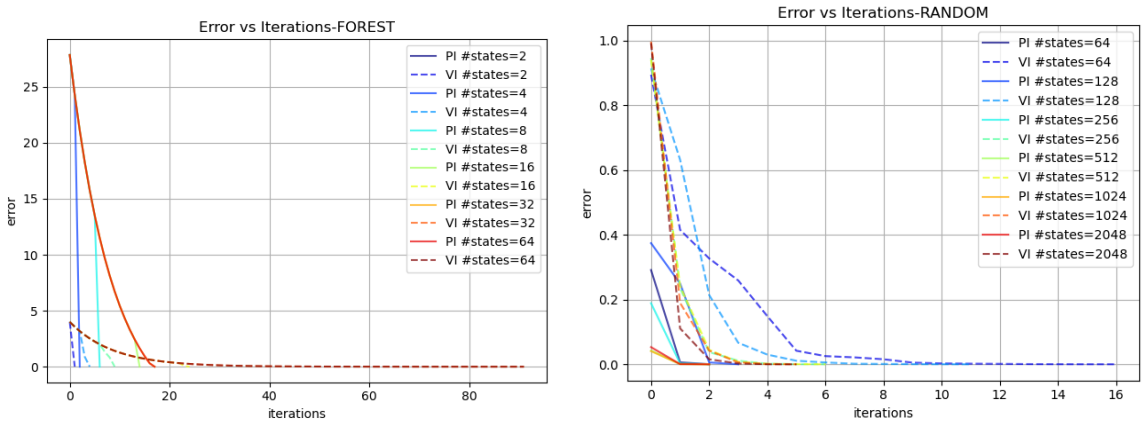

Figure 3: Error vs Iteration Over Number of States, for PI and VI

VI starts off with a large error for RANDOM and takes longer to converge, but we see the opposite for FOREST. So there is no specific relationship here. Interestingly, even for a smaller MDP size in terms of S and A in FOREST, PI was faster than VI until #States=64. For larger state space in RANDOM, and larger action size (|A|=16), VI initially was off as it assumed a random V(s) matrix unlike PI which assumes a policy and VI took longer to converge.

PI requires searching for a both Policy Evaluation and Policy Improvement, and the two steps are repeated until policy stabilizes. Instead, VI repeats until the value function is optimal, then the policy found from it is the optimal policy. So, VI is supposed to be faster than PI. This way, VI is faster to converge, which is evident in the Fig (3).

4.4. Impact of Discount Factor on Iterations and Time

It appears from Fig(4) below higher discount factor causes more runtime and iterations for both PI and VI. The iterations grow exponentially for FOREST until 0.9 and then it drops for PI. But for VI it keeps increasing. The time also increases with higher discount. This is likely because the Bellman Eq(1), fluctuates more if discount factor is higher. So it takes more iteration and time to converge for both PI and VI. However, it is empirical observation based on S=32,A=2 for FOREST and S=2048, A=16 for RANDOM. Other state/action combinations were not explored.

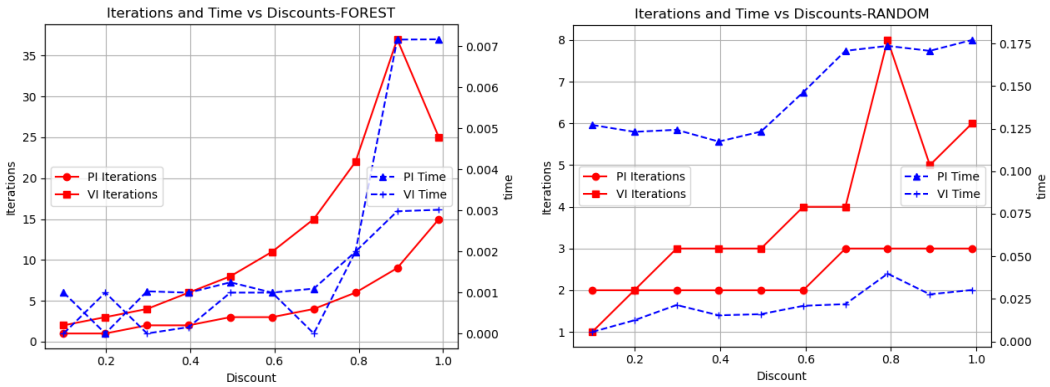

Fig (4): Error plot for PI and VI

4.5. Hyper parameter search for QL

For the model-free RL algorithm, Q-Learning was chosen. Since QL largely depends on the intial exploitation vs exploration strategy, the relevant parameter epsilon, along with the Learning Rate parameter, alpha, are employed in a hyper parameter search process. The hyper parameter process is a simple, manual, graphical process. The best set of (alpha and eps) is chosen based on which maximizes the average Bellman Eq (1) value. The rate at which the initial exploration probability eps is reduced (eps_decay) is kept constant at 0.99. The decay rate for epsilon is 0.99 and the final minimum epsilon is 0.1.

