Undergraduate Project Report
2021/22

Program Optimisation Using Reinforcement Learning

Name:	Xiao Zheng
School:	International School
Class:	2018215107
QMUL Student No.:	190014607
BUPT Student No.:	2018212779
Programme:	Telecommunications Engineering with Management

Date: 20-04-2022

Table of Contents

Abstract

With the rapid development of the Internet economy, more and more people choose to join the Internet industry and become programmers. However, the massive expansion of programmers is bound to lead to the decline of the average level, resulting in the problems of chaotic format, low efficiency, excessive memory occupation and so on. Although many enterprises have carried out necessary training for employees, manual code optimisation is still very time-consuming and laborious. In addition, many integrated development environments also have embedded code optimizers. However, these code optimizers only provide guidance on program optimization according to big data and do not make tentative modifications and give targeted suggestions for the program itself. At the same time, the application of artificial intelligence is also developing rapidly. There are also many research results on reinforcement learning and machine learning, a classification between supervised learning and unsupervised learning. Therefore, I want to combine the two, and how to use machine learning, especially reinforcement learning, to automatically modify and optimize programs and codes has become an urgent problem to be solved.

Starting from deep learning, this report explores the use of two value-based reinforcement learning algorithms, Q-learning and SARSA, to optimize the program based on specific optimization methods. In the process of exploration, a successful modelling method based on the program line and status is confirmed, and the program optimization of runtime is successfully realized through the return function based on the runtime. In addition, based on the experimental scenario of adding the register modifier in front of variables and training for a short program, Q-learning has an advantage of more than 80% by comparing the convergence speed of Q-learning and SARSA algorithms for each episode length and return per step. Finally, based on this experiment, this report also verified several suggestions that help to optimize the program.

摘要

随着互联网经济的迅速蓬勃发展，越来越多的人选择了加入互联网行业成为程序员。然而，程序员的大量扩张势必造成平均水平的下降，导致程序员的代码出现格式混乱、效率低下、占用内存过大等问题。尽管现在有许多企业对员工展开了必要的培训，但是手动进行代码优化还是非常费时费力。除此之外，许多集成开发环境也有内嵌代码优化器，然而这些代码优化器只会根据大数据对程序优化提供指导意见，并没有针对程序本身进行尝试性的修改并给出针对性的意见。与此同时，人工智能相关应用也在火速发展，关于强化学习，机器学习下一种介于监督学习和无监督学习中的分类也有众多研究成果。因此，我想把二者结合起来，而如何利用机器学习，尤其是强化学习对程序和代码进行自动修改和优化就成为了一个亟待解决的问题。

本文从深度学习出发，探究利用 Q-learning 和 SARSA 两种基于值的强化学习算法，基于特定的优化方法对程序进行优化。在探究的过程中确认了一种基于程序行和程序状态的成功的建模方法，并成功通过基于运行时间的回报函数实现了对运行时间的程序优化。另外，基于以在变量前添加 register 修饰符为行动，针对一段简短程序展开训练的实验场景，通过比较 Q-learning 和 SARSA 两种算法对于每代长度和每步回报的收敛速度，Q-learning 拥有超过 80%的优势。最后，基于本实验，此报告也验证了几条有助于优化程序的建议。

Chapter 1: Introduction

Admittedly, enormous codes and programmes are being generated by programmers every day and every minute. We write codes to assist us almost everywhere, from doing repetitive and boring statistical tasks to exploring possible answers to complicated questions and circumstances, such as Go or dynamic programming. Also, numerous Internet enterprises require numerous programmes to conduct business. Therefore, in such a large-scale application scenario, a high-quality and effective programme is especially important, which could not only cut down processing time but also save resource costs. However, the reality is that not all programmers have the ability and energy to improve their code, so a tool that can help programmers optimise their code is urgently needed.

At the same time, machine learning, especially reinforcement learning, is also a hot research topic in today's era. At present, many product-level applications of reinforcement learning have been realized, such as Horizon, an open-source platform for Facebook application-based reinforcement learning (Gauci et al. 2019), and decision service, a decision service application developed by Alekh Agawal and others (Agarwal et al. 2016).

Based on the above two points, I came up with the idea of using reinforcement learning to optimize code. Of course, there are many mature cases of code optimization. Many integrated development environments, such as Microsoft vs code, IntelliJ and PyCharm, have built-in code analysis optimizers. In addition, DeepMind has also launched a commercial advanced code analysis system for enterprise users. However, on the premise of big data, these applications use data to deconstruct the code and put forward optimization suggestions. They do not generate targeted optimization schemes by comparing the effects before and after optimization for a single programme.

In this project, I try to use Q-learning, a value-based reinforcement learning tool, to analyse and optimize the sample program based on some pre-defined improvement methods and finally generate a state operation comparison table (Q table), so that human or artificial intelligence can use this table to optimize code and improve operation efficiency.

This report is made up of five chapters. The first chapter is Introduction, and in the second chapter, Background, this report will provide some necessary background information and theory related to the project, including machine learning, supervised and unsupervised learning, reinforcement learning, and Q-learning and SARSA. In addition, this report will also mention several recent achievements revolving around reinforcement learning and programme optimisation in this chapter. Then in the third chapter, Design and Implementation, this report will display my reinforcement learning environment and the procedure of developing and training the artificial intelligence. After that, in the Results and Discussion chapter, this report will analyse statistics and outcomes generated in the experiment, and then discuss them and draw some conclusions. Finally, in the last chapter, Conclusion and Further Work, this report will summarize the whole project and provide some useful prospects.

Chapter 2: Background

2.1 Machine Learning

Machine learning is one of the most significant recent developments in artificial intelligence whose basic concept is, that we do not teach the machine how to do something, but instead, we expect it to learn by itself. In this way, the machine can learn how to deal with complex tasks directly from experience or data.

Generally, in machine learning systems, computers are usually trained in a large database of the same task, and then write their code to perform one of the tasks. A large part of this involves identifying patterns in these tasks and then making decisions based on these patterns.

For instance, let us say if a company wants to hire new staff, and 1,000 interviewees have sent their applications and resumes. This amount is too much for the interviewers if they wish to screen one by one. Hence, you want to train a machine to complete this task. In which case, you need to record the resumes of many past candidates, and for each of them, you have had a record of whether the person was finally hired. In order to train the machine, you decide to take out half of your resumes and let the machine find out the pattern by learning whether these resumes have successfully applied for a job. In this way, when the machine receives a resume, it can judge whether the person is suitable for employment. After training, you can test the machine with the other half of your resumes (samples). If the test result shows a high enough success rate, that is, the probability of the machine making correct judgement is high enough, then you can safely announce that the machine is applicable to judge whether an interviewee is a right person for employment from his or her resume. There is no need for human judgment at any stage.

2.2 Supervised Learning and Unsupervised Learning

The example we talked about in the previous section is a typical application of supervised learning, where there is always a teacher, or we could say a supervisor, who holds the answer for each test case and would evaluate each action taken by the machine and provide feedback which is a score or Boolean value in most cases. Supervised learning is a basic branch of machine learning, and its training speed has great advantages over unsupervised learning.

However, in real life, there are often such problems: lack of sufficient prior knowledge, so it is difficult to label categories manually, or the cost of manual category labelling is too high. Naturally, we hope that the computer can complete these tasks for us, or at least provide some help. Solving various problems in pattern recognition based on training samples with unknown categories (not labelled) is called unsupervised learning. This situation often occurs when the data set is too large (over millions) for us to classify manually, as well as there might be some samples too hard for us to tell exactly which category they belong to, but only to provide a possibility.

Here we are talking about clustering. Let us take image classification as an example.

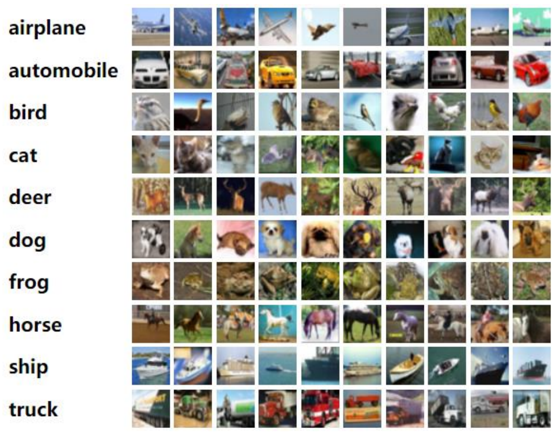


Figure 2.2.1 The CIFAR-10 Dataset

Figure 2.2.1 is the famous CIFAR-10 dataset, which is labelled, of course. In supervised learning, we put all the pictures with their labels into the machine and train, then the machine will learn the characteristics of each category and be able to classify new input images into those 10 types. However, if we clear all the labels, then this task will become unsupervised learning. The machine will no longer be restricted to the 10 defined clusters, but instead, it will try to form new clusters according to a standard, such as the Euclidean distance in the characteristic domain. In which case, the machine just finishes the task, but we could not tell exactly what these clusters represent.

Essentially, unsupervised learning is a statistical method that can find some potential structures in unlabeled data. Besides clustering, dimensionality reduction is also a common application of unsupervised learning, where the machine studies all statistics and finds out a way to cut off redundant information.

2.3 Reinforcement Learning

Reinforcement learning generates from unsupervised learning, but it does not belong to unsupervised learning. Similar to unsupervised learning, there is no label at first, but there is no dataset either. All we have is rules. The machine starts from nothing and takes one step, then it applies the rule to the current state and gets an evaluation or a score. After that, the machine takes more and more different steps and receives corresponding feedback. For that positive feedback, the machine will increase the weight of corresponding actions, and for that negative feedback, the machine will do the opposite. All the machine intends to do is to discover actions that can bring a higher reward and take them, which has a closer relationship with human beings than supervised and unsupervised learning. When we were babies, our parents would praise and award us for good behaviours and would scold and punish us if we did something bad like breaking vases.

