The Source Code Plagiarism Detection using AST

Xiao Li
School of Computer Science and Technology
SouthWest University for Nationalities
Chengdu 610041, P.R.China
x266@163.com

Xiao jing Zhong
School of Computer Science and Technology
SouthWest University for Nationalities
Chengdu 610041, P.R.China
x267@163.com

Abstract—**In the instruction of computer courses, some students copy other's source code as themselves. In order to detect this plagiarism accurately, researchers did a lot.In this paper,we described a source code plagiarism detection technologe based on AST,This technologe can detect the plagiarism accurately when the position of functions is changed by plagiarist.At first,transforming the programs to the AST using ANTLR,and then,abstracting the function subtrees from the AST,at last,compare the function subtree using LCS,get the simalarity between programs.**

Keywords-plagiarism detection, AST,ANTLR, LCS

I INTRODUCTION

In instruction of computer courses, the plagiarism [1] in students makes the instruction insignificantly; the plagiarists get their work by duplicating others' source code and sometimes making a little edit without any knowledge about programming. It is hard for instructor to detect the plagiarism, for there are alwalys many students in a class. So some people are dedicated to designning and implementing a kind of plagiarism detection system of source code. And nowadays, there are some sophisticated systems for plagiarism detection, such as Jplag [2], MOSS [3] and YAP [4] etc.

In this paper, we intruduced a new plagiarism detection technologe of source code based on AST. The part 1 talks about the typical plagiarism detection of source code; the part 2 talks about the new plagiarism detection based on AST; the part 3 talks about the implementing of the system; the part 4 is the conclusion.

II RELATED WORK

The typical systems for plagiarism detection such as Jplag MOSS YAP, which have a common method for transact the program.At first, transform the programs into token streams; and then, compare the token streams by some algorisms for computing similarity [5-7]. In this paper, we called this typical system as plagiarism detection of source code based on token, the detection processes of which always as follow:

1) Delete the comments, identifiers and whitespaces;
2) Transform the source code into token streams;
3) Compare the token sequences, compute the similarity.

In step 1,all the comments and whitespaces are omited, the identifiers such as name of variable and function are omited too; in step 2,the token streams take the place of source code; in step 3,compare the token sequences of difference programs, get the similarity;in step 3, the typical algorism is RKRGST[2,4,8].

The advatage of the above mentioned systems is which can detect the simple plagiarisms, such as change the name of variable, function, comments and whitespaces; it is also effective to detect the plagiarism by duplicating the source code absolutely without any changes. However, these kinds of systems have a big defect, which are ineffective for detecting the plagiarism when the positions of the blocks in the program are changed!

The changes of positions of blocks in program are acclaimed by the plagiarist, for which need no knowledge of programming and can be done easily. In order to validate this, author did a lot of tests,all the results indicate:the plagiarism detection based on token is in sensitive to detect the plagiarsim when the positions of block are changed.There is the data of one experiment in figure 2-1.In this experiment,there are two c++ source files,file1 and file2.File1 is the source code,file2 is the piratical edition of file1 by changing the positions of four functions in file1.When detect these two files in Jplag,the similarity of them is 87%!which is unreal

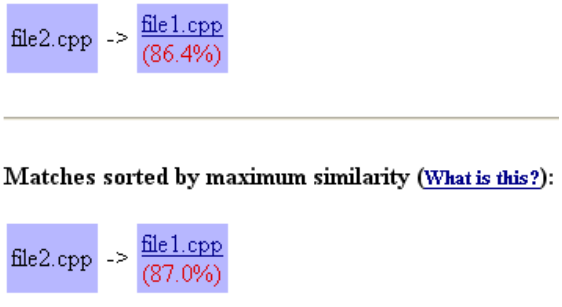

Figure 1 the detection results of file1 and file2 in Jplag

978-0-7695-4196-9/10 $26.00 © 2010 IEEE
DOI 10.1109/IPTC.2010.90

What is the reason? Because they can not distinguish the statements, expressions and functions.It can compare different file only when the file is seen as a whole.If the position of block is changed in the file, which is considered as a differen file.The similarity is decreased of course.

III THE PLAGIARISM DETECTION BASED ON AST

In order to resolve the problem in typical system of plagiarism detection, we advance a new solution, which partition the source code before comparing it, and then compare the portions in different files.This solution omit the orders of the portions.The keystone of this way is to break the file up, how to do it? What is the particle size? The particle size can be token, expression, statement and function, how to choose? In this solution, the operating steps are as follow:

1) Transform the source code into the form of AST[10];
2) Split the AST of source code into subtrees,the subtree is a complete semantic structure;
3) Compare the subtrees of two programs, calculate the similarity of programs.

