DOI: https://doi.org/10.24113/ijoscience.v3i3.60

Plagiarism Detection Framework: A Technique for Detecting Source Code Plagiarism

Ankur Nagaich
M.Tech Scholar
Department of ECE
Shri Ram College of Engineering and Management
Gwalior, M.P., India
ankurnagaich1@gmail.com

Anuj Bhargava
Professor
Department of ECE
Shri Ram College of Engineering and Management
Gwalior, M.P., India
medc.mtech13@gmail.com

Prashant Badal
Professor
Department of ECE
Shri Ram College of Engineering and Management
Gwalior, M.P., India
medc.mtech13@gmail.com

***Abstract*— Academic dishonesty is a universal problem. The educational community across the world is facing the increasing problem of plagiarism. This widespread problem has motivated the need of an efficient, robust and fast detection procedure that is difficult to be achieved manually. Detecting duplicated text among natural language artifacts is a well-documented task. However, performing similar analysis on source code presents unique problems. Source-code plagiarism detection in programming, concerns the identification of source-code files that contain similar and/or identical source-code fragments. In this paper, a brief discussion of source code Plagiarism is presented as well as comparative study of the application of various techniques in textual similarity processing on source code.**

Keywords—Source-Code, Programming Language, Plagiarism Detection, Textual Similarity;

I. INTRODUCTION

Academic dishonesty is a serious issue that may result in severe consequences. Notably, among a number of offenses, plagiarism is given specific attention. The term plagiarism refers to the appropriation of, buying, receiving as a gift, or obtaining by any means material that is attributable in whole or in part to another source, including words, ideas, illustrations, structure, computer code, etc. As numerous tools exist for automated plagiarism detection (PD) for natural language text, institutions often include plagiarism filters as part of the assignment submission process. The abstract nature of computer source code, however, limits the feasibility of applying such tools to computational artifacts [1]. Although programming languages have notoriously rigid syntax specifications to negate possible ambiguities in program semantics, other features, such as arbitrary identifier names, variable whitespace, and nonlinear sequencing of code, present difficulties in textual similarity analysis unique to program source code.

Plagiarism of source-code is a growing problem due to the growth of source-code repositories, and digital documents found on the Internet. Plagiarism is considered as one of the most severe problems in academia, due to the availability of source-code found on-line and also due to sharing of source-code solutions among students. Essentially, plagiarism in computer programs occurs when a person reuses source code authored by someone else and fails to acknowledge to author in [1]. Plagiarism detection in software programs, concerns the identification of source-code files that contain similar and/or identical source-code fragments. When two programs have been written for solving the same problem in the same programming language, it is very likely that these source code solutions will contain code which is similar. For this reason, ample consideration and scrutiny must take place prior to classifying two programs as similar. Importantly, within plagiarized files, similarity does not occur by coincidence; the similar source-code fragments must share source-code similarity which is significant enough to classify the two programs as similar [2]. This particularly applies to student solutions, where a programming problem is assigned to a class of students who are required to provide their solution written in a particular programming language.

Hence, the challenge in plagiarism detection systems is to retrieve files which have significant code, and not to overwhelmingly detect files which contain several small and insignificant fragments occurring in several files (this is considered as noise in the data), as this will add the extra burden of time on the academic, searching through a large number of files to detect the ones which contain proof of plagiarism. Similar source-code fragments which are common across many files, essentially add noise to the problem of plagiarism detection [3].

In section II, we review different techniques of source code plagiarism detection. Section III discusses about some existing techniques for source code plagiarism

detection. In Section IV, the plagiarism detection framework is proposed. Our discussion and findings are summarized and the paper is concluded in section V.

II. RELATED WORK

The concerns about source code plagiarism increasingly rose since 1977. The assessment of students' programming submissions has an important effect on the whole computing educational procedure. It is of a great importance to evaluate the programming skills of each student, but the evaluation results become misleading and unreal due to the plagiarism problem. One of the successful policies to prevent and decrease this problem is plagiarism detection. Manual detection was found to be inefficient but it is effort and time consuming (evaluating n programs requires O(n2) cost). Hence, an automated plagiarism detection system becomes essential, which lead to the emergence of a series of plagiarism detection systems started since mid-1970s, [3, 4].

Parker and Hamblen, [1], defined plagiarism in software as: "a program which has been produced from another program with a small number of routine transformations". The most challenging aspect in code-plagiarism detection is the techniques that the implicated students tend to use to disguise the copied code in order to mislead the grader.

