Project 4 Markov Decision Processes

Name: Luping Yang

GATech ID: lyang38

***Abstract*— In this project, two MDP problems (Frozen Lake and Taxi) are solved using value iteration and policy iteration, when we know the model, i.e. the rewards and transition probability. Then, two model-free reinforcement learning algorithms (SARSA and Q-Learning) are applied to solve the two MDP problems and the impact of hyper-parameters are studied.**

Keywords*— *Value Iteration, Policy Iteration, SARSA, Q-Learning.

I. INTRODUCTION (FROZEN LAKE)

In this section, the Frozen Lake problem is described. The Frozen Lake is a classic problem in reinforcement learning. According to GYM, the agent, who is playing the Frozen Lake game, controls the character to move from start to the goal in a grid world. Most of the grids are frozen and safe to pass, but there are a few hole grids. If the character arrives at the hole grids, the game terminates, and the agent receives 0 reward. The agent get reward of 1 for successfully moving the character to the Goal grid. The key features of the problem are summarized below.

A. *States (Frozen Lake)*

As shown in Figure 1, for an $N \times N$ game, there are N^2 states, each state is the grid the character arrives. "S" denotes the start grid. "G" denotes the goal grid. The dark purple grids are hole grids. The blue grids are frozen safe grids.

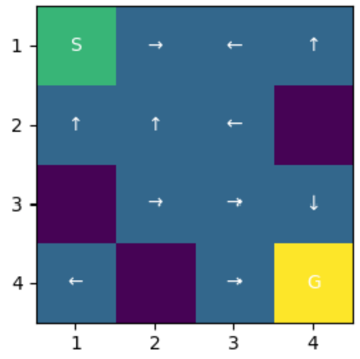


Fig. 1. Illustration of 4x4 Frozen Lake game.

B. *Actions (Frozen Lake)*

There are four actions that the character can choose to go: left, right, up, and down. When the character takes an action, the character will successfully accomplish that action with probability of 1/3, and will end up with a wrong action perpendicular to the intended action with 1/3 probability, for example, if the character chooses to go up, she/he will go up with probability of 1/3, and will go right with probability of 1/3 or go left with probability of 1/3.

C. *Rewards (Frozen Lake)*

The character will start the game at location "S". If she/he arrive at a Frozen safe grid, she/he receives 0 reward, and the game continues. If she/he arrives at a Hole, she/he receives 0 reward and the game ends. If she/he arrives at the Goal, she/he receives reward of 1 and the game ends.

I got familiar with the problem, when I took Reinforcement Learning in 2020 Fall Semester and I implemented Q-Learning and SARSA to solve Frozen Lake problem. The problem is interesting to me because the outcome of an action is uncertain which is interesting to see if SARSA and Q-Learning can uncover the underlying probability model and rewards. Another reason is that we can specify the size of Frozen Lake and see the impact of problem complexity on Value Iteration, Policy Iteration, SARSA, and Q-Learning.

II. VI VS PI (FROZEN LAKE)

In this section, I apply Value Iteration (VI) and Policy Iteration (PI) to solve Frozen Lake and compare their performances.

In this section, both VI and PI are applied to Frozen Lake with different size: 4×4, 8×8, 12×12, and 16×16. Different sizes are selected to explore the impact of problem complexity on the performances of the two algorithms. Different discount factor, gamma, are chosen for VI and PI to demonstrate the importance of discount factor.

A. *Convergence (Frozen Lake)*

Figure 2 shows the number of iterations taken for VI and PI to converge using different gammas, [0.8, 0.9, 0.99, 0.9999, 0.999999, 0.99999999] and map size.

For Valuation Iteration, the convergence is defined as the maximum value change for all states is less than 10^{-12}. For Policy Iteration, convergence is defined as there is no change in optimal policy for two consecutive iterations. For PI, we need calculate the value function in each iteration, the threshold I used to define the convergence of value function in PI is 10^{-3}. This threshold should not be too large for better calculation of value function and should not be too small, if too small, it will take a long time to calculate the value function in each iteration when the policy is still significantly suboptimal.

In Figure 2, the upper subplot shows the number of iterations of VI. It is observed that the number of iterations increases with gamma, which makes sense, because the larger the gamma, the slower for an exponential function to converge. The number of iterations also increase with the size of the game, which also makes well sense, because the more states, the longer it takes for the final rewards to spread to the value of each state.

In Figure 2, the lower subplot shows the number iteration for PI. Similar patten is observer for PI as for VI: the larger the gammas, the greater the number of iterations; the larger the size of the game, the greater the number of iterations. But for PI, the patten has more variation than value iteration. My understanding is that, since the policy is randomly initialized, sometimes we might be lucky to choose an optimal action at some key states, thus PI converges faster in some cases.

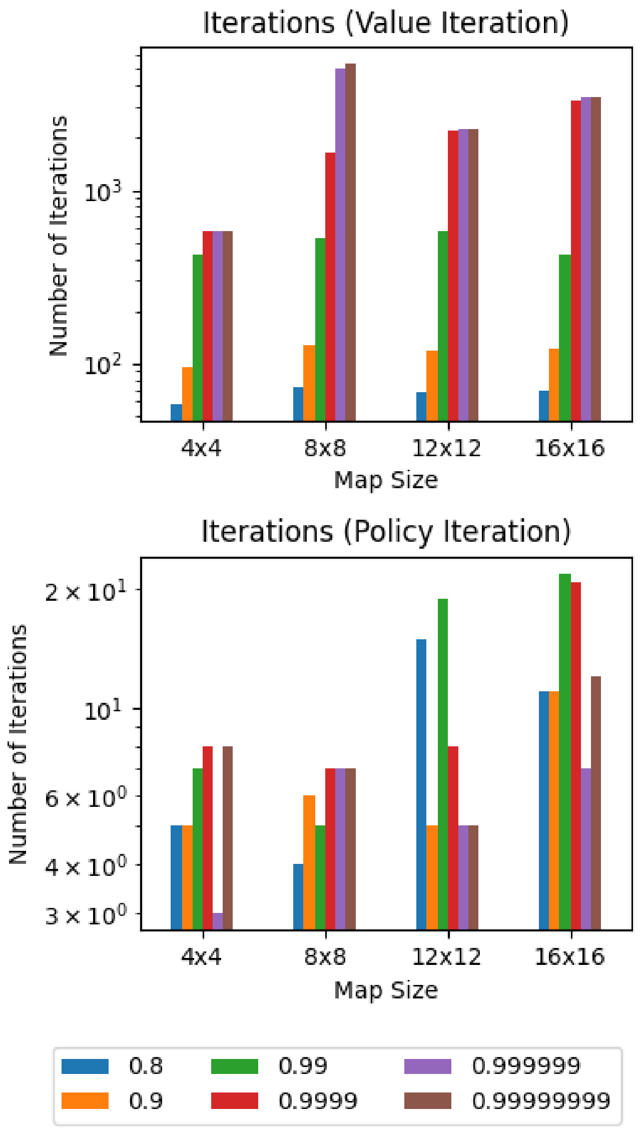


Fig. 2. Number of Iteration to converge for Frozen Lake.

In Figure 2, comparing VI to PI, it is obvious that PI takes much smaller number of iterations to converge. This does not mean that PI is more efficient than VI, because in each iteration of PI, it needs to calculate the value function. The value functions are calculated either by linear algebra or some sort of "value iteration", both of which are computationally expensive.

