Solving Dynamic Perfect Mazes Using RL-based Algorithms

Omkar Chekuri[1] and Yonathan Hendrawan[2]

University of Oklahoma*

omkar.chekuri@ou.edu[1], yfhendrawan@ou.edu[2]

Abstract

Reinforcement Learning (RL) is an area of machine learning where an agent learns by interacting with the environment by taking a sequence of actions to gain rewards on accomplishing some goal. The aim here is gain maximum cumulative rewards by taking the best possible actions. The environment is a model that represents the real-world problem/scenarios where decisions need to be made to achieve an end goal. Some of the examples of such environments are games, autonomous driving etc. In our problem we designed a game environment from scratch that is used for generating and solving dynamic perfect mazes. We used two reinforcement algorithms Q-learning and Multi agent Sarsa to solve these mazes. We compared the performance of these reinforcement learning algorithms with a maze solving algorithm. The result showed that Q-learning and Sarsa gave better result than Wall Follower algorithm in term of the number of steps taken to reach the goal. They can also solve the maze with a projectile. However, the solving becomes more difficult the bigger the maze is.

Introduction

Maze is a combination of paths and walls in a constrained space. The objective of solving a maze is usually for the player to find the path connecting the entry/starting point to the exit/ending point. For this project, we will use dynamic perfect maze. A perfect maze has a layout that does not have any circular paths or isolated spaces. For this project, we used dynamic perfect maze. The perfect here means that the maze layout does not have any circular paths and any isolated space. Whereas, the dynamic means that the maze layout will be created automatically using Maze Generating Algorithms such that every time the algorithms are run, they will create a new maze layout. In this way, we can create as many experiment set as possible. We used Depth First as the Generating Algorithms and Wall Follower as the Solving Algorithm.

Because of the different layout and size of the dynamic maze, it is better for us if we use a model free RL algorithms. That is the reason we used Q-learning and Sarsa as the RL algorithms. Both are model-free algorithms. The difference is that Sarsa learns action values while following the policy, whereas Q-learning does the learning using greedy policy approach.

The novelty of our project is that, to the best of our knowledge, there is no literature that compares RL algorithm with Maze solving algorithm in solving dynamic perfect mazes. There is also no literature of solving dynamic perfect mazes with a projectile using RL algorithms. We implemented our own version of Sarsa with multiple agents and multi-threading.

Contributions

We divided the project work as follows. Yonathan was the major contributor for building the maze environment, Depth First Search Maze Generating Algorithm, Wall Follower maze Solving Algorithm, Q-learning, and projectile movement.

Omkar was the major contributor for building Sarsa algorithm, Multi agent Sarsa algorithm, running experiments and plotting the results.

Related Work

The first reference (Osmanković and Konjicija 2011) reports their work on implementing Q-learning to solve a maze. They used MATLAB implementation of the Q-learning and Prim dynamic maze generator. They measured the system performance by execution time and average number of iterations. Our project took a similar approach in terms of using Q-learning to solve dynamic maze. However, there are several differences when we compare them. First, we built our RL algorithms and maze generator from scratch. Second, other than Q-learning, we also built Sarsa and Multi Agent Sarsa. Moreover, we also compared the RL performance with a maze solving algorithm: Wall Follower. Nonetheless, the paper gives a clear explanation regarding the method and the implementation.

The second reference (Stapelberg and Malan 2019) describes the attempt of conducting an analysis of a system that combines Reinforcement Learning, Fitness Landscapes, and Local Optima Networks to solve three problem instances: Vacuum world, Mazes, and Fruit collection task. Although they made their system to solve the problems, compared to our system, their work's aim was more into

* Both authors are CS5033 students - Machine Learning 2020

investigating the RL global structure. That is the first difference. The second difference is that they used five static mazes as their experiment materials, whereas we used dynamic mazes. Lastly, their system can only solve the maze in a small percentage of attempts. Despite the differences, we adopted some of their maze modelling approach, such as: the rectangular representation of the maze and cells, the agent can only observe their immediate surrounding environment, the agent must find the shortest possible path, and the path is starting from top-left corner.

