Sakshi Singh [CID : 01925165]

25 October 2023

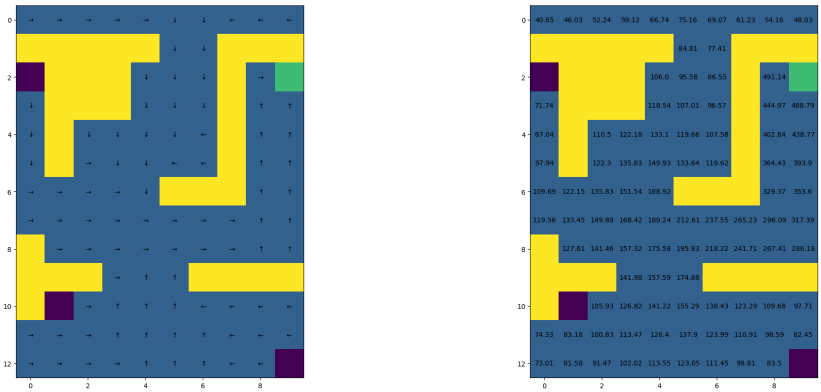
Reinforcement Learning : Coursework 1

1 Question 1 : Dynamic Programming

1.1 Question 1.1

I chose the Value Iteration algorithm to solve the maze problem. Unlike policy iteration, value iteration obviates the need for policy evaluation and policy improvement steps, making it inherently more computationally efficient. To prove this, I tested both my value iteration code and policy iteration code for a series of 15 runs and obtained the average time to convergence. Value iteration took 0.45 seconds while policy iteration took 1.57 seconds. In terms of design, I performed synchronous backups, where all state values are updated in unison based on the previous iteration's values. While asynchronous backups can converge quicker, for this problem where I had full knowledge of the state space and the state space being small allowed for synchronous method to be more structured and consistent.
I opted for a fixed threshold of 0.0001 in the algorithm to balance computational efficiency with precision. A smaller threshold might increase accuracy but require more computational time, while a larger one could fasten convergence at the expense of precision. I also assumed that the environment provides deterministic transition probabilities and rewards.

1.2 Question 1.2



(a) Optimal Policy (b) Optimal Value Function

Figure 1: Optimal Policy and Optimal Value Function for Value Iteration Algorithm

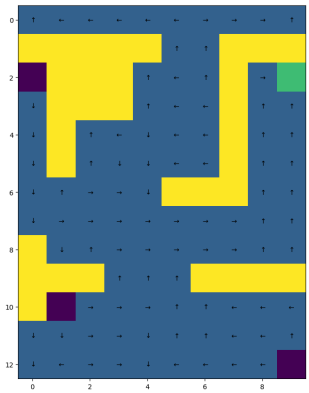
Upon executing the Value Iteration method, I derived an optimal policy for navigating the maze. As seen in Figure 1 (a), this policy graphically showcases the best action (e.g., move up, down, left, right) to take at each state.
Figure 1 (b) illustrates the optimal value function. High values signify states that, when reached, lead to more efficient pathways to the goal, while lower values represent states further from the optimal path or closer to obstacles.

1.3 Question 1.3

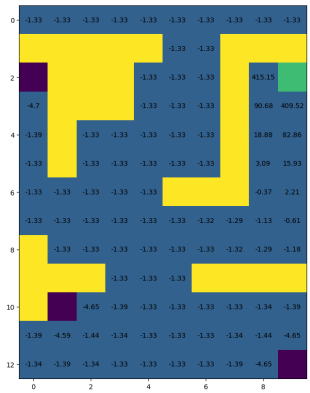
Influence of γ (Discount Factor) on Optimal Value Function and Policy:

- For $\gamma < 0.5$:
  When the discount factor is less than 0.5 ($\gamma = 0.25$), the agent becomes myopic in its decision-making. It places higher importance on immediate rewards rather than future potential rewards.The optimal value function, in this scenario, will reflect lower expected returns from states since it heavily discounts future rewards.
  From Figure 2 (a), we notice that states that are far away from the goal have similar values of -1.33 and this value increases as we get closer to the goal state. Figure 2 (b) shows that the agent finds it difficult to choose the action that moves it towards the goal state when its farther away from it.