To see the actual time consumed by VI and PI, the time to convergence is plotted in Figure 3. In Figure 3, it is observed that the time used by VI and PI to converge increase with gamma and with size of the game. The time used by VI is much longer than PI, indicating that PI is more efficient for Frozen Lake in my setting of experiments. It is worth emphasizing that PI is not universally more efficient than VI across different games and settings. In my setting, the threshold of convergence for VI is 10^{-12}, which might be a too strong requirement for VI's convergence. Also, the threshold of convergence for value function in PI is 10^{-3}, which might be well chosen, so that PI runs efficiently for this game.

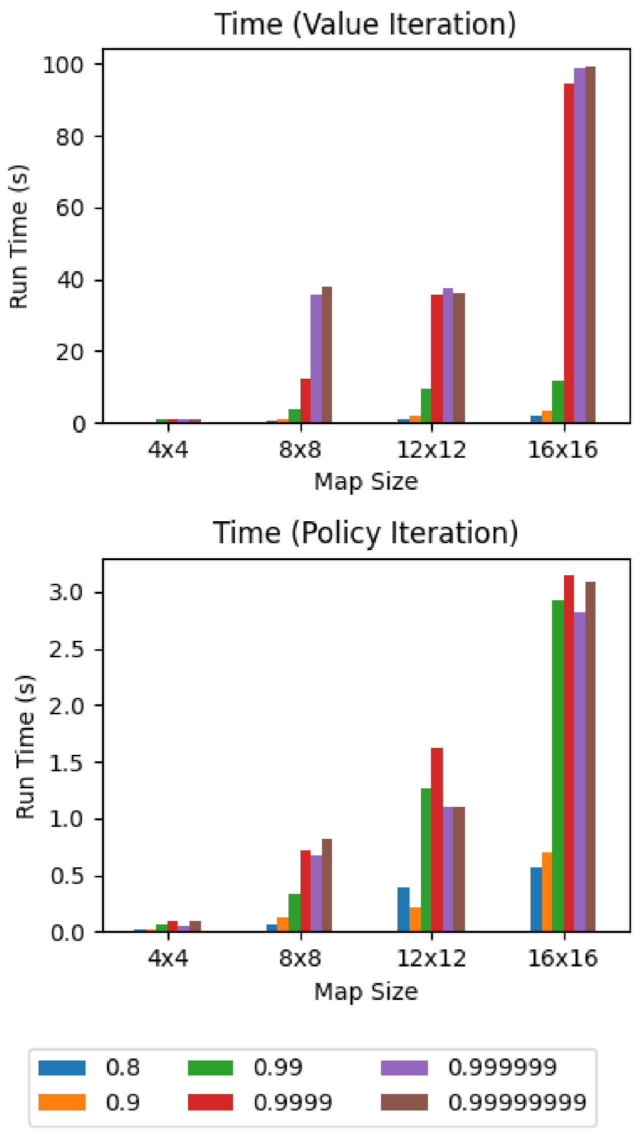


Fig. 3. Time to converge for Frozen Lake.

B. *Performance (Frozen Lake)*

To evaluate the performance of the optimal policy generated by VI and PI, the following are conducted: 1) comparison of optimal policy; 2) play the Frozen Lake game 1000 times using the generated optimal policies and calculate the average rewards of 1000 games.

Figure 4 plots the optimal policies generated by VI and PI. The left columns display the optimal polices of VI; the right column display the optimal policies of PI. From Figure 4, it is observed that the optimal policies are the same for 4×4 Frozen Lake game but are different for games with larger map size. The reason might be: 1) even through the algorithms converge, the optimal policies might not be found; 2) there might exist multiple optimal policies.

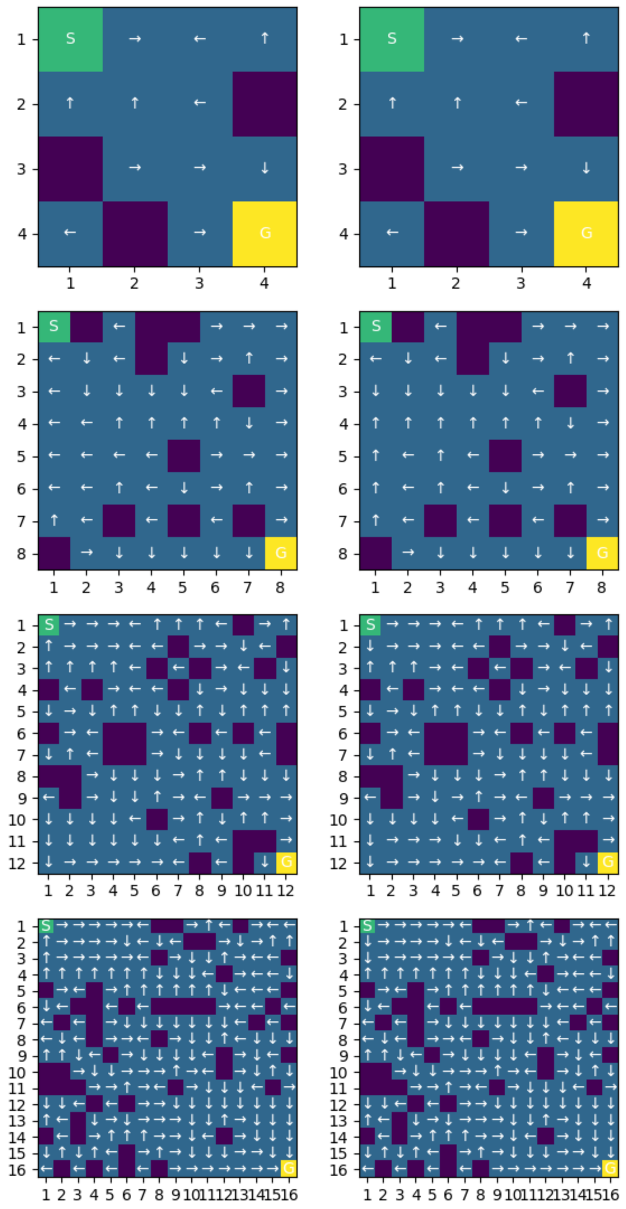


Fig. 4. Comparison of policies for Frozen Lake. (left displays optimal polices generated by VI; right displays optimal policies generated by PI)

In Figure 4, one interesting observation is that: in the 8×8 game, there are three holes in Row 7, but each is separated by a frozen safe grid. The best action suggested by the optimal policies of VI and PI for the in-between frozen safe grid is to go left, which means that optimal policies suggest the character head towards the hole, which is counterintuitive. But after thinking a while, it starts to make sense to me. If I were the agent, I would suggest the character go down, but the character will end up going down in only 1/3 chance and 1/3 chance to fall into the left hole and 1/3 chance to fall in the right hole. However, according to the optimal policies, the character should go left, then the character will end up with probability of 1/3 falling into the hole and 1/3 chance to go up and 1/3 chance to go down. Thus, the chance to fall into hole is 1/3 smaller if go left suggested by optimal policies, which is better than go down suggested by my intuition.

