Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (14): 2843-2847, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.2843.2847

Programs Similarity Measure Based on Tree Structure and Eigenvector

[1,2]Dongmei Li, [1]Di Zhang, [3]Zhifang Wei and [1]Jianxin Wang
[1]School of Information Science and Technology, Beijing Forestry University, 100083, Beijing, China
[2]School of Computer and Information Technology, Beijing Jiaotong University, 100044, Beijing, China
[3]Department of School of Economics and Management, Tsinghua University, 100083, Beijing, China

Abstract: Program similarity measure technology is to detect the similarity among the programs by certain means. It is widely used in teaching and protection of intellectual property rights. Most current program similarity measure technologies suffer from low accuracy. Based on previous studies of program similarity measure method, this study proposes a method based on tree structure and eigenvector. Firstly, the actual frequency of keywords in the program is counted through employing hierarchical tree structure. Sencondly, the frequency is applied to generate eigenvector of program and the traditional method based on vector is improved. Finally, a program similarity measure system named Cplag is implemented which can be used to measure C language program similarity. Experimental results indicate that CPlag has apparent advantages in some aspects compared with famous Jplag.

Key words: Program similarity, keyword statistics, hierarchical tree structure, eigenvector

INTRODUCTION

Programming courses often require students to submit the electronic version of the program. Coping source codes from each other or from the network is common. The program similarity measure is to measure the degree of similarity between the submitted codes by certain means (Inoue and Wada, 2012). To a large extent, the program similarity measure tools can help teachers to improve the efficiency of checking job and reduce the plagiarism phenomenon. In addition, it also plays an important role in improving the effect of teaching and the consciousness of intellectual property rights protection. The authoritative definition of program similarity has not yet been recognized. One definition is that a code clone is a code portion in source files that is identical or similar to another (Kamiya *et al*., 2002).

In general, code plagiarism can be classified to three categories: code layout, syntactic equivalent conversion and semantic equivalent conversion (Xiong *et al*., 2010). Jones (2001) summarizes 10 kinds of plagiarism meanswhich are listed from low-level plagiarism to high-level plagiarism: (1) Verbatim copying, (2) Changing comments. (3) Changing white space and formatting, (4) Renaming identifiers, (5) Reordering code blocks, (6) Reordering statements within code blocks, (7) Changing the order of operands/operators within expressions, (8) Changing the type of data, (9) Adding redundant statements or variables and (10) Replacing control structures by equivalent structures. In addition, Zhao *et al*. (2008) claims that constant replacement could also be a kind of plagiarism. It is obviously that modifications mentioned are simple and easy to make, while the structure and function of program do not change (Aimmanee, 2011). Currently, there are some problems in current plagiarism measure, such as low measurement accuracy, high algorithm complexity. In this study, we develop a new more efficient and effective similarity measure tool, called CPlag.

REVIEW OF PROGRAM SIMILARITY MEASURE

The research of algorithms of similarity measure for programs has a history of more than 40 years. Along with its development, the process of current plagiarism measure often can be divided into two modules: The conversion of program format and the similarity measurement (Whale, 1988; Huang *et al*., 2010). The whole measure procedure can be divided into four parts: Pretreatment, format conversion, comparative units generation and matching algorithm. The purpose of pretreatment is to eliminate some useless information for the similarity measure. Its operation usually includes: unifying the code layout, eliminating hollow lines and comment lines, etc., (Grier, 1981; Faidhi *et al*., 1987). Codes are converted into kind of a format without changing the logic structure. Intermediate codes can be normalized text, tree structure, graph, etc. Then, the intermediate codes are

Corresponding Author: Dongmei Li, School of Information Science and Technology, Beijing Forestry University, 100083, Beijing, China

cut into fine-grained units, which are generally called comparative units. Common comparative units are word, string, eigenvector, tree node, graph structure, etc. According to different forms of comparative units, different matching algorithms are employed to detect the similarity. Current representative algorithms of similarity measure for programs are based on the identifier, metric, tree structure, graph, etc.

There are two kinds of typical systems of similarity measure based on identifier: YAP (Wise, 1992, 1996) series and JPlag system (Prechelt *et al*., 2002). YAP series only compare marked string composed by the keywords in the target language dictionaries and map synonym into the same form without complete syntax analysis about the target programming language. The experimental results showed that YAP is very effective to detect plagiarism of programs submitted by students. JPlag generates marked strings by the source codes and compares each pair of marked strings to measure the similarity. Similarity detection based on identifier and marked string do not have much difference in essence. They can boil down to the detection based on text and their prototype can be found in the study of copying detection of natural language documents.

