ISSN:0975-9646

Software Plagiarism Detection Techniques: A Comparative Study

Divya Luke[1], Divya P.S[2], Sony L Johnson[3], Sreeprabha S[4] ,Elizabeth.B.Varghese[5]

[1]*Marthoma College of Management and Technology, Perumbavoor,*
[2]*Narayanaguru College of Engineering, Kanyakumari,*
[3, 4, 5] *Mar Baselios College of Engineering and Technology, Trivandrum*

Abstract **-Plagiarism defines as "the act of the writings of another person and passing them off as one's own. The fraudulence is closely related to forgery and piracy - practices generally in violation of copyright laws. Software plagiarism has been an important issue in software industry for intellectual property and software license protection, especially for open source projects. Thus it is important to develop robust and effective approaches to software plagiarism detection. In this paper we compare six tools for detecting plagiarism: GPlag, JPlag, Marble, MOSS, plaggie and SIM. The criteria we used for qualitative comparison are supported languages, extendibility, presentation of results, usability, exclusion of template code, exclusion of small files, historical comparison, submission or file based rating, local or web-based and open source.**

Keywords: **Plagiarism, Program Dependency Graph (PDG), tokens, source code.**

I. INTRODUCTION

Rapid development of internet technologies simplified sharing any kinds of data. Extremely notable is also sharing the source codes. Consequently, today's "copy-paste" generation is a subject of a notable problem of plagiarism. It is present in many areas, from educational and research areas to software development.

There are two types of plagiarism are more occurs:
1. Textual plagiarisms: this type of plagiarism usually done by students or researchers in academic enterprises, where documents are identical or typical to the original documents, reports, essays scientific papers and art design.
2. A source code plagiarism: also done by students in universities, where the students trying or copying the whole or the parts of source code written by someone else as one's own, this types of plagiarism it is difficult to detect.

An important differential between source code plagiarism [1] and free text plagiarism is that the methods used to detect both of these differ. Source code detection is a well-understood area that has not recently been the focus of much research. It is thought to be easier to detect source code plagiarism than free text plagiarism since the language that can be used is constrained to a set of defined key words and since any plagiarism is most likely intra-corpal in nature [2]. Free text plagiarism contains [3] an effectively unlimited number of possible words that can be used and plagiarism may be intra or extra-corpal. Research on detecting plagiarism in free text is more recent and ongoing and has become possible due to the increasing availability of cheap computer processing power.

This paper is about comparing different source code plagiarism detection systems. Comparisons of program plagiarism detection tools [4] can be roughly divided into two categories: feature comparisons and performance comparisons. Feature comparisons are qualitative comparisons; they describe the properties of a tool, like which programming languages it supports, whether it is a local or a web-based application, which algorithm is used to compare the files. By nature, such a comparison is purely descriptive, and based on such a comparison it is difficult to say which of the tools should be considered 'the best'. Performance comparisons are quantitative comparisons; they typically compare the results of tools, rather than their properties.

In section II software plagiarism detection tools has been discussed. Next, in section III a comparative study of different plagiarism detection tools are discussed.

II. SOFTWARE PLAGIARISM DETECTION TOOLS

A. GPlag

GPlag [5] was developed by Chao LIU, Chen Chen, Jiawei Han at the University of Illinois-UC, Urban in 2006. GPlag, which detects plagiarism by mining program dependence, graphs (PDGs). A PDG is a graphic representation of the data and control dependencies within a procedure. The PDG thus developed from original program and modified program are checked whether it is copied or not by graph isomorphism. In order to make GPlag scalable to large programs, a statistical lossy filter is proposed to prune the plagiarism search space. The program dependence graph, first proposed by Ferrante, it has previously been used in the identification of duplicated code for the purpose of software maintenance.

