Markov Decision Processes

1. Markov Decision Processes (MDP)

Markov decision processes in reinforcement learning can be defined as a model where an agent interacts with the environment, learning by taking actions from one state to another and through the resultant rewards. The problem ends when the agent reaches the terminal state and the ultimate goal is to maximize the rewards along the way.

The Markov Property:

MDPs follow the Markovian property which states that the current state captures/contains all the information/history from all the previous states and that current state's information is enough to make future decisions/actions.

MDPs can be deterministic or non-deterministic(stochastic). Deterministic MDPs mean that when an action a is taken from a state s, it always takes us to a new state s'. Stochastic means that when an action a is taken from a state s, there is a probability of ending in a new state s1' and a chance of ending up in a different new state s2' and so on.

Transition probability matrix: (T(s,a,s') or P(s'|s,a))

Given a set of states S and action A, it gives the probability of ending up in another new state taking an action.

Taking action a	Ending in S1	Ending in S2	Ending in S3
From state S1			
From state S2			
From state S3			

Action: Set of actions that can be executed on a state.

Reward: It is the immediate reward received by the agent by transitioning to a state s taking an action a.

1.1. Frozen Lake MDP

This is a frozen lake, where the goal of the agent is to navigate from the start state to the goal where the frisbee is located through the frozen parts of the lake and avoiding the holes.

There are 4 four different definition of states,

S: starting state, safe

F: frozen surface, safe

H: hole / terminal state

G: goal / terminal state

Rewards:

a step on frozen surface = -0.01

On the hole = -1

Goal = 1

I have designed this a grid-world problem with a large number of states as 400 (20x20). This problem has different kinds of safe pathways which range from easy to difficulty levels to navigate through and it will be interesting to see the reinforcement learning algorithms perform on this MDP. Its simplicity in terms of rewards will help us follow the problem easily. I have used the Irl python package's implementation of the problem.

1.2. Forest Management MDP

The forest must be managed by taking actions either to wait or cut the trees, keeping in mind whether to keep the trees for the wildlife or make money by cutting the trees.

I have designed this as a non-grid world problem with a small number of states of 5.

This problem is defined as an MDP as,

States: The different states are given by the age of the forests. The youngest state is 0.

Transition probability matrix: There is a probability (p=0.1) of a wild fire destroying the forest and returning to its youngest state 0. The matrix is given by taking actions wait/cut the probability of ending in new state.

Action: wait	S0	S1	S2	S3	S4
S0	0.1	0.9	0	0	0
S1	0.1	0	0.9	0	0
S2	0.1	0	0	0.9	0
S3	0.1	0	0	0	0.9
S4	0.1	0	0	0	0.9

Action: cut	S0	S1	S2	S3	S4
S0	1	0	0	0	0
S1	1	0	0	0	0
S2	1	0	0	0	0
S3	1	0	0	0	0
S4	1	0	0	0	0

For example, from S0, if we take action to wait, there is a 0.1 probability that wild fire burns and takes us to the youngest state s0 and 0.9 probability that we move to s1 i.e., the forest ages. Contrarily, if we choose to cut the trees, there is a 100% probability that we will return to the youngest state s0 from all states.

Rewards: Rewards are given for taking actions across each state of the forest. We get the maximum reward of 4 to keep the forest until the oldest state.

Action	Wait	Cut
S0	0	0
S1	0	1
S2	0	1
S3	0	1
S4	4	2

This problem is interesting because by defining a non-grid world problem small, we can easily understand the complexities involved and the learnings of the algorithms on these kinds of real-world problems.

2. Algorithms

2.1. Value Iteration

The value iteration algorithm involves two steps – **finding optimal value function** and **policy extraction.** We start with a initial value for all states, then finding value for each state through iterations using the bellman equation and converging to the optimal value function. From this, we can get the optimal policy to be followed at each state. The value of a state is the immediate reward received and the discounted future rewards that will be received taking actions from that state onward.

2.2. Policy Iteration

Policy iteration algorithm involves two steps – **policy evaluation** and **policy improvement**. We start with initial random policy and evaluate the policy by calculating the utilities for states when it follows that policy. Then we improve the policy until it converges to the optimal policy.