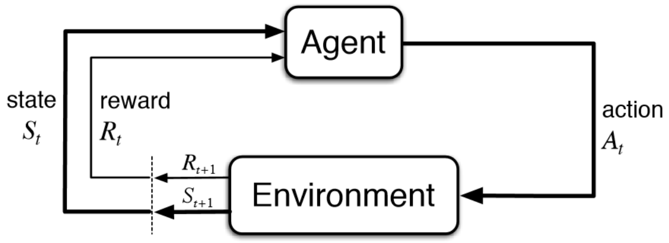


Figure 2.3.1 Reinforcement Learning State Transition Diagram

Figure 2.3.1 shows a basic state transition diagram of reinforcement learning. The agent first observes the environment and selects the appropriate action according to the obtained state information. Then the environment receives the action, makes corresponding feedback, and enters a new state at the same time. Finally, the agent obtains the reward from the environment and adjusts to the next action. Actually, this mathematical model is called the Markov decision process, which is the basis of reinforcement learning.

2.4 Q-learning and SARSA

Here we are talking more about reinforcement learning. The next question in reinforcement learning is, how do we define rules that calculate rewards? There are three kinds of approximate reinforcement learning algorithms: Value-based, Policy-based and Actor-Critic-based (Si et al. 2022). The value-based algorithm, as its name indicates, used linear or nonlinear functions to parameterize the value function. However, for most basic algorithms in this category, their characteristic basis function needs to be manually selected, which is highly dependent on prior knowledge and has certain limitations. Policy-based approximation method parameterizes the policy function, estimates the gradient of the objective function relative to the policy parameters, and then optimizes the parameters by using the gradient rise algorithm to obtain the optimal or local optimal policy. This method is suitable for reinforcement learning problems with continuous action space, but it belongs to round-based updating, and the learning speed is slow.

Moreover, in the process of gradient estimation, the instability of strategy updating caused by large variance is difficult to avoid. The actor-Critic learning framework estimates the value function and strategy function at the same time. Different from the simple strategy gradient algorithm, the parameter update of the Actor part no longer uses the Monte Carlo method to estimate the value function but obtains the approximate value function from the Critic part.

Q-learning is a value-based algorithm and one of the most important reinforcement algorithms, which is proposed by Watkins in 1989 (Watkins and Dayan 1992).

```
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., ε-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$ ;
    Until $s$ is terminal
```

Figure 2.4.1 Q-learning Pseudo Code

The initial Q-learning applies a greedy strategy in the decision-making process, which is in the fifth line in Figure 2.4.1. This strategy always chooses the best action according to the current state-action value, but experiments turn out that this strategy cannot get a satisfactory outcome and arrives at a local optimum easily. Hence, to force the machine to avoid the local optimal trap, the ε-greedy strategy is applied, where the machine is permitted to apply actions that seem not the best at the moment for exploring more possibilities. However, since ε, the exploration probability is fixed, with the knowledge agent get is more and more precise, the performance of the system will decrease if we still decide to take a large number of exploration activities.

```
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action $a$, observe $r$, $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$ ;
        $a \leftarrow a'$ ;
    Until $s$ is terminal
```

Figure 2.4.2 SARSA Pseudo Code

The state-action-reward-state-action (SARSA) algorithm develops from Q-learning, but it is different from Q-learning because SARSA is an on-policy algorithm. Figure 2.4.2 shows the pseudo-code for SARSA, and we can discover that, different from the Q-learning algorithm in

Figure 2.4.1, the action that SARSA observed to update its Q value will certainly become the next action, that is, SARSA will always do as it observes. In contrast, Q-learning, which is an off-policy algorithm, is greedy and always seeks actions that bring the most reward, ignoring the dangers.

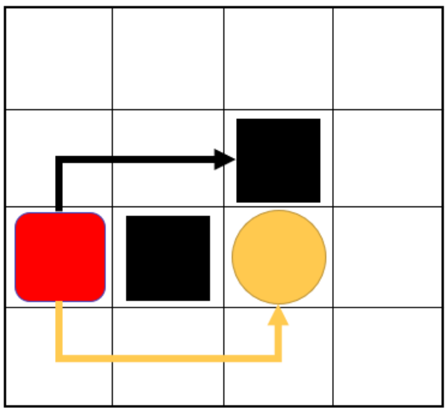


Figure 2.4.3 A Maze Problem

I would prefer to use the example of solving the maze to illustrate the concept of Q-learning. In Figure 2.4.3, there is a maze problem. The rounded rectangle represents the agent, and its task is to attempt to reach the circle which represents the goal (with positive reward) but avoid stepping into the rectangle which represents the trap (with negative reward). Each cell in the table stands for a state. At first, the agent will explore the maze randomly. However, after some time (episodes), when the agent arrives at the state which is shown in Figure 2.4.3 again, it will probably select to follow the path below rather than above, as the knowledge stored in the brain tells that the path below would bring a larger mathematical expectation (Q-value). Also, since Q-learning is greedy, the Q-value on the current state would be updated based on the action of going downwards, which is the most valuable next action.

Starting from Q-learning, researchers have developed many progressive reinforcement learning algorithms, theories and discoveries. Jang et al. (2019) summarized the development of Q-learning algorithms and categorized Q-learning algorithms into single and multi-agent approaches. Guo et al. (2002) developed a Q-learning algorithm based on the Metropolis criterion, which is called SA-Q-learning and shows a faster convergence speed than traditional Q-learning, and also succeeds in avoiding the degeneracy in performance due to excessive explorations. Cho et al. (2007) proposed an influence map called QIM to reduce the time needed for the learning phase. Marco and Juergen (1998) introduced TD(λ)-methods into Q-learning and formed a new Q-learning algorithm named Q(λ)-learning, which is actually an online reinforcement learning method with linear average complexity in the number of actions.

2.5 Program Optimisation

As for program optimisation, there are also many researchers managed to find ways to improve programs from different aspects. Liu and Qiu (2014) proposed the program business logic adjustment from database optimisation which brings a remarkable optimisation effect. Amouzegar (1999) put forward a new global optimisation method, through restating the nonlinear bilevel programming problems into a global optimisation problem, to help deal with nonlinear two-level programming problems. Du and Zhao (2010) focused on integer programming problems, and they mentioned multi-agent optimisation algorithms and DNA calculation to provide a summary of the intelligent algorithm in integer programming.

In addition, other than new algorithms, there are also lots of other optimisation methods. Ni (2009) mentioned 7 optimisation methods in C programming, including using register variables, replacing array subscript with pointer, bit operation, mathematical optimisation, and so on. Cai (2018) explored embedded C Program Design, and he wrote that the flexible use of assembly and C mixed programming can optimize the program. Yang et al. (2009) started from DSP, utilizing intrinsics to substitute complex C and C++ codes and enhancing code performance significantly from the processor level. Yang and Zhang (2005) sought solutions for Visual Basic 6.0, and they drew some conclusions like using more constants instead of Variant variables can both speed up the execution of the program and reduce memory usage. Yin et al. (2019) found a way to represent the salient information of an edit and can be used to apply edits to new inputs.

Chapter 3: Design and Implementation

3.1 Environment

This project is designed on Windows 10 Professional Edition, version 21H2 with Intel® Core™ i5-8250 CPU. The integrated design environment used in this project is PyCharm 2021.3.3 Community Edition, and the program is tested on Python 3.9.7 on win32.

The input programs, also the programs that need to be modified are written in C since C is simple and basic. The agent, in other words, the machine is written in Python, since Python is the best language for machine learning and there are lots of examples showing how to use Python to generate an artificial intelligence.

In late phases, in order to cut down the time of the training process, the machine is trained on a virtual machine provided by Amazon Web Service. The virtual machine is based on Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1025-aws x86_64).

All the programs and codes used in this project can be fetched from my GitHub Page (zhengx-2000 2022).

The reinforcement learning environment is developed from OpenAI Gym but is not imported into the Gym package. The Q-learning brain program and execution program are modified from the tutorial lesson on Morvan Zhou's GitHub Page (Morvan Zhou 2020).

3.2 Reward Function

Although in the traditional Markov decision process, the order of elements should be state, action, state transition probability matrix, and finally reward, here we disrupt the order of them for the convenience of facilitating the description of the design process.

We talk about reward function in the first place. The reward function describes the feedback that the environment returns to the agent after a certain action. So, setting the reward function must depend on what we want to teach the agent. In the early term of this project, I carried out three useful indications that can describe how well an optimisation method supplies to the program, which are runtime, memory space and program size. However, it is hard to accurately measure the memory space used, and each time we execute the same program, the memory space might have different values. In addition, program size seems not so urgent in daily life, and many large files cannot be separated or reduced for necessary reasons like modularization. Hence, I only take runtime as the factor of the reward function in this project. Because we hope to encourage actions that reduce the runtime and punish actions that do the opposite, the reward function is defined as follows: reward equals the proportion of time that reduces by the action, concerning the runtime of code executed in the last action.

For example, if the runtime of the program (code) in the last step (action) is 0.8 seconds, and after modifying the program, the runtime has become 0.5 seconds, then we can define the reward of the current action as (0.8-0.5)/0.8, which is about 0.375.

Hence, in which case, if a modification does not decrease the runtime, but instead increases it, then the reward would be negative, and the agent receives punishment.

3.3 Pre-experiment

To find out what methods are appropriable for program optimisation, we need to do some pre-experiments. I gathered five action clusters and wrote a test program to see whether these actions are useful (zhengx-2000 2022). Since in Section 3.2 I mentioned that runtime is the only standard for checking the optimisation effect, in the test program, each input program was compiled and executed 1000 times and outputted an average execution time.

Table 3.3.1 Summary of the Pre-experiment

Action Cluster	Description	Result of Improvement
1	Using more additions instead of multiplications	0.6%
2	Avoiding too many dependencies and jumps inside the loop	0.03%
3	Using different For loop structures	0.3% in maximum
4	Adding register modifier before variables	86.4%
5	Using different types of variables	69.7% in maximum

Table 3.3.1 is the summary of the whole pre-experiment, and we will talk about them in the following part.

3.3.1 Cluster 1: Using more additions instead of multiplications

In traditional theory, the operation time of multiple additions is lower than that of multiplication.

In this experiment, I constructed a 1920×1080 two-dimensional array and tried to use a double loop to calculate the sum of the array. In the improved program, I added a pointer to the outermost loop to decompose one of the multiplication operations.

The result shows that this operation can improve the runtime by about 0.6%.

