AE 598: HW 2 Tabular Methods

Alex Hausser

I. Introduction

This report explores the use of tabular methods to solve a Markov Decision Process (MDP). The algorithms discussed include Monte Carlo (MC) Control, SARSA, and Q-Learning. These are model-free, value-based methods that use tabular representations to estimate value functions or action-value functions.

The three methods are applied to the Frozen Lake environment in Gymnasium [1]. This report clearly defines the MDP and describes in detail the approaches used to solve it. Learning results are shown for each method and a comparison is made. The effect of changing the environment's `is_slippery` flag is also investigated to see how it changes the optimal policy and algorithm efficiency. This flag corresponds to changing the transition functions from deterministic to stochastic.

II. Methods

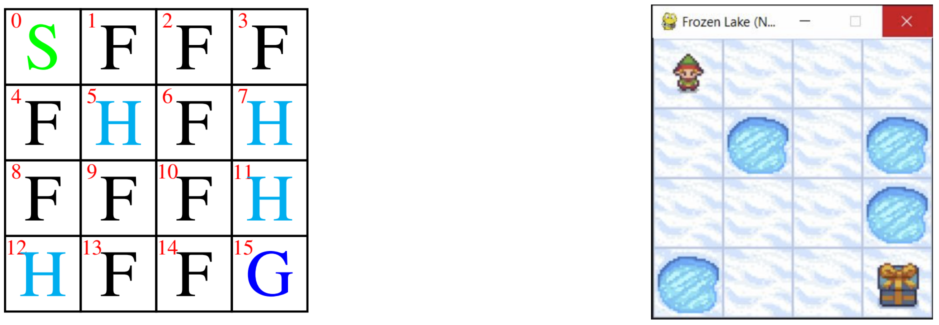
A. Frozen Lake MDP

The Frozen Lake MDP is defined and explained below in the following subsections. It contains the state space, initial state, action space, transition function, reward function, and discount factor.

1. State Space

The state space for the Frozen Lake environment is equal to the observation space. It consists of a four-by-four grid with each grid space corresponding to a state in the state space. Thus, there are 16 distinct states within the state space. The states are indexed starting at zero from the top-left corner, moving sequentially across each row. The numbering continues row by row until it reaches the the last state numbered 15.

There are four different tiles that each state represent. These include S for the start tile, F for the frozen tile, H for the hole tile, and G for the goal tile. The corresponding state space can be seen in Fig. 1a, and a visual of the rendered environment is seen in Fig. 1b.



(a) Frozen Lake grid layout showing the index of states (red), start tile (S), goal tile (G), frozen ice tiles (F), and hole tiles (H).

(b) Visual of the rendered Frozen Lake Environment.

Fig. 1 Visualizations of the Frozen Lake environment.

2. Action Space

The action space consists of four single integer values. The agent can choose from the four discrete actions in the action space at each timestep, with each integer corresponding to a specific movement direction for the player. The action space is defined below:

0: Move left **1**: Move down **2**: Move right **3**: Move up

3. Transition Function

The transition function for the Frozen Lake MDP is known, but this paper studies model-free methods; thus, the environment's dynamics and transition probabilities will be assumed to be unknown. Instead, the methods examined will learn directly from experience with the experience through samples of the states, actions, and rewards.

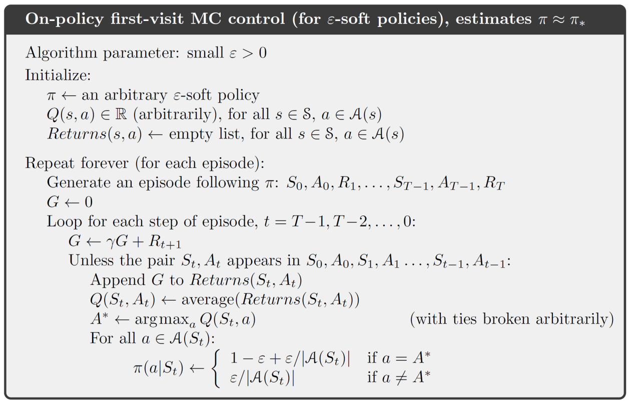
4. Reward Function

The reward schedule for each type of state tile is defined as zero, except the goal tile which has a reward of one. Similar to the transition function, prior knowledge of the reward function is not required for the algorithms described in this paper and learning of the reward scheme will be done through interaction with the environment. Additionally, the discount factor of future rewards is set to 0.95.

