# Grading Student Programs – A Structural Similarity Approach

The Open University of Hong Kong
Bachelor of Computing with Honors in Internet Technology (BCOMPHIT)
Computer Project Final Report

Supervisor: Dr. Oliver AU
Name: Alex, CHEN Yong Da
OUID: 11102974

## Abstract

Assessing numerous student assignments is a time consuming task for course instructors. In introductory programming courses, students make mistakes with patterns. Plus, several education initiatives such as EdX, Coursera, and Udacity are racing to provide online courses on introductory programming, the huge size of student assignments make manual assessment be an impossible mission. Efficiency of grading assignments and a short feedback time is significant in these massive open online courses (MOOC). Automatic grading can mitigate effort of teachers, and it can enhance the learning experience in programming courses by providing short feedback time.

The background review is about the application of functional testing in grading student programs. This traditional method will execute student programs and check their actual outcome, then compare with the expected output to evaluate the correctness of programs. But this method can't show the structure of program source code.

My system uses LLVM to capture the control flow graph of the program. According to the similarity between graphs, we can group similar programs together. After grouping, teachers can access assignments group by group instead of one by one.

## Declaration

I, Alex, CHEN Yong Da, student ID 11102974 certify that the work is original and I have utilized guidance of my supervisor in completing this project, and that the content which is not my own has been attributed and referenced properly. There should be no copyrighted content without permission to use. There should be no confidential data. I have retained a copy of this submission (where written or electronic)

Signature

Date

## Acknowledgement

I want to express my deepest thanks to my project supervisor Prof. Oliver Au, he gave me great inspiration and guidance during the period of project development. Oliver taught me how to do research, and how to express my ideas. His professional knowledge and patience helped me overcome many difficulties.

I would like to thank Prof. Andrew Lui for commenting my views and offering anonymous student source code files. It was a great help for the project testing and adjustment. Andrew is enthusiastic in education and very helpful to students.

I also like to thank my friend and schoolmate Calvin Poon sincerely for his practical advices and proof reading to my report.

Last but not least, I would like to thank my father, mother and sister. They brought me to this world and accomplished with my growth.

## Table of Contents

## List of Tables

## List of Figures

# Chapter 1. Introduction

## 1.1 Overview

The proposed method is based on program structural similarity to grade program assignments of a student. In chapter 1, I discussed the needs in automatic program grading and what this project will achieve. The related works and methods related to automatic program assignment grading and the applications of these approaches were discussed in chapter 2. In chapter 3, I will introduce LLVM compiler toolchain and other framework that helps to construct an abstract representation, abstract representation is more convenient to do comparing. Also, I need a prototype system to prove the concept and a proper evaluation plan to evaluate the performance of this method. The prototype system execution results had been descripted in chapter 4. Chapter 5 is the conclusion of this report.

## 1.2 Project Aim

Assessing numerous student assignments is a time consuming task for course instructors. In introductory programming courses, students make mistakes with patterns. Plus, several education initiatives such as EdX, Coursera, and Udacity are racing to provide online courses on introductory programming, the huge size of student assignments make manual assessment be an impossible mission. Efficiency of grading assignments and a short feedback time is significant in these massive open online courses (MOOC) [1]. Automatic grading can mitigate effort of teachers, and it can enhance the learning experience in programming courses by providing short feedback time.

## 1.3 Project Objectives

The previous report was focused on source program styling and software metrics. These kind of metrics usually performs on large or medium project for measuring software quality. Software metrics in small program such as student exercises may not show any meaningful feedback. Therefore, I changed approach to handle small programs.

A Command Line User Interface allows user to interact with the system;
A Graph Representation Component base of LLVM Pass. It takes a source file as input, and outputs a control flow graph in JSON format;
A Graph Similarity Component takes two control flow graph in JSON format and output a boolean value to indicate that they are similar.

The proposed system should be able to group similar program together, then the teacher can assess student program group by group.

## 1.5 Impact and Value of this Project

Students will make similar mistakes in introductory programming courses [1]. Automatic grading can save effort form assessing program assignment, so instructors can focus on other things.