3.3.2 Cluster 2: Avoiding too many dependencies and jumps inside the loop

When the compiled program is sent to the CPU for execution, the CPU can automatically apply a pipeline structure to deal with loops parallelly to some extent. However, if the calculation or logistics inside the loop is too complicated, the CPU is unable to optimise the execution process. In this experiment, I nested a complex if-else statement inside a For loop, while in the improved program, multiple For loops were used to solve the conditional judgement.

The result shows that this operation can bring about a 0.03% gain in runtime.

3.3.3 Cluster 3: Using different For loop structures

Some Internet materials show that using different For loop structures may affect the running speed of the program. For example, for the count parameter "i" in the loop, using "i++", "++i", "i--", and "i+=1" will bring different results. In this experiment, I used the above four-loop structures to complete the same task of self-multiplication.

The result shows that "i+=1" takes the most time, while "++i" takes the least, and the proportion of improvement of runtime from "i+=1" to "++i" is about 0.3%.

3.3.4 Cluster 4: Adding register modifier before variables

Register variables are stored in the register, so they can be extracted and modified faster. However, if the register is occupied with too much data, then the overall execution time might be slowed down somehow. In this experiment, I designed a task of calculating the sum exponentially, and in the optimised program I added the register modifier before all variables.

The result shows that after adding the register modifier, the mean execution time was cut down by 86.4%.

3.3.5 Cluster 5: Using different types of variables

There are plenty of variable types in C: int, double, float, long, short, unsigned int, unsigned long, unsigned short, etc. Different variable type occupies different memory space and may affect the execution time. In this experiment, all the programs are required to complete addition, subtraction, multiplication and division thousands of times, but the variables used for calculation are different. For consistency, all input numbers for calculation are integers.

The result shows that the largest optimisation is about 69.7%, which happens between Unsigned Long and Double.

After the pre-experiment, I finally decided to choose cluster 4, adding register modifier, to become the action implemented in the final program, for its explicit optimisation effect and convenience for modification.

3.4 Action

Action defines how the agent transfer between states. From Section 3.3, the only action I need to implement in the reinforcement learning framework is adding “register” before specific variables. If we treat the program as a text file, then the operation can be seen as adding a word and a space before another word. Thus, the implementation becomes relatively easy, and all we are required to do is detect the target string and replace it with a new string that contains a “register”.

The next question is, what if many matched results can be modified, how should we determine which result is the one that the agent wants to modify? In this project, I assume that there is only one matched result in one line, which is consistent with the design specification, where two declaration statements would never appear in the same line. Even if this situation happens, the modification would be applied to all matched results in the same line simultaneously. Therefore, the modification action is accompanied by a line number in this project, and when the environment receives a modification action, it would search in the specific line and try to modify.

What if the agent does not want to modify? We must provide an action that allows the agent to travel to the next state without doing anything, which is called the skip action in this project.

3.5 State

The state is where the agent exists, in other words, it is the observation of the current situation. All the actions that the agent make are based on the state, so it is significant to opt for the most outstanding characteristics that represent the state to help make the decision. In Section 3.4, the modification action contains the line number information. So, it would be best if we add the line number into the state. What is more, if the program has been successfully modified, the next state we observe will be based on the new program. Thus, the state must also contain information for the program being observed, which is implemented by adding modification records into the state. To summarize, the state is made up of two parts. The first part contains the modification result which represents the program being observed, while the second part contains the line number which implies the line where the agent observes for modification.

In Q-learning and SARSA, we always update the Q-value based on the next state. But for some states which are called terminals, there is no next state. Hence, the next state for terminals is defined as themselves. We will talk more about terminals in Section 3.6.

3.6 Special Cases

Table 3.6.1: Dealing with Cases

Situation	State	Reward	Information
Modification Failure	No Change	0.0	None
Modification Success but with Syntax Error	Withdraw Modification	-1.0	Terminal
Modification Success but with Test Error	Withdraw Modification	-1.0	Terminal
Modification Success but with Overtime	Withdraw Modification	-1.0	Terminal
Absolute Success	Update	Proportion of Improvement	None
Goal (optional)	Update	Proportion of Improvement	Terminal

Because we directly modify the code in text form when we are making modifications (actions), we have to check the availability of the program before we try to run the program and measure the runtime. In this project, there are six different cases after the modification action that need to be taken into consideration and treated differently.

The first case is called modification failure, which means that after scanning the whole line, we cannot find a matched piece of code that can be applied with modification. For instance, if the agent wants to replace ‘int’ with ‘double’ in a line that does not contain ‘int’, then it is for sure that this action is invalid. The state would remain the same since no change is made, and for reward, we would give it zero as this action should be neither encouraged nor punished.

Next, if we successfully execute the action and modify the code, the next question would be whether the new program is runnable. It is hard for us to examine syntax errors, but the compiler can. Hence, we simply try compiling the new program and check the return value by utilizing the Python function 'subprocess.call()'. For this situation, we have to withdraw the action and return to the previous state since the agent arrives at a terminal and no further action is permitted. In addition, we decide to give a negative reward to reduce the possibility of such unwelcome action.

Then, even if the modified program passes the compile test, we must continue to test the output value. When we try to modify the program structure, it might give an incorrect output. In which case, we treat it the same as compile error, withdraw the modification and apply a negative reward (-1.0).

After that, it is about overtime. If the program takes too much time to execute, then it may fall into a dilemma like an infinite loop. In addition, overtime slows down the training speed for each episode. Thus, in this project, I set an upper limit for the time used while executing the test program, which is 200% concerning baseline (the original program). Similarly, this situation is treated the same as syntax and test errors.

Finally, after detecting three possible errors and no error is found, we can say this attempt of action is an absolute success and update the state, assign the reward as Section 3.2 states and go on with the next action.

The last situation is about the goal, which is optional but applied in this project. A goal is able to be set if and only if we know what the ultimate anticipation is. In this project, I have done pre-experiments in Section 3.3 and know what the goal state is, so I can halt the current episode and avoid future useless attempts. But for most situations, we do not know the answer, so the agent will keep exploring even if it has reached the ideal state.

You may ask why we need a goal state. If no goal state is set, the agent will keep exploring in the environment even if it has reached the ideal state, where any other more actions will bring a zero or negative reward. In such case, if we do not halt the agent, the negative reward in the next step may influence the reward of the goal state, disturbing agent's judgement. Furthermore, the efficiency of the training process will be lowered down, and the charts I am going to show in Section 4.4 will be totally different.

But determining the goal state is also a tough task. In Section 2.4, I compared the reinforcement learning environment to the maze problem. However, the environment in this project is different from the maze. In a maze there is only one certain goal, and any other state generates zero or negative reward. But when it comes to the program optimisation, each successful modification will bring a reward, whose value is determined only if we take the change and test it, and the value varies each time we measure it! Thus, the best choice for us to take is to do abundant pre-experiments and find the ideal state according to the Q-table (state transition probability matrix) generated in the last episodes. The result of this pre-experiment will be displayed in Section 4.2.

3.7 Program Structure

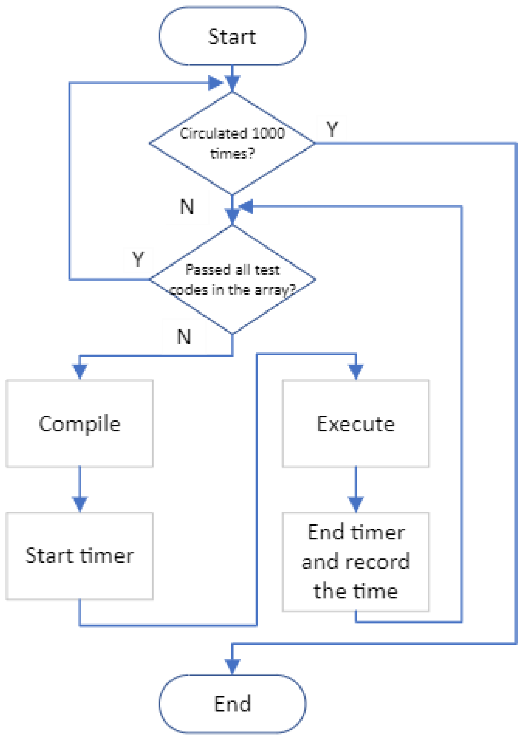

Start
Circulated 1000 times?
Y
N
Y
Passed all test codes in the array?
N
Compile
Start timer
Execute
End timer and record the time
End

Figure 3.7.1 Pre-experiment Flow Chart

Figure 3.7.1 exhibits the flow chart of the test program in the pre-experiment. Each of the input programs is compiled and executed 1000 times to guarantee the correctness of the result.

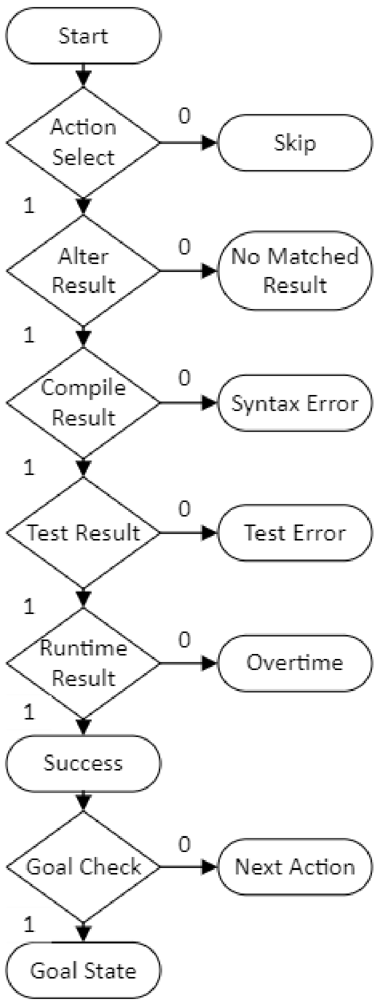


Figure 3.7.2 Core Judgement Sequence in Step() (action function)

Figure 3.7.2 concludes how the program deals with special cases in Section 3.6. When the environment receives an action, it will go through six judgement statements in total to determine the reward and the next state.

