Reinforcement Learning Experiments

Rui Hu
Rhu63@gatech.edu
CS 7641 Machine Learning Assignment 4

1. Abstract

The main objective of this assignment is to use reinforcement learning methods to solve MDPs and compare the influence of different hyperparameters along the way. Our approaches include Policy Iteration, Value Iteration, and Q Learning, which are common ways to solve MDP problems.

2. The Environments

The two problems used in this assignment are Frozen Lake environment from OpenAI gym and Forest Management from MDPToolBox.

2.1 Frozen Lake

Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake. The table below summarized the states, actions, and rewards of this problem. [1]

States	Actions	Reward
S Starting Position	0 Left	Reach goal(G): +1
G Goal	1 Down	Reach hole(H): 0
F Frozen Surface	2 Right	Reach frozen(F): 0
H Hole in the Ice	3 Up	

Although simple at first glance, this problem is useful and interesting because many real-world problems can be abstracted or simplified into a similar fashion. For example, when a robot wants to explore a territory such as Mars. Besides, its stochasticity also resembles the scenarios when robots face unpredicted problems.

2.2 Forest Management

The problem is stated as follows: a forest is managed by two actions: Wait and Cut. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest. [2]
{0, 1 . . . S-1} are the states of the forest, with S-1 being the oldest. Let Wait is action 0 and Cut is action 1. After a fire, the forest is in the youngest state, that is state 0. The reward = 4 when the forest is in its oldest state and action Wait is performed. The reward =2 when the forest is in its oldest state and action Cut is performed. This problem is useful and interesting because it mimics the business environments and stock markets where people need to make decisions at each time step to maximize overall gains. Also, profits need to be realized, otherwise they may be lost.

3. The algorithms

3.1 Policy Iteration and Value Iteration

Both policy iteration and value iteration are methods to solve MDPs. In policy iteration, we start by choosing an arbitrary policy π. Then, we iteratively evaluate and improve the policy until convergence. We evaluate a policy π(s) by calculating the state value function:

$$V(s) = \sum_{s',r'} p(s', r|s, \pi(s))[r + \gamma V(s')]$$

Then, we calculate the improved policy by using one-step look-ahead to replace the initial policy π(s):

$$\pi(s) = arg\max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$$

Here, r is the reward generated by taking the action a, γ is a discount factor for future rewards and p is the transition probability.
In value iteration, we compute the optimal state value function by iteratively updating the estimate V(s). We start with a random value function V(s). At each step, we update it:

$$V(s) = \max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$$

The update step is very similar to the update step in the policy iteration algorithm. The only difference is that we take the maximum over all possible actions in the value iteration algorithm.

3.2 Q-learning

The goal of Q-learning is to learn a policy that tries to maximize the discounted, cumulative reward:$R_t = \sum_{t=0}^{n-1} \gamma^t r_{r+1}$, where γ is the discount factor, R_t is the reward at time t. The main idea is to find a function Q*: State × Action -> R, that could tell us the return given action and state. Then we could construct a policy that maximizes our rewards: $\pi^*(s) = argmax_a Q*(s, a)$
The way to find Q function is through Temporal Difference Learning:
$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$, (St, at, Rt+1, St+1, at+1) is a transition, and α is the learning rate.

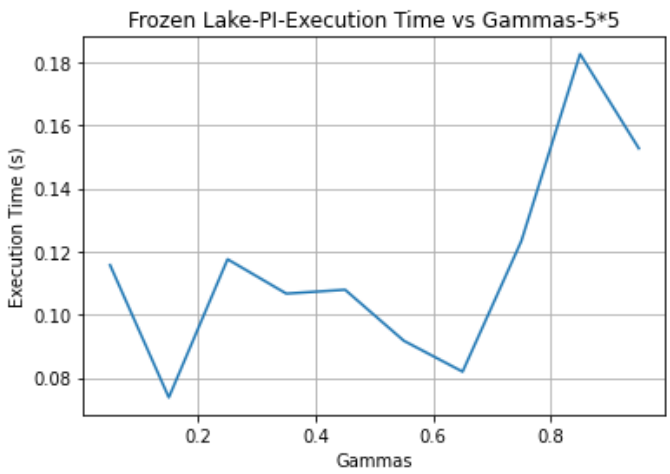
4. The experiments

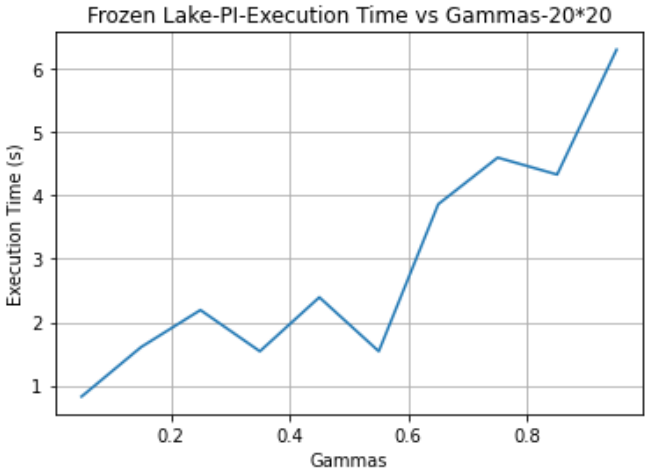
4.1 Frozen Lake

For frozen lake problem, the small size is 5*5, and large size is 20*20, which has 400 states. The figures for small problems are shown in left, and those for large problems are shown in right. Different gammas (discount factors) are tested in the problem, which range from 0.05 to 0.95.

