MC Control, SARSA, and Q-Learning for the Frozen Lake MDP

Ruben Hernandez
AE 598: Reinforcement Learning
University of Illinois Urbana-Champaign

I. Introduction

The goal of this project was to implement three model-free reinforcement learning algorithms in the Frozen Lake gymnasium environment: On-Policy first-visit MC Control, SARSA (on-policy TD(0)), and Q-learning (off-policy TD(0). These methods were implemented under both stochastic slippery dynamics (`is_slippery=True`), and deterministic dynamics (`is_slippery=False`) on a 4×4 map populated by 16 states defined in by rows "SFFF", "FHFH", "FFFH", "HFFG" (where "S" represents the start, "H" is a hole, and "G" is the goal). States of "H" and "G" were considered terminal in this Markov Decision Process (MDP). Across all methods we used epsilon-greedy policy extraction to enable a balance between exploration and exploitation during training. Performance for each method was analyzed using the average discounted evaluation return in relation to the total number of training time steps (environment steps) used. This allowed for comparisons of convergence speed and policy optimality between all three methods implemented for this MDP.

A. Markov Decision Process

The MDP for this problem was defined using the Frozen Lake gymnasium documentation [1] by its state-space, action-space, transition probability matrix, reward matrix, and discount factor. The state-space consisted of 16 discrete states represented by the integers 0-15. The action space consisted of four discrete actions, 0:left, 1:down, 2:right, 3:up. We implemented model-free methods using the gymnasium environment, so the transition probabilities were not used in the RL algorithms. Regardless, the dynamics were governed by two modes, (`is_slippery=True`) and (`is_slippery=False`). With slippery conditions enabled, the agent could slip in a direction perpendicular to its intended action direction with a likelihood of `(1 - success_rate) / 2`. Our success rate was set to 1/3, which meant the agent could slip either left or right of the intended action with a combined likelihood of 2/3 while it would perform the intended action with a likelihood of 1/3. The reward function was defined by rewards of (0, 0, 0, 1) for respective states (S, F, H, G). We also used a fixed discount factor, γ of 0.95.

II. Methods

Model-free methods are unique to model-based methods because you can't extract a policy, π, from the state-value function, $V(s)$, as this requires a lookahead. For model-free methods we obtain π directly from the state-action values $Q(s, a)$ using an ϵ-greedy policy update. The foundational algorithms for MC Control, SARSA, and Q-learning were obtained from Sutton and Barto [2].

A. ϵ-Greedy Policy

Creating an ϵ-greedy policy requires selecting the next action in an ϵ-greedy way. ϵ is a float ranging from 0 to 1 that represents the indicates the balance between exploration and exploitation during training. As seen in algorithm 1, if the randomly generated value (0 to 1) was less than epsilon, the next action would be any random action. If the value was greater than epsilon, the next action taken would be the greedy action represented by the highest state-action value for the given state. This method allows for the next policy to be build primarily greedy (for ϵ<0.5) while still exploring new states to allow for new, possibly better, paths to be found.

Algorithm 1 Epsilon-Greedy Action Selection

if rand$(0, 1) < \epsilon$ **then**
    $a \leftarrow$ random action
else
    $a \leftarrow \arg\max_a Q(s, a)$
end if
return a

B. Monte Carlo Control (On-Policy, First-Visit)

Monte Carlo Control is a sample based method which estimates $Q(s, a)$ using returns obtained from full episodes (episodic) [2]. This project required the use of on-policy, first-visit MC control, meaning that the agent updates the same policy it uses to act, and it only updates $Q(s, a)$ for each state-action pair on the first occurrence per episode.