In this technique, the step one is emphases.In stead of transforming program into token stream but AST, which can denote the structure of a program more clearly.in step two, splitting the AST.the keystone is the particle size.if splitting the program as token, the result is same as the typical plagiarism detection system; if splitting the program as statement, the particle size is too small to work; at last, we choose to split the program as function [11, 12]. In step three, compare the function subtrees. In order to reduce the calculation of comparing subtrees, only the subtrees in different programs which nodes number is different between 20 can be compared.

In computer science, an abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is 'abstract' in the sense that it does not represent every detail that appears in the real syntax. For instance, grouping parentheses are implicit in the tree structure, and a syntactic construct such as an if-condition-then expression may be denoted by a single node with two branches.One node in the tree denotes a construct in the source code,and the sub-nodes of which denote the component parts of the node.So it is possible for us to get a construct such as function from the AST by splitting the AST.

IV SYSTEM IMPLEMENTING

It is easy to generate the AST of the source code, however, the AST is always not we want, it is too detail. What we want in this paper is aN AST which omit the comment, identifier and whitespace. We get this kind of AST by using ANTLR. The user can edit the lexer and parser grammar to generate the AST which they want.

```
: (' ' | '\t' | '\n' | '\r')    { skip(); }
;
COMMENT
: '/*' (.)* '*/'    { skip(); }
;
LINE_COMMENT
: '//' ~('\n'|'\r')*    ('\r\n' | '\r' | '\n') { skip(); }
| '//' ~('\n'|'\r')* { skip(); }
;
```

Figure 2 c++ grammar in ANTLR

ANTLR is a sophisticated parser generator. You can use to implement language interpreters, compilers, and other translators. ANTLR's highest-level construct is a grammar, which is essentially a list of rules describing the structure of a particular language. From this list of rules, ANTLR generates a recursive-descent parser that recognizes sentences in that language (recursive-descent parsers are the kind of parsers you write by hand). The language might be a programming language or simply a data format, but ANTLR does not intuitively know anything about the language described by the grammar.figure 3 is an AST segment of a c++ program file.

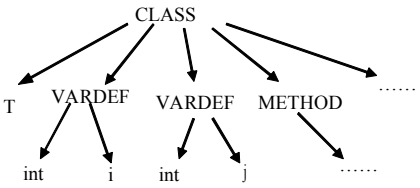

Figure 3 AST of a c++ class

In this paper,using LCS algorism to match the function subtrees.In order to reduce the number of matches, only the subtrees which number of nodes is greater than a value (such as minimalNodSize) can be matched.Howerver, when comparing program A and B, if A generates n function subtrees which nodes number is larger than minimalNodSize, and B generates m function subtrees which nodes number is larger than minimalNodSize, and then there needs m * n matchings ... ... the amount of calculation is huge!In order to reduce the operation, building a hash table, for the subtree a1 in A, only those subtree hash values equal to the hash values of a1 can be compared with a1. For the programs A and B, if m>=n, the number of the similarities detected can be up to n; to make the n-similarity values as: S1, S2 ... ... Sn, the following expression is the similarity of A and B:

$$sim=\frac{S1+S2+\cdots+Sn}{n}$$

The time complexity of the plagiarism detection based on AST is O (n^3), after Optimization, the time complexity can be O (n^2).

V CONCLUSION

Comparing the prototype system based on AST with Jplag for small batches of 40 files, the test results as Table 5-1:

TALBE 1 SYSTEM'S SUCCESS RATES COMPARING

success rate / plagiarism type	the system based on AST	Jplag
fully copy	100%	100%
chang the comments,whitespace and identifiers	100%	100%
chang the turns of statements	92%	90%
chang the turns of function declare	100%	77%
equivalent Replacement the expression	91%	94%

From the table is not difficult to find, the plagiarism detection based on AST is fully able to detect the plagiarism that changed the function declaration orders, On the contrary, the effect of Jplag in this regard is poor.

The technologe imported in this paper existes some problems and need further improvement, such as follow:

When you need to add a new language, which need to increase a new set of lexical grammars, parser grammars and operational rules of the tree, but the typical detection method based on token, when need to add new language, adding new lexical grammars is ok. This hopes to improve in the future.

[1] Verco KL, Wise MJ. Software for detecting suspected plagiarism: compareing structure and attribute-counting systems. In: Proceedings of the 1st Australian Conference on Computer Science Education. 1996. pp3-5.

[2] Prechelt L, Malpohl G, Philippsen M. Finding plagiarism among a set of programs with JPlag. Journal of Universal Computer Science, 2002,8(11):1016~1