The third reference (Takadama and Fujita 2004) mainly does the comparison of results obtained from simulations of Q-learning and SARSA in bargaining games to understand the sensitivity of these learning methods and explore the criteria for using these methods. The paper uses bargaining game where two agents implement different or same learning methods or to play the game. The goal is to achieve mutually beneficial agreement which is different from our problem domain of maze solving algorithms falling under the broad category of graph search. The results show that the payoff between the learning algorithms are the same while the negotiation process size differs, and SARSA attains superior results to Q-learning but with less rationale. These results indicate that discount factor might be an important criterion that should be selected based on the learning mechanisms as it affects the length of the negotiation process size. The other indication is, although some learning methods perform better, they might lack a good rationale in making decisions. So, if we want to understand the difference between rationale and performance in solving a problem, we need to use multiple agents with different learning algorithms to getter better insights. Thus, solving, using agents that use different learning mechanisms, should be considered to understand the performance and rationale better.

The fourth reference (Vidhate and Kulkarni 2016) describes and compares how agents cooperate in a multiagent environment by various methods such as strategy sharing and joint rewards algorithm and cooperative multi agent learning algorithm to converge faster in huge search spaces. In strategy sharing, the agents learn from averaging the Q-tables of other agents. In joint rewards algorithm, they use joint rewards instead of rewards which is calculated by weighted sum of rewards from other agents to promote cooperation among agents. Then the paper talks about multi agent learning algorithm which has two parts, independent learning, and cooperative learning. At defined points, the agent collects the Q-tables of the other agents and calculates the average of the Q-tables to be used in cooperative learning. A modified version works by, instead of taking the average of the Q-tables, taking the weighted average of the Q-table based on the expertness of the agents and uses it in the cooperative learning to solve the rest of the algorithm. The new proposed algorithm showed better convergence. In our project, we built a modified version of Joint Rewards algorithm (multi agent Sarsa) using multithreading to solve the maze.

Experiment Review

Hypothesis

We hypothesized that reinforcement learning algorithms (Q-Learning, Sarsa and multi agent Sarsa) can solve dynamic perfect mazes in comparable performance, in terms of the number of steps taken by the agent to solve the maze, with non-reinforcement learning based Maze Solving algorithm. Furthermore, we hypothesized that (Sarsa and Q-Learning) algorithms can solve dynamic perfect maze while avoiding projectiles.

Experiments

The maze is built using two-dimensional array. We used Wall and Cell Object for generating and solving the maze using Depth First Search and Wall Follower algorithm. For the Q-learning, Sarsa and multi agent Sarsa, we converted the array into simple integer array: zeros to denote cells and ones to denote walls. The Q-learning, Sarsa and multi agent Sarsa agent can move in four directions, namely north, south, east, and west. That means, our state representation was in the form of {x, y, n, s, e, w}; where x and y represent the agent's position, and n, s, e, and w represent the direction choices. In every step of the episodes, the agent must check whether the next state is valid or not. Validity here means that the next state must be a cell that the agent can move into, not a wall. The finishing condition is reached when the agent arrive at the bottom-left cell of the maze. In such case, the agent was given a reward of +1, otherwise, the reward was -0.01. We built every algorithm used in the experiments by ourselves.

Experiment 1 was implemented using Q-learning with learning rate 0.2, discount rate 0.95, epsilon 0.2, 100 episodes, and 1000000 steps for each episode. We conducted this experiment for various maze sizes.

Experiment 2 was implemented using Sarsa with the same parameters as in experiment 1. We also experimented using several maze sizes.

Experiment 3 was implemented using multi agent Sarsa with the same parameters as in the experiment 1. In this experiment, we run four Sarsa agents concurrently. Every ten episodes, the algorithm picked the maximum Q (S, A) to be used for all agent for the next episodes.

Experiment 4 was implemented using Wall Follower solving algorithm. It is a deterministic algorithm in the sense that it always produces the same path result, given the same maze layout. We counted the number of steps produced and use this number to compare the performance of the previous experiments.

These four experiments are used to test our first hypothesis. We use the number of steps taken by the agent to reach the goal as the metrics to decide whether the RL algorithms can produce comparable result to Wall Follower. The next experiments test our second hypothesis.

Experiment 5 had a projectile that are moving vertically in the middle of the maze and can move through the walls, one cell/wall for each step. Whenever the agent hits the projectile, it will get -10 reward and the episode will be restarted. We used Q-learning with the same parameters for this experiment.

Experiment 6 used Sarsa with moving projectiles. The parameters and projectile's attributes are the same as in experiment 5.