Algorithm 2 Monte Carlo Control with ϵ-Greedy Policy

for each episode $1, \ldots, N$ **do**
    Initialize $G \leftarrow 0$, $visited \leftarrow \emptyset$
    Generate an episode $\{(s_t, a_t, r_t), (s_{t+1}, a_{t+1}, r_{t+1}), \ldots\}$ by following an ϵ-greedy policy w.r.t. Q
    for each (s_t, a_t, r_t) in the episode, traversed in reverse order **do**
        $G \leftarrow r_t + \gamma G$
        if $(s_t, a_t) \notin visited$ **then**
            $visited \leftarrow visited \cup \{(s_t, a_t)\}$
            $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
            $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \dfrac{1}{N(s_t, a_t)}(G - Q(s_t, a_t))$
        end if
    end for
end for
return Q

Algorithm 2 details the implementation of our MC Control algorithm. The algorithm iterates over the user specified number of episodes, following the same policy throughout an episode before improvement. After an episode is complete, we compute the recursive discounted return using Eq. 2 over the reversed path (s, a, r) from the episode. The standard discounted return is seen in Eq. 1. Visited state-action pairs are tracked so that only the first visit is used to update the state-action value. If it is a first visit, Q(s, a) is updated using an incremental mean over all episodes where that state-action pair was visited. The incremental sample mean is a great estimator because it is truly unbiased.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T \tag{1}$$

$$G_t = r_t + \gamma G_{t+1} \tag{2}$$

MC Control continues this process over many episodes until, ideally, the policy extracted from $Q(s, a)$ converges towards optimal. The downside to this method is that it requires a large amount of full episodes to be performed, with unsuccessful runs not yielding any useful data due to the scarcity of rewards in this MDP. Additional hyper-parameters in the code function were `Q_init` and `count_init` which were used to warm start the function during training and evaluation by resuming training from the last checkpoint.

C. SARSA (On-Policy TD(0))

SARSA is an on-policy, temporal-difference (TD) method that, unlike MC, updates $Q(s, a)$ during an episode using a one-step bootstrapping method [2]. This method updates $Q(s, a)$ using Eq. 3, where α is the learning rate and (s', a') are the next state and action given the current policy, π.

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[\, r + \gamma Q(s', a') - Q(s, a) \,\big] \tag{3}$$

Algorithm 3 SARSA (On-Policy TD(0))

for each episode $1, \ldots, N$ **do**
  Initialize state s and choose action a using ϵ-greedy policy w.r.t. Q
  for each step of the episode **do**
    Take action a, observe reward r and next state s'
    if s' is non-terminal **then**
      Choose next action a' using ϵ-greedy policy w.r.t. Q
      $Q(s,a) \leftarrow Q(s,a) + \alpha[\, r + \gamma Q(s',a') - Q(s,a)\,]$
    else
      $Q(s,a) \leftarrow Q(s,a) + \alpha[\, r - Q(s,a)\,]$
      break
    end if
    $s \leftarrow s'$, $a \leftarrow a'$
  end for
end for
return Q

Algorithm 3 details the implementation of SARSA (on-policy, TD(0)) used for this project. For each step per episode we take an ϵ-greedy action from Q and observe the resulting next state and reward. If s' is non-terminal, we choose the next ϵ-greedy action from $Q(s', a')$ and update $Q(s, a)$ towards the target, $r + \gamma Q(s', a')$ at a rate of α. If the state is terminal, there is no next state or action and the target reduces to r. Finally, (s, a) are set to (s', a') and continue for every step in N episodes. Again, `Q_init` was used in the code to warm start during training and evaluation.

Because SARSA is incremental, it can generally improve the policy more rapidly than MC Control. It also tends to have a lower variance to MC Control, and can be tuned using α, where a lower learning rate corresponds to slower policy improvement with less variance, and a higher α corresponds to faster improvement with higher variance. As a downside, SARSA is more biased than MC Control due to bootstrapping using its current value estimates. To truly converge to a single optimal policy using SARSA, both ϵ and α would need to decay over training time.

D. Q-Learning (off-policy TD(0))

Off-policy TD(0) Q-learning is very similar to the SARSA algorithm implemented in this project. In this project, Q-learning acts using an ϵ-greedy (exploratory) policy but updated toward the greedy target $r + \gamma * max_{a'} Q(s', a')$. Therefore, the agent learns about the greedy policy while collecting data from the exploratory behavior policy. This allows Q-learning to converge to the optimal greedy policy rapidly while performing sufficient exploration to evaluate alternate paths over a large number of episodes. The only major difference between the SARSA and Q-learning pseudo-code is seen in the update step 4.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[\, r + \gamma \max_{a'} Q(s',a') - Q(s,a)\,\big] \tag{4}$$

This updates the value of the state-action pair using the target greedy policy, regardless of the next action taken. Like with MC Control and SARSA, `Q_init` was used in the code to warm start throughout training and evaluation.

