Similarities in Source Codes

Marek ROŠTÁR*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
rostarmarek@gmail.com

Abstract. With an increasing popularity of different programming languages, a problem of finding similar parts of source codes across different programming languages is rising. Finding such parts of codes can be useful for improving source code quality or identifying potential plagiarism. In current day and age there are multiple ways of identifying similarities in the source code or text documents. Most known are text/token based methods, which can be strengthened with stronger preprocessing of given source codes. In this work we focus mainly on identifying similarities using abstract syntax tree. We also explore the possibilities of applying different levels of preprocessing of source code and its benefits from the performance point of view.

1 Introduction and related work

In current day and age with the development of software there is an increase of problems concerning plagiarism, but also it is quite common to see repeated use of source code parts. These two issues are the main reasons to explore the task of source code comparison, since it can detect most of plagiarism attempts and also highlight which parts of our source code are we using repeatedly. Such parts we may consider to turn into some sort of library or plugin, to improve our source code quality.

Plagiarism is these days one of the relevant problems of the academia, affecting not only text documents but also most of intellectual property including source codes. It is common that students inspire themselves with some work they found on the Web.

In this work we focus on detecting plagiarism in source code specifically (considering its special features in comparison to the standard text documents). Methods used to detect plagiarism in source code differ from methods used to detect plagiarism in text documents [1, 2], since the text in source code does not carry only meaning but some sort of function as well.

Plagiarists try to deceive anyone viewing their work with a multitude of different plagiarism attacks, which for plagiarism in source codes, the basic ones are as follows: altering comments in source code, altering whitespaces present in source code, altering names of variables, altering the order of parts of the source code, altering algebraic expressions in source code, translating source code into another programming language and extracting parts of source code.

---

* Bachelor degree study programme in field: Informatics
Supervisor: Dr. Michal Kompan, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

While altering comments in source code, the plagiarist leaves the whole structure of the source code and its functionality intact, but alters the visual part of the source code by either adding, removing or changing the content of the comments. Comments usually bear no functionality in the source code and usually are used to explain the source code.

Altering whitespaces is very similar to altering comments, but it is worth to note that this type of attack is not always applicable, since there are some programming languages in which whitespaces bear functionality. One of such programming languages is Python, in which whitespaces are used to denote blocks. Altering the names of variables will as well change the visual structure of the source code while not interfering with its functionality.

While altering the order of parts of the source code the plagiarist will change the order of separate blocks of source code. This does not interfere with the functionality of the source code and changes it visual structure. It is worth noting that in some languages order of functions matters. In these languages if some function will be using another one that is in original source code written sooner and in the altered source code the order is reversed, the function used in the other function has to be declared before it is used.

Altering algebraic expression takes advantage of commutative and distributive properties of certain algebraic operations. There is also the possibility of changing some parts of expressions that compare two numbers in a way that it will have the same logic but it will be visually different.

Translating source code to different programming language is not always a plagiarism attack, since sometimes the language to which is the source code being translated to does not have all the tools and structures necessary being used in the original programming language. In our work we focus on such sort of translation, which does not include any additional work from the person that translated the source code.

Extracting parts of the source code means that the plagiarist will remove some part of the source code and put it into a separate file. He will then include in the source code and call the functionality removed from original source code from the external file, and thus changing the visual structure of the source code while not interfering with its functionality.

The comparison of different plagiarism detection methods over various plagiarism attacks was discussed in [2], where the authors tested performance of different plagiarism detection methods fare against different types of plagiarism attacks on small scale source codes.

In [1], different types of plagiarism are discussed as well as methods used to detect plagiarism in general not only in source codes. This work goes over the differences in the literal plagiarism and intelligent plagiarism.

There are some works that explore combining different methods of detecting plagiarism [7, 8] to create so-called hybrid methods of plagiarism detection. In these specific works combination of tree based methods with either semantic based methods or token methods is applied.

Park et al. [5] discusses the application of source code abstraction for large scale source codes to improve the efficiency and accuracy of detecting similarities in those source codes. It proposes an automated abstraction method, which gets rid of large part of the source code, which is deemed as unimportant. Chillowicz et al. [3], improves method based on abstract syntax tree by creating fingerprints of compared documents and comparing them.