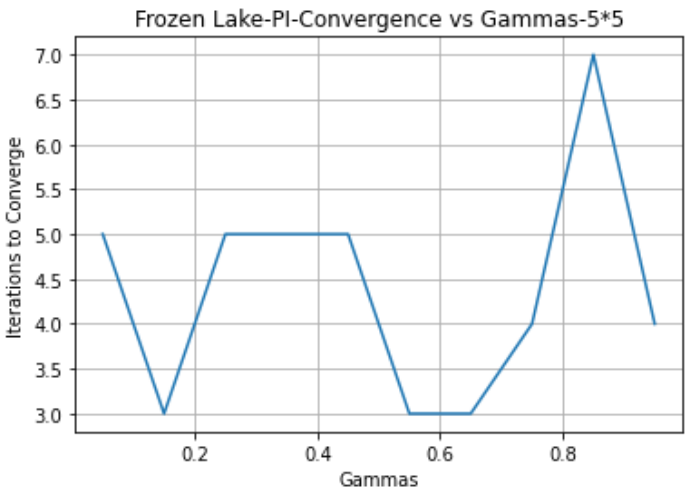
a. Policy Iteration

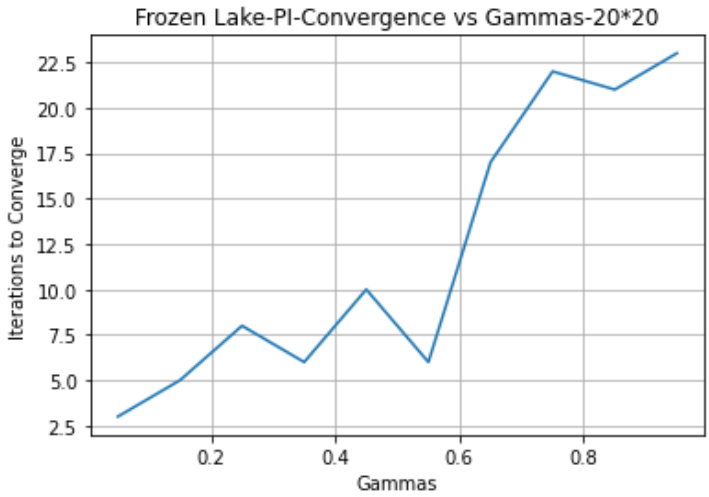
Execution time is plotted vs gamma in the figures below. It generally takes more iterations to converge with higher gammas, and hence the running time is longer.



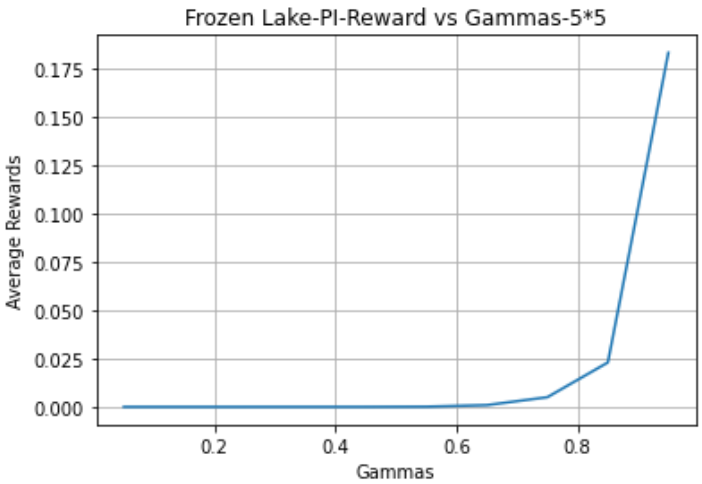


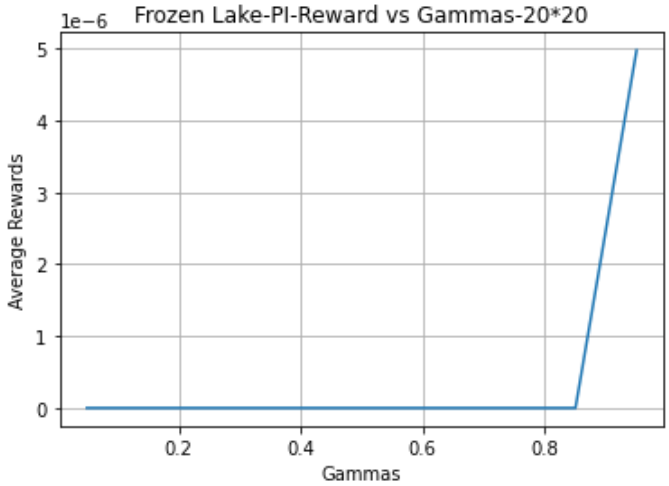
When problems becomes larger (moving from 5*5 to 20*20), policy iteration takes many more iterations to converge. Also, If we compare the pictures to those of Value Iteration's, we can find that Policy iterations take fewer iterations to converge than value iterations. This is because policy iteration stops when policy stops changing but value iteration will keep running until a certain threshold is met.

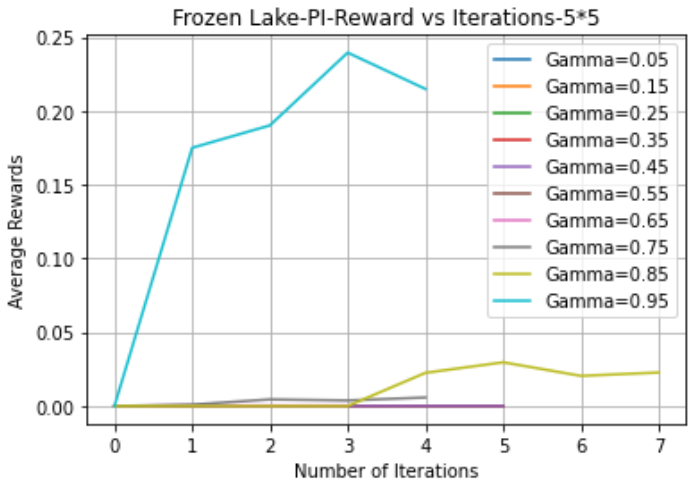


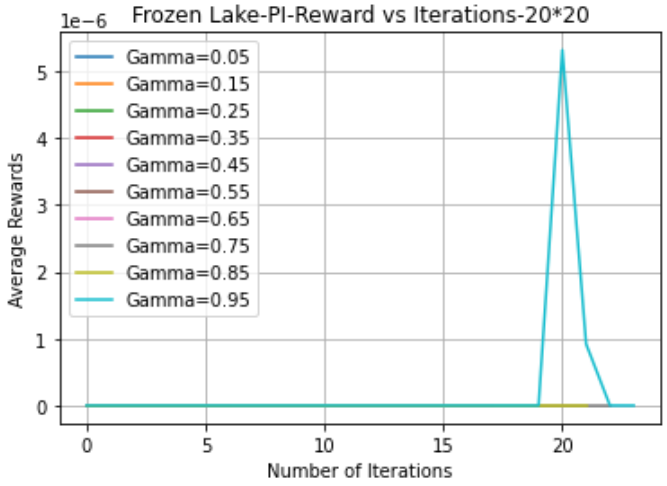


Average rewards vs different gammas and average rewards vs different number of iterations for the 2 problems are plotted in the four figures below. Higher gamma generally takes more iterations to converge. Also, higher gammas will mathematically bring higher rewards because rewards are discounted less.