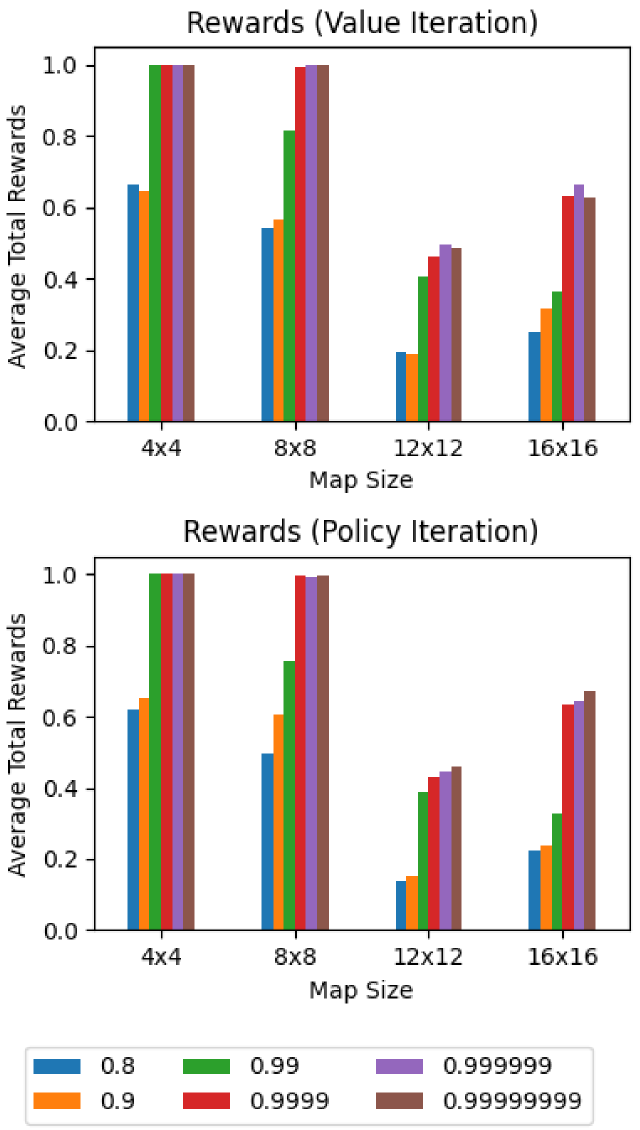


Fig. 5. Comparison of rewards for Frozen Lake.

Figure 5 shows the average rewards achieved by the agent adopting the optimal polices generated by VI and PI and playing the game 1000 times. Looking at Figure 5, it is observed that as the size of game increases, we need to choose gamma closer to 1, which makes sense. Since the Frozen Lake only pay reward of 1 at the end of game when the character arrives at goal grid. We need to select a large gamma so that reward in the far future can be accounted. If we choose a small gamma, the far-future reward will quickly go to zero after a few rounds of discount and could **NOT** be populated to early states. For large size game such as 12×12 and 16×16, the optimal policies only achieved average rewards around 0.5, even with gamma=0.99999999. In Figure 5, comparing the average rewards of VI and PI, it is observed that the performances of VI and PI are very close.

III. Q-LEARNNG VS SARSA (FROZEN LAKE)

In this section, I implemented Q-Learning and SARSA and compared their performance.

Q-Learning is model-free on-policy algorithm. Model-free means that we do not know the model, which are the transition probability and rewards. But we learn transition probability and rewards by playing the game. On-policy means that the action chosen at each step (except the first step) is the best action based on the current-available optimal policy, which is derived from the Q-table.

SARSA is model-free off-policy. Off-policy means that the action chose at each step is by epsilon-greedy method: the agent has a ϵ-probability to randomly choose an action or $(1-\epsilon)$-probability to choose the best action based on the current-available optimal policy.

For Q-Learning and SARSA, FrozenLake-v0 defines "solving" as getting average reward of 0.78 over 100 consecutive trials. Thus, Q-Learn and SARSA is determined to achieve convergence if average reward of 0.78 over past 100 consecutive trials is achieved and there is no significant change in the average rewards for the past 100 episodes. To evaluate the impact of hyper-parameters, I ran 2000 episodes even when the algorithms converge.

Due to the huge computation time, only 4×4 game is experimented. In this section, the following hyper-parameter are tuned for Q-learning and SARSA:

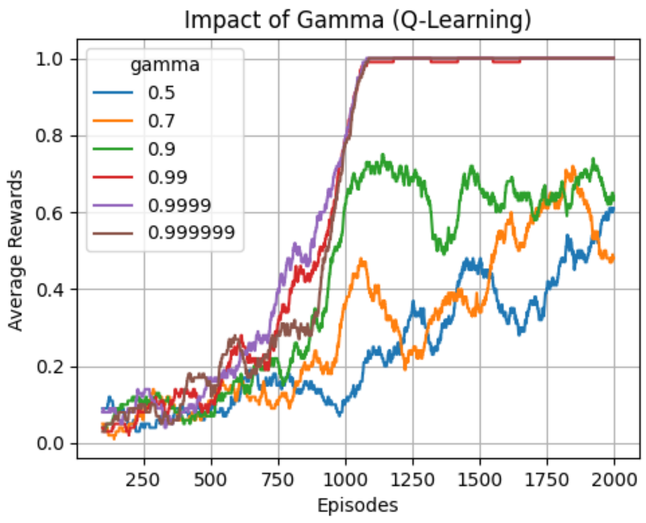
- gammas = [0.5, 0.7, 0.9, 0.99, 0.9999, 0.999999]
- alphas = [0.05, 0.1, 0.3, 0.5, 0.8, 0.9, 0.95]
- epsilon_inits = [0.4, 0.5, 0.8, 1]
- epsilon_mins = [0, 0.001, 0.01, 0.1]
- epsilon_decays = [0.0001, 0.001, 0.01]

When comparing the impact of one of parameter, the values used for other parameters are: gamma=0.99, alpha=0.3, epsilon_init=1, epsilon_min=0, epsilon_decay=0.001.

A. *Discount Factor, Gamma (Frozen Lake)*

Figure 6 shows the impact of discount factor, gamma, on the convergence and performance of Q-learning and SARSA. It is observed that gamma values larger than or equal to 0.99 work well for Q-learning and SARSA: both algorithms converge and find the optimal policy. This is consistent with VI and PI: for gammas larger than or equal to 0.99, VI and PI achieves average reward of 1.

For gammas less than or equal to 0.9, the both Q-learning and SARSA do not achieve rolling average rewards higher than 0.78: the rolling average rewards go up and down in the range that is significantly below 0.8. The reason is that the Frozen Lake only pay reward of 1 at the end of game when the character arrives at goal grid. We need to select a large gamma so that reward in the far future can be accounted. If we choose a small gamma, the far future reward will quickly go to zero after a few rounds of discount and could **NOT** be populated to early states.



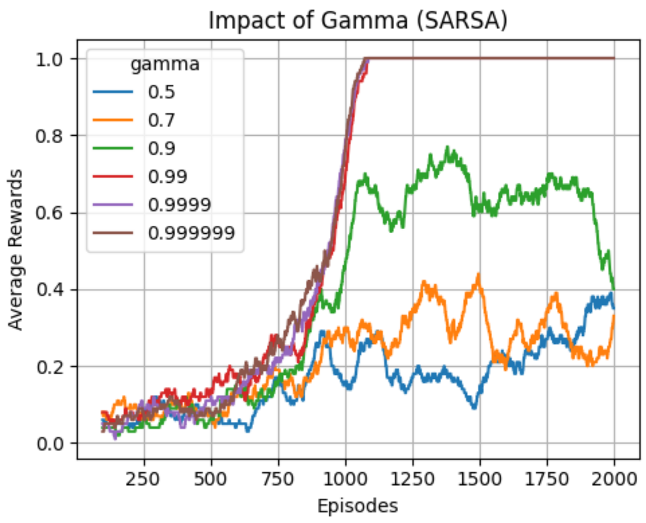


Fig. 6. Impact of gamma on Q-learning and SARSA.

B. *Learning Rate, Alpha (Frozen Lake)*

Learning rate, alpha, sets how much we want to learn from one observation. Setting alpha to 0 means that the Q-values are never updated, which means that nothing is learned. Setting alpha to a high value such as 0.9 means that learning can occur quickly but is more susceptible to noises or shocks. So alpha not only impact the speed of convergence but also influence whether we can achieve convergence.