Here, initial eps=0.1 means only 10% exploration and in 90% case choosing the best action for the state from its Q(s,a) table. For eps=1.0 means a completely greedy random exploration approach which takes an action completely by chance and does not use its previous experience at all. Eps=0.55 is a balanced middle-ground option. However at each iteration, eps is reduced by 0.99 and it may drop until 0.1.

Alpha values used are {0.1, 0.5, 0.9} indicating low, medium, and high learning rate. For Eps, {0.1, 0.55, 1.0} values are used. The same values for decay rate (0.99) and minimum learning rate (0.1) are kept.

Since the problems do not have any terminating state, they are non-episodic. So instead of running for multiple episodes, every time, 30000 iterations were allowed. Since there is no termination and there is no convergence, QL runs for the given 30000 iterations.

The choices for the hyperparameter values are kept small to keep the computation process manageable. For the same reason, the number of states and actions are kept constant. For FOREST, S=32, A=2 is used. For RANDOM, S=2048, A=16 is used. For the two problems a constant discount factor=0.99 is used.

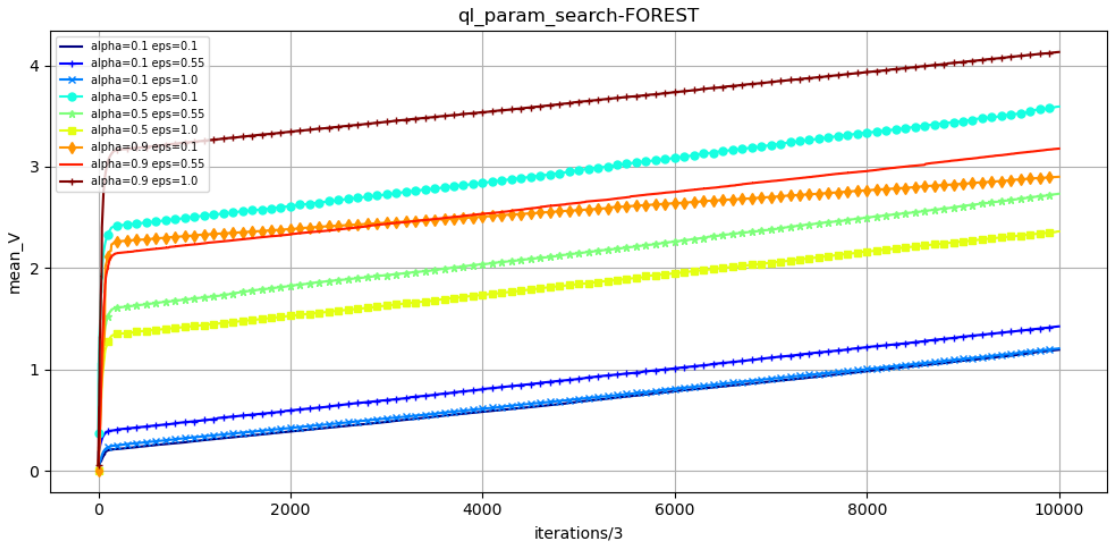

Fig (5): QL Hyper Param Search for FOREST

From Fig(5), it shows the alpha=0.9 and initial exploration eps=1.0 makes the maximum mean Bellman function value for the MDP. Apparently, in FOREST problem, each state can go to State=0 by taking action='Cut'. So the transition matrix is mostly non-zero. So to completely explore all the transitions, FOREST needs to explore all the scenarios by taking random actions initially. Then gradually it exploits its experiences from Q(s,a) matrix.

The QL charts show iteration/3 as X-axis as Hiive library only report 10000 values, i.e. for every 3 iterations for the 30000 iterations defined.