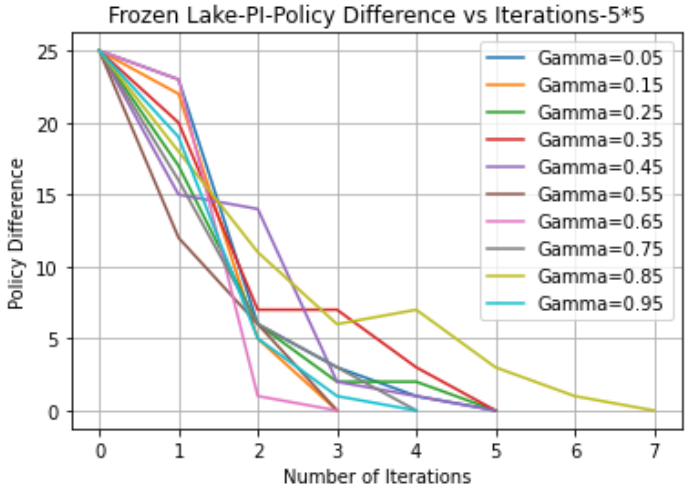


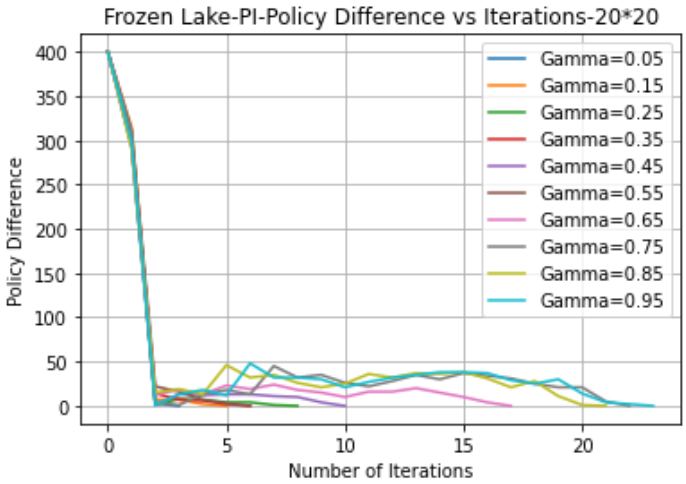






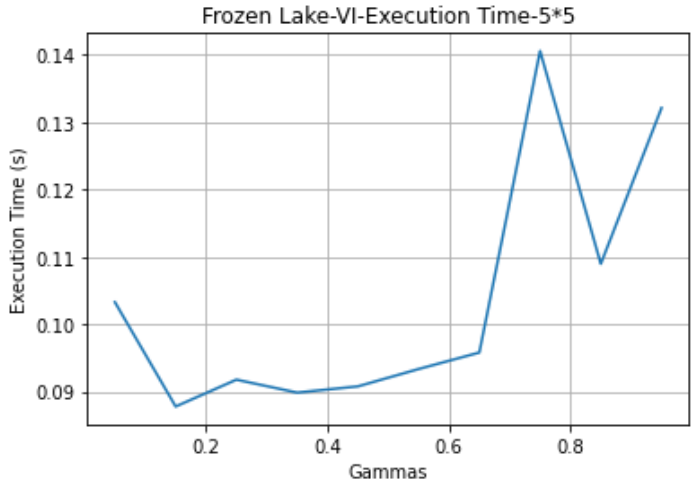
The difference between policies for different gamma values are shown below. We can see that the policy stops changing in all scenarios. Besides, higher gammas require more iterations to converge.