The Running Karp-Rabin Greedy-String-Tiling algorithm (RKR-GST) is a well-known token matching algorithm developed initially within the YAP3 plagiarism detection tool [2]. JPlag, [4], A Token-based system that is freely available on the Internet for academic use. It supports four different programming languages; Java, C, C++ and scheme and it is platform independent. It output the similarity scale between each pair of programs in the dataset. The major limitation of JPlag is that it requires parsing the dataset; if a program fail to be parsed it will be omitted from the dataset.

MOSS (Measure of Software Similarity), [5], A free available plagiarism detection system for academic usage only. MOSS supports eight different programming languages and two platforms; UNIX and Windows. It uses a string matching algorithm to divide the source-code programs into k-grams, hash them, select a subset of these hashes as fingerprints and finally compare these fingerprints.

Arwin, [6], lists the most common disguises; which are changing comments, changing formatting, changing identifiers, changing the order of operands in expressions, changing data types, replacing expressions by equivalents, adding redundant statements, changing the order of time-independent statements, changing the structure of iteration statements, changing the structure of selection statements, replacing procedure calls by the procedure body, introducing no structured statements, combining original and copied program fragments and the translation of source code from one language to another.

Kustanto and Liem [7] proposed a tool for detecting source-code plagiarism among programs written in the LISP and Pascal programming languages. Their approach is a token-based approach which essentially comprises of two steps: firstly, it parses the source code and transforms it into tokens, and thereafter compares each pair of token strings obtained in the first step using the RRKRGST algorithm.

More recently, Muddu et al. [9] propose a token representation approach for programs written in the Java programming language, and then use the RKRGST algorithm to detect code similarity. They compared their approach to other plagiarism detection tools, namely the Copy Paste Detector (CPD). One of the problems encountered is that files must parse to be included in the comparison for plagiarism, and this can cause similar files that were not parsed to be missed. Other hybrid approaches include that of Ajmal [10] who proposed a source-code plagiarism detection system which also utilizes a greedy string tiling algorithm.

However, Fuzzy logic approaches have been successfully applied to cluster source-code for the recovery of source-code design patterns, source code mining, to assess similarity within program. Samples, to derive rules in order to detect security defects in programs, to find traceability links between reports and source-code. Giovanni Acampora and Georgina Cosma [11], proposes a novel Fuzzy-based approach to source-code plagiarism detection, based on Fuzzy C-Means and the Adaptive-Neuro Fuzzy Inference System (ANFIS).

III. EXISTING SOURCE CODE PLAGIARISM DETECTORS

Source code plagiarism or it known as programming plagiarisms typically done by students in universities and colleges is outlined act or trial to use, reuse, convert and modify or copy the entire or the a part of the source code written by someone else and utilized in your programming while not citation to the owners. Source code detection primarily needs human intervention if they use Manual or automatic source code plagiarism detection to make a decision or to work out whether or not the similarity because of the plagiarism or not. Manual detection of source code during a massive number of student homework's or project it's thus troublesome and desires highly effort and stronger memory, it appears that impossible for a big number of sources. Plagiarism detection system or algorithms utilized in source-code similarity detection are often classifies

according to Roy and Cordy [12] are often classified as based on either:

- Strings: Search for precise textual matches of segments, as an example five-word runs. Fast, however are often confused by renaming identifiers.
- Tokens: Like strings, however using a lexer to convert the program into tokens 1st. This discards whitespace, comments, and identifier names, making the system more robust to easy text replacements. Most academic plagiarism detection systems work on this level, using completely different algorithms to measure the similarity between token sequences.
- Parse Trees - build and compare parse trees. this permits higher-level similarities to be detected. As an example, tree comparison will normalize conditional statements, and find equivalent constructs as just like each other.
- Program Dependency Graphs (PDGs) - a PDG captures the actual flow of control in a program, and permits a lot of higher-level equivalences to be placed, at a larger expense in quality and calculation time
- Metrics - metrics capture 'scores' of code segments according to certain criteria; as an example, "the variety of loops and conditionals", or "the variety of different variables used". Metrics are easy to calculate and may be compared quickly, however may also cause false positives: 2 fragments with identical scores on a set of metrics could do entirely different things.
- Hybrid approaches - as an example, parse trees + suffix trees will combine the detection capability of parse trees with the speed afforded by suffix trees, a sort of string-matching knowledge structure".