Students can gain feedback faster than human assessment if the automatic grading system applied. It will improve the learning experience for someone new in programming.
We borrow ideas from plagiarism detection and malware classification to do program comparison [2] [3] [4].

# Chapter 2. Literature Review

## 2.1 Functional Testing Approach

Functional testing is one of black-box testing techniques. One test case of functional test includes a set of input data, a series of steps and the expected outcome. Testers perform the predefined steps and input data to the target system, then compare the actual result and the expected outcome.

If the actual result is equivalent to the expected, this test case counts as a success, otherwise it is a fail. This intuitional idea has been widely using in program grading and online judging for over ten years [5] [6].

### PSGE [5]

PSGE is Unix-based Program Submission and Grading Environment. PSGE applies functional testing approach to automatic grading student programs.

First, the instructor needs to design the assignment specification, it standardizes the structure of student assignment program's input and output. After collection of the student programs, PSGE compiles all programs, and initiates programs execution. Interactive data come from input data file, and outputs are captured in result files. The PSGE program checker finds differences between actual program outputs and expected program outputs, and then reduces program score accordingly.

### Characteristic of Testing Approach

This is a straightforward way to judge a program's functionality, the system logic implementation of this approach should not be a big challenge. These kind of systems can ensure the program correctness efficiently.

But we can't know about the implementation detail of student program with this kind of checking. For example, the student needs to write a procedure that can sort a list in ascending order. If we only concern about input and output, we may not know about the internal sorting algorithm, statement structure and coding practices. If we know more about the student assignment, we could provide more specific and helpful feedbacks.

## 2.2 Code Style Grading

We focus on student program as a structured text file instead of a compiled binary file. By looking at the program source, we can learn programming style of developer. By using program parser techniques, we can recognize statement structures, even algorithms used in the programs [7] [8] [9].

### Formal Language and Program Parser

Unlike human language, computer programming language is a formal language, which has a strict syntax and unambiguous semantics. Because of the strict syntax, we can use automata theory to transform the program source from paint text to structured representation, syntax tree for example. Program parser aims to find syntax violations in the program source. A customized parser can be used to check for the coding style.

### Checkstyle [9]

Checkstyle is a development tool to automate the process of checking Java code style [8] [9], to see if there are any violations of Java coding standard.

In parsing phase of compiler front end, we could use a context free grammar to define a language syntax [7]. If the token sequence can't match with any grammar, we consider that is a syntax error. Checkstyle uses the same idea to do validation on the customized rules, it's based on ANTLR, a parser-generator written in Java.

Based on checkstyle, we can design a set of styling rules to students. The style rule violation will cause a score deduction. That could be a possible grading criteria for a student program. But this kind of tools is usually integrated with IDE or text editor, these tools provide some instant suggestions to the developer. Students will only need a single button click to fix most of styling mistakes. So we might not need to spend too much effort into this approach.

## 2.3 Code Block Structure Comparison

We focus on the abstract representation of program procedures. According to the similarity of abstract representation between programs, we can divide programs into groups [10] [11].

### Tree-based Representation

Abstract Syntax Tree (AST) is the outcome of program parser. It keeps the most detail of a program, and it is easy for manipulation. Parser removes the unnecessary symbols and whitespaces, and it uses the tree data structure to maintain the program semantics. For examples, the order of child nodes can represent the order of statements, the hierarchy of the tree can represent the expression precedence.

### Graph-based Representation

Control Flow Graph (CFG) is an abstraction for program control flow. CFG is a directed graph, each node of the graph represents a basic block and each edge represents the control flow between basic blocks. To build a CFG we need to extract the basic blocks first, then we add edges to show the control flow between these basic blocks [12].
A Basic Block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end. [12]

### Measuring Similarity

Techniques of measuring program similarity are widely used in the software plagiarism detection, clone detection and malware classification [2] [3] [4].

Text based or token based similarity is used in the software plagiarism detection. However, there are several common attack targets on them [13].

- Verbatim copying, changing comments
- Changing white space and formatting, renaming identifiers
- Reordering code blocks, statements within code blocks
- Changing the order of operators in expressions
- Changing data types and adding redundant statements or variables

To deal with these attacks, we can use structural comparison. It focuses on program structure instead of paint text.