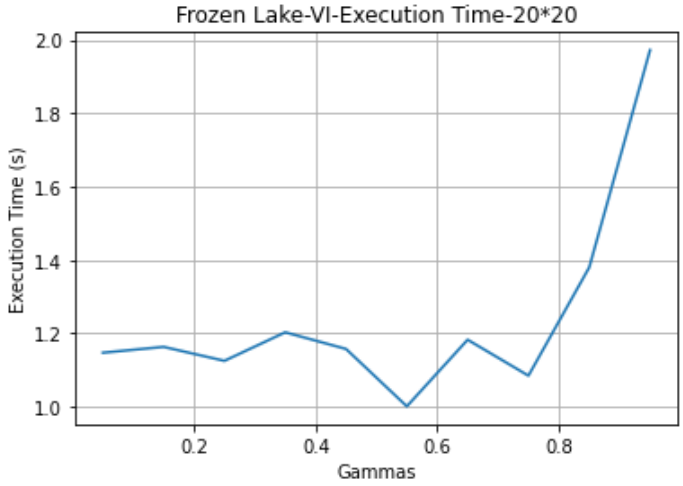




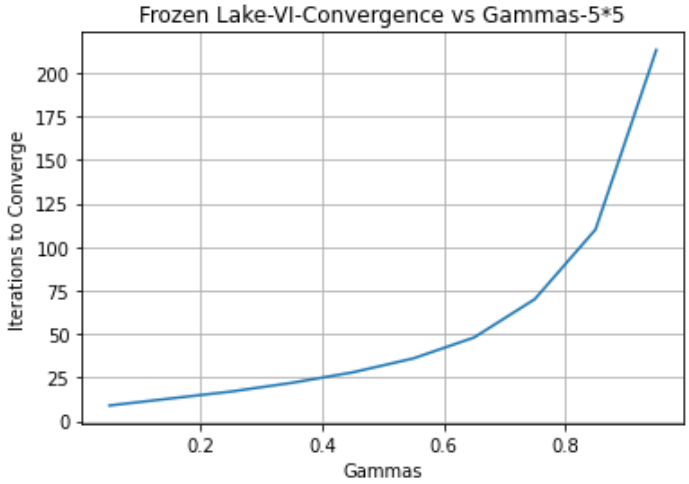
b. Value Iteration

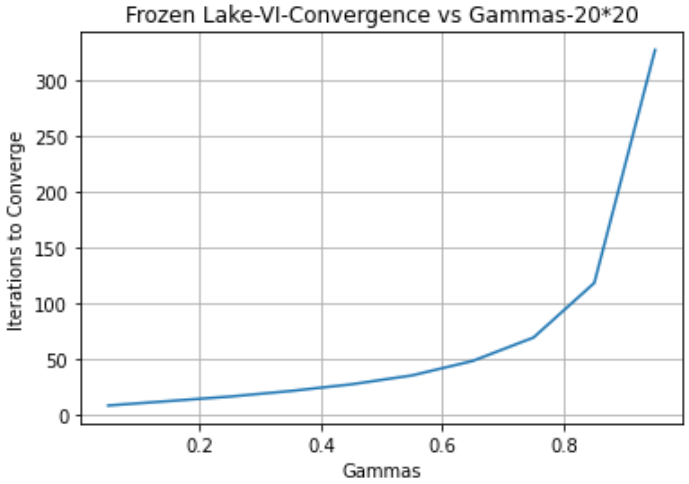
Execution time is plotted vs gamma in the figures below. It generally takes more iterations to converge with higher gammas, and hence the running time is longer.



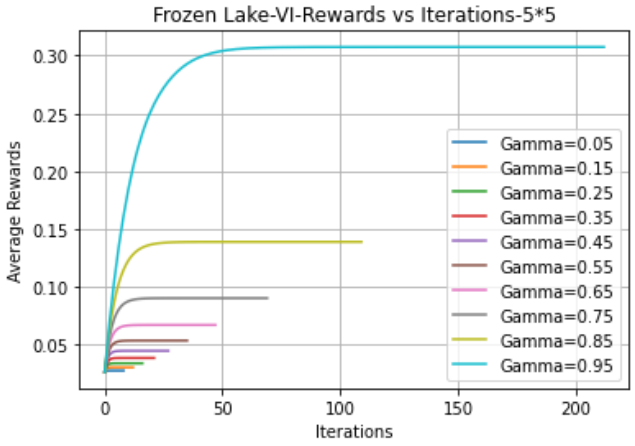


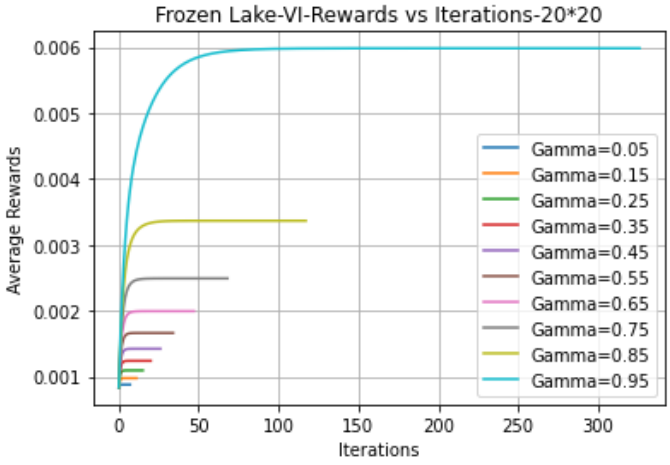
If we compare the figures below to policy iteration, we can see that value iterations take more iterations to converge than policy iterations. This is because policy iteration stops when policy stops changing but value iteration will keep running until a certain threshold is met.



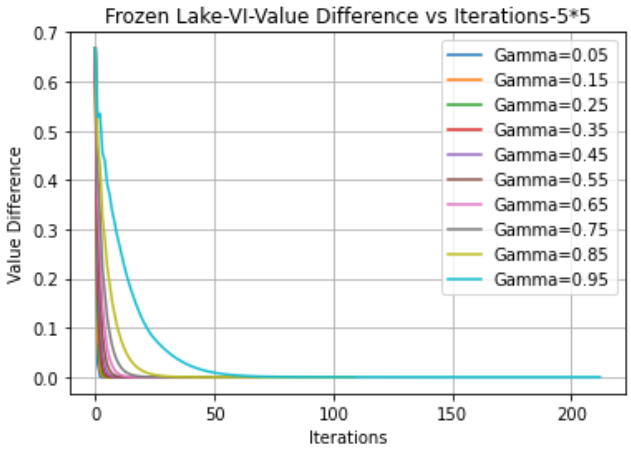


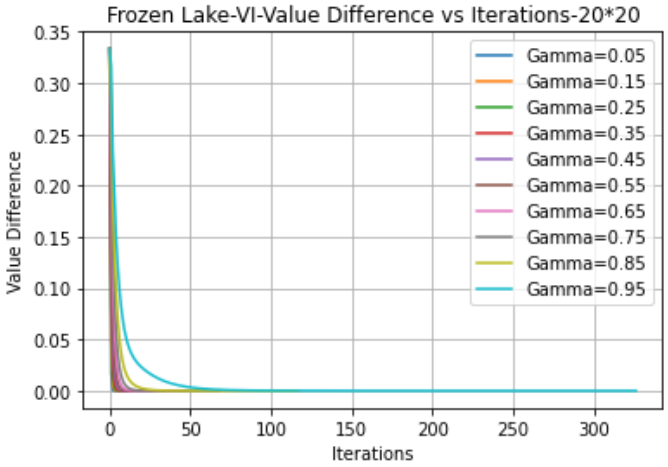
Average rewards vs different number of iterations for the 2 problems are plotted in the figures below. We can see that higher gamma requires more iterations to converge. The stopping criteria is that the change in value function < 1e-10. Even for gamma = 0.95, the algorithm already achieved maximum rewards after 50 iterations, but still kept running because stopping criteria hasn't been met. This makes value iteration less popular than policy iteration. Also, higher gammas will mathematically bring higher rewards because rewards are discounted less.