IMPLEMENTATION OF GPLAG

Algorithm GPlag (P, P0, K, Ý, α)
Input P: The original program
P0: A plagiarism suspect
K: Minimum size of nontrivial PDGs, default 10
Ý: Mature rate in isomorphism testing, default 0.9
α: Significance level in lossy filter, default 0.05
Output: F: PDG pairs regarded to involve plagiarism
1: G = The set of PDGs from P
2: G0 = The set of PDGs from P0
3: GK = {G|G € G and |G| > K}
4: G0K = {G0|G0 € G and |G0| > K}
5: for each G€ GK

6: let G0 K, G= {G0|G0 € G0 K, |G0| , Ý|G|, (G, G0) passes filter}
7: for each G0 € G0K, G
8: if G is Ý-isomorphic to G0
9: F = F Ü (G, G0)
10: return F;

This algorithm outlines the work-flow of GPlag, a PDG based plagiarism detection tool. It takes as input an original program P and a plagiarism suspect P0, and outputs a set of PDG pairs that are regarded as involving plagiarism. By examining these returned PDG pairs, it is possible to confirm plagiarism and/or eliminating false positives. At lines 1 and 2, PDGs of the two programs are collected. Then at lines 3 and 4, PDGs smaller than K are excluded. Finally, from lines 5 to 10, GPlag searches for plagiarism PDG pairs. For each g that belongs to the original program, line 6 obtains all g0's that survive both the lossless and the lossy filters. And line 8 performs the Ý-isomorphism testing.

B.JPlag

JPlag [6] was developed by Guido Malpohl at the University of Karlsruhe. In 1996 it started out as a student research project and a few months later it evolved into a first online system. In 2005 JPlag was turned into a web service by Emeric Kwemou and Moritz Kroll. JPlag converts programs into token strings that represent the structure of the program, and can therefore be considered as using a structure-based approach. For comparing two token strings JPlag uses the Greedy String Tiling" algorithm as proposed by Michael Wise but with different optimizations for better efficiency.

JPlag is a system that finds similarities among multiple sets of source code files. JPlag currently supports Java, C#, C, C++, Scheme and natural language text. JPlag has a powerful graphical interface for presenting its results. It takes input as set of programs, compares these programs pair wise (computing for each pair a total similarity value and a set of similarity regions), and provides as output a set of HTML pages that allow for exploring and understanding the similarities found in detail. JPlag works by converting each program into a stream of canonical tokens and then trying to cover one such token string by substrings taken from the other (string tiling).
JPlag operates in two phases:
1. All programs to be compared are parsed (or scanned, depending on the input language) and converted into token strings.
2. These token strings are compared in pairs for determining the similarity of each pair. During each such comparison, JPlag attempts to cover one token stream with substrings ("tiles") taken from the other as well as possible. The percentage of the token streams that can be covered is the similarity value. The corresponding tiles are visualized by the interface

C.Marble

Marble [7] is a tool developed in 2002 at Utrecht University. Marble is a simple, easily maintainable tool that can be used to detect cases of suspicious similarity between Java submissions. Marble uses a structure-based approach to compare the submissions. It starts by splitting the submission up into flies so that each file contains only one top-level class. The next phase is one of normalization, to remove details from these files that are too easily changed by students: a lexical analysis is performed implemented in Perl using regular expressions that preserves keywords like class, for and frequently used class and method names like String, System, and tostring. Comments, excessive white-space, string constants and import declarations are simply removed; other tokens are abstracted to their token type. Marble is mainly tailored to Java/C#, but variants made and applied to PHP, Perl and XSLT. Marble includes two phases, they are
1. The normalisation phase: In this phase it transforms source code into a special form suited for literal comparison.
2. The detection phase: actually performs the comparisons and ranks the results.

Normalisation removes unessential detail from source files. In particular, details those are easy to change without changing the behaviour of the program. It is done either by tool, or by hand. For example consider a Java source file. Then split them up into a separate file for each class. For each of these files, residing at top level, normalise the Java source code. In code normalisation the comments and literal strings and characters are removed and map identifiers except for keywords (while), special constants (true), special methods (wait) and special types (String). It keeps these special identifiers to avoid false positives and retain symbols like assignments, braces, arithmetic symbols.

D.MOSS

Moss [8] is an acronym for Measure Of Software Similarity. Moss was developed in 1994 at Stanford University by Aiken et al. It is being provided as a web service that can be accessed using a script. The moss submission script works for Unix/Linux platforms and may work under Windows with Cygwin. To measure similarity between documents, moss compares the standardized versions of the documents: moss uses a document fingerprinting algorithm called winnowing. Document fingerprinting [9] is a technique that divides a document into contiguous substrings, called k-grams, with k being picked by the user. Every k-gram is hashed, and a subset of all the k-gram hashes is selected as the document's fingerprint.

