LopezMOSS: An Implementation of Measure of Software Similarity Algorithms

\*Note: Sub-titles are not captured in Xplore and should not be used

Emil John Lopez  
De La Salle UniversityBataan, Philippines  
emil\_lopez@dlsu.edu.ph

*Abstract*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract*. (*Abstract*)

Keywords—component, formatting, style, styling, insert (key words)

# Introduction

Plagiarism is an age-old problem in intellectual property law. However, the speed of data transfer and the rise of the Web has made it easier to do the act and has made detection much harder. Furthermore, adjuging someone of commiting the act turns out to be very complicated and can lead the overseer and lawyers in a legal grey area, where it is unsure whether the act violates the law. It is important to examine its definition first to attempt to solve the problem. According to a paper, actions that fall under plagiarism include: (1) turning in someone else’s work, (2) copying someone’s idea without giving credit, (3) not putting quotation marks, and (4) changing words only without changing the structure of the sentence [1].

In educational institutions, software assignments are usually subject to this form of malpractice and so it threatens the integrity of the educational process. In particular, it is subject to the first and second definitions stated above. However, given the number of students each professor must handle, and the large amount of effort required to perform ad hoc comparisons between them, there is a need for a tool that reliably detects plagiarism and can look past obsfucation, reordering, refactoring and other methods of deception [2].

The first significant solution to this problem was introduced in 1994 by an associated professor in UC Berkeley. It used *winnowing,* a local document fingerprinting algorithm that grouped grammatical tokens in groups of some number of tokens, hashed them to minimize storage space, and counted the frequency of each group using their hash value. This is directly derived from other techniques such as Karp-Rabin String Matching [3].

Fingerprinting algorithms calculate numbers (which we call fingerprints) to help identify a document. Conflicts within these numbers usually indicate that some part of a document (or segment of code in this case) is similar to another document. Better fingerprinting algorithms have since been derived from this method, but it remains that the foundation of most of them is either Karp-Rabin String Matching or *n*-gram fingerprinting [4].

This paper will present an implementation of a variation of these key fingerprinting algorithms. In particular, it showcases a version of *n-*gram fingerprinting. The implementation will also have a *graphical user interface* (GUI) that will allow its user to simply pick directories which correspond to projects.

# RELATED WORKS

There have been many attempts at solving the code plagiarism problem since its conception in 1994. The following contains a list of all such services, including some implementation details in their design.

Most of these solutions have been found to fall under one of these three (3) categories: (1) text-vased, which uses plain text as its input, (2) token-based, which uses tokens, fundamental grammatrical units, and (3) model-based, which creates different models for the source code. Qualitatively, it might be accurate to say that the last two algorithms, which uses organized data structures, will be slower for larger dataset, which might prove problematic. However, for accuracy and efficiency, the third is most likely the best for large scale anti-plagiarism detection [5].

One such solution uses the concept of a *software birthmark,* a distinct characterestic of machine code. It involves the use of a type of birthmark called Dynamic Key Instruction Sequence, which can be directly extracted from compiled machine code. This means ordinary source code obsfucation does little to nothing to help evade detection because this will rely instead on a graph data structure built from the machine code of the program. At a lower level, it can even detect compiler optimizations and other tooling that might differentiate two similar pieces of code after compilation. [6].

Another solution, which takes the concept of n-gram fingerprinting and other token-based algorithms one step further is the Weight Abstract Syntax Tree Kernel method, which creates a syntax tree from the source code (like a compiler would) and the tree of two pieces of code to calculate the similarity between the. This approach works better than more popular implementations such as Sim and JPlag because it takes the actual grammatical structure of the tokens into account instead of simply just comparing them as independent entities. [7]

JPlag is a web service that attempted to find similarities amongst a group of programs written in Java, Scheme, C, or C++. It processes code in two phases: (1) the programs are parsed and are effective turned into tokenized strings, then (2) it uses a method called **Greedy String Tiling**, where it tries to match substrings of each token strings to each other as much as possible [8].

Another such tool is *Parikshak* [9], which is similar to Jplag in that it uses the **Running-Kark-Rabin Greedy String Tiling** algorithm. It breaks down the program into tokens, then proceeds to group each token of size n, aptly called *n-gram representation.* It then converts these groups into corresponding numbers (each digit of which represents the type of the token). To generate a similarity score from such comparisons, it uses the **Jaccard Similarity Coefficient**, the quotient of the total number of fragments that appear in both programs and the number of fragments different between the two programs.

# METHODOLOGY

## Header

# Using the Template

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

# Acknowledgment

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

# References

|  |  |
| --- | --- |
| [1] | H. Maurer, F. Kappe and B. Zaka, "Plagiarism-A Survey," *Journal of Universal Computer Science,* vol. 12, no. 8, 2006. |
| [2] | K. Bowyer and L. Hall, "Experience Using "MOSS" to Detect Cheating On Programming Assignments," *Deparment of Computer Science and Engineering.* |
| [3] | S. Scheilmer, D. Wilkerson and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," *SIGMOD,* 2003. |
| [4] | M. Heon and D. Murvihill, "Program Similarity Detection with Checksims," *Worchester Polytechnic Institute,* 2015. |
| [5] | M. Ducarik, E. Krsak and P. Hrkut, "Current trends in source code analysis, plagiarism detection and issues of analysis big datasets," in *International scientific conference on sustainable, modern, and safe transport*, Zilina, 2017. |
| [6] | Z. Tian, Q. Zheng, T. Liu, M. Fan, E. Zhuang and Z. Yang, "Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences," *IEEE Transactions on Software Engineering,* pp. 1217-1235, 2015. |
| [7] | D. Fu, Y. Xu, H. Yu and B. Yang, "WASTK: A Weighted Abstract Syntax Tree Kernel Method for Source Plagiarism Detection," *Scientific Programming,* vol. 2017, 2017. |
| [8] | L. Prechelt, G. Maphohl and M. Philippsen, "Finding Plagiarisms among a Set of Programs with Jplag," *Journal of Universal Computer Science,* vol. 8, no. 11, pp. 1016-1038, 2003. |
| [9] | S. Sharma, C. S. Sharma and T. Veena, "Plagiarism Detection Tool "Parikshak"," in *International Conference on Communication, Information & Computing Technology*, Mumbai, 2015. |
| [10] | M. Agrawal and D. K. Sharma, "A State of Art on Source Code Plagiarism Detection," in *International Conference on Next Generation Computing Technologies*, Dehradun, 2016. |