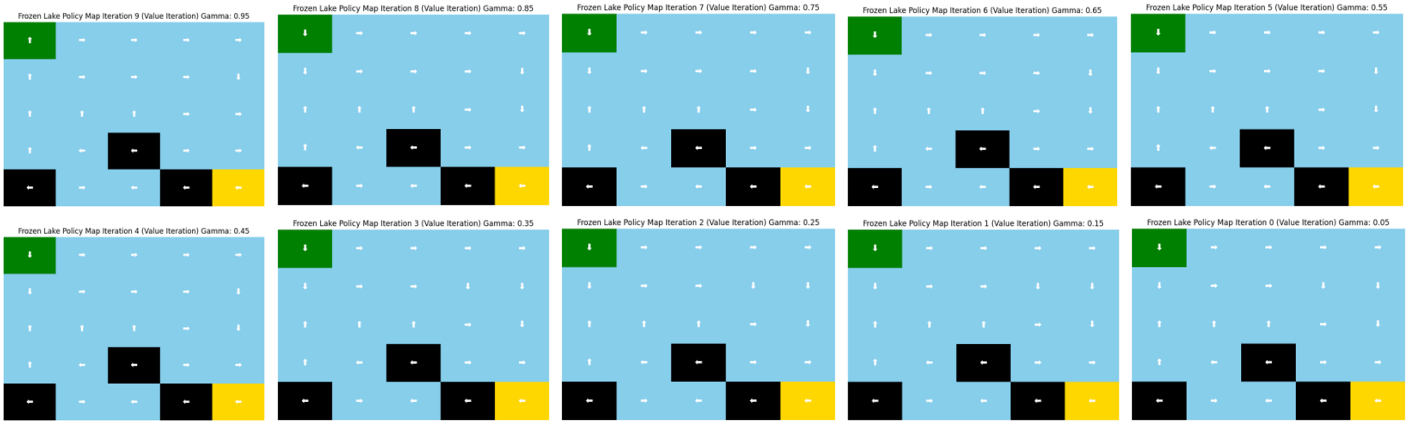


The change in the value function for different gammas are shown in the figures below. Eventually all scenarios converge. However, when problem size is larger and gamma is higher, value iteration requires more iterations to converge.



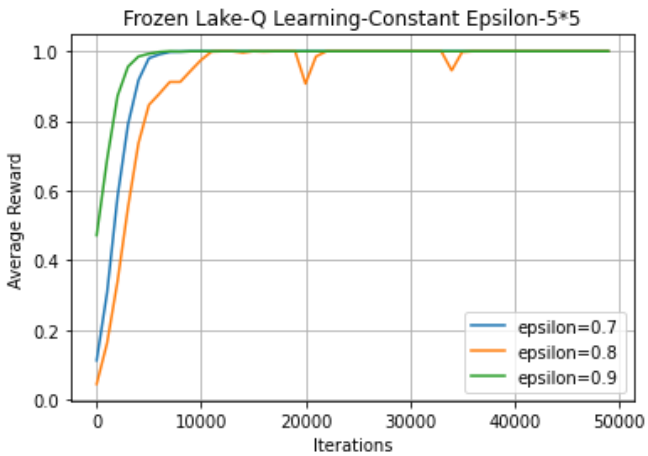


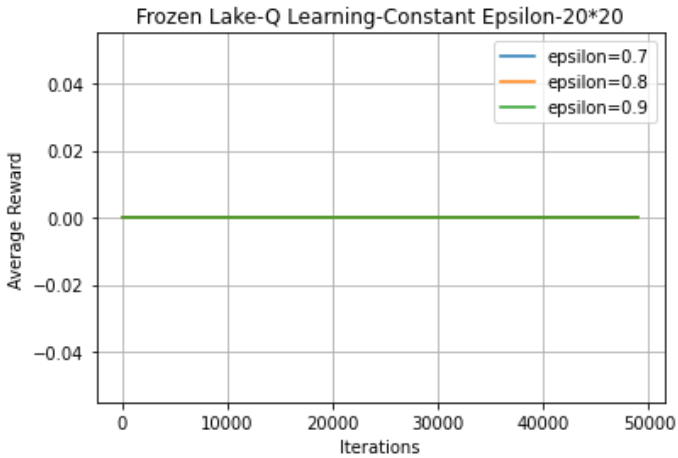
The final policies with different gammas are shown in the pictures below. Green square is the starting point, gold is goal, blue squares are ice, and black squares are holes.



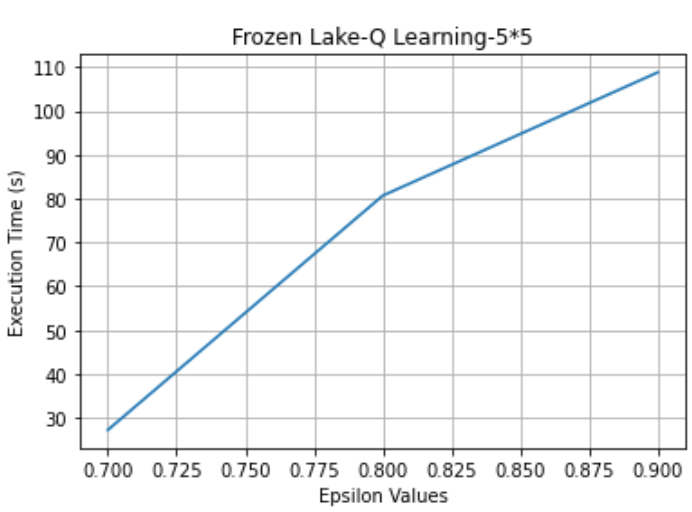
c. Q Learning

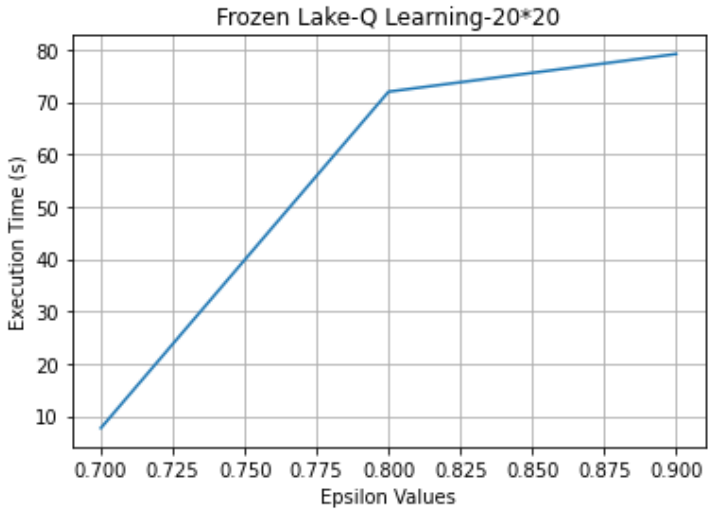
Epsilon is the exploration factor. With probability p<epsilon, the agent will take the best action, otherwise it will randomly select an action to explore the environment. For the small problem, epsilon = 0.9 helps the agent to get rewards the fastest. All three epsilons eventually converge to stable rewards. However, the frozen lake problem becomes much more complex when its size increase (because it has holes and agents will get stuck). Therefore, for problem size 20 * 20, q-learning still didn't cumulate any reward after 50000 iterations.





