# [COMP6247] Lab One Report

Feng Xie — 30502322 — fx1n18@soton.ac.uk

## 1 Introduction

In this lab, the main aims are to understand how dynamic programming reduces execution time and to be familiar with OpenAi Gym environment and understand how Q-Learning can be applied to the Mountain Car problem.

## 2 Question one

### 2.1 Methodology

Each Fibonacci number is the sum of two preceding numbers. In the code given, the number sequence starts from 1 and 1. In dynamic programming, we try to divide the problem into simpler sub-problems and store the solutions of sub-problems in a data structure so we can just look up the previously computed solution when the same sub-problem occurs later. The Fibonacci sequence is suitable for applying dynamic programming because the sub-problems can be computing the values of the two preceding numbers.

To show how dynamic programming in which pre-computed partial solutions are re-used makes the algorithm efficient, we need to write another program which compute the Fibonacci number without the use of dynamic programming and compare the execution time of the two programs given the $n^{th}$ number to be solved.

### 2.2 Results

We have a program which computes the Fibonacci number in the normal way and another program which applied dynamic programming to compute the Fibonacci number. The figure 1 shows the time comparison of the two programs when we compute the Fibonacci sequence from one number to 30 numbers.

From the figure, it is clear that the program with dynamic programming runs much faster than the other.

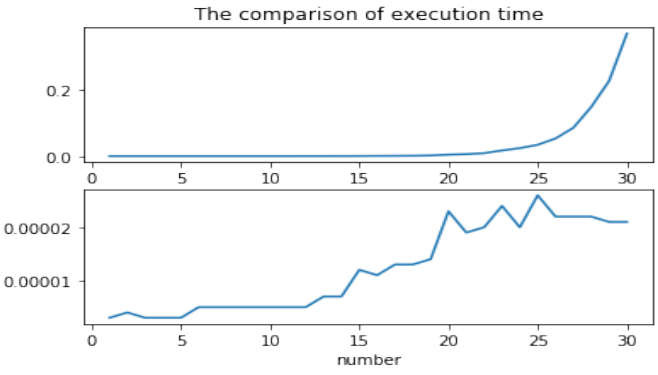



Figure 1: The comparison of execution time

# 3 Question Two

## 3.1 Methodology

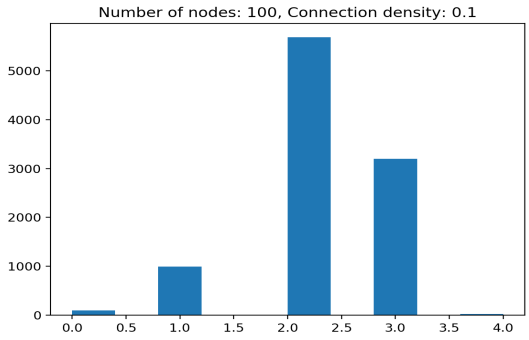
Dynamic programming can also be used to compute the shortest path in a network. To understand the network connectivities, we can construct a directed graph of 100 nodes with the connection density varying from 0.1 to 0.5 and the cost of each link can be set to one. The density can be implemented by applying uniform distribution to the probability of connections. For example, if the density is set to 0.3, and a random sample drawn from a uniform distribution over 0 and 1 is lower than 0.3, then a connection will be built between a pair of nodes.

A few histogram will be plotted to show how the connection density affects the pairwise shortest distances.

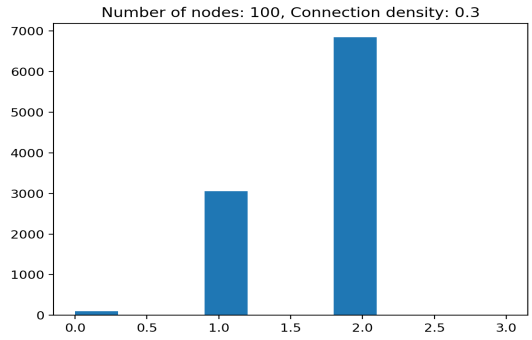
## 3.2 Results

Experiments were made with different connection densities and the results are shown as below. Each histogram shows the pairwise shortest distances in the network.

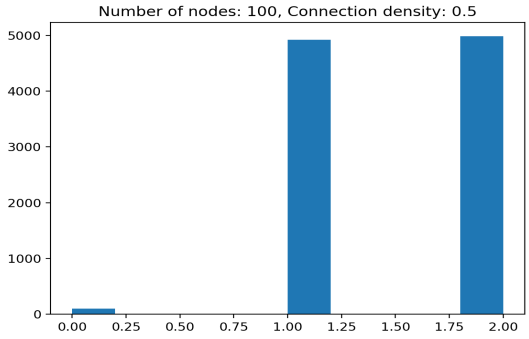
It can be seen from the results that as the connection density increases, the distribution of shortest distances between pairs of nodes is likely to be in a smaller range of values. When the density is 0.1, the longest distance between pairs of nodes can reach four, which means that a node need to pass at least four edges to reach its neighbor. When the density is 0.5, the longest distance between pairs of nodes is as short as two. Additionally, a distance of zero refers to the distance between a node and itself.




