SPL-1 Project Report, 2020

SOURCE CODE PLAGIARISM DETECTION

SE 305: Software Project Lab 1

Submitted by
Saad Sakib Noor
Exam Roll No. : 1108
BSSE Roll No. : 1122
BSSE Session: 2018-19

Supervised by
Md. Saeed Siddik
Designation: Assistant Professor
Institute of Information Technology

SIGNATURE PAGE

PROJECT: Source Code Plagiarism Detection

AUTHOR: Saad Sakib Noor

DATE SUBMITTED: 26th August, 2021

SUPERVISED BY: Md. Saeed Siddik

Assistant Professor

Institute of Information Technology,
University of Dhaka

SUPERVISOR'S APPROVAL: 

Table of Contents

1. Introduction

Source code plagiarism—otherwise known as programming plagiarism—is, simply put, using (aka copying or adapting) another person's source code and claiming it as your own without attribution. Parker and Hamblen define source code plagiarism as "a program which has been produced from another program with a small number of routine transformations." Source code plagiarism can vary from copy-pasting small amounts of program source code to copying large chunks of source code and masking everything with some techniques to disguise copied programs.

Source-code plagiarism detection in programming, concerns the identification of source-code files that contain similar and/or identical source-code fragments. Based on the analysis of the characteristics and defects of the existing program code similarity detection system, a method of source code similarity detection. Code segments usually occur due to replication from one place and then rewriting them into another section of code with or without variations/changes are software cloning and the copied code is called clone.

Various researchers have reported more than 20–59% code replication. The problem with such copied code is that an error detected in the original must be checked in every copy for the same bug. Moreover, the copied code expands the effort to be done when augmenting the code .

However, the code quality analysis (improved quality code), replication identification, virus recognition, facet mining, and bug exposure are the other software engineering tasks which require the mining of semantically or syntactically identical code segments to facilitate clone detection significant for software analysis .

2. Objectives

- Making the project so it can handle many files at once
- Detecting source code plagiarism upto level-3(levels described in part 3.4 of this report)
- Making the program as efficient as possible
- Making the UI as user friendly and easy to use as possible
- Keeping the project source code as organized as possible

3. Background Study

3.1 CODE CLONE

Code clone is a computer programming term for a sequence of source code that occurs more than once. It can be located either within a program or across different programs owned or maintained by the same entity. Cloning unnecessarily increases program size.

It requires the developers to find and update several fragments while changing any module. This section contains the related topics which should be known before introducing code clones.

3.2 CODE FRAGMENT

A Code Fragment (CF) is any sequence of code lines (with or without comments). Clone is detected using comparison between the fragments in a source code. It can be of any type of code, for example function

definition, begin-end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base.

3.3 Source Code Based Plagiarism Detection Techniques

I. Lexical Similarities
II. Parse Tree Similarities
III. Program Dependence Graphs
IV. Metrics

3.3 (I) Lexical Similarities

Converts source code into a stream of lexical tokens from which the compiler extracts meaning: the source. During the lexical analysis phase, the source code undergoes a series of transformations . Some of these transformations, such as the identification of reserved words, identifiers are beneficial for plagiarism detection.

Consider the following two snippets of C Code:

Code-A

int A= 3, B=4;
int C=A+B;
printf(“%d\n”, C);

Code-B
int x=3, y=4; int sum= x+y;

```
printf("%d\n", sum);
```

Lexical Stream of two C codes are:

int_decleration_int_declaration_int_declaration_print_function

int_decleration_int_declaration_int_declaration_print_function

Both the C snippets will have the exact lexical stream. So these two codes are copied.

3.3 (II) Parse Tree Similarities

The parse tree or derivation tree built from the lexical for a program also exhibits structure for a given program A compiler, during the compilation process builds a parse tree which represents the program. The parse tree will have the same structure for both the snippets of code as the lexical streams are the same. An algorithm for detecting plagiarism using this method would first, parse each program. Next, for each pair of parse trees, it attempts to find as many common subtrees as possible. Use this number as a measure of similarity between the two programs.

3.3 (III) Plagiarism detection by similarity analysis using software metrics

Software metrics are:

- Number of function calls
- Number of used or defined local variables
- Number of used or defined non-local variables
- Number of parameters
- Number of statements

- Number of branches
- Number of loops

Each fragment characterized by a set of features measured by metrics Metrics computation requires the parsing of source code to identify interesting fragments. Metrics are simple to calculate and can be compared quickly False positives: two fragments with the same scores on a set of metrics may do entirely different things.

