Environment. Technology. Resources. Rezekne, Latvia
Proceedings of the 12th International Scientific and Practical Conference. Volume II, 61-64

Modern Algorithms to Identify Plagiarism

Aleksejs Grocevs
Software engineering departmentRiga Technical University
Riga, Latvia
Aleksejs.Grocevs@rtu.lv

Natālija Prokofjeva
Software engineering department Riga Technical University
Riga, Latvia
Natalija.Prokofjeva@rtu.lv

***Abstract*—Nowadays information technology industry is growing extremely fast. To solve business needs, address researcher demand for problem solving a lot of programs are built from scratch. However, not all developers are fair enough to align their products with the corresponding library or another (open source) product licenses, i.e. copyrights are being violated, intentionally or due to familiarization with another source code. To address this issue in past decade multiple plagiarism detection techniques and algorithms were invented. Despite the fact, that many of them are capable of code comparison on meta-level, modern Integrated Development Environments (IDEs) provide convenient way to modify program source code without actual re-writing, preserving the original code workflow and avoiding plagiarism detection. This paper will compare and identify available approaches to apprehend this issue, as well as provide insights for the future this problem mitigation.**

Keywords— plagiarism, algorithm, abstract syntax tree, graph.

I. Introduction

This document oversees the existing plagiarism detection algorithms, the main idea of the article is in comparison of most popular algorithms with insights towards their types and structures they do operate with. During the document the text, structure based and semantic algorithms have been covered.

With the purpose of diving in details of plagiarism detection algorithms the analysis of common structures like AST (abstract syntax tree) and Graph structures have been performed, hence mentioned structures are quite common solutions of building algorithms of mentioned types are being used those days.

The following specific algorithms have been reviewed in the document: The Greedy String Tilling, Kolmogorov complexity algorithm and Fingerprint method. Additionally, the abovementioned approaches were comparative analyzed.

II. ALGORITHM TYPES

Initially the plagiarism detection algorithms were based on quantitative comparison of different program's characteristics like: average string length within the program, number of variables are being defined and are being used, average variable naming length, number of "if else" operators. So the two similar programs would have close or equal number of mentioned constructions. Such algorithms were based on correlation counting principle and were well renown as "attribute counting systems".

The main disfunction of such algorithms happened because the only overall number of certain constructions haven't been analyzed, instead of in-depth analysis of program's logic to be performed. So the similar inclusion of code being plagiarized within the original program couldn't be discovered.[3]

Algorithms shown above provides just the overall, quite approximate result of correlation within the program's syntax. The better result could have been provided by the so cold "control-flow graph" algorithms which are being oriented to compare the changes within the programs structure.

The above stated analysis led the authors towards the comparison of the most modern algorithms from the text based, structure based and semantic algorithms types.

In the first part of this article the major plagiarism detection algorithms will be reviewed with the detailed comparative analysis afterwards. Nowadays there are three main types of plagiarism detection algorithms: those are based on the text, structure or semantics analysis.

Text algorithms are based on the text based program code perception, as an alphabet where one symbol is equal to the proper operator from the programming language, named token. All the text algorithms are based on the idea of such token or token groups comparison within the programming code.

Granularity of the token groups within the code to be inter-compared proportionally equals the effectiveness of the algorithm.

Historically text algorithms were the first widely used ones, their effectiveness is based on investment should be made in recurrent comparison constructions of the interchangeable symbol blocks to assure the most frequent combinations are being checked.[2]

Print ISSN 1691-5402
Online ISSN 2256-070X
http://dx.doi.org/10.17770/etr2019vol2.4058

Structure based algorithms are based on the analysis of the program's structure, where the program logic interpreted in the AST (abstract syntax tree) are being compared. AST (abstract syntax tree) or the Graph of the program's flow are the two names of the program logic tree implementation [1]. Both in AST (abstract syntax tree) or in the Graph the algorithm operates with the program logic as a tree, where such constructions as "if. else." are being perceived as the branches of the tree. Therefore, each program is being perceived as a tree structure what results in quite convenient recurrent comparison between different branches. Important to note that the algorithms using AST (abstract syntax tree) structure are being quite complex in realization and quite advanced supporting technical infrastructure is being required [1].

Semantic algorithms are the algorithms using the Graph relations with two different types of elements, where one element represents the operator type and the connected element represents the type of relation, they are being in. Based on that Graph there is possibility to check each tree limb independently resulting with the quite effective way of comparison between the different code structures.

Each of the main plagiarism detection algorithm types are quite similar and gives the opportunity to analyze the program or it's parts. Such algorithms work well against typical ways of plagiarism, where students do try to change the places of operators within the code.