Malware analyst doesn’t have the source code of virus in most cases, so they usually focus on behavior of malware. Detecting code signature is an effective way to identify malware. Due to its efficient and low false positive rate [14]. However, the efficiency of signature extraction are being weakened by the software self-modification techniques and the rapid growth of malware variants [15].

Finding program structure similarity can be used to discover new malware variants [16] [2] [14]. We can borrow this idea to classify student programs.

# Chapter 3. Methodology

## 3.1 Overview

In order to group student assignment into groups, a prototype system needs to be developed. This system should be able to accept a bunch of program source files and produce the group information. Teachers will only need to assess the assignment group by group.

## 3.2 Framework and Theory

The prototype system targets on C programs written by year one students. The small size programs need a reasonable memory usage and acceptable time consuming, so it should be easier to implement the prototype system.

### Capturing Control Flow

Clang and LLVM is used to extract the control flow of the source file. LLVM is a compiler infrastructure, which provides a modern intermediate code optimizer. A set of modular compiler components built around a well specified code representation known as the LLVM intermediate representation ("LLVM IR") [17] [18].
Clang is a compiler front-end build on LLVM and it is targeting on C and C++. LLVM and Clang adopt a modular based design, developer friendly open source license (BSD-like) and outstanding performance. So LLVM is widely used in industry product, open source project and academic research [19] [17].

LLVM and Clang on an extendable system design. Developers can extend and enhance the original compilation process by adding new customized modules.
The LLVM Pass mechanism allows developers to extend the LLVM source base, then put their functions into the new subclass. This new subclass will be compiled into a static library and registered into the compilation process.

The following picture shows the role of the LLVM Pass in the compilation process. It is a pipes and filters design pattern, and the data stream is the LLVM IR.

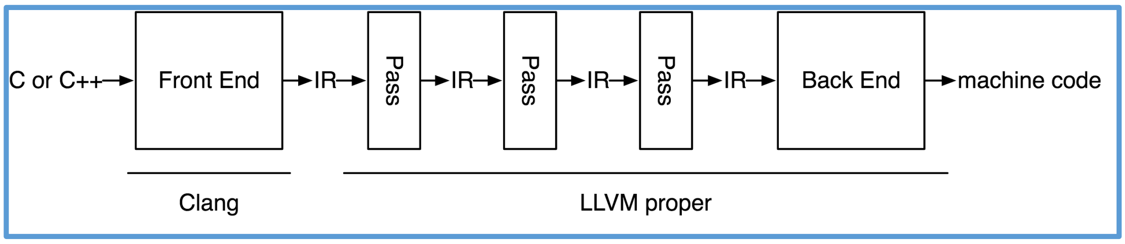

*Figure 1 role of LLVM Pass [18]*

RapidJSON is a C++ JSON library. It has good documentation and clear interface. It helps the transformation from the LLVM IR stream to the customized JSON structure.

### Compare Two Control Flow

A control flow graph is a sparse directed graph. Finding the size of the maximum common subgraph of two control flow graphs could be one way to measure how similar they are [20].

NetworkX is a Python graph library, which provides rich graph operations [21]. The subgraph function and isomorphism related functions will help in finding common subgraph.

## 3.3 System Design and Development

### Capture Control Flow Graph

This component will be based on LLVM Pass mechanism. It takes a source code file as input and output a control flow graph.

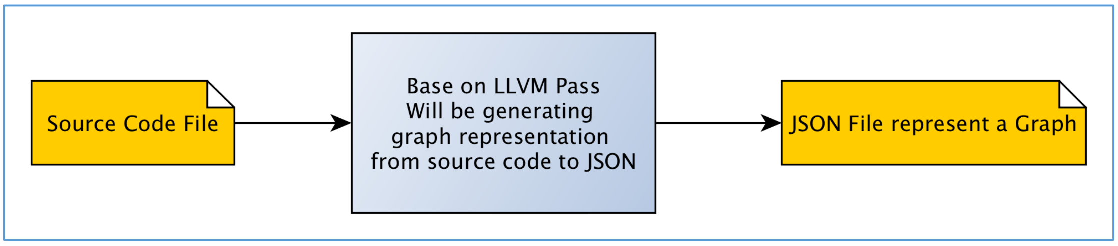
Source Code File
Base on LLVM Pass Will be generating graph representation from source code to JSON
JSON File represent a Graph

*Figure 2 Component of Graph Generation*

The LLVM IR contains all basic block and program branches information. I use LLVM Pass mechanism to traverse all basic blocks, then save the information into JSON file.

