2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications

An AST-Based Code Plagiarism Detection Algorithm

Jingling Zhao[1,2], Kunfeng Xia[1,2], Yilun Fu[3], Baojiang Cui[1,2]
1.School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China
2.National Engineering Laboratory for Mobile Network Security, China
3.China Electric Power Research Institute, Beijing, China
Email: xkfzone@163.com, 18611917796@163.com

***Abstract*—In modern software engineering, software plagiarism is widespread and uncurbed, developing plagiarism detection methods is imperative. Popular technologies of software plagiarism detection are mostly based on text, token and syntax tree. Among these plagiarism detection technologies, tree-based plagiarism detection technology can effectively detect the code which cannot be detected by the other two kinds of technologies. In this paper,we propose a more effective plagiarism detection algorithm based on abstract syntax tree (AST) by computing the hash values of the syntax tree nodes, and comparing them. In order to implement the algorithm more effectively, special measurement is taken to reduce the error rate when calculating the hash values of operations, especially the arithmetic operations like subtraction and division. Results of the test showed that the measurement is reliable and necessary. It performs well in the code comparison field, and is helpful in the area of protecting source code's copyright.**

Keywords*—plagiarism detection, tree-based technology, code comparison, abstract syntax tree, hash value.

I. INTRODUCTION

Plagiarism had been defined as "unacknowledged copying of documents or programs" [1]. With the rapid development of open source ideas, many documents and source code are available on the internet and are easy to access [2]. Large numbers of open source systems are also available today. Meanwhile, software plagiarism also exist, especially, it is becoming more serious with the rapid development of the Internet. Plagiarism problem will be more serious when it exists in the commercial software. Departure of technicians and hacking activities can result in the leakage of source code. If rival firms get the code and use the code in their own products, it may bring great loss. So software copyright protection has drawn increasing attention, and software plagiarism detection becomes more and more important. In the procedure of software copyright protection and obtaining evidence about computer crime, it is necessary to take software plagiarism detection to determine ownership of the copyright. For example, Source code is a part of the software which forms the copyrighted work with interface design, algorithm together, but some people copy the source code without the permission of the author. Even to the open-source software, Some plagiarist do not comply with the license declared by the open source organization, and simply just take the source code as own. And in the field of computer education, academic plagiarism has been around for quite a while [3]. many students submit the source code they get from the Internet with no or a little modification in order to complete the task assigned by the professors. In order to help protecting the copyright of software, we need to continually study the code plagiarism detection technology.

In the early times of the conception of plagiarism, the main technology of detecting copy was based on file. This method is very simple and determines whether the two files similar by comparing the calculated values of the files. But, this technology can only detect the plagiarism with no modification. With the development of technology, many researchers have made a lot of research in this field and proposed lots of algorithms over the past decade [4][5], gradually forming three popular technologies of plagiarism detection: text-based, token-based and tree-based technology. Text-based plagiarism detection technology treats the source code as pure text, divides the text by line and compares the code line by line to detect the plagiarism between the files. The main algorithm used in this technology is the Longest Common Subsequence (LCS) algorithm. But, the algorithm has obvious limitations. Only if all strings are the same, the source code can be considered to be similar. Token-based technology is the improvement of text-based technology, some famous detecting tools, such as CP-Mine, CCFinder, Winnowing, and JPlag are all token-based [6][7][8]. They all can not detect the modification of renaming, reordering, and inserting null strings. Tree-based plagiarism detection is a new technology compared with token-based and text-based methods, it detects plagiarism from the structure of the syntax tree and can detect those plagiarism cases can not detected by token-based and text-based technology. So, it develops very fast in past a few years.

Recently, with further study, some improved algorithms have been proposed based on the three plagiarism detection technology [9]. Mikkel Jonsson Thomsen and Fritz Henglein conducted further study about tree-based plagiarism detection using rolling hashing, suffix Trees and dagification [10]. Sandhya Narayanan and Simi S proposed a new algorithm based on distance measure method on the 7th ICCSE, 2012. The algorithm can check the properties of the language syntax to identify the statement usage and then outputs the detailed fingerprint of the source code [11]. As to semantic-based detection technology, many scholars dedicated to the research of semantic analysis and they have proposed some new methods and algorithms. Sergey Butakov, Marina Kim and Svetlana Kim proposed a low RAM footprint algorithm for small scale projects [2]. Osman puts forward a method of Semantic Role Labeling (SRL) to detect plagiarism [12]. Georgina Cosma and Mike Joy propose a new plagiarism detect method of using Latent Semantic Analysis [13]. With further study in the field of semantic-based plagiarism detection, some efficient and practical algorithms will be widely used and we need not to consider additional language grammar rules, in addition to providing the source code, which reduces the cost of analysis.