III. Results

In order to compare on-policy first-visit MC Control, on-policy TD(0) SARSA, and off-policy TD(0) Q-learning used in OpenAI Gymnasium's tabular Frozen Lake environment, we plotted learning curves using methods from Albrecht et al. [3]. Additionally, we plotted the trained policies for all three methods and state-value function heatmaps extracted by $V(s) = \max_a Q(s, a)$. The results were analyzed for both `is_slippery=True` and `is_slippery=False`. The learning curve plots were also tested with $\alpha = 0.025$ and $\alpha = 0.1$ to observe the effects of learning rate on evaluation returns for SARSA and Q-learning. All other plots were obtained using a constant $\alpha = 0.025$ and $\gamma = 0.95$. All methods used the same environment configuration, detailed in section I.

A. Learning Curves

The learning curves for each model-free RL algorithm were plotted by comparing evaluation returns at specified checkpoints to environment time steps. Evaluation returns were computed at 100 evenly spaced checkpoints from zero to 20,000 episodes. At each checkpoint, $\pi(s)$ was extracted from $Q(s,a)$ and evaluated over 500 episodes to obtain the average return, G, of that policy at the respective checkpoint. Environment time steps were obtained by using a counter that increased by 1 at every action step taken for every method during training. This counter was also extracted at each checkpoint. Finally, the average evaluation return versus the number of time steps were plotted to show the learning curves of different algorithms and α values.

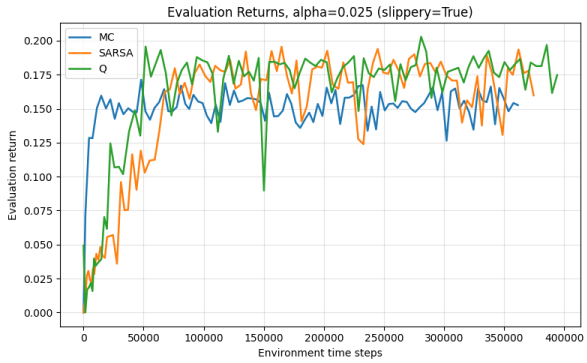


Fig. 1 Learning Curves - is_slippery=True, alpha=0.025.

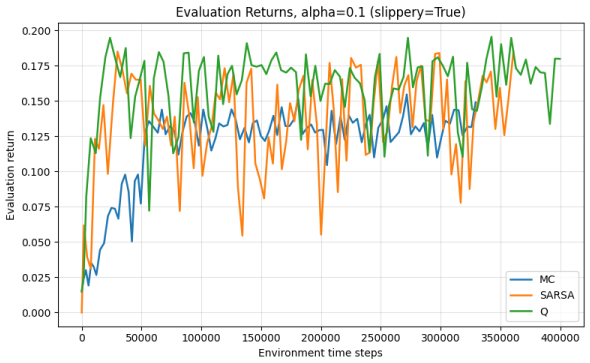


Fig. 2 Learning Curves - is_slippery=True, alpha=0.1.

When analyzing the evaluation curves in Fig. 1, 2, 3, we notice that the steady-state evaluation returns are much lower when `is_slippery=True` than `is_slippery=False`. This can be expected due to the discounted return. When there is no slip the goal can be consistently reached in 6 actions, while slippery conditions cause longer paths with unintended actions and recoveries. The increased number of steps to reach the goal causes the return to be discounted much more drastically in slippery conditions. Additionally, Fig. 3 shows that all methods converge to the optimal policy rapidly without stochastic slip conditions, but SARSA has a sharp dip to zero-return at one checkpoint. Because SARSA bootstraps from the executed ϵ-greedy a', some non-ideal exploratory steps around an unsafe state could change the greedy choice and cause consistent failures at that checkpoint.

