Jestr

Journal of Engineering Science and Technology Review 9 (1) (2016) 8-12

JOURNAL OF
Engineering Science and Technology Review

Research Article

www.jestr.org

Discovering Clusters of Plagiarism in Students' Source Codes

L. Moussiades

Computer and Informatics Engineering Department,Eastern Macedonia & Thrace Institute of Technology
Kavala, St. Loukas 65404 Kavala, Greece

Received 30 September 2015; Accepted 2 March 2016

Abstract

Plagiarism in students' source codes constitutes an important drawback for the educational process. In addition, plagiarism detection in source codes is time consuming and tiresome task. Therefore, many approaches for plagiarism detection have been proposed. Most of the aforementioned approaches receive as input a set of source files and calculate a similarity between each pair of the input set. However, the tutor often needs to detect the clusters of plagiarism, i.e. clusters of students' assignments such as all assignments in a cluster derive from a common original. In this paper, we propose a novel plagiarism detection algorithm that receives as input a set of source codes and calculates the clusters of plagiarism. Experimental results show the efficiency of our approach and encourage us to further research.

Keywords: plagiarism detection, clustering, similarity, programming assignments, keyword based programming language

1. Introduction

It is well known that plagiarism in students' assignments is quite widespread [1]. Programming assignments do not constitute an exception. Taking into consideration that the teaching of programming languages is typically project based, we can see that plagiarism in students' programming assignments is a major menace to the educational procedure [2]. Therefore, plagiarism detection is necessary for the benefit of the educational process. However, detecting plagiarism manually is a time consuming and tedious task, as it requires the comparison of submitted source codes in pairs. Moreover, detecting plagiarism manually becomes even more difficult as students tend to modify the plagiarized code in order to avoid detection. Thus, software systems oriented towards plagiarism detection in source codes appeared in mid-70s. These early systems, known as attribute counting systems [3, 4, 5, 6, 7], adopt various software metrics, e.g. the number of code lines, the number of declared variables, the number of control or loop structures, etc., to calculate the similarity between two source codes. In the mid-90s, structure-metric systems show up. They calculate a similarity between sources based on structure comparison [8, 9, 10]. Systems that are not considered attribute counting nor structure metric are also available [1, 11, 12, 13, 14, 15, 16]. We remark that JPlag [10], which is a structure-metric system, is probably the most famous one.

All of the aforementioned plagiarism detection systems receive as input a set of source codes and produce a set of pairwise similarities. However, the result of plagiarism detection formed in terms of pairwise similarities is not always convenient. From tutor's perspective, what is actually required is the clusters of plagiarism, i.e. clusters of students' assignments where all assignments in the same cluster derive from a common original. The detection of clusters of plagiarism allows the tutor to confront plagiarism much more efficiently. For example, a tutor may decide to invite all members of such a cluster in order to admonish them. Besides, plagiarism detection becomes clearer when the whole cluster is detected, and students tend to accept easier their fault. Finally, clusters of plagiarism may be used by the tutor in order to set learning groups and promote cooperation between students.

In this paper, we propose a plagiarism detection algorithm for source codes, which accepts as input a set of programming assignments and outputs the clusters of plagiarism. Our approach supports every keyword-based language. Experimentally, we apply our approach on a set of real data and we show that it is efficient and compares favourably to existing plagiarism detection systems.

The rest of the paper is structured as follows. Section 2 presents types of typical modifications of the plagiarized code that students tend to do in order to avoid detection. In section 3, we discuss the representation of source codes that we propose. An algorithm for the detection of clusters of plagiarism that may exist within an input set of source codes is proposed in section 4. Experimentation with a real data set is presented in section 5. Finally, in section 6, conclusions are reported and further research topics are highlighted.

2. Types of attacks

Plagiarism detection in students' source codes becomes too difficult task because students tend to modify the plagiarized code in order to avoid detection. Several types of such

* E-mail address: lmous@teiemt.gr
ISSN: 1791-2377

modifications, known as types of attacks, have been highlighted by [10]. Below, we present the types of attacks classified into five categories depending on whether and how our approach confronts them.