Last one or two years, some new detection technologies

978-1-4673-8315-8/15 $31.00 © 2015 IEEE
DOI 10.1109/BWCCA.2015.52

have been proposed [14][15]. Style-based plagiarism detection is one of the new technologies, it mainly includes two steps. Firstly, the main features representing a coding style are extracted. To determine the plagiarized codes each code is compared to all other codes submitted to the system and also it is compared to its owner's style. Secondly, the extracted features are used in three different modules to detect the plagiarized codes and to determine the giver and takers of the codes [16]. Another new technology is based on citation and Citation-based plagiarism detection subsumes methods that use citations and references for determining similarities in order to identify plagiarism. This technology is mainly used to detect similarity of documents. In the academic environment citations and references of scholarly publications have long been recognized for containing valuable semantic information about the content of a document and its relation to other works [17].

This paper proposed a code comparison algorithm based on abstract syntax tree. We raise the efficiency of comparison by transforming the storage format of the syntax tree twice, converting the tree-like structure into a linear list and regrouping the sub-trees according to the number of sub-nodes. Thus, the efficiency improves. In addition, the algorithm also takes special measurement to reduce mistakes when calculating the hash value of the operation like subtraction and division.

We will introduce the create procedure of the abstract syntax tree (AST), and give an example of the AST structure. Secondly, we describe the code comparison algorithm in detail (we call it AST-CC algorithm in the rest part of this paper). Then an experiment will be shown to evaluate the AST-CC algorithm. Finally, we conclude with a discussion of the AST-CC algorithm.

II. The Structure of Abstract Syntax Tree

Since we want to take into account the syntax factor of the source code, rather than remain at the text level, we need to find a data structure that can store the syntax information of the source code, and that's the abstract syntax tree generated from the source code.

Firstly, we need to preprocess the source code.Secondly, we get the lexical and syntax analysis of the source code.Finally, we will allocate memory space, generate a syntax tree node, and record the corresponding node type, as well as the position in the source code.

When the token generated by the lexical analysis tool matches a special rule in the syntax analysis compiler, an abstract tree node will be created and some information of the node will be record, such as the location of the node, the name of the node, the line number of the source file.Algorithm 1 is a source code example, and the AST structure of Algorithm 1 is shown in Figure 1.

Algorithm 1 Source code

```
void fun1( )
{
    int a = 2;
    int b = 5;
    fun2(a, b);
}
```

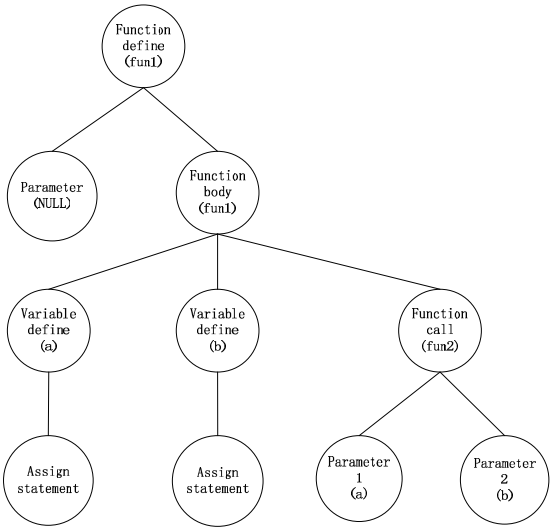

Figure 1: Abstract syntax tree generated from Algorithm 1

Syntax tree can be seen from Figure 1. It can fully reserve the syntax structure information of the source code, and can also be used to perform a comprehensive analysis of the source code. For example there are two statements CalculateA (int a, int b, int c), and CalculateB (int x, int y, int z). The latter copies from the former, and then changes the function name and parameter names. In the syntax tree, they are all expressed as a "function call" node with three sub-nodes. We suppose that the hash value of "function call" node is 20, the hash value of parameter is 15, then the hash value for both of the above two function call statements is (function call) +3 * Hash (int type argument) = 20 +3 * 15 = 65, so we can find that the two statements are similar by using syntax tree.