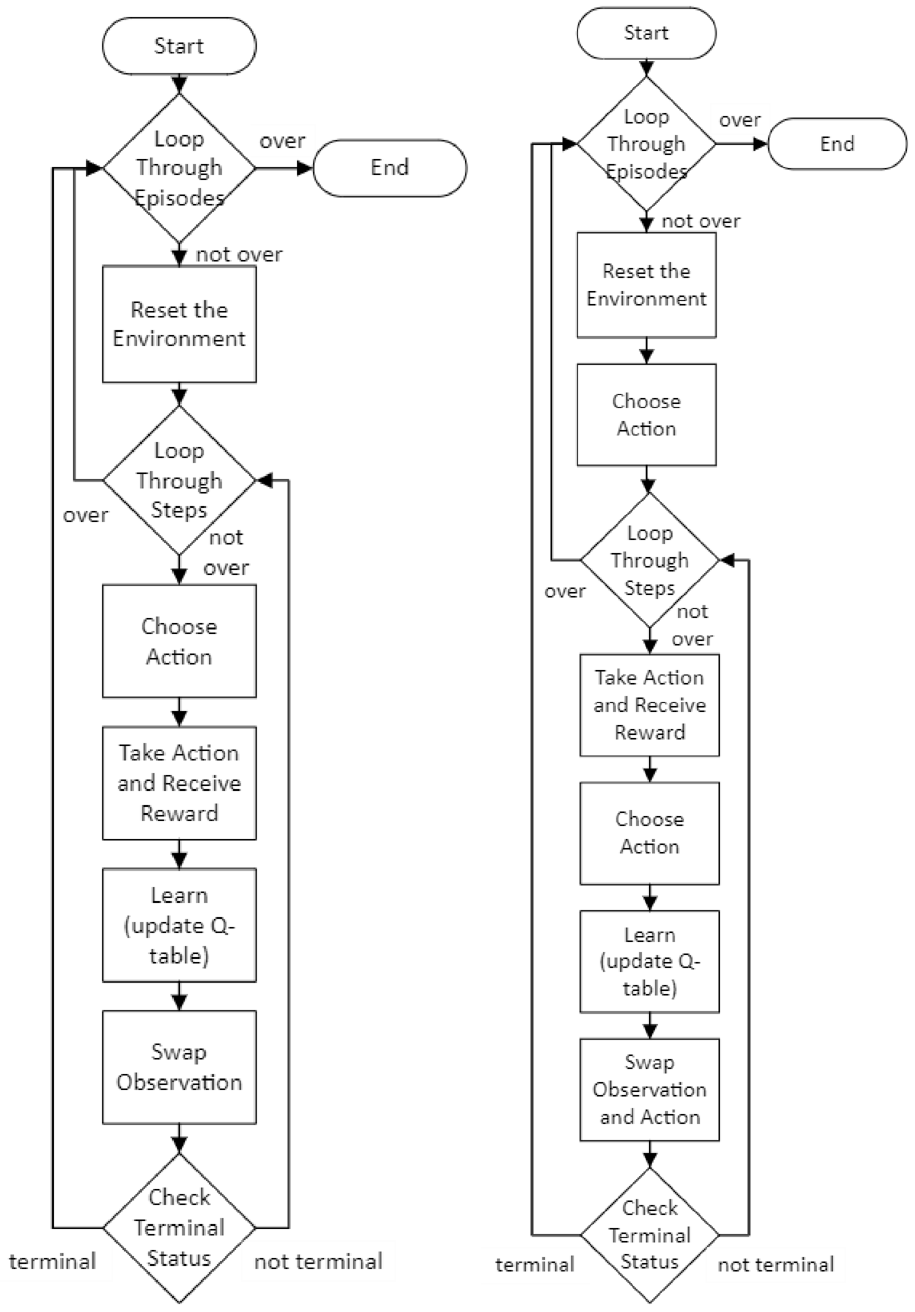
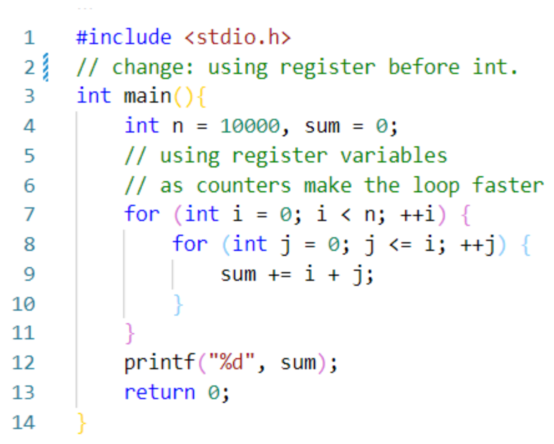


Figure 3.7.3 Work Process of Q-learning and SARSA

Figure 3.7.3 tells the training process of Q-learning and SARSA in this experiment. Q-learning on the left only stores the observation information, and the next action it will choose in the next step is based on the new Q-table and may not be the one it learns. However, the chart of SARSA on the right shows how State-Action-Reward-State-Action works, and the action SARSA chooses and learns will be the action it takes in the next step.

Chapter 4: Results and Discussion

4.1 The Input Program



```c
#include <stdio.h>
// change: using register before int.
int main(){
    int n = 10000, sum = 0;
    // using register variables
    // as counters make the loop faster
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j <= i; ++j) {
            sum += i + j;
        }
    }
    printf("%d", sum);
    return 0;
}
```

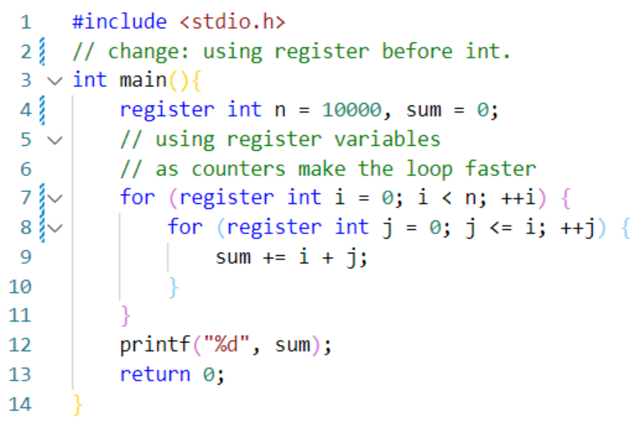
Figure 4.1.1 Input Program Register_copy.c

Figure 4.1.1 shows the input program in the reinforcement learning framework, which is also the one we tested in the pre-experiment in Section 3.3.4. As is stated in Section 3.4, for convenience, we define the action in the reinforcement learning environment as adding the register modifier before variable statements (int).

4.2 State Transition Probability Matrix

Table 4.2.1: Q-table Result for Register_copy.c (part, without goal state)

Action State Reward	**0**	**1**
7(3,6)	0	0.812716017
6(3)	0.040594637	0.690985057
3	0.0000220875	0.411040297
6(3,7)	0	0.40244213
3(7)	0	0.270202244
7(3)	0	0.240524897
7	0	0.039203263
7(6)	0	0.004556588
6(7)	0	0.003248167



```
1   #include <stdio.h>
2   // change: using register before int.
3   int main(){
4       register int n = 10000, sum = 0;
5       // using register variables
6       // as counters make the loop faster
7       for (register int i = 0; i < n; ++i) {
8           for (register int j = 0; j <= i; ++j) {
9               sum += i + j;
10          }
11      }
12      printf("%d", sum);
13      return 0;
14  }
```

Figure 4.2.1 The Goal State

Table 4.1.1 exhibits a part of the Q-table result (the state transition probability matrix) without setting the goal state. For convenience, only the positive results are listed here. The row represents the state, where the first number indicates the current line number being observed and the other number in the parentheses indicates the present program state, where "3,6" means the state after making modifications to line indexes 3 and 6, which are line 4 and 7 in real line number. The column represents the action, where "0" stands for skipping (line index +1) and "1" stands for change, where the current line index is added to state if modification success. The data in the table is the reward expectation after taking such action in such a state. For instance, the reward expectation for state 6(3) and action 1 is about 0.691, which means that modifying line index 6 in the program after having successfully made a modification in the line index 3 may bring a reward of 0.691.

From Table 4.2.1, we know that making modifications in line index 3, 6 and 7 gets the most of rewards, so the goal state is set to Figure 4.2.1, where three register modifiers are added to the program.

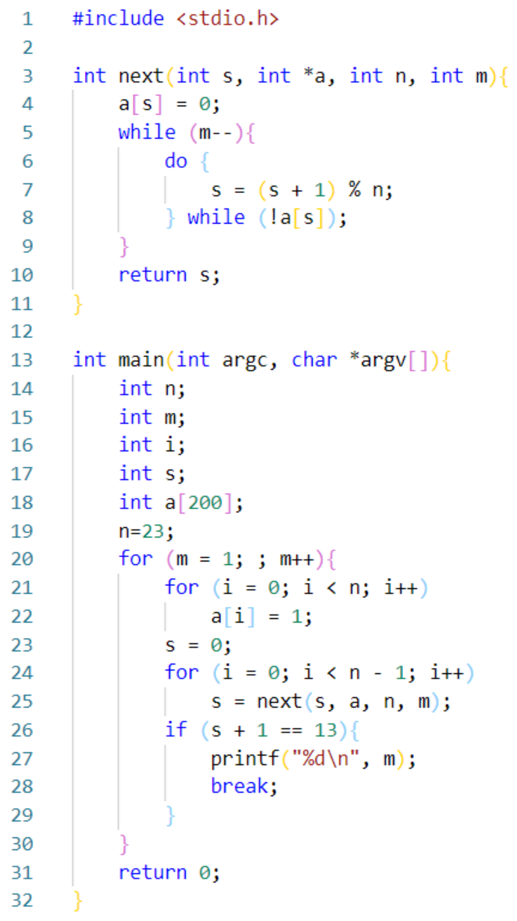
Table 4.2.2: Q-table Result for Register_copy.c (part, with goal state)

Action / State / Reward	**0**	**1**
0	0.007804	0.000108
1	0.019979	0.000387
2	0.041704	-0.08648
3	0	0.071095
3(3)	0.006085	-0.10466
4(3)	0.021464	0.000156
5(3)	0.065752	0.001038

6(3)	0.001803	0.173146
6(3,6)	0.337665	-0.07726
7(3,6)	0	0.636504
7(6)	0	0.001732
2(3,7)	0	-0.0199
5(3,7)	0.000217	0
6	0	-0.00024
7	0	0.002987

Table 4.2.2 is a part of the Q-table result with setting the goal state after training. By reading it, we can draw the conclusion that adding the goal state will not influence the final result, and reinforcement learning can really help optimize the program.

