Q-Learning Based Solution For Solving Mazes

Thang Doi
Data Science and Artificial Intelligence
HUST, School of Information Communication and Technology

December 29, 2024

I. Introduction

The maze problem is a classic challenge in artificial intelligence and robotics. The primary objective is to guide an agent from a starting point to a target point within the maze. This problem can be divided into two subproblems: solving mazes in deterministic and non-deterministic environments.

In deterministic environments, the agent has complete knowledge of the maze, including all states and goals. As a result, several search algorithms, such as A* and the Depth-First Search (DFS) algorithm, can be applied to find the optimal path. However, in non-deterministic environments, the agent cannot cover the entire maze; it can only identify its current state, making it difficult to implement a search algorithm. This scenario becomes a type of decision-making problem. Therefore, the reinforcement learning algorithm is suitable for solving this problem.

In this project, the second section shows the maze problem formula, And in the Main Work section, the main algorithm includes the maze generate algorithm, and the Q-learning algorithm to solve the maze problem. Implementing code of this project store in github

II. Problem Formular

This section illustrates the maze problem in reinforcement learning. Structured learning in reinforcement learning can be described in the figure below.

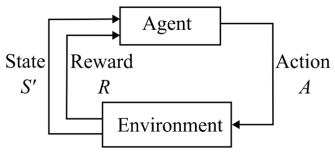

Figure 1: Reinforcement Learning Structure

Reinforcement learning [1] has several components: Agent, Environment, Action, Reward, and State. It can be represented as follows: for each state in the environment, the agent takes an action, receives a reward from the environment, and transitions to a new state. Therefore, the learning process can be represented by a Markov Decision Process (MDP) (S_t, A_t, R_t, S_{t+1}), where the next state depends only on the current state. In the case of the maze problem, the agent starts at a random location, and the learning process terminates when the agent reaches the goal location. This project uses value-based reinforcement learning to estimate the reward of each state and action from current to future. We can see in the formula show below.

$$Q(s,a) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^T r_{t+T}$$
$$Q(s,a) = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-1} r_{t+T})$$
$$Q(s,a) = r_t + \gamma Q(s',a)$$

$$\text{Temporal Different} = r_t + \gamma Q(s',a) - Q(s,a)$$

Where $Q(s,a)$ is the expectation reward of state s when taking action a, in the Q-learning algorithm, temporal differences mainly used to update the Q-value table.

II. Main Work

The following section shows the main work in this project, including a maze generate problem, and reinforcement learning algorithm in training agent.

2.1 Maze Generate Algorithm

The Maze generate problem is a classical problem in graph theory. A maze can be formally defined as a structure consisting of a set of cells, typically represented in a two-dimensional grid, where some cells are designated as walls and others as paths. The purpose of this problem breaking walls between each cell.

The maze can be represented as a graph, where the cells are vertices, and the edges represent possible paths between adjacent cells. Therefore, the solution to the maze generation problem can be modeled as a spanning tree or a cycle graph, where a spanning tree ensures connectivity without

loops, while a cycle graph includes cycles within the maze, it mean that a maze with just one path from start to goal, and otherwise more than one path. In this work, implementing two algorithm Depth First Search (DFS) and Prime algorithm for maze generation.

The first is DFS [2] with no cycle path in maze, agent can only forward one path from start to goal.

Algorithm 1 DFS Maze Generation

Input: N_c (Number of columns), N_r (Number of rows)
Output: A randomly generated maze
Initialize M as a grid maze of size $(N_r \times 2 - 1) \times (N_c \times 2 - 1)$ with all cells set to -1
Initialize G: a graph with $N_c \times N_r$ nodes, and L: a list to mark unvisited nodes
`curr_node = start` as a random selection from G ($S_{i,j} \in G$)
while `L` is not empty **do**
    `adjacents = valid_adjacent(curr_node)`
    if `adjacents` is empty **then**
        Remove `curr_node` from `L`
        if `L` is empty **then**
            Set `goal = curr_node`
            break
        end if
        continue
    else
        `next_node = random_selection(adjacents)`
        `G[next_node] = curr_node`
        `curr_node = next_node`
    end if
end while
`// Using the spanning tree to break walls in the grid maze`
`// Node in graph mapping to maze coordinates:` $(i, j) \rightarrow (i \times 2, j \times 2)$
for `each node in G` **do**
    `M[i * 2][j * 2] = 0` ▷ Mark the node as open in the maze
    `node_connect = G[node]`
    `dx, dy = distance(node, node_connect)`
    `M[node[0] + dx][node[1] + dy] = 0` ▷ Break the wall between nodes
end for

The second algorithm is Prim's [2] Algorithm, used to generate a spanning tree. The process begins by creating a spanning tree based on Prim's method, which ensures that all nodes are connected with no cycles. After constructing the spanning tree, random reconnections are introduced to create cycles within the graph. This means that the agent can have multiple paths from the start node to the goal node, offering greater flexibility and variability in navigating the maze.

Algorithm 2 Prime Maze Cycle Generation

Require: N_c (Number of columns), N_r (Number of rows)
Ensure: A randomly generated cycle maze
Initialize M as a grid maze of size $(N_r \times 2 - 1) \times (N_c \times 2 - 1)$ with all cells set to -1
Initialize G: a graph with $N_c \times N_r$ nodes
Initialize L: a list to mark connected edges, and N_e: number of edges created, set to 0
while $N_e < N_c \times N_r$ **do**
    `node` $\leftarrow$ `random_node(`G`)`
    `list_adjacents` $\leftarrow$ adjacent nodes of `node` in L
    if `list_adjacents` is empty **then**
        `remove node from` G
        continue
    else
        `adjacent` $\leftarrow$ `random_node(list_adjacents)`
    end if
    if `not is_created_cycle(node)` **then**
        G[`node`][`adjacent`] $\leftarrow$ `False`
        L[`node`][`adjacent`] $\leftarrow$ `True`
        Increment N_e
    end if
end while
`create_cycle`(G, `number_cycle`)
Perform wall-breaking process (like DFS)
return M

2.2 Q-Learning Algorithm

Q-learning [3] is a fundamental reinforcement learning algorithm used to train agents to make sequential decisions in an environment. It is a model-free method, meaning it does not require prior knowledge of the environment's dynamics, and it seeks to find an optimal policy that maximizes the cumulative reward over time.