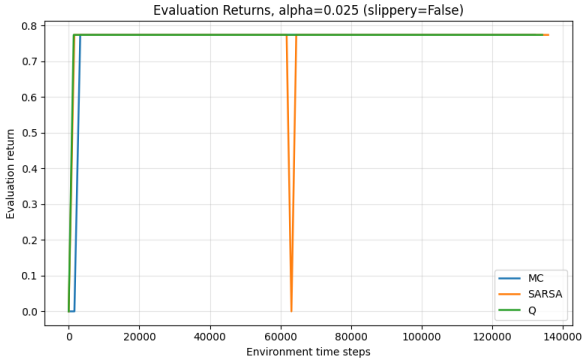


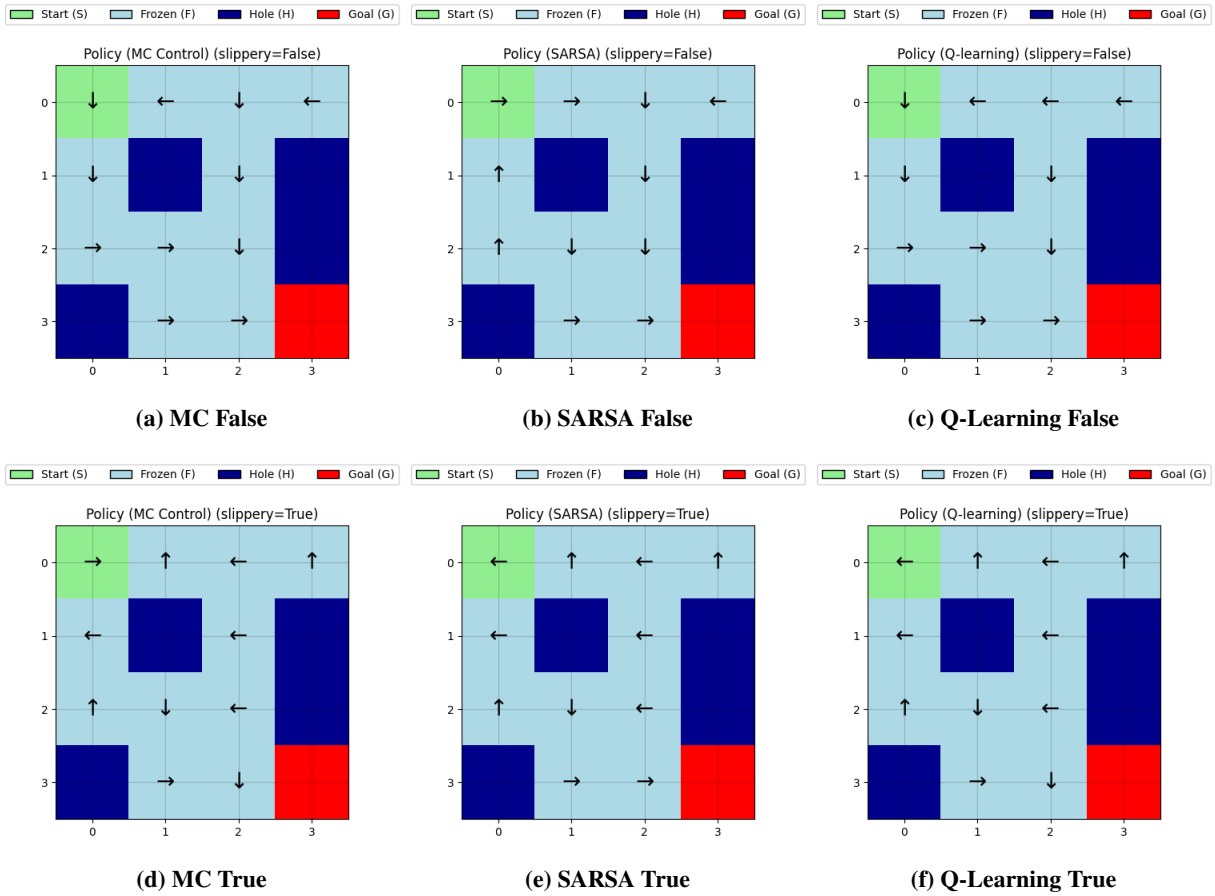
Fig. 3 Learning Curves - is_slippery=False, alpha=0.025.