3.3 (IV) Plagiarism Dependence Graphs (PDG)

PDG is a graph representation of the source Code .It is a directed, labeled graph which represents the data and the control dependencies within one procedure .Basic statements like variable declarations, assignments, and procedure calls are represented by vertices in PDGs. It depicts how the data flows between statements and how statements are controlled by other statements. The data and control dependencies between statements are represented by edges between vertices in PDGs .Data and control dependencies are plotted in solid and dashed lines respectively.

3.4 The Types of Copying/Cloning Source Code Are:

- **Type-1:** Identical code fragments except for variations in whitespace, layout and comments. Figure 1 shows this type of code clone.

Original Code :

```
void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
                sum+=i;
                prod=prod*i;
                foo(sum,prod);
        }
}
```

Copied Code :

```
  void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
                sum+=i;
                prod=prod*i;
                foo(sum,prod);
        }
}
```

- **Type-2:** Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments. Figure 2 shows this type of code clone.

Original Code :

```
void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
                sum+=i;
                prod=prod*i;
                foo(sum , prod);
        }
}
```

Copied Code :

```
void sumProd(int n)
{
        float s=0.0;
        float p=1.0;
```

```
        for(int j=0 ; j<=n ; j++)
        {
                sum+=j;
                prod=prod*j;
                foo(s , p);
        }
}
```

- **Type-3:** Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments. Fig 3 shows this type of code clone.

Original Code :

```
void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
                sum+=i;
                prod=prod*i;
                foo(sum ,prod);
```

```
        }
}
```

Copied Code :

```
void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
            sum+=i;
            prod=prod*i;
            foo(sum );
        }
}
```

- **Type-4:** Two or more code fragments that perform the same computation but are implemented by different syntactic variants. Fig 4 shows this type of code clone.

Original Code:

```
void sumProd(int n)
{
```

```
        float sum=0.0;
        float prod=1.0;
        for(int i=0 ; i<=n ; i++)
        {
                sum+=i;
                prod=prod*i;
                foo(sum ,prod);
        }
}
```

Copied Code :

```
void sumProd(int n)
{
        float sum=0.0;
        float prod=1.0;
        int i=0;
        while(i<=n)
        {
                sum+=i;
                prod=prod*i;
                foo(sum,prod);
                i++;
```

```
            }

}
```

4. Scope

Source code plagiarism can be detected using various algorithms which are very efficient. With better algorithms, it is possible to even detect code clones of level-4, but in this project I have only worked upto level-3. I have mainly used lexicographical similarities and edit distance to detect clones. I have also tried using abstract syntax tree to detect source code plagiarism.

5. Challenges

- Working with a whole folder filled with many C codes and handling them was difficult.
- It was challenging to manage all the classes and keep the code clean and understandable as I haven't worked on any OOP project as big as this one.
- Deleting all the comments and unnecessary spaces off of a code
- Saving the result table in csv format
- Implementing JavaFX in different parts of the code
- Implementing algorithms such as Edit Distance in java.

6. Project Description

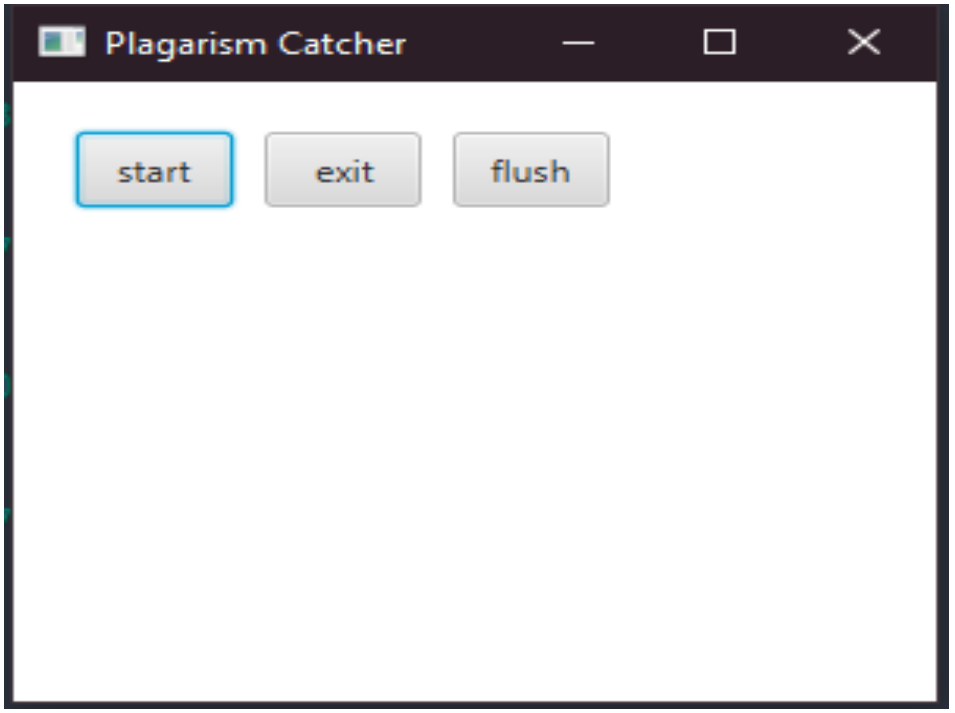
In this project, I apply some code clone detection techniques to detect the clone between multiple C source codes. I apply lexical

