How do various plagiarism detection tools differ in their detection results when applied to java/spring coding exercises

Dennis Goßler Dennis Wäckerle

November 16, 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]

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

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

5.1.1 Moss comparison algorithm

5.1.2 The results

5.1.3 Integration into an automated evaluation pipeline

5.2 JPlag

JPlag is a plagiarism detection tool, which was developed by in 2000 at the University of Karlsruhe to help with the detection of plagiarized coding exercises. At that the tool was only available as a WWW service and could analyze C, C++. Scheme and Java programs (Perchelt et al., 2000, p. 4). Currently, the tool is available as an open source application which can be run locally and can be used with a CLI or a Java API. JPlag also currently supports 12 programming languages including the original four languages and more modern languages such as Kotlin ("JPlag - Detecting Software Plagiarism", 2022, Supported Languages) while also allowing to add new languages (Sağlam, 2022).

5.2.1 JPlag's comparison algorithm

JPlag's algorithm is split into two parts the tokenizing and the comparison of the token strings. During the tokenization process all programs are parsed and converted into token strings. In the second phase all token strings are compared in pairs to determine their similarity. This comparison uses a specially optimized version of the Greedy String Tiling algorithm. This algorithm tries to cover one token string with substrings of the other string. The percentage of the covered token strings is the similarity between the two programs (Perchelt et al., 2000, p. 10).

Tokenization The tokenization process is the only language dependent process of the JPlag algorithm (Perchelt et al., 2000, p. 10). The extracted tokens represent syntactic elements of the language like statements or control structures (Sağlam, 2022, How are submissions represented? — Notion of Token). During the parsing process each file is parsed with the result being a set of abstract syntax trees(AST) for each submission. Each AST is traversed depth first with nodes representing grammatical units of the language. During the traversal when entering and exiting a node a token can be created that match the type of the node and will then be added to the current list of tokens. There are block type nodes which can represent classes, if expressions or other elements of the language which have a corresponding beginning and end. When creating tokens from those nodes each have a corresponding "BEGIN" and "END" token. The token list should always have a pair of matching "BEGIN" and "END" tokens (Sağlam, 2022, How does the transformation work?).

Comparing token strings JPlag's comparison algorithm is essentially just the greedy string tiling algorithm for the comparison of two strings, however it is differently optimized to improve its runtime(Perchelt et al., 2000, p. 5). The goal of the algorithm is to find a set of substrings which are not only the same but also satisfy these three 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)

These rule have some consequences. The first rule doesn't allow the matching of code that have been duplicated while the second rule makes the reordering of the source code not viable. Furthermore, the third rule is introduced since short matches are more likely to be spurious (Perchelt et al., 2000, p. 11).

When these rule 3 is applied sequentially then a greedy algorithm consisting of two phases will be created.

In phase 1 the longest contiguous match in the two strings is searched. This is done by 3 nested loops. The outer loop iterates over all extracted tokens in string A, the second loop compares token T from string A with every token in string B. If two tokens are identical then the innermost loop extends this match as far as possible (Perchelt et al., 2000, p. 11). These matches are called tiles which are permanent and unique (Wise, 1993, p. 3).

In the second phase all matches of the maximal length are marked, when this happens all matched tokens are also marked and can't be used for any further matches. This satisfies the first rule. These two phases will be repeated until not further matches are found (Perchelt et al., 2000, p. 11). Also, of importance is the MinimumMatchLength, which defines the length at which maximal matches and any tile below the maximum matches are ignored. The smallest possible value for MinimumMatchLength is 1. However, in general this value should be higher than 1 since at that length it is unlikely that a match will be significant (Wise, 1993, p. 3)

JPlag only considers two strings to match 100% when the shorter string is completely matched by the longer string. The logic being that the longer string represents a program which has been entirely copied and then extended. The similarity between two programs can be calculated with the following formula.

$$sim(A,B) = \frac{2 + coverage(tiles)}{|A| + |B|}$$

$$coverage(tiles) = \sum_{match(a,b,length) \in tiles} length$$

(Perchelt et al., 2000, p. 13).