III. The Code Comparison Algorithm

After we get the syntax tree structure of the source code, what we need to do is comparing the syntax tree. Since most of the syntax tree is quite complex and bulky, the direct comparison of the syntax tree node is difficult, and the efficiency will be very low. So, we consider the syntax tree's hash value, and calculate the value based on the type of each node and its sub-tree of the syntax tree. Then compare the hash values. As a result, the difficulty of comparison is reduced and the information of the syntax tree is not lost.

The main steps are as follows: First, call the hash value generation algorithm of the abstract syntax tree to traverses

the syntax tree, and get the hash value and other related information of the suspected code and original code's syntax tree respectively. Then classify each sub-tree by its sub-node number, and store the information to an array of linked lists. Then call the hash comparison algorithm to compare the sub-tree with the same number of sub-node in the syntax trees generated from the suspected code and the original code. Finally record the sub-trees with the same hash value and their corresponding source code position, and highlight the same part in source files.

A. Syntax tree comparison algorithm

The following is the description of the syntax tree comparison algorithm:

- Read information of the suspected code from HashListArray at position where the array index is equal to the threshold (that's, the number of sub-nodes is equal to the threshold value, because we stores the information of the sub-tree with n sub-nodes in HashListArray[n]), and then compare with original code.
- Traverse the arrays of SuspectedHashListArray and OriginalHashListArray which store the information of the suspected code and original code respectively. After that, sort these two arrays and make sure the sequence of the nodes is stored by the node number and hash value.
- Compare the hash values of the sub-trees with the same sub-node number. If they share the same hash value, then store their positions in the source files into database.

We can describe the algorithm in Algorithm 2.

Because that sub-trees with different sub-node numbers represent different structures, only pairs of sub-trees with the same sub-node number need comparing. Thus we can reduce the number of comparison and the complexity of the algorithm.

B. Algorithm to avoid false positive report

By simply adding the hash value of the sub-nodes, we get the hash value of the sub-tree, and in this way we can deal with the plagiarism tricks like breaking the sequence of the statement, but it also led to some false positive report, which means the similar code we reported is not truly similar. Before we take special measurement, we will report that "$x-y*z$" is similar to "$y*z-x$". Suppose that the hash value of x is 3, the hash value of subtraction is 10, and the hash value of "$y*z$" is 18, then the hash value of "$x-y*z$" is $3+10+18=31$, and the hash value of "$y*z-x$" is $18+10+3=31$. So we get a false report. To avoid this situation, we give different weights to the left and the right sides of some special operations such as subtraction, division, and so on. As a result, false reports as mentioned in the last paragraph can be avoided. The hash value of "$x-y*z$" is equal to hash (x) + hash (-) +2 * hash ($y*z$), and the hash value of "$y*z-x$" is equal to hash ($y*z$) + hash (-) +2 * hash (x), so the hash values of the two statements mentioned above "$x-y*z$" and "$y*z-x$" are respectively $3+10+2*18=49$ and $18+10+2*3=34$. In this way, we can avoid some false reports.

Algorithm 2 Comparison

```
Arrays.sort(SuspectedHashListArray);
//SuspectedHashListArray[0] represents the root node Arrays.sort(OriginalHashListArray);
//OriginalHashListArray[0] represents the root node
for i = 0, j = 0;
i <SuspectedHashListArray.length&&
j < 0riginalHashListArray.length; do
  if SuspectedHashListArray [0].compareTo(OriginalHashListArray [0]) == 0 then
    Add all the nodes into the database;
    break;
  end if
  P1=SuspectedHashListArray[i].jjgetparent();
  P2=OriginalHashListArray[j].jjgetparent();
  /*
  *If the hash value of parent is equal,
  *skip the comparison of their children
  */
  if P1!=null&&P2!=null&&P1.compareTo(P2)==0 then
    i++;
    j++;
    continue;
  end if
  if SuspectedHashListArray[i].compareTo (OriginalHashListArray [j])< 0 then
    i++;
  else
    if SuspectedHashListArray[i].compareTo (OriginalHashListArray [j])> 0 then
      j++;
    else
      Add the nodes into the database;
      i++;
      j++;
    end if
  end if
end for
```

IV. The Evaluation of AST-CC Algorithm

The advantages of AST-CC algorithm are following: It converts the tree structure of storing form into linear list so that it can increase the speed of polling; It groups the syntax tree information based on the number of the sub-node to reduce unnecessary comparing operation; It saves the analysis result of the original source code into the database so that it will be more efficient when reuse it.