(a)




(b)




(c)

Figure 2: Different connection densities for a network of 100 nodes

# 4 Question Three

## 4.1 Methodology

The Mountain Car problem was solved by learning a control policy using Q-Learning. The idea of Q-Learning for this problem is that after running a certain number of episodes to update the Q-table, the table is able to store the best action for every state given the position and velocity of the car so that the car learns how to reach the goal position faster by itself.

Exploration and exploitation were implemented by setting a small value of exploration probability, for example, 0.05. If a sample randomly drawn from a uniform distribution over the range of zero and one is lower than 0.05, the program will choose a random action from the action space. If the random sample is higher than 0.05, the program will look up the Q-table and choose the action with the highest action value at that state.

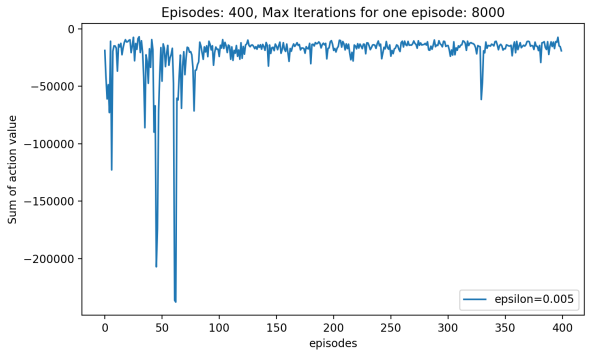
Since the parameters of the states, positions and velocities, are continuous in the Mountain Car problem, we need to discretize them. One way to do it is scaling them so that they will be limited in a specified range of discrete states. In the code given, the number of discretized states was set to 40. As for the action spaces, they were discretized into three actions. 0 means pushing left, 1 means no push and 2 means pushing right.

To show how the algorithm converges, the action values were summed up for every iteration in an episode. The maximum numbers of iterations and episodes were set up so that the evolution of the sum of action values can be plotted to show the convergence. To show how convergence changes, the exploration probability was increased and decreased from the default value of 0.05.

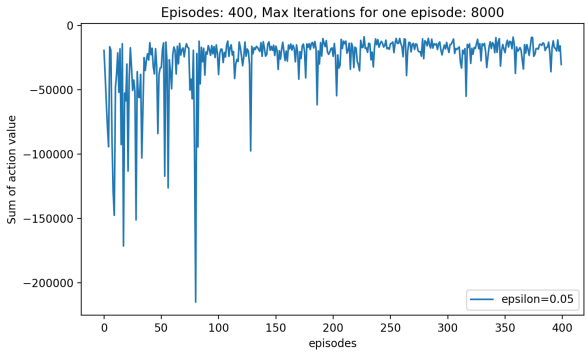
## 4.2 Results

The numbers of maximum iterations and episodes were set to 8000 and 400 respectively. And the values of exploration probability "epsilon" were set to 0.005, 0.05 and 0.5 for experiments. The results are shown as below.

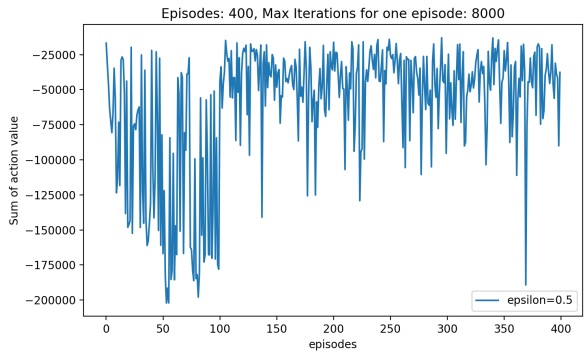
It is easy to find out that when epsilon was small, the sum of action values got steady and converged faster. Since epsilon is the exploration probability, a higher value of epsilon means that we are taking more random actions instead of choosing the action with the highest value in the Q-table. So when the epsilon was 0.5, the sum of action values for a single episode was difficult to converge because we were still taking random walks half of the time.




(a)




(b)




(c)

Figure 3: Convergence for different values of epsilon

# 5 Conclusion

Dynamic programming has a great advantage in reducing computational cost and thus can be applied to many problems such as computing the shortest path in networks. Q-Learning is good for learning a control policy and the convergence depends on how we tune the parameters such as the exploration probability and the number of episodes.

# References

[1] https://openai.com/

[2] https://github.com/openai/gym/wiki/MountainCar-v0