-Detail the specific optimizations

5.2.2 Preliminary results

When using JPlag to check the similarity of known plagiarized code it had similarity value for the plagiarized m0 code of 91.49& and 90.51% while the none plagiarized code had similarity of around 40%. The results for m4 were somewhat less cleat since they had a similarity value of 84.77& and 76.2%. Which however would still lead to an investigation of the submitted code.

It should also be noted that no files were excluded and that there was no base code passed to JPlag, which explains the high similarity between unrelated programs as all submissions for a milestone use the same base code.

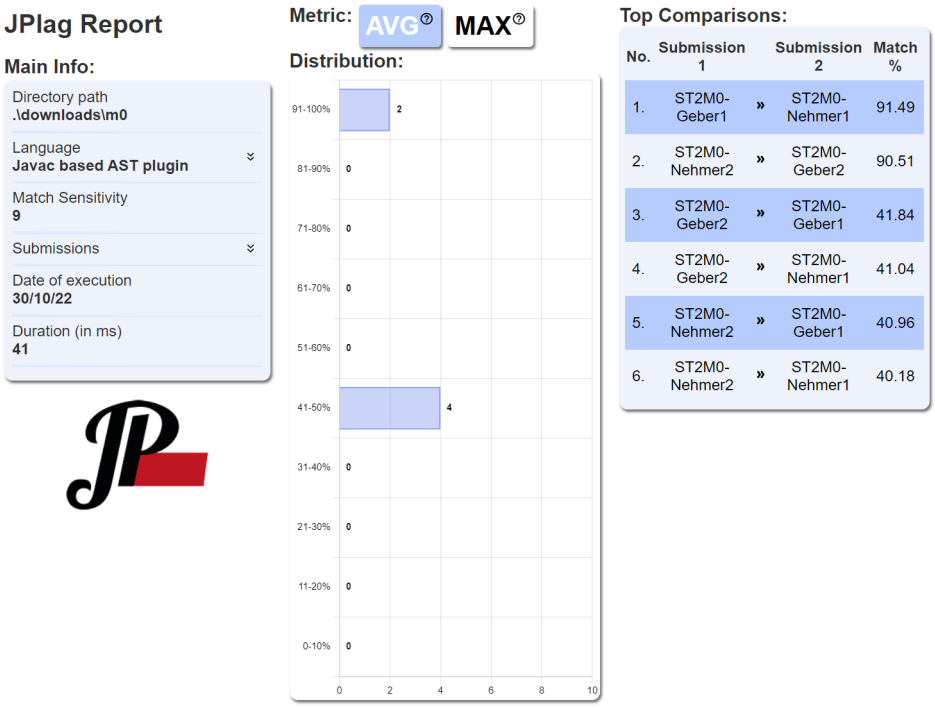

Figure 1 The JPlag overview showing the similarities of m0 code

-need base code for m0 -need base code and another submission for m4

5.2.3 Integration into an automated evaluation pipeline

JPlag probably show the most potential to be used in an automated pipeline as it not only provides a CLI but also a Java API, while also being able to be run locally("JPlag - Detecting Software Plagiarism", 2022) which means that there are no submission limits like in MOSS and that JPlag can be easily integrated into the existing pipeline. However, results are always saved as a Zip file and are view in a web application which can also be run locally. This poses a problem as???

-How does the pipeline work, are files permanently saved or deleted after it finishes evaluation, can they be accessed

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)

-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)

5.3.2 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 is a source code plagiarism detection tool. It aids instructors and graders to detect plagiarism within a group of assignments written in languages such as C, C++, Java, PHP, XML, Python, ECMAScript, Pascal or VHDL (plaintext works too, but is less precise). AC incorporates multiple similarity detection algorithms found in the scientific literature, and allows their results to be visualized graphically." (Freire, 2022a) The first version of "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, 2022a)

2016 AC version 2 was released. In the version 2.0 the developers switched to Maven and Antr4[2]. The developers of AC also moved their git repository to GitHub an internet hosting service for software development and version control.