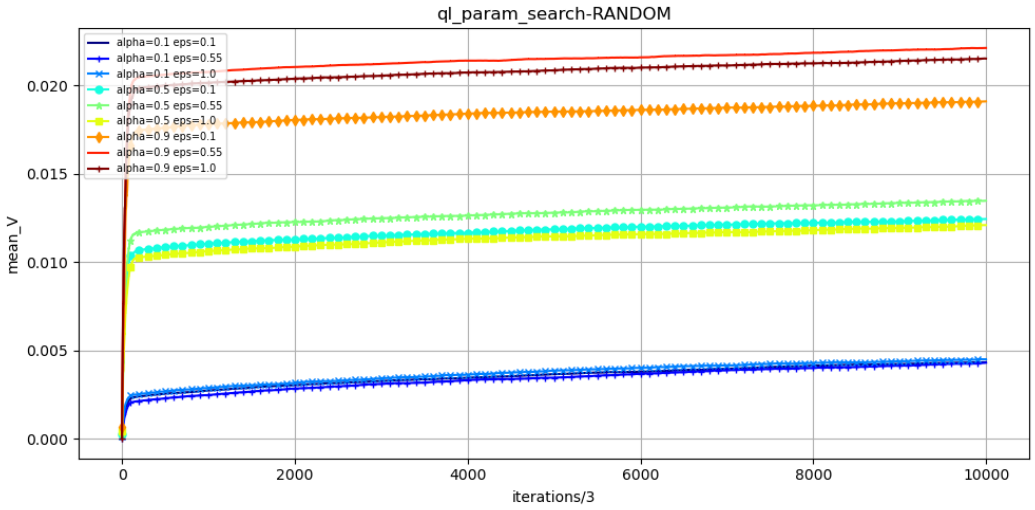

Fig (6): QL Hyper Param Search for RANDOM

On the other hand, RANDOM has the best mean V value with alpha=0.9 and eps=0.55. This problem is truly random, and as a result it makes sense to carefully take a balanced exploration vs exploitation tactic at each iteration by taking initial eps=0.55.

Notice, the higher the alpha values are, the mean V values are higher as well. This is expected because the Bellman Eq(1) is updated faster with higher alpha value.

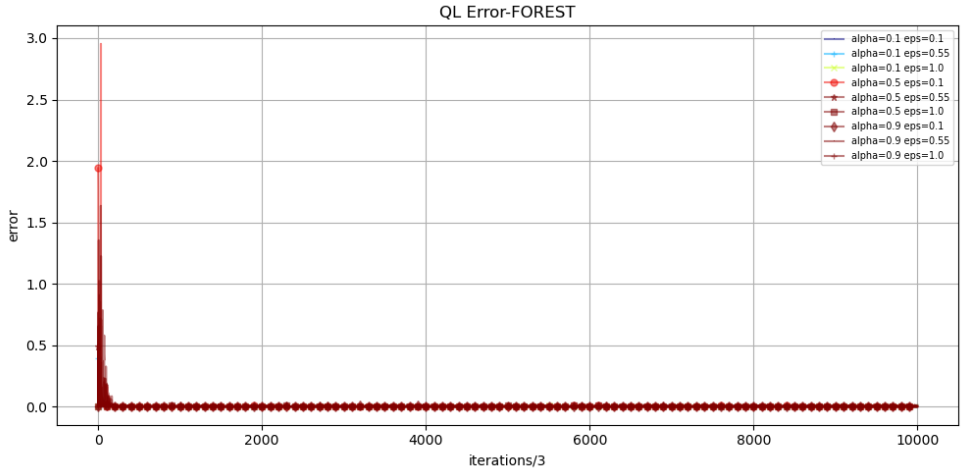

Fig (7): QL Error vs Iteration Search for FOREST

However Fig (7) shows the vast number of iterations have minimal impact on reducing the error defined in Eq(4) above but yet are still run anyway.

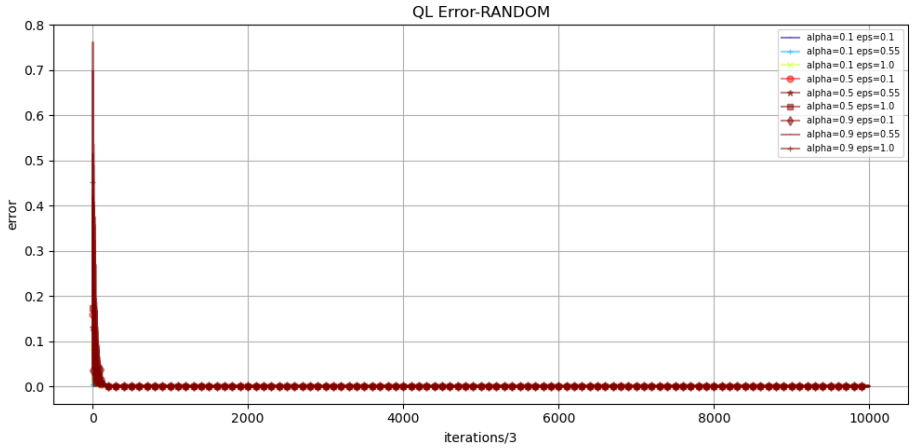

Fig (8): QL Error vs Iteration Search for RANDOM

The same observation is made for RANDOM as well. Less number of iterations than the set 30000 could have performed the same. So the number of iterations could could be a hyperparameter up for tuning as well.

