AE 598 HW2: Tabular Methods

Yavan Meda

I. Nomenclature

$\mathcal{S}$	=	Set of all possible states.
$\mathcal{A}$	=	Set of all possible actions.
$P(s'\|s,a)$	=	Transition probability function (unknown in model-free setting).
$R(s,a)$	=	Expected immediate reward (unknown in model-free setting).
γ	=	Discount factor, $\gamma \in [0,1]$.
$\pi(a\|s)$	=	Policy, probability of selecting action a in state s.
$s_t, a_t, r_{t+1}, s_{t+1}$	=	Sampled transition from the environment.
G_t	=	Expected discounted return starting at time t.
$P(s'\|s,a)$	=	Transition probability function (unknown in model-free setting).
$Q(s,a)$	=	Action-value function, expected return starting from (s,a).
$q_*(s,a)$	=	Optimal action-value function.
$R(s,a)$	=	Expected immediate reward (unknown in model-free setting).

II. INTRODUCTION

A. Problem Statement

The goal of this assignment is to implement multiple model-free reinforcement learning algorithms on the Frozen Lake problem and compare their performance. The Frozen Lake problem [1], developed by MIT's Gymnasium library, constitutes navigating a frozen lake, avoiding hazards, and dealing with the slippery surface in order to reach a goal. The agent must move across the lake whilst avoiding holes in the frozen lake's surface towards the goal.

B. Model

The model of the frozen lake is a 4x4 array of tiles. Each tile represents a state in which the agent can exist: ice, a hole, or the goal. The hole tiles and the goal tile are the terminal states. The actions the agent can take are to move one tile left, right, up, or down; the agent will continue to make moves until it reaches the goal or falls into a hole. The agent starts in the upper most left tile. The environment can be instantiated such that the ice tiles are slippery, resulting in a 1/3 chance of the agent moving in its desired direction and a 1/3 chance of the agent moving either left or right relative to its chosen direction. The Frozen Lake Environment is defined as:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{1}$$

in which the agent does not have access to the transition function P nor reward function R, necessitating the use of model-free algorithms. This involves the agent interacting with the environment and recording its experiences in the form of tuples $(s_t, a_t, r_{t+1}, s_{t+1})$, in which it will assess the return G_t it earned during the episode, using that to inform how it should maximize its policy. The specific implementation of the various model-free reinforcement learning algorithms is discussed in the next section.

III. METHODOLOGY

The first model-free algorithm that was implemented was the Monte Carlo Control algorithm with ε-soft policy 1. The algorithm works by first collecting experience in the form of episodes. Once the algorithm completes an episode, it calculates the set of rewards it earned in that episode and tracks the average reward that each state-action pair (S_t, A_t) produced, using those averages to maximize its action value function $Q(S_t, A_t)$ such that it converges on the optimal

policy. However, an important thing to note is the ε-soft policy, whereby the algorithm is made to always engage in some level of exploration by never allowing any one action to have a zero probability of being chosen for a given state.

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|\mathcal{A}(s)|}, & \text{if } a = \arg\max_{a'} Q(s, a'), \\ \dfrac{\varepsilon}{|\mathcal{A}(s)|}, & \text{otherwise.} \end{cases} \tag{2}$$

By including ε-soft policies, it ensure that the algorithm does not remain indefinitely in a local minimum, ensuring that it can find the optimal policy. The specific pseudo-code for its implementation can be seen below.

A. Monte Carlo Control with ε-soft policy [2]

Algorithm 1: On-policy first-visit MC control (for ε-soft policies)

Initialize arbitrary ε-soft policy π, $Q(s, a)$, and empty Returns(s, a) for all s, a;
repeat
  Generate episode $(S_0, A_0, R_1, \ldots, S_T)$ following π;
  $G \leftarrow 0$;
  for $t \leftarrow T - 1$ **to** 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$;
    if (S_t, A_t) *not in* $(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})$ **then**
      Append G to Returns(S_t, A_t);
      $Q(S_t, A_t) \leftarrow$ avg(Returns(S_t, A_t));
      $A^* \leftarrow \arg\max_a Q(S_t, a)$;
      foreach $a \in \mathcal{A}(S_t)$ **do**
        $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(S_t)|}, & a = A^* \\ \frac{\varepsilon}{|\mathcal{A}(S_t)|}, & a \neq A^* \end{cases}$;
until *forever*;

