# MSc Dissertation

# Final Report

# MSc in Telecommunications

**Student:** Jinhao Wu

**Student number:** 18110353

**Project Title:**

Reinforcement learning with routing

**Supervisor:** Prof. Miguel Rio

University College London

Dept. of Electronic and Electrical Engineering

2018

1. **Abstract**

Text here, maximum of 1 a page

**Table of Contents**

Add a table of contents including section numbers and page numbers. You can also include a nomenclature table and a list of figures and tables, as appropriate to your project.

# Introduction and Problem Statement

## Introduction

In recent years, as the scale of the network grows larger and larger, the network becomes more and more complex and variable, the traditional routing algorithms are gradually difficult to adapt to the current networks. To solve this thorny problem, many network engineers and scientists are committed to design new generation routing algorithms to adapt to current growing network. The difficulties of designing a new routing algorithm is how to adapt to the dynamic changes of network links and the complex uncertainty of the network load [1]. Therefore, how to solve these two problems is a challenge for researchers.

The traditional static routing algorithm performs routing according to some certain fixed rules, so these routing algorithms cannot response to the network changes and fluctuations in time to adjust the rules. It is necessary to manually update the rules to adapt to the new network environment. The dynamic routing algorithms can dynamically perform routing based on the current state of the network. The current dynamic routing algorithms can be divided into two categories, namely the distance vector based routing algorithm and the link-state based routing algorithm. The distance vector based routing algorithm calculates the distance vectors of the links in the network, and then performs routing according to the calculation results. These routing protocols only distribute routing information to neighbour nodes, so these protocols are easier to be implemented and maintained. However, for complex networks, their routing performance show obvious deficiencies due to the limitation of the distance vector. The typical distance vector based routing protocols are IGRP and RIP. In a link-state based routing protocol, each node has a network topology of the entire network, so such algorithms can better adapt to different network environments to make optimal decisions. However, the routers in a link-state based algorithm need to obtain routing information from all other routes regularly by flooding, when the network becomes complex, such algorithms also show significant deficiencies due to the large network burden.

Since machine learning has developed rapidly in recent years, its high adaptability and high performance have quickly attracted the attention of many network engineers. They have designed many dynamic routing algorithms by using the machine learning methods. Among them, the routing algorithms with reinforcement learning were received considerable attention. In the field of machine learning, the reinforcement learning is different from traditional supervised learning and unsupervised learning that require input of training data sets. It can automatically obtain the training samples through the interactions between agents and the environment. Therefore, the reinforcement learning has high biological relevance and high learning autonomy which is very suitable for the network routing [2]. There are seven important elements in the reinforcement learning, namely environment, agent, state, policy, action, reward and observation. The agent is learner and decision maker in the reinforcement learning, and everything that interacts with the agent is included in the environment. These interactions are continually ongoing, the agent makes actions, and the environment responds to these actions by returning rewards and observations, and transferring the agent to new states [3]. A complete agent and environment pair define a learning task, which is an instance of the reinforcement learning, see figure 1.

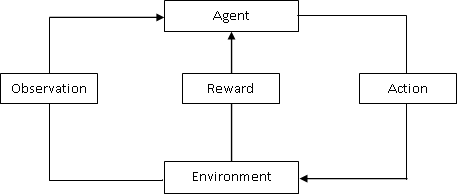


Figure 1: The reinforcement learning process

## Problem statement

The aim of this project is to build a virtual network environment in a Python program and realize a reinforcement learning based dynamic routing algorithm. The specific requirements are shown as follow. Firstly, this virtual network environment should contain several virtual routers to form a network topology. These virtual routers should have basic routing functions such as receiving packets, forwarding packets, queuing packets, neighbour discovery and loop prevention. Secondly, the parameters of these virtual routers should be initialized differently, such as different link capacities, different processing performance, different buffer sizes, different number of ports, and so on. Thirdly, the reward and observation returning behaviour of the network environment should also be implemented, including detecting whether the packet is delivered, whether the packet times out, whether the link is congested, and so on. Fourthly, a network traffic model that balances authenticity and simulation friendliness should also be designed. Fifthly, the implementation of the routing algorithm should be based on a more advanced reinforcement learning method which is called deep reinforcement learning, that is, a reinforcement learning model combined with supervised learning. Finally, the program should also include a statistics module which can record the routing performance of each virtual router and plot a series of routing performance graphs at the end of the simulation.