2 Plagiarism detection methods

There are several views used to detect plagiarism hierarchies. In this work we will differentiate four basic approaches [6]:

- Text based methods
- Token based methods
- Tree based methods
- Semantic based methods

Another example of method division into groups is where we divide methods into two groups depending on the fact if they take into consideration the meaning of the part they are comparing or not. In this view, we would put only text based methods in the group that does not take into consideration the meaning of source code.

2.1 Text based methods

Text based methods are generally (in the context of source code plagiarism) the quickest of all of the types of plagiarism detection methods. On the contrary, these are also most vulnerable to plagiarism attacks. Text based methods go step by step through the source code while comparing strings in them. Such methods can return various metric depending on which algorithm belonging to this type is used.

One of such algorithm is LCS algorithm [8], which returns the longest common subsequence between both source codes that are being compared through this algorithm.

The advantages of using text-based methods are that they are fast, they don't require complicated data structures and are generally easier to understand. The disadvantages are that they are vulnerable to attacks, which means they often need to have the source codes heavily pre-processed to counter these plagiarism attacks, and they do not take into consideration the meaning and context of parts of the source code.

2.2 Token based methods

Token based methods are based on the serialization of parts of the source code into tokens which in the next phase replace the actual source code. This tokenization is done by the mean of lexical analysis of the source codes before comparing them, and selecting important parts which are then turned into tokens.

Lexical analysis can be done for example by creating a finite-state machine, to which we will define rules containing regular expressions. Using such lexical analysis, it is easy to see that through the rules we can select parts of the source code that we find important and we can discard the rest. After we have selected the important parts, we compare these altered source codes using text based methods.

Advantages of token based methods are that they are more resistant plagiarism attacks than text based methods, while keeping close in their computation time to them. Disadvantage compared to text-based methods is slightly slower computation time and the addition of some data structures in lexical analysis.

2.3 Tree based methods

Tree based methods are based on converting source code into a data structure of tree and after the conversion comparing the trees instead of source codes themselves.

These methods usually create trees by doing lexical analysis, similar to token based methods. But afterwards they use the source code after the lexical analysis in a syntactical analysis, that takes blocks of source code and transforms them into nodes adding information about them, like position in source code. After the transformation is done the hash function is used to compute hash values of given nodes. Then the comparison of the trees on a node to node basis is performed. Nowadays there are parser we can use to do the lexical and syntactical analysis, instead of having them as a separate part.

Advantages of this method are that it is resistant to plagiarism attacks and it can find behavioral changes, which can be used for finding malicious source code [4]. Its disadvantages are time consumption and addition of new data structures.

2.4 Semantic based methods

Semantic based methods do take into account not only the meaning of the source code, but also the context of parts of the source code. Thanks to this it can handle polysemy and synonymy. One of such semantic based methods uses program dependency graphs, in a similar way as the tree based methods use trees. Source codes are converted into graphs and then to compute similarity metric, we need to convert the graphs into adjacency matrices. When the matrices differ in size the smaller one extended with rows and columns of zeros to have the same size as the bigger one. Then we need to convert the matrices into vector and by comparing those, we get the similarity metric.

3 Source code abstraction

When we are working with source code we can remove for us unimportant parts and thus create an abstract source code. We use this sort of pre-processing often when we compare source code to find similarities in given source codes. There are multiple levels of abstraction, and we can determine how strong abstraction we need based on how big is the source code [5]. For small and medium scale source codes we often use only low levels of abstraction, but for large scale source codes we often use strong levels of abstraction. Low levels of abstraction commonly remove comments, unifies whitespaces, unless it is written in a programming language in which whitespaces denote blocks. Stronger levels of abstraction can alter different things, such as removing declaration of variables, values of strings or algebraic expressions from the source code.

Thanks to abstraction, we can vastly reduce the volume of the source code and with that speed up the process of comparing the source code. It can also remove some false positive cases, but it can also create other false positive, by removing important parts of the source code that we did not think were important.

4 Proposed solution

In this work we propose method of detecting plagiarism using abstract syntax tree so let's go now go into detail about this method. Our method can be divided into 5 steps: construction of the abstract syntax tree, computing hash values of nodes of the abstract syntax tree, adding information about node, comparing the trees and computation of similarity metric.