Algorithm 3 Q-Learning Algorithm

Input: State space S, Action space A, Learning rate α, Discount factor γ, Exploration rate ε, Number of episodes N
Output: Q-table $Q(s,a)$
Initialize $Q(s,a) =$ arbitrarily for all $s \in S$, $a \in A$
for episode $= 1$ to N **do**
    current state $s =$ start location
    while s is not terminal **do**
        if s not in $Q(s,a)$, then update $(s, valid_a)$
        t : random probability
        $$a = \begin{cases} \text{a random action } a \in A_s, & \text{if } t > \varepsilon, \\ \arg\max_a Q(s,a), & \text{otherwise.} \end{cases}$$
        Take action a, observe reward r and next state s' from environment
        Update $Q(s,a)$:
        $$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$
        $s \leftarrow s'$
    end while
end for
return $Q(s,a)$

Double Q-learning [4] is an enhancement of the traditional Q-learning algorithm designed to address the overestimation bias that can occur when updating Q-values. In standard Q-learning, the maximum Q-value $max_a Q(s,a)$ is used to estimate the future reward, which may lead to overly optimistic value estimates.

Double Q-learning mitigates this issue by maintaining two separate Q-tables Q_1 and Q_2. During updates, one Q-table is used to select the action, and the other is used to evaluate it.

Algorithm 4 Double Q-Learning Algorithm

Input: State space S, Action space A, Learning rate α, Discount factor γ, Exploration rate ε, Number of episodes N
Output: Two Q-table $Q_1(s,a)$, $Q_2(s,a)$
Initialize $Q_1(s,a) =$, $Q_2(s,a) =$ arbitrarily for all $s \in S$, $a \in A$
for episode $= 1$ to N **do**
    current state $s =$ start location
    while s is not terminal **do**
        if s not in $Q_1(s,a)$, $Q_2(s,a)$ then update $(s, valid_a)$
        t : random probability
        $$a = \begin{cases} \text{a random action } a \in A_s, & \text{if } t > \varepsilon, \\ \begin{cases} \arg\max_a Q_1(s,a), & \text{if } table - select = 0, \\ \arg\max_a Q_2(s,a), & \text{if } table - select = 1 \end{cases} & \text{otherwise.} \end{cases}$$
        Take action a, observe reward r and next state s' from environment
        If table-select == 0 :
        $$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\left[r + \gamma \max_{a'} Q_2(s',a') - Q_1(s,a)\right]$$
        If table-select == 1
        $$Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\left[r + \gamma \max_{a'} Q_1(s',a') - Q_2(s,a)\right]$$
        $s \leftarrow s'$
    end while
end for
return $Q_1(s,a)$, $Q_2(s,a)$

III. Results and Application

To visualize the agent's learning process in a maze environment, the project implements a GUI main program. The structure and functionality of the program are illustrated in the figure below.

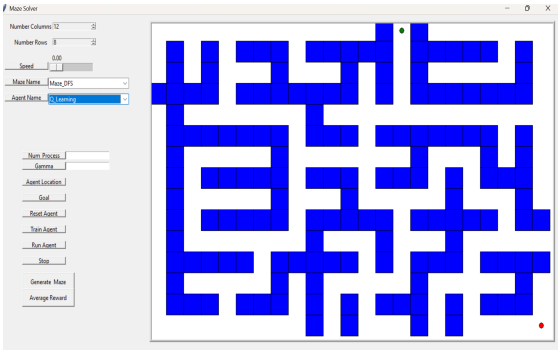

Figure 2: GUI project

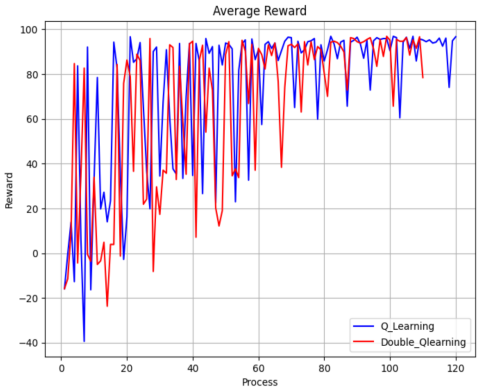

Figure 3: Average reward of Q-learning and Double Q-learning Algorithm

1 IV. Conclusion

This project implement a simple reinforcement learning algorithm that is my interest topic. Q-learning is a tabular algorithm, therefore it have several limitation such as inefficient in learning with large state and action space. In the future, I decide develop enhancent algorithm deep reinforcement learning having SOTA compare with traditional reinforcement.

Acknowledgments

I would like to express our sincere gratitude to Teacher Michel Toulouse, my instructor for the final semester Project 1, for providing me with excellent guidance and support, enabling us to successfully propose and complete this project.

References

[1] Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

[2] Wikipedia contributors. (n.d.). *Maze generation algorithm*. Retrieved from `https://en.wikipedia.org/wiki/Maze-generation-algorithm`

[3] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge.

[4] Van Hasselt, H. (2010). *Double Q-learning*. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems (NIPS 2010)*.