Reinforcement Learning - Mid Semester Report

Ron Darmon (ID. XXXXXXXXX), Roei Arpaly (ID. XXXXXXXXX)

Submitted as Mid Semester project report for the RL course,
Reichman University, 2023

1 Introduction

Reinforcement learning is a type of machine learning that involves an agent learning to interact with an environment in order to maximize a reward. It has been applied to a wide range of tasks, including control, optimization, and decision-making. In the context of maze navigation, the agent can be thought of as the person or robot attempting to find their way through the maze, while the environment is the maze itself. The reward can be defined as the agent reaching the end of the maze or achieving some other goal.
There are many approaches to solving mazes using reinforcement learning, and in this work we focus on four specific algorithms: Dynamic Programming, Monte Carlo, SARSA, and Q-Learning.

Dynamic Programming is a method for solving optimization problems by breaking them down into smaller sub-problems and storing the solutions to these sub-problems in a table.

Monte Carlo methods involve using random sampling to estimate the expected value of a function.

Q-Learning is an off policy algorithm where the agent learns the optimal action-value function by following an off-policy learning rule, which means it learns the optimal policy while following a different (possibly sub-optimal) policy. This is done by updating the Q-values based on the maximum expected future rewards.

while, SARSA is a variant of Q-Learning but on-policy algorithm that uses a sample of the current state and action to update the Q-value.

The primary objective of this study is to evaluate and contrast the efficacy of four algorithmic approaches in comparison to both a random walker and each other within varying maze environments of different dimensions (5x5, 15x15, 25x25). These environments are characterized as stochastic, in which there is a 10% probability of the agent's intended action resulting in failure to reach the desired state. It is hoped that the results of this research will furnish valuable insights into the strengths and limitations of the analyzed algorithms, thereby facilitating the advancement of future strategies for solving maze-like problems.

1.1 Related Works

Reinforcement Learning, An introduction, Second Ediition, Richard S.Suttion, Andrew G. Barto, MIT press, cambridge, 2018
Bertsekas, D. P. (1987). Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall, Englewood Cliffs, NJ.

2 Solution

2.1 General approach

There are several approaches that could be taken to solve the problem of navigating a maze using reinforcement learning. Some options include:

1. Dynamic Programming is a method of breaking down a problem into smaller sub-problems and solving them using a table-based approach. Markov Decision Process (MDP) satisfy this condition while also addressing overlapping sub-problems which can be solved by reusing a cached solutions. MDP utilise the Bellman Equation which gives recursive decomposition for breaking down the optimal value function into two pieces (the optimal behaviour of one step and then the optimal value of the remaining steps), And the Value function which stores and reuses solutions. This method is well-suited for problems with a clear structure and optimal substructure, such as mazes with a fixed layout and a single optimal path. It can be used to find the optimal path in a maze by creating a table that stores the optimal solution for each sub-problem, and then using this table to find the overall optimal solution. This approach is efficient and can be used to find the optimal path in a maze quickly. However, Dynamic Programming is limited as it is not always possible to allocate enough memory to store the entire table on a given machine, for large/complex problems.

2. Monte Carlo methods: In the context of solving a maze with an agent, Monte Carlo methods can be used to find the optimal path by simulating random actions and observing the resulting rewards. The agent would start at the beginning of the maze and choose a random action based its state. The agent would then receive a reward and update its estimates of the expected rewards for each state and action based on the observed rewards. Over time, the agent would learn which actions tend to lead to high rewards, and would use this information to guide its movements through the maze. The agent would continue to simulate random actions and observe rewards until it reaches the goal or a stopping criterion.

3. SARSA and Q-Learning are both types of reinforcement learning algorithms that can be used to solve problems such as navigating a maze. These algorithms work by updating the value of each state and action based on the expected future reward. An agent would use SARSA or Q-Learning to learn the optimal path through the maze by exploring different paths and updating the value of

each state and action based on the rewards it receives. The agent would start at the beginning of the maze and choose an action (e.g. move west, east, north, or south) based on the current state. The agent would then receive a reward for the chosen action and update the value of the current state and action based on the expected future reward.
SARSA is an on-policy algorithm, which means it updates the value of the current state and action, based on the next state and action that the agent will take, rather than the optimal action. This approach involves taking a sample of the current state and action, and using that sample to update the Q-value, which is a measure of the expected future rewards for a given state and action. This allows the agent to update its estimates of the expected rewards for each state and action based on the next state and action it will take, rather than the optimal action. Q-Learning (Off Policy), on the other hand, uses the maximum expected future reward to update the value, which means it updates the value of the current state and action based on the best possible next state and action that the agent could take. Both SARSA and Q-Learning can be used to solve mazes by training the agent to learn the optimal path through the maze.
SARSA is generally considered to be better suited for problems with stochastic environments, while Q-Learning is better suited for deterministic environments.