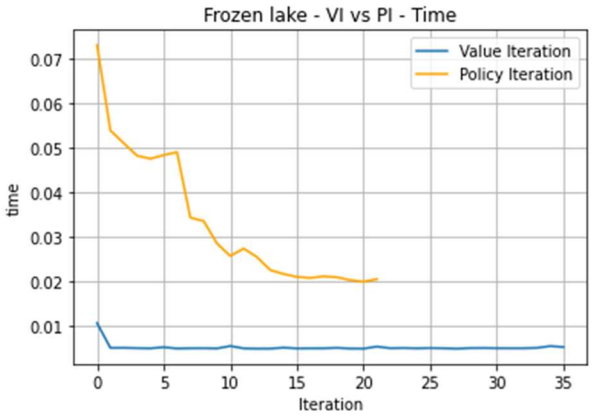
2.3. Q Learning

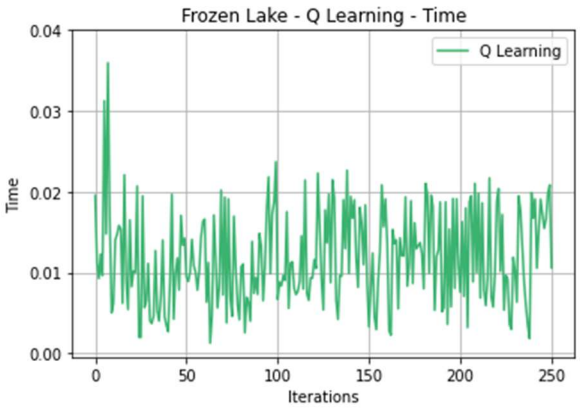
I have chosen this as another RL algorithm. It is a model-free reinforcement learning algorithm. It finds the optimal policy which maximizes the expected reward starting from a current state factoring in all the future possible rewards too. We construct a Q table which consists of the expected rewards for each state action pair, from which the optimal policy can be extracted as the action which gives the maximum reward for that state. The Q-table is updated using a discount factor which weighs the immediate rewards higher than the future ones.

3. Solving Frozen Lake MDP

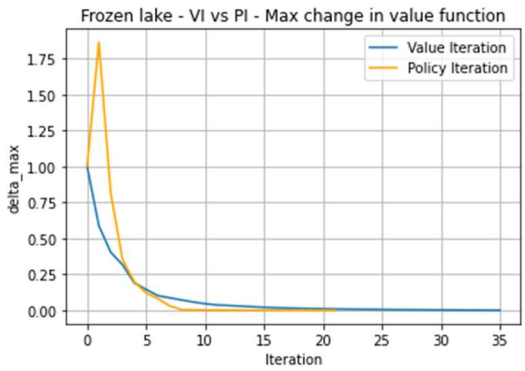
I have solved the frozen lake problem using value iteration, policy iteration and Q-Learning algorithms and plotted various result metrics.

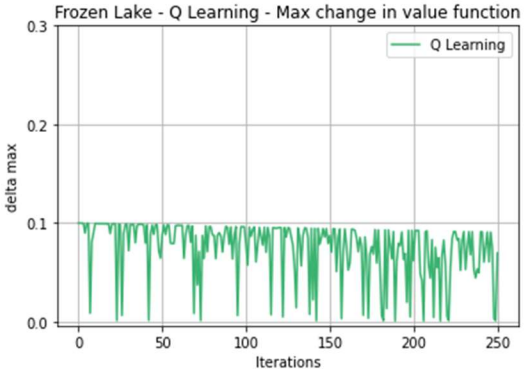
Time: From the graph we can see the time taken by the algorithms for each iteration until convergence is plotted. Value iteration takes a consistent time across all iterations, this is because we are calculating different values for states over each iteration. But policy iteration starts with higher time, which might be because of the initial policy it started out with. Q-learning also does not vary much since we have to note that the scale is very small.





