Code signatures for plagiarism detection

Dennis Goßler Dennis Wäckerle

November 8, 2022

Abstract

Contents

1 Introduction

2 How Does Plagiarism Detection Work

Fand den Abschnitt bei moss ganz nett könnte man ja mit reinpacken Dennis G.

Moss and other plagiarism detection tools are not perfect, so a human should go other the results, and it should be checked if the clams are valid. "In particular, it is a misuse of Moss to rely solely on the similarity scores. These scores are useful for judging the relative amount of matching between different pairs of programs and for more easily seeing which pairs of programs stick out with unusual amounts of matching. But the scores are certainly not a proof of plagiarism. Someone must still look at the code." (Aiken, 2021)

3 Use Case and Software Experiment

3.1 Use Case

3.2 Software Experiment

4 Criteria for Evaluation

5 Evaluation of the different Tools

	C	C++	C#	Java	Kotlin	Python	PHP	VB.net	Javascript	Is expandable?
MOSS	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
JPlag	No	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes
Plaggie	No	No	No	Yes	No	No	No	No	No	Yes[1]
AC2	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes

Table 1 [The native supported programming languages for each plagiarism detection algorithm]

5.1 MOSS

Moss clams to be one of the best cheating detection algorithms. "The algorithm behind moss is a significant improvement over other cheating detection algorithms (at least, over those known to us)." (Aiken, 2021)

Moss supports the following programming 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 and HCL2

5.2 JPlag

5.2.1 JPlag's comparison algorithm

- Functions in two phases 1. all programs are parsed and converted into tokens 2. tokens strings are compared in pairs. Tries to cover one token stream with substrings of the other token. Percentage of covered token streams is the similarity. (Perchelt et al., 2000, p. 10)

[1] Plaggie is open source, therefore a custom tokenizer for different languages can be implemented.

- Tokenizing -> only language dependent process(Perchelt et al., 2000, p. 10) - Tokens represent syntactic elements e.g. statements or control structures(Sağlam, 2022, How are submissions represented? — Notion of Token)

- Transformation - each file is parsed -> result set of Abstract Syntax Trees for each submission - each AST traversed depth first, nodes are grammatical units of language - when entering and exiting a node a token can be created and added to the token list - block type node e.g. classes or if expressions have corresponding begin and end tokens. Token list should have balanced pairs of matching begin and end tokens (Sağlam, 2022, How does the transformation work?)

- Comparing token strings - Essentially Greedy String Tiling - When comparing two strings A and B, the same subsstrings should be found satisfying following rules 1. "Any token of A may only be matched with exactly one token from B." 2. "Substrings are to be found independent of their position in the string." 3. "Long substring matches are preferred over short ones" (Perchelt et al., 2000, p. 11) - Applying the third rule sequentially leads to a greedy algorithm consisting of two phases 1. Two strings are searched for the biggest contigous matches. 3 nested Loops. Outer loop iterates over tokens in string A. Second loop compares Token T to Tokens in String B. Inner loop, if identical Tokens, extends the match as far as possible. Collects Set of longest common substrings 2. Marks all matches of maximal length. All tokens are marked and can not be used for further matches. Satisfies rule 1 - Repeat until no further matches found (Perchelt et al., 2000, p. 11) - tile is a unique and permanent association of a substring from A with a matching substring from B(Wise, 1993, p. 3) - MinimumMatchLength is defined must be atleast 1, should probably be higher, since it is unlikely that such a match will be significant (Wise, 1993, GST)

-similarity considers 100% if the shorter string is completly covered (Perchelt et al., 2000, p. 13)(See maths formula)

5.2.2 The results

-comparisons without base code -successfully discovered plagiarism for m0 91.49% and 90.51% match -Other comparisons around 40% -need base code

-m4 less clear 84.77%-76.2% -need base code and another submission

5.2.3 Integration into an automated evaluation pipeline

-Can be executed locally -Has a cli and an api -> allows automation -Visualization in external web application -> might pose a problem

-known unsuccessful attacks - Changing the comments - Changing the indentation - Method and variable name changes(Ahtiainen, 2006, Known successful attacks)

-know successful attacks - Moving inline code to separate methods and vice versa - Inclusion of redundant program code - Changing the order of if-else blocks and case-blocks(Ahtiainen, 2006, Known unsuccessful attacks)

5.3 Plaggie

-Was the only open source tool when release(Ahtianien et al., 2006), JPlag now also open source

5.3.1 The Algorithm

-Uses CUP as a Parser -Uses standard LALR(1) parser generation("CUP", n.d.) -GST with no special optimization attempts -Algorithm was extended to support exclusion of common code(Ahtiainen, 2006, 4. Algorithm used)

5.3.2 The results

-TODO still have to run it.

5.3.3 Integration into an automated evaluation pipeline

-only Java 1.5(Ahtiainen, 2006; Ahtianien et al., 2006) -no further development since 2006 -has a cli -> can be automated -results as html files

5.4 AC2

"AC was born in the Escuela Politécnica Superior of the Universidad Autónoma de Madrid to deter and detect source-code plagiarism in programming assignments." (Freire, 2022)

5.4.1 AC2 comparison algorithm

The output that AC2 generates is visualization base. So AC it will not provide a value like "percentage of copy" instead, "it will create graphical representations of the degree of similarity between student submissions within a group" (Freire, 2022)

5.4.2 The results

5.4.3 Integration into an automated evaluation pipeline

6 Conclusion

References

Ahtiainen, A. (2006). *Readme for plaggie*.

Ahtianien, A., Surakka, S., & Rahikainen, M. (2006). *Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises*.

Aiken, A. (2021). *A system for detecting software similarity*. https://theory.stanford.edu/~aiken/moss/

Cup. (n.d.). Technische Universität München. Retrieved November 7, 2022, from http://www2.cs.tum.edu/projects/cup/

Freire, M. (2022). *Ac*. https://github.com/manuel-freire/ac2

Perchelt, L., Malphol, G., & Phlippsen, M. (2000, March 28). *Jplag: Finding plagiarisms among a set of programs*.

Sağlam, T. (2022, September 16). *4. adding new languages*. Retrieved October 30, 2022, from https://github.com/jplag/JPlag/wiki/4.-Adding-New-Languages

Wise, M. J. (1993). *String similarity via greedy string tiling and running karp-rabin matching*.