Figure 7 display the impact of alpha on the convergence of Q-learning and SARSA. It is observed that only alpha=0.05 does not work for Q-learning and alpha=0.05 and 0.1 do not work for SARSA, indicating that a wide range of alpha works well for Q-learning and SARS. The reason might be that the 4×4 game is still relatively easy to solve.

C. *Epsilon-Init (Frozen Lake)*

Epsilon controls the balance of exploration and exploitation. A good epsilon can well balance exploration and exploitation: we want our algorithms to explore more in the early stage so that they have a better estimate of the transition probability and rewards, especially when the rewards happen in the far-future; on the other hand, we want our algorithms to exploit after they have explored sufficiently.

Epsilon_init controls the exploration and exploitation in the early stage. At this stage, we want to explore more.

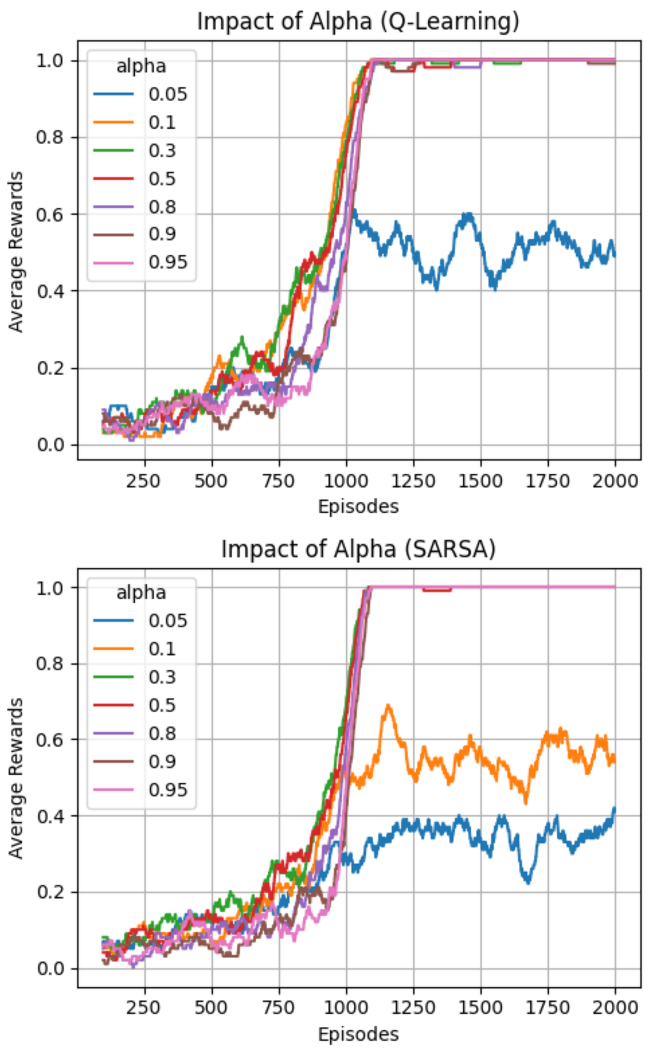


Fig. 7. Impact of alpha on Q-learning and SARSA.

Figure 8 presents impact of initial epsilon value on convergence of Q-learning and SARSA. From Figure 8, it is observed that the pattern of rolling average rewards is similar for Q-learning and SARSA. In subplot 3 of Figure 8, the epsilon goes linearly to 0 when episode is around 350 for epsilon_init=0.4. For Q-learning and SARSA, the average rewards are close to 0, when epsilon_init=0.4, indicating that using this initial epsilon value is not enough for Q-learning and SARSA to explore and discover lump-sum reward in the far-future.

For epsilon_init=[0.5, 0.8, 1.0], there are still interesting observations: 1) for these initial epsilon values, both Q-learning and SARSA work; 2) but a smaller initial epsilon value make both Q-learning and SARSA converge faster: indicating that Q-learning and SARSA can converge faster if they can start exploit earlier if they have explored enough.

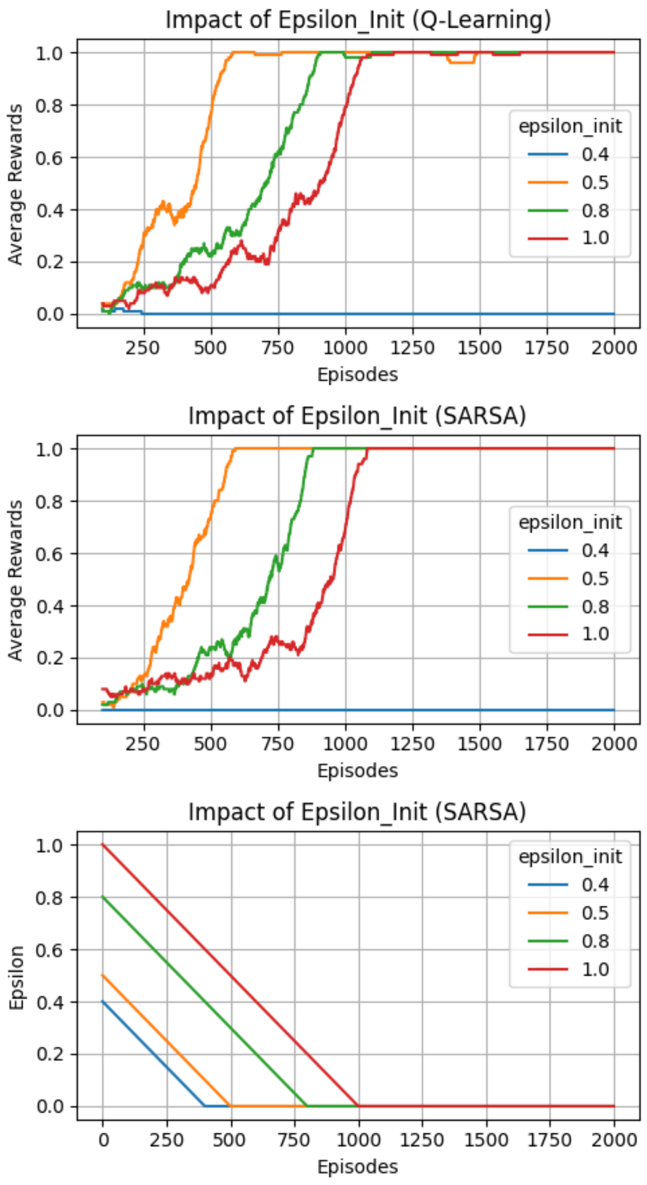


Fig. 8. Impact of epsilon_init on Q-learning and SARSA.

D. *Epsilon-Min (Frozen Lake)*

Epsilon-min controls the exploration and exploitation in the late stage. At this stage, we want to exploit more.

In Figure 9, similar pattern is observed for Q-learning and SARSA. The rolling average rewards decrease with the increase of epsilon_min, indication that it is best for the two algorithms to exploit once they have explored enough and found the optimal policy, which totally makes sense.

The subplot 3 of Figure 9 shows that the epsilon value is still 0.1 after 1000 episodes when epsilon_min=0.1. When epsion_min=0.1, the rolling average rewards of Q-learning is around 0.6, whereas the rolling average rewards of SARSA is even lower, around 0.5. This is because SARSA still explores more than Q-learning at the late stage: each action of SARSA is generated by ϵ-greedy, whereas only the first action of Q-learning is generated by ϵ-greedy, which is the difference between off- and on-policy.