# Context, Background and Literature Review

## Reinforcement learning in routing

The earliest proposed machine learning dynamic routing algorithm can be traced back more than 20 years ago. Boyan and Littman [5] proposed the prototype of the reinforcement Q-learning routing algorithm in 1994. Kennedy and Eberhart [6] proposed the swarm intelligence algorithm in 1995. Rojas [7] proposed the application of neural networks in network routing in 1996. Gen and Cheng [8] proposed the genetic algorithm in 1999, and Zhang and Fromherz [9] implemented the reinforcement Q-learning routing algorithm for practice in 2006. Forster [10] summarized and compared several machine learning algorithms that were popular at the time in 2007. He stated that the reinforcement learning routing algorithm has smaller network overhead, so the reinforcement learning routing algorithm is more suitable for large scale and energy constrained networks. Yau et al. [11] in 2012 stated that the reinforcement learning can bring context awareness and intelligence to routing algorithms. The context awareness can enable network routers to quickly observe the state of the network environment, while the intelligence can enable network routers to learn the optimal routes by processing the observed network environment information [11]. In other words, the context awareness and the intelligence make the network nodes alive and unlike other traditional dynamic routing algorithms passively acquiring the network information, the routers in the reinforcement learning routing algorithm can actively acquire the network information and update the routing policy more efficiently.

## Reinforcement learning fundamental

### Model-based reinforcement learning

Most of the proposed reinforcement learning algorithms are based on the Markov decision process (MDP). The MDP is a mathematical model of sequential decision that proposed by the Russian mathematician Andrei Markov [12]. It refers to the process in which an agent learns the most rewarding policy in an environment whose states have Markov property [12]. Markov property indicates that the next state of the system is only related to the current state, but not to the previous state [12]. In an environment whose states have Markov property, the set of all states that the agent experiences each time from the initial state to the terminal state is called a Markov process, which is also known as an MDP episode. The set of finite number of Markov processes is called a Markov decision process, that is, learning the policy of obtaining the maximum reward from a finite number of Markov processes. A Markov decision process consists of five basic elements {S, A, T, R, P}. S is a set of all states in the environment; A is a set of all actions that the agent can make; T is a state transfer probability matrix, which describes that the probability matrix of agent moving from one state to another state; R is the reward matrix, which describes the set of all rewards given to the agent by the environment according to the actions of the agent; and P is the current policy matrix of the agent, which describes the set of the optimal actions of the agent for the each state of the environment.

Sutton and Barto [13] formally proposed a complete reinforcement learning theory in 1998, and they defined some basic concepts of the reinforcement learning. Firstly, they defined long-term reward, which is the discount sum of all rewards that the agent will get from the current state to the terminal state, see equation 1

(1)

Where γ is the discount rate and 0 < γ <1, k is the iteration and when the MDP does not have terminal state the k can increase to infinite, and rt+1 is the reward of the current state.

Secondly, with the definition of the long-term reward, other two important concepts were defined, which are the state-value function vπ(s) and state-action-value function qπ(s,a). They respectively describe the expected long-term reward that the agent can obtain from a certain state to the terminal state under the policy π, and the expected long-term reward that the agent can obtain from a certain state to terminal state under the policy π after making a specific action a, see equation 2 and 3.

(2)

(3)

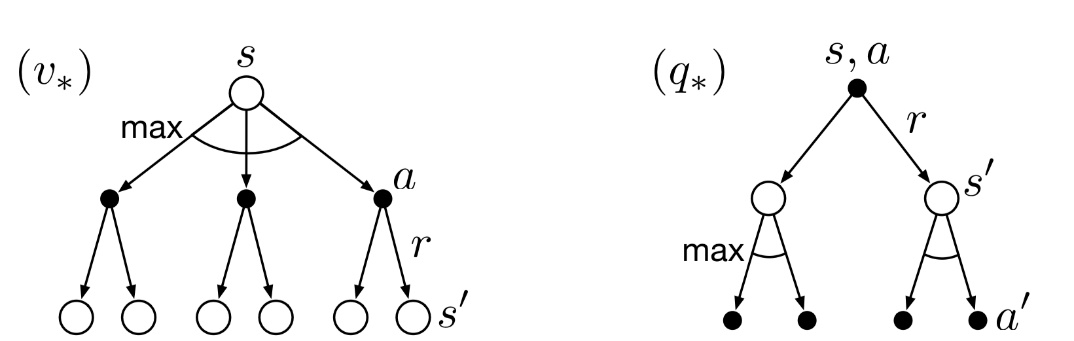
These two value functions are the objective functions of the reinforcement learning. Just like the weights of the neural network constantly updating in supervised learning to minimize the objective error function, the goal of the reinforcement learning is to update the policy to minimize (or maximize) the two value functions. The easiest way to update a policy in reinforcement learning is the iterative method, which requires the implementation of Bellman equations [14]. The Bellman equation was proposed by American mathematician Charlie Berman, which is an important mathematical tool to solve the problem of dynamic programming. It is also an important theoretical basis for many engineering control theories and economic capital pricing theories [14].

