Comparison of Model-Free Reinforcement Algorithms in Frozen Lake Environment

Shishir Bhatta*
University of Illinois Urbana-Champaign, Champaign, IL, 61820

This report explores three "model-free" reinforcement learning algorithms meant to solve Markov Decision Processes (MDPs): on-policy first-visit MC control, SARSA, and Q-learning. Model-free algorithms do not require the transition probabilities associated with the environment, thus making them reliable in scenarios in which the model is unknown. The three algorithms take different approaches to tackling the issue, either being on-policy or off-policy. The algorithms are trained on the Frozen Lake algorithm, which is a 4x4 grid in which the agent must navigate around holes to reach the destination. Each method learns an optimal action-value function $Q(s, a)$ and corresponding policy. The performance is evaluated by plotting the evaluation return and the resulting policies and value functions to demonstrate their convergence characteristics and various trade-offs. Q-learning and SARSA seem to be very similar in their behavior, perhaps due to their use of similar ϵ-decat policies, while MC Control operates in a very different manner.

I. Nomenclature

γ	=	discount factor
ϵ	=	exploration rate
π	=	policy
π^*	=	optimal policy
A	=	action space
a	=	action
F	=	frozen ice state
G	=	goal state
H	=	hole state
i	=	iteration index
k	=	policy iteration index
n	=	number of states or iterations
R	=	reward function
r	=	reward
S	=	state space
s	=	state
s'	=	next state
t	=	time step
$Q(s, a)$	=	action-value function
$Q^\pi(s)$	=	action-value function under policy π
Q^*	=	optimal action-value function

II. Introduction

This report compares three methods of model-free reinforcement learning to solve Markov Decision Problems (MDPs). Model-free means that the algorithm learns from experience rather than relying on a model of how the environment works. A model would typically consist of transition dynamics and a reward functions, so model-based

*Undergraduate Student, Aerospace Engineering

algorithms such as policy improvement use the model to their advantage, whereas model-free algorithms attempt to learn a value function Q and policy π from direct trial-and-error.

The three algorithms in question are on-policy first-visit MC control, SARSA, and Q-learning, each of which has its own pros and cons. For starters, MC control and SARSA are on-policy, while Q-learning is off-policy. On-policy algorithms improve and learn about the same policy that they're currently using to interact with the environment, whereas off-policy algorithms can use the experiences of another policy (target policy) while following a separate policy (behavior policy). All methods are reliable ways to solve MDPs in model-free settings, but have their own drawbacks.

For this report, the algorithms are tested in Gymnasium's Frozen Lake, which is a 4x4 grid that consists of regular spaces and holes. One cell is the destination, so the agent must successfully navigate through the grid without falling in any holes. If the agent falls in a hole or reaches the destination, the game will end, so the agents should be able to converge on an optimal policy and action-value function to reach the end. In addition, there is a slippery and non-slippery environment, in which the slippery environment doesn't guarantee the direction the agent will move in.

III. Problem Definition

The environment that the agents will interact with can be defined as an MDP. To show this, the state space, action space, reward function and transition problems should be defined.

A. State Space

The 4x4 "frozen lake" that the agents will work is as follows:

$$\begin{bmatrix} F_0 & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & G \end{bmatrix} \tag{1}$$

where F_0 is the initial starting cell. The state space can be represented as a 16-length vector, with each index in the vector representing a state in the frozen lake. The vector will be the result of the "flattened" operation on a matrix, so:

$$S = \begin{bmatrix} 0 & 1 & \dots & 15 \end{bmatrix} \tag{2}$$

where each index represents a state, enumerating from the top left, to the right, then down, so on, and so forth.

B. Action Space

The action space is the four directional movements the agent can take. The agent cna move up, down, left or right at any given cell as long as there is a cell for the agent to occupy in that direction. If the cell is located on an edge, it can not move in that direction and nothing happens, but it is still a valid action in any cell. Thus, the action space can be represented as such:

$$A = \begin{bmatrix} \text{up= } 0 & \text{down= } 1 & \text{left= } 2 & \text{right= } 3 \end{bmatrix} \tag{3}$$

C. Reward Function and Discount Factor

The reward function is fairly simple in this problem, as there are no partial rewards. It can defined as:

$$R(s, a, s') = \begin{cases} 1 & s = G \\ 0 & s \neq G \end{cases} \tag{4}$$

The discount factor reduces the reward of future states, providing a higher weight for short-term rewards. In this problem, the factor will be as such:

$$\gamma = 0.95 \tag{5}$$

IV. Implementation

All code was developed in a Jupyter Notebook using Python. The algorithms were implemented by taking advantage of the NumPy and Gymnasium libraries. The environment is created using the Gymnasium library, so the environment can be unwrapped to access the observation and action space. The environment can also be made to be slippery or not, allowing for testing in both scenarios. The SARSA algorithm is an on-policy temporal difference (TD) control algorithm, and Q-learning is an off-policy TD control algorithm. All the algorithms implement a decaying ϵ policy to ensure convergence through the greedy in-the-limit with infinite exploration (GLIE) theorem.