There are many methods developed by researcher for source code plagiarism detection as discussed below:

A. SIM

SIM is employed to detect plagiarism of code written in Java, C, Pascal, Modula-2, Lisp, Miranda [13]. SIM is additionally used to check similarity between plain text files. SIM converts the 6 source code into strings of token and so compare these strings by using dynamic programming string alignment technique. This system is additionally used in DNA string matching. The alignment is incredibly expensive and exhaustive computationally for all applications as a result of for large code repositories SIM isn't scalable. The source code of SIM is available publicly however it's no more actively supported.

B. MOSS

MOSS is accessible free to use in academics and it's accessible as an online service .MOSS support ADA programs, Java, C, C++, plain text and Pascal [14]. At a similar time moss conjointly support UNIX system OS and windows operating systems. First of all MOSS convert source code into tokens then use robust winnowing algorithm. Robust winnowing algorithm is introduced by Schleimer et al. however the internal detail of operating of this formula is confidential. This algorithm takes the document fingerprints by choosing a set of token hashes. Within the comparison method of set of files, "MOSS creates an inverted index to map document fingerprints to documents and their positions at intervals every document. Next, every program file is employed as a query against this index, returning a list of documents within the collection having fingerprints in common with the query." the amount of matching fingerprints of every pair of document within the set of files is that the results of MOSS. MOSS sort these results and show highest-score matches to user.

C. Plague

One of the earliest structure-oriented systems is Plague. Plague solely support programs written in C [15]. This tool works in many steps. At the very beginning, source code is regenerate into structure profiles. When this Plague uses Heckel algorithmic program to check generated structure profiles of first step. The algorithm is essentially designed for plain text files and it's introduced by Paul Heckel. Plague returns ends up in list and so use an interpreter to process this list to point out results in the way, in order that common user can recognize it easily.

D. JPlag

JPlag is offered publicly as free accessible service. JPlag may be accustomed check plagiarism of source code written in C, C++ and Java [16]. We tend to gave a directory of programs as input in JPlag. Initial of all source code within the directory is parsed so remodeled into token strings. While in process of transformation, JPlag compare these strings by using Running Karp-Rabin Greedy String tiling (RKR-RGST) rule. The comparison result then shown in hypertext markup language file, which may be visited by using any browser. Within the hypertext markup language file of results main page contains pairs of programs that are assume to be plagiarized.

E. PDE4Java

Plagiarism Detection Engine for Java (PDE4Java) detects code-plagiarism by applying data processing techniques. The engine consists of 3 main phases; Java tokenization, similarity measurement and cluster. It's an elective default tokeniser that creates it versatile to be used with nearly any programming language. The system provides

a visualizing illustration for every cluster besides the textual representation [17].

IV. PLAGIARISM DETECTOR FRAMEWORK

This section introduces an innovative computational intelligence framework for the purpose of analyzing source-code in the context of source-code plagiarism detection. The Plagiarism detection framework consists of two phases i.e. learning phase and detection phase as shown in figure 1.

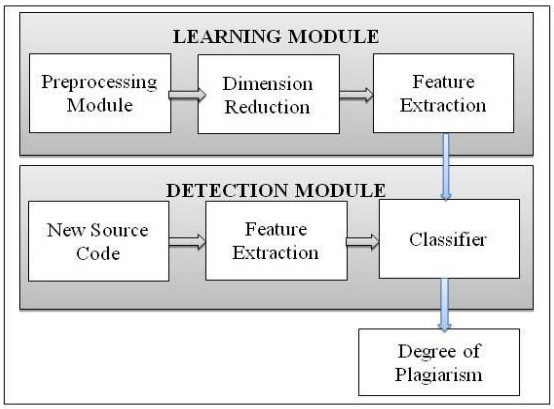

Figure 1: Plagiarism detection framework

The learning phase consists of preprocessing, feature extraction and classifier module. The purpose of the plagiarism preprocessing module is to pre-process the source-code files in such way that it removes unnecessary and meaningless terms and characters in order to reduce the size of the data to more efficiently capture the semantic representation of each source-code file. This module tokenize, process of breaking a stream of text up into words, called tokens. Thereafter the following pre-processing parameters are also applied: conversion of upper-case letters to lower case, removing terms that occur in one file because such terms hold no information about relationships among terms across files, removing terms solely composed of numeric characters, removing syntactical tokens (e.g. semi-colons, colons), removing terms consisting of a single letter. The following source-code specific pre-processing parameters are also applied: removal of comments, removal of language reserved terms, removal of obfuscation parameters found in terms, obfuscators that join two words together are removed such that the two words are treated as a single word.