analysis, Software Metrics to detect plagiarism between given C source codes. By applying these techniques, it gives a resulting source code match between all the codes given. By seeing the results we can easily detect type-1 and type-2 clones. In some cases, type-3 clones can also be detected. All the results are stored in a csv file so the results would be easy to interpret.

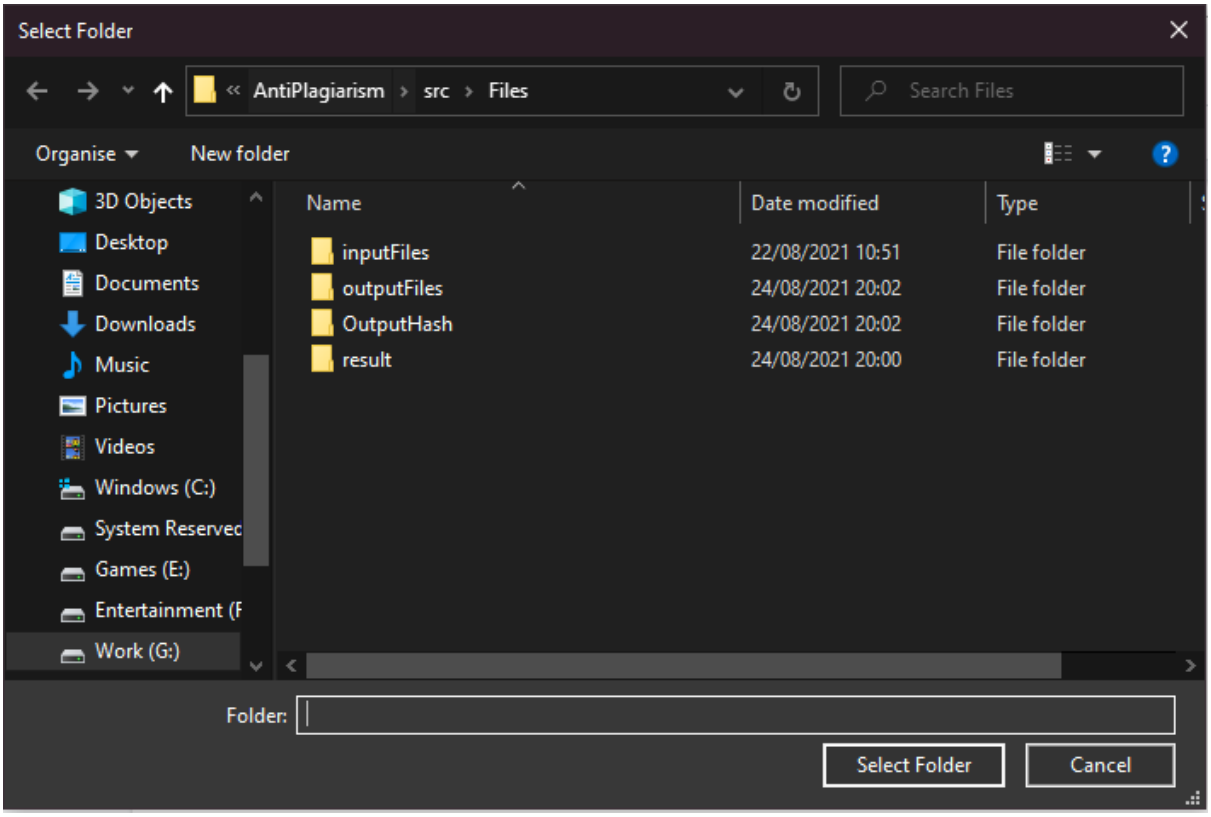
7. User Manual

All the C source codes that are needed to be checked have to be in the same folder. There must not be any unnecessary files in that folder.