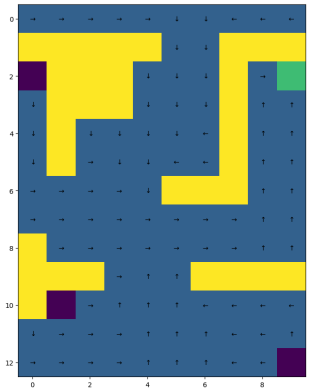
- For $\gamma > 0.5$:
  With the discount factor greater than 0.5 ($\gamma = 0.75$), the agent becomes more farsighted. It considers future rewards with greater weightage. Consequently, the optimal policy derived tends to be more strategic, looking several steps ahead and planning for long-term gains. The optimal value function will generally indicate higher expected returns, especially for states that might lead to more substantial future rewards.
  From Figure 2 (c), we notice that we can find the optimal policy even from states that are further away from the goal state. Figure 2 (d) shows the value function as higher values even for states further away from the goal as we are not discounting them much.



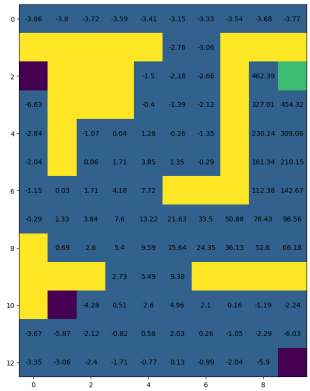
(a) $\gamma = 0.25$, p= 0.86



(b) $\gamma = 0.25$, p= 0.86



(c) $\gamma = 0.75$, p= 0.86



(d) $\gamma = 0.75$, p= 0.86

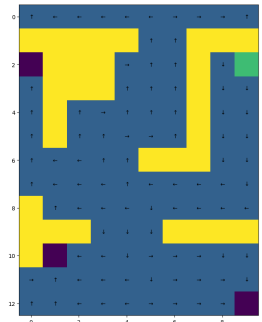
Figure 2: Optimal Policy and Optimal Value Function for different values of γ

Influence of p (Probability) on Optimal Value Function and Policy:

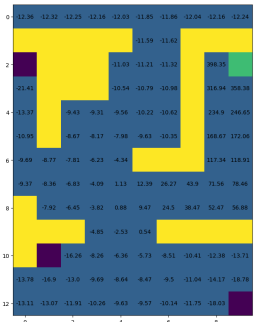
- For $p < 0.25$:
  When p is less than 0.25 ($p = 0.1$), it represents a more uncertain environment where the intended action doesn't always result in the expected outcome. The agent might avoid paths near obstacles even if they seem shorter, due to the high risk of unintended actions. As a result the agent has greater probability of moving in directions that it does not intend to and so the optimal policy is seen to move away from the goal as seen in Figure 3 (a). The optimal value function shown in Figure 3 (b) reflects this uncertainty, with states near obstacles having significantly lower values.

- For $p = 0.25$:
  When p is equal to 0.25, there's an equal chance of moving in any direction regardless of the intended action and as a result the agent has difficulties moving towards the goal as seen in Figure 3 (c). The optimal value function as seen in Figure 3 (d) suggests that states equidistant from the goal have similar values, given the equal unpredictability of any action.

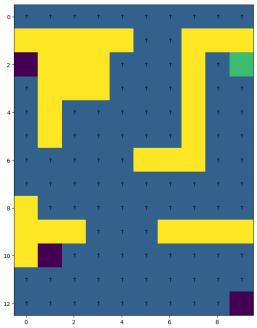
- For $p > 0.25$:
  When p is greater than 0.25 ($p = 0.8$), the environment becomes more deterministic. As such, the agent can confidently choose actions that lead directly to states with higher rewards. The optimal policy in this scenario becomes more stable and direct as seen in Figure 3 (e). The value function of a state becomes more predictable since we have a higher confidence that taking an action will result in the expected next state. This can lead to higher values especially for states that have direct paths to rewards as seen in Figure 3 (f).