The effects of learning rate, α, are clear to see with `is_slippery=True`. MC Control does not use α, so the comparison lies in SARSA and Q-learning. When α is lower (Fig. 1), SARSA and Q-learning both reach their steady-state in ~ 50.000 environment time steps and fluctuate $\sim \pm 0.025$ from that value. At a higher α, they reach steady-state in $\sim 25,000$ time steps but fluctuate $\sim \pm 0.05$ from that value. This demonstrates that a lower α results in slower improvement and lower variance, while a higher α results in faster improvement with higher variance.

Figures 1 and 2 show that MC Control averaged less steps per episode than SARSA and Q-learning, illustrated by the 20,000 episodes ending in less environment time steps than both other methods. MC Control also maintained a slightly lower steady-state than SARSA and Q-learning. MC Control is truly unbiased and episodic, so it only improves its policy after completing an episode and requires lots of data to build an accurate $Q(s,a)$ (law of large numbers). Additionally, it requires a non-zero episodic reward for the data to be useful in improving the policy. Due to these factors, MC Control likely had more early terminations which were not useful in improving its estimates, resulting in less steps per episode and a lower average return. Finally, Fig. 1, 2 show Q-learning learning slightly faster than

SARSA, matching expectations due to its off-policy use of a greedy target.

B. Trained Policies



(a) MC False **(b) SARSA False** **(c) Q-Learning False**

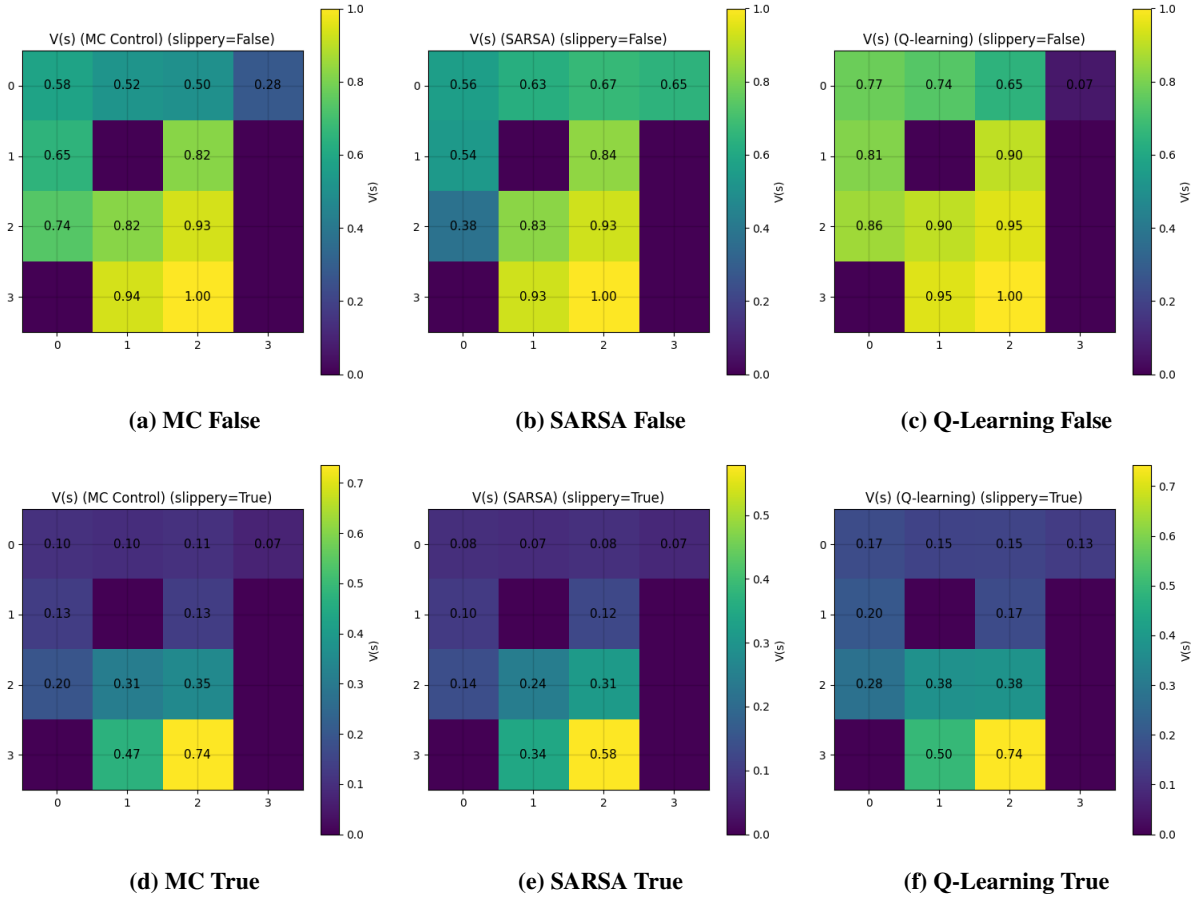
(d) MC True **(e) SARSA True** **(f) Q-Learning True**