- $0 \leq \epsilon \leq 1$ - the exploration rate, which begins at 1 during the first episode and decreases to 0 at the last episode.
- γ - the discount factor, as described earlier.
- α - the learning rate, implemented in the on-policy algorithms, which controls how much the parameters are updated based on new information.
- Number of episodes - an episode is a complete sequence of interactions with the environment, from to terminal state. The algorithms should all be trained on an equal amount of episodes.
- Environment - The environment in which the agent acts in, either slippery or not.

A. On-Policy First-Visit MC Control

This algorithm follows an ϵ-soft algorithm, which assures continual exploration since every action has a probability $\geq \epsilon|A|$. It also only uses the first-visit returns, which means it only counts the first occurrence of each state-action pair in an episode. It is also on-policy, which means it learns from and improves from the same policy.

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

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-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 π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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 $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a Q(S_t, a)$ (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. 1 On-Policy MC Control Algorithm (from Barton-Sutton)

B. SARSA (On-Policy TD(0)

This algorithm learns the action-value function $Q(s,a)$, learning and improving from the same policy. The TD(0) indicates that it uses a one-step temporal difference, updating immediately after a step instead of waiting for ht episode to end. The key difference in this algorithm is Q is updated using the actual next action a' instead of the optimal action. There is also some ϵ-decay to ensure the algorithm converges to the $Q*$.

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., ε-greedy)

Loop for each step of episode:

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ε-greedy)

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$

$S \leftarrow S'$; $A \leftarrow A'$;

until S is terminal

Fig. 2 SARSA Algorithm (from Barton-Sutton)

C. Q-Learning (Off-Policy TD(0))

This algorithm is off-policy unlike the other algorithms. It learns about an optimal ϵ-greedy policy while following a ϵ-soft algorithm, which can be more exploratory.

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., 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 S is terminal

Fig. 3 Q-Learning Algorithm (from Barton-Sutton)

V. Results

The algorithms were run for 1000 episodes in both slippery and non-slippery environments. Both SARSA and Q-learning implemented an ϵ-decay algorithm to ensure convergence, which helped improve performance.

A. Evaluation Returns

One of the ways to measure the effectiveness of an algorithm during its training is by tracking the evaluation return. The evaluation return tracks the mean returns from the algorithm with its current Q and π until the episode is complete. This gives an idea of how the algorithm is improving over time. The evaluation returns were tracked in both slippery and non-slippery environments.

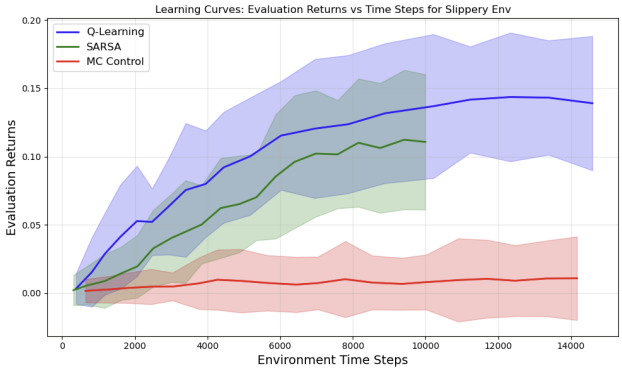

Fig. 4 Evaluation Returns in the Slippery Environment

From Figure 4 and 5, the general trend seems to be that the algorithms steadily return higher rewards as they iterate through time steps. The reason each algorithm ran for a different number of timesteps is that the number of steps in each episode varies based on the actions taken by the agent. Some episodes may end sooner than others if they reach a terminal state in fewer steps. Even so, Q-learning and SARSA seem to perform better than MC Control, and this may be due to the ϵ-decay policy. As the number of times each state-action pair is visited, the ϵ decreases, which means the agents select more exploitative actions over exploratory. The ϵ-decay for SARSA and Q-learning chooses to decay the ϵ after every episode, which resulted in better results. The same can be said for both non-slippery and slippery environments; however, Q-learning and SARSA seemed to have converged very fast, hence the few number of time steps taken. No policy seemed to have converged in the case of the slippery environment. Overall, Q-learning seemed to have the highest evaluation returns of the three algorithms.

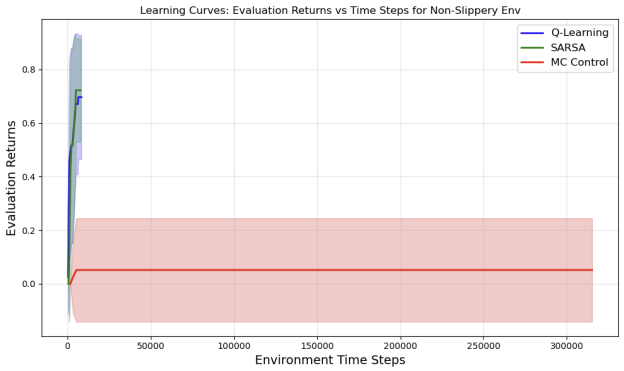

Fig. 5 Evaluation Returns in the Non-Slippery Environment

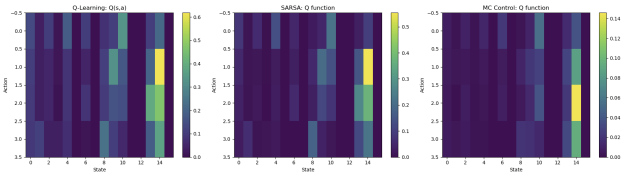

Fig. 6 Heat Map of the State-Action Value Function (for Slippery)