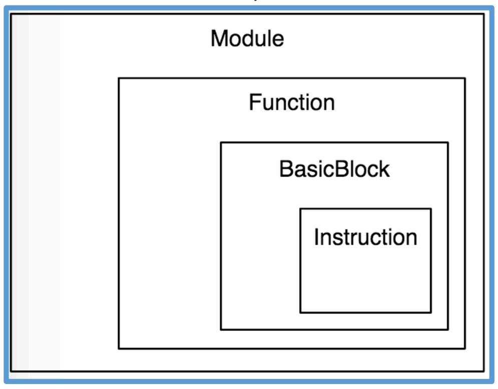

*Figure 3 Intermediate code organization [18]*

### Graph Similarity Component

A python program will be developed to compare two control flow graphs. The python program accepts two control flow JSON file and transform them to NetworkX graph format.

Setting the number of nodes of a control flow graph be its size. The size of the common subgraph of two graphs needs to be calculated. This calculated size uses to distinguish two control flow graphs are similar or not. For example, 80% similarity is configured, two graphs with size 10 are similar, if a size 8 common subgraph is found.

Pseudocode for showing the above logic:

```
A Graph Size = 10
B Graph Size = 10

Expected Size = (A Graph Size + A Graph Size) / 2 * Similarity Percentage

IF Common Subgraph with Expected Size is Found:
   A and B are considered similar
ELSE:
   A and B are not considered similar
```

## Grouping Control Flow Graph

There is a straightforward algorithm to show the grouping steps. First, this module traverses all student programs, if there is a student program similar with one of the known group, then the student program will be put into that category, otherwise a new group contain this program will be created.

Pseudocode for showing the grouping logic:

```
known_groups = []

for program in all_programs
{
   for group in known_groups
   {
      if program is similar with group[0]
      {
         put program into group
         continue to check next program
      }
   }

   not similar to any one of known groups, create a new group, then append the new group to known_groups
}
```

# Chapter 4. Results

## 4.1 Prototype System

Based on LLVM Pass mechanism and NetworkX python graph library, a prototype system was developed to perform the evaluation.

For the first part of the prototype system, a C++ program extends the LLVM source base and follow the LLVM Pass interface. The compiled static library was successfully injected into the target program compilation process of Clang, and this library was successfully traverse the control flow of procedures. Also the JSON library worked as expected, it outputs the procedure control flow in the customized JSON structure.

The second part is a Python program, it reads in two JSON file and transform to NetworkX graph format. It was successful in checking graph isomorphism and finding common subgraph.

The last part also is a Python program. According to the result of the second part, this part can distinguish control flow graphs and put them into different groups.

## 4.2 Observation Results

### Data Preparation

The system is targeted at small size programs. 108 small C program source files from year one students were prepared.

### Environment Setup

First, the program source needs to be converted to JSON control flow. The following Clang command can invoke the customized LLVM Pass library to do the conversion.

```
clang -Xclang -load -Xclang "<The LLVM Pass Static Library Path>" "<The Program
Source>"
```

After getting all JSON control flow files, a Python command line interface can be interacted to perform graphs grouping.

```
python test_network_similarity.py /home/vienna/FYP/cfg/g  0.80
```

The above command will retrieve JSON control flow files and turn them into NetworkX graph objects. The parameter 0.80 means the system will consider the two graphs are similar, if a common subgraph with 80 percent size is found.