A. Modification of code formatting; insertion, modification, or deletion of comments; change of identifiers names, modifications of the text in I/O statements, modification of program output or of its formatting, modification of constant values, reordering within blocks of variable declarations, Modifications of scope, e.g. extension of temporary variables scope by declaring them in surrounding blocks; statement reordering in absence of data dependences; insertion, modification, or deletion of modifiers, modification of control structures, e.g. a "for loop" is replaced by a "while loop", and replacement of mathematical operators without affecting the surrounding expression, e.g. the expression ++i is replaced by expression i+=1.
B. Global reordering of variables and blocks. It applies to global identifiers only.
C. Insertion of dead blocks or dead identifiers, i.e. blocks of code or identifiers, which have been declared but not used.
D. Split or merge of variables declarations, temporary variables and/or sub expressions addition or removal.
E. Modification of data structures and/or structural redesign of code.

Our approach confronts attacks of type A based on the representation of source codes that we employ. Attacks of type B are confronted based on the similarity measure that we use. Attacks of type C may be easily confronted by checking if a declared identifier is also invoked or not. Attacks of type D are confronted partially by our approach. Finally, modifications of type E are not considered attacks as modifications of data structures and structural redesign of code implies deep understanding of the code. We remark that our approach or any other plagiarism detection software does not confront modifications of type E. More details about the confrontation of attacks are given below in this paper.

3. Representation of source code

The Plagiarism Detection Algorithm that we propose, or PDA for short is based on the representation of source code as sets of indexed keywords. We remark that these sets are quite similar but not identical to indexed sets of weighted substitute and user-defined keywords defined in [2].

For a keyword based programming language L, let R(L) represents the set of reserved words of L. Assume that user additions and/or deletions of suitable elements in R(L) produce a set named U(L). An example of such an operation is the addition of the keyword String, which although is part of a programming language, typically, it is not a reserved word, e.g. C++ and Java. Finally, assume that for each element in U(L), a substitute value may be defined resulting in a set of extended keywords E(L). An example of such a substitution would be the substitution of reserved words "for" and "while" with the keyword "loop".

Definition 1: If b represents a block of source code in programming language L, then its set of indexed keywords, symbolically K(b), contains an indexed reference for each word in b that is also included in E(L).

For example, if b is a C++ block of code and it contains 3 integer declarations (language keyword is "int") and reserved word "int" has not been deleted from E(L) then $\{\text{int}1, \text{int}2, \text{int}3\} \subseteq \text{K}(\text{b})$. Additional example: If the word "String" is included in E(L) and b contains 2 String declarations then $\{\text{String}1, \text{String}2\} \subseteq \text{K}(\text{b})$. Finally, if "loop" is the substitute value for reserved words "for" and "while" and b contains 2 "for" and 2 "while" then $\{\text{loop}1, \text{loop}2, \text{loop}3, \text{loop}4\} \subseteq \text{K (b)}$.

The representation of source code as sets of indexed keywords facilitates the confrontation of all attacks of type A. In Fig. 1, function f2 has been produced by applying attacks of type A to function f1.

More precisely, the following types of attacks have been applied to function f1: Change of identifiers names, reordering of variables declarations, statement reordering, modification of control structure, replacement of mathematical operators, and split of variable declarations. However, these types of attacks can be confronted by the representation we propose and the adoption of a suitable set of extended keywords. For example, if set

E (C++)= {bool, string, int, [,], {, loop, <,
++, =, false, if, >, true, &&, }, -, return}

is used with word "loop" to substitute both "for" and "while", then both functions are represented by same set of indexed keywords:

$$\text{k}(\text{f}1) = \text{k}(\text{f}2) =$$
$$\{\text{bool}1, \text{string}1, \text{int}1, [1,]1, \text{int}2, \text{bool}2, \{1, \text{loop}1, \text{int}3, <$$
$$1, + + 1, [3,]4, = 1, \text{bool}3, \text{false}1, \text{if}1, > 1, =$$
$$2, \text{true}1, \text{if}2, \&\&1, \{2, -1, = 3, = 4, \ \text{false}2, \ \}1,$$
$$\text{return}1, \ \}2 \ \}.$$

Therefore, f1 is 100% similar to f2 independently of the similarity measure used. Actually, the proposed representation facilitates the confrontation of all attacks of category A.