Figure 2: The backup figures of value function vπ(s) and qπ(s,a) [14]

Figure 2 are the backup diagram of the state-value function and state-action-value function. The former describes the possible actions the agent can make when it locates at a state S, and the subsequential states to which it may be transferred. The latter describes the possible states to which the agent may be transferred when it locates at a state S after making an action a, and the subsequential actions that the agent may make. The Bellman function can be derived from these two backup diagrams. Firstly, the two value functions can be rewritten as follow

(4)

(5)

where π(a|s) is the probability of the agent that making an action a at the state S under the policy π, is the reward that the environment giving to the agent when agent transfers from state S to state S’, and is the probability of the agent transferring from state S to state S’ after making an action a.

Substituting equation 5 to equation 4, the state-value function vπ(s) can be rewritten as

(6)

Substituting equation 4 into equation 5 the state-action-value function qπ(s,a) can be rewritten as

(7)

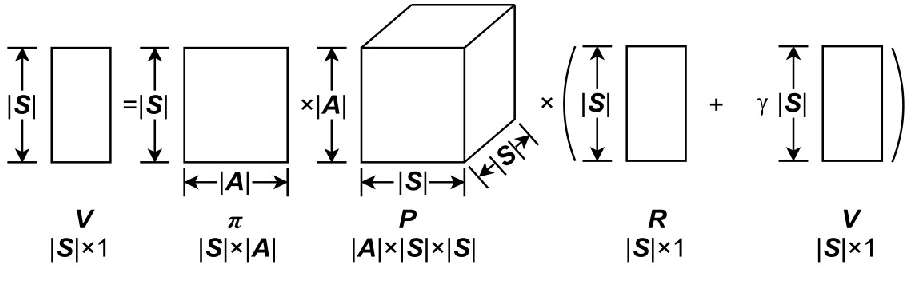
The equation 6 and 7 are the bellman functions in the reinforcement learning. There is an important assumption in MDP which is the time independency. The time independency refers to the value functions of two same states at different instants of times in an MDP will eventually become same with the continuous evolution of the MDP. This assumption is based on the condition that the state transition process of MDP lasts long enough, and eventually each state transition enters a stable state [14]. With this assumption, the Bellman function can be transferred into the form of matrix operation, see figure 3.

Figure 3: The matrix operation of Bellman function of state-value function [14]

Then the Bellman function of state-value function can be rewritten as

(8)

The Bellman function of the state-action-value function can also be rewritten into the from of matrix, then the values of these two value functions can be derived by solving the matrix equations or using iteration method. In general, the iterative methods are easier to be implemented because the matrix equation is always difficult to be solved.

When the value of the state-value function is known, the state-action-value function can be derived, then the policy can be updated as follows

, , (9)

The policy of the agent at state s is the action with the maximum state-action-value function at state s. When the agent gets the new policy, the new value function will be derived again and the policy will also be updated again. It can be proved that the policy will become stable to an optimal policy within finite iterations [15]. This learning method is called the policy iteration method, which is the easiest way in the reinforcement learning. In this method, both the state transition probability matrix and the reward matrix which belong to the environmental information are known. The reinforcement learning model with known environmental information is called the model-based reinforcement learning model.