### Execution Results

The execution results of

```
python test_network_similarity.py /home/vienna/FYP/cfg/g  0.85
```

Table 1 Result of similarity 0.85

| | |
|---|---|
| Number of nodes of Group 1 | 21 |
| Number of nodes of Group 2 | 65 |
| Number of nodes of Group 3 | 2 |
| Number of nodes are grouped with others | **88** |
| Number of groups include single node | **20** |
| Total number of nodes | **108** |

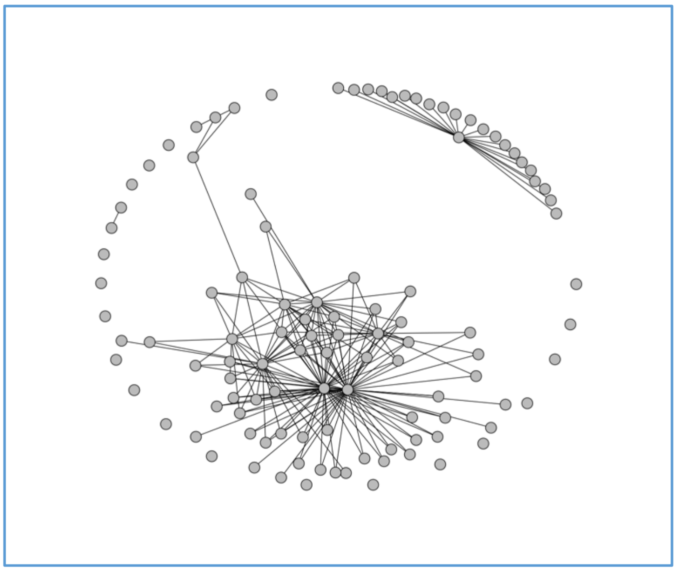
Figure 4 Result Picture of similarity 0.85

Then we try another similarity value 0.95.

```
python test_network_similarity.py /home/vienna/FYP/cfg/g  0.95
```

Table 2 Result of similarity 0.95

| | |
|---|---|
| Number of nodes of Group 1 | 21 |
| Number of nodes of Group 2 | 26 |
| Number of nodes of Group 3 | 2 |
| Number of nodes are grouped with others | **49** |
| Number of groups include single node | **59** |
| Total number of nodes | **108** |

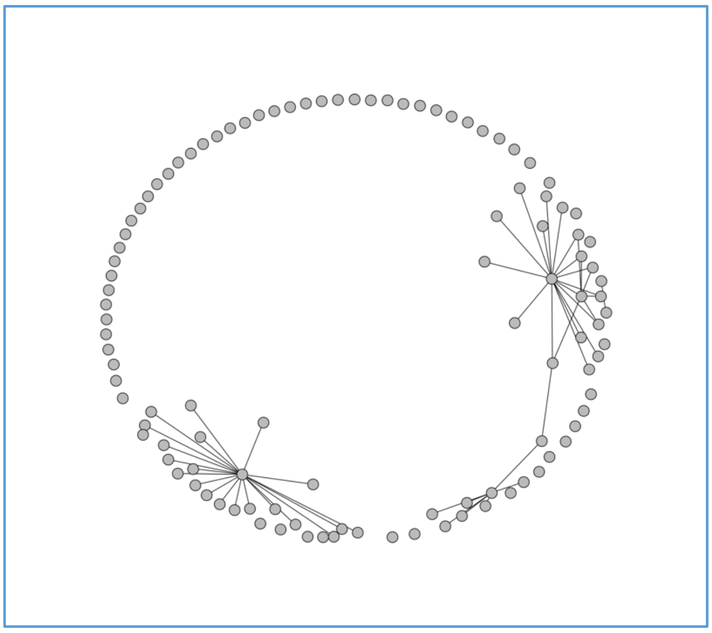
*Figure 5 Result Picture of similarity 0.95*

Depends on the similarity parameter value, the running time of control flow grouping can take over ten minutes (Ubuntu virtual machine, Intel i5 2 cores with 2 GB RAM). Because there is no human interaction needed in the running period, ten minutes could be acceptable.

In the result of similarity parameter 0.85, there are three groups contain 88 nodes, the rest of 20 nodes are not similar to others. Theoretically, teacher needs 3 + 20 = 23 times of assessments instead of 108 times one by one assessment.

In the result of similarity parameter 0.95, there are three groups contain 49 nodes, the rest of 59 nodes are not similar to others. Then teacher needs 3 + 59 = 62 times to assess the system result. When the similarity requirement increased, the number of groups include only single node will increase too.