4. DQN - (not utilized in the scope of the project). Deep Q-Network is a variant of Q-Learning that uses a deep neural network to approximate the Q-function. This allows DQN to handle high-dimensional state spaces, such as those encountered in video games and robotics.

5. Hybrid approaches: It is also possible to combine multiple reinforcement learning algorithms or to use a combination of reinforcement learning and other techniques, such as evolutionary algorithms or planning algorithms. This can allow for the strengths of each approach to be leveraged and may lead to better performance in some cases.

Ultimately, the choice of approach will depend on the specific characteristics of the maze and the goals of the agent. It may be necessary to try multiple approaches with different initialisation and hyper-parameters and compare their performance in order to determine the most effective solution.

2.2 Design

The presented work utilizes Python as the programming language and Google Colab as the platform for running the experiments. The training time for the reinforcement learning algorithms varied, with some taking a coupe of seconds and others a little longer. One of the technical challenges encountered was the utilization of code not inside a function, which resulted in the use of global variables and caused issues with duplicated code in later experiments.
Additionally, the use of non-object-oriented programming (OOP) code added complexity to the process. The notebook employed in the experimentation

process is quite extensive and required a significant number of imports for the setup. The implemented architectures for the reinforcement learning algorithms include Dynamic Programming, SARSA, Q-learning, and Monte Carlo.

3 Experimental results

In this work, we performed a grid search to tune the parameters of the three reinforcement learning algorithms - Monte Carlo, SARSA, and Q-Learning, in order to converge to the optimal route in the maze. The grid search allowed us to explore a range of parameter values for each algorithm and evaluate their performance. By optimizing the parameters, we were able to improve the convergence rate and accuracy of the algorithms, resulting in more efficient and effective maze navigation.

Overall, our results showed that the combination of reinforcement learning and a grid search optimization method was a powerful approach for solving the maze. By carefully tuning the parameters of the algorithms, we were able to achieve significantly better performance than with the default parameter values. This highlights the importance of parameter optimization in reinforcement learning and the potential benefits of using a systematic approach such as grid search (if it is computationally possible to do so).
We evaluated the performance of the different approaches on a 25x25 Maze environment. The objective of utilizing Dynamic Programming was to investigate firstly if there is a potential for obtaining the optimal policy to the problem, owing to the feasible table size of 4x25x25.

In the course of our experimentation, we attempted to employ the Monte Carlo algorithm on a 25x25 maze environment in various configurations and with varying parameter settings. Despite these efforts, the algorithm failed to converge to an optimal solution, most likely due to the randomness of the exploration (needing to reach each time to the terminal state of the maze which is highly unlikely considering the small amount of episodes). As a result, we deemed it necessary to exclude Monte Carlo from further consideration in our analysis.

In our examination of the Q-Learning algorithm, we initiated the process by utilizing a GridSearch methodology to identify the optimal hyperparameter values. Through this process, we were able to pinpoint combinations that converged at the minimal number of episodes. This was achieved by the implementation of a heuristic technique, specifically the use of a rolling average to mitigate the volatility of the rewards curve, as well as through a thorough analysis of the comparison rewards graph for each hyperparameter combination. The heuristic employed in this study not only aims to identify the optimal policy, but also seeks to minimize the number of episodes required for convergence by identifying when the learning curve reaches a stable plateau.
We conducted an empirical evaluation of the proposed Q-Learning algorithm