After running the program, we'll be met with a menu which lets us choose from a few options.



By clicking the "Start" button, the user will have the option to select the folder on which the C source files to be tested are located.

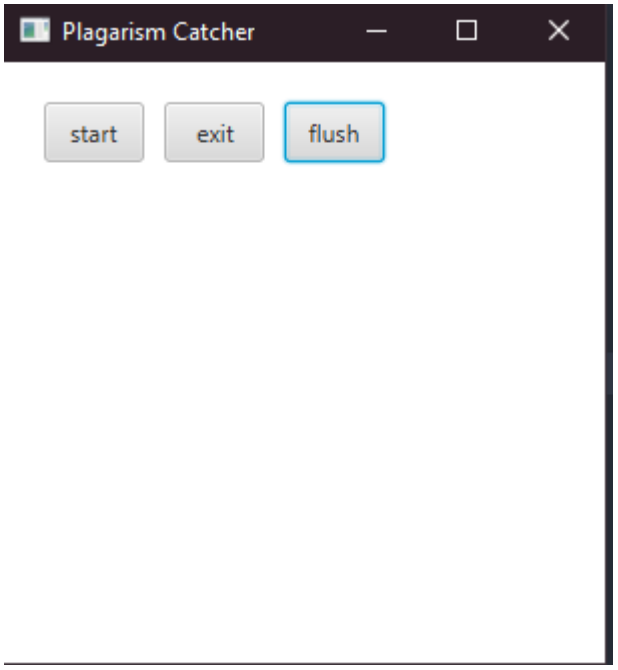


After the user selects the correct folder, The user will be given a choice to select a name for the result file, so if the user wants to save the result as “1stTermResult”, the program will save it as “1stTermResult.csv” in the “Files/Result” directory. The user can open the csv file to see the results directly.

file	003.c	004.c	001.c	002.c
003.c	0.0%	72.5312%	80.7215%	70.9713%
004.c	72.5312%	0.0%	76.5166%	83.5735%
001.c	80.7215%	76.5166%	0.0%	75.8357%
002.c	70.9713%	83.5735%	75.8357%	0.0%

The resulting csv file would look like this. Here, we can see the match between 2 files on the table. For example, the match between “003.c” and “004.c” is almost 72.5%.

Users can also flush, by pressing the “flush” button, the program will automatically delete all the files created by the program previously, so that a new folder can be processed freshly.



NB: this is a mandatory process if the user wants to work with a different folder later on.

8. Conclusion

The code-clone detection is an emerging issue in the software ecosystem which degrades the software’s comprehensibility as well as maintainability. Therefore, its analysis and detection is necessary for improving the quality, structure and design of the software system. This evaluations are not only intended for experts in clone detection, but also intended for potential new users and builders of clone detection-based tools and applications. It is hoped that it may also assist in identifying remaining open research questions, avenues for future research, and interesting combinations of techniques.

9. Appendix

I apply these plagiarism detection techniques to know about these techniques and code analysis . My Supervisor also suggested that I do this project.

10. Reference

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, Comparison and Evaluation of Clone Detection Tools, Transactions on Software Engineering, 2007, pp. 33(9):577-591.

[2] M. Gabel, L. Jiang and Z. Su, Scalable Detection of Semantic Clones, in: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 321-330.

[3] B. Baker, A Program for Identifying Duplicated Code, in: Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, 1992, Vol. 24:4957, 24:49-57.

[4] T. Kamiya, S. Kusumoto and K. Inoue, CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, IEEE Transactions on Software Engineering, 2002, 28(7):654-670.

[5] V. Wahler, D. Seipel, J. Gudenberg and G. Fischer, Clone Detection in Source Code by Frequent Itemset Techniques, in: Proceedings of the 4th

IEEE International Workshop Source Code Analysis and Manipulation, SCAM 2004, pp. 128-135.

[6] W. Yang, Identifying Syntactic Differences between Two Programs, Software Practice and Experience, 1991, 21(7):739

Project Git link: https://github.com/SaadDroid/AntiPlagiarism