Similarity measure based on metric is to use a series of metric values instead of source program (Donaldson *et al*., 1981). Because there is no conversion of source program, similar fragments can be located trivially. Similarity measure based on tree structure is to build a syntax tree through the analysis of the syntactic structure of programs and measure the similarity by matching the syntax tree node. The measure could detect the plagiarism with a totally different text structure. But a syntax tree with N nodes is compared with O (N^3) time complexity. Generally speaking, the average lines of program statements will be generated about 10 syntax tree nodes and comparison of each subtree has a time complexity of O (L^4) (L is the sum of statements). The cost of building a syntax tree is expensive, but the layer of a tree structure is clear. This study presents a method to count the actual frequency of keywords in the program through employing hierarchical tree structure. The frequency counted is used as eigenvector of program to measure the degree of similarity. We improve the traditional method based on vector by combining the advantages of length similarity and direction similarity in eigenvector. The proposed method is validated to be feasible and effective by our program similarity measure system named Cplag.

ALGORITHM OF SIMILARITY MEASURE

Considering the plagiarism method of constant replacement, we put forward a method of data type redefinition and finally summed up 11 kinds of common code plagiarism methods: (1) Verbatim copying. (2) Changing white space and formatting, (3) Changing comments, (4) Constant replacement, (5) Renaming identifiers, (6) Redefining data type, (7) Reordering statements within code blocks, (8) Changing the order of operands/operators in expressions, (9) Adding redundant variables, (10) Replacing control structures with equivalent structures and (11) Reordering code blocks. Combined with the characteristics of C language structure, we measure the similarity of program source code through the following procedure.

The CPlag takes the advantage of C++ language features and lexical rules to normalize the source program, uses keywords as marked strings, takes advantage of tree hierarchy characteristics to calculate the number that all keywords are actually used, uses vectors to identify two source programs, employs the Euclidean distance measurement vector to calculate the difference degree of two programs, uses the relationship between the similarity and difference degree to measure the similarity and normalizes the result of the similarity between 0.0 and 1.0. The closer it gets to 1.0, the higher the similarity.

Normalizing code: In order to improve the precision and operation efficiency of similarity measurement, here are four rules to normalize the source code.

Rule l. Delete blank, carriage returns and annotations.
Rule 2. Define a dictionary named DEFINEwhich is used to store redefinitions of data types.
For example: typedef int Status;
All the 'Status' in the source code will be replaced by 'int'.
Rule 3. The constant replacement made by const and define will be replaced by the constant value.
For example: const double PI = 3.14;
Define PI 3.14
All 'PI' in the source code will be replaced by 3.14.
Rule 4. The for loop is replaced by the while loop.
Here are the while loop and for loop grammar rules in C++ language:
<loop > :: = while '(' <exp> ')' <sta>
| for '(' [<exp>] ';' [<exp>] ';' [<exp>] ')' <sta>
For example, three statements of the following for loop are not omitted. Those statements can be transfered according to the rules of grammar.
for '(' sta1 ';' sta2 ';' sta3 ')' sta
Is equivalent to
sta1 while '(' sta2 ')'
'{'
'{' sta '}'
sta3
'}'

Through normalizing code, CPlag removes invalid fragment and replaces equivalent statements or structure and we can realize the measure of plagiarism method (1), (3), (6), (10).

Keyword statistics: The CPlag analyses nested blocks by scanning source program to construct the tree structure,

identifies the data type of the identifier and statistics the actual usage of each data type so as to accumulate the actually used times of keywords.

In CPlag, tree nodes are represented with the struct, as is shown in Fig. 1. Each node includes: variable type, the number of variables used and the node pointer.

Tree structure is used to store nested relations of code blocks. The CPlag traverses the tree for the statistics of keywords' actually used times. For a tree with N nodes, the time complexity is O (N).

Here is a keyword statistics example, as is shown in Fig. 2: The CPlag builds the tree structure to identify blocks nested depth and identifies each variable's data type in the subtree and the actual used times. The tree structure built is shown in Fig. 3, each nested block corresponds to a subtree.

Keywords' occurrence times and the statistical results of the actually used times is shown in Table 1.

By statistics on the actually used times of keywords, CPlag can detect plagiarism method (2), (4) (5), (7), (8), (9), (11).