AC2 lists some advantages over the other plagiarism detection tools on their GitHub page. In this regard, it is

- "local, and does not require sending data to remote servers (not the case of Moss)local, and does not require sending data to remote servers (not the case of Moss)"
- "robust, using Normalized Compression Distance as its main measure-of-similarity, but with the possibility of integrating other, additional measures to gain better pictures of what is going on (JPlag, Moss and Plaggie have hard-coded analyses, mostly based on sub-string matching after tokenization)."
- "heavy on information visualization. AC will not provide "percentage of copy"; instead, it will create graphical representations of the degree of similarity between student submissions within a group, so instructors can build their own explanations regarding what really happened (see here and here for papers on AC's visualizations)."

(Freire, 2022a)

5.4.1 AC2 comparison algorithm

In order for the AC algorithm to check files for equality, it allows adding folders or zip-files into it. Two filters can be applied to these files. The first filter selects the submissions that will be examined, and the second filter is determining which files are compared to each other.

For example student s1 has in his subfolder 'a.c' and student s2 has in his folder 'aa.c'. So the first filter should determine that each submission starts with the letter 's'. The second filter would look for files that ended with the string '.c'.

When all submission and files are correctly grouped together the analysis part can start. "The default analysis (Zip NCD), uses normalized compression distance (NCD) to detect redundancy (= similarity) between submissions. If AC2 knows the programming language of its source files,

[2] "In computer-based language recognition, ANTLR (pronounced antler), or ANother Tool for Language Recognition, is a parser generator that uses LL(*) for parsing. ANTLR is the successor to the Purdue Compiler Construction Tool Set (PCCTS), first developed in 1989, and is under active development."(Wikipedia contributors, 2022)

they will be tokenized prior to comparison. This will allow AC2 to ignore comments, whitespace and exact identifiers, so that simply renaming variables or altering formatting will not affect similarity calculations." (Freire, 2022b)

NCD (normalized compression distance) Todo: NCD beschreiben und sagen wie es in AC2 verwendet wird.

Tokenization If AC2 can recognize the programming language it will use a lexer and a parser to tokenize the text file. "AC2 uses Antlr4[2] grammars to generate lexers (= tokenizers) and parsers for languages." (Freire, 2022b) Therefore it is relatively easy to add support for a new programming language.

5.4.2 Preliminary results

In order to present the advantages of AC2, several projects were compared using the AC2 algorithm. As explained in section 5.4.1 AC2 comparison algorithm, the filters were used to select and group the submissions. After passing the project files, the algorithm analyzes them and creates an output window. The output that AC2 generates is mainly visual based. So Ac "will create graphical representations of the degree of similarity between student submissions within a group". (Freire, 2022a). See figure 2 and figure 3.

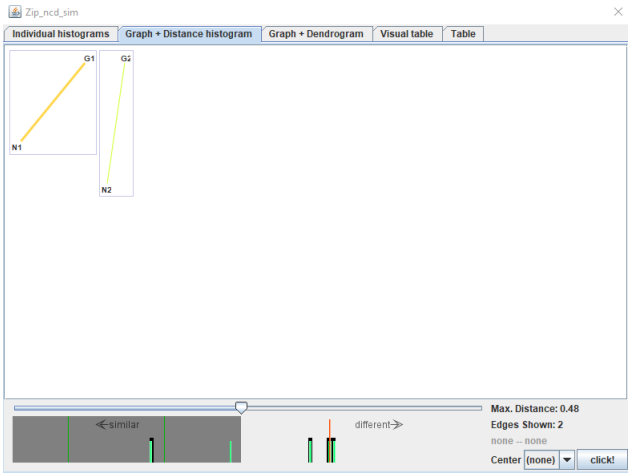

Figure 2 The graphical output of AC2 from milestone [0]

Figure 2 shows milestone 0 and how the different projects have copied from each other. This suggests that project N1 has maybe copied its code form G2 and N2 from G2.