4.3 Program Comparison



```c
#include <stdio.h>

int next(int s, int *a, int n, int m){
    a[s] = 0;
    while (m--){
        do {
            s = (s + 1) % n;
        } while (!a[s]);
    }
    return s;
}

int main(int argc, char *argv[]){
    int n;
    int m;
    int i;
    int s;
    int a[200];
    n=23;
    for (m = 1; ; m++){
        for (i = 0; i < n; i++)
            a[i] = 1;
        s = 0;
        for (i = 0; i < n - 1; i++)
            s = next(s, a, n, m);
        if (s + 1 == 13){
            printf("%d\n", m);
            break;
        }
    }
    return 0;
}
```

Figure 4.3.1 Input Program min_c.c

Table 4.3.1: Q-table Result for min_c.c (part, without goal state)

State \ Action / Reward	**0**	**1**
13	0.037249586	1.486037897
14(13)	0.024562798	1.296167037
15(13,14)	0.042003767	0.994684219
16(13,14,15)	0	0.561426177
16(13,14)	0	0.204779138
14	0	0.20119985
15(13)	0.000050357	0.161246521
15(14)	0.000293883	0.160471118
16(14,15)	0	0.147188036
15(13,14,16)	0	0.137449714
16(13,15)	0	0.116631427
13(14,15,16)	0	0.091541986
14(13,15,16)	0	0.069171241
16(14)	0	0.022426924
16(13)	0	0.011093428
14(13,15)	0	0.006165283
15	0	0.005325637
17(13,14)	0	-0.0199
17(14,16)	0	-0.0199
17(13,15,16)	0	-0.01

Figure 4.3.1 is another test input program which is simplified and applied to a hostel problem, and Table 4.3.1 is a part of the Q-table result without setting the goal state. Throughout the table, we can discover that adding register modifier before int variables, including variable statements in line indexes 13, 14, 15 and 16 will bring a more positive result. However, the attempt of adding a register modifier before line index 17 will bring a negative outcome, for the reason that register variables are stored in the register with no address and adding a register modifier before an array will make the elements in the array inaccessible.

4.4 Q-learning versus SARSA

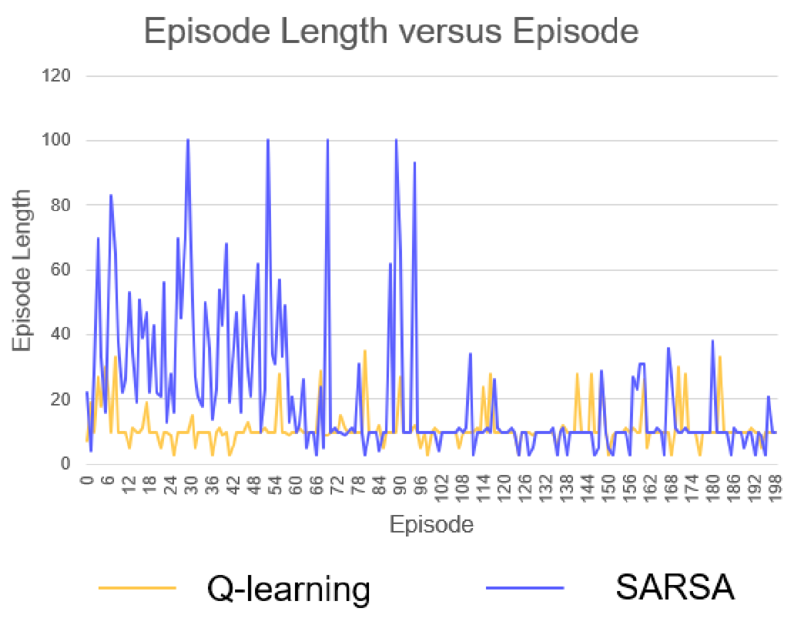


Figure 4.4.1 Episode Length versus Episode

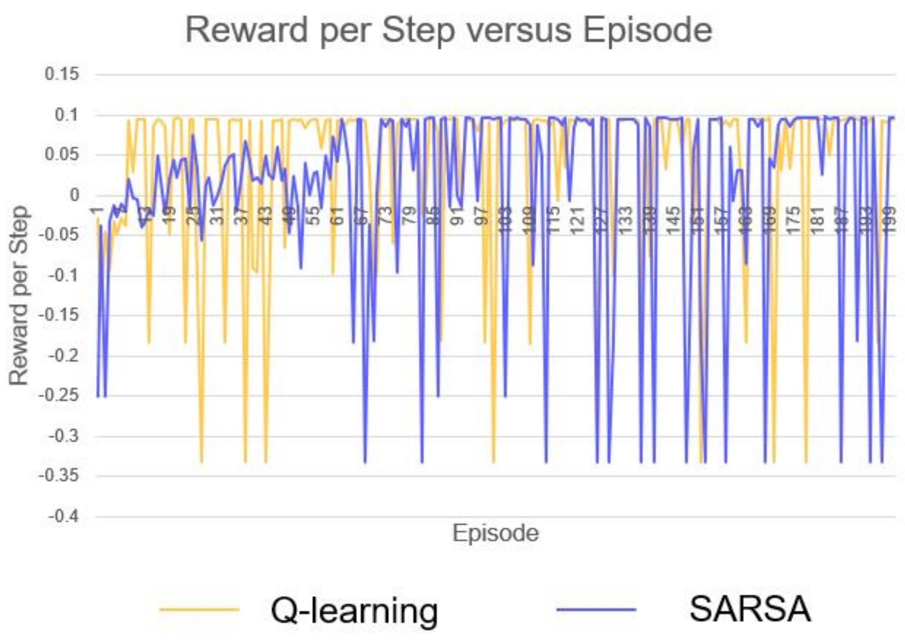


Figure 4.4.2 Reward per Step versus Episode

Figure 4.4.1 and Figure 4.4.2 show the difference between Q-learning and SARSA with 200 training episodes of Register_copy.c. Figure 4.4.1 is about the episode length of each episode, and a shorter episode length means that the agent reached the terminal faster. It should be noticed that because we have set the goal state in this experiment and the goal state is also a terminal, plus that Q-learning and SARSA are also convergent algorithms (Watkins 1989; Singh et al. 2000), the ideal state is 10. Figure 4.4.2 concerns the reward per step of each episode, and a larger value indicates that the agent is able to opt ideal for actions. Again, due to the convergence property of both algorithms (Watkins 1989; Singh et al. 2000), the ideal state is set to values above 0.09.

As mentioned in Section 2.4, Q-learning is a greedy algorithm while SARSA is more conservative, and the two diagrams confirm the statement. In both of the two diagrams, Q-learning performs a faster convergence speed to the ideal state than SARSA. If we compare the first time of two consecutive ideal states of the two algorithms, Q-learning (10) is 63 episodes, 86.3% ahead of SARSA (73) in episode length versus episode, and Q-learning (10) also leads SARSA (65) by 55 episodes, 84.6% in reward per step versus episode. Hence, Q-learning is more suitable for this experiment.

Chapter 5: Conclusion and Further Work

5.1 Conclusion

In this report, I searched for the combination of reinforcement learning and program optimisation and established a reinforcement learning system which successfully find optimisation suggestions on two programs based on the action of adding register modifier before variable statements. In addition, by comparing two value-based reinforcement learning algorithms, Q-learning and SARSA, Q-learning showed a faster convergence rate and more suitable than SARSA in the experimental environment. Also, from the pre-experiments, adding register modifier before variables and using different types of variables can accelerate the execution speed to some degree, while using more additions instead of multiplications, avoiding too many dependencies and jumps inside the loop and using different For loop structures will bring slight improvement.

5.2 Further Work

There are much more works that can be done in the future. For some problems, I cannot provide a suitable solution, and for other problems, I do not have enough ability to realize the solution.

Firstly, the action in the report is only for one line in the program, so the modification is easy to implement by replacing strings. But this method is not practical when we wish to make structural modifications like changing the loop structure. In which case, we must break the program into a tree, analysing the function and relationship among all the words in the program, like what Marc Brockschmidt and Alex Gaunt (2019) has tried in their research blog.

Secondly, I used the line index and the modification history to represent the state of the program, and there should be some more appropriate ways to finish the task.

Thirdly, I chose runtime as the only evaluation criterion for examining the effectiveness of modification, but using memory space may be more suitable if we could find a way to measure easily.

Fourthly, at present, the transition amid states is linear. When the agent reaches the line index 3, for example, it can only go to the line index 4 if it does not want to make a change. There is no way for the agent to jump to another line, and if it misses the opportunity of changing in one line, it must go through all the lines in the program and returns to the first line, then waiting for the next time it reaches that line.

Lastly, the two test input programs are still simple in this project, and I limited the largest search range of the program in 100 lines. Hence, the effect of the machine in enterprise application is still uncertain.

But all in all, this is a tiny study finished in several months, and I would expect its further development in the future.

References

Agarwal, A., Bird, S., Cozowicz, M., Hoang, L., Langford, J., Lee, S., Li, J., Melamed, D., Oshri, G., Ribas, O., Sen, S., & Slivkins, A. (2016). Making contextual decisions with low technical debt. *ArXiv*.

CAI Zepeng. (2018). The exploration and analysis on the optimization and application of embedded C program design. *Morden Information Technology*. Vol.2. no.1, 35-36.

CJCH Watkins. (1989). *Learning from delayed rewards [PhD dissertation]*. University of Cambridge. accessed 25 April 2022.

CJCH Watkins & P Dayan. (1992). Q-learning. *Machine Learning*, 8(3), 279-292

DU Hu-kang & ZHAO Ying-kai. (2010). Survey on intelligent optimization algorithms for solving integer programming problems. *Application Research of Computers.* Vol.27. no.2, 408-412.

Gauci, J., Conti, E., Liang, Y., Virochsiri, K., He, Y., Kaden, Z., Narayanan, V., Ye, X., & Chen, Z. (2019). Horizon: Facebook's open-source applied reinforcement learning platform. In *ICML 2019 RL4RealLife*.

