University of Waterloo

#### Faculty of Engineering

#### Department of Electrical and Computer Engineering

Assignment 2 Report

Maze World

### ECE 493 University of Waterloo

Jian Peng Xu

20665166

[jp4xu@uwaterloo.ca](mailto:jp4xu@uwaterloo.ca)

4A Computer Engineering

5 July 2021

Contents

[Cover Page 1](#_Toc76664082)

[1 Algorithms 3](#_Toc76664083)

[1.1 SARSA 3](#_Toc76664084)

[1.2 Q-Learning 3](#_Toc76664086)

[1.3 Expected SARSA 3](#_Toc76664087)

[Based on the Bellman update rule for expected salsa, we uses the expected value of 𝑸(𝑺0, 𝒂) for deciding all the actions: 3](#_Toc76664088)

[1.4 SARSA(λ) 4](#_Toc76664089)

[2 Quantitative Analysis 4](#_Toc76664090)

[3 Conclusion 6](#_Toc76664091)

# Algorithms

## SARSA

## SARSA, as an on-policy Temporal Difference control algorithm, learns the action value function. As you can from figure 1, the learn function starts off with a state-action pair (s, a) and finishes with a new state – s\_. We then use the epsilon-greedy policy to select the next action a\_. Next, we update the state-action value Q(S, A) by using the SARSA formula 𝐐(𝐒, 𝐀) ← 𝐐(𝐒, 𝐀) + 𝜶 [𝑹 + 𝜸𝑸(𝑺0, 𝑨0) − 𝑸(𝑺, 𝑨)]. From this equation, we can see that the TD target is being calculated as 𝑹 + 𝜸𝑸(𝑺0, 𝑨0) by adding the reward with the multiplication of the gamma with the decayed state-action value 𝑸(𝑺0, 𝑨0). The purpose of subtracting the 𝑸(𝑺, 𝑨) from the result is to calculate the TD error. We then multiply the result with alpha and add the original 𝑸(𝑺, 𝑨). The whole SARSA update changes the state of S to 𝑺0 and action A to 𝑨0. Every iteration will lead to the bellman update accordingly. Eventually, the state-action value Q(S, A) will converge to the optimal policy.

Text

Description automatically generated

## Q-Learning

## Expected SARSA

## Based on the Bellman update rule for expected salsa, we uses the expected value of 𝑸(𝑺0, 𝒂) for deciding all the actions:

**𝐐(𝐒, 𝐀)** ← 𝐐(𝐒, 𝐀) + 𝜶 [𝑹 + 𝜸 ∑𝒂 𝝅(𝒂 | 𝑺′) 𝑸(𝑺0, 𝒂) − 𝑸(𝑺, 𝑨)]

Expected SARSA is similar to Q-Learning, expected that instead of choosing the maximum of the Q(S’, a) it uses the expected value of Q(S’, a) for all the actions:

𝐐(𝐒, 𝐀) ← 𝐐(𝐒, 𝐀) + 𝜶 [𝑹 + 𝜸 𝑬[𝑸(𝑺0, 𝑨0) | 𝑺0] − 𝑸(𝑺, 𝑨)]

← 𝐐(𝐒, 𝐀) + 𝜶 [𝑹 + 𝜸 ∑𝒂 𝝅(𝒂 | 𝑺′) 𝑸(𝑺0, 𝒂) − 𝑸(𝑺, 𝑨)]

The probabilistic distribution for this Bellman will be as follow:

Text

Description automatically generated with medium confidence

The equation calculates the next expected value of the next state- action using the probabilistic distribution. It then updates the state by using the Bellman equation above. The bellman equation first calculates the TD target by adding the expected value - 𝛾 𝐸 [𝑄 (𝑆0, 𝐴0) | 𝑆0] with the reward - R and then subtract the old estimation of Q(S, A) from it to get TD error. The next step will be applying the Bellman update to the current estimate by multiplying the alpha with the TD error. Eventually, we add the new updates with the old estimate Q(S, A). This method updates the bellman equation iteratively on each step to finally converge the state-action value Q to the optimal policy.

Text

Description automatically generated

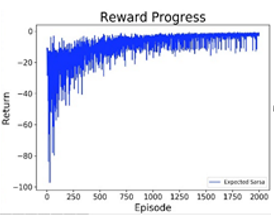
## SARSA(λ)

# Quantitative Analysis

**SARSA in Maze 1: SARSA in Maze2: SARSA in Maze 3:**

|  |  |  |
| --- | --- | --- |
| Chart  Description automatically generatedmax reward = 0.3 | max reward = -0.1 | max reward = -0.7 |
| Med\_Last100 = 0.3 | Med\_Last100 = -0.3 | Med\_Last100 = -1.9 |
| Var\_Last100 = 0.02902 | Var\_Last100 = 0.0521 | Var\_Last100 = 4.78 |

**Expected SARSA in Maze 1: Expected SARSA in Maze2: Expected SARSA in Maze 3:**

Chart

Description automatically generatedChart

Description automatically generated

|  |  |  |
| --- | --- | --- |
| max reward = 0.3 | max reward = -0.1 | max reward = -0.7 |
| Med\_Last100 = 0.3 | Med\_Last100 = -0.3 | Med\_Last100 = -1.75 |
| Var\_Last100 = 0.0209 | Var\_Last100 = 0.046 | Var\_Last100 = 3.76546 |

From the plots of Sarsa and Expected Sarsa, we can see that through maze 1 to maze 3, both algorithms are converging in a slower rate with low max reward value but high variance. This is because the maze is getting hard to find the solutions from 1 to 3. In Maze 1, two walls and two pits which are overall the easiest maze for the agent to find out solutions comparing with in Maze 2 that has nine walls with no pits and in Maze 3 with 13 walls and 3 pits.

harder. Therefore, maze 3 is harder than maze 2 and maze 2 is harder than maze 3. The more different maze is, the worse performance the RL algorithms get.

# Conclusion