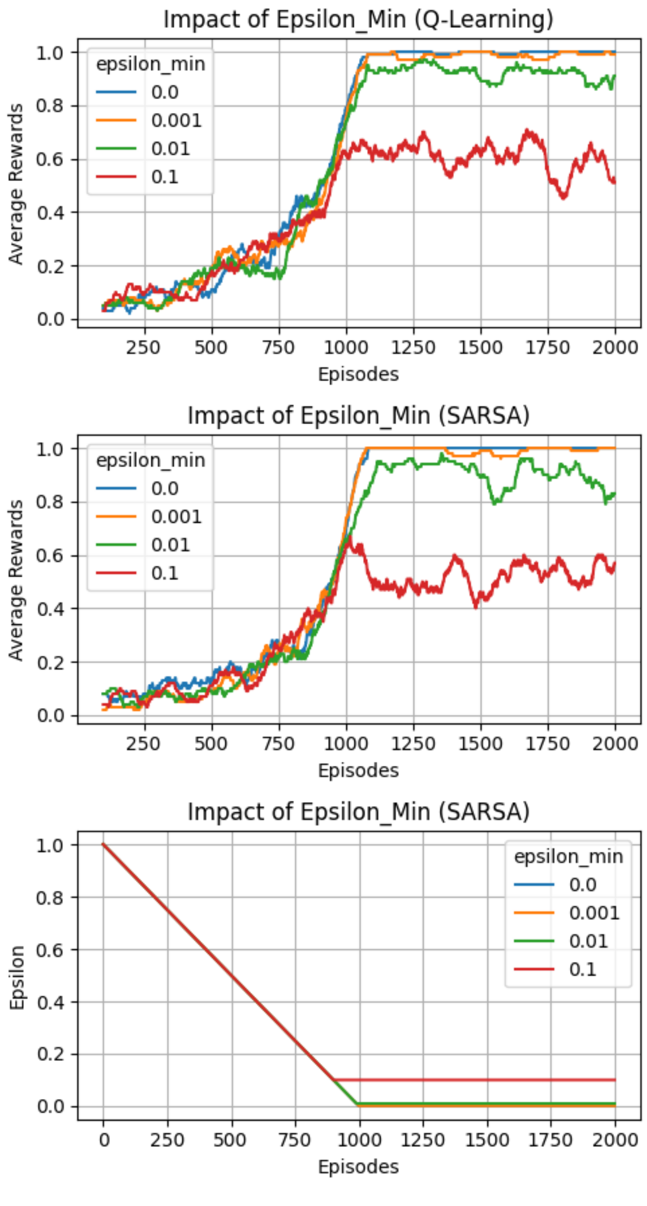


Fig. 9. Impact of epsilon_min on Q-learning and SARSA.

E. *Epsilon-Decay (Frozen Lake)*

Epsilon-min controls the speed of transition from exploration to exploitation.

In Figure 10, the results of three epsilon_decay values are displayed: 0.0001, 0.001, 0.01. When epsilon_decay=0.01, the speed of transition from exploration to exploitation is very fast: the algorithms stop exploring when episode=100. Exploration is significantly insufficient. Thus, SARSA does not make any progress in this game and get 0 average rewards. The case for Q-learning is better; it achieves a rolling average reward around 0.5. One interesting question to ask is why the rolling average reward is 0 for SARSA whereas 0.5 for Q-learning. My understanding is that the information extracted by exploration is not enough to counteract the noise added by SARSA's ϵ-greedy method: each action of SARSA is generated by ϵ-greedy, whereas only the first action of Q-learning is generated by ϵ-greedy.

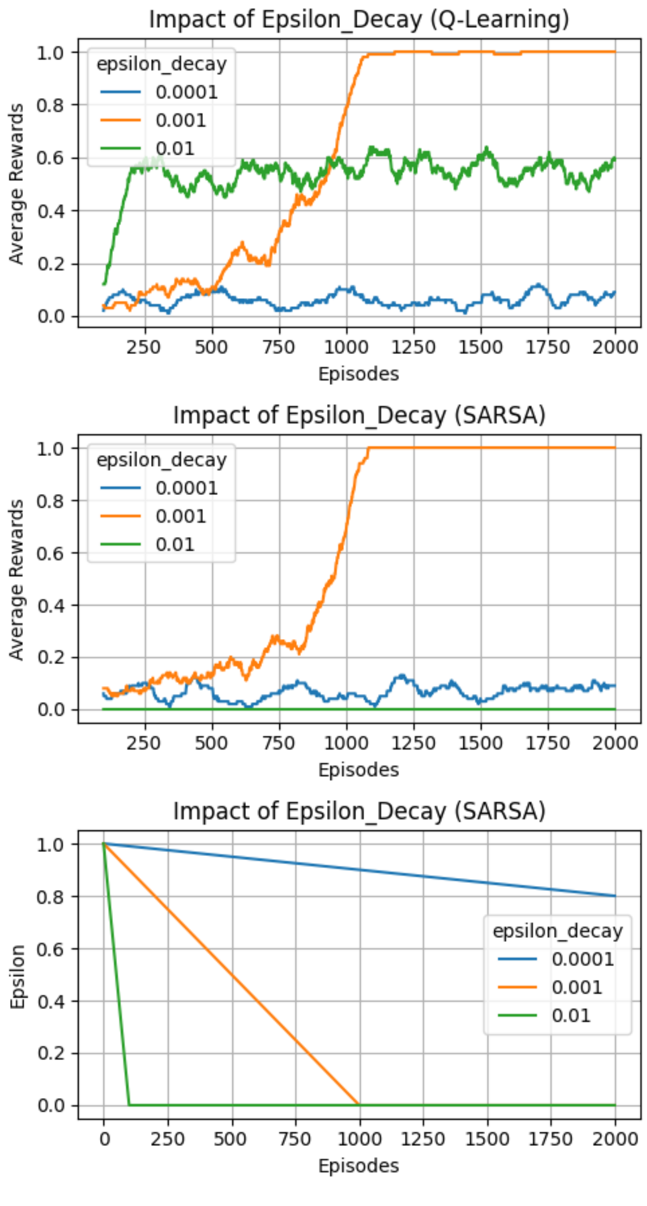


Fig. 10. Impact of epsilon_decay on Q-learning and SARSA.

When epsilon_decay=0.001, the speed from exploration to exploitation is just right: the algorithms stop exploring when episode=1000. Both Q-Learning and SARSA converge when episode is around 1100. After that, both algorithms achieve almost perfect score.

When epsilon_decay=0.0001, the speed of transition from exploration to exploitation is very slow: the algorithms are still exploring when episode=2000. But when the algorithms stop exploring when episodes=10000, the two algorithms will start to converge.

IV. Introduction (Taxi)

In this section, the Taxi problem is described. As shown in Figure 11, in Taxi problem is a grid game, and there are 4 locations indicated by R(ed), B(lue), G(reen), and Y(ellow). At the beginning of each episode, the taxi starts off at a random grid and the passenger locates at a random grid. The taxi first needs to drive to the passenger's grid, pick up the passenger, take the passenger to the destination grid (one of the four specified locations), and lastly drop off the passenger. Once the passenger is dropped off, the game terminates. There are walls marked as "|", which the taxi cannot pass through, whereas the taxi can pass through grids separated by ":".