# Chapter 5. Conclusion

In this report, we proposed a transformation and grouping system. The proposed system transforms the program source code to control flow graph, then compares control flow graphs and distinguish them into groups.

The proposed system provided another way for teachers to grade student program assignments. Teachers will not be necessary to open every program source file to grade the program thinking and algorithms, just need to pick one from a group of assignments.

The research and technology in this report also related to program analysis, program clone detection and program plagiarism detection.

The existing system support only small C program, the next steps includes: development of more language source to abstract representation modules, support more language assignment grading; research on new program similarity measurement method to support bigger program and support inter-procedure program comparison.

## References

[1] R. Singh, S. Gulwani and A. S. Lezama, "Automated Feedback Generation for Introductory Programming Assignments," *PLDI '13 Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation,* vol. 48, no. 6, pp. 15-26, June 2013.

[2] Z. Zhao, "A virus detection scheme based on features of Control Flow Graph," *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC),* pp. 943 - 947, August 2011.

[3] B. Beth, "A Comparison of Similarity Techniques for Detecting Source Code Plagiarism," 12 May 2014.

[4] S. Cesare and Y. Xiang, Software Similarity and Classification, Springer Briefs in Computer Science Springer, 2012.

[5] E. L. Jones, "Grading Student Programs - A Software Testing Approach," *Journal of Computing Sciences in Colleges,* vol. 16, no. 2, pp. 185-192 .

[6] A. Kosowski, M. Małafiejski and T. Noiński, "Application of an online judge & contester system in academic tuition," *ICWL'07 Proceedings of the 6th international conference on Advances in web based learning,* pp. 343-354, 2007.

[7] A. Aho, M. Lam, R. Sethi and J. D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison-Wesley Longman Publishing Co., Inc., 2006.

[8] K. Ala-Mutka, T. Uimonen and H.-M. Jarvinen, "Supporting Students in C++ Programming Courses with Automatic Program Style Assessment," *Journal of Information Technology Education,* vol. 3, pp. 245-262, 2004.

[9] "checkstyle," [Online]. Available: http://checkstyle.sourceforge.net/. [Accessed 30 12 2015].

[10] M. V. Janičić, M. Nikolić, D. Tošić and V. Kuncak, "Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments," *Information and Software Technology,* vol. 55, no. 6, pp. 1004-1016, June 2013.

[11] T. Wang, X. Su, Y. Wang and P. Ma, "Semantic similarity-based grading of student programs," *Information and Software Technology,* vol. 49, no. 2, pp. 99-107, February 2007.

[12] M. J. Harrold, G. Rothermel and A. Orso, "Representation and Analysis of Software," [Online]. Available: http://www.ics.uci.edu/~lopes/teaching/inf212W12/readings/rep-analysis-soft.pdf.

[13] J.-H. Ji, G. Woo and H.-G. Cho, "A Plagiarism Detection Technique for Java Program Using Bytecode Analysis," in *Third 2008 International Conference on Convergence and Hybrid Information Technology*, 2008.

[14] S. Shang , N. Zheng , J. Xu, M. Xu and . H. Zhang, "Detecting malware variants via function-call graph similarity," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, Hangzhou, 2010.

[15] Kaspersky Lab, "Mobile malware growth hits all-time high," 2013. [Online]. Available: http://www.infosecurity-magazine.com/news/mobile-malware-

growth-hits-all-time-high/.

[16] J. Lee, . Jeong and . Lee, "Detecting metamorphic malwares using code graphs," *SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing,* pp. 1970-1977, 2010.

[17] "The LLVM Compiler Infrastructure," [Online]. Available: http://llvm.org/.

[18] A. Sampson, "LLVM for Grad Students," [Online]. Available: http://adriansampson.net/blog/llvm.html. [Accessed 3 August 2015].

[19] "clang: a C language family frontend for LLVM," [Online]. Available: http://clang.llvm.org/.

[20] C. Collberg and J. Nagra, "Software Similarity Analysis," in *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, Addison-Wesley Professional, 2009.

[21] N. d. team, "Overview - NetworkX," 01 May 2016. [Online]. Available: https://networkx.github.io/.