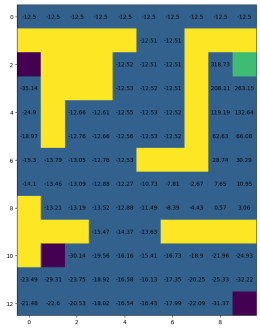
(a) $\gamma = 0.92$, p= 0.1



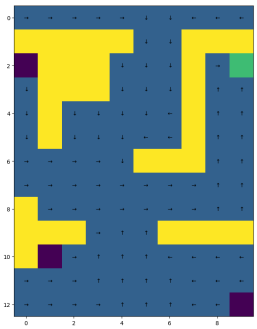
(b) $\gamma = 0.92$, p= 0.1



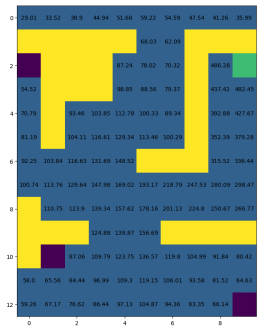
(c) $\gamma = 0.92$, p= 0.25



(d) $\gamma = 0.92$, p= 0.25



(e) $\gamma = 0.92$, p= 0.8



(f) $\gamma = 0.92$, p= 0.8

Figure 3: Optimal Policy and Optimal Value Function for different values of p

2 Question 2 : Monte-Carlo Reinforcement Learning

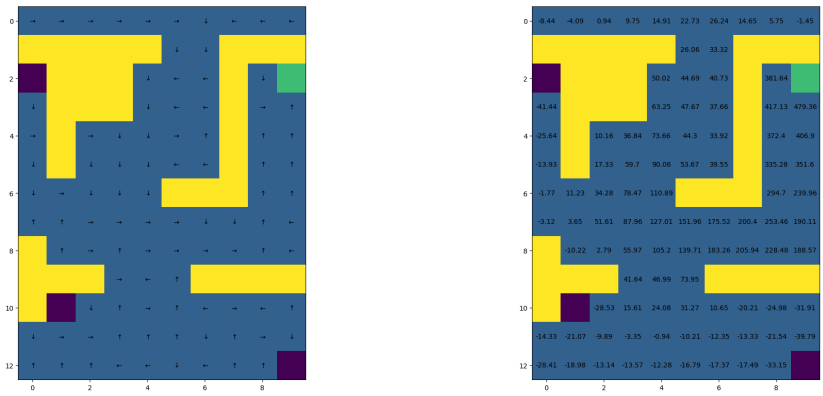
2.1 Question 2.1

To solve the maze problem, I used the Monte-Carlo Every Visit method using an on-policy approach. Every-Visit MC accounts for all occurrences of a state-action pair in an episode, capturing all possible experiences an agent undergoes. This approach, in contrast to only considering the first occurrence of a state-action pair, ensures that all rewards following a state-action pair are utilized to calculate the average return, providing a more comprehensive evaluation. An on-policy method was employed which allowed the agent to learn and update its policy based on the experiences it gathers while following that same policy. On-policy approach also avoids introducing high variance in learning updates unlike the off-policy. The on-policy approach also benefits from the online updates which optimize the current policy based on the latest experiences, allowing the agent to adjust and improve its strategy in real-time.

To balance exploration and exploitation, an ϵ-greedy policy with an initial ϵ value of 0.4 was implemented. This of 0.4 signifies that during the early stages of training, there is a 40% chance of choosing a random action irrespective of its known value. This is crucial because early on, the agent has limited knowledge about the environment, so a higher level of exploration ensures a broader

understanding and discovering more potential rewards. As the episodes progress, the value of ϵ decays, shifting the agent's strategy towards exploitation, where it can use its knowledge to maximize rewards. Moreover, I employed a decayed ϵ to ensure that *GLIE* is updated and as a result ensures that the algorithm converges. Overall, the ϵ value chosen provides a good balance between exploration and exploitation and a more stable convergence in 5000 episodes chosen than lower ϵ values. I also implicitly assumed that the environment's dynamics remain stationary, implying consistent transition probabilities and reward structures. Therefore, instead of a fixed learning rate α, $\alpha = 1 \div count(s,a)$. This allows $V(a)$ and $Q(s,a)$ to converge by law of large numbers. By choosing 5000 episodes, I aimed to strike a balance between computational efficiency and thorough learning.