We used a simple metrics to prove the second hypothesis. If the agent can reach the goal, despite the threat of a moving projectile, we consider it to be successful.

For doing the experiments and comparisons, we do them together. We divided the workload equally.

Results and Analysis

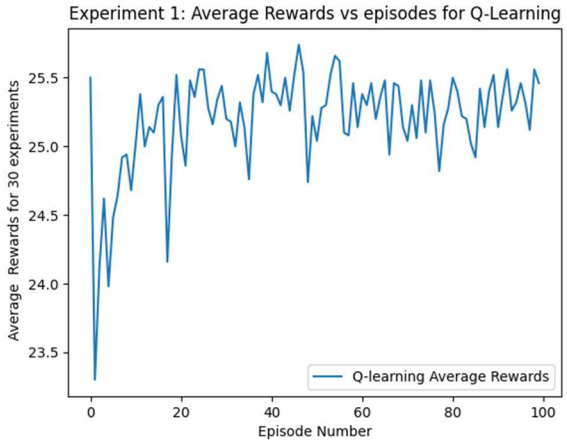

Figure 1. Q-learning (3x3 Maze) result.

Figure 1 shows the result of running 100 experiments for Q-learning algorithms. The line represents the averaged values from 30 experiments. After 100 episodes, it achieved around 25.4 as the total reward for 3 x 3 maze size.

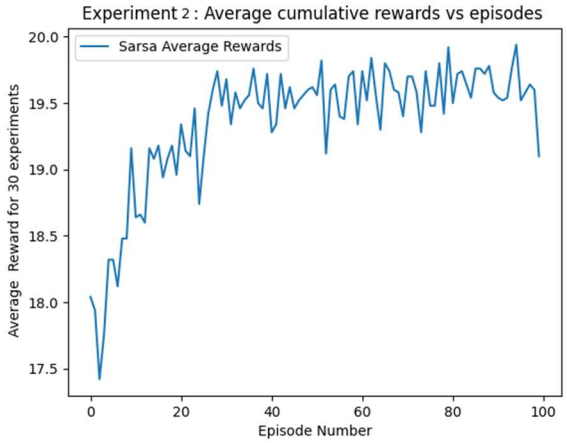

Figure 2. Sarsa (5x5 Maze) result.

Figure 2 shows the result of running Sarsa algorithm for 5 x 5 maze size. It achieved around 19.2 total reward after 100 episodes.

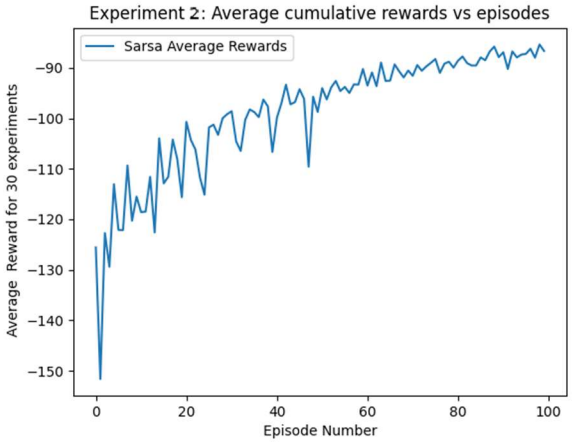

Figure 3. Sarsa (15x15 Maze) result.

Figure 3 shows the result of Sarsa algorithm on 15 x 15 maze size. It achieved around -87 total reward at the end of the episodes.

Figure 2 and 3 show that our dynamic maze generating algorithm works fine and the RL algorithm can also solve mazes of different sizes. We also tried with various maze sizes and the results are consistent.

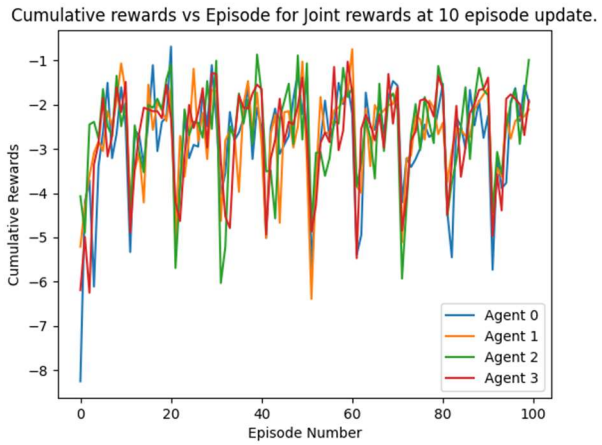

Figure 4.1. Multi agent Sarsa with 10-episode update rule on a 5x5 maze

Figure 4.1 shows the result of experiment 3. The Q (S, A) updates every ten episodes with epsilon 0.3 on a 5x5 maze made the graph unstable with high variance. It may be due to inefficient implement a thread scheduling to prevent for race condition while updating the Q-table. The other reason might be when solving small state spaces there might not be greater advantage using multiple agents for learning.