B. Tabular Method Algorithms

1. On-Policy First Visit Monte Carlo Control

The first method explored in this report is the on-policy first-visit Monte Carlo control algorithm, which assumes ε-soft policies. This approach is based on generalized policy iteration, where learning proceeds through the loop of policy evaluation followed by policy improvement. Unlike methods that depend on exploring starts, this technique ensures sufficient exploration by using ε-greedy policies. By balancing exploitation of known high-value actions with occasional exploration of suboptimal ones, the algorithm makes it possible to learn an optimal policy without requiring the exploration of initial states. For the algorithm, the ε hyperparameter was set to 0.35, as lower values resulted in insufficient exploration and learning. With this value, the algorithm achieved a reasonable level of exploration, enabling effective policy learning. The pseudocode for the algorithm is pictured in Fig. 2.



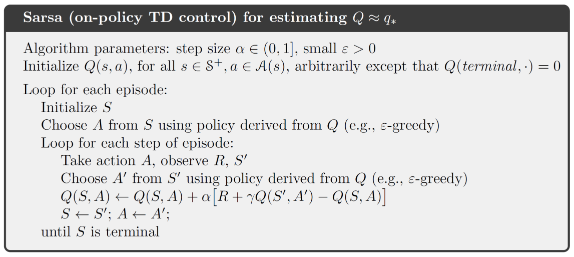
```
On-policy first-visit MC control (for $\varepsilon$-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

Fig. 2 On-Policy Monte Carlo Control Pseudocode [2].

2. SARSA

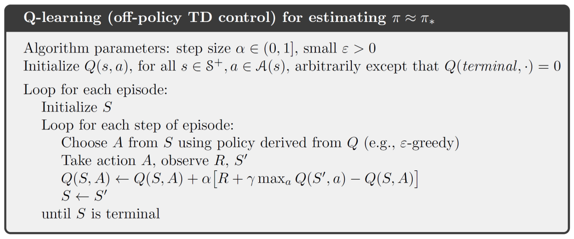
The second algorithm discussed in this report is SARSA, an on-policy TD(0) method. Like the Monte Carlo method described earlier, SARSA is based on generalized policy iteration, but it instead relies on temporal-difference (TD) learning. Unlike Monte Carlo methods, which use the full sampled return from an episode rollout, SARSA estimates the return by using a target that incorporates both the immediate reward and the action-value of the next state-action pair. This makes SARSA a bootstrapping method, where the current estimate is updated using information from a future estimate rather than a complete return.

Instead of learning state values as in some other approaches, SARSA directly learns action values and updates its knowledge by transitioning between state-action pairs. The learning rate of the algorithm is defined by the hyperparameter α which must follow Robbins-Monro convergence conditions to ensure convergence to the optimal policy. In practice, these conditions often lead to slow learning so a constant $\alpha = 0.1$ was chosen [3]. The pseudocode for SARSA is seen in Fig. 3.



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

Algorithm 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 $Q(terminal,\cdot) = 0$
Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal
```

Fig. 3 SARSA Pseudocode [2].