2.2 Question 2.2



(a) Optimal Policy (b) Optimal Value Function

Figure 4: Optimal Policy and Optimal Value Function for Monte-Carlo Algorithm - Every Visit Method

As seen in Figure 4 (a), optimal policy obtained provides a strategic map for the agent, directing its movement in a way that likely maximizes the accumulated rewards while navigating the maze's obstacles. From Figure 4 (b), The optimal value function suggests a clear direction of movement for the agent to maximize its cumulative reward. States closer to the green square, tend to have higher values, indicating they are part of favorable paths.

2.3 Question 2.3

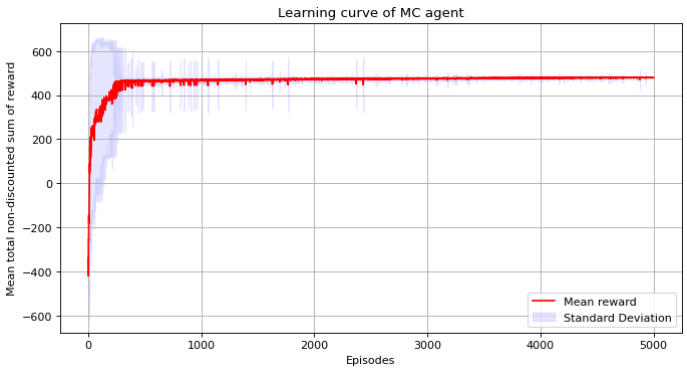


Figure 5: Mean non-discounted sum of rewards vs. Episodes plot of a MC-learning agent for 25 training runs.

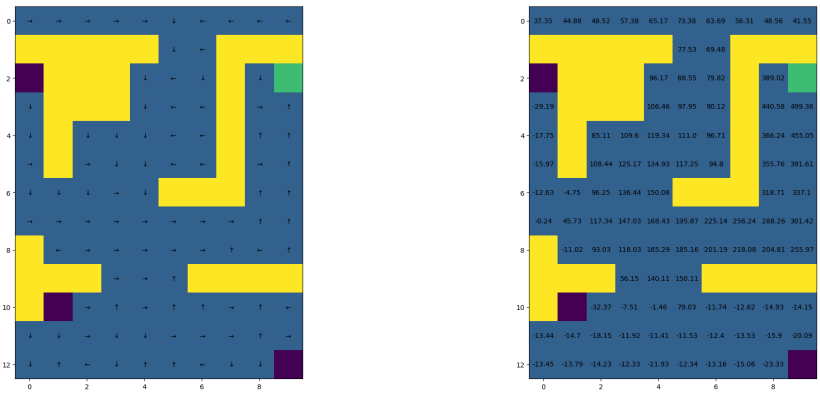
Figure 5 shows the learning curve of our MC agent for 25 training runs. From the Figure 5 we see that at the beginning of training, the agent is typically exploring the environment. The rewards might fluctuate greatly, leading to a large standard deviation. Over time, as the agent samples more returns and updates its value estimates, it starts to converge on a policy that maximizes its expected reward. Moreover, the variability (shown by standard deviation) of rewards reduce as the agent consistently employs its learned policy.

3 Question 3 : Temporal Difference Reinforcement Learning

3.1 Question 3.1

Initially, I tested both the SARSA and Q-learning methods to solve the maze problem as they both converge to optimal policy and value function. However, in the end I employed the SARSA method, an on-policy Temporal Difference algorithm to solve the maze problem. The main reason for choosing SARSA method was that it learns the value of the policy that it's currently executing, which in our case is the ϵ-greedy policy. By doing so, SARSA offers a real-time evaluation of policy quality by taking into account both exploration and exploitation in its learning process. This proves especially advantageous in more complex environments where the action taken during the exploration phase might lead to penalties. As a result, SARSA computes a safer path as well as higher mean reward than Q-learning. In terms of parameter settings, I set the ϵ value for the ϵ-greedy policy at 0.1 to balance exploration and exploitation. An ϵ of 0.1 indicates that approximately 10% of the time, the agent will try a new action, while 90% of the time, it will capitalize on its acquired knowledge. This low ϵ value was preferred to ensure that the agent doesn't get stuck in local optima by being overly exploratory. Moreover, using a small ϵ value allows us to use the ϵ-greedy policy without decay. The learning rate α dictates how much of the new experience influences the current knowledge or *Q(s,a)* of the agent. I chose a fixed rate of $\alpha = 0.3$, as opposed to a decaying one, to ensure that the agent maintains a consistent approach to new information, regardless of its stage in the learning process. Moreover, choosing $\alpha < 0.3$ leads to slow learning, with the agent becoming stuck in sub-optimal policies. I initiated the Q-values with random numbers to ensure no initial bias towards any action for a given state. This randomness in the initial phase helps the agent in exploring various paths. Lastly, I operated under the assumption that with adequate episodes of 2000, the Q-values would converge, allowing the agent to derive an optimal or near-optimal policy for the maze.