### Model-free with Monte-Carlo method

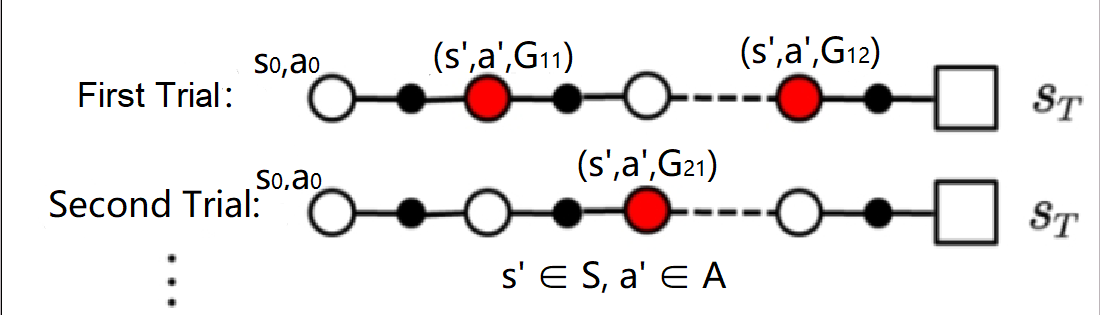
The environmental information in most reinforcement learning instances is unknown, and these instances belong to model-free reinforcement learning model. For model-free reinforcement learning model, the policy iterations cannot be used directly. The Monte-Carlo method is a good approach to the model-free reinforcement learning problems. Because the value function describes the expectation of the long-term reward of a state or a state-action pair, the Monte-Carlo method can be used to estimate the value function. The idea of the Monte-Carlo method is to do trials and use the testing data to estimate the value of the value functions, then the policy iteration method can be applied subsequently.

Figure 3: The trials in the Monte-Carlo method [16]

Figure 3 is the trials in the Monte-Carlo method, the agent firstly locates at a random state in the environment and makes an action under the current policy, then the agent will be transferred into a new state by the environment. The agent continuously moves in the environment until the agent enters the terminal state, which is called a trial sequence. When the model gets an enough number of trial sequences, the long-term rewards of each state-action pairs that appear in the trial sequences can be calculated, then the average values can be calculated as the estimations of the value functions of these state-action pairs, see equation 10.

𝑎’∈𝐴, s’∈𝑆 (10)

Where N(s’,a’) is the Nth appearance of the state-action pair (s’,a’) in the trial sequences.

When the model gets the estimation values of all state-action pairs, the policy can be updated as follow

𝑎∈𝐴, s∈𝑆 (11)

Where ε is an exploration rate that 0< ε<1. The policy updating in the model-free learning is ε-greedy method rather than the fully greedy method in the model-based learning. This means that for a state the action with the largest state-action-value function in the policy has a larger probability of being selected and other actions of this state have an identical smaller probability of being selected. The purpose of the ε-greedy is that the model needs to be explorative to all state-action pairs, otherwise there will be some state-action pairs that may never be experienced by the agent.

The Monte-Carlo method is a good approach, but it is not used commonly in most cases. The Monte-Carlo method is not efficient because it always requires full trial sequences to calculate the average value of state-action-value functions.

### Model-free with Q-learning

The Q-learning is a more efficient way used in the model-free problems. The Q-learning also does trials, however, the value estimation of the state-action-value function in the Q-learning can be performed simultaneously with trials. The average of the state-action-value function in the Monte-Carlo method can be rewritten into the incremental form as follow

(12)

where k is the kth appearance of the state-action pair (s’,a’) in the trials. Then 1/k in the equation 12 can be replaced by a constant α, see equation 13

(13)

The equation 13 is the moving average of the state-action-value function. Then the state-action pair (s’,a’) can be replaced by (st,at).

(14)

Next, the long-term reward G(st,at) in the equation 14 can be divided into two parts, the one-step reward of state-action pair (st,at) and the maximum state-action-value function of the next state-action pair (st+1,at+1), see equation 15

(15)

1. **Results**

Text here, the entire section should not be more than 30 pages. Further results of secondary importance can be included in Appendices at the end of the dissertation.

* 1. **Methodology/Design Description**

Text here

* 1. **Analysis of Results**

Text here

1. **Conclusions**

Text here, should not be more than 2 pages.

**References**

(no limit on the number of references)

1. A. Smith and B. Jones, “Method to derive a reference list,” *Journal of References*, vol. 10, no. 3, pp. 34-38, Jan. 2010.
2. A. Smith, B. Jones, and C. Watson, “An improved method to derive a reference list,” *Proc. Internat. Conf. on References*, ICR 2011, pp. 45-50, Feb. 2011.

**Appendices**

No limit to the number of pages.

Add appendices with additional results, proofs, etc., as appropriate to your project. Listing of source codes is discouraged unless the particular portions of the source code are critical to the development of your project and they are discussed in the main part of your text.