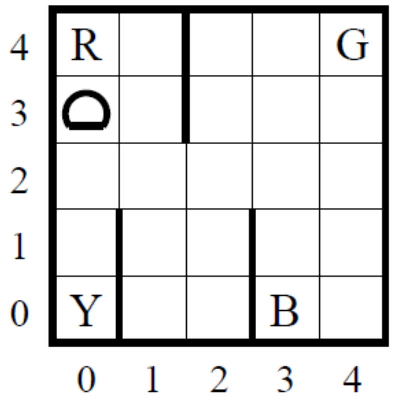


Fig. 11. Illustration of Taxi game. (Figure from [2])

A. States (Taxi)

As shown in Figure 11, the Taxi problem is a grid game of size 5×5. There are 500 states: 25 grids, 5 locations for the passenger, and 4 destinations.

B. Actions (Taxi)

There are six actions in Taxi problem:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pick up passenger
- 5: drop off passenger

When the taxi driver takes an action, the driver will accomplish that action with probability of 1. That is to say each action is deterministic.

C. Rewards (Taxi)

The reward for each action is -1 and additional reward of +20 for successfully dropping off the passenger in the destination grid. If the taxi pickup or drop off the passenger illegally, there is a reward of -10.

This problem is interesting to me because it has a hierarchical structure. This problem consists of two subproblems: 1) pick up the passenger; 2) deliver the passenger. It is interesting to see how the reinforcement learning algorithms can solve this problem.

V. VI vs PI (Taxi)

In this section, I apply Value Iteration (VI) and Policy Iteration (PI) to solve Taxi problem and compare their performances. In this section, both VI and PI are applied to Taxi problem of size 5×5. Different discount factor, gamma, are chosen for VI and PI to demonstrate the importance of discount factor.

A. Convergence (Taxi)

Figure 12 shows the number of iterations taken for VI and PI to converge using different gammas, [0.8, 0.9, 0.99, 0.9999].

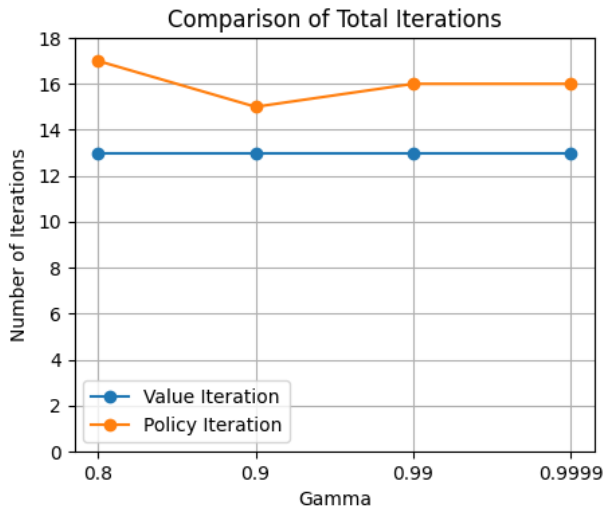


Fig. 12. Number of Iteration to converge for Taxi problem.

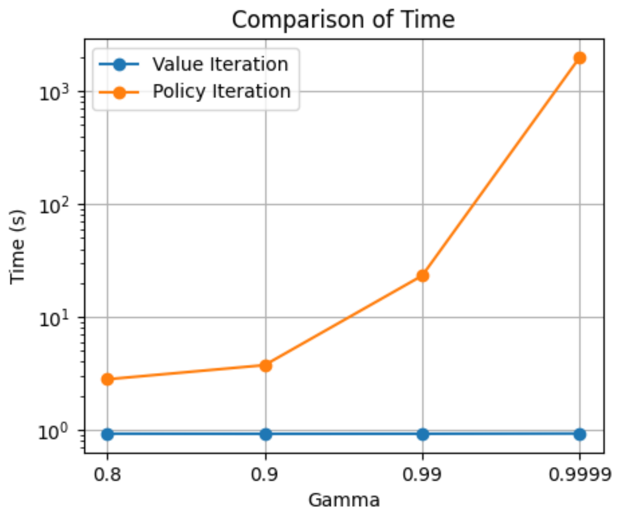


Fig. 13. Time to converge for Taxi problem.

For Valuation Iteration, the convergence is defined as the maximum value change for all states are less than 10^{-12}. For Policy Iteration, convergence is defined as there is no change in optimal policy for two consecutive iterations. For PI, we need calculate the value function in each iteration, the threshold I used to define the convergence of value function in PI is 10^{-3}.

In Figure 12, The number of iterations is compared. The number of iterations of VI is even smaller than PI, which is surprisingly different from Frozen Lake problem. In Frozen Lake, the number of iterations of VI is much larger than that of PI. The possible reason might be that there are many more states in Taxi problem than in Frozen Lake problem. It takes more iterations for policy to converge.

To see the actual time consumed by VI and PI, the time to convergence is plotted in Figure 13. In Figure 13, it is observed that the time used for VI and PI to converge increase with gamma. The time used by PI is much longer than VI, indicating that VI is more efficient for Taxi problem.

B. *Performance (Taxi)*

To evaluate the performance of the optimal policy generated by VI and PI, the following are conducted: 1) comparison of optimal policy; 2) play the Taxi game 1000 times using the optimal policies and calculate the average rewards.

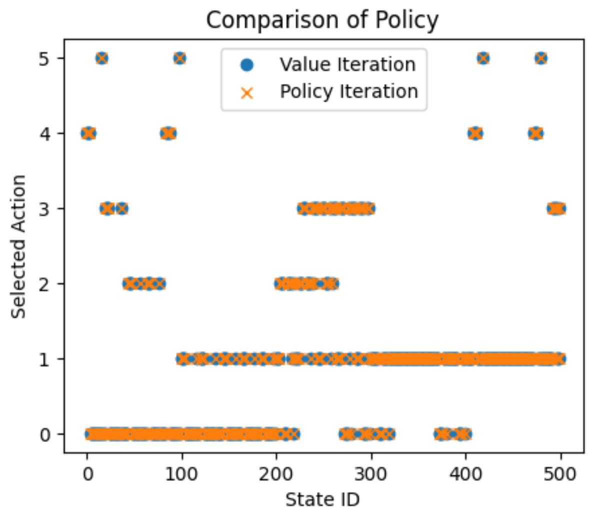


Fig. 14. Comparison of policies for Taxi problem.

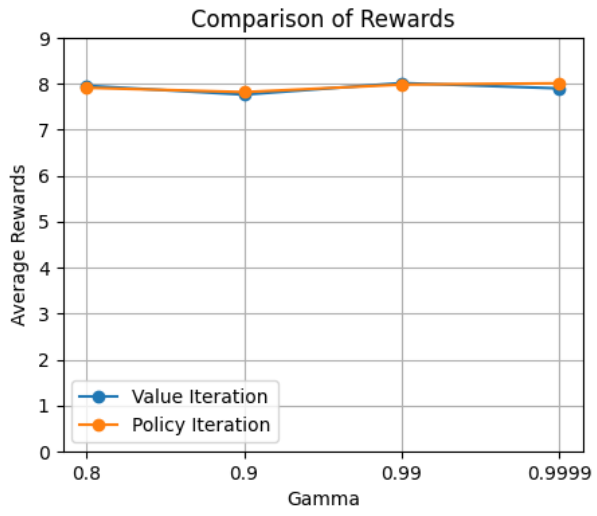


Fig. 15. Comparison of rewards for Taxi problem.

Figure 14 plots the optimal actions against their states for VI and PI. It is observed that the optimal policies are the same for VI and PI.

Figure 15 shows the average rewards achieved by the Taxi agent adopting the optimal polices generated by VI and PI and playing the game 1000 times. Looking at Figure 15, it is observed that the average rewards achieved by VI and PI are very close. The optimal policies of VI and PI under different discount factor achieve similar average rewards, indicating that the selected gamma values work well for VI and PI, and Taxi problem is less sensitive to discount factor.