B. State-Action Value Functions

All the algorithms cam eup with similar policies. Figures 6 and 7 show the Q, where the horizontal values are the various states and the vertical values are the actions. The lighter values indicate the greater values for those actions at each state. As expected, the goal state G has values of 0 and so do the other H states, which is important. This means the algorithms learned where the holes are and found no value in them. States closer to G have more yellow i.e. 14 and 9, since they are simply one move away from reaching the goal and receiving a reward. Although all the algorithms didn't converge to a single $Q*$ shared among themselves, they came close to reaching the same policy. The slippery environments has lower values for most actions at each state, since it is possible to move in undesired directions, but the non-slippery case has a lot more yellow and green, indicating extremely high values for certain state-action pairs.

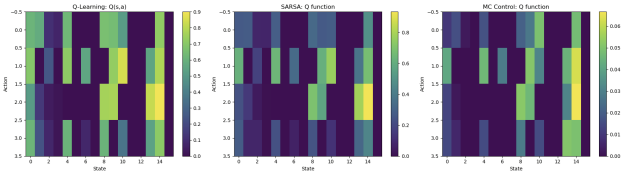

Fig. 7 Heat Map of the State-Action Value Function (for Non-Slippery)

C. Policies

The policies for each of the algorithms are plotted in figures 8 and 9, where the optimal action is plotted on the vertical axis for each state. For most scenarios it seemed like the optimal policy was moving left (value is 0), but the algorithms seem to have somewhat similar policies. It is very clear for state 8, the agents believe moving right is the most optimal move in the slippery environment, which doesn't necessarily match with the non-slippery environment. In the non-slippery environment, the Q-learning and SARSA algorithms have almost exactly the same policies, which might mean they were very tightly correlated, just like in the evaluation returns. They also share more similarities in the slippery environment, whereas the MC Control seems to be operating differently, focusing on exploiting certain actions in specific states.

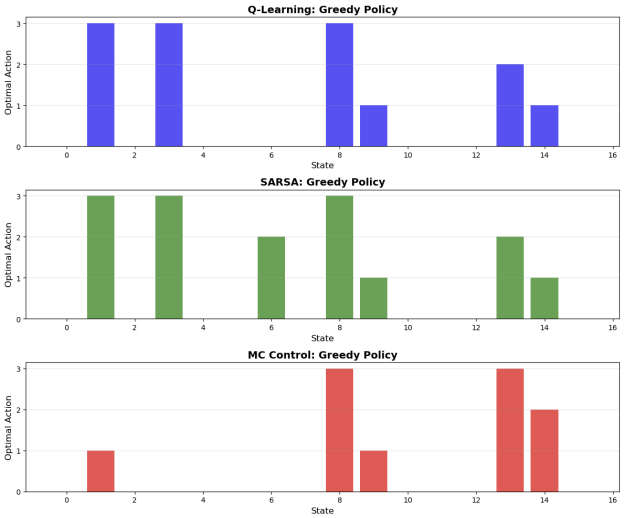

Fig. 8 Policy for Slippery Environment

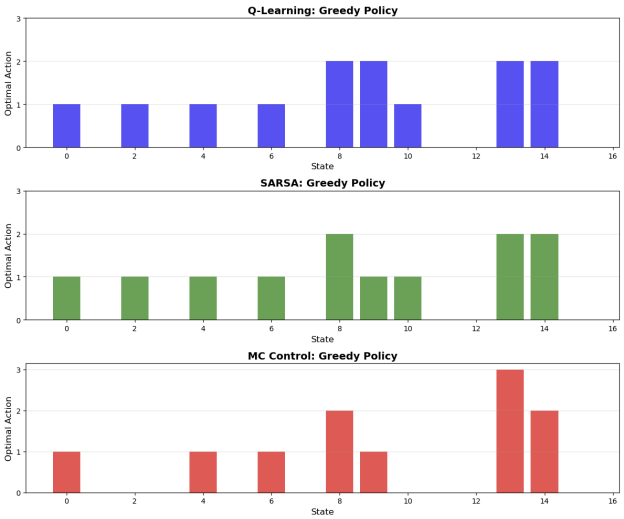

Fig. 9 Policy for Non-Slippery Environment

VI. Conclusion

Three model-free reinforcement algorithms were implemented in the Froze Lake gymnasium in both slipper and non-slippery modes. The algorithms were MC Control, SARSA and Q-learning, with the latter 2 employing an ϵ-decay policy to tradeoff between exploitation and exploration. The MC-Control used a ϵ-soft policy, which may have resulted in it not being as similar as the other policies in its results. Q-learning seemed to be the most effective in gaining higher evaluation returns, but SARSA was very close.