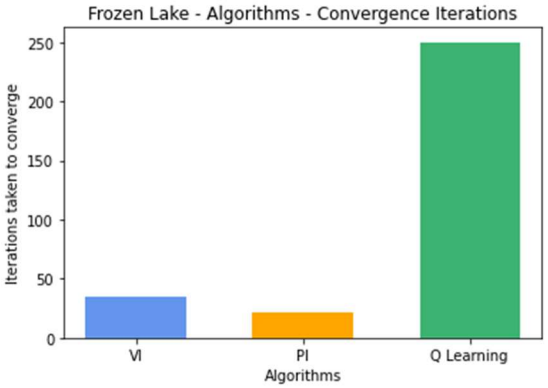
Change in value function: For value iteration, it initially sees high changes in the value function, but as the neighbouring states values keeps propagating over iterations, the change becomes constant after 10 iterations. Policy iteration also sees huge changes at the start and then as it converges to the near optimal policies around 8 iterations the change is constant. Q-learning follows an oscillation but its negligible on a smaller scale.

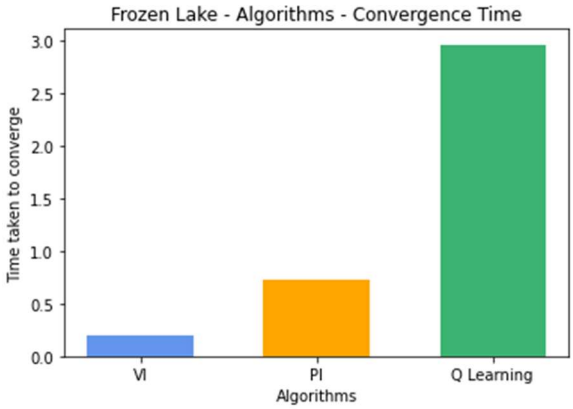




Convergence: In terms of iterations, Policy iteration performs best converging in just 21 iterations, with value iteration coming next at 35 iterations. Q-learning takes around 250 which is huge number of iterations and time as well.

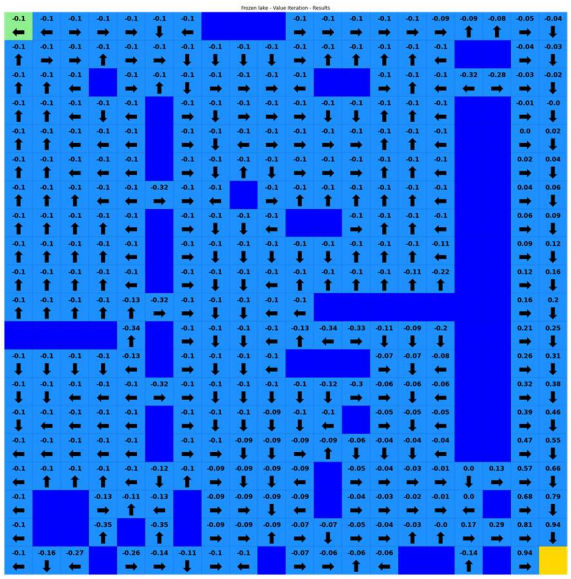
Policy iteration takes more time to converge than value iteration, since some of the initial policies might have involved longer routes ending in the same states repeatedly.



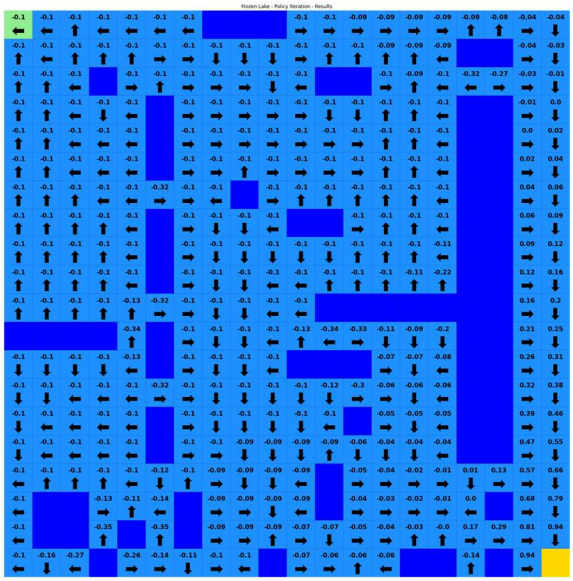


We can find the results of the optimal values/policies solved using the algorithms. It is interesting to see along the holes, where the policies are mostly leading the agent away from the surrounding the holes which might result in a reward of -1 which is very high compared to the long route negative step rewards of -0.01

