Universiteit Leiden
Opleiding Informatica

Identification of Scratch projects' Similarity

Using Clustering Algorithms

Name: Shangyi Tang

Date: 10/11/2020

1st supervisor: Efthimia Aivaloglou
2nd supervisor: Felienne Hermans

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The purpose of this thesis is to propose a method to cluster Scratch projects according to their similarity.

Scratch is a web-based visual programming environment, which supports young programmers in their first steps in coding. Due to the widespread use of Scratch among programming beginners, Scratch has recently received significant attention from the computing education research community. Datasets composed of projects scraped from the public Scratch projects repository have been used to research the learning trajectories and programming practices applied by young learners. However, the public Scratch projects repository contains not only self-made projects, but also projects created using guidance materials like tutorials, books or MOOCs. The inclusion of such projects in the research datasets can affect the obtained results, and this thesis intends to contribute in solving this issue.

The motivation is to pick out the self-made Scratch projects of a large dataset of public projects from the ones that were made using guidance materials. The method that we applied is composed of two parts. The first part is scraping JSON files of Scratch projects and parsing those JSON files to store information of Scratch projects into a MySQL dataset. The second part is clustering Scratch projects. This part is divided into two steps: similarity calculation on scripts and similarity calculation on projects. In both steps, the calculation use methods of Simhash algorithm and K-means++ algorithm. In the first step, the calculation uses block types as vectors. In the second step, the calculation uses clusters of scripts, which results from the first step, as vectors. As a result, Scratch projects are clustered to a certain extent, and there are still many optimization points that could be implemented to improve the accuracy of the work.

Keywords

Scratch, JSON, Scrape, Parse, Similarity Calculation, Simhash, K-means++, Cluster, Block, Script, Project.

Acknowledgement

Firstly, I would like to thank my supervisors, Efthimia Aivaloglou and Felienne Hermans. Thanks for their careful patient guidance on my master thesis and organizing PERL that helps me understand more about programming education and have more chance to communicate with other students.

Secondly, I want to thank my classmates and friends during my master's study, who help me meet with difficulties, especially Dapeng Wang. He gave me various help since I came to the Netherlands, a whole new environment for me.

Last but not least, I would also like to thank my parents, who support me both financially and mentally. Thank them for giving me a chance to experience a study in a foreign country.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Introduction

Scratch is a block-based programming language developed for teenage and younger programmers, one of the visual programming environments suitable for teaching programming concepts. It is a quick-start environment for programming beginners, which provides a convenient and intuitive interface for users to be initiated and learning programming. Also, it has a website to provide an online environment for users, and currently, there are over 45 million projects publicly available on the website. Due to its large volume, This projects dataset can be valuable in giving insights and analyzing the habits or trends of those users' programming practices.

This public dataset downloaded from the website is widely used in recent Scratch-related research. For example, Aivaloglou and Hermans[1] researched Scratch users programming habits using a dataset of over 250,000 projects downloaded from the Scratch website during March 2016; Boe et al.[3] used 432 Scratch programs to test the automated system called Hairball, which can point out potential errors and assist in inspecting the implementation of Scratch programs; Moreno and G. Robles[11] download 100 projects from the Scratch website and analyze them to test the hypothesis that irregular programming habits can generally be found in the Scratch community; Moreno-León and G. Robles[12] use a tool named Dr. Scratch to evaluate the Computational Thinking skills of Scratch projects. Through those works, valuable insights on the programming practices and computational thinking skills

applied by Scratch users have been gained.

Since Scratch is for programming beginners, there are many tutorials (Programming in Scratch[1], Scratch: Programming for Kids (8+)[2]), books (Coding Games in Scratch[3]) and MOOCs (CoderDojo[4]) for beginners' reference. On one hand, it helps young programmers to get started. On the other hand, it also means that programs created following those tutorials and examples are not self-created, and therefore they are not so valuable as original Scratch programs at the research level. Therefore, for researches and experiments on Scratch, removing those guided programs may lead to more accurate results. A possible way to increase the proportion of original self-created programs is to identify remove guided programs from the original dataset according to the similarities between programs. We expect that guided projects have similar code structures to the original tutorials they learned. Since Scratch projects consist of scripts, we decide to calculate similarities according to scripts' structures built with blocks. By analyzing distributions of blocks and scripts, we propose to cluster Scratch projects.

This thesis aims to propose a method to calculate similarities between Scratch programs, among to identify and remove duplicated programs from the dataset. For the specific implementation, I use the Simhash algorithm and K-means++ clustering algorithm as fundamental methods to build the program of similarity calculation on Scratch programs. Scratch program consists of scripts, which are units of code in the Scratch program, and each script consists of blocks, which are the smallest units containing simple functions in the Scratch. To calculate similarities of Scratch projects, we need to start the experiment from the smallest part. In summary, I implement both the Simhash algorithm and the K-means++ algorithm, respectively, on scripts and Scratch projects. Firstly, I split scripts in Scratch programs into blocks, dividing them into block types, and treating them as fingerprints of scripts according

[1]https://www.edx.org/course/programming-in-scratch

[2]https://www.edx.org/course/scratch-programming-for-kids-8

[3]https://books.google.com/books/about/Coding_Games_in_Scratch.htmlid=z1I9tAEACAAJ

[4]https://coderdojo.com/

to the Simhash algorithm's method. Then I implement the K-means++ algorithm to cluster scripts, using block fingerprints as vectors. As a result, I get a set of script groups. After clustering scripts, I use them as the vectors of Scratch projects according to the Simhash algorithm. Finally, by using these vectors, I cluster Scratch programs using the K-means++ algorithm.

We firstly applied this process to a dataset of 508,704 Scratch projects that we scraped from the Scratch repository. However, due to the large time cost on the calculation, we reduce the dataset's size to 10,000 projects. It resulted to a set of 534 clusters containing 5,444 projects in total, which means that 4,566 projects are unique. The largest cluster has 421 projects, and the second-largest cluster includes 257 projects. Scratch projects are clustered to a certain extent, while there are still some points to improve the result's accuracy.

The rest of this thesis is structured as follows: Background, Related Work, Method and Experiment, Results, Discussion, and Conclusion and Future Work.

Chapter Two

Background

This chapter describes platforms, technologies and concepts that form the background of the work in this paper. The basics of Scratch are outlined in section 2.1. Scratch projects are scraped in JSON format (section 2.2) using web scraping technology (section 2.3) through API function (section 2.4). JSON files are parsed, and parsed data are stored in MySQL database (section 2.5). During Similarity calculation process, the method of Simhash algorithm that belongs to Hash algorithm (section 2.6) and K-mean++ algorithm that belongs to clustering algorithm (section 2.7) are used.

2.1 Scratch

Scratch is a block-based visual programming language developed for teenage or younger programmers. It is developed by the Lifelong Kindergarten Group affiliated with MIT (Massachusetts Institute of Technology).

Scratch provides a block-like interface, using a Web page as a carrier, to help users program their works smoothly, and has been translated to over 70 different languages. Therefore, Scratch is easy to get started for the studying of beginners of programming worldwide. It is suitable for helping beginners understand the structure and basic operating principles of programs.

2.1.1 Constitution of Scratch Projects

Project

A project[1] is the unit of a complete Scratch program. The project consists of sprites[2], which can be seen as individuals in projects like characters, objects, and even the background. Sprites contain scripts, which are the core parts of the Scratch project. This structure helps programming beginners to sort out codes according to their functions.

Script

Scripts[3], which can be said as the central part of projects, are a combination of code blocks. They are essentially the code in Scratch programs. Their structures decide the functions of sprites. According to different collocations of blocks, scripts can realize various sprites' orders, thereby executing different actions on sprites.

Block

Blocks[4] can be seen as the skeleton of projects. A single block does nothing to the project, but they become effective in groups. Blocks have several types, and different types manage different simple functions. They can be divided according to shapes, such as hat blocks (start point of a script), stack blocks (main commands), cap blocks (endpoint of a script), etc. Alternatively, they can be divided according to functions like motion blocks (commands that move sprites), sound blocks (commands to play sounds), etc. Scratch also provides custom blocks for skilled users to create their own commands. According to the growth of users' programming level, the potential of blocks can also be developed.

[1] https://en.scratch-wiki.info/wiki/Project
[2] https://en.scratch-wiki.info/wiki/Sprite
[3] https://en.scratch-wiki.info/wiki/Scripts
[4] https://en.scratch-wiki.info/wiki/Blocks

2.2 JSON Format

JSON (JavaScript Object Notation) is a data-interchange format that was derived from JavaScript. JSON is a suitable format for the machine's generating and writing since it is a language-independent data format. Besides, it is easy for programmers to read and write. Consequently, JSON has become a widely used format for data exchange.

JSON mainly has two kinds of structures, one is a collection of name/value pairs, and another is an ordered list of values. These structures can be easily transferred to an array or dictionary in programming languages; thus, data inside could be easily mined. In this project, JSON files downloaded from the Scratch website are used for gathering the structure data of Scratch projects.

2.3 Web Scrapers

Web scraping, also called a web crawler, automatically gathers some specific data from websites. Its method is to access a specific website or several websites with similar domains and structures, then download online data according to some fixed settings automatically. This enables getting large datasets from the network, which can be used for analysis in various research fields.

2.4 API

API (Application Programming Interface) is a set of functions and procedures which allows access to an operating system, a database, or other services.

API allows users to access and get data they want without learning their specific source code or details of their working mechanism. This fact allows people to access and retrieve data from the extensive database securely and efficiently. In this project, the Scratch website's API is used in the web scrape of the JSON file.

2.5 MySQL

MySQL is a commonly used Relational Database Management System (RDBMS). MySQL can be accessed by and combined in many programming languages, thus can be easily combined into different kinds of programs. Besides, it can also be used independently. These characters improve the versatility of MySQL.

2.6 Hash Algorithms

A hash algorithm is a function that converts a data string into a numeric string output of fixed length. The output string is generally much smaller than the original data. Hash algorithms are designed to be collision-resistant, meaning that there is a very low probability that the same string would be created for different data[5]. It is a method to transfer data into signs, thereby improving the utilization of storage space, accelerating data query efficiency, and can also do digital signatures to ensure the security of data transmission. Therefore, the Hash algorithm is widely used in Internet applications.

2.6.1 The SimHash Algorithm

In computer science, SimHash is a technique for quickly estimating how similar two sets are. The algorithm is used by the Google Crawler to find near-duplicate pages[6]. Charikar[4] created it. Its general method is dimensional reduction. Specifically, this algorithm transfers entity individuals into vectors to simplify the relationship between them. Then the Hamming Distance between individuals is calculated to determine the similarity between them.

[5]http://www.digitizationguidelines.gov/term.php?term=hashalgorithm

[6]https://en.wikipedia.org/wiki/SimHash

Hamming Distance

The Hamming distance means the difference between two strings of equal length. It calculates XOR (exclusive or) operation between two binary strings, and the volume of 1 in the result would be the value of Hamming distance. It is named after the American mathematician Richard Hamming[7].

Process of Simhash Algorithm

The Simhash algorithm is divided into five steps: word segmentation, hashing, weighting, merging, and dimensional reduction. The specific process[8] is as follows:

1) Word Segmentation

   Perform word segmentation to a sentence to obtain useful feature vectors and then set several weights such as 1-5 for each vector.

2) Hashing

   Calculate the hash value of each vector through the hash function. The hash value is an n-bit signature composed of a binary number.