4. Description of PDA

Our first step towards the detection of plagiarism is the comparison between assignment units. More precisely, we have to calculate a similarity measure for each pair of assignment units. Note that each assignment unit consists of several global blocks, i.e. a block of code that is not included in any other block; thus, all global variables, global constants; global functions and global classes are considered global blocks. We represent an assignment unit as the set of its global blocks where each block is represented by its set of indexed keywords. More formally, if A represents an assignment unit and b a global block, then A is represented by set $\text{P (A)} = \{\text{K(b)} \ \forall \ \text{b} \in \text{A}\}$.

Since we represent an assignment unit as a set, a similarity measure for sets can be used to signify assignments that are suspect for plagiarism. Several such measures have been proposed in literature. Most known similarity measures for sets include Jaccard Coefficient [17] as well as Purity and Entropy [18]. However, Jaccard Coefficient has certain disadvantages [19] as well as Purity and Entropy have. More precisely, although two sets may not be equal, they may appear maximum Purity. Also, Entropy between two sets may be minimum although the

sets may differ. Therefore, we adapt a similarity measure proposed in [21] where it is shown that it overcomes disadvantages of Jaccard coefficient and those of Purity and Entropy. Given a pair of assignment units, A, B, a similarity measure $\mu(A, B)$ is given by

$$\mu(A,B) = \frac{\frac{\sum_{b\in P} \max_{\beta\in R} \frac{|b \cap \beta|}{|b \cup \beta|}}{|P|} + \frac{\sum_{b\in R} \max_{\beta\in P} \frac{|b \cap \beta|}{|b \cup \beta|}}{|R|}}{2}$$

where b and β represent global blocks, $P = \{k(b)\ \forall\ b \in A\}$ and $R = \{k(b)\ \forall\ b \in B\}$.

It is trivial to prove that the following conditions hold true:

$$\mu(A, B) = 1 \Leftrightarrow A = B$$

$$\mu(A, B) = \mu(B, A)$$

We remark that the aforementioned conditions (i) and d (ii) are essential for every function that should correspond to the common sense of similarity. Also note that μ is unaffected by the blocks' order; therefore, reordering of global blocks, i.e. attacks of type B, are confronted based on the nature of similarity measure that we propose.
Algorithm PDA is presented below.

Algorithm PDA

Input: A set of assignment units $\{A_k\}$ in a language L,
A set of extended keywords E(L),
A cut-off criterion value c

Output: The clusters of plagiarism in $\{A_k\}$

1. $V = S = \{\}$
2. Create graph $G = \{V, S\}$
3. For each pair A, B in $\{A_k\}$
   a. If $(A \notin V\}$ add A to V
   b. If $(B \notin V\}$ add B to V
   c. If $(\mu(A, B) \geq c)$
      add $\big(A, B, \mu(A, B)\big)$ to S
4. return clustering*(G)*

The input of PDA includes sets $\{A_k\}$ and E(L), which have been explained already. The purpose of cut-off criterion is to eliminate relations between assignment units when the pairwise similarity is less than the cut-off value. We remark that a cut-off criterion value is used in most plagiarism detection systems. The idea of PDA is first to calculate the similarities between pairs of assignments and then formulate an appropriate graph and find the clusters of plagiarism by clustering the graph. Therefore, in steps 1 and 2 of PDA we create an empty graph G. Then, in step 3, we add vertices and links to graph G, formulating a weighted graph G, whose each vertex represents an assignment unit and a weighted link connecting two vertices represents the similarity value between the corresponding assignment units. Finally, in step 4, a graph-clustering algorithm may be used to reveal the clusters of plagiarism. A lot of graph clustering algorithms have been proposed [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 18, 20]. However, most of them require a pre-specified number of clusters as input. In our problem, such a number is not known, i.e. the number of clusters of plagiarism cannot be pre-specified. Therefore, we need a clustering algorithm that automatically calculates an optimum number of clusters based on the structure of the input graph. Such algorithms are the well-known modularity clustering [28, 29, 25, 26, 27]. We have experimented with all four aforementioned clustering algorithms.