Solved using Value Iteration

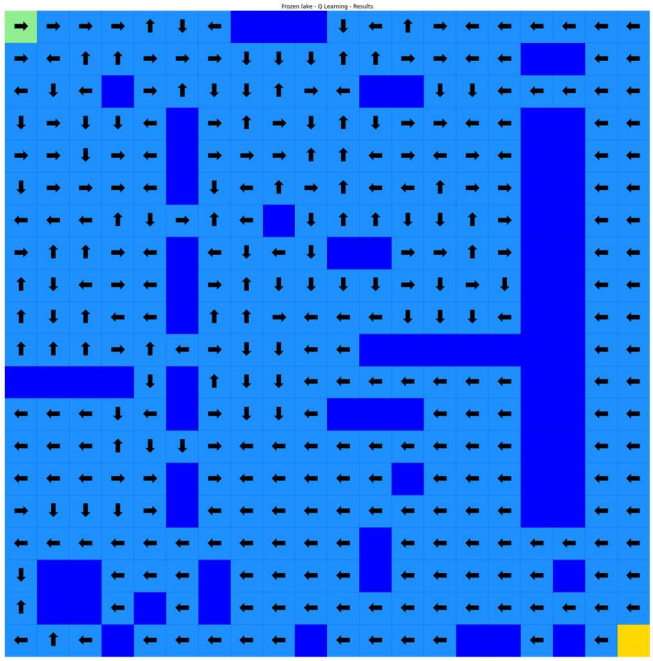


Solved using Policy Iteration



It is interesting to note for Q-learning how the last two columns lead toward the holes. The discounted future rewards when high, chooses to take the reward of -0.01 or -1 over the discounted +1.

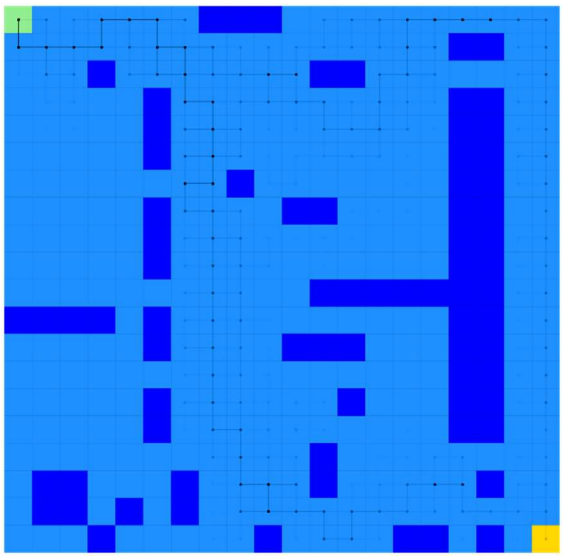
Solved using Q-Learning



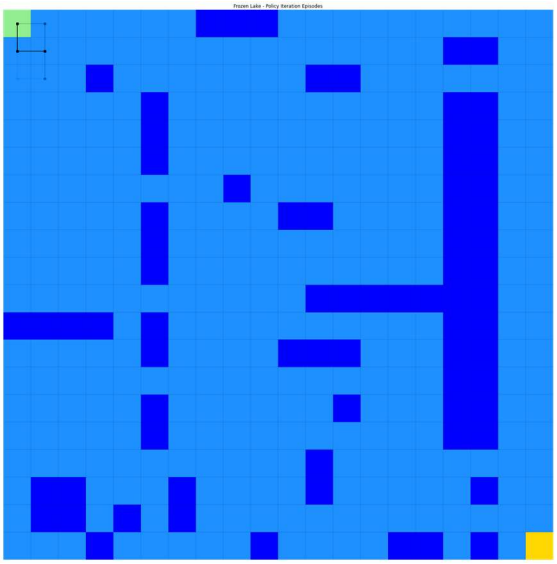
Scoring the optimal policy: After finding the optimal policy through convergence, I have evaluated the performance of the algorithms using their optimal policy. I have plotted various episodes.