3) Weighting

   Based on the hash value, all feature vectors are weighted (W = Hash * weight), and when the current digit of the hash value is 1, the hash value and the weight are positively multiplied, and when it is 0, the hash value and the weight are negatively multiplied.

4) Merging

   Combine the weighted results of the vectors to one string.

5) Dimensional Reduction

[7]https://en.wikipedia.org/wiki/Hamming_distance

[8]https://blog.csdn.net/lengye7/article/details/79789206 (Chinese Website)

For the cumulative result of the n-bit signature, if it is greater than 0, set it to 1, otherwise set it to 0 to get the sentence's Simhash value. Finally, the similarity of different sentences can be judged based on the Hamming distance.

2.7 Clustering Algorithms

Clustering algorithm (Clustering analysis) is a statistical method of classification problem and a generally used algorithm in Data mining. It is categorized as one of the unsupervised learning methods in machine learning, which does not use the input data's tutor signal. After the clustering method is used, the learning result is shown as categorized groups. The clustering algorithm itself is not a specific formula but a general method to solve clustering tasks. According to different focus points and efficiency on a calculation, various algorithms can be used in clustering depending on specific issues.

2.7.1 K-means algorithm

K-means algorithm is an iterative clustering analysis algorithm. This algorithm's method is dividing individuals into k groups, and in every group, there is a center point, which is the average of all individuals in the corresponding group. The specific process is as follows:

1) Select k objects from the data as the initial cluster centers.

2) Calculate the distance of each cluster object to the cluster center and assign them to the cluster, which contains the closest center point.

3) Calculate the new center in each cluster.

4) Loop 2) and 3) until the maximum number of settled iterations is reached; Clusters become stable, Or other conditions require to stop the loop.

2.7.2 K-means++ algorithm

The k-means algorithm has two major theoretic shortcomings[9]:

· First, it has been shown that the worst-case running time of the algorithm is super-polynomial in the input size.

· Second, the approximation found can be arbitrarily wrong concerning the objective function compared to the optimal clustering.

The K-means++ algorithm is an optimized algorithm based on the K-means algorithm[10]. It solves the second theoretic defect by improving the initialization process. In the original K-means algorithm, initial centers are chosen randomly. Using this method, it may choose various centers close to each other, which could be in one cluster if one of them is not a center. In comparison, the K-means++ algorithm in the initialization strategy is choosing centers that are far away from each other. This change avoids centers to get together and makes the algorithm guaranteed to find a solution that has higher qualities in time spending and concrete results.

[9]https://en.wikipedia.org/wiki/K-means%2B%2B

[10]https://en.wikipedia.org/wiki/K-means%2B%2B

Chapter Three

Related Work

Work that is related to this thesis falls into three areas: Scratch, machine learning, and K-means based clustering, which are described in the following sections.

3.1 Related Work on Scratch

Scratch is a visual programming environment that helps programming beginners increase their programming skills and understand computer programs' structure. Research on Scratch mainly concentrates on the computational thinking skills practiced with Scratch and on Scratch users' habits and trends.

Maloney et al.[9] introduces the Scratch programming language itself and its environment. This paper describes that Scratch is a visual programming environment to support young programmers' self-directed learning through tinkering and collaboration with peers. The Scratch programming environment makes the script execution visible and feedbacks timely. Besides using abstract variables like most text-based programming languages, Scratch makes variables concrete to help users more comfortable understanding the gimmick of computer programs.

Meerbaum-Salant, Armoni, and Ben-Ari[10] research Scratch users' programming habits. Visual programming environments like Scratch are widely used for young people to get

started on computer science and programming. While beginning with Scratch, users may have two habits that are not conducive to the development of general computational thinking and programming skills on programming languages for developed programmers. One is that users may start building their programming methods from a single Scratch block instead of the whole concept of a program. Another one is that they may do not have an idea of the structure of a program. Currently, it is not clear if those habits would affect the users' future programming learning in the wrong way, or they can outgrow these bad habits at last. It is still a task waiting for researchers to explore.

Moreno and G. Robles[11] focus on verifying if bad habits on Scratch programming are commonly found in the Scratch community by downloading and analyzing 100 Scratch projects with two plugins that extended the hairball tool, an analyzer for Scratch programs. The experiment concentrates on two issues, object naming and code repetition, and after the analysis on 100 projects, 79% and 62% respectively have those two bad habits. For naming habits,they suggest that setting naming variables mandatory to avoid users using automatically given names for variables. For code repetition, they found that the habit may be improved as experience accumulate. Since Scratch is for young programming beginners, this practice may happen more frequently than by experienced programmers.

Aivaloglou and Hermans[1] did an exploratory study on the Scratch repository. This paper explores the characteristics of Scratch programs in three dimensions, using the dataset of 250,000 scraped projects from the Scratch website. The first dimension is to understand the structure of Scratch programs, including their size and complexity. The numbers of blocks in scripts and sprites are statistically analyzed to measure this three dimensions. The second dimension is to know the programming habits and characters of Scratch users. For this question, block categories are statistically analyzed in order to clear the structures of programs. The third dimension is to recognize code smells in scratch programs. Four types of code smell, duplicated code, dead code, large script, and large sprite focus on solving this question. Different processes are implemented for those analyses. As a result of three

dimensions: 1) although large size projects exist, most of them are small, 2) projects are mostly in simple structures, and 3) Scratch programs do suffer from code smells, including large scripts and unmatched broadcast signals.

Recently, Amanullah and Bell[2] research progression of Scratch users' skill by doing analysis on 34,832 users' created Scratch projects in one year. The measured standard of the quality in programming skill is based on elementary patterns used in projects. According to their analysis, most users do not demonstrate significant development in their programming skills. The key elements, which are measures of this research, keep in low usage in their projects. This may because the position of the Scratch is a visual environment that may not be suitable for users to understand the structure of code in depth. Therefore, Scratch users may not quickly grasp the method of some elementary patterns in general programming. To solve this problem, they proposed that educators may combine Scratch with other programming teaching materials to help programming beginners understand that programming is based on the range of element usage instead of the length of the code.

3.2 Similarity Calculation.

Hatzivassiloglou, Klavans, and Eskin[7] presents a new method to the traditional text similarity calculation by using a method of Linguistic Feature Combinations with machine learning. The machine learning algorithm used in this paper is RIPPER. RIPPER is a rule-based learner. Its general method is looping the process of growing (learning rules), Pruning (discard useless rules), and Optimizing (reduce error results). The method of evaluation model according to the relationship between words in one passage is about the rules using in detecting short passages' similarity in this paper. Those relationships are divided into three categories: 1) 'Order,' which means the relative order of two words or the position of two words in the passage, 2) 'Distance,' which can be understood as several other words between those two words, and 3) 'Primitive,' which means the relationship between two words in

their specific meanings, such as the noun 'bird' and the verb 'fly,' has a relationship in their meanings and is comfortable to appear simultaneously.

Sadowski and Levin[13] do an experiment on using the Simhash algorithm to do the similarity detection, while mostly hash algorithms are used to separate and obscure data. The authors have developed the Simhash algorithm as a hash algorithm to calculate similarity using a set of hash keys and auxiliary data to determine the similarity between individuals. As a result, they found an unbalanced weighting scheme where only a few tags (preselected target individuals) were used to compute a key worked best on a realistic file set, which means that simple individuals may be better for getting a good result.

Fu et al[6] propose a similarity search method for encrypted documents based on the Simhash. With the wide usage of cloud computing, online data tend to be protected in encrypted form instead of exposed in plain text to avoid privacy leaked. However, those protections seriously hinder traditional searching function based on keyword searching. Therefore, this paper does experiments on the Simhash similarity searching scheme. For the solution, each document is transformed into a fingerprint with the Simhash, and the hamming distance is used as the similarity score between documents. Besides, the trie-based index is adopted to accelerate searching speed by solving the top-k problem (find top k most qualified elements from a dataset).

3.3 K-means Clustering.

Clustering analysis is a fundamental task in unsupervised machine learning, and the K-means algorithm is a generally used method in clustering experiments. Although the method of K-means was first published over 60 years ago[8], it is still a commonly used clustering method. On the one hand, it illustrates the versatility of this method. While on the other hand, keeping using k-means during this long period also describes the difficulty of designing a general-purpose clustering algorithm.

Wagstaff et al.[15] propose an idea to add some rules to constrain the k-means algorithm during clustering computing. According to some extra information about the problem domain, the authors set some 'must link' relationships and 'must not link' relationships between individuals and clusters. As they implement this method on lane finding in GPS data and some other experiments, the result shows that with Constrains combined with the actual situation, the k-means algorithm performs much better than the original one. Besides, since the assignment of instances to clusters can be order-sensitive, some individuals could not find a cluster for them; proper backtracking and optimization may need the latter works.

Erman, Arlitt, and Mahanti[5] use three clustering algorithms to implement network traffic classification. Those clustering algorithms are K-means, DBSCAN, and AutoClass. K-means and DBSCAN are two unsupervised clustering algorithms that are not used for network traffic classification previously, and AutoClass is the generally used algorithm for it. According to evaluations for three algorithms, all of them have their advantages. The AutoClass algorithm produces the best in accuracy, and K-means comes to the next. Although the DBSCAN algorithm has the lowest accuracy, it produces better clusters according to their experiments. The K-means algorithm has the right balance between accuracy and time cost since it and DBSCAN calculate much faster than AutoClass. Therefore, the K-means algorithm may be the most suitable algorithm for this problem.

Sculley[14] presents two modifications to the popular k-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. One is approaching mini-batch optimization to the original K-means algorithm, thereby replacing online stochastic gradient descent to shorten calculation time significantly. Another one is making cluster centers sparse to improve accuracy with the lower time cost.

Chapter Four

Method and Experiment

This section describes the method and experiment process of the project. The main process of this project is divided into two parts:

1) Creating a dataset of Scratch projects by scraping and parsing JSON files from the Scratch Website.

2) Using a clustering algorithm to calculate the similarity of Scratch projects.

In the first part, we use the web scraping method to download source files (JSON files) of Scratch projects from the Scratch website. Then I use Python to parse Scratch projects' structures from JSON files and store them into a MySQL database built with MySQL Workbench.

In the second part, I use the Simhash algorithm and K-means ++ clustering algorithm to calculate Scratch projects' similarity by clustering them. The overview of this process is as follows:

1) Loading structure data from MySQL database generated in the first part and generating the list of scripts by analyzing those project data.

2) Using blocks in scripts as vectors and implementing the Simhash algorithm and K-means++ algorithm to cluster scripts.

3) According to the result in the previous step, using scripts as a vector of projects and implementing the Simhash algorithm and K-means++ algorithm to cluster projects.

4.1 Creating a Dataset of Scratch Projects

This approach is the beginning step of this project. In this part, two main processes are implemented: scraping JSON files and parsing JSON files.

The purpose of this part is to get a dataset of the most recent Scratch projects, and the goal of the dataset's volume is to contain over 500,000 projects. In the Scratch website, the source files of projects are stored as JSON files. Thus details of projects can be found by downloading and reading those files. Those files can be downloaded through the HTTP request of the Scratch website. Afterward, data in JSON files can be transferred to dictionary format, and necessary parameters in it can be picked out. Finally, those parameters would be stored in a MySQL database, and this database would be the dataset of Scratch project structures. Therefore, the experiment of this project is divided into two parts: scraping and parsing. The scraping part concentrates on the download of a large number of JSON files from the Scratch website. In contrast, the parsing part concentrates on the analysis of those downloaded JSON files.