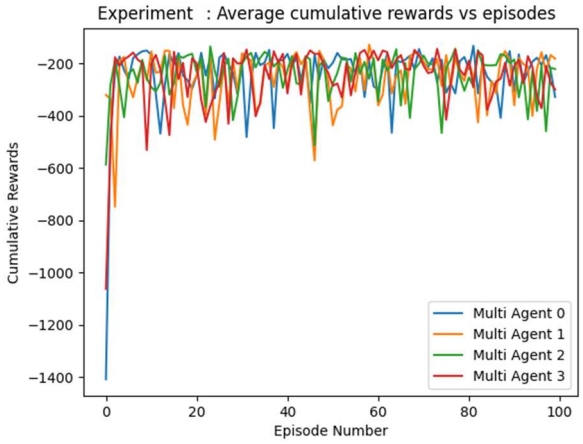

Figure 4.2. Multi agent Sarsa with every episode update rule on a 35x35 maze

Figure 4.2 shows the result of experiment 3 variation. The multi agents' Q (S, A) are updated on every episode with epsilon 0.1 on a 35x35 maze made the graph more stable with low variance. This shows that the more the agents communicate, the better they perform. The higher exploration may not be necessary when using multiple agents to learn in large state spaces.

For comparing the RL algorithms with Maze solving algorithm, we run 30 experiments and then measured the number of steps taken by agent to reach the goal. Out of those 30 experiments, 15 experiments are from 3 x 3 maze size, and 15 are from 4 x 4 maze size. Table 1 shows the result. For Multi Agent Sarsa, we did also 30 experiments, and then we took the best performing agent's steps. In this case, we also use every episode update rule as shown in figure 4.2.

Algorithm	Average steps
Wall Follower	19.7
Q-learning	11.7
Sarsa	11.7
Multi Agent Sarsa	23.8

Table 1. Average steps taken to reach the goal

From the result, we can see that the performance of RL algorithms is comparable to that of Wall Follower algorithm. In fact, they have better result. This proves that our first hypothesis is correct for the Sarsa and Q-learning. Our custom multi agent Sarsa cannot perform better than the Wall follower algorithm which shows that it needs much improvement, or the smaller states pace might be the reason.

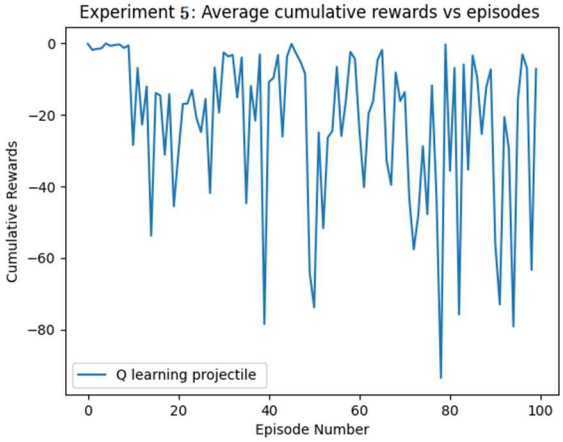

Figure 5. Q-learning on 3x3 maze with a projectile.

Figure 5 shows the result of experiment 5. Whenever the agent hits the projectile, the graph took a dip. On the other hand, if the agent can reach the goal, the graph increased sharply. Despite the ups and downs, Q-learning managed to reach the goal on several episodes.