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

Algorithm 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 $Q(terminal,\cdot) = 0$
Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal
```

Fig. 4 Q-Learning Pseudocode [2].

3. Q-Learning

The third algorithm analyzed in this report is Q-Learning, an off-policy TD(0) method. Unlike on-policy methods, Q-Learning learns the value of the optimal policy by using experience generated from a separate behavior policy. While Q-Learning is quite similar to SARSA in structure, their update rules differ. In Q-Learning, the update targets the value of the best possible action in the next state, rather than the action actually taken, as is done in SARSA. This focus on the hypothetical optimal action makes Q-Learning fundamentally off-policy. The pseudocode for Q-Learning is shown in Fig. 4.

III. Results

A. Learning Curves

To evaluate the performance of the three RL algorithms outlined above, the learning curves of each were plotted. The learning curves are represented by the average discounted evaluation returns as explained in Albrecht et al. [3]. The x-axis corresponds to the number of learning steps which is defined as the cumulative number of learning updates across episodes. A comparison between the slippery and non-slippery environments for the three algorithms is plotted in Figs. 5 and 6 for training over 10,000 episodes.

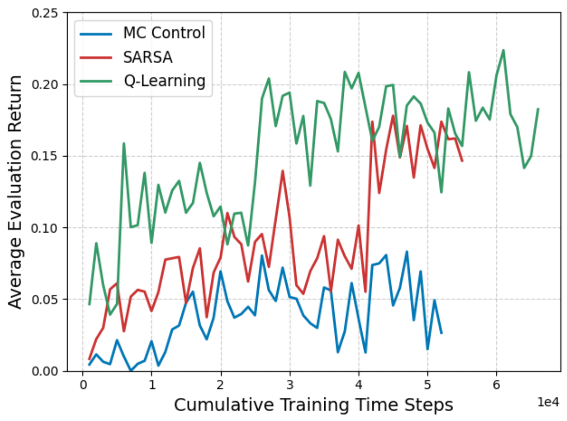


Fig. 5 Slippery learning curves.

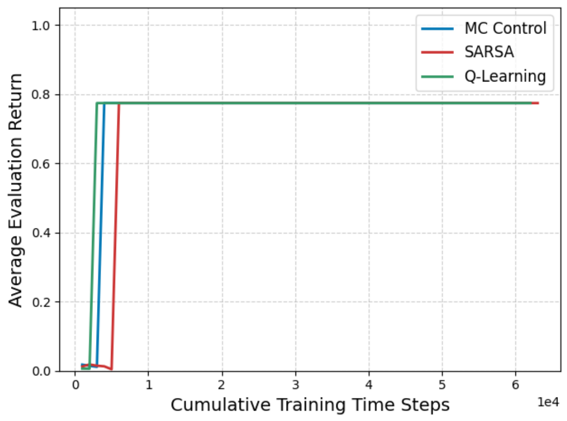


Fig. 6 Non-slippery learning curves.

It should be noted these learning graphs are only one possibility, and rerunning the algorithms will produce similar but not exact results. The stochasticity in the exploration and transition function for the slippery environment causes a wide variety of possible results, but one noticeable trend is the upward trajectory of each of the lines. Over the training steps, it is apparent that each algorithm did increase its average evaluation return. Hence, each algorithm did successfully trend toward a more optimal policy from the arbitrary initial policy.

The large downward fluctuations seen in Fig. 5 for each algorithm are attributed to the relatively large ε value of 0.35 which can abruptly reduce the Q-value due to the exploration of a new poor state-action pair. Because the ε-greedy behavior policy forces the agent to frequently explore, it regularly takes non-optimal actions that lead to holes with 0 overall reward. This frequent injection of poor data into the Q-table creates high variance in the average evaluation return. Despite this variance, Q-Learning exhibits the highest average performance throughout the training, demonstrating the robustness of its off-policy update in a highly stochastic environment. In contrast, MC Control shows the lowest overall performance, highlighting its disadvantage in high-variance environments where its reliance on full-episode returns amplifies stochastic noise compared to the TD methods.

The performance comparison changes dramatically when analyzing the results in the non-slippery (deterministic) environment shown in Fig. 6. Here, all three algorithms converge extremely quickly to the near-optimal return of approximately 0.78 within the first 10,000 cumulative training steps. This stability is expected because the deterministic nature of the environment removes all variability in state transitions, allowing the algorithms to quickly and confidently map out the single optimal path. While Q-Learning appears marginally faster initially, the primary takeaway is that in simple, deterministic environments, all three tabular control methods are able to achieve a stable, optimal policy with minimal training experience.

B. Optimal Policies