Using AST-CC algorithm, we could detect the plagiarism such as block copy, name substitute (function names, class names, and variable names), adding "typedef" statement after copying, order breaking of the statements, parameters, switch cases etc. In addition, AST-CC algorithm also takes special measurements to the operation like subtraction, division. These can make detection more accurate.

In order to prove that AST-CC algorithm improves the accuracy of code comparison, we had a lot of experiment using the source code getting from the open source software. Here we only show two sample source codes in Algorithm 3 and Algorithm 4. The environment of our experiment is: Microsoft

E:\Method Samples\Algorithm3.java			E:\Method Samples\Algorithm4.java		
1	1	public class JTreeTable extends	1	1	public class JTreeTable extends
2	2	{	2	2	{
3	3	protected TreeTableCellRenderer tree;	3	3	protected TreeTableCellRenderer tree;
4	4	public JTreeTable(TreeTableModel treeTableModel)	4	4	public void updateUI()
5	5	{	5	5	{
6	6	setIntercellSpacing(new Dimension(0, 0));	6	6	super.updateUI();
7	7	if (tree.getRowHeight() < 1) {	7	7	if (tree != null) {
8	8	setRowHeight(18);}	8	8	tree.updateUI();
9	9	}	9	9	}
10	10	public void updateUI()	10	10	LookAndFeel.installColorsAndFont(this, "Tree.bac
11	11	{	11	11	}
12	12	super.updateUI();	12	12	public JTreeTable(TreeTableModel treeTableModel)
13	13	if (tree != null) {	13	13	{
14	14	tree.updateUI();	14	14	setIntercellSpacing(new Dimension(0, 0));
15	15	}	15	15	if (tree.getRowHeight() < 1) {
16	16	LookAndFeel.installColorsAndFont(this, "Tree.bac	16	16	setRowHeight(18);}
17	17	}	17	17	}
18	18	public JTree getTree() {	18	18	public JTree getTree() {
19	19	return tree;}	19	19	return tree;}
20	20	}	20	20	}

Figure 2: Plagiarism code detected by using AST-CC

Windows XP Professional SP3 on AMD Athlon (tm) 64 X2 Dual Core Processor $4400 + 2.29$ GHz, 1.00 GB of memory.

Algorithm 3 Example of the method declarations (extract from JtreeTable.java in JPPF-2.1.1-full)

```
public class JTreeTable extends JTable
{
    protected TreeTableCellRenderer tree;
    public JTreeTable(TreeTableModel treeTableModel)
    {
        setIntercellSpacing(new Dimension(0,0));
        if (tree.getRowHeight()< 1){
            setRowHeight(18);
        }
    }
    public void updateUI()
    {
        super.updateUI();
        if (tree != null){
            tree.updateUI();
        }
        LookAndFeel.installColorsAndFont(this,
"Tree.background", "Tree.foreground", "Tree.font");
    }
    public JTree getTree(){
        return tree;
    }
}
```

Algorithm 3 is extracted from JTreeTable.java getting from JPPF-2.1.1-full. JPPF enables computation-intensive applications to run on any number of computers, in order to greatly reduce their processing time. We got Algorithm 4 after modifying Algorithm 3 by randomly exchanged the locations of several method declarations. And the function of this part of code is the same.

Algorithm 4 Source code example by randomly exchanging the location of several method declarations in Algorithm 3

```
public class JTreeTable extends JTable
{
    protected TreeTableCellRenderer tree;
    public void updateUI()
    {
        super.updateUI();
        if (tree != null){
            tree.updateUI();
        }
        LookAndFeel.installColorsAndFont(this,
"Tree.background", "Tree.foreground", "Tree.font");
    }
    public JTreeTable(TreeTableModel treeTableModel)
    {
        setIntercellSpacing(new Dimension(0, 0));
        if (tree.getRowHeight()< 1){
            setRowHeight(18);
        }
    }
    public JTree getTree(){
        return tree;
    }
}
```

The analyzing result of Algorithm 3 and Algorithm 4 using the AST-CC algorithm is separately shown in Figure 2. Compared with the result we generated using CloneDr, we find that the AST-CC can detect all the method declaration clones where CloneDr can only detected some clones separately.