5. Experimentation

Here, we report results of experiments that we have conducted on a data set consisting of 124 student assignments on a project in C++. An experienced human reviewer has evaluated the data set. In Fig. 1, each disconnected component represents a cluster of plagiarism and each vertex represents an assignment that belongs to a cluster of plagiarism. Therefore, according to human reviewer, a total of 14 assignments are plagiarism transcripts and they form 4 clusters of plagiarism.

As we have already mentioned, algorithm PDA sets up a graph G such as the set of vertices represent the students' assignments and the set of links represent the similarity between two assignments. A prerequisite for the computed clusters of plagiarism to be correct is that G has been set up based on reliable data. The reliability of G may be expressed in terms of Precision and Recall [33]. Precision denotes the number of assignments that are actually plagiarized transcripts divided by the number of assignments that PDA considers as plagiarized transcripts. Recall denotes the number of assignments that the algorithm considers as plagiarized transcripts divided by the number of the actual plagiarized transcripts. In Fig. 3, we present precision and recall of PDA for values of cut-off criterion from 1 to 0.4. For comparison, in Fig. 4, we present corresponding results of the well-known plagiarism software called JPlag.

In both Fig. 3 and Fig. 4, the horizontal axis represents values of the cut-off criterion. If cut-off value is 1, i.e. if only pairs of assignments with a similarity of 1 are considered plagiarized pairs, both PDA and JPlag appear absolutely precise. However, the recall is quite low, which reveals that a lot of plagiarized transcripts could be found for lower cut-off values. Indeed as the cut-off value decreases the recall increases. As it can be seen in Fig. 4, JPlag best performance is attained at cut-off value equal to 0.8, where precision equals to 1 and recall equals to 0.6. For higher cut-off values, precision lowers faster than recall grows. Down to cut-off value equal to 0.8 JPlag performs better than PDA as a comparison between Fig. 3 and Fig. 4 shows. However, for lower cut-off values, PDA performs better. More precisely, at cut-off value equal to 0.55, PDA appears precision and recall equal to 1, i.e. reveals all of the actual plagiarized transcripts. Therefore, we consider that the overall performance of PDA is better than the performance of JPlag for this dataset. Moreover, PDA produces the correct graph G for cut-off value equal to 0.55.

Regarding the clustering, as we have already noted, we have tested 4 algorithms, namely the ICR, the ILR, the modularity clustering and MajorClust. At cut-off value equal to 0.55, algorithms ICR, ILR and modularity clustering reveals the actual clusters of plagiarism, i.e. the clusters as they detected by the human reviewer (Fig. 2). We remark that the particular clustering problem is a relatively easy one since graph G (the graph to be clustered) is weighted and the clusters of plagiarism are not connected components of G. MajorClust fails to reveal the clusters of plagiarism because as we found the algorithm gets confused when the input graph contains not connected components.

Of course, the aforementioned result has been achieved for the optimum value of cut-off criterion. However, the calculation of the optimum cut-off value requires an exhaustive search from the human reviewer. Here, we have performed an exhaustive search in order to evaluate PDA. However, in a typical scenario we do not want to perform an exhaustive search. On the contrary, our purpose is to eliminate the human reviewer work. In the experimental data set the optimum cut-off value is 0.55. Based on this observation, we propose the use of the value 0.5 for the cut-off criterion. We assume here that there is no plagiarism transcript with similarity lower than 0.5. We remark that although this assumption is true for the experimental data set, it should be checked using sampling for other datasets. Therefore, a scenario for the use of PDA is to run it for cut-off value near to 0.5 and then check the results for assignments that have been placed in a cluster of plagiarism although they are not connected strongly to the cluster. For example, the clusters of plagiarism that have been detected by PDA for cut-off value equal to 0.5 are presented in Fig. 5. Continues lines in this figure represent similarity between assignments in the same cluster, while dotted lines represent similarity between assignments belonging to different clusters. Moreover, black vertices represent assignments, which actually belong to a cluster of plagiarism while white vertices represent assignments, which have been placed by PDA to a cluster of plagiarism incorrectly. A human reviewer can easily detect the "white" assignments as they are connected with only few assignments in their cluster and with low similarity, i.e. similarity < 0.55.