We can see that value iteration performs the best, with many episodes leading to the goal state. PI keeps circulating within the initial states over. Q learning navigates halfway through but never reaches the goal state.

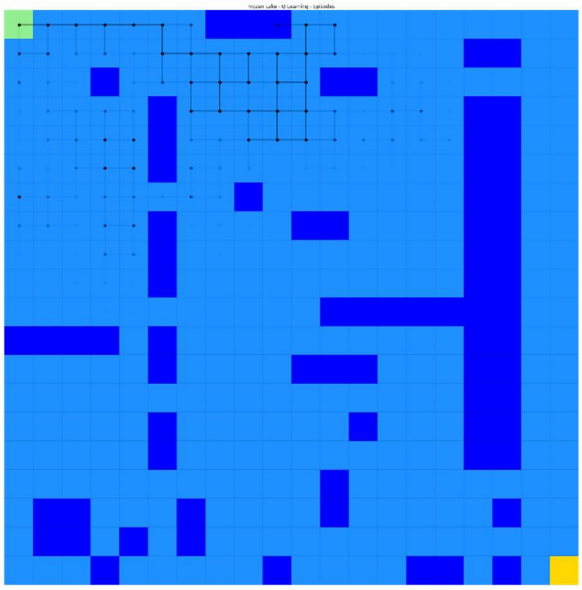
Scoring policy – Value Iteration



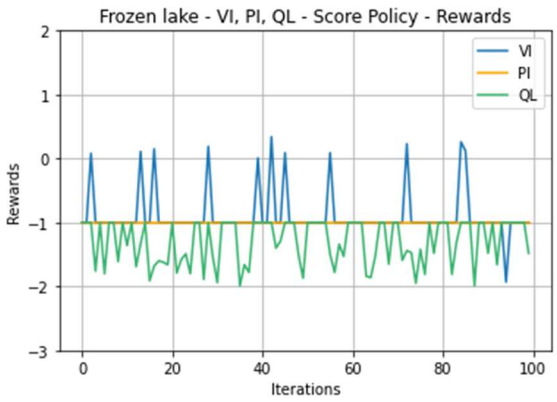
Scoring policy – Policy Iteration

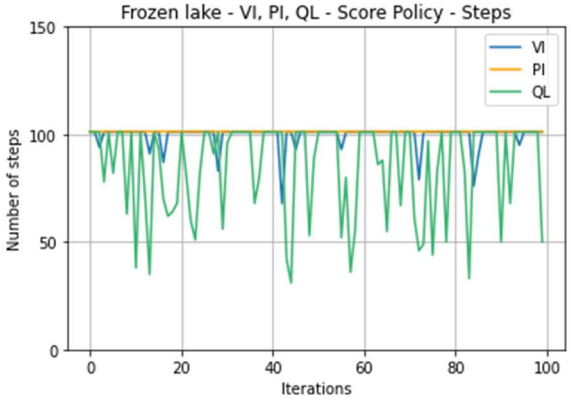


Scoring policy – Q Learning



I have plotted the rewards and number of steps taken scoring the optimal policy over 100 episodes/iterations. VI yields better rewards for various episodes(the ones which reach the terminal state). Q-learning yields poor rewards across all walks.





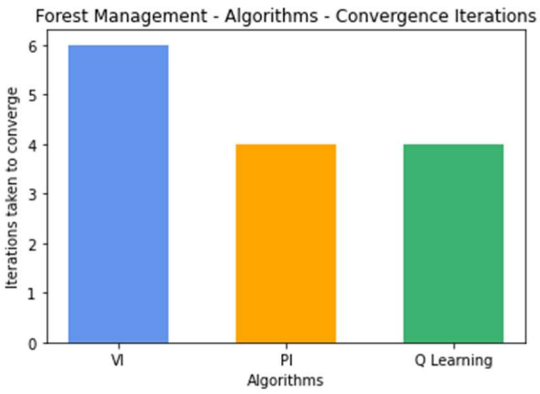
We can conclude that for the frozen lake problem, value iteration performed best in terms of converging to an optimal policy yielding better rewards in less time and iterations. Q-learning performed the worst of all.