3.2 Question 3.2



(a) Optimal Policy (b) Optimal Value Function

Figure 6: Optimal Policy and Optimal Value Function for Temporal Difference Algorithm - SARSA Method

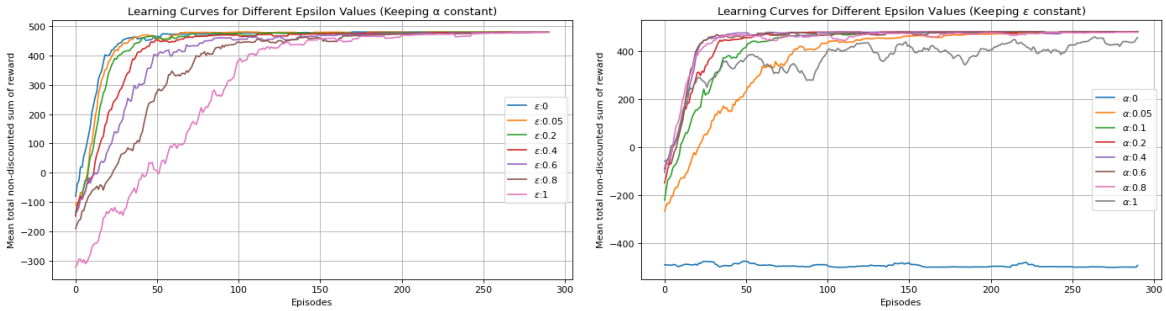
Like in Question 2.2, optimal policy obtained provides a strategic map for the agent, directing its movement in a way that likely maximizes the accumulated rewards. This is seen in Figure 6 (a). Figure

6 (b) shows the optimal value function obtained. States closer to the green square, tend to have higher values, indicating they are part of favorable paths.

3.3 Question 3.3

- Varying the exploration parameter ϵ :
  The exploration parameter ϵ determines the trade-off between exploration and exploitation. A higher ϵ encourages the agent to explore the environment more frequently, leading to a potentially rougher learning curve as the agent often makes random choices and increases the convergence rate. This can be seen in Figure 7 (a) for values of $\epsilon > 0.2$. Conversely, a lower ϵ prioritizes exploitation, which results in increased convergence rate as seen in in Figure 7 (a). As we are using a greedy policy, which prioritizes exploitation values of $0.05 \leq 0.2$ show fast convergence. However, such low exploration rate values can hinder the agent's ability to adequately explore the environment, potentially leading to convergence on sub-optimal policies. In theory, while a decay in ϵ over time might be used to ensure eventual convergence, maintaining a non-zero ϵ ensures continuous exploration, striking a balance between rapid learning and the assurance of optimal convergence.

- Varying the learning parameter α :
  The learning rate α determines how much of the new *Q(s,a)* estimate we adopt.The equation for the Q-value update in the SARSA method is given by:
  $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
  At high values of α, the agent gives more weightage to the new experiences.This can be beneficial in non-stationary environments but might cause oscillations or instability in the learning curve. This can be seen on Figure 7 (b) as the value of α is increased to 1. At low values of α, the agent gives more weightage to the past knowledge. This might result in slower learning but can lead to more stable estimates. From Figure 7 (b), we see that when α is 0, the agent doesn't learn from its experiences and the agent keeps getting negative rewards and doesn't show any improvement over time. In theory, to guarantee convergence, α must meet specific conditions like the Robbins-Munro condition.



(a) Effect of varying the exploration parameter ϵ on learning curve.

(b) Effect of varying the learning parameter α on learning curve.

Figure 7: Impact of varying the exploration parameter ϵ and the learning rate α on learning curves of TD algorithm.