The Scratch website uses the JSON format to transmit data to the user's browser, which can be found as source files through the web console in browsers (the browser used in this project is Firefox). Most details needed for building the dataset are therefore stored in those JSON files. JSON files on the Scratch website can be accessed through the API address provided from the official Scratch website, which can also be found through the web console. The goal of the dataset's volume is 500,000 projects. Thus this is the minimum number of JSON files that would be downloaded in this project.

4.1.1 Scraping Source files (JSON files) from the Scratch Website

Process of Scraping JSON files

The scraping of JSON files can be divided into two steps: getting the list of the identifiers (ids) of recent projects and downloading JSON files of specific projects.

Since there are millions of projects stored on the Scratch website, a way to get their API address automatically is needed to scratch JSON files of projects. Fortunately, the Scratch website has a page[1] to show the list of the most recent or trend projects. From the JSON files of this page, we can get the ids of projects listed on this page. By collecting those ids, the list of projects we need to scrape can be captured. After gathering the ids, we can use them to generate the API address of specific JSON files. Afterward, the same way of scraping can be used in those addresses to download JSON files, which include details of Scratch projects.

Since those JSON files store the complete information of the corresponding Scratch projects, we still need a process to pick up that information we need for building a dataset of recent Scratch projects in a format that is usable for later analysis.

Experiment of Scraping JSON files

For the scraping program, I first refer to the structure of Kragle[2], which is programmed by C# and is the prototype of the program in this project, and then use the package 'urllib' in python to build the scraping program for JSON files stored in Scratch website according to Kragle. Algorithm 1 shows the primary process of the Scrape part.

[1]https://scratch.mit.edu/explore/projects/all/recent

[2]https://github.com/Felienne/Kragle

Algorithm 1 Scraping Scratch projects.

$id\_list = []$
$L\_id =$ length of id_list
while $L\_id < 500,000$ **do**
    (step 1, get id list)
    for i in range (0, 625) **do**
        (The recent page (footnote 3) stores 10000 projects in maximum, and display 16 projects at once, thus 10000 / 16 = 625 loops would cover all projects.)
        read the JSON file of the $i + 1$ page of recent projects page
        for j in range (0, 16) **do**
            Append the $j + 1$ id in current JSON file to id_list
        end for
    end for

    (step 2, download JSON files)
    for i in range (0, 10000) **do**
        (10000 is the maximum number of projects stored in the recent page)
        generate the api address using the $i + 1$ id in id_list
        download the JSON file by reading the api address
    end for
    $L\_id =$ length of id_list
end while

As can be seen from Algorithm 1, the Scrape part can be divided into two steps: get id list and download the JSON file. For the step of getting id list, a nested-loop, i in range (0,625) and j in range (0,16), is used to scrape id because of the structure of the recent

page[3] of the Scratch website itself. The recent page displays 16 projects by default. Then as we scroll down, another 16 projects appear. With this pattern, we can get an API address that includes 16 projects, and as we change the parameter of address, we can get previous projects' id. For every adjustment of the parameter, which corresponds to the page number after scrolling down, 16 new ids of projects can be found. Then by constantly trying, it can be found that the recent page only contains 10,000 most recent projects in maximum, which means that we can download 10,000 JSON files of projects at most in once. Thus the ranges of the nested-loop are decided. For i, the loop times are the times of visiting scrolled pages, which is 625 (10,000 divided by 16), and for j, the loop times is the volume of id in one page, which is 16.

Also, because of the 10,000 limits in one execution, we need to run this program much time to get enough volume(500,000) of JSON files. To avoid repeated or useless execution, two measures are implemented. The first one is making a time interval between each complete execution by one hour. Due to the website's limited store, we need to wait to update new projects to get the most recent data. The second one is using a list to save the id of downloaded JSON files. When downloading the JSON files according to the id list, every id would be saved in a list. In this process, if the id is detected that it already exists in the list, the download using this id would be interrupted, and the next download starts.

Also, if the access frequency is too fast, the Scratch website would delay responding for a while. Therefore, I set a delay in the requests to avoid the delay in replying. Expressly, I set a time pause of 1 second after every ten requests. The limit of 10 times access in 1 second is written on the website of introduction on Scratch version 2, and fortunately, it also works on current Scratch version 3.

[3]https://api.scratch.mit.edu/explore/projects?limit=16&offset=0&language=en&mode=recent&q=*

4.1.2 Parsing JSON files to store data

Process of Parsing JSON files

As written above, those JSON files store extra information that is not needed for this project. The parsing part concentrates on this problem, and this process will filter out useless parameters for this project in JSON files.

Firstly, each JSON file is read and converted to a dictionary format in python to be available to pick out critical data. During this step, the JSON module in python is used to parse the JSON files. This module reads the JSON file and converts data inside the dictionary format, which enables the program to access data inside those files. Those extracted data are then stored in a MySQL database, which is related to the structure of the project. The details of the database structure are described in the Experiment section below.

Implementation on Parsing JSON files

Firstly, MySQL workbench, an integrated interface of MySQL, is used to create all four tables' items, and parameters in every table are in the MySQL before the parsing on JSON files using python program. Since the creation using this software is more comfortable than using programming codes. Afterward, parsing itself and storing data into the MySQL database are executed in the python program.

In the database, data are assigned to four tables: Projects, Scripts, Blocks, and Monitors. Tables in the database refer to different levels of project structures. Projects table stores the id and the used Scratch version of projects, and it can be seen as the outer and general structure. The second level of structure, which means the information of sprites(objects) contained in the projects, is saved in the Scripts table. Blocks table contains the parameters of blocks, which are the component of scripts. The last table, Monitors, is not a natural part of the structure in projects. It stores the information about the monitor on several parameters in the program.

Table 4.1 to 4.4 show the structure of the 4 created tables in MySQL database separately. 'PK' in these tables refer to the primary key, which is a unique number for identifying the project in the corresponding table. 'FK' in these tables mean foreign key, which shows the relationship between tables and structures' connection. The arranging method of the attributes in tables refers to "A Dataset of Scratch Programs: Scraped, Shaped and Scored."

Table 4.1 shows the database schema of the Projects table. Among the three attributes in the table, 'No.' is the primary key. 'ID' is the original id of the project on the Scratch website. 'version' refers to the Scratch version used when the project is created.

Table	Key	Attribute(Description)
Projects	PK	No.(Numbering of projects)
		ID(ID of projects in Scratch website)
		version(version number of Scratch used in the project)

Table 4.1 Database Schema of table: Projects

Table 4.2 shows the database schema of the Scripts table. Among five attributes in the table, 'Script id' is the primary key. 'Project id' is the original id of the project in the Scratch website, which is used to recognize which project this script belongs to. 'Sprite No' is the serial number of the sprite in the project since usually there would be multiple sprites in a project. 'Sprite name' is the sprite's original name. 'block vol' refers to the number of blocks in the sprite.

This table contains scripts in a unit of the sprite. Since the Scratch project's JSON file contains scripts in the same sprites together, this structure is more suitable for transferring data from JSON files. To split scripts in one sprite, the Blocks table would help a lot.

Table	Key	Attribute(Description)
Scripts	PK	Script_id(numbering of scripts)
	FK	Project_id(ID of project which the script belongs to)
		Sprite_No(numbering of sprite in the project)
		Sprite_name(name of sprite in the project)
		block_vol(number of blocks in the script)

Table 4.2 Database Schema of table: Scripts

Table 4.3 shows the database schema of the Blocks table. 'Block id' is the primary key. 'Project id', 'Sprite No', 'Sprite name' has the same meaning and usage as table 2. 'Block No' refers to the order of the block in the sprite. 'Block name' is the original name of the block. 'opcode' means the settled general type of blocks in Scratch. 'next' and 'parent' contain the name of the next and previous block, respectively. 'para1' and 'para2' are parameters contained in the blocks according to their general type.

As mentioned above in Table 4.2, parameters in this table are suitable for splitting scripts in the same sprite. The key attributes are 'Block name', 'next', and 'parent'. These three attributes can be used to connect blocks and generate the structure of scripts. Besides, by counting the Null value of 'parent' with the same 'Sprite No' and 'Sprite name', initials of scripts can be found, which also means the number of scripts in the sprite.

Table	Key	Attribute(Description)
Blocks	PK	Block_id(numbering of blocks)
	FK	Project_id(ID of project which the script belongs to)
		Sprite_No(numbering of sprite in the project)
		Sprite_name(name of sprite in the project)
		Block_No(numbering of block in the sprite)
		Block_name(name of block)
		opcode(type of block)
		next(name of next block of current block)
		parent(name of parent block of current block)
		para1(value of parameter 1)
		para2(value of parameter 2)

Table 4.3 Database Schema of table: Blocks

Table 4.4 shows the database schema of the Monitors table. 'Monitor id' is the primary key. 'Project id' is the id of the project, which contains this monitor section. 'Monitor name' refers to the name of the monitored target. 'mode' means the name of the data type of target. 'opcode' refers to the general type of monitor in Scratch. 'Sprite name' is the name of the sprite, which contains the monitored value.

Table	Key	Attribute(Description)
Monitors	PK	Monitor_id(numbering of monitors)
	FK	Project_id(ID of project which the script belongs to)
		Monitor_name(name of monitored target)
		mode(mode of monitor)
		opcode(type of monitor)
		Sprite_name(name of monitored sprite)

Table 4.4 Database Schema of table: Monitors

Algorithm 2 shows the process of the Parsing part. As can be seen from the algorithm, in this part, the main work is to build the database in MySQL using the parameters parsed from JSON files. In the algorithm, firstly, the MySQL database is built using MySQL Workbench. The details of the database are explained above. Then the database is connected with the python program to contain the parsed data from JSON files. For the parsing of JSON files, after reading the list of the id of JSON files, the task can be divided into four steps, and each step corresponds to one table in the MySQL database. Every step is represented by loops used to access all JSON files. In one cycle of the loop, the next moves are implemented: Firstly, the JSON file with the corresponding id is opened, and JSON format data inside would be transferred to dictionary format. Secondly, the required parameters are picked out in the dictionary. Details of the required parameters are written with four tables in this report above. Finally, those parameters are written into the MySQL database.

Algorithm 2 Parsing JSON files of Scratch projects.

Build a new MySQL database
Connect to MySQL database using Python
L=number of JSON files
Append all id of JSON files into list $List\_J$
for i in range $(0, L)$ **do**
    $Project\_id = List\_J[i]$
    Open the JSON file which $id = List\_J[i]$ as dictionary $dict\_j$
    Extract $version$ from $dict\_j$
    Insert $Project\_id$ and $version$ in this JSON file into table: Projects in MySQL database
end for
for i in range $(0, L)$ **do**
    Open the JSON file which $id = List\_J[i]$ as dictionary $dict\_j$
    Extract $Project\_id$, $Sprite\_No$, $Sprite\_name$, $Block\_vol$ from $dict\_j$
    Insert above parameters into table: Scripts in MySQL database
end for
for i in range $(0, L)$ **do**
    Open the JSON file which $id = List\_J[i]$ as dictionary $dict\_j$
    Extract $Project\_id$, $Sprite\_No$, $Sprite\_name$, $Block\_No$, $Block\_Name$, $opcode$, $next$, $parent$, $para1$, $para2$ from $dict\_j$
    Insert above parameters into table: Blocks in MySQL database
end for
for i in range $(0, L)$ **do**
    Open the JSON file which $id = List\_J[i]$ as dictionary $dict\_j$
    Extract $Project\_id$, $Monitor\_No$, $monitor\_name$, $mode$, $opcode$, $sprite\_Name$ from $dict\_j$
    Insert above parameters into table: Monitors in MySQL database
end for
Execute pymysql.commit() to implement the insert command

4.2 Similarity Calculation on Scripts and Scratch Projects

This is the second part of the project, and the goal is to get a evaluation of similarity between Scratch projects.