VI. Q-LEARNNG VS SARSA (TAXI)

In this section, I apply Q-Learning and SARSA to solve Taxi problem and compare their performance under different hyper-parameters.

For Q-Learning and SARSA, the convergence is defined if there is no significant change in the average rewards for the past 100 episodes. For comparison of hyper-parameters, I ran 2000 episodes.

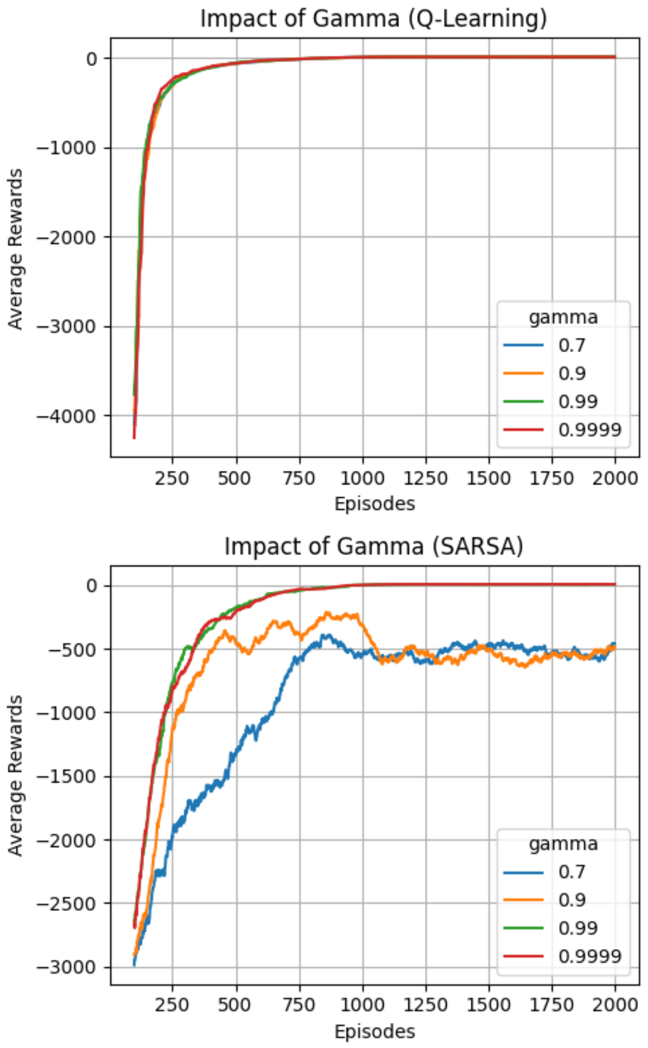


Fig. 16. Impact of gamma on Q-learning and SARSA.

In this section, the following hyper-parameter are tuned for Q-learning and SARSA:

- gammas = [0.7, 0.9, 0.99, 0.9999]
- alphas = [0.05, 0.1, 0.3, 0.5, 0.8, 0.9, 0.95]
- epsilon_inits = [0.4, 0.5, 0.8, 1]
- epsilon_mins = [0, 0.001, 0.01, 0.1]
- epsilon_decays = [0.0001, 0.001, 0.01]

When comparing the impact of one of parameter, the values used for other parameters are: gamma=0.99, alpha=0.3, epsilon_init=1, epsilon_min=0, epsilon_decay=0.001.

A. *Discount Factor, Gamma (Taxi)*

Figure 16 shows the impact of discount factor, gamma, on the convergence and performance of Q-learning and SARSA. Figure 16 shows that all the select gamma values work well for Q-learning. But SARSA requires gamma larger than or equal to 0.99 to work well, which might be due to that SARSA explores more than Q-learning.

B. *Learning Rate, Alpha (Taxi)*

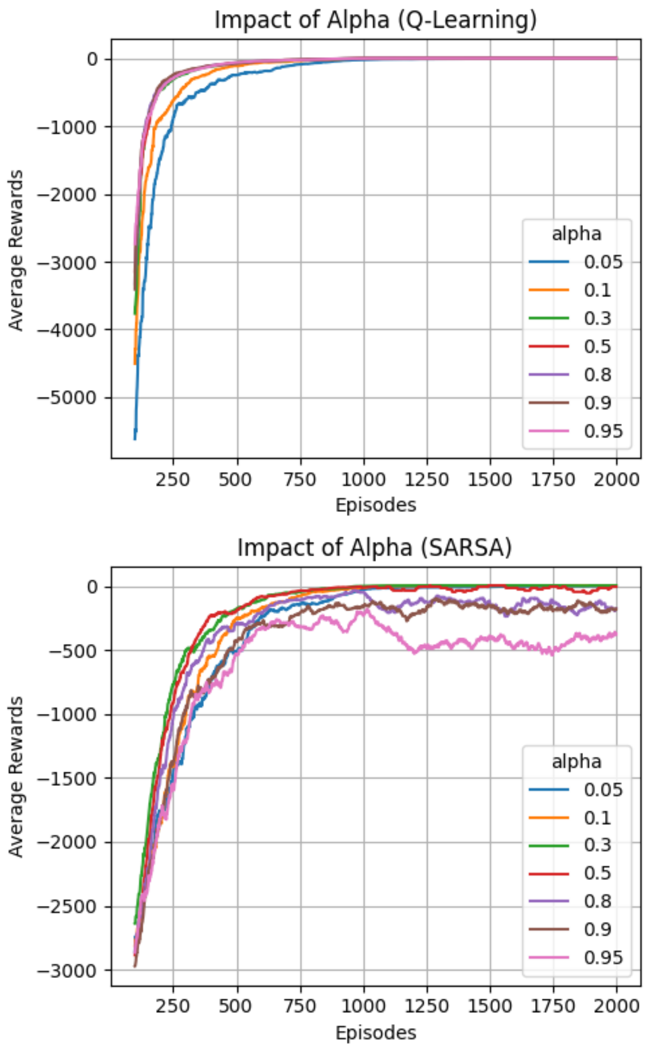


Fig. 17. Impact of alpha on Q-learning and SARSA.

Figure 17 display the impact of alpha on the convergence of Q-learning and SARSA. It is observed different alpha values influence the convergence speed of Q-learning, but under these alpha values Q-learning eventually finds the optimal policy. However, alpha larger than or equal to 0.8 do not work well for SARSA: SARSA is random at each step and we do not want the shocks to influence too much.

C. *Epsilon-Init (Taxi)*

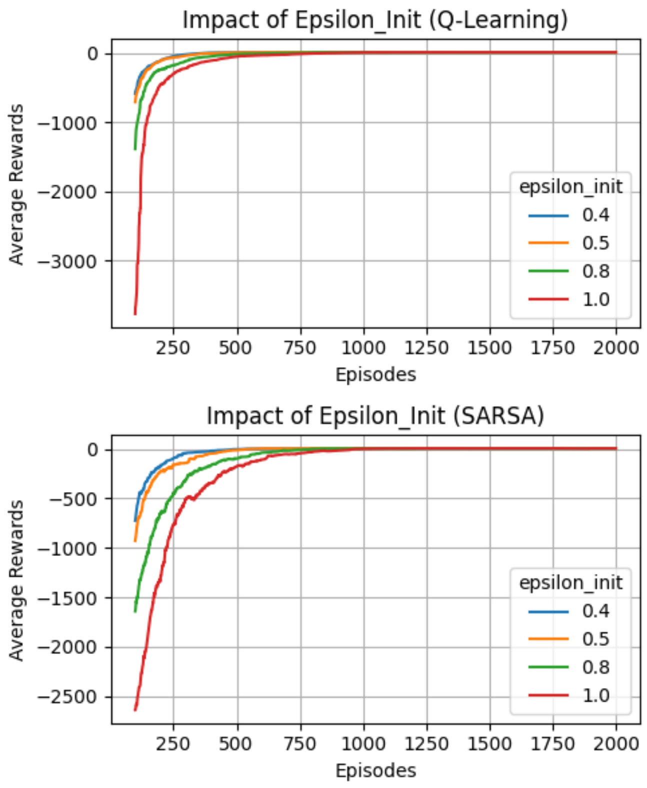


Fig. 18. Impact of epsilon_init on Q-learning and SARSA.