In dimension reduction module, the proposed framework reduce the database size and hence space complexity by removing noise and irrelevant data. Then in detection module, the proposed framework detects the new source code to analyze degree of similarity. Different classifier can be used to determine degree of similarity such as fuzzy logic based classifier, adaptive Neuro based classifier, Fuzzy C-Means, Self-Organizing Map, hybrid classifier, etc.

V. CONCLUSION

Plagiarism in source-code submissions is a serious problem that has motivated researchers to find effective automated detectors. This paper proposed a plagiarism detection engine for source-code files. Source-code plagiarism detection in programming, concerns the identification of source-code files that contain similar and/or identical source-code fragments. Mainly, the system computes and displays the similarity value between each pair of programs in a dataset. This paper proposes an approach to cluster source-code for detecting clusters which could contain similar and hence plagiarized files. The proposed approach appears to overcome the several problems currently encountered by plagiarism detection algorithms and many of the existing algorithms and tools. To detect source code plagiarism, the proposed approach may optimize the speed the detection process as well as the accuracy of finding degree of plagiarism.

REFERENCES

[1] A. Parker and J. Hamblen. Computer algorithms for plagiarism detection. IEEE Transactions on Education, 32(2):94–99, 1989.

[2] Michael J. Wise. YAP3: Improved detection of similarities in computer program and other texts. ACM, SIGCSE, 28:130–134, 1996.

[3] Fintan Culwin, Anna MacLeod, and Thomas Lancaster. Source code plagiarism in UK HE computing schools, issues, attitudes and tools. Technical report, South Bank University (SBU) SCISM Technical Report, 2001.

[4] Michael Philippsen Lutz Prechelt, Guido Malpohl. Finding plagiarism among a set of programs with JPlag. Journal of Universal Computer Science, 8(11):1016–1038, 2002.

[5] Alex Aiken. Moss: A system for detecting software plagiarism, 2005.

[6] Christian Arwin and S.M.M. Tahaghoghi. Plagiarism detection across programming languages. Proceedings of the 29th Australasian Computer Science Conference, 48:277–286, 2006.

[7] C. Kustanto and I. Liem, “Automatic source code plagiarism detection,” in Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD ’09. 10th ACIS International Conference on, May 2009, pp. 481–486.

[8] C. G. and J. M., “An approach to source-code plagiarism detection and investigation using latent semantic analysis,” Computers, IEEE Transactions on, vol. 61, no. 3, pp. 379–394, March 2012.

[9] B. Muddu, A. Asadullah, and V. Bhat, “Cpdp: A robust technique for plagiarism detection in source code,” in

Software Clones (IWSC), 7th International Workshop on, May 2013, pp. 39–45.

[10] O. Ajmal, M. Missen, T. Hashmat, M. Moosa, and T. Ali, “Eplag: A two layer source code plagiarism detection system,” in Digital Information Management (ICDIM), 2013 Eighth International Conference on, Sept, 2013, pp. 256–261.

[11] Giovanni Acampora and Georgina Cosma, “A Fuzzy-based Approach to Programming Language Independent Source-Code Plagiarism Detection”, IEEE, 2015.

[12] http://www.cs.queensu.ca/queensu.ca/TechReports/Reports/2007-541.pdf.

[13] David Gitchell 81 Nicholas Tran, “Sim: A Utility For Detecting Similarity in Computer Programs”, ACM, 1999, pp. 266-270.

[14] Aiken A Moss. A system for detecting software plagiarism, http://www.cs.berkeley.edu/~aiken/moss.html.

[15] M. J. Wise, “Detection of Similarities in Student Programs: YAP'ing may be Preferable to Plague'ing,” ACM SIGSCE, 2002.

[16] Lutz Prechelt, Guido Malpohl, Michael Phlippsen, “JPlag: Finding plagiarisms among a set of programs”, Technical Report, University of Karlsruhe, Germany, 2000.

[17] Jadalla, A. Elnagar, “A. PDE4Java: Plagiarism Detection Engine for Java sourcecode: a clustering approach”, IJBIDM, vol. 3, issue 2, 2008, pp. 121-135.