In summary, this part is divided into three steps: generating a script dataset (section 4.1 and 4.2), calculating the similarity of scripts (section 4.3.1-4.3.3), and calculating the similarity of projects (section 4.3.4). The last two steps could be further decomposed into the distance calculating part and the clustering part. The detailed steps are as follows:

Firstly, using the dataset, which stored the blocks and the relationship between blocks extracted from JSON files to generate the scripts' dataset, which is ordinally combined blocks. Secondly, using the Simhash algorithm to calculate the hamming distance between scripts. In this step, volumes of blocks in each type are used as the vectors to calculate distances. Those distances are used to compute the clusters in the next clustering step. Thirdly, using the k-means++ algorithm to calculate the similarity of scripts, thereby getting the scripts' clusters. Fourthly, the results obtained above are used as the projects' vectors to calculate distances using the Simhash algorithm. Finally, using the result above to cluster and calculate the similarity of projects.

4.2.1 Data Processing on JSON Files

Process of Data Processing

According to the structure of JSON files downloaded from the Scratch website, following the practice described in section 4.1, we stored data in the format of blocks and relationships between blocks, specifically, the name of a block of the previous block and the next block in the script. Therefore, the construction of scripts could be obtained from these relationships of blocks.

To structure the scripts, the following process was followed: Firstly, grouping the blocks that belong to the same project on the same list. Secondly, every list picks out the first

block of all scripts, which do not record the last block name. Every first block would be the first individual in new lists, which referred to scripts. Thirdly, distributing other blocks in the previous list to new lists according to their recorded previous block name in the stored data. Finally, lists of the script, which are composed of blocks, are generated. Afterward, distances of these lists of scripts for the clustering are needed to be calculated. The SimHash Algorithm is used in this step.

Implementation on Data Processing

With the experiment in the first part, we get a dataset of scratch projects' structure and the structure of blocks in those projects. The dataset contains 508,704 Scratch 3.0 projects. In this experiment, we firstly do the data processing on these projects.

As the research project shows, since the JSON files store relationship between blocks, instead of storing scripts directly, we need to get the dataset of scripts for the last experiment. Therefore, I use a dataset of projects and blocks to generate a dataset of scripts. Algorithm 3 and 4 show the process of generating scripts.

Algorithm 3 describes the first step, which is grouping of the blocks according to projects they belong to. This algorithm uses a loop to extract information on which project the block belongs to from the MySql dataset. In each loop, firstly, we use the package PyMySql in python to implement SQL code to get the current block's status from the dataset. The status is specifically its block id and the project id it belongs to. Secondly, the Project id is checked for splitting blocks. If this block is the first one or has a different project id with the previous one, then create a new list with a serial number of the project, and add the current block id into the list. The serial number of projects is used for the loop in Step 2. If the project id is the same as the previous, add block id to the same list with the previous one. Repeat these two implements until all blocks are in lists. This approach works because during the research project, JSON files, which each only contains one project, therefore blocks in one project are always continuously.

Algorithm 3 Generate Scripts. Step 1: Group blocks according to projects they belong to.

Use Sql code to get the number of blocks stored in dataset, named n_block
list_id=list of
for $i = 0 : n\_block$ **do**
    Use Sql code to get status of blocks which $block\_id = i$
    $Project\_id\_i$ = id of project which consists of block i
    if $i == 0$ **then**
        Create a blank list(contains blocks in same projects) $list\_project$ and add i in it, assign a serial number to the list
        $Project\_id\_t = Project\_id\_i$
    end if
    if $Project\_id\_t! = Project\_id\_i$ and $i > 0$ **then**
        Create a list and append i in it, assign a number to the list
        $Project\_id\_t = Project\_id\_i$
    else
        append i into current created list
    end if
end for

Algorithm 4 describes the second step, which is grouping blocks to scripts. In this step, since JSON files do not contain blocks in the continuous order to build up scripts (which can be surmised that blocks may be stored in the order of created time,) approaches are different from the first step. Step 2 is divided into two loops. In the first loop, firstly, in each loop, I use SQL code to get its block id, project id, its original name in the original project, and original name of its 'parent'. 'Parent' here means the previous block in the script, which is according to JSON files' storage structure. Secondly, if the 'parent' name is blank, which means this block is the first block of a script, then create a new list with the serial number of

the script and add the current block id into the list. If added, delete the block from the list of projects. The serial number of the script is used in the next loop. The first loop continues until all blocks are checked. In the second loop, each loop handles one list of projects and continues until the project's list becomes a blank one. In the loop, all blocks in the project list are checked if its 'parent' is contained in any list of the script in the same project. If the 'parent' is found, add a block to the same list of the script with 'parent' and delete the block in the projects' list. The 'parent' individuals are judged by the original name of blocks in JSON files from the MySql dataset. Simultaneously, the project and script lists are recorded and extracted using serial numbers of project and script in the code.

With the above experiments, lists of scripts contain blocks are generated.

Algorithm 4 Generate Scripts. Step 2:Group blocks to scripts.

1:
2: Get number of created lists in step 1 as n_list
3: Each created list $list\_project$ is named $list\_project\_[0 : n\_list]$
4: **for** $i = 0 : n\_list$ **do**
5: **for** $j = 0 : list\_project\_i$ **do**
6: Use Sql code to get status of block j in $list\_project\_i$
7: **if** Block j has a blank 'Parent' **then**
8: Create a list(contains blocks in same script) named $list\_script$ and append $Block\_id$ of block j in it, assign a serial number to the list, and also record which project's blocks they contain
9: Delete block j in $list\_project[i]$
10: **end if**
11: **end for**
12: **end for**
13: **for** $i = 0 : n\_list$ **do**
14: $len\_list\_project$=length of $list\_project\_i$
15: **while** $len\_list\_project! = 0$ **do**
16: **for** $j = 0 : len\_list\_project$ **do**
17: **for** $k = 0$: number of $list\_script$ included in same project **do**
18: **if** block j has a 'Parent' in $list\_script[k]$ **then**
19: append block j into $list\_script[k]$
20: record that block j is appended
21: break
22: **end if**
23: **end for**
24: **end for**
25: Delete recorded appended blocks from $list\_project\_i$
26: **end while**
27: **end for**

4.2.2 Use of the SimHash Algorithm to Calculate Distances between Scripts

Process of Distance Calculation between Scripts

Since scripts in Scratch are consist of blocks, different numbers and types of blocks could inform the index of similarity between scripts. Therefore, I decide to use the method of the Simhash algorithm to calculate the distances between scripts. In this method, blocks are treated as vectors of scripts. The type of blocks is the number of vectors, and the number of same type blocks in one script referred to the weight of each vector. With these distances, the clustering calculation using the K-means++ algorithm could be implemented.

At the step of calculating distances between projects, the same method could be applicable. The detail is written in the section on implementing projects below.

Implementation on Distance Calculation between Scripts

With lists of scripts obtained from the previous section, the hamming distances could be computed. From the downloaded JSON files, we extracted 278 different block types, and when generating vectors, each type refers to one element in each vector. Therefore, every vector contains 278 elements. All 278 elements between two scripts should be subtracted to calculate the total distance during the distance calculation. Although this 278 subtraction in one distance calculation could finish instantly, the calculation of all distances does not. In the early stage of the experiment, the distance calculation part is included in the clustering part. However, during the experiment, we find that distance calculation for all spends too much time. Therefore, I decided to directly calculate all distances and store them into a dataset for similarity calculation. However, the time cost of one total distance calculation would grow exponentially as the total volume of projects. To calculate 508,794 projects' distances, according to the rough calculation based on the time cost of a small part computing, it would spend in years to finish. Considering the time cost, I reduced the number

of projects to 10,000, which takes acceptable time cost in computing. After the reduction, it took around ten days to get all distances. The volume of scratch projects is reduced from 508,704 to 10,000. Besides, I have categorized scripts according to their length and omitted distance calculations between long scripts (equal to or longer than 10) and scripts that have a massive difference in length between each other in order to reduce the time cost of distance calculations between scripts. For the original volume, the distance calculation would spend years finishing computing according to the calculation on small test parts of scripts. While after the reduction, the total calculation time could be compressed to 1-2 weeks, which is acceptable. Algorithm 5 6 and 7 show the process of the calculation of distances.

Algorithm 5 describes the first step to calculate distance: generate vectors. This step creates vectors of scripts according to the type of blocks contained. Firstly, the list of block types names is generated according to the types of the block stored in the MySql dataset. Secondly, for all lists of the script, new lists for storing the vectors are created. The numbers of all block types included in the corresponding script are counted and stored in each vector list.

These are specific steps of storing vectors: 1, create an empty list of the vector corresponds to a script list. 2, count the number of blocks with one of the types and add them to the vector list. 3, loop step 2 until all types of blocks are counted. 4, loop steps 1 to 3 until the vector list is generated for all script lists.

Algorithm 5 Calculate Distance. Step 1: Generate vectors.

Use Sql code to get the name of all block types
block_type=list of name block type
len_bt=length of *block_type*
n_s = number of *list_script*
for $i = [0 : n\_s]$ **do**
    $len\_sc = list\_script\_i$
    Create a blank list $list\_vector\_i$ which has same serial number with $list\_script\_i$
    for $j = [0 : len\_sc]$ **do**
        $Counter\_v = 0$
        for $k = [0 : len\_bt]$ **do**
            if $list\_script\_i[j] == block\_type[k]$ **then**
                $Counter\_v+ = 1$
            end if
            Append $Counter\_v$ into $list\_vector\_i$
        end for
    end for
end for

Algorithm 6 describes the second step of calculating distance: divide scripts according to length. In this algorithm, scripts are divided into ten different lists according to their length to reduce the latter calculation of the Simhash algorithm. For the length computing, I use a loop to add all value in one vector list to get them since the volume of vectors is also the number of blocks.

Algorithm 6 Calculate Distance. Step 2: Divide scripts according to length.

1: Create 11 lists $list\_len$ which have serial numbers $1 - 10$

2: **for** $i = [0 : n\_s]$ **do**

3: sum_v = sum of vectors in $list\_vector\_i$

4: append $list\_vector\_i$ into $list\_len$ which serial number = sum_v (if $sum\_v > 10$, append into $list\_len\_10$)

5: **end for**

Algorithm 7 describes the third step of calculating distance: implement Simhash Algorithm. In this algorithm, only scripts of length 2-9 have participated in the calculation. The reason for removing scripts of length 1 is because Scratch programs valid from the combination of at least two blocks. While the reason for removing scripts of length ten or more is more based on time constraints, and they have less potential to be similar to others due to their length. The algorithm only compares scripts that can be similar; expressly, I set a rule that when a script has over 50% structure are the same with another one, they have a chance to be similar. In this experiment, if half blocks or more in a script are the same, their distance is calculated. If that distance is shorter than half of those two's shorter script, information of this distance (two individuals and the length of distance) would be recorded.

For the algorithm's implementation, firstly, two vector lists are picked out for the distance calculation. Secondly, get the sum of both vectors and pick out the smaller sum. Thirdly, calculate the difference between each corresponding vector of two lists and calculate the sum of them. Fourthly, if the sum of difference is less than half of the smaller sum of vectors, add information of this distance (id of two scripts and value of distance) into the dataset.

Algorithm 7 Calculate Distance. Step 3: Implement Simhash Algorithm.