Moss is an automatic system for determining the similarity of programs. Moss can currently analyse code written in the following languages: C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.Moss is also being provided as an Internet service. In response to a query the Moss server produces HTML pages listing pairs of programs with similar code. Moss also highlights individual passages in programs that appear the same, making it easy to quickly compare the files. Finally, Moss can

automatically eliminate matches to code that one expects to be shared (e.g., libraries or instructor-supplied code), thereby eliminating false positives that arise from legitimate sharing of code.

E.Plaggie

Plaggie [10] is a source code plagiarism detection engine meant for Java programming exercises. In appearance and functionality, it is similar to JPlag. Plaggie must be installed locally and its source code is open. Plaggie was developed in 2002 by Ahtiainen et al. at Helsinki University of Technology. It is a stand-alone command line Java application. The basic algorithm used for comparing two source code files is the same as for JPlag: tokenization followed by Greedy String Tiling. The authors mention that they did not implement the optimisations that were implemented in JPlag. Plaggie can check programs that are written in Java 1.5 also known as Java 5. Plaggie is GNU-licensed.

F.SIM

SIM [11] is a software similarity tester for programs written in C, Java, Pascal, Modula-2, Lisp, Miranda, and for natural language. It was developed in 1989 by Dick Grune at the VU University Amsterdam. The process SIM uses to detect similarities is to tokenize the source code first, then to build a forward reference table that can be used to detect the best matches between newly submitted files, and the text they need to be compared to.

SIM detects similarities between programs by evaluating their correctness, style, and uniqueness. Each program is first parsed using the flex lexical analyser, producing a sequence of integers (tokens). The tokens for keywords, special symbols, and comments are predetermined, while the tokens for identifiers are assigned dynamically and stored in a shared symbol table; whitespace is discarded. The token stream of the second program is grouped into sections, each representing a module of the program; each section is separately aligned with the token stream of the first program. An alignment of two strings is performed by inserting spaces between characters to equalise their length. An alignment scoring scheme is used to calculate similarity. This rewards matches involving two identifiers by two points and other matches by one point. It also penalizes mismatches involving two identifiers by two points and other mismatches by one point. SIM can handle name changes and reordering of statements and functions.

III. RESULTS AND DISCUSSIONS

Parker and Hamblen define source code plagiarism[12] 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 program. So it is very important to find out an effective plagiarism detection tool.

A.Criteria for qualitative comparison

The criteria that we use for our qualitative comparison of the selected tools are:

1. Supported languages: The minimal requirement for tools to be included in this comparison was to support plagiarism detection in Java source code files, but some tools support several other languages.
2. Extendability: It is the ability of a tool to be adapted or configured so that it can be used for other programming languages.
3. Presentation of results: After the running of the tools, a lot of effort has to be done to check if found similarities between files. In most cases, this takes a lot more time than running the query itself. Therefore, it is important to present the results in such a way that post processing can be done as efficiently as possible. A good presentation of the results should at least contain the following elements:
   Summary: Here meta data like the total number of submissions, the successful parses, the parameters used for running the detection and a chart showing the distribution of similarities over the result should be shown. Such a histogram can help identifying the range of similarities that clearly represent no plagiarism, and the range of values that should be investigated further.
   Matches: The matches should be listed sorted by similarity, in a comprehensive way. This can be done pair wise, or in clusters. It should also be possible to set a certain threshold on the minimum similarity to include in the result overview.
   Comparison tool: To be able to easily compare pairs that are marked as 'similar' it is helpful if there is an editor that is able to display both files next to each other, highlighting the similarities.
4. Usability: Another criterion is the ease of use of the tool. It should be possible for a user to use the tool without having to spend a lot of effort in getting the tool to work.
5. Exclusion of template code: It is normal for student programming assignments to share some common base code from which students have to complete an assignment. Also, it often happens that something explained in the lectures can be used in a programming assignment. In both cases, the results of a search for plagiarism may include many legitimate matches that do not indicate plagiarism, but can be explained by one of the previously mentioned legitimate causes. Some plagiarism detection tools allow the user to place such legitimately shared code in a common base file that will be ignored during the detection phase. This can help prevent a lot of false positives.
6. Exclusion of small files: Related to the exclusion of template code is the exclusion of small files. Very small files-such as so called Java beans, that only consist of attributes and their getter and setter methods-are most likely to return high similarity scores. This is simply a result of the way such classes are implemented and does not indicate plagiarism. A tool can either mention the file size in its result, which can help in detecting false positives caused by small files,

or it may provide a way to exclude files up to a certain size.