GUO Mao-Zu, Wang Ya-Dong, SUN Hua-Mei & LIU Yang, (2002). Research on q-learning algorithm based on metropolis criterion. *Journal of Computer Research and Development, 39,* no.6, 684-688.

Jang, B., Kim, M., Harerimana, G. & Kim, J. W. (2019). Q-Learning algorithms: a comprehensive classification and applications. *IEEE Access, 7,* no.2019, 133653-133667.

Kyungeun Cho, Yunsick Sung & Kyhyun Kim. (2007). A production technique for a q-table with an influence map for speeding up Q-learning. In *2007 International Conference on Intelligent Pervasive Computing. Proceedings. 2007*. (pp. 72-+).

LIU Lixia & QIU Xiaohua. (2014). A database optimization method based on program business logic adjustment. In *2nd IEEE International Conference on Progress in Informatics and Computing (PIC). Proceedings. 2014* (pp. 424-427).

Macor Wiering & Jurgen Schmidhuber. (1998). Fast online Q(λ). *Machine Learning, 33*, 105-115.

Mahyar A. Amouzegar. (1999). A global optimization method for nonlinear bilevel programming problems. *IEEE Transactions on Systems Man and Cybernetics Part B-*

cybernetics, 29(6), 771-777.

Marc Brockschmidt & Alex Gaunt. (2019). *Beyond spell checkers: enhancing the edit process with deep learning*. Retrieved April 26, 2022, from Microsoft website: https://www.microsoft.com/en-us/research/blog/beyond-spell-checkers-enhancing-the-editing-process-with-deep-learning/

Morvan Zhou. (2020, November). *Reinforcement-learning-with-tenserflow*. Retrieved April 19, 2022, from GitHub Website: https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow/

NI Rui-xiao. (2009). Analysis and research on the technology of C programming. *Computer Technology and Development*. Vol.19. no.12, 251-254.

Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt & Alexander L. Gaunt. (2019). Learning to represent edits. In *ICLR 2019*.

Satinder Singh, Tommi Jaakkola, Michael L. Littman & Csaba Szepesvari. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*. Vol. 38. Issue 3, 287-308.

SI Yanna, PU Jiexin & SUN Lifan. (2022). Review of the research on approximate reinforcement learning algorithms. *Computer Engineering and Applications*. Retrieved April 12, 2022, from website: https://kns.cnki.net/kcms/detail/11.2127.TP.20220112.1105.002.html.

YANG Guangyu, GAO Xiaorong, WANG Li & WANG Zeyong. (2009). Technology of C/C^{++} program optimization based on TI C^{6000} DSP. *Modern Electronic Technology*. Vol.32. no.8, 56-59.

YANG Mao & ZHANG Hai-jun. (2005). The methods of optimizing procedure code of Visual Basic 6.0. Journal of Lanzhou Polytechnic College. Vol.12. no.1, 13-16.

zhengx-2000. (2022, April). *Aptorl*. Retrieved April 19, 2022, from GitHub Website: https://github.com/zhengx-2000/aptorl/

Acknowledgement

Special thanks to my supervisor, who is always willing to answer my confusion about the project and provide practical suggestions.

Special thanks to my parents who take care of my living so that I can deal with the project wholeheartedly.

Special thanks to my teammates in the research group who put forward some pieces of advice in the initial stage.

It is a tough journey for me. At first, I joined in the winter camp for the interest of artificial intelligence and the game, then I surprisingly won the first prize and knew my supervisor for the first time. Several months later, I participated in the research group of my supervisor and finally transferred the research outcome to my graduate project! However, I did not know how to deal with the project at that time and I thought that this project was impossible to establish! I have to say it is my supervisor who told me how to simplify the project, and he also encouraged me by saying that he would accept even if I finally drew the conclusion of failure, as long as I post all the research process in the project. Luckily, I finished the project step by step and generated a report that is not so perfect.

COVID-19 has spread all over the world for about three and a half years, which has also brought lots of troubles to me. I was separated from my dear classmates and teachers for nearly two years in total, and I will probably be unable to attend the graduate ceremony. No one could forecast the severity of the disaster in the past, and it seems that we are still unable to find the solution now. What can I do? I do not know exactly, and I just want to put some words in my graduate report to record the present, on April 26, 2022.

Appendix

北京邮电大学本科毕业设计（论文）任务书

Project Specification Form

Part 1 – Supervisor

论文题目 **Project Title**	Program Optimisation using Reinforcement Learning		
题目分类 **Scope**	Data Science and Artificial Intelligence	Research	Software
主要内容 **Project description**	Many of our programs are not well optimised. While compilers are now designed to optimise programs by techniques like multi-passes, it is possible to further improve programs on an algorithm level or a source code level, such that the execution time or the resources required to run the program can be reduced. In this project, the student will try to explore the application of the Reinforcement Learning (RL) for program optimisation. The main goal is to identify an appropriate programming language and a set of permissible changes that can be made to the source codes such that the RL framework can optimise the input programs accordingly. It is also crucial to propose and define the reward function to drive RL in the right direction.		
关键词 **Keywords**	reinforcement learning, programming		
主要任务 **Main tasks**	**1** To learn about RL framework and choose the programming language for the project		
	2 To implement the reward function and the changes into a RL framework to identify and propose a reward function and a set of permissible changes/moves that can be used by RL to optimise programs		
	3 To implement the reward function and the changes into a RL framework		
	4 To evaluate the results of the optimisation on a number of example programs and compare with 1-2 other similar approaches.		
主要成果 **Measurable outcomes**	**1** Implementation of a suitable reward function in the chosen RL framework		
	2 Implementation of the set of permissible changes in the chosen RL framework		
	3 A set of example programs which are optimised by the algorithm		

北京邮电大学 本科毕业设计（论文）任务书

Project Specification Form

Part 2 - Student

学院 **School**	International School	**专业** **Programme**	**Telecommunications Engineering with Management**		
姓 **Family name**	Zheng	**名** **First Name**	Xiao		
BUPT 学号 **BUPT number**	2018212779	**QM 学号** **QM number**	190014607	**班级** **Class**	2018215107
论文题目 **Project Title**	Program Optimisation Using Reinforcement Learning				
论文概述 **Project outline** **Write about 500-800 words** **Please refer to Project Student Handbook section 3.2**	Having written so many exercise codes in the programming-related courses including Programming Fundamentals, Introductory Java Programming, Interactive Media Design and Production, etc., while reviewing them, I could always find out somewhere that can be optimised, no matter by replacing “x = x + 1” with “x++” or changing the data structure, like using linked lists instead of structural array, or anything else. Besides, the experience of studying Artificial Intelligence Principles was so tough. In the class, I was required to design a Python project to train an artificial intelligence to identify the category of a piece of news, such as technological, agricultural or entertaining news. However, the execution and training time for the model is so long that I must wait several hours or even half of a day to finish the run and get the result, with the possibility of unexpected outcome or execution error, forcing me to stay in front of the laptop and check regularly about the execution status. Therefore, a question that comes to me is, whether there is a way which can accelerate the execution speed of or make other improvements to my programme automatically without altering the result? Admittedly, several practical solutions or products have been generated to deal with the problem. For instance, the Release version of GCC can automatically analyse and modify the code on the compiler level. Also, IDEs (Integrated Development Environment) such as Visual Studio Code, IntelliJ IDEA and PyCharm have integrated code checkers to correct grammar mistakes and provide advice. Besides, other software companies have also published some AI-based code assistant services. For example, DeepCode (now called Snyk Code) is a powerful code analyser, which is powered by machine learning based on an AI engine, to enable developers to build software securely during development, not trying to find and fix problems after the code is compiled. This engine learns from millions of open-source commits and is paired with Snyk’s Security Intelligence database, creating a continually growing code security knowledge base. [1] In addition, as for earlier attempts, in 2019, a research team from Microsoft published an article which shows their struggles in enhancing the editing process with deep learning. [2] In the article, the team shows us how they use two neural networks, an edit encoder network and an editor, to finish the process of recognizing the patterns in the code and applying learned edits. As for the project, I will look on the solution on an algorithm or a source code level and try to use the power of reinforcement learning to run the identification and optimisation process automatically. Firstly, I will start from the very beginning, from studying Python and basic reinforcement learning theory, and				

	then determine the most crucial part of the reinforcement learning, the reward function, to test the validation of several optimisation methods I collected from the Internet or other sources. After that, if the methods are all permissible and effective, I will try to build the Q-table and produce a programme that make the optimisation process automatically. This programme is specially designed for specific programs which have some explicit spaces for improvement, like long loop or large amount of variables statement. Finally, I will also look for other similar optimisation approaches, and compare mine with them to observe the effect. The train set and test set of data will all collected from online open-source channels like GitHub. The whole project will be written in Python on PyCharm platform which are both official and the most recent version, but the programs to be optimised will be C or Java. Reinforce learning would be involved in the project, and I would design a reward function to train the artificial intelligent and test the effect. To check the result, a user can input his or her program to the software, and an optimised program would be generated, with some improvements on the execution speed or other facets and no changes to the output when compared with the original program. [1] Snyk Code - Snyk User Docs, https://docs.snyk.io/products/snyk-code, cited on 16/11/2021. [2] Beyond spell checkers: Enhancing the editing process with deep learning, https://www.microsoft.com/en-us/research/blog/beyond-spell-checkers-enhancing-the-editing-process-with-deep-learning/, cited on 16/11/2021
道德规范 **Ethics**	Please confirm that you have discussed ethical issues with your Supervisor using the ethics checklist (Project Handbook Appendix 1). [YES/NO] YES

	Summary of ethical issues: (put N/A if not applicable) N/A
中期目标 **Mid-term target.** **It must be tangible outcomes,** **E.g. software, hardware or simulation.** **It will be assessed at the mid-term oral.**	A runnable program that uses the evaluation function to show the extent of improvement between original and optimised program.