(Only implement on $list\_len\_2$, $list\_len\_3$, ..., $list\_len\_9$)

(The loop below describes one process between two scripts that have chance to be similar.)

Set two $list\_len$ as $list\_len\_a$ and $list\_len\_b$

sum_v_a = sum of vectors in $list\_len\_a$

sum_v_b = sum of vectors in $list\_len\_b$

$sum\_min = min(sum\_v\_a, sum\_v\_b)$

$sum\_dis = 0$

for $i = [0 : len\_bt]$ **do**

    $sum\_dis = sum\_dis + |list\_len\_a[i] - list\_len\_b[i]|$

end for

if $sum\_dis < sum\_min/2$ **then**

    Add $[list\_len\_a, list\_len\_b, sum\_dis]$ to dataset

end if

4.2.3 Use of the K-means++ Algorithm to Cluster Scripts

Process of Clustering Scripts

The method of K-means++ Algorithm is used to cluster scripts. For the initialization, k different scripts were chosen as the center points of clusters. Therefore, k is also the number of clusters. Center points are the first individuals in those clusters. Other scripts are assigned to those clusters according to the distance between those center points. Generally, they are added to the cluster where the closest center point belonged to. For the specific process, step 1 is randomly choosing a script that does not belong to any cluster as a center of a new cluster. Step 2 is adding all other scripts, which do not belong to any cluster, and which are close to the center point script. Loop these two steps until all scripts belong to clusters.

When this process is finished, the first iteration of clustering is also complete.

From the second iteration, each whole iteration is a loop. Firstly, picking out one new center individual has the least total distance from all other individuals from each cluster. Secondly, abandoning old relationships in clusters and redistributing all other individuals to new clusters to which the closest center individual belongs.

Several adjustments on the K-means++ algorithm have been made accordingly to adjust to this specific experiment. Firstly, in the original K-means++ Algorithm, distances between individuals are calculated during the clustering process, while to reduce time costs of repeating the calculation of distance, in this project, I calculate them in advance and store them into a dataset, thereby accelerating the speed of clustering computing. Secondly, the way of determining k value was according to the connections between individuals. Elbow Method, Silhouette Coefficient, and Canopy clustering algorithm are commonly used to determine the k value. While for this project, since Elbow Method and Silhouette Coefficient would use different k value to do repeated testing to get the best solution, finally, they are not suitable because of too long time cost. For the Canopy clustering algorithm, since different scripts have no similarity, this algorithm could not help me get the rough k value. I choose to determine the k value written in the first iteration above, which uses actual conditions in this project as constraints to determine the k value. Thirdly, New center individuals during the second and later iterations are chosen from original individuals instead of computing new center points and inserting them into clusters. This adjustment is due to the dataset's structure, which only contained information on distances between scripts instead of vectors. While still because of this dataset, scripts without distance connections would not be distributed to the same cluster, which solves the drawback of always using original individuals as center points to some extent.

Implementation on Clustering Scripts

The clustering scripts experiment implements the method of K-means++ algorithm combined with the actual situation in the project and is divided into two iterations. This part uses the dataset of distance generated in the previous experiment to fasten the calculation speed. Algorithm 8, 9 , 10 and 11 describe the process of clustering scripts.

Algorithm 8 describes the first iteration of clustering. The specific procedure is:

1) Create a list of $list\_all\_2$, which contains all ids of scripts with distances between others (According to the dataset of distance information).

2) Randomly choose a script from the list, create a list representing a new cluster and add those lists in it. Then remove that script from $list\_all\_2$.

3) Add id of all scripts in $list\_all\_2$, that have a distance with the script chosen above, into the created cluster list. Then remove those scripts from $list\_all\_2$.

4) Loop 2) and 3) until the length of $list\_all\_2$ equals to or is smaller than 1. If equals 1, add the last script to a cluster list, which contains the script that has a distance from the last one.

This algorithm performs the initialization of clustering according to the concept of k-means++ algorithm, choosing points far apart as initial center points of cluster. According to the structure of the distance dataset, it is elementary to find all other scripts that have a distance value with the center point. This factor has two advantages in fastening the speed of clustering. The first one is: it would be faster to find scripts in one cluster. Generally, in the K-means++ algorithm, the distance between individuals need to be calculated temporary. While in this project, since distances are calculated previously, close scripts could be obtained in batches by searching the dataset. This character shortens the time of generating clusters. The second advantage is: it would be easier to find the next center point. Since scripts with

distance values between them can be easily found, the opposite is also the same. Scripts without distance values could be treated as they sandwich a very long distance because they have no similarity. Therefore, when a center point of a cluster is decided, the range of other center points can be narrowed quickly.

Algorithms 9, 10 and 11 describe different steps of the second iteration of clustering. These three algorithms constitute a complete iteration. Algorithm 9 is the initialization of the iteration; Algorithm 10 is the main clustering part; Algorithm 11 is the subsequent processing after clustering.

Algorithm 9 describes the process of getting new center points in each cluster. The specific procedure is:

1) Create a new blank list *list_center* to contain new center points in each cluster.

2) Pick up a script in a cluster generated in the previous iteration, calculate the sum of distance values between this script and all other scripts in the same cluster. Whenever there is no distance value between two scripts, add 999999999 (a substantial value) to the sum.

3) Implement 2) to all scripts in a cluster, the script with the smallest sum is added to *list_center* as a new center point. If there are multiple smallest sum, choose one of them randomly.

4) Implement 2) and 3) to all clusters to get all new center points of clusters.

In this algorithm, new center points are decided according to their smallest sum of distance in each cluster. Since the center belongs to the cluster, scripts that have a lack of distance values would be given a large penalty in the sum calculation by adding a tremendous value. This makes sure that scripts that have the most "connection" on distance with others would have the most significant chance to be the center point.

Algorithm 10 describes the clustering process in the second iteration. The specific procedure is:

1) Create a new blank cluster list for every new center point in $list\_center$.

2) Add all other script id (except center point scripts) to $list\_store\_it2$.

3) Pick up a script in $list\_store\_it2$, then check the distance between this script and all scripts in $list\_center$ on by one. Add those script to a cluster list which contains the closest center point script. If there are multiple closest scripts, choose one randomly. If there is no distance value between any of the center point, add this script to the cluster where it belongs in the previous iteration (according to the serial number of clusters in both iterations).

4) Implement 3) to all scripts in $list\_store\_it2$.

This algorithm is the central part of the iteration. It redistributes scripts to the same or different clusters, which improves the accuracy of similarity between scripts. This algorithm's main difference with the general K-means++ algorithm is that this algorithm uses individuals that already exist as center points of clusters instead of creating a new center point for clusters. This change has both advantages and disadvantages. The disadvantage is that the center points maybe not the absolute center point. While on the other hand, since those center points are some of the individuals already exist in the dataset, they can be picked out as the representative of corresponding clusters for later research, which is the advantage of this change. Another advantage is that the dataset of distance can be used directly to generate new clusters, accelerating the calculation speed.

Algorithm 11 describes the subsequent processing on clusters that contain only one script after clustering. The specific procedure is:

1) Check cluster lists after clustering, and pick out all lists which only contain one script. Add those scripts into $list\_one\_script$.

2) Remove those scripts from *list_center*.

3) Pick up a script in *list_one_script*, then check the distance between this script and all scripts in *list_center* on by one. Add those script to a cluster list which contains the closest center point script. If there are multiple closest scripts, choose one randomly. If there is no distance value between any of the center point, do nothing to it.

4) Implement 3) to all scripts in *list_one_script*.

This algorithm is another change from the general K-means++ algorithm, which is doing extra clustering calculation on each solitary script, which occupies the whole cluster by itself. Compared with the disadvantages of reducing the number of clusters, which affect little due to its small size (only one script in a cluster), giving them a chance to converge with similar scripts with them helps increase the quality of similarity.

Algorithm 8 Cluster scripts. Iteration 1

Load dataset of distance information as arrays.
According to recorded script ids in dataset, create a dataset of scripts named $list\_all$.
$list\_all\_2$=$list\_all$
while $len(list\_all\_2) > 1$ **do**
    Randomly choose a script in $list\_all\_2$ named s_rdm.
    Remove s_rdm from $list\_all\_2$.
    Create a blank array and add s_rdm in it.
    Search distance arrays and get the list of scripts that has a distance with s_rdm named $list\_rdm$
    Add all scripts in $list\_rdm$ to the array which s_rdm belongs to.
    Remove all scripts in $list\_rdm$ from $list\_all\_2$
end while
Count the generated lists($list\_generate$) number $list\_num$
for $i = [0 : list\_num]$ **do**
    if $len(list\_generate[i]) == 1$ **then**
        s_single = the only script in $list\_generate[i]$
        for $j = [0 : list\_num]$ **do**
            if $j! = i$ **then**
                Search distance arrays to check if s_single has a distance with any scripts in $list\_generate[j]$
                if $list\_generate[j]$ has any scripts that has a distance with s_single **then**
                    Add s_single into $list\_generate[j]$
                    Remove s_single from $list\_generate[i]$
                end if
            end if
        end for
    end if
end for

Algorithm 9 Cluster scripts. Iteration 2. Step 1: Calculate new center points.

```
list_new_center = []
for i = [0 : list_num] do
    for j = [0 : list_generate[i]] do
        distance_sum = 0
        ind_j = list_generate[i][j]
        if j == 0 then
            ind_cen = ind_j
            dis_min = 999999999
        end if
        for k = [0 : list_generate[i]] do
            if j! = k then
                ind_k = list_generate[i][k]
                dis_jk = sum(|ind_j − ind_k|)
                distance_sum = distance_sum + dis_jk
            end if
        end for
        if distance_sum < dis_min then
            list_new_t = []
            list_new_t.append(ind_j)
            dis_min = distance_sum
        end if
        if distance_sum == dis_min then
            list_new_t.append(ind_j)
        end if
    end for
    list_new_center.append(a random value in list_new_t)
end for
```

Algorithm 10 Cluster scripts. Iteration 2. Step 2: Clustering.

Create lists $list\_cluster\_it2$ which number equal to the length of $list\_new\_center$, using scripts in $list\_new\_center$ as new center points.(Also have same serial number with $list\_generate$)
Add all script id into $list\_script\_store\_it2$.
Remove all script id in $list\_new\_center$ from $list\_script\_store\_it2$.
for $i = [0 : len(list\_script\_store\_it2)]$ **do**
    $script\_t = list\_script\_store\_it2[i]$
    Calculate distances between $script\_t$ and all scripts in $list\_new\_center$
    if The smallest distance exists **then**
        Add $script\_t$ to $list\_cluster\_it2$ which corresponding closest center point belongs to.
    else
        Add $script\_t$ to $list\_cluster\_it2$ which has the same serial number with $list\_generate$ which $script\_t$ belongs to in previous iteration.
    end if
end for

Algorithm 11 Cluster scripts. Iteration 2. Step 3: Processing on clusters which only contains one script.

Check all $list\_cluster\_it2$ and pick out lists which only contains one script.
Store those scripts into $list\_one\_script$
Delete those lists from all $list\_cluster\_it2$
$list\_new\_center2$ = rest of scripts after remove $list\_new\_center$ from $list\_new\_center$
for $i = [0 : len(list\_one\_script)]$ **do**
    $script\_t = list\_one\_script[i]$
    Calculate distances between $script\_t$ and all scripts in $list\_new\_center$
    if The smallest distance exists **then**
        Add $script\_t$ to $list\_cluster\_it2$ which corresponding closest center point belongs to.
    end if
end for

4.2.4 Distance Calculation and Clustering on Scratch Projects

Process of Clustering on Scratch Projects

After the similarity of scripts is calculated, this result is used as vectors of projects since Scratch projects are consist of scripts. The number of script clusters is the number of vectors (as the type of blocks for scripts), and the numbers of scripts in the same cluster are weights of vectors(as the number of same type blocks in the same script). Therefore, distance calculation methods and clustering implemented on computing similarity on scripts could also apply to computing similarity on projects.