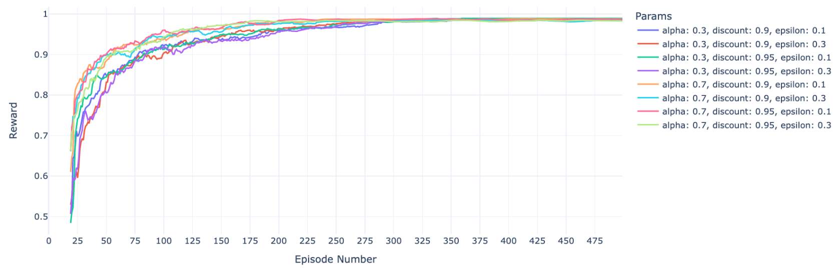

Figure 1: Smoothed reward curves of Q-Learning with different parameters

by implementing the best parameter configuration and generating a video to demonstrate its ability to effectively solve the stochastic version of the maze task. Furthermore, we employed a visualization technique to gain insight into the exploration and exploitation strategies employed by the agent, by observing the distinct states explored in each episode. This helped us to gain a deeper understanding of the behavior of the agent and how it learned to solve the maze.

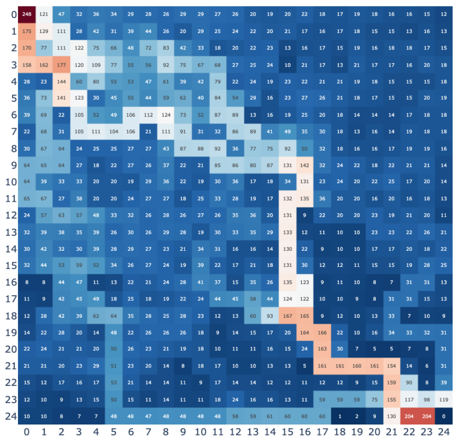

Figure 2: Distinct states visits

We observed that the agent tended to focus some of its exploration on the lower-left portion of the maze. Based on this observation, we hypothesized that blocking one of the main routes to that area with a negative reward would discourage the agent from traversing that path, and thus encourage it to explore other regions of the maze. Therefore, in order to test this hypothesis, we implemented this modification to the environmental setup and re-ran the evaluation.

The modification to the environment was effective, as evidenced by the reduction in the number of episodes required for the agent to converge on a solution.

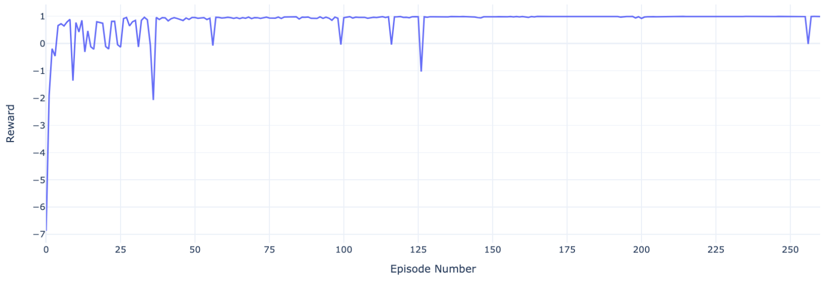

Figure 3: Convergence with planted rewards

Specifically, the agent reached convergence in approximately 200 episodes, as opposed to the initial 260 episodes, when the negative reward was not implemented. This suggests that the negative reward served to discourage the agent from exploring the irrelevant area of the maze. However, it is important to note that the agent still managed to explore that area to some degree, as there are other paths that lead to it, and also due to the inherent randomness of the stochastic environment, which can cause the agent to select random actions and accidentally traverse that area.

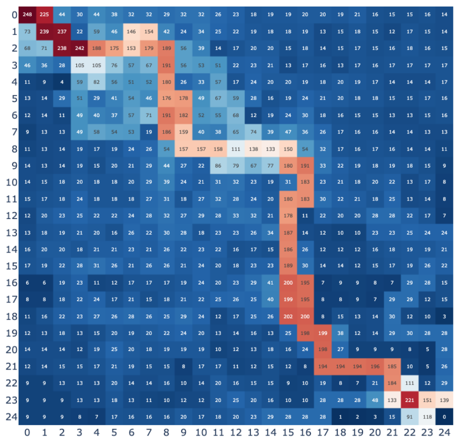

Figure 4: Distinct states visits with planted rewards

We attempted to experiment with a positive reward, however, the agent would repeatedly exploit it, failing to complete the maze. We also considered placing the reward adjacent to the final state, in hopes that the agent would happen upon it due to the stochastic nature of the environment, but ultimately decided to position it at the terminal state.

Our analysis revealed that the agent explored less the bottom-left part of the maze after the implementation of the modification. Furthermore, to test the robustness of the algorithm, we re-ran the evaluation with a fixed number of episodes (200) and observed that the agent was still able to converge and solve the maze within the limited number of episodes.