Work Plan (Gantt Chart)

Fill in the sub-tasks and insert a letter X in the cells to show the extent of each task

	Nov 1-15	**Nov 16-30**	**Dec 1-15**	**Dec 16-31**	**Jan 1-15**	**Jan 16-31**	**Feb 1-15**	**Feb 16-28**	**Mar 1-15**	**Mar 16-31**	**Apr 1-15**	**Apr 16-30**
Task 1 [To learn about RL framework and choose the programming language for the project]												
Learn Python basis	X											
Learn machine learning basis	X	X										
Learn about reinforcement learning framework		X	X									
Choose the programming language		X	X									
Task 2 [To identify and propose a reward function and a set of permissible changes/moves that can be used by RL to optimise programs]												
Find useful changes/moves to the program				X	X							
Check the usability and permissibility of the changes/moves					X	X						
Find possible reward functions					X	X						
Compare and determine the reward function						X	X					
Task 3 [To implement the reward function and the changes into a RL framework]												
Implement the reward function						X	X					
Build the RL framework							X	X				
Test and optimise the framework							X	X	X			
Build the Q-table								X	X			
Task 4 [To evaluate the results of the optimisation on a number of example programs and compare with 1-2 other similar approaches]												
Output and evaluate the result										X	X	X
Find other approaches, input the example programs, and generate results											X	X
Make comparation among the program in the project and other approaches											X	X

北京邮电大学 本科毕业设计（论文）初期进度报告

Project Early-term Progress Report

学院 School	International School	**专业 Programme**	**Telecommunications Engineering with Management**		
姓 Family name	Zheng	**名 First Name**	Xiao		
BUPT 学号 BUPT number	2018212779	**QM 学号 QM number**	190014607	**班级 Class**	2018215107
论文题目 Project Title	Program Optimisation Using Reinforcement Learning				

已完成工作 Finished work:
The main task of the early term is learning Python, Machine Learning and Reinforcement Learning.

Part 1: Python Basis
Link: https://mofanpy.com/tutorials/python-basic/interactive-python/

Python is a powerful object-oriented programming language. With the approaching of artificial intelligence and big data era, Python has become the most popular programming language in the world. (Based on the PYPL-index, an analysis of Google search trends for programming language tutorials) Although Python has some disadvantages like no clear definition of data type and low execution speed, it still has plenty of advantages, and the most outstanding among which is that Python is inclusive. Thanks to numerous powerful packages to import and utilize, Python is compatible with lots of application scene, including crawler, AI, data, analysis, visualization, web, service, etc.

We often consider Python as a "programmer-friendly" language, for its convenience in coding. As for the same function, we would use about thirty minutes to programme in Java or Go, but in Python, we would use approximately ten minutes to do so. Besides, we adopt indentation to substitute columns, and we could integrate for and if statements into one single line to cut off redundant lines, and furthermore, lambda also enables us to declare simple functions in just one line, which reduces coding time further.

To some extent, the great success of Python somehow counts on its three mainstream third-party libraries: NumPy, Pandas and Matplotlib. Compared to list included in original Python, NumPy Array is fast in execution speed, since that NumPy prefers to store data in a piece of successive physical address. Also, data stored in NumPy Array are likely in one data type, which is also conducive to batch data calculation. Pandas is a package on NumPy, but Pandas can store more information than NumPy. At present, we often consider NumPy as List in Python, and Pandas, similarly, is like Dictionary in Python. In addition, Pandas has more functions for processing data and richer types of information. In particular, you have some tables containing characters, and Pandas can help you process and analyse these character data tables. Of course, there are many other functions, such as processing lost information, multiple ways to merge data, reading and saving in a more readable form, and so on. However, Pandas also has shortcomings: its operation speed is slightly slower than NumPy. Matplotlib comes from MATLAB, and it inherits powerful and useful plotting functions from MATLAB to draw charts and present your data. I have applied Matplotlib in my Computer Vision course, and it helps me to print results after image processing.

Part 2: Machine Learning Basis
Link: https://mofanpy.com/tutorials/machine-learning/ML-intro/machine-learning/

Generally, machine learning can be divided into three categories: supervised learning, unsupervised learning and semi-supervised learning. Just like human being, in supervised learning, there are always a teacher, or we could say a supervisor, to evaluate every action and give feedback, in most cases, a yes or no statement. So, after training, the artificial intelligence would probably choose actions in a certain circumstance that leads to a positive result. Unsupervised learning, on the contrary, has no supervisor. In unsupervised learning, it is often the case that the artificial intelligence utilized a formula or an evaluation system to evaluate its current state and generate a score by itself. In which case, no good or bad evaluation is provided, and the artificial intelligence must learn how to differentiate different outcomes. For instance, let say we want to train an artificial intelligence that can judge whether a picture is a cat or a dog. In supervised learning, we would provide the artificial intelligence with thousands of pictures labelled cat or dog. But in unsupervised learning, no label is provided, and the artificial intelligence must extract features in all images and divide them into two different types. Typically, training using supervised learning is much faster than unsupervised learning, but there are still some cases that we could not give explicit definitions of a state, and unsupervised learning should be applied in the circumstance. At last, semi-supervised learning is the combination of both supervised learning and unsupervised learning.

Neural Network (NN) is a common application of supervised learning, which takes advantage of biological neural network, using forward and back propagation to update neurons, so as to generate a good nervous system. In addition, Convolutional Neural Network (CNN) no longer processes the input information of each pixel, but processes each small pixel area on the picture, which strengthens the continuity of picture information. So that the neural network can see a graph instead of a point and deepen the neural network's understanding of the picture.

Part 3: Reinforcement Learning Basis
Link: https://mofanpy.com/tutorials/machine-learning/reinforcement-learning/intro-RL/

Reinforcement Learning (RL) is a type of machine learning algorithm which belongs to unsupervised learning. In reinforcement learning, the artificial intelligence will start from nothing, through constant attempts, learn from mistakes, finally find the law and learn the method to achieve the goal. Reinforcement learning is a large family, which includes many algorithms, such as Q Learning and Sarsa which utilize table learning to select specific behaviour through the value of behaviour, or Deep Q Network using neural network to learn, or Policy Gradients which directly output behaviour, or some other algorithms like Monte Karlo Tree Search which observe their surroundings, imagine a virtual environment and learn from it.

Part 4: Q Learning and Sarsa Basis
Link: https://mofanpy.com/tutorials/machine-learning/reinforcement-learning/intro-q-learning/
https://mofanpy.com/tutorials/machine-learning/reinforcement-learning/intro-sarsa/

Q learning is an off-policy algorithm, and its pseudo code is as follows:

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
      Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$ ;
  Until $s$ is terminal
```

In this pseudo code, $Q(s,a)$ refers to the score in state s with action a, r refers to the reward after taking such action. $\varepsilon,\gamma,\alpha$ are three parameters, where ε controls the randomness of action selection, γ controls the weight of the best choice, and α states the learning efficiency.

On the other hand, Sarsa is an on-policy algorithm, and its pseudo code is as follows:

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$ ;
    $a \leftarrow a'$ ;
  Until $s$ is terminal
```

When comparing Q Learning and Sarsa, we could find differences between off-policy and on policy. In Q Learning, when arriving at a state, we would check which action will bring the maximum reward and update the Q table, but we might not take that most favorable action (ε-greedy policy determines the action). In which case, we could say that off-policy is greedy, which always seek for the largest reward, ignoring the dangers. In contrary, Sarsa is more practical. In Sarsa, each action the artificial intelligence observes will definitely become the action that the artificial intelligence takes in the next step. Consequently, when it comes to on-policy, the artificial intelligence will carefully examine each possible actions and thus avoid those dangerous states, which is useful in application scenes when making mistakes costs a lot.

As for this project, I would probably seek for solutions using Q Learning. For example, let’s say if I have found some useful moves in the next research phase, then by applying Q Learning, the artificial intelligence would check the results after applying each move, and then find out the best move and update the Q table. After training with sufficient samples, a comprehensive Q table would be formed, and it could be used by the artificial intelligence to find out best optimisation moves for each test programme by analysing and pairing it to existing states.

Part 5: Choosing the Programming Language

After comparing some programming languages, I have decided to use Python as programming language, for its uniqueness in machine learning area. Also, as for the programming language been optimised, I would choose C (C++), because C is a process oriented and abstract general programming language, which is easier to optimise compilation and execution process.

是否符合进度? On schedule as per GANTT chart?
[YES/NO]
YES

下一步 Next steps:
Identify and propose a reward function and a set of permissible changes or moves that can be used by Reinforcement Learning to optimise programs.

北京邮电大学 本科毕业设计（论文）中期进度报告

Project Mid-term Progress Report

学院 School	International School	**专业 Programme**	**Telecommunications Engineering with Management**		
姓 Family name	Zheng	**名 First Name**	Xiao		
BUPT 学号 BUPT number	2018212779	**QM 学号 QM number**	190014607	**班级 Class**	2018215107
论文题目 Project Title	Program Optimisation Using Reinforcement Learning				

是否完成任务书中所定的中期目标？Targets met (as set in the Specification)?
[YES/NO] YES

已完成工作 Finished work:

Task 2.1: Find useful changes or moves to the program
Five pairs of optimisation changes have been considered in this part, and only three of them are listed here.

Change 1: Avoid multiplication and division inside loops and redundant calculation, since multiplication and division relatively require more time than addition and subtraction to calculate.
Found from website: https://www.cnblogs.com/jcchen1987/p/4362879.html

Change 2: Avoid too many dependencies and jumps inside the loop, since CPU can automatically apply a pipeline structure to deal with loops parallelly to some extents. However, if the calculation or logistics inside the loop is too complicated, then the CPU is unable to optimise the execution process.
Found from website: https://www.cnblogs.com/jcchen1987/p/4362879.html

Change 3: Using different structures of 'for' loops, since different expressions inside the 'for' statements like 'i++' and '++i' might make a difference.
Found from discussion with project members in Reinforcement Learning for Automatic Software Tuning.

Task 2.2: Check the usability and permissibility of the changes or moves
The test environment is as follows:
System: GNU/Linux 5.11.0-1.25-aws x86_64 from Amazon EC2 Ubuntu 20.04.3 LTS
Test codes are all available on https://github.com/zhengx-2000/aptorl/speedtest

Test codes for change 1: add_rplc_multi1.c, add_rplc_multi2.c
In add_rplc_multi1.c, we code the loop in the simplest and most familiar way.
In add_rplc_multi2.c, we try to use a temporary pointer to cut down multiply calculations in the loop.