About the method of implementation on the Simhash algorithm and K-means++ algorithm in clustering Scratch projects, since the processes are totally the same with experiments

executed in clustering scripts, details of these parts would not be described in this section.

The only difference between clustering on scripts and Scratch projects is the process of generating vectors. Algorithm 12 shows the process of converting script clusters to vectors of Scratch projects. The specific procedure is:

1) Pick up a cluster of the script (generated in clustering scripts process), then create an empty list $list\_cluster\_project\_id$ corresponding to the script cluster. For every script in the cluster, add the corresponding project id, which that script belongs to, into that list.

2) Implement 1) to all clusters of the script.

3) Create lists of project vectors $list\_project\_vector$, which contains several 0. The volume of 0 is the same as the number of script clusters. Every value (currently is 0) corresponds to a script cluster.

4) Get a project id in $list\_cluster\_project\_id$, add 1 to a value, which corresponds to the same script cluster with $list\_cluster\_project\_id$, in $list\_project\_vector$.

5) Implement 4) to all project ids in all $list\_cluster\_project\_id$.

This algorithm's main point is to connect the result of the clustering script to the last clustering project. Since every script belongs to one of the projects, I copy the script clusters and replace script ids with project ids to which the corresponding script belongs. Therefore, the source data of the Scratch project could be generated. Afterward, the work is to turn the main body from clusters of project ids to vectors. I treat every cluster as a way of vector, then elements in clusters would be the volume of vectors of Scratch projects. Then I distribute those elements to project vectors, and the complete vectors are generated. After this process, the Simhash algorithm and K-means++ algorithm is implemented for Scratch project vectors.

Algorithm 12 Convert script clusters to vectors of Scratch projects.

1: Load clusters of scripts as lists $list\_script\_cluster[0:i]$
2: Connect to MySQL database
3: **for** $i = [0 :$ number of $list\_script\_cluster]$ **do**
4: Create a new list $list\_cluster\_project\_id[i]$
5: **for** $j = [0 :$ length of $list\_script\_cluster[i]]$ **do**
6: Get the Script id of $list\_script\_cluster[i][j]$ as $script\_id\_t$
7: Check the MySQL database to find $script\_id\_t$ belongs to which Scratch project and get the project id as $project\_id\_t$
8: Add $project\_id\_t$ in $list\_cluster\_project\_id[i]$.
9: **end for**
10: **end for**
11: Create a blank list $list\_vector\_project$ for every Scratch project which id is included in $list\_cluster\_project\_id$
12: **for** $i = [0 :$ number of $list\_vector\_project]$ **do**
13: **for** $j = [0 :$ number of $list\_script\_cluster]$ **do**
14: Add 0 to $list\_vector\_project[i]$
15: **end for**
16: **end for**
17: **for** $i = [0 :$ number of $list\_cluster\_project\_id]$ **do**
18: **for** $j = [0 :$ length of $list\_cluster\_project\_id[i]]$ **do**
19: Set project id of $list\_cluster\_project\_id[i][j]$ as $project\_id\_t$
20: Get corresponding $list\_vector\_project$ which has the same id with $project\_id\_t$, set that list as $list\_vector\_project\_t$
21: Add 1 to number i value in $list\_vector\_project\_t$
22: **end for**
23: **end for**

Chapter Five

Results

5.1 Created dataset of Scratch Projects

5.1.1 Scraping part

The download of JSON files started on October 22nd, 2019, and ended on December 24th, 2019. The downloading lasted for almost two months. 60,000 to 80,000 projects were updated every week, which means that the same volumes of JSON files could be scraped from the current page on the Scratch website.

During this period, 510,609 JSON files were downloaded from the Scratch website. These files occupy the capacity of around 68 gigabytes on the computer. However, when checking the format of JSON files, there are two different types of formats. Comparing the difference of contents inside between two types, I found that the format difference came from different Scratch versions (version 2 and version 3) used by projects. Therefore, during the parsing of JSON files, files with different versions should be treated differently.

5.1.2 Parsing part

As mentioned above, there are two different formatting types of JSON files (version 2 and version 3). Therefore, parsing's first step was to divide JSON files into two groups according

to their Scratch versions. After the separation of JSON files, there are 508,704 projects using version 3 and 1851 projects using version 2. Besides, 54 JSON files are garbled inside. Those garbled files are abandoned in the latter parsing. Then about projects using version 2, since their volume 1851 occupies only a small part of all projects and has different data structures, these data were removed from the dataset, considering time and efficiency. Consequently, the final source JSON files of recent Scratch projects have a volume of 508,704. After the parsing of JSON files, the Projects table contains 51,055 rows, which is the only table that includes information about projects using Scratch version 2. The latter tables only store parameters of projects using Scratch version 3. Scripts table contains 4,992,727 rows. The monitors table contains 427,479 rows.

The code of this process is available at Github[1], and the parsed dataset is available at Google drive[2].

5.2 Similarity Calculation on Script Level

During the similarity calculation process, several data were obtained. Since the calculation on the full dataset spend too long time, we have to reduce the dataset for analysing. After reducing the volume of projects to 10,000 (chosen randomly from the original 508,704 Scratch projects in the dataset), this smaller dataset contains 552,839 scripts. Table 5.1 shows the specific numbers of scripts that consist of different numbers of blocks in reduced 10,000 projects.

[1]https://github.com/skylartsy/Scraping-and-Parsing-JSON-files-from-the-Scratch- Website

[2]https://drive.google.com/file/d/1hK0EsemPBg4SBrLnq92vXCkvqONbKv96/view?usp=sharing

Number of Blocks in a Script	Number of Corresponding Scripts
1	165,032
2	103,304
3	49,638
4	33,601
5	23,539
6	21,386
7	17,403
8	14,648
9	11,476
Equal to or more than 10	112,812

Table 5.1 Quantities of scripts of different size (10,000 projects).

Figures 5.1 and 5.2 show the result of the distribution of scripts in clusters after the first and the second iteration, respectively. Each bar refers to a cluster. As Figure 5.1 shows, in the result of the first iteration, there are 1,207 clusters in total, and the largest cluster contains 47,404 scripts. The largest cluster is more extensive than all other clusters; all other clusters do not even come to half of the largest one. Few clusters contain over a thousand scripts, and others contain hundreds. Most clusters contain dozens or just several scripts. As Figure 5.2 shows, in the result of the second iteration, there are 843 clusters in total. The largest cluster contains 19,313 scripts. The second-largest cluster contains 16,144 scripts, which is close to the size of the largest one. For other clusters, several contain thousands or hundreds of scripts, and other most clusters contain dozens or several scripts.

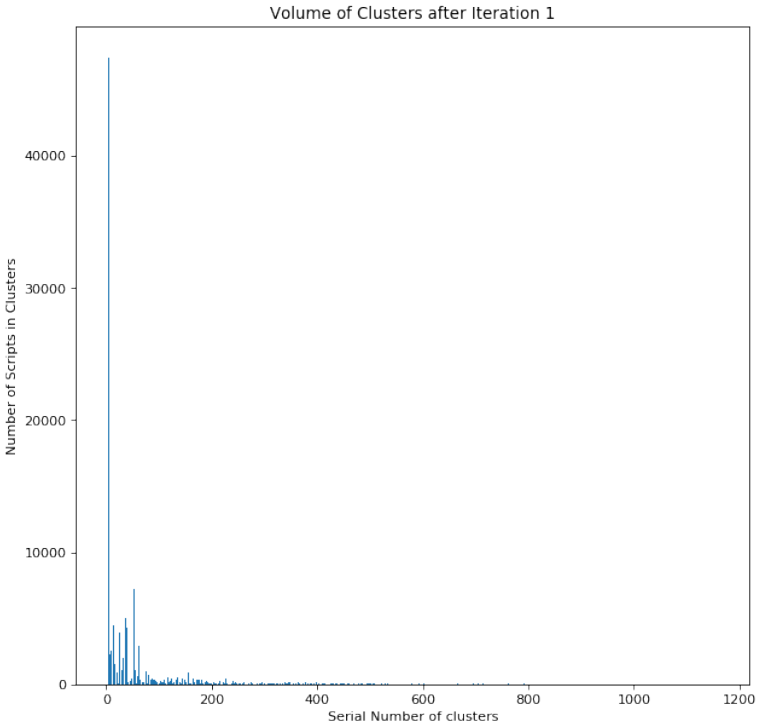


Figure 5.1 Volume of Script Clusters after Iteration 1.

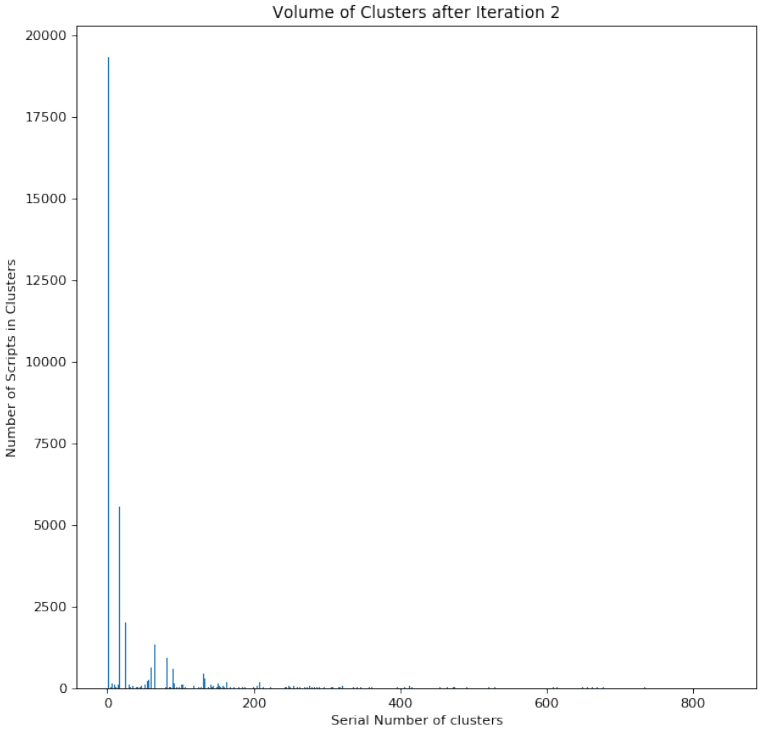


Figure 5.2 Volume of Script Clusters after Iteration 2.