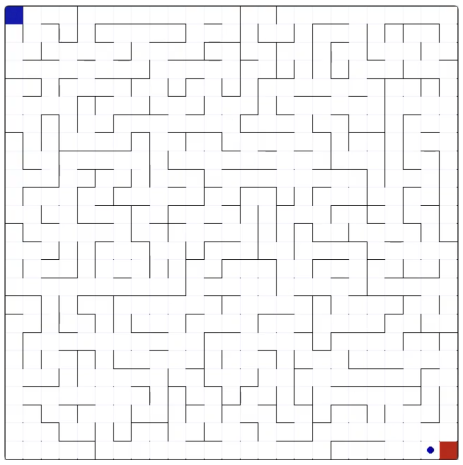

Figure 5: Agent is reaching terminal state

We conducted the same approach also for SARSA and observed that the SARSA algorithm converged slightly faster than the Q-Learning algorithm, with the number of episodes required being reduced by approximately 25 when the negative reward was implemented. As mentioned in the General Approach section, SARSA is better suited for stochastic environments because it takes into account the next action that the agent will take when updating the Q-values, as opposed to Q-Learning which only considers the maximum Q-value of the next state. Lastly, We also compared Q-Learning against SARSA.

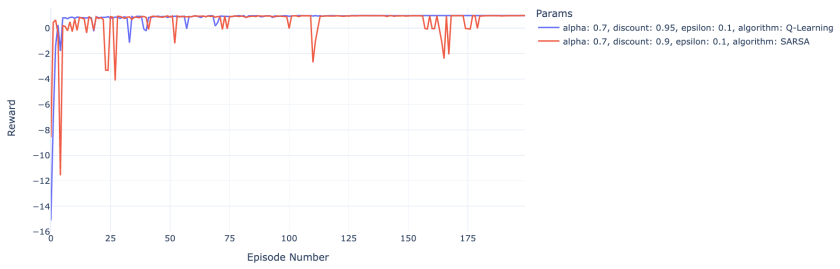

Figure 6: Q-Learning VS SARSA reward comparison

The results of the study indicate that there is no discernible variation in the reward function between the two approaches.
This is noteworthy, as the algorithms employed utilize different methodologies to navigate and solve the maze task at hand. Furthermore, the analysis of running times revealed that the three approaches utilized (Dynamic Programming, Q-Learning, and SARSA) exhibit similar computational efficiency for this

environment (maze 25x25).

Algorithm	**Runtime**
Dynamic Programming	$\sim$10.5 seconds
Q-Learning	$\sim$12 seconds
SARSA	$\sim$8.5 seconds

4 Discussion

In this study, we aimed to investigate the use of reinforcement learning algorithms for solving a maze navigation task. We chose to focus on four specific algorithms: Dynamic Programming, Monte Carlo, SARSA, and Q-Learning, which are all well-known in the field of reinforcement learning. We used a grid search approach to optimize the parameters of Monte Carlo, SARSA Q-Learning algorithms, allowing us to explore a wide range of parameter values.

The grid search methodology allowed us to evaluate the performance of the algorithms under a variety of conditions. Our results showed that this approach led to significant improvements in both the convergence rate and accuracy of the algorithms. This suggests that proper parameter tuning plays a crucial role in achieving optimal performance when using reinforcement learning for solving maze navigation tasks.

In addition to optimizing the parameters of the algorithms, we also employed a heuristic of a rolling average to mitigate the volatility of the rewards curve and minimize the number of episodes required for convergence. This heuristic helped to smooth out the rewards curve and reduce the variability in the performance of the agent.

Furthermore, we implemented a negative reward mechanism to discourage the agent from exploring irrelevant regions of the maze. This helped to reduce the number of episodes required for convergence, as the agent was able to focus its exploration on the most relevant parts of the maze.

Overall, the results of this study highlight the potential of reinforcement learning algorithms for solving maze navigation tasks. However, it also highlights the importance of proper parameter tuning, heuristics, and other techniques in achieving optimal performance. Future research could focus on expanding the size of the maze, incorporating additional obstacles and testing the scalability of the algorithms.

5 Code

`https://colab.research.google.com/drive/1Yy1UyOsDhe_iWFTDuQJvq_KsL7yjETCp`