Once each algorithm was trained for 10,000 episodes, the greedy policies were stored and plotted on the grid world map. For each of the algorithms in the non-slippery environment, the deterministic optimal policy was found. This policy follows the quickest trajectory to the goal, and always avoids moving toward holes. Figure 7 plots the policy on the grid world map. The blue background of each tile corresponds to the corresponding Q-value from the Q-table where a darker color is a larger Q-value.

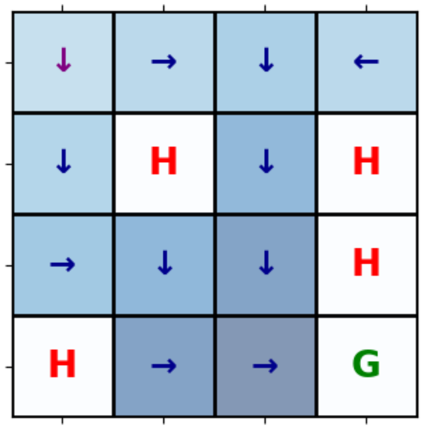


Fig. 7 Non-slippery environment optimal policy for all three algorithms.

Although the non-slippery environment produced the deterministic optimal policy for each algorithm, there were some differences in policy between algorithms for the slippery environment. As shown in Figs. 8-10, MC Control, SARSA Control, and Q-Learning Control each learned distinct patterns of optimal actions in the presence of stochastic transitions and ε greedy policies.

The MC Control policy map (Fig. 8) shows arrows mostly pointing toward the goal in states closer to the goal, reflecting mostly optimal decisions in those areas. However, near the initial states, the arrows sometimes point toward suboptimal directions, including multiple instances where arrows direct toward holes. This indicates that the learned policy is not perfect and still makes occasional errors. Similar patterns are observed in the SARSA and Q-Learning policies, where the top two rows of the grid exhibit more randomness and arrows occasionally point toward holes. This behavior is expected due to the sparse reward structure where the only reward is received at the goal state. As a result, states near the start are assigned lower expected returns because actions from those states face a higher risk of falling into holes and must take a longer path to the goal, making learning more challenging in these regions.

Even with the differences in policy arrows, each algorithm had almost exactly the same Q-table distribution as seen by the blue shading. The shaded regions indicate that all three algorithms agree on the value landscape within the environment. This convergence of Q-values suggests that, despite some policy variations, all methods are converging to similar value functions and ultimately consistent evaluations of the environment's state-action pairs.

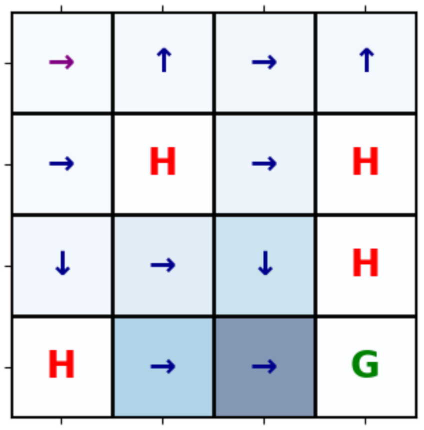


Fig. 8 Slippery environment optimal policy for MC Control.

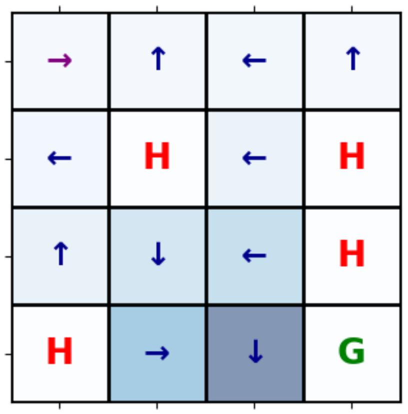


Fig. 9 Slippery environment optimal policy for SARSA Control.

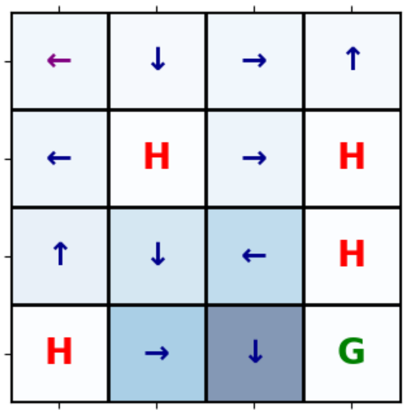


Fig. 10 Slippery environment optimal policy for Q-Learning Control.

IV. Conclusion

In this study, three tabular reinforcement learning algorithms were implemented and compared in the Frozen Lake environment. The primary result of this study is that environment stochasticity is the dominant factor affecting algorithm performance. In the deterministic (non-slippery) case, all three algorithms rapidly converged to a near-optimal average return of approximately 0.78 and produced the identical optimal greedy policy. This demonstrates that in low-variance problems, the choice of algorithm among these three methods has little impact on the final policy obtained.

In the stochastic (slippery) environment, the methods showed divergence in performance. Q-Learning produced the highest average return, while Monte Carlo Control produced the lowest and was the most inconsistent due to using full episode returns during learning, which amplified the stochastic variation. Despite these differences, the shaded Q-tables indicated that all three methods converged to a similar underlying value distribution, demonstrating that each algorithm trended successfully toward the optimal policy even under stochasticity.

References

[1] Foundation, F., "FrozenLake Environment Documentation," `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`, 2025. Accessed: Sep. 18, 2025.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2$^{\text{nd}}$ ed., MIT Press, Cambridge, MA, 2018.

[3] Albrecht, S. V., Christianos, F., and Schafer, L., *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*, MIT Press, Cambridge, MA, 2024.