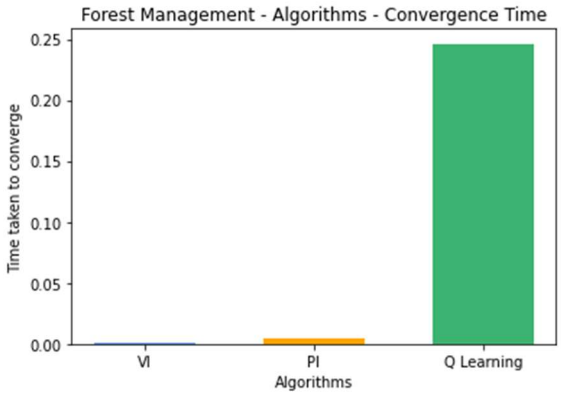
4. Solving Forest Management MDP

I have used the pymdptoolbox python package's implementation of the forest management problem. It is interesting to note that the agent has to wait until the last state (oldest) to yield the maximum reward or it can cut the trees in the intermediate states to yield a small reward and start out at the youngest state s0 and repeat that over and over again.

With higher discount rates it would minimize that future huge reward 4 and make the policies to focus on the immediate rewards. Lower discount rates would focus more on waiting till the end to reap that future reward. But we should also take in the probability that there might be a wildfire and end clearing the forest without any reward.

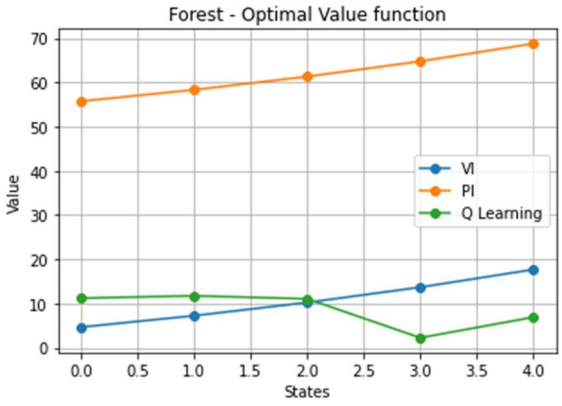
Convergence: Since the problem size is small, we can see that VI takes 6 iterations to converge, with PI and Q Learning with 4.





Optimal Value function:

The values are higher in range for PI. For both PI and VI, the values constantly increase over each state, which depicts that as the forest ages, the expected reward would increase since we get the highest reward of 4 for the oldest forest kept for the wild life.



Q table:

State x Action	Action: Wait	Action: Cut
S0	11.20289475	10.31996863
S1	9.69542194	11.73641347
S2	0.27543007	11.05502928
S3	2.24828728	0
S4	0.38354772	6.90618878

Optimal Policy:

All the optimal actions from VI and PI is wait. But we know already that wait yields the maximum reward of the forest aging of about 4.

Algorithms	S0	S1	S2	S3	S4
VI	Wait	Wait	Wait	Wait	Wait
PI	Wait	Wait	Wait	Wait	Wait
QL	Wait	Cut	Cut	Wait	wait

5. Citations

Ashraf, Mohammad. “Reinforcement Learning Demystified: Markov Decision Processes (Part 1).” *Medium*, Towards Data Science, 11 Apr. 2018, towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690.

Wong, Ryan. “Getting Started with Markov Decision Processes: Reinforcement Learning.” *Medium*, Towards Data Science, 3 Feb. 2019, towardsdatascience.com/getting-started-with-markov-decision-processes-reinforcement-learning-ada7b4572ffb.

“Example Case Using RewardingFrozenLake¶.” *Example Case Using RewardingFrozenLake - Lrl 1.0.0 Documentation*, lrl.readthedocs.io/en/latest/example_solution_frozen_lake.html.

“Markov Decision Process (MDP) Toolbox: Example Module¶.” *Markov Decision Process (MDP) Toolbox: Example Module - Python Markov Decision Process Toolbox 4.0-b4 Documentation*, pymdptoolbox.readthedocs.io/en/latest/api/example.html#mdptoolbox.example.forest.