6. Conclusions and further research

In this paper, we propose algorithm PDA that receives as input a set of computer programs and detects possible clusters of plagiarism within the input set. PDA consists of two parts. The first part calculates similarities between pairs of programs. For this part we have proposed a novel algorithm, which appears to perform favourably compared to the well-known JPlag algorithm. For the second part, i.e. the computation of clustering, algorithms ICR, ILR and Modularity clustering appears to perform efficiently. PDA suites very good the need of the tutor who teaches a keyword based programming language and wishes to check the students' assignments for clusters of plagiarism. The visualization of the clusters of plagiarism helps the human reviewer to examine the correctness of the produced clusters. What is missing is a visualization that will reveal the similarity between two sources. However, we believe that the similarity measure we use is suitable for such visualization, thus our further research on the topic is oriented toward this direction. In addition, we will experiment with more data sets in order to propose more accurately a value for the cut-off criterion.

References

[1] Mei, Z. An XML plagiarism detection algorithm for procedural programming languages. 2010 International Conference on Educational and Information Technology (ICEIT), 17-19 Sept., 2010, V3-427 - V3-431. IEEE, Chongqing.

[2] Moussiades, L., and Vakali, A. PDetect: A Clustering Approach for Detecting Plagiarism in Source Code Datasets. The Computer Journal, 2005, 48 (6), 651-661.

[3] Ottenstein, K. An algorithmic approach to the detection and prevention of plagiarism, ACM SIGCSE Bulletin, 8, 1976, 30-41.

[4] Grier, S. A tool that detects plagiarism in Pascal programs. ACM SIGCSE Bulletin, 1981, 13, 21-25.

[5] Donaldson, J., Lancaster, A. and Sposato, P. A plagiarism detection system. ACM SIGCSE Bulletin, 1981, 13, 15-20.

[6] Faidhi, J. and Robinson, S. An empirical approach for detecting program similarity and plagiarism within a university programming environment, Computers & Education, 1987, 11, 11-19.

[7] Allen, F. and Cocke, J. A program data flow analysis procedure. Communications of the ACM, 1976, 19, 137-147.

[8] Verco, K. and Wise, M. Software for detecting suspected plagiarism: comparing structure and attribute counting systems. In Proc. First Australian Conf. on Computer Science Education, July 3-5, 1996, 86-95. Sydney, Australia. ACM Press.

[9] Wise, M. YAP3: improved detection of similarities in computer program and other text. In Proceedings 27th SIGCSE Technical Symposium on Computer Science Education, February 15-18, 1996, 130-134. Philadelphia, USA. ACM Press.

[10] Prechelt, L., Malpohl, G., and Philippsen, M. Finding plagiarisms among a set of programs with Jplag. Journal of Universal Computer Science, 2002, 8, 1016-1038.

[11] Ribler R., and Abrams, M. Using visualization to detect plagiarism in computer science classes. In Proc. IEEE Symposium on Information Visualization, October 9-10, 2000, 173-178. Salt Lake City, UT, IEEE Computer Society, Los Alamitos, CA

[12] Poongodi , D. and Tholkkappia, G. A. 2013. An Automatic Method for Statement Level Plagiarism Detection in Source Code Using Abstract Syntax Tree. Int. J. of Adv. Research in Comp. & Comm. Engineering. , Vol. 2, Issue 4, pp 1923-1938