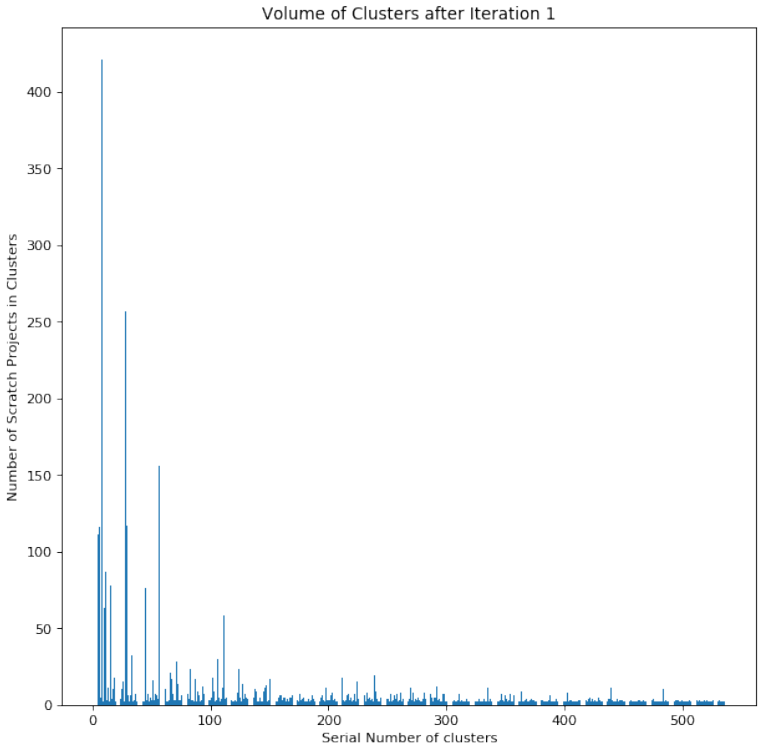


Figure 5.3 Volume of Scratch Project Clusters after Iteration 1.

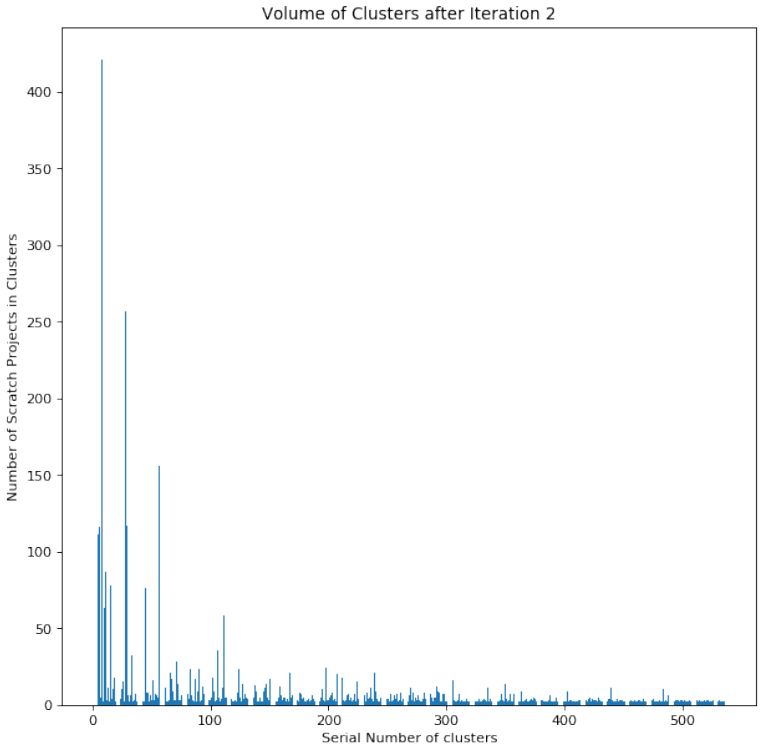


Figure 5.4 Volume of Scratch Project Clusters after Iteration 2.

5.3 Similarity Calculation on Script Project Level

Figures 5.3 and 5.4 show the result of the distribution of Scratch projects in clusters after the first and the second iteration, respectively. Each bar refers to a cluster. As Figure 5.3 shows, as a result, of the first iteration, there are 534 clusters in total. The number of projects which belong to any of the clusters is 5444 out of the 10,000. The largest cluster contains 421 Scratch projects, and the second-largest cluster contains 257 projects. Several clusters contain more than 100 projects, and others contain dozens of projects. As Figure 5.4 shows, as a result, of the second iteration, there are 534 clusters in total. The number of projects which belong to any of the clusters is 4387. The largest cluster contains 421 projects. The second-largest cluster contains 257 projects. Several of them contain more than 100 projects for other clusters, and others contain dozens of projects.

Serial Number of Cluster	Volume of Cluster	Project id of the Center Point
5	421	346141665
25	257	342703760
53	156	346590306
0	132	349506831
26	117	344013337
3	116	348380741
2	111	355429803
8	87	349171132
12	78	345822731
42	76	349097738

Table 5.2 Serial number, volume and center point id of top 10 largest clusters.

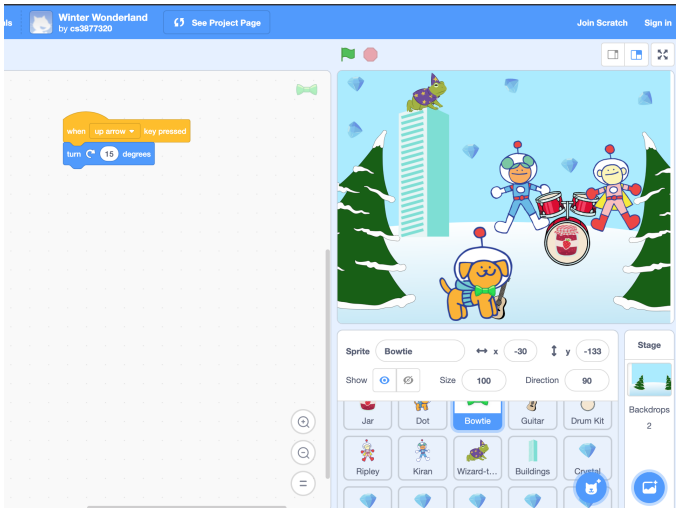
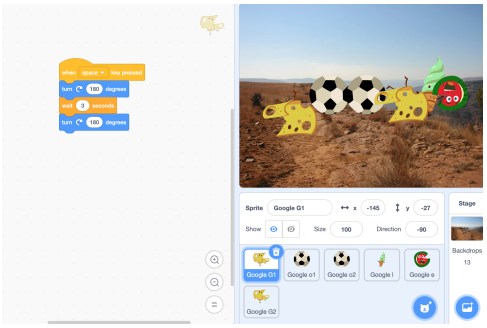
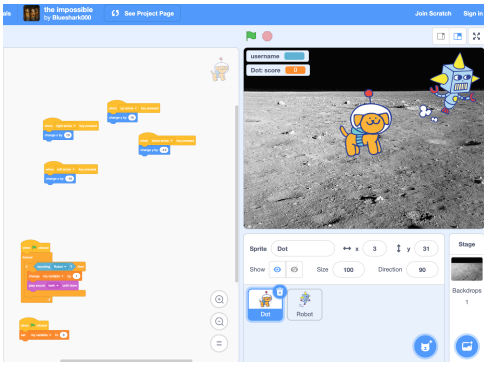


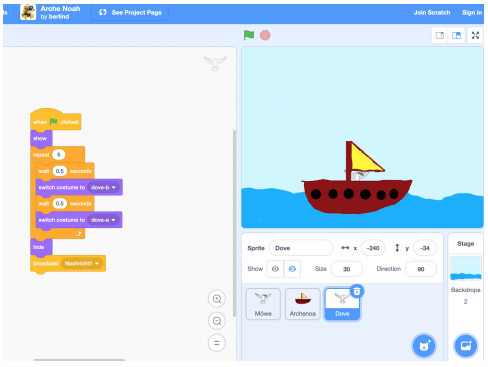
Figure 5.5 Scratch edit page of the center point project (id: 346141665) in the largest cluster.



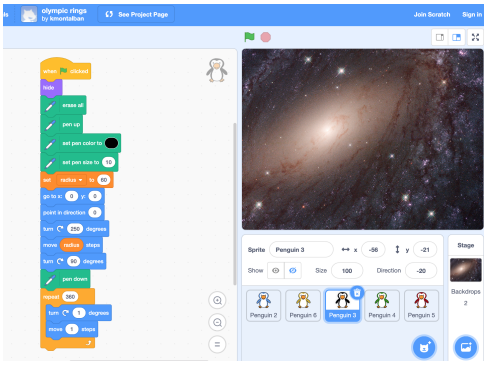
(a) Project id: 346141507.



(b) Project id: 349731058.



(c) Project id: 347967717.



(d) Project id: 344333910.

Figure 5.6 Some Scratch edit pages of projects randomly chosen from the largest cluster.

To manually inspect a sample of the result, I sort the volume and pick up the top 10 most massive clusters after the second iteration. Table 5.2 shows the serial number, volume, and center point id of the top 10 largest clusters. Serial numbers are cluster identifiers which are the same as the horizontal axis in figure 5.4; Volumes are the number of Scratch projects contained in the cluster; Center point ids are ids of the corresponding center of each cluster, which are unique and can be used to recreate its url at the Scratch project website[3]. Afterward, I inspected the project website manually in those large clusters. Figure 5.5 and 5.6 show some specific Scratch edit pages of different projects in the largest cluster (which volume is 421). Figure 5.5 shows the page of the center point project of the cluster, and projects are shown in figure 5.6 are some randomly chosen individuals.

The project in Figure 5.5 has 15 sprites. 4 of which contain no scripts and for other 11 sprites, each one contains a 2-block script. In figure 5.6, the project in figure(a) has six sprites, and each sprite contains a 3 or 4 block script, which is similar to the scripts in the project in figure 5.5. The project in figure (b) has two sprites, and the first sprite contains five 2-block scripts and one 6-block script. Project in figure (c) has three sprites, and they contain four scripts in total. Figure (d) has five sprites, and each sprite contains one long script (longer than ten-block).

[3]If id is 346141665, its Scratch url is: https://scratch.mit.edu/projects/346141665/editor/.

Chapter Six

Discussion

6.1 Discussion on Experiments

I initially tried to calculate similarities between all downloaded Scratch projects (508,794 projects). However, during the actual operation, the similarity calculation was too time and resource consuming to be applied in the entirety of the dataset. Besides the large number of projects, the vector's length is also a reason for time-consuming calculations. During the calculation of script similarity, I use the block type as the vector of each script. Distances between projects are also calculated before the similarity calculation.

Calculating distances in advance has both advantages and disadvantages. The advantage is that the time cost of computing distances during the K-means++ algorithm could be significantly reduced. In general, in the K-means++ algorithm, distances between center points, and other individuals would be calculated multiple times to check which cluster the individual should belong to. As preparing distances as a dataset at present, by searching the data inside, distances between one individual and all other center points could be calculated at once. Thus, each individual's closest center point could be easily found, skipping the step of calculating distance one by one. Simultaneously, this approach is not suitable for all kinds of K-means++ algorithm because the dataset of distances may be a large size, depending on individuals' density. If there are too many close individuals, the data of distances are

too large to be stored in the storage device, and this approach could not be implemented. About the disadvantage of this method, fixed distances limited the algorithm to generate new center points. In this algorithm, center points of clusters are chosen from existing individuals instead of creating new temporary center points. This approach's shortcoming is that the center point in this algorithm may not be the absolute center. Since those center points are not 'tailor-made', they may have some deviation from real centers. For this disadvantage, since the distance dataset is generated, individuals that are not 'connected' with distances would not directly be grouped into one cluster. Thus, the grouping error would not happen, and strongly 'connected' individuals, which have minimal distance between them, would still be categorized into one cluster. The main effect of this method would be weakly 'connected' individuals.

The optimization of the algorithm is still a topic. One possible way is to decrease the volume of elements, which are block types, in each vector. However, different with strings converted from words in general similarity calculations, the importance of different block types is hard to judge; since different types have different functions, their weight in a vector should be the same, and thus they are hard to be omitted. Maybe similar functions could be categorized into the same element, depending on further experiments.