Figure 18 presents impact of initial epsilon value on convergence of Q-learning and SARSA. From Figure 18, it is observed that the pattern of rolling average rewards is similar for Q-learning and SARSA. Both Q-learning and SARSA converges and find optimal policy. But Q-learning converges faster than SARSA.

D. *Epsilon-Min (Taxi)*

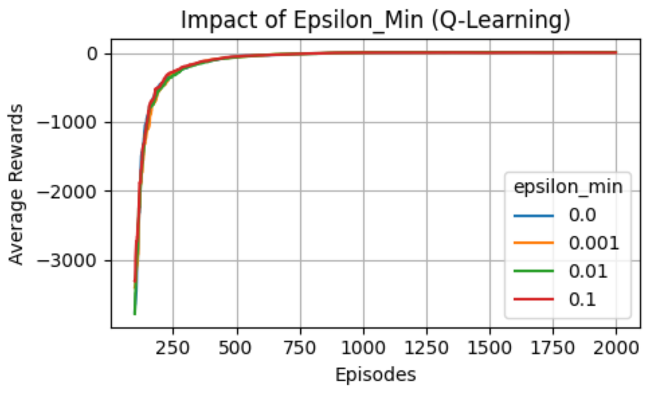
In Figure 19, similar pattern is observed for Q-learning and SARSA. For the selected epsilon_min values, both Q-learning and SARSA find optimal policies.

E. *Epsilon-Decay (Taxi)*

In Figure 20, the results of three epsilon_decay values are displayed: 0.0001, 0.001, 0.01. When epsilon_decay=0.01, the speed of transition from exploration to exploitation is very fast: the algorithms stop explore when episode=100. But it seems that with epsilon_decay=0.01, both Q-learning and SARSA have explored enough and converged to the optimal police the fastest.

When epsilon_decay=0.001, the algorithms stop exploring when episode=1000. Both Q-Learning and SARSA converge when episode=1000. After that, both algorithms achieve almost perfect score.

When epsilon_decay=0.0001, the speed of transition from exploration to exploitation is very slow: the algorithms are still exploring and epsilon=0.8 when episode=2000. Both Q-learn and SARSA do not converge yet. But when the algorithms stop exploring when episodes=10000, I suspect that the two algorithms will start to converge.



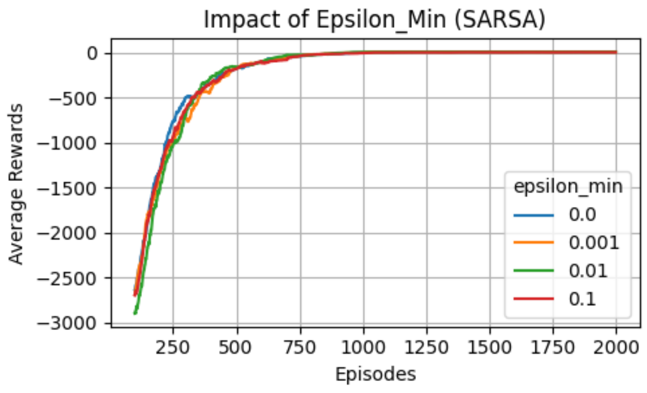
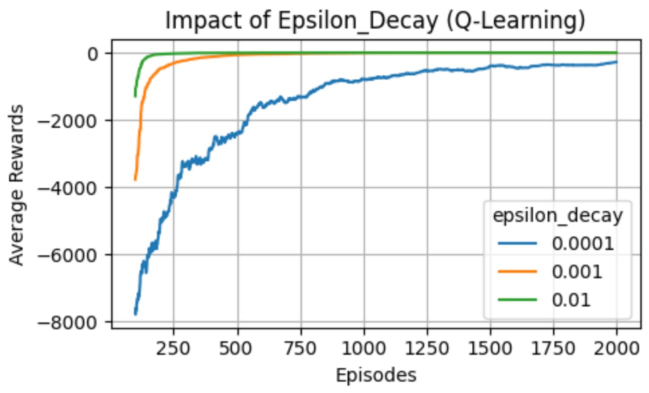


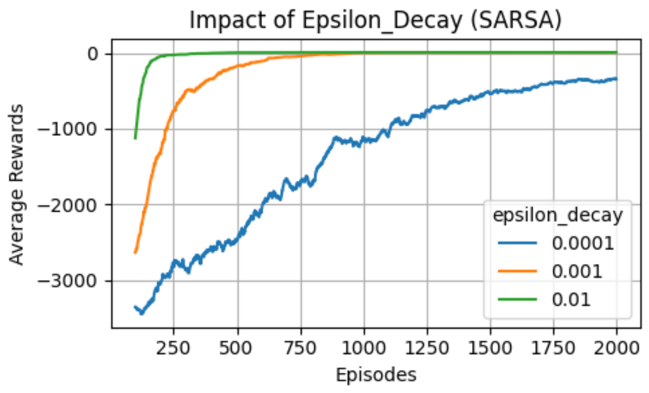
Fig. 19. Impact of epsilon_min on Q-learning and SARSA.

Conclusion and Future WOrk

In project, I applied Value Iteration and Policy Iteration to Froze Lake with different map size under different gamma values, and to Taxi problem under different gamma values. It is found that VI uses much longer time to converge than PI in Frozen Lake, whereas PI uses much long time to converge than VI in Taxi problem, indicating that the efficiency of VI and PI really depends on the problem and the thresholds for convergence.

Then I use Q-learning and SARS to solve Frozen Lake and Taxi problems and test their performance under 5 sets of hyper-parameters. It is found that hyper-parameters are critical for Q-learning and SARSA. If the game pays rewards in the far-future, then a large discount factor close to 1 should be used. If we want to limit the impact of one step, then a small learning rate should be used. Epsilon controls the balance between exploration and exploitation. It is important for Q-learning and SARSA to explore sufficiently before they stop exploring.





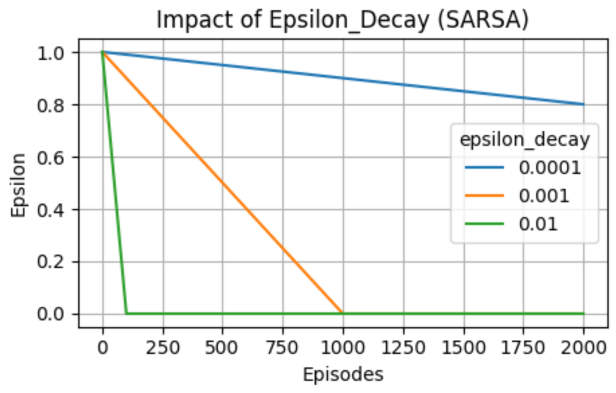


Fig. 20. Impact of epsilon_decay on Q-learning and SARSA.

References

[1] https://gym.openai.com/envs/FrozenLake-v0/

[2] T Erez, Y Tassa, E Todorov, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", 2011.

[3] https://gym.openai.com/envs/Taxi-v3/