Fig. 4 Comparison of trained policies for MC Control, SARSA, and Q-Learning under `is_slippery=False` and `is_slippery=True`.

For non-slippery conditions in Fig. 4, all of the trained policies resulted in one of the two possible 6-action paths. Although this is ideal, in other cases our use of a constant epsilon could cause certain random actions towards the end of training to shift the trained policy away from optimality. This could occur in both non-slippery and slippery conditions, and could be resolved by decaying ϵ to zero over the course of training.

Policies resulting from slippery conditions in Fig. 4 are all nearly identical. For the safest policy, state $(2, 1)$ should be the only state where it is possible for the agent to fall into a hole. Pointing the desired action into a hole at that state gives the agent a greater likelihood to slip into a safe state than it does to take the desired action into the hole. Aside from state $(1, 1)$ no other state for any method results in the agent slipping or stepping into an unsafe state. Overall, all three methods resulted optimal or near-optimal policies.

C. State-Value Functions



(a) MC False **(b) SARSA False** **(c) Q-Learning False**

(d) MC True **(e) SARSA True** **(f) Q-Learning True**

Fig. 5 State-value heatmaps for MC Control, SARSA, and Q-Learning under `is_slippery=False` and `is_slippery=True`. Each heatmap visualizes the estimated value function $V(s) = \max_a Q(s,a)$.

Across all methods and slippery conditions in Fig. 5, the heatmaps show a clear state-value gradient that increases towards the goal and drops with distance and proximity to holes. This confirms that all agents accurately learned the general trend and objective of the MDP. Across both slip-conditions, Q-learning performed the best, with values closest to the maximum expected discounted reward at its starting state. This is consistent with its off-policy greedy target resulting in a more direct push toward the optimal greedy policy while maintaining the same level of exploration as the other methods.

IV. Conclusion

The experiments performed using OpenAI Gymnasium's Frozen Lake environment showed that on-policy first-visit MC Control, on-policy TD(0) SARSA, and off-policy TD(0) all learn effective policies to successfully navigate the MDP. When `is_slippery=False`, each method reached the optimal 6-action solution, with returns near the expected maximum discounted return. When `is_slippery=True`, steady-state returns were lower due to longer trajectories with wasted steps due to slip, causing heavier discounting. Q-learning had the best performance, learning slightly faster and obtaining higher start-state values than SARSA, while MC Control had slightly less variance but learned more slowly. Additionally, the learning curves demonstrated the direct correlation between increasing α, increasing improvement speed, and increasing variation. The stability and convergence of all three methods could be improved in future experiments by decaying ϵ and α to force convergence to a single greedy policy.

References

[1] Gymnasium, "Frozen Lake - Gymnasium Documentation," https://gymnasium.farama.org/environments/toy_text/frozen_lake/, 2024.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An Introduction (2nd ed.)*, 2nd ed., The MIT Press, 2018.

[3] Albrecht, S. V., Christianos, F., and Schäfer, L., *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*, MIT Press, 2024. URL https://www.marl-book.com.