6.2 Limitations of Applying Similarity Calculation

The experimental run of the similarity calculation was affected by several issues. Due to several limitations, the experiment could not be carried out precisely originally planed. The limitation generally comes from time cost, and it is mainly reflected from three points. They are computing performance of the device, period of the experiment, and Scratch users' modifications on their projects.

The first and second points could be considered together. The original volume of the Scratch project dataset is 508,704, which could take several years to implement its similarity

calculation on the device[1] used in this experiment. This time period is calculated by calculating the running time in 100 randomly picked projects as a sample dataset, then multiply the time by the ratio between the full dataset with the sample dataset. The calculation for 100 projects would take dozen minutes, while for 508,704 projects, the time cost increase exponentially to several years. This computing time is far beyond the time limit of the experiment (several months to one year). Therefore, I have to reduce the volume to reduce the time cost. Also, calculating all distances before the similarity calculation is also an approach to reduce computing time cost on the device.

The effect of the third point refers to result inspection after the similarity calculation. The dataset used in this experiment was downloaded from October to December 2019. Unavoidably, the similarity calculation takes a long period for computing. During this period, Scratch users who created projects in the dataset may has modified their codes, which change the structure of scripts. After the similarity calculation, the displayed projects during manual observation maybe not the original programs downloaded previously . Unfortunately, this problem is unavoidable.

6.3 Discussion on Results

Scripts' Similarity Calculation

As can be seen from figure 5.2, after the similarity calculation, most grouped scripts are assigned to several clusters, and other clusters only contain little scripts. Compared with figure 5.1, which results from the first iteration, the overall trend of distribution of scripts has not changed between two iterations. Compared with the first iteration, the enormous cluster volume decreases to almost half in the second iteration. It relatively accommodates more script clusters to become less, but those who keep the volume seem to contain more scripts. Due to the time cost, I only implement two iterations on scripts. However, since

[1]CPU: Dual-core E5 2630V2, RAM: 48G, ROM 480G

there are still some changes in the distribution of individuals, in the case where time permits, more iterations implemented on scripts similarity calculations would lead to a result in higher accuracy.

Scratch Projects' Similarity Calculation

As figure 5.4 shows, Scratch projects' clusters have an apparent difference in volume after the calculation in the second iteration. Most clusters only contain projects at a single-digit level, while several clusters contain hundreds of projects. The size of the largest cluster is 421. Compared with the first iteration shown in figure 5.3, there is no noticeable change after the second iteration. Both figures have a similar distribution of Scratch projects, and both of them have the same size as the top 10 most massive clusters in the same serial numbers. The size of those clusters is shown in table 5.2. According to Figures 5.1 to 5.4, clusters of projects seem more stable across iterations than clusters of scripts. Since clusters of projects are determined by clusters of scripts to a certain extent, a more accurate result of script clusters may lead to a different result. Besides, checking center points' similarity and combining similar points in one cluster may also be feasible because the primitive number of clusters is generated randomly to some extent.

Accuracy of the Result According to inspected Scratch Edit Pages.

Figure 5.5 and 5.6 show Scratch Edit Pages of example projects in the largest Scratch project cluster. Figure 5.5 shows some structure of the center point project in the largest cluster, and figure 5.6 shows four other projects in the same cluster. Comparing figure 5.5 and 5.6 (a), each sprite contains no more than one script for both of them, and each script has a similar structure to each other. Therefore, these two projects appear to be similar.

Comparing Figure 5.5 and 5.6 (b), although they have a similar structure in some 2-block scripts, they are different in the structure of sprites. The former has many sprites consisting of one script, while the latter has only two sprites, which each consists of multiple scripts.

From this comparison, it is indicated that the structure of sprite should also be an essential factor in similarity calculation. However, two projects may have many similar scripts, which are distributed in sprites in a different structure.

Compare figure 5.5 and 5.6 (c), they have little similarity between scripts. However, they are still be categorized into the same cluster. One possible reason may be the defect of the distance calculation algorithm between Scratch projects. Since the project in figure 5.6 (c) only has four scripts in total, when calculating the distance, this project only affects less than 4 in the distance in terms of value, which means that this kind of projects has more chance to be 'close' with many other projects. To avoid this kind of incorrect distance, a potentially solution is to only calculate distance between projects which have intersection clusters between their scripts.

Compare figure 5.5 and 5.6 (d), similar problem with comparing figure 5.5 and 5.6 (c) happens between them. Project in figure 5.6 (d) also has a small number of scripts in total, and this may be the reason for categorizing this project, which seldom has similarity with the center point, into the cluster. Another problem in this comparison comes from the omitted part of the algorithm, removing ten or longer scripts from distance calculation. If the available computing resources were enough to compute distances of larger scripts, it may be shown with better accuracy.

Chapter Seven

Conclusion and Future Work

7.1 Conclusion

In this experiment, the similarity calculation of Scratch projects is implemented in two steps. The first step is scraping and parsing JSON files, which are source files of Scratch projects, from the Scratch Website. The scraping period was from October 22nd to December 24th in 2019, and 508,704 projects (using Scratch version 3) were downloaded. Information on the structure of blocks is parsed from JSON files and stored in a MySQL dataset. The second step is implementing similarity calculation on Scratch projects using the generated MySQL dataset. This step is also divided into similarity calculation on scripts and similarity calculation on Scratch projects. Both parts use the Simhash algorithm method to generate vector (fingerprint) of individuals and use the method of K-means++ algorithm to calculate similarities. When calculating the similarity of scripts, the type of blocks are used as an index of the vector. When the calculating similarity of projects, clusters of scripts, which is the result of scripts' similarity calculations, is used as the vector's index.

About the result of this experiment, although Scratch projects are categorized to a certain extent, there are still many tasks that need to be solved. Firstly, the speed of computing is not fast enough. Looking for methods to accelerate computing speed is a possible approach, and another upgrading point would be using a better device for computing, which may not

be controllable currently. Secondly, the algorithm itself needs to be modified in order to improve the accuracy of the result.

7.2 Future Work

For future work, there are several points to improve and develop the similarity calculation on Scratch projects. There are two main optimization points: optimizing the current algorithm and adding a new index for the similarity calculation.

Optimizing the current algorithm

The first point would be calculating difference between scripts of all lengths. In the current algorithm, long scripts (10 or longer) are removed from the calculation because of the time cost. If an infinite computing environment or if computing speed is optimized, implementing the algorithm to all scripts would result in better accuracy. Similarly, implementing the algorithm to the full dataset instead of a small part would produce a more valuable result.

The second point would be adjusting the standard of determination in similarity during the distance calculation process. Currently, 50% is the judgment criterion of similarity between individuals. However, for scripts of different lengths, 50% refers to different block numbers, and more blocks would lead to longer distance easier, which may not be a suitable value for all individuals. Therefore, using a variable standard according to the length of individuals may lead to a better result.

The third point is combining similar block types in vectors to reduce computing time cost. In this experiment, 278 block types are used for the index of vectors, which lead to exponential growth in computing time. If block types in similar functions are combined, the vectors' length would also shorten, which may lead to much faster calculation speed.

Adding new index for the similarity calculation

Since the current algorithm still has many weak points that can not compute a higher quality result, adding some new approaches may be the right way for improvement.

The first point is adding a part of sprites' similarity calculation. According to this experiment's result, the sprite is also an essential element to decide the similarity, which was ignored in the current algorithm. Since sprites are carriers of scripts, the similarity calculation could be inserted between the calculation after scripts before projects, using scripts as the vector. Then the result of sprites' clusters could be used as project vectors.

The second point is using variables in Scratch projects as additional vectors. In the current algorithm, variables in Scratch projects are not used for similarity calculation. While they also have potential to be the measure in similarity calculation. Therefore, while ensuring that the complexity will not increase too much, adding them as supporting vectors may also be a potential improvement.

The third point is doing some processing on invalid scripts, which are dead codes in Scratch projects. In many projects, there are some invalid scripts, which are single blocks or do not have hat blocks[1]. According to the different purposes of the similarity calculations, they can be ignored or focused on. For example, if the calculation focuses on the similarity in functions, those invalid scripts would be 'trash' and need to be removed from the calculation. However, if the calculation wants to roll out projects' similarity in structures or user habits, they may become an essential factor for the judgment.

[1] https://en.scratch-wiki.info/wiki/Hat_Block

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. "How Kids Code and How We Know: An Exploratory Study on the Scratch Repository". In: *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ICER '16. Melbourne, VIC, Australia: Association for Computing Machinery, 2016, pp. 53–61. ISBN: 9781450344494. DOI: 10.1145/2960310.2960325. URL: https://doi.org/10.1145/2960310.2960325.

[2] K. Amanullah and T. Bell. "Analysis of Progression of Scratch Users based on their Use of Elementary Patterns". In: *2019 14th International Conference on Computer Science Education (ICCSE)*. 2019, pp. 573–578. DOI: 10.1109/ICCSE.2019.8845495.

[3] Bryce Boe et al. "Hairball: Lint-Inspired Static Analysis of Scratch Projects". In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 215–220. ISBN: 9781450318686. DOI: 10.1145/2445196.2445265. URL: https://doi.org/10.1145/2445196.2445265.

[4] Moses S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 380–388. ISBN: 1581134959. DOI: 10.1145/509907.509965. URL: https://doi.org/10.1145/509907.509965.

[5] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. "Traffic Classification Using Clustering Algorithms". In: *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*. MineNet '06. Pisa, Italy: Association for Computing Machinery, 2006, pp. 281–286. ISBN: 159593569X. DOI: 10.1145/1162678.1162679. URL: https://doi.org/10.1145/1162678.1162679.

[6] Zhangjie Fu et al. "Privacy-preserving smart similarity search based on simhash over encrypted data in cloud computing". In: *Journal of Internet Technology* 16.3 (2015), pp. 453–460.

[7] Vasileios Hatzivassiloglou, Judith L Klavans, and Eleazar Eskin. "Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning". In: *1999 Joint SIGDAT conference on empirical methods in natural language processing and very large corpora*. 1999.

[8] Anil K Jain. "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8 (2010), pp. 651–666.

[9] John Maloney et al. "The Scratch Programming Language and Environment". In: *ACM Trans. Comput. Educ.* 10.4 (Nov. 2010). DOI: 10.1145/1868358.1868363. URL: https://doi.org/10.1145/1868358.1868363.

[10] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. "Habits of Programming in Scratch". In: *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*. ITiCSE '11. Darmstadt, Germany: Association for Computing Machinery, 2011, pp. 168–172. ISBN: 9781450306973. DOI: 10.1145/1999747.1999796. URL: https://doi.org/10.1145/1999747.1999796.

[11] J. Moreno and G. Robles. "Automatic detection of bad programming habits in scratch: A preliminary study". In: *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. 2014, pp. 1–4. DOI: 10.1109/FIE.2014.7044055.

[12] Jesús Moreno-León, Gregorio Robles, et al. "Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills". In: *Scratch conference*. 2015, pp. 12–15.

[13] Caitlin Sadowski and Greg Levin. "Simhash: Hash-based similarity detection". In: (2007).

[14] D. Sculley. "Web-Scale k-Means Clustering". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178. ISBN: 9781605587998. DOI: 10.1145/1772690.1772862. URL: https://doi.org/10.1145/1772690.1772862.

[15] Kiri Wagstaff et al. "Constrained k-means clustering with background knowledge". In: *Icml*. Vol. 1. 2001, pp. 577–584.