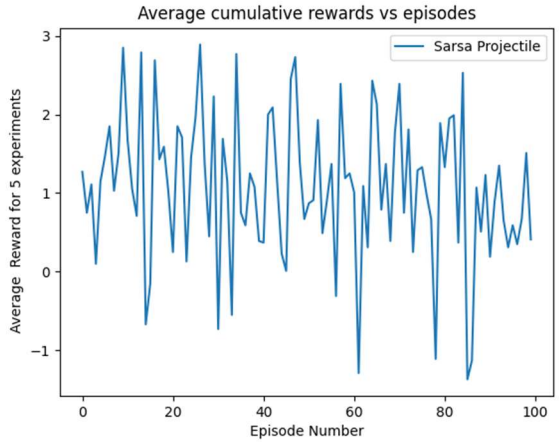

Figure 6. Sarsa on 3x3 maze with a projectile.

Figure 6 shows the result of running Sarsa on 3 x 3 maze with a projectile. Like the Q-learning equivalent, the graph has its sharp decreases and increases. This shows that Sarsa managed to reach the goal on several occasions.

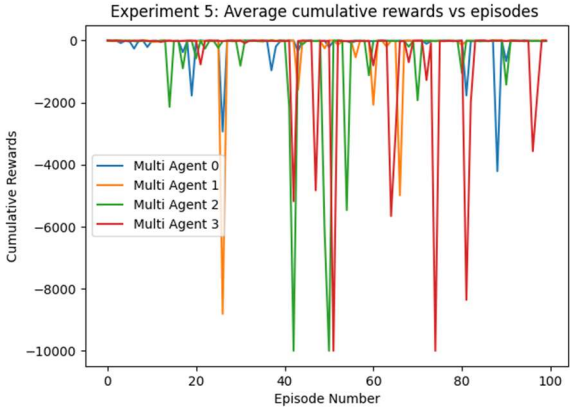

Figure 7. Multi Agent Sarsa on 3x3 maze with a projectile.

Figure 7 shows the result of running multi agent Sarsa on 3 x 3 maze with a projectile. The graph has its sharp decreases and increases. This shows that multi agent Sarsa also managed to reach the goal on several occasions.

We also tried to use different sizes of maze with a projectile to test Q-learning and Sarsa whether they can reach the goal or not. We acquired the result that the bigger the maze size, the harder for the agent to reach the goal. Therefore, for our second hypothesis, the result shows that RL algorithms can solve the maze despite the existence of a projectile, but it is becoming more difficult when the maze size is increasing.

Implementation and Learning

After making Q-learning, Yonathan learned that the algorithm is powerful. The main pseudocode has only fewer than ten lines of code. Yet, it can outperform the Wall Follower algorithm and solve the maze-with-projectile problem. The only disadvantageous part of it is in designing the state representation of the maze problem and keeping track the maze world rules and the correct values for the data structures.

For Omkar, implementing Sarsa and comparing its performance with other algorithm implementations made him having better insights about the algorithm. He also implemented modified version of Sarsa called (Multi agent Sarsa) with multiple agents each running on its own thread using multi-threading in python. We understood the issues of multithreading and how to make use of multi agent learning algorithms to some degree. We also learned about different strategies by which agents share their knowledge to reach the goal state quicker.

Summary

Q-learning and Sarsa can be used to solve dynamic perfect mazes as shown in the results. Compared to Wall Follower solving algorithm, the RL algorithms produced better result. They can also solve the maze with a projectile, even though it is getting harder the bigger the maze becomes.

Future Work

Implementing thread scheduling for collaborative system among many agents in solving the maze is a direction for our future work. Implementing Strategy Sharing algorithm for multi agent system is another path that we could pursue in the future. Changing the maze-projectile state representation such that it incorporates the projectile might give a better result. For example, since the projectile movement is based on the steps, when the agent hits the projectile, the algorithm will update the step in the state representation to a negative number.

References

Osmanković, D. and Konjicija, S., 2011, May. Implementation of Q—Learning algorithm for solving maze problem. In 2011 Proceedings of the 34th International Convention MIPRO (pp. 1619-1622). IEEE.

Stapelberg, B. and Malan, K.M., 2019, July. Global structure of policy search spaces for reinforcement learning. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 1773-1781).

Takadama, K. and Fujita, H., 2004. Q-learning and Sarsa agents in bargaining game. in North American Association for Computational Social and Organizational Science (NAACSOS).

Vidhate, D.A. and Kulkarni, P.A. 2016, Enhanced Cooperative Multi-Agent Learning Algorithms (ECMLA) using Reinforcement Learning. 2016 International Conference on Computing, Analytics and Security Trends (CAST), 556-561.