7. Historical comparisons: With this criterion it denotes the ability of a tool to compare a new set of submissions with submissions from older incarnations of the same programming assignment, without again mutually comparing the older incarnations. So there must be a way to distinguish older submissions from newer ones. This is done either by indicating the new and old submissions when starting the tool, or by putting different incarnations in different directories.
8. Submission or file-based rating: Whether a tool rates the submissions by every separate file or by submission that is a directory containing the files of which the program consists greatly influences the readability of the output. When a submission consists of multiple files, it is important that the plagiarism detection is performed for each file in the submission, since the detection of plagiarism in only one of the files of the submission is enough to consider the whole submission as being plagiarized and therefore invalid. When a submission based rating is used the comparison might still be file based. Then the question is how the file comparison scores are combined into a score for the whole submission.
9. Local or web-based: Some tools are provided as web services. This requires a lecturer to send the student assignments over the network. Here you take a risk of exposing confidential information to the outside world. Other tools have to be downloaded and run locally.
10. Open source: An advantage of open source is of course the possibility of extending or improving the program to better suit the situation you intend to use it for.

In Table 1 we summarize the evaluation for easy comparison. Instead of mentioning all supported languages of the tools again, we have simply counted them. For the criteria 3 – Presentation of results and 4 - Usability we introduce a scale of 1 to 5, 1 meaning poor and 5 meaning very good.

As given in Table 1 Gplag can be used for all languages. It is the fastest method for finding plagiarism pairs takes 0.1 second for the whole procedure to complete. The output is presented as table which contains list of procedures together with what disguises are applied to each of them. By using graph isomorphism algorithm we can easily find out which are the isomorphic pairs used. Exclusion of template code and small files helps to increase the efficiency. Gplag is open source and is available as web based and as local service.

Jplag mainly supports Java, C#, C, C++, Scheme and natural language text. In Jplag program is converted into token strings using parser and scanner depending on the programming language used. JPlag presents its results as a set of HTML pages. The pages are sent back to the client and stored locally. The main page is an overview that includes a table with the configuration used to run the query, a list of failed parses, a chart showing the distribution of the similarity values, and listings of the most similar pairs, sorted by average similarity as well as by maximum similarity. One distinctive feature of JPlag is the clustering of pairs. This makes it easier to see whether a submission is similar to several other submissions. It is possible to submit a base code directory containing files that should be excluded from the comparison.

Table 1
Comparison of plagiarism tools

Feature	GPlag	JPlag	Marble	Moss	Plaggie	SIM
1 - Supported languages	All	6	1	23	1	5
2 - Extendability	Yes	No	No	No	No	Yes
3-Presentation of results (1-5)	5	5	3	4	4	2
4.Usability	5	5	2	4	3	2
5 - Exclusion of template code	Yes	Yes	No	No	Yes	No
6 - Exclusion of small files	Yes	Yes	Yes	Yes	No	No
7 - Historical comparisons	No	No	Yes	No	No	Yes
8 - Submission or file based	Submission	Submission	File	Submission	Submission	File
9 - Local or web-based	Web/Local	Web	Local	Web	Local	Local
10-Open source	Yes	No	No	No	Yes	Yes

In Marble, it mainly supports Java, Perl, PHP and XSLT is experimental. The language-dependent part is the normalization phase, which can easily be adapted for similar programming languages. Here the results are outputted to a script named sorted or unsorted. The user may also choose not to run the script, but to open it in a text-editor and manually investigate the suspects. If submissions are stored in an appropriately ordered file system that is one directory per assignment, divided into subdirectories for the different incarnations, which are divided into subdirectories for the different reviewers, that contain the submissions of that incarnation , then Marble is able to compare each file of a new submission to not only the files from submissions inside the same incarnation, but also to those in older incarnations without comparing the older submissions among themselves.