[13] Tian, Z.; Zheng, Q.; Liu, T.; Fan, M.; Zhuang, E.; Yang, Z. "Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences", Software Engineering, IEEE Transactions on, On page(s): 1217 - 1235 Volume: 41, Issue: 12, Dec. 1 2015

[14] Pramono, Y.W.T.; Suhardi "Detecting plagiarism in cross-platform mobile applications: Case study: Game application similarity in Symbian platform and Android platform", Information Technology Systems and Innovation (ICITSI), 2014 International Conference on, On page(s): 159 – 164

[15] Baby, J.; Kannan, T.; Vinod, P.; Gopal, V. "Distance indices for the detection of similarity in C programs", Computation of Power, Energy, Information and Communication (ICCPEIC), 2014 International Conference on, On page(s): 462 – 467

[16] Jianglang Feng, Baojiang Cui and Kunfeng Xia , "A Code Comparisor Algorithm Based on AST for Plagiarism Detection" , Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on,pp.393 -397

[17] Chakrabarti, S. Mining the Web. Morgan Kaufmann, Publishers, Oxford, 2003

[18] Zhao, Y., and Karypis, G. Evaluation of hierarchical clustering algorithms for document datasets. In Proceedings of the 11th International Conference on Information and Knowledge Management, 2002, 515-524, New York: ACM.

[19] Leydesdorff, L. On the Normalization and Visualization of Author Co-citation Data: Salton's cosine versus the Jaccard Index. Journal of the American Society for Information Science and Technology, 59(1), 2008, 77-85.

[20] Zhao, Y., and Karypis, G. Empirical and theoretical comparisons of selected criterion functions for document clustering. Machine Learning, 2004, 55(3), 331.

[21] Kaburlassos, V., Moussiades, L. and Vakali, A. Fuzzy Lattice Reasoning (FLR) Neural Computation for Weighted Graph Partitioning. Neurocomputing, 2009, Volume 72, Issues 10-12, 2121-2133.

[22] Brandes, U., Gaertler, M., & Wagner, D. Engineering graph clustering: Models and experimental evaluation. Journal of Experimental Algorithmics, 2008, 12, article no. 1.1.

[23] Jain, A. K., Murthy, M. N., and Flynn, P. J. Data clustering: A review. ACM Computing Surveys, 1999, 31(3), 264-323.

[24] Flake, G. W., Tarjan, R. E., and Tsioutsiouliklis, K. Graph clustering and minimum cut trees. Internet Mathematics, 2004, 1(4), 385-408.

[25] Moussiades, L. and Vakali, A. Clustering dense graphs: a web site paradigm. Information Processing & Management, 2010, Volume 46, Issue 3, 297-267

[26] Moussiades, L., and Vakali, A. Mining the Community Structure of a Web Site. Fourth Balkan Conference in Informatics, 2009, 239-244, Thessaloniki, IEEE.

[27] Stein, B., and Niggemann, O. On the nature of structure and its identification. In Graph-Theoretic Concepts in Computer Science, 1999, 122-134. London: Springer Berlin Heidelberg

[28] Newman, M. E. J. and Girvan, M. Finding and evaluating community structure in networks. Phys., 2004, Rev. E, 69, 026113, doi: 10.1103/PhysRevE.69.026113

[29] Newman, M. E. J. Fast algorithm for detecting community structure in networks. Physical Review 2004, E, 69, 066133.

[30] Gower, J. C., and Ross, G. J. Minimum spanning trees and single linkage cluster analysis. Applied Statistics, 1969, 18(1), 54-64.

[31] Chekuri, C. S., Goldberg, A. V., Karger, D. R., Levine, M. S., and Stein, C. Experimental study of minimum cut algorithms. In Proceedings of the eighth annual ACM-SIAM symposium on discrete algorithms, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997, 324-333

[32] Flake, G., Lawrence, S., and Giles, C. L., Efficient identification of web communities. In Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining, 2000, 150-160, Boston: ACM (PNAS) (Vol. 99, pp. 7821-7826).

[33] Hand, D., Mannila, H. and Smyth, P. Principles of Data Mining. Bradford Books, MIT Press, MA, 2001.