4.6. Comparison of PI, VI, and QL

To compare the PI, VI, and QL, the states, actions, gamma are fixed so the same MDP problem is used to compare. . For FOREST, S=32, A=2 is used. For RANDOM, S=2048, A=16 is used. For the two problems a constant discount factor=0.99 is used for all the RL algorithms.

For QL, 30000 iterations were allowed for both MDPs. However, for FOREST, alpha=0.9, eps=1.0 is used. But for RANDOM, alpha=0.9, eps=0.55 is used.

Since the number of states are large and the problems do not have 2D representation, a metric is used to compare the found policies by PI, VI, QL. The percent of match in the policy vector is used as the comparison metric.

Table 1: % Match Between Policies

FOREST				RANDOM			
	PI	VI	QL		PI	VI	QL
PI	x	100	56.25	PI	x	11.5234	9.86328
VI	100	x	56.25	VI	11.5234	x	10.9375
QL	56.25	56.25	x	QL	9.86328	10.9375	x

In Table(1) above it shows, for FOREST, PI and VI found the same policy. However, QL's policy only matched 56% with PI and VI's policy. Likely, by having a more rigorous hyperparameter search process, a comparable QL policy could be found.

For RANDOM, the PI and VI policies match only 11%, whereas QL policy match less than 10% with its peers. In addition to the hyperparameter search issue, this random MDP has very large states, and may have needed more iterations of QL due to a complex transition structure. So more samples of <S,A,R,S'> pairs could have helped since QL assumes it does not know the true transition and reward matrix or function and only relies on its experience with the environment.

Based on that, Table(2) shows VI is the fastest for both MDPs though it takes more iterations than PI. By direct estimation of Bellman value function in Eq(1), VI achieves this over PI. QL is the slowest and takes many more iterations yet its mean Value function is less than that of PI or VI. Also PI does the best job finding the maximum expected reward. All these findings are expected from the discussion of these algorithms in section 2.

Table 2: Performance Comparison for PI, VI, QL

FOREST					RANDOM				
	Time	Iters	Mean V	Total Reward		Time	Iters	Mean V	Total Reward
Algorithm					Algorithm				
PI	0.007	15	82.055	101.030	PI	0.177	3	6.925	7.688
VI	0.003	25	21.895	40.871	VI	0.030	6	0.439	1.336
QL	2.375	30000	2.901	1.000	QL	14.220	30000	0.022	0.808

5. Conclusions

In this assignment, two non-episodic (continuous) tasks are modeled as MDP. To find the optimal policy for these MDPs, two model-based policy finding Reinforcement Learning (RL) algorithms: Policy Iteration (PI), Value Iteration (VI) are contrasted against the model-free Q-Learning algorithm which does not require the explicit models of the Transition and Reward matrix unlike the priors.

The lessons learned are:

- Modeling a MDP is a non-trivial task and takes domain-knowledge
- Higher discount factor provided higher reward, choosing a discount factor is important for modeling a MDP problem
- PI and VI are faster to find optimal policies when the MDP is known, i.e. Transition and reward functions are known
- QL can still find optimal policies even when the MDP is unknown by interacting with the MDP system and observing the reward it gets for taking an action at the current state and which state it ends up at
- VI is often faster than PI. QL is slow.

- QL hyperparameter optimization is challenging

Limitations:

- Robust hyper parameter optimization for QL could be employed if time permitted
- Adding a grid-world MDP along with FOREST, and RANDOM could make the comparison more insightful
- It would be interesting to apply Randomized Optimization Algorithms to find the best policies in place of PI and VI

6. References

6.1. Andreas Kirsch. Hiive, https://github.com/hiive/hiivemdptoolbox

6.2. MDP Toolbox. https://pymdptoolbox.readthedocs.io/en/latest/index.html

6.3. Some texts from this assignment here are used in my personal website for my note in www.asifrehan.com