III. Common structures are being used within the algorithms

Alongside with review of certain algorithms author would like to review common structures they are being based on: AST (abstract syntax tree) and Graph structure.

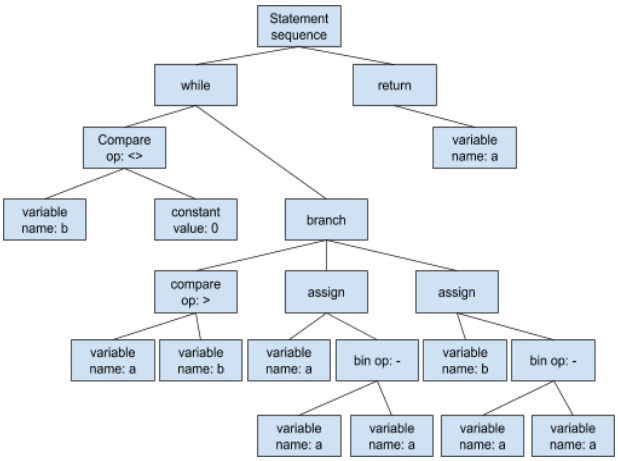

Fig. 1. AST (abstract syntax tree) structure [1]

Figure above (Fig 1.) shows AST (abstract syntax tree) represented in the form of interdependent blocks of code. On the Fig. 1. there could be seen quite different constructions even in such short code example. Comparing part of such tree with the another one there is possibility to successfully fight such simple plagiarism ideas as replacement of the code parts within one program, quite common case in the coursework plagiarism in the high schools.

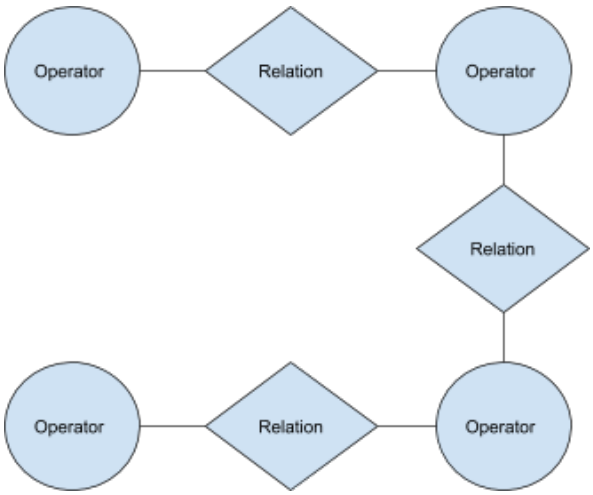

Fig. 2. Graph structure [1]

The Graph structure provides the different idea to the Structure based and Semantic plagiarism detection algorithms types.

Within the Graph structure program code is been transferred to the graph where vertex and edges are being defined by the operators and their relations.[9]

Mentioned structures are being widely used within the various specific plagiarism detection algorithms.

Use of this structure is useful in searching the core functions, blocks in code within the different programs, may result in finding plagiarism even on idea level within students solving one complex problem.

IV. Specific Algorithm Realizations

"The Greedy String Tiling" is the subset of the "Text" plagiarism detection algorithm type. This algorithm interprets to programs to be compared as two lines of text based on the unified alphabet (usually named the token variety) and results with the one line of text - the variety of the unique tokens.

The main two ideas of the comparison are: algorithm does not count in coincidences with the too small number of same tokens. The major findings to count in examining the result are the longer coincidences even if summarized length of the small coincidences is bigger. Such logic could be easily explained, so there could be same parts of code constructions to be used by two different developers, but they are many in numbers and short, on the other hand the long lines of same tokens is the clear sign of possible plagiarism within the programmed solution. [3]

Therefore, not counting in the smaller coincidences just protects the algorithm from the possibility of random matches.

"Kolmogorov complexity algorithm" : "K (s) = min{|p|, U(p) = s}", the algorithmic complexity *K (s)* of a string is the length of the shortest program *p* that produces *s* running on a universal Turing machine *U*. Algorithm counts the length of non-matched tokens between the matching sequences. [5]

This method is being called (an information-based sequence distance). The main advantage of this method

is its versatility, since it will understand the appearance of matching element based on every possible match principle, what makes it much more universal than other plagiarism detection algorithms. So, as the "Kolmogorov complexity algorithm" states, the smaller is the average length of non-matched tokens the higher is the probability of plagiarism within two programs.

"Fingerprint method" - specific realization of "Text" type plagiarism detection algorithm. In "Fingerprint" method there are stored vocabulary of "token combinations" are being named "Fingerprints".