The second algorithm that was compared is the State, Action, Reward, State, Action algorithm, also known as the SARSA algorithm. This algorithm differs from the Monte Carlo algorithm by the fact that it evaluates its state-action function $Q(S, A)$ incrementally as it makes decisions in the episode, rather than only after the episode is complete, allowing it to learn continuously. Take note of the fact that, along with ε, another parameter, α, is introduced in this algorithm. Below is the specific line of pseudo-code that involves the use of α.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right], \tag{3}$$

α modulates how quickly the algorithm learns by calibrating how big or small the state-action function $Q(S, A)$ update actually is, allowing the rate by which the algorithm learns the optimal policy to be controlled.

B. Sarsa (on-policy TD control) for estimating $Q \approx q_*$ [2]

Algorithm 2: Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Parameters : step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
repeat
  Initialize S;
  Choose A from S using policy derived from Q (e.g., ε-greedy);
  repeat
    Take action A, observe R, S';
    Choose A' from S' using policy derived from Q (e.g., ε-greedy);
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$;
    $S \leftarrow S'$;
    $A \leftarrow A'$;
  until *for each step of episode*;
  until S is terminal
until *for each episode*;

The third and final algorithm that was implemented is the Q-Learning algorithm. This algorithm shares many similarities with the SARSA algorithm, notably the incremental learning process, as well as the inclusion of ε and α. However, while SARSA updates the state-action function $Q(S_t, A_t)$ according to the action already taken, Q-Learning updates the state-action function $Q(S_t, A_t)$ according to the best possible action. This results in a greedier policy optimization process whilst remaining somewhat exploratory.

C. Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$ [2]

Algorithm 3: Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Parameters : step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
repeat
  Initialize S;
  repeat
    Choose A from S using policy derived from Q (e.g., ε-greedy);
    Take action A, observe R, S';
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$;
    $S \leftarrow S'$;
  until *for each step of episode*;
  until S is terminal
until *for each episode*;

IV. RESULTS AND ANALYSIS

A. Non-Slippery Environment

With the environment set to have deterministic transitions, the resulting evaluation return plots were erratic 5a, particularly from the SARSA algorithm. One reason for this may be due to the previously mentioned deterministic transition functions, which make it harder to discover the goal while exploring as it would require the agent to make the right decision for multiple states. However, once the goal is discovered, the policy improves rapidly.

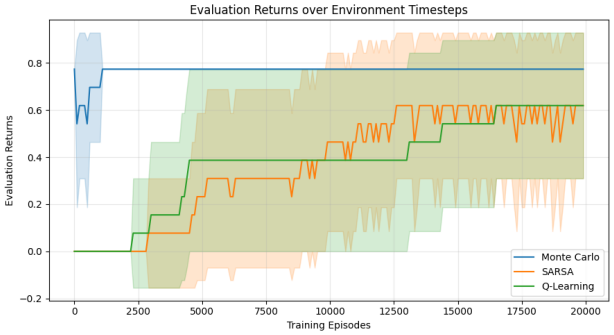
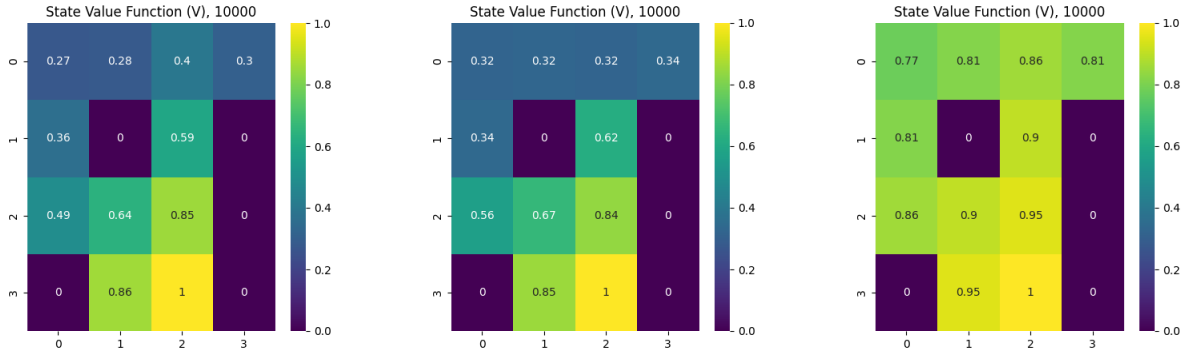


Fig. 1 Description of the image.