Similarity judgment: Specific similarity measurement principle of CPlag is as follows:

- Use two vectors to mark the result of the statistics number that all keywords are actually used, get the eigenvectors a and b and the vector angle is β
- Based on the characteristics of vector, CPlag measures the similarity of eigenvectors from the direction and module: taking advantage of the cosine equation of vector angle to calculate the direction similarity, as is shown in Eq. 1. And the Eq. 2 is used to judge the module similarity
- Due to both of the direction and module has decisive role in the eigenvectors' similarity. In other words,

BD_Variable: String
B_Variable: Int
N_Node: TreeNode

Fig. 1: Structure of the tree node

```
#include<iostream.h>
int fun (int x,int y) {
        int a=0,b=0,c,d;
        a++;
        {
                a=0;
                while (a<10){
                        {
                                double b=2.5;
                                a=a+b*2;
                        }
                }
                a++;
        }
        b=b+1;
        cout<<"a="<<a<<",b="<<b<<endl;
        return x+y+a+b;
}
void main (){
        float a=fun(1,2);
        a++;
        cout<<a<<endl;
}
```

Fig. 2: Sample of keywords statistics

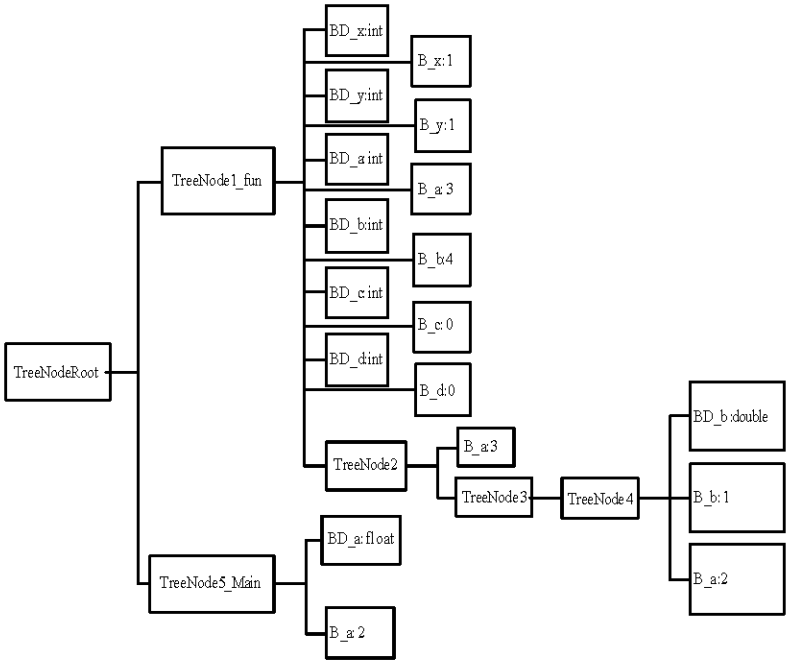

Fig. 3: Construction of tree structure

Table 1: Keywords statistics

Keywords	int	while	double	return	void	float
Occurrence	4	1	1	1	1	1
Actual frequency	14	1	1	1	1	2

when two vector's direction and module is completely different, or not at the same time, the two vectors can be judged as completely different. While eigenvectors are the same, their direction and module are exactly the same. Both the direction and module are fully independent relationship. According to the principle of mathematical statistics and probability theory, CPlag obtains the final eigenvector similarity by combining the similarity of direction and module. As is shown in Eq. 3. The closer the result gets to 1.0, the higher the similarity.

$$S_1 = \cos\beta = \frac{a * b}{|a| * |b|} \quad (1)$$

$$S_2 = 1 - \frac{|a - b|}{(|a| + |b|)} \quad (2)$$

$$S = S_1 * S_2 \quad (3)$$

EXPERIMENTAL RESULTS

Code similarity measure auxiliary software tools are represented by MOSS system (Schleimer *et al*., 2003) and JPlag. It is thought that detecting effect of JPlag is better compared with MOSS system, meanwhile CPlag has carried improvement on detection of identifier by employing tree structure to statistic actually used times of keywords. It is of representative significance to compare with JPlag.

Binary search algorithm is selected as the source program for measure of plagiarism by single method and 11 kinds of plagiarism method is used to get the corresponding 11 samples. CPlag can detect plagiarism by single method completely, while JPlag is able to detect plagiarism method (1), (2), (3), (4), (5), (8), (11) , but for other single plagiarism method, the effect is not good.

A website (http://www.sei.buaa.edu.cn/buaasim) provides a set of programs coped by comprehensive plagiarism methods. The programs are used to test and compare the measurement effect of CPlag and JPlag.

The similarity threshold of JPlag depends on the values of paired comparison and similarity below the threshold will not be returned.

For the convenience of making figure, the similarity of programs whose similarity are not returned is assumed

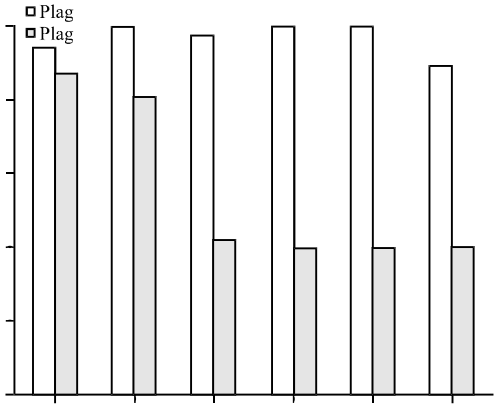

Fig. 4: Comparison of measure results

0.4, a decimal of the lowest similarity of program set. Comparison result of several typical comprehensive plagiarism measure by CPlag and JPlag is shown in Fig. 4.