So, in this method the search and comparison are not being handled on token to token principle, but on the other hand by searching specific token, or token combination in codes are being compared.[8]

The "Fingerprint" is much more convenient for various interdependent searches, where search is being handled against the "Fingerprint" base, could be stored in format of simple DB solution. [4]

Usually the "Fingerprint" plagiarism detection algorithm is being realized in following steps:

1.) use of hashing principle for the sequence of tokens (program);

2.) received subset of hash-codes to be put within the hash table;

3.) Comparison of hash tables with the "Fingerprint base" the subsets with the higher risk for plagiarism will be defined.

TABLE I. Plagiarism detection algorithm comparison

	Algorithm name	*Pros*	*Cons*
1.	Greedy String Tiling	Effective in comparing one-to-one program codes	Is not effective enough to use this type of text algorithms with DB of code samples
2.	Kolmogorov complexity algorithm	Versatility, since it will understand the appearance of matching element based on every possible match principle, what makes it much more universal than other plagiarism detection algorithms	Is not effective enough to use this type of text algorithms with DB of code sample
3.	Fingerprint method	"Fingerprint" is much more convenient for various interdependent searches, where search is being handled against the "Fingerprint base", could be stored in format of simple DB solution.	Infrastructure with quite high-performance characteristics is being required to support this solution

Comparison of the "Greedy String Tiling", "Kolmogorov complexity algorithm" and "Fingerprint method" algorithms has shown the different specifics of the, above mentioned algorithms, there are different situations and circumstances they could be applicable and the most effective in.

Each of the reviewed plagiarism detection algorithms could be used in the high school environment, but the "Fingerprint method" algorithm looks to be the most useful one while checking the student's developed program against the DB of the program samples from the previous courses. On the other hand, "Kolmogorov" algorithm should be chosen for the more complicated, choice situations.

V. Other Conclusions

Concluding the article following findings should be highlighted:

Major plagiarism detection algorithm including their specific realizations can handle the plagiarism problem effectively, on the other hand there are proper pros and cons against choosing the one or another of them.

For example, no other algorithm except the "Fingerprint" algorithm can work effectively with the big number of code examples to perform the comparison with. If this type of solution is being required the "Fingerprint" algorithm is the most effective one, since it's speed of work is directly dependent from the number of code examples to compare with.

There are big difference in terms of quality and quantity within the performed analysis, since the "text" type plagiarism detection algorithm are not so precise in terms of detecting the certain logical constructions in comparison with "structure-based algorithms" or "semantic algorithms".

On the other hand, "structure-based algorithms" which use AST (abstract syntax tree) or "semantic algorithms" using the Graph structure do provide the much clearer picture of logical structures are being used within the compared programs code.

Within the big number of existing plagiarism detection algorithms there are the most effective ones like: "Fingerprint" algorithm and algorithms based on the AST (abstract syntax tree), since they are the only ones which can effective and automate the plagiarism check using the DB of comparative examples in the process.

Mentioned algorithms go assure the expected speed based on hash-tables usage in comparison process. This is the main reason of these algorithms' usage in the high schools, where are lots of student's generated code examples on a quite limited number of topics within the disciplines.

Here is very important to understand that during each year students of different courses do repeatedly work on quite similar problems within their courses, since there is no real possibility to significantly differ the course content as frequent as new groups of students do arrive, on the yearly basis at least.

This is the main reason for using "Fingerprint" and AST (abstract syntax tree) based algorithms against the Database of the student works submitted from the previous years.

References

[1] Baxter I., Clone Detection Using Abstract Syntax Trees. ICSM. 2008.

[2] Mishne G., Source Code Retrieval using Conceptual Similarity, RIAO, Vaucluse, 2004.

[3] Chen X., Francia B., Shared Information and Program Plagiarism Detection, IEEE Information Theory. 2004.

[4] A.Grocevs, N.Prokofjeva, "Modern programming assignment verification, testing and plagiarism detection approaches." Proceedings of the IVUS International Conference on Information Technology, pp. 61-64, 2017.

[5] Prechelt L, Malpohl G, Philipsen M. Finding plagiarism among a set of programs w/th JPlag. J. UCS. 2002 Nov 28;8(11): 1016.

[6] Whale G. Plague: plagiarism detection using program structure. School of Electrical Engineering and Computer Science, University of New South Wales, 1988.

[7] Joy MS, Sinclair JE, Boyatt R, Yan JK, Cosma G. Student perspectives on source-code plagiarism, International Journal for Educational Integrity. 2013;9(1):3-19.

[8] A. Parker and J. Hamblen. Computer algorithms for plagiarism detection. IEEE Transactions on Education, 32(2):94099, 1989.

[9] Fintana FA, Mangiacavalli M, Pochiero D, Zanoni M. On experimenting refactoring tools to remove code smells. InScentific Workshop Proceedings of the XP2015 May 25 (p. 7). ACM