The value functions of the trained agents can be seen below 5b. Note the value function of the Q-Learning trained agent, which shows an improved value function compared to the other agents.

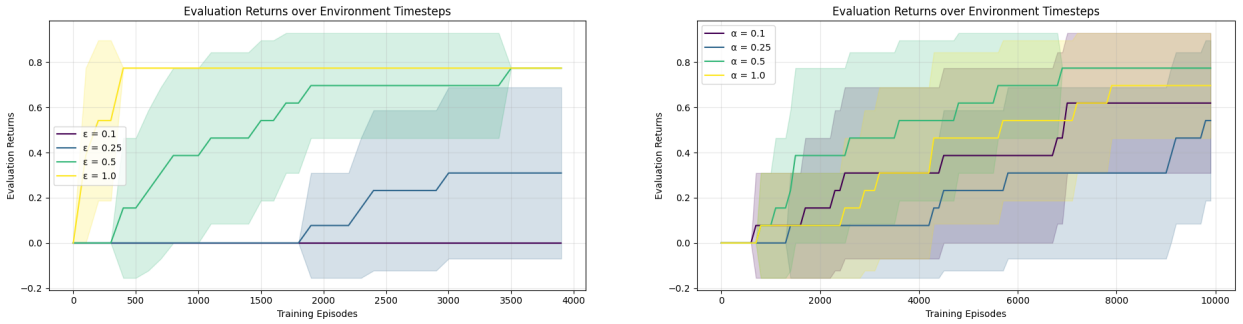


(a) Monte Carlo Value Function **(b) SARSA Value Function** **(c) Q-Learning Value Function**

Fig. 2 Value Function Plots of the Trained Agents after 10000 episodes

Examination of the plot showing evaluation returns with varying ε 3a confirms the previous statement regarding the difficulty of exploration in a deterministic environment. One would expect to see the rate of learning decrease as ε increases due to the ε-soft policy being further away from the ε-greedy policy [3]. However, because of the difficulty of finding the reward in this environment, the increased exploration resulted in the agent finding the reward state faster, ultimately speeding up the rate of learning.

The plot showing evaluation returns with varying α shows a lesser effect, with none of the evaluation return plots showing a significant advantage over another. This could also be a result of the deterministic environment, where by the effect of alpha on the learning rate is muted due to the Q-Learning algorithm being able to easily determine the optimal action.



(a) Varying ϵ **(b) Varying α**

Fig. 3 Q-Learning performance comparison across exploration and learning rate parameters.

B. Slippery Environment

The slippery environment results in a much different evaluation return plot 4, with the Monte Carlo algorithm under-performing in contrast to the deterministic environment, and the Q-Learning algorithm performing the best. The plots themselves are less straight than in the non-slippery environment, reflecting the stochastic transition function's nature.

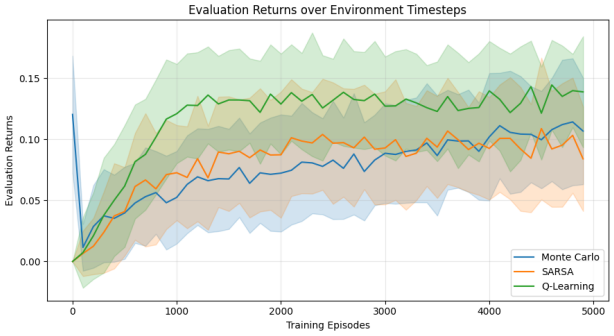
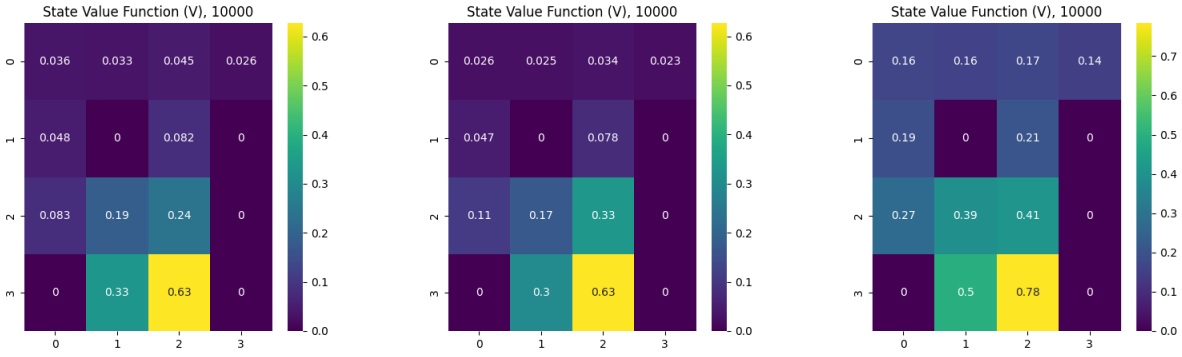