Figure 3 illustrates that G1, N1 and N2 have probably copied from each other, since a correlation exists between the 3 projects. This again illustrates the advantage of AC2 over the other plagiarism detection programs, since it is easier to detect multiple correlations between projects.

Table 2 and table 3 show how far apart the individual projects are. This distinguishes AC2 from other plagiarism detection programs, as they often reflect the similarity in percent to the user.

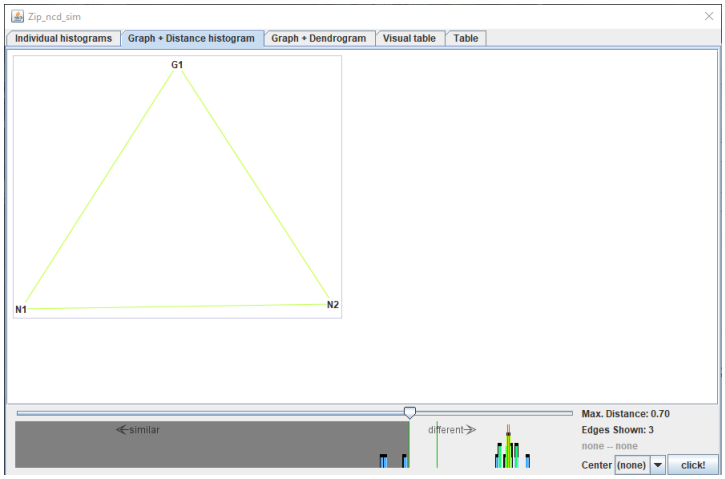

Figure 3 The graphical output of AC2 from milestone [4]

One	The other	Distance
ST2M0-Geber1	ST2M0-Nehmer1	0.2926
ST2M0-Geber2	ST2M0-Nehmer2	0.4609
ST2M0-Geber1	ST2M0-Nehmer2	0.6302
ST2M0-Nehmer2	ST2M0-Nehmer1	0.6705
ST2M0-Geber2	ST2M0-Geber1	0.6718
ST2M0-Geber2	ST2M0-Nehmer1	0.6811

Table 2 [Table of Milestone [0] distance between projects]

6 Conclusion

One	The other	Distance
ST2M4-Geber1	ST2M4-Nehmer2	0.6538
ST2M4-Geber1	ST2M4-Nehmer1	0.6603
ST2M4-Nehmer2	ST2M4-Nehmer1	0.6957
ST2M4-Geber2	ST2M4-Nehmer1	0.6809
ST2M4-Geber1	ST2M4-Geber3	0.8647
ST2M4-Geber3	ST2M4-Nehmer1	0.8657
ST2M4-Geber2	ST2M4-Geber3	0.8772
ST2M4-Geber3	ST2M4-Nehmer2	0.8792
ST2M4-Geber2	ST2M4-Nehmer2	0.8809
ST2M4-Geber2	ST2M4-Geber1	0.8823

Table 3 [Table of Milestone [4] distance between projects]

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*. Retrieved November 12, 2022, from 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. (2022a). *Ac*. https://github.com/manuel-freire/ac2

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

Jplag - detecting software plagiarism. (2022). JPlag. Retrieved November 12, 2022, from https://github.com/jplag/JPlag

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*. JPlag. Retrieved October 30, 2022, from https://github.com/jplag/JPlag/wiki/4.-Adding-New-Languages

Wikipedia contributors. (2022). Antlr — Wikipedia, the free encyclopedia [[Online; accessed 13-November-2022]]. https://en.wikipedia.org/w/index.php?title=ANTLR&oldid=1115716413

Wise, M. J. (1993). *String similarity via greedy string tiling and running karp-rabin matching*. University of Sydney, Department of Computer Science.