For every pair of files a similarity rating is computed.

In Moss, it supports almost all programming languages. The output of moss is an HTML presentation with clickable links and an integrated HTML diff editor that allow for easy navigation through the results. It is placed on a web page on the moss web server. When registering to Moss a submission script is mailed that can be used to upload the submissions. Moss allows one to supply a base file of code that should be ignored if it appears in programs. By default, moss compares files based on submissions or directories; however, the submission script exposes an option that allows for file to file comparison.

Plaggie supports Java 1.5 and the results are shown in plain text on the standard output which is stored in a graphical HTML format (using frames). It also offers an option to disable the plain text output. The output includes a table showing statistics such as the distribution of the different similarity values, the number of files in submissions, etc. The HTML report includes a sortable table containing the top results and their various similarity values. For further inspection a submission can be clicked which leads us to a side-by-side comparison of the files, highlighting the similarities. Configuring Plaggie has to be done via a configuration file that is placed in the directory containing the submissions. Running Plaggie is done using its command line interface. Template code can be excluded by providing the file containing the template code. In addition, Plaggie offers the possibility to exclude code from the comparison based on filename, subdirectory name, or interface. Plaggie compares file by file, but accumulates the results per submission.

SIM supports C, Java, Pascal, Modula-2, Lisp, Miranda and natural language texts. SIM can be readily extended by providing a description of the lexical items of a new language. The results of SIM are presented in a at text file that first outputs some general information about the compared files, such as number of tokens of each of the files, total number of files, names, etc. SIM's output is on a per-file basis; however, files are only mutually compared if they come from different submission directories.

IV. CONCLUSION

In this paper we have compared six plagiarism detection tools with respect to ten tool features. Performance was compared by a sensitivity analysis on a collection of intentionally plagiarized programs and on a set of real life submissions. The performance was also compared by examining the top 10 results for each tool to the results of the others. The results of the comparison give good insight into the strong and weak points of the different tools.

Our findings from the comparison can be summarized as follows:

- By comparing these tools the most efficient one is GPlag.
- Many tools are sensitive to numerous small changes.
- All tools do well for the majority of single refactoring, but many tools score rather badly.
- A striking result of the top-10 comparison is that the top-10's for GPlag, JPlag, Marble and MOSS are fairly similar, whereas the top-10's of Plaggie and SIM differ quite a lot from the other four
- Along the way we have discovered a few cases where a more detailed investigation of the behaviour is needed.

REFERENCES

[1] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared information and program plagiarism detection," *IEEE Trans. Inf. Theory*,vol. 50, no. 7, pp. 1545–1551, 2004.

[2] Webster's Online Dictionary, www.webstersonline- dictionary.org

[3] A. Aiken *et al.*, Moss: A System for detecting software plagiarism

[4] Schiller Rosita M., E-Cheating: Electronic Plagiarism, Journal of the American Dietetic Association, 105 (7), 2005, pp. 1058 - 1062.

[5] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. *Gplag: Detection of software plagiarism by program dependence graph analysis.* In the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD06, pages 872- 881. ACM Press, 2006.

[6] http://wwwipd.ira.uka.de/jplag/.

[7] Jurriaan Hage. *Programmee plagiaa detectie met marble*. Technical Report UU-CS-2006-062, Department of Information and Computing Sciences, Utrecht University, 2006.

[8] http://theory.stanford.edu/~aiken/moss/.

[9] Steven Burrows, S. M. M. Tahaghoghi, and Justin Zobel. Efficient plagiarism detection for large code repositories. Softw. Pract. Expert, 37(2):151-175, 2007.Univ. California, Berkeley,CA,2005[Online].Available: www.cs.berkeley.edu/aiken/moss.html

[10] Aleksi Ahtiainen, Sami Surakka, and Mikko Rahikainen. Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises. In Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on computing education research, pages 141,142, New York, NY, USA, 2006. ACM.

[11] D. Gitchell and N. Tran, "Sim: A utility for detecting similarity in computer programs," in *Proc. Tech. Symp. Comput. Sci. Ed.*, 1999, pp.266–270.

[12] Source Code Plagiarism: Proceedings of the *ITI 2009 31st Int. Conf. on Information Technology Interfaces,* June 22-25, 2009, Cavtat, Croatia.