V. Conclusion

We mainly introduce an AST-based detection plagiarism (AST-CC algorithm) in this paper .The algorithm can effectively detect plagiarism with the process of generating hash values and comparing them. In the mean time, AST-CC algorithm can eliminate the potential errors of some arithmetic operations, such as subtraction, division, modulo arithmetic and so on, which have been proved by the result of our experiments. In terms of efficiency, AST-CC algorithm can work in a more efficient state because of the storage form of AST, which have improved largely compared with the previous. So we believe the AST-CC algorithm will be widely used in the future.

Acknowledgment

This work was supported by National Natural Science Foundation of China (No.61272493).

References

[1] M. L. A Joy, M.: Plagiarism in Programming Assignments. IEEE Transactions on Education vol. 42, pp. 129-133, 1999.

[2] Sergey Butakov, Marina Kim and Svetlana Kim: Low RAM footprint algorithm for small scale plagiarism detection projects. Information Science and Applications (ICISA), 2012 International Conference. Page(s): 1-2.

[3] Daniela Chuda, Pavol Navrat, Senior Member, IEEE, Bianka Kovacova, and Pavel Humay: The Issue of (Software) Plagiarism: A Student View. IEEE TRANSACTIONS ON EDUCATION, VOL. 55, NO. 1, FEBRUARY 2012.

[4] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In SAS, pages 40-56, 2001.

[5] Jens Krinke. Identifying similar code with program dependence graphs. 1095-1350/01 IEEE, pages 301-309, 2001.

[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: A tool for finding copy-paste and related bugs in operating system code. In OSDI, pages 289-302, 2004.

[7] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, 28(7):654-670, July 2002.

[8] Chanchal K. Roy, James R. Cordy, Rainer Koschke. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. Science of Computer Programming, February, 2009.

[9] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone Detection Using Abstract Syntax Trees Proc. IEEE Int'l Conf. Software Maintenance (ICSM '98), pp. 368-377, Nov. 1998.

[10] Thomsen, M.J. ; Henglein, F.: Clone detection using rolling hashing, suffix trees and dagification: A case study. Software Clones (IWSC), 2012 6th International Workshop. Page(s): 22-28.

[11] Narayanan, S. ; Simi, S.: Source code plagiarism detection and performance analysis using fingerprint based distance measure method. Computer Science & Education (ICCSE), 2012 7th International Conference. Page(s): 1065-1068.

[12] Osman, A.H. ; Salim, N. ; Binwahlan, M.S. ; Twaha, S. ; Kumar, Y.J. ; Abuobieda, A.: Plagiarism Detection Scheme Based on Semantic Role Labeling. Information Retrieval & Knowledge Management (CAMP), 2012 International Conference. Page(s): 30-33.

[13] Georgina Cosma and Mike Joy : An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis. IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, NO. 3, MARCH 2012. Page(s): 379-394.

[14] Wibowo, Agung Toto , Sudarmadi, Kadek W. , Barmawi, Ari M.: Comparison between fingerprint and winnowing algorithm to detect plagiarism fraud on Bahasa Indonesia documents. Information and Communication Technology (ICoICT), 2013 International Conference. Page(s): 128-133.

[15] Chan, P.P.F. ; Hui, L.C.K. ; Yiu, S.M.: Heap Graph Based Software Theft Detection. Information Forensics and Security, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 8, NO. 1, JANUARY 2013. Page(s): 101-110.

[16] Arabyarmohamady, S. ; Moradi, H. ; Asadpour, M.: A Coding Style-based Plagiarism Detection. Interactive Mobile and Computer Aided Learning (IMCL), 2012 International Conference. Page(s): 180-186.

[17] Bela Gipp, Norman Meuschke, Joeran Beel: Comparative Evaluation of Text- and Citation-based Plagiarism Detection Approaches using GuttenPlag. Jun. 2011 Proceeding of the 11th annual international ACM/IEEE joint conference.