From former two comprehensive plagiarism which can be detected completely by single (11) (4) (5), JPlag has a lower similarity to different degrees; From latter four comprehensive plagiarism, it is obviously that plagiarism method (7) and (10) can make much influence to the similarity. Plagiarism method (7) and (10) are kind of changes to the statement. The measure of comprehensive plagiarism of method (7), (10) and the other methods made by JPlag is even lower than similarity threshold and this explains that JPlag can only identify the plagiarism of the lexical level. While comprehensive plagiarism has little or no influence to the measure of CPlag.

Jplag does not need preprocess source codes and supports similarity measure of text and other programming languages. This is where CPlag is inferior to JPlag.

CONCLUSION

When CPlag is compared with traditional measure method based on metric, both of them consider the program syntax characteristics and are convenient to locate plagiarism code clips in the source code. Besides, CPlag takes both the module similarity and the direction similarity into consideration, so measure effect is better. when compared with measure technology based on the tree structure, CPlag does not consider the semantics of the program characteristics, weakening the accuracy of test results. But the complexity of tree similarity matching algorithm is high, making the efficiency low. So measure technology based on tree structure is more difficult to apply to large-scale software systems.

Currently, CPlag is not perfect for the measure of the expression separation plagiarism and can not detect the

plagiarism of changing data types. Data types that can be detected are not broad enough, the next step is to expand structure, class, template and other data types into detection. CPlag is designed only for programs written in C or C++ language and can be further extended to the detection of Java and other languages.

ACKNOWLEDGMENT

This study is supported by the National Nature Science Foundation of China (No. 61170628) and the National College Students' Training Programs of Innovation and Entrepreneurship (No. 201310022051).

REFERENCES

Aimmanee, P., 2011. Automatic plagiarism detection using word-sentence based S-gram. Chiang Mai J. Sci., 38: 1-7.

Donaldson, J.L., A.M. Lancaster and P.H. Sposato, 1981. A plagiarism detection system. Proceedings of the 12th SIGCSE symposium on Computer science Education. February 4-6 1981, New York, USA., pp: 21-25.

Faidhi, J.A.W. and S.K. Robinson, 1987. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Comput. Edu., 11: 11-19.

Grier, S., 1981. A tool that detects plagiarism in Pascal programs. Proceedings of the 12th SIGCSE Symposium on Computer Science Education, February 4-6, 1981, New York, USA., pp: 15-20.

Huang, L.L., H.Y. Huang and S.M. Shi, 2010. Method for fingerprint selection orienting to code similarity detection. Comput. Engin. Applic., 46: 169-171.

Inoue, U. and S. Wada, 2012. Detecting plagiarisms in elementary programming courses. Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), May 29-31, 2012, Chongqing University, pp: 2308-2312.

Jones, E.L., 2001. Metrics based plagarism monitoring. J. Comput. Sci. Colleges, 16: 253-261.

Kamiya, T., S. Kusumoto and K. Inoue, 2002. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. Trans. Software Eng., 28: 654-670.

Prechelt, L., G. Malpohl and M. Philippsen, 2002. Finding plagiarisms among a set of programs with JPlag. J. Univ. Comput. Sci., 8: 1016-1038.

Schleimer, S., D.S. Wilkerson and A. Aiken, 2003. Winnowing: Local algorithms for document fingerprinting. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 9-12, 2003, San Diego, California, USA., pp: 76-85..

Whale, G., 1988. Plague: Plagiarism detection using program structure. Department of Computer Science Technical Report 8805, University of NSW, Kensington, Australasian.

Wise, M.J., 1992. Detection of similarities in student program: YAP?ing may be preferable to Plague'ing. Proceedings of the 23rd SIGCSE Technical Symposium on Computer Science Education, March 5-6, 1992, Kansas City, Missouri, USA., pp: 268-271.

Wise, M.J., 1996. YAP3: Improved detection of similarities in computer program and other texts. Proceedings of the 27th SIGCSE technical symposium on Computer Science Education, March 10-12, 1996, New York, NY, USA., pp: 130-134.

Xiong, H., H.H. Yan and T. Guo, 2010. Code similarity detection: A survey. Comput. Sci., 37: 9-14.

Zhao, C.H., H.H. Yan and M.Z. Jin, 2008. Approach based on compiling optimization and disassembling to detect program similarity. J. Beijing Univ. Aeronautics Astronautics, 34: 711-715.