To create abstract syntax tree of the source code we use the parser. Thanks to this step we get rid of some unimportant parts of the source code. After the tree is completed we compute hash values of the nodes, where for leaves the value will be the hash of their content and for other nodes the hash value is the sum of the hash values of its children and its hash value. This eliminates plagiarism attacks that alter the order of source code, because even if we swap two children of a node, value in the node will remain unchanged. When we have computed hash values of the nodes of the tree we can add additional information about notes such as number of children nodes or type of node, which speed up the comparison process. This step is optional. Then we can compare the trees on a node to node basis and after we finish we compute similarity metric (e.g., cosine similarity, Jaccard index).

Proposed approach is designed to include different levels of abstraction on source code before we convert the source code into the tree. There are in total three levels of abstraction, ranging from removing only comments and unifying white spaces to the highest level, in which we replace strings values, remove variable declarations, unify the names of variables and greatly reduce the volume of the source code.

4.1 Evaluation

In the experimentation phase we will compare the computation time for comparison between the different levels of abstraction and performance of proposed approach (in the context of detected

similarities and codes). In order to compare our results to the state-of-the-art solutions we also compare our method with results of MOSS, which uses token based method. For the comparison we will use dataset obtained from bachelor course Artificial intelligence on our faculty and the SSID dataset[1].

5 Conclusions

The problem of plagiarism is on the rise in the academia. It affects not only text documents but it spread its influence over different intellectual property such as source codes. Plagiarism of source codes differ from plagiarism of text documents, for example in attacks which plagiarist takes to obscure the fact that they have stolen given intellectual property, which we have discussed.

Currently there are multiple methods of detecting plagiarism in source codes. We have divided these methods into four groups of text based, token based, tree based and semantic based methods of detecting plagiarism. These methods take different time to compare source codes and are differently vulnerable to plagiarism attacks.

We can reduce amount of the time needed to compare source code by using abstraction on them to remove unimportant parts of the source code at the risk that we may lose some information when removing parts of the source code.

In our experiment we compare if using stronger abstraction on source code before sending it into abstract syntax tree and comparing the trees is viable. We use three different levels of abstraction and we compare the performance of our algorithm to a free source code comparison tool, which used token based method to detect plagiarism.

References

[1] Alzahrani, S. M., Salim, N., & Abraham, A.: Understanding plagiarism linguistic patterns, textual features, and detection methods. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, (2012), vol. 42, no. 2, pp. 133–149.

[2] Beth, B.: A Comparison of Similarity Techniques for Detecting Source Code Plagiarism. (2014).

[3] Chilowicz, M., Duris, E., & Roussel, G. Syntax tree fingerprinting for source code similarity detection. *IEEE Int. Conference on Program Comprehension*, (2009), pp. 243–247.

[4] Neamtiu, I., Foster, J. S., & Hicks, M.: Understanding source code evolution using abstract syntax tree matching. *ACM SIGSOFT Software Engineering Notes*, (2005), vol. 30, no.4, 1.

[5] Park, S., Ko, S., Choi, J. J., Han, H., & Cho, S.-J.: Detecting Source Code Similarity Using Code Abstraction Categories and Subject Descriptors. *Proc. of the 7th Int. Conf, on Ubiquitous Information Management and Communication - ICUIMC '13*, (2013), pp 1–9.

[6] Tao, G., Guowei, D., Hu, Q., & Baojiang, C.: Improved Plagiarism Detection Algorithm Based on Abstract Syntax Tree. *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth Int. Conf. on*, (2013), pp. 714–719.

[7] Wu, S., Hao, Y., Gao, X., Cui, B., & Bian, C.: Homology detection based on abstract syntax tree combined simple semantics analysis. *Proceedings - 2010 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT 2010*, (2010), pp. 410–414.

[8] Zhang, Y., Gao, X., Bian, C., Ma, D., & Cui, B.: Homologous detection based on text, Token and abstract syntax tree comparison. *Proc. 2010 IEEE Int. Conf. on Information Theory and Information Security, ICITIS 2010*, (2010), pp. 70–75.

[1] http://wing.comp.nus.edu.sg/downloads/SSID/