Test codes for change 2: break_if1.c, break_if2.c
In break_if1.c, we place a complicated if-else statement in the loop.
In break_if2.c, we separate three if-else statements into three same loops and see whether there is difference.

Test codes for change 3: for_loop1.c, for_loop2.c, for_loop3.c, for_loop4.c, for_loop5.c
In for_loop1.c, we use i++ to increase count variable.
In for_loop2.c, we use i-- to circulate the loop.
In for_loop3.c, we use ++i to check the difference between pre-increasement and post-increasement.
In for_loop4.c, we use i+=1 to see if there is something different.

In for_loop5.c, we divided the loop into four separated loops to explore differences.

Test codes for change 4: Register_B.c, Register_O.c
In Register_B.c, we use int variables.
In Register_O.c, we use register int variables to see if there is improvement.

Test codes for change 5: Double.c, Float.c, Int.c, Long.c, Short.c, UnsignedInt.c, UnsignedLong.c, UnsignedShort.c
All these eight programmes complete the same task of calculating addition, subtraction, multiplication and division, but they use different variables as their title indicated during the calculation.

Test Programme: exectimer_auto.py

```
"""
A Programme to compile and run test C programmes automatically
several times, and finally display mean execution time for each one.
Modified from the previous version of Matthew Tang
Author: Xiao Zheng
"""
import time
import numpy as np
import subprocess

test_num = 1000
code = ['add_rplc_multi1', 'add_rplc_multi2']
# code = ['break_if1', 'break_if2']
# code = ['for_loop1', 'for_loop2', 'for_loop3', 'for_loop4', 'for_loop5']
# code = ['Register_B', 'Register_O']
# code = ['Double', 'Float', 'Int', 'Long', 'Short', 'UnsignedInt', 'UnsignedLong', 'UnsignedShort']
time_used = np.zeros([len(code), test_num])

for i in range(test_num):
    for j in range(len(code)):
        subprocess.call('gcc ' + code[j] + '.c -o ' + code[j], shell=True)    # To avoid results
      executed in
       # previous episodes influence next execution, the programmes have to be compiled each
       # time before execution

        # print(f'program = {code[j]}')
        t = time.perf_counter() # start time
        x = subprocess.call('./' + code[j])    # just run with all outputs to stdout
        t = time.perf_counter() - t # end time
        time_used[j, i] = t

        # print out messages
        # print(f'execution time = {t:02f} sec\nreturn value = {x}\n')
for i in range(len(code)):
    print('mean time for code ' + code[i] + '.c in ' + str(test_num) + ' cases: ' +
str(time_used[i].mean()))
```

Test results:
All programmes receive correct outputs, but their extents of optimisation are different.
Change1:

```
mean time for code add_rplc_multi1.c in 1000 cases: 0.0017305909567512572
mean time for code add_rplc_multi2.c in 1000 cases: 0.0017194716893136502
```

```
Change 2:
mean time for code break_if1.c in 1000 cases: 0.0015885910987854003
mean time for code break_if2.c in 1000 cases: 0.0015881837164051832

Change 3:
mean time for code for_loop1.c in 1000 cases: 0.0016406945595517754
mean time for code for_loop2.c in 1000 cases: 0.0016537874061614275
mean time for code for_loop3.c in 1000 cases: 0.0016471961648203432
mean time for code for_loop4.c in 1000 cases: 0.001640748315025121
mean time for code for_loop5.c in 1000 cases: 0.001640339597593993

Change 4:
mean time for code Register_B.c in 1000 cases: 0.138306049445644
mean time for code Register_O.c in 1000 cases: 0.02069015857158229

Change 5:
mean time for code Double.c in 1000 cases: 0.049295287599787116
mean time for code Float.c in 1000 cases: 0.04942889773705974
mean time for code Int.c in 1000 cases: 0.058878951204940674
mean time for code Long.c in 1000 cases: 0.14316591325867922
mean time for code Short.c in 1000 cases: 0.05904091823240742
mean time for code UnsignedInt.c in 1000 cases: 0.06380128407105803
mean time for code UnsignedLong.c in 1000 cases: 0.15279860339360313
mean time for code UnsignedShort.c in 1000 cases: 0.06367871726630256
```

The result shows that the optimization effect for change 1, 2 and 3 is slight, which is less than 2%, while the effect for change 4 and 5 is relatively notable. In change 4, the improvement has reached 85.1%, and in change 5, the largest optimization is about 69.7%, which occurs between Unsigned Long and Double.

Task 2.3: Find possible reward functions
There are three factors to help us judge whether the programme has been optimised. The first is runtime, since time is valuable, and we all want our programme to get correct output faster. The second factor is memory space been used, since it is favourable for us to save up more memory space which could be used for other tasks in the meantime. The third one is programme size, since a more concise programme completing the same task is welcomed by every programmer and easy to read.

Task 2.4: Compare and determine the reward function
In nearest approach, we only use runtime to determine the effect of optimisation, and it is the simplest and most effective way, so we determine that the reward is 1 if runtime has decreased, 0 for any other situation. As for memory space been used, it is hard to calculate accurately, and also, it may vary a lot each time we compile and execute the test programme, so this standard is discarded. When it comes to programme size, it is easy somehow but not so urgent. In real situation, we do not care too much about the programme size, some large files are necessary for some reasons.

尚需完成的任务 Work to do:
Build a Reinforcement Learning framework and apply it to build a usable Q-table for test programmes.
Adjust factors in Q-Learning to increase accuracy.

存在问题 Problems:
Two useful approaches for optimisation belong to same category (changing data types), and more other optimisation methods need to be found out and tested.

拟采取的办法 Solutions:

Increasing the execution base time for test codes to check out whether there would be more effective changes.

论文结构 Structure of the final report:

1. Specification
2. Abstract
3. Table of Contents
4. Introduction
5. Background

5.1 Reinforcement Learning
Showing the characteristics of Reinforcement Learning and its relationship with Machine Learning.
5.2 Q-learning
Showing the reason of choosing Reinforcement Learning and advantages of applying Q-learning in the project.

6. Design and Implementation

Showing Reinforcement Learning environment and procedure of developing the artificial intelligence.

7. Results and Discussion

7.1 Trained Q-table
Listing some successful (and maybe failed) Q-tables with its original test codes.
7.2 Test Results
Showing codes before and after Reinforcement Learning Optimisation to see the difference, and applying the generated Q-table to the same code several times and different codes to examine its stubbornness and uniqueness.
7.3 Discussion

8. Conclusion and Further Work
9. References (Bibliography)
10. Acknowledgements
11. Appendices
12. Risk and Environmental Impact Assessment

北京邮电大学 本科毕业设计（论文）教师指导记录表

Project Supervision Log

学院 School	International School	**专业 Programme**	Telecommunications Engineering with Management		
姓 Family name	Zheng	**名 First Name**	Xiao		
BUPT 学号 BUPT number	2018212779	**QM 学号 QM number**	190014607	**班级 Class**	2018215107
论文题目 Project Title	Program Optimisation Using Reinforcement Learning				

Please record supervision log using the format below:

Date: dd-mm-yyyy
Supervision type: face-to-face meeting/online meeting/email/other (please specify)
Summary:

Date: 02-11-2021
Supervision type: online meeting
Summary: discussed the project specification

Date: 17-11-2021
Supervision type: Microsoft Teams
Summary: received written feedback on the draft specification

Date: 18-11-2021
Supervision type: Microsoft Teams
Summary: discussed the draft specification

Date: 10-01-2022
Supervision type: Microsoft Teams
Summary: discussed and modified the draft early-term report

Date: 09-02-2022
Supervision type: Microsoft Teams
Summary: discussed about the definition of states in the project

Date: 14-02-2022
Supervision type: Microsoft Teams
Summary: discussed about Chinese title of the project

Date: 21-02-2022
Supervision type: Microsoft Teams
Summary: discussed the research orientation of the project

Date: 27-02-2022
Supervision type: Microsoft Teams
Summary: discussed and modified the draft mid-term report

Date: 01-03-2022
Supervision type: Microsoft Teams
Summary: discussed project supervision log

Date: 07-03-2022
Supervision type: Microsoft Teams
Summary: discussed and modified the mid-term viva ppt and video

Date: 16-03-2022
Supervision type: online meeting
Summary: discussed the current process and key problems

Date: 21-03-2022
Supervision type: Microsoft Teams
Summary: discussed about the elimination of state factors

Date: 24-03-2022
Supervision type: Microsoft Teams
Summary: discussed about expansion of the action space

Date: 28-03-2022
Supervision type: Microsoft Teams
Summary: discussed the complete version of reinforcement learning environment

Date: 11-04-2022
Supervision type: online meeting
Summary: held the mock viva and provided feedbacks on ppt slides

Date: 16-04-2022
Supervision type: Microsoft Teams
Summary: discussed the first version of the report

Date: 23-04-2022
Supervision type: Microsoft Teams
Summary: discussed the second version of the report

Date: 26-04-2022
Supervision type: Microsoft Teams
Summary: discussed the third version of the report and the risk assessment

Date: 27-04-2022
Supervision type: Microsoft Teams
Summary: checked the final version of the report

Risk and environmental impact assessment

The title of this project is Program Optimisation using Reinforcement Learning, which is a research project of software in data science and artificial intelligence. The main purpose of the project is to develop a system based on the Reinforcement Learning (RL) that applied in program optimisation.

Table A Risk Assessment Table

Description of Risk	Description of Impact	Likelihood Rating	Impact Rating	Preventative Actions
Data Explosion	Low Risk	1	1	Delete junk files manually
Losing File	Moderate Risk	2	2	Save the copy of the file in a safe place

Table A summarized risks in this project. The first risk is about data explosion, which may happen when the program is disrupted accidentally. Because a large amount of test files is generated while executing the program, and if they cannot be deleted at the end of the program, then they would remain in the directory. The second risk is about losing files. As stated before, junk files are eliminated at the end of the program, and the rule of removing them is based on the name of the file. So, if the original program or other file in the directory match the rule, then they will be removed directly when the program finishes, with no possibility to recover. Hence, I suggest saving copies of important files in a safe place before execution, or building a separate space for the program to run.