Fig. 4 Description of the image.

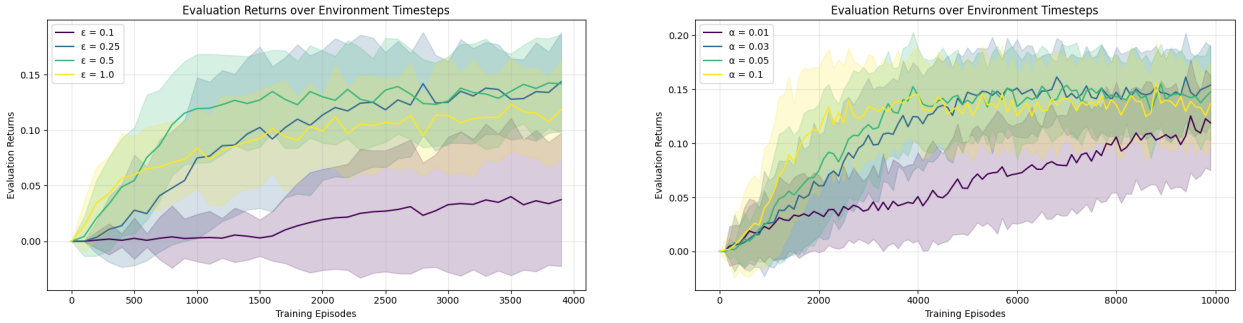
Examination of the state value functions shows a similar outcome to the deterministic environment, with the agent trained by the Q-Learning algorithm having an improved value function.



(a) Monte Carlo Value Function **(b) SARSA Value Function** **(c) Q-Learning Value Function**

Fig. 5 Value Function Plots of the Trained Agents after 10000 episodes in Slippery Environment

The impact of the variation of ε 6ais more in line with what is normally observed, as the optimal ε is 0.5, with higher ε resulting in a slower learning curve. The impact of α is greater in the stochastic environment 6b, as the optimal action is less clear in this environment, making the rate at which the Q-Learning algorithm chooses those actions more important.



(a) Varying ϵ **(b) Varying α**

Fig. 6 Q-Learning performance in slippery Frozen Lake environment.

V. Conclusion

This paper's aim was to compare model-free reinforcement learning algorithms in the Frozen Lake environment with both non-slippery and slippery environments. The Monte Carl with ε-soft policy algorithm, SARSA algorithm, and Q-Learning algorithm were implemented in this environment, and their trained agent's value functions were compared. The Monte Carlo algorithm had the highest evaluation return in the deterministic environment, while the Q-Learning algorithm had the highest in the stochastic environment. However, the Q-Learning had the best policy in both environments. Furthermore, the evaluation returns of the Q-Learning algorithm with varying ε and α were observed, showing their differing effects in both deterministic and stochastic environments. Notably, the deterministic environment preferred maximum exploration due to the difficulty finding the reward, and resulted in α being less effective on controlling the learning rate. In contrast, the stochastic environment had an optimal ε of 0.5, and α had a greater effect in controlling the learning rate.

References

[1] Foundation, F., "FrozenLake Environment (v1) – Gymnasium," `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`, 2023. Part of the Gymnasium RL suite, accessed October 2025.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., Adaptive Computation and Machine Learning Series, MIT Press, Cambridge, MA, 2018.

[3] Albrecht, S. V., Christianos, F., and Schäfer, L., *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*, Cambridge University Press, Cambridge, U.K., 2024.