

CP8319/CPS824 Reinforcement Learning

Winter 2021

Assignment 2

Rohaam Ahmed

PhD Student - Computer Science

rohaan.ahmed@ryerson.ca

March 23, 2021



Instructor: Dr. Nariman Farsad

1. Monte Carlo GLIE

1.c. Deterministic Monte Carlo GLIE

First-Visit Monte Carlo

It took 0.01026071310043335 minutes to execute 1,000 iterations.
The success rate of the policy across 500 episodes was 100.00 percent.

The resulting policy map can be seen in Fig 1

	0	1	2	3
0	>	>	v	<
1	>	^	v	HOLE
2	<	HOLE	v	<
3	HOLE	>	>	GOAL

Figure 1:

1.d. Stochastic Monte Carlo GLIE

First-Visit Monte Carlo

It took 0.1323713501294454 minutes to execute 10,000 iterations.
The success rate of the policy across 500 episodes was 19.60 percent.

There are several possible reasons for the low accuracy in the stochastic case:

- The environment is highly stochastic, thus the state change per action is not as expected for a large percentage of steps. This means the agent is unable to learn, with a high degree of certainty, which action in each state leads it closer to the Goal state.
- The rewards are “sparse”, i.e., there are large durations where no reward is seen, thus making it difficult for the agent to map rewards to past actions.
- There are no negative rewards (penalties) for falling into a hole.

2. TD SARSA and Q-Learning

2.a. TD SARSA

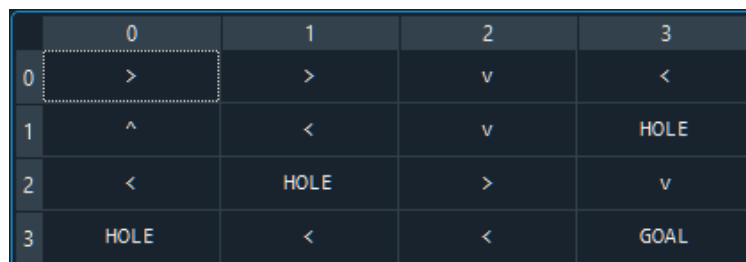
Deterministic

Temporal-Difference

It took 0.0013014634450276693 minutes to execute 1,000 iterations.

The success rate of the policy across 500 episodes was 100.00 percent.

The resulting policy map can be seen in Fig 2



	0	1	2	3
0	>	>	v	<
1	^	<	v	HOLE
2	<	HOLE	>	v
3	HOLE	<	<	GOAL

Figure 2:

Stochastic

Temporal-Difference

It took 0.015299387772878011 minutes to execute 10,000 iterations.

The success rate of the policy across 500 episodes was 15.20 percent.

The reason for the low accuracy is similar to Section 1.d., the high stochasticity makes it difficult for the agent to map rewards with actions performed in the past.

2.b. Q-Learning

Deterministic

Q-Learning

It took 0.001302011807759603 minutes to execute 1,000 iterations.

The success rate of the policy across 500 episodes was 100.00 percent.

The resulting policy map can be seen in Fig 3

Stochastic

	0	1	2	3
0	>	>	v	<
1	^	<	v	HOLE
2	<	HOLE	>	v
3	HOLE	<	<	GOAL

Figure 3:

Q-Learning

It took 0.01518313487370809 minutes to execute 10,000 iterations.
The success rate of the policy across 500 episodes was 4.20 percent.

The reason for the low accuracy is similar to Section 1.d., the high stochasticity makes it difficult for the agent to map rewards with actions performed in the past.

2.c. Improving Q-Learning and TD SARSA

We can improve the performance of the algorithms (both Q-Learning and TD SARSA) by tuning the rewards so that they are not sparse, and introducing negative rewards.

Namely, we make the following changes to the rewards:

- Reward of achieving the Goal State = +10.0.
- Reward of falling into one of the Hole States = -10.0
- Add a "Living Penalty", i.e., a negative reward for each time-step where the Goal State is not achieved = -1.0
Doing this incentivizes the agent to learn "quickly".

The above measures alone are sufficient to improve the learning significantly for 10,000 iterations, from under 25 percent to above 80 percent. However, we will also increase the number of iterations to the maximum allowable limit of 100,000, which allows us to achieve nearly perfect accuracy.

Result:

Q-Learning

The average success rate of the policy over 10 training sessions of 100,000 iterations was 93.60 percent. However, only one out of the 10 sessions did not achieve 100.00 percent accuracy (36 percent), making it an outlier.

Temporal-Difference

The average success rate of the policy over 10 training sessions of 100,000 iterations was 100.00 percent, with a Variance of 0.0.

The resulting policy map after one of the training session can be seen in Fig 4

	0	1	2	3
0	v	>	<	^
1	^	^	<	HOLE
2	^	HOLE	>	v
3	HOLE	>	>	GOAL

Figure 4:

As can be seen, tuning the reward to incentivize learning, introducing negative rewards, and adding a living penalty helps the agent learn the optimal policy. Based on experience, reward tuning is a vital optimization step in any real-world Reinforcement Learning algorithm. It should be noted that increasing the iterations from 10,000 to 100,000 alone (without tuning the rewards) did not result in any noticeable learning improvement. Thus, the improvements were primarily due to rewards tuning.

A drawback of adding a living penalty is that the agent sacrifices exploration for quick learning. In some cases, once an agent is able to find a “good enough” path to the Goal, it may not attempt to find a more optimal path in the future. This can be mitigated by adding a living penalty which increases with time, starting from 0.