The figures below show execution time vs different epsilon values for both problems. Higher epsilon means the agent will randomly explore the environment more. Hence, the execution time will be longer.



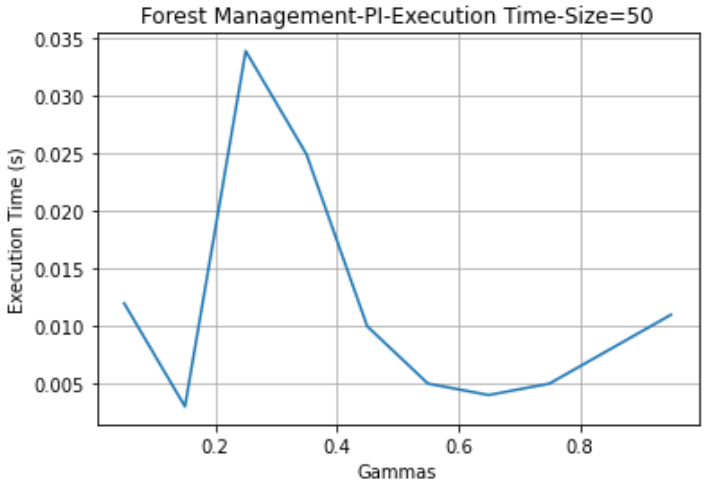


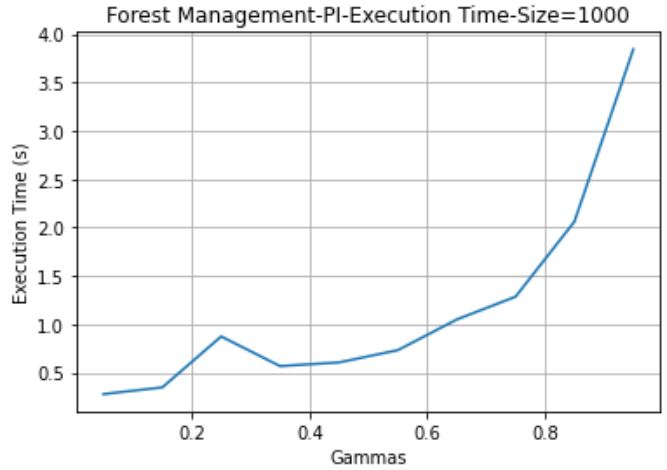
4.2 Forest Management

For Forest management problem, the small size is 50 states, and large size is 1000 states. The figures for small problems are shown in left, and those for large problems are shown in right. Probability of fire (p) equals 0.1. Different gammas (discount factors) are tested, which range from 0.05 to 0.95.

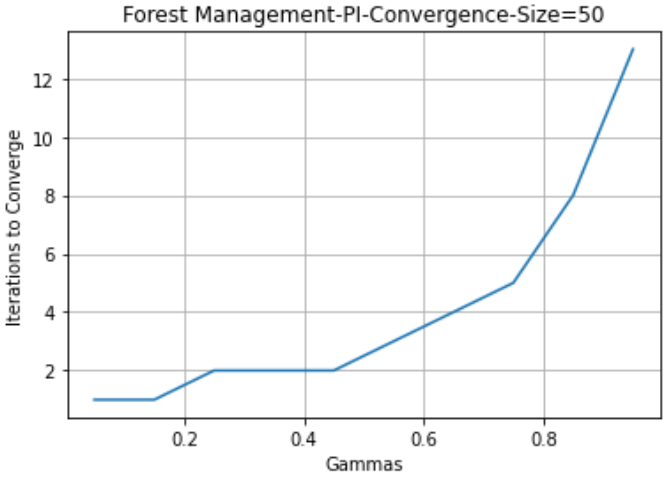
a. Policy Iteration

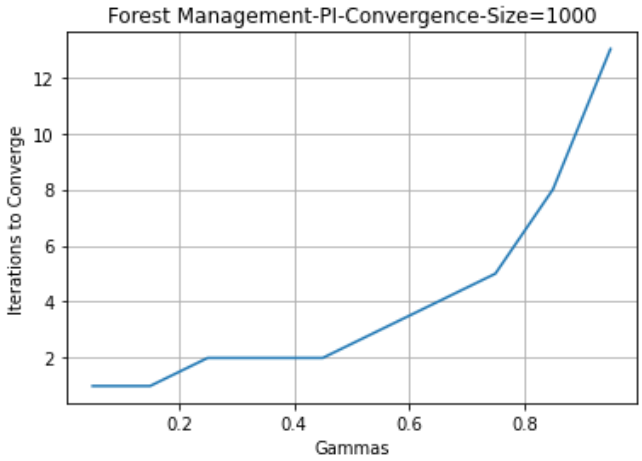
The two figures below show execution time. When problem size becomes larger (1000 vs 50), running time increases accordingly.



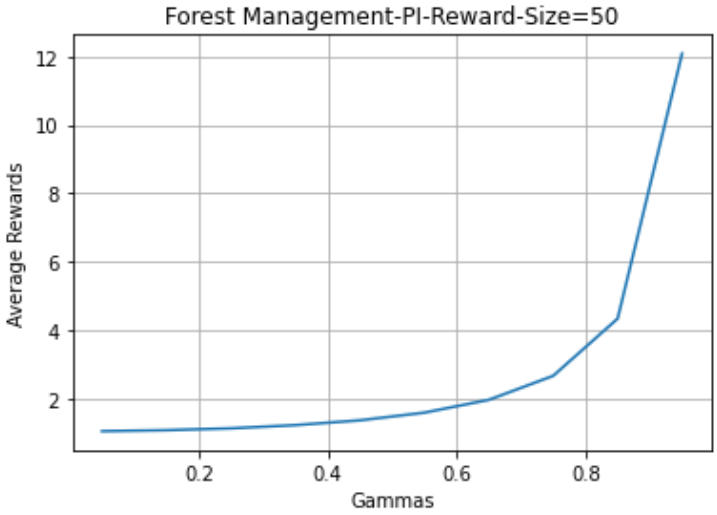


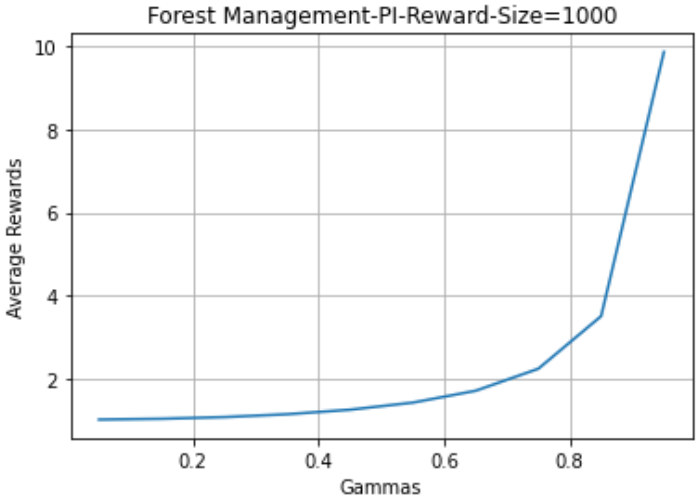
Again, we observe that when gamma increases, the iterations to converge increases. For this problem, the number of iterations to converge highly depends on its burning probability p and reward structure, so we can see that the number of iterations in these two scenarios didn't change much.





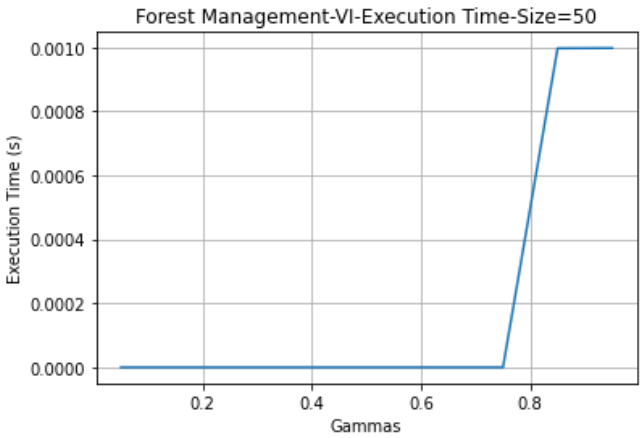
Higher values of discount factor will make the rewards discounted less. Therefore, we can find that when gamma is higher, average reward is higher.

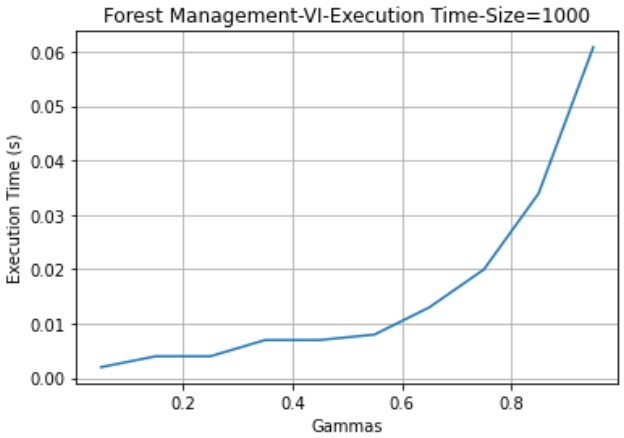


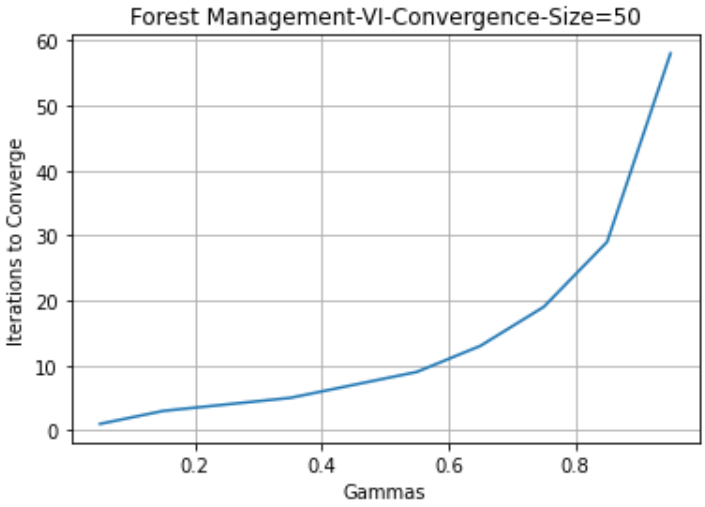


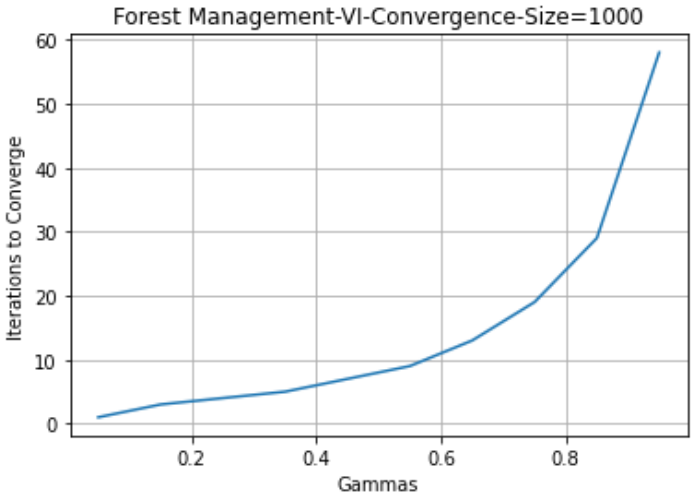
b. Value Iteration

The four charts below show for small and large problems, the execution time and iterations to converge vs different gamma values. When discount factor is higher, the algorithm generally takes more iterations to run and more time to converge.

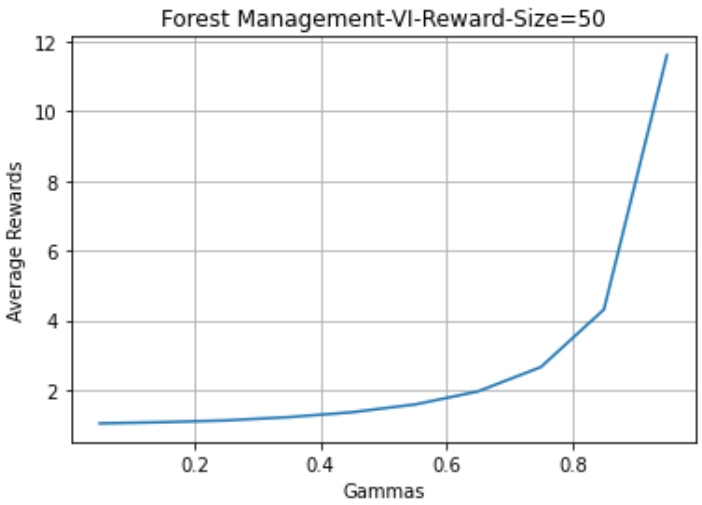


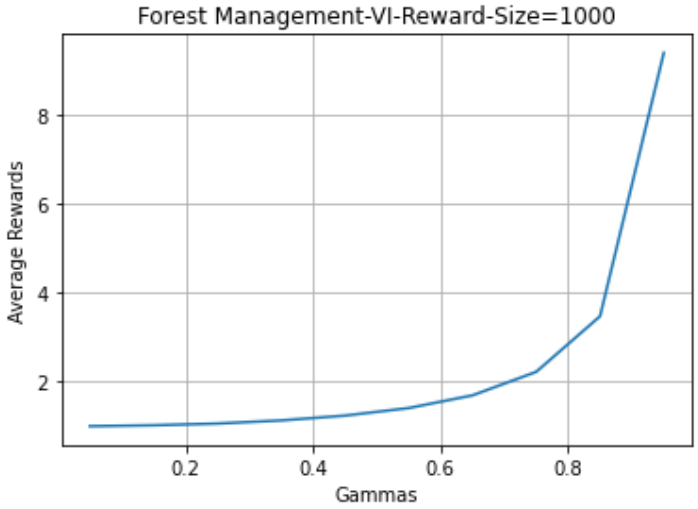






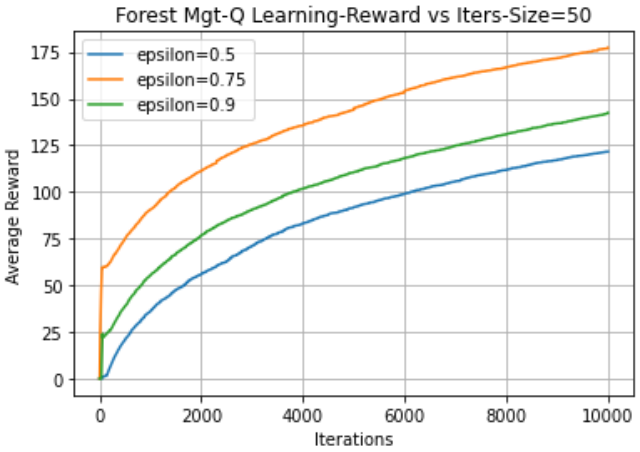
Higher values of discount factor will make the rewards discounted less. Therefore, we can find that when gamma is higher, average reward is higher.

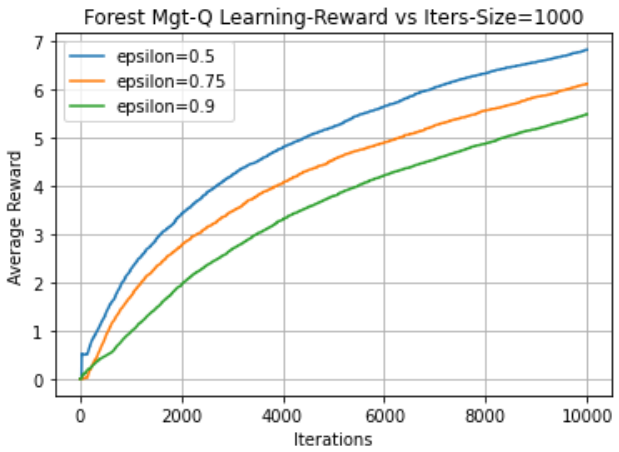




c. Q Learning

Epsilon is the exploration factor. With probability p<epsilon, the agent will take the best action, otherwise it will randomly select an action to explore the environment. We can find that when problem size is large, epsilon=0.5 helps the agent to maximize its rewards. And for the small problem, the best epsilon becomes 0.75. This implies that when problems become more complex, we generally want the agent to explore a little more to get better rewards. And for simple problems, the better strategy is simply to exploit the best action.





5 Conclusion

Compared to Value Iteration and Policy Iteration, which are model-based algorithm, Q learning is a model-free algorithm. Both Value Iteration and Policy Iteration require a "model of the environment" in that it incorporates knowledge of how to transition from one state to another. And when model is known, policy iteration and value iterations usually take fewer iterations to converge than Q learning. However, in real-world, the model is often unknown. Therefore, the exploration-exploitation factor in Q learning help the agent obtain a relatively good policy. The overall comparison between the 3 algorithms is shown in the table below.

	Policy Iteration	Value Iteration	Q Learning
Type	Model based	Model based	Model free
Initialization	Random policy	Random value function	Random Q table
Speed	Faster	Slower	Slowest
Convergence	Guaranteed to converge	Guaranteed to converge	Guaranteed to converge under certain circumstances
Number of Iterations	Low	Medium	High

6 Reference

[1] Frozen Lake. https://gymnasium.farama.org/environments/toy_text/frozen_lake/
[2] Forest Management. https://pymdptoolbox.readthedocs.io/en/latest/api/example.html
[3]. Value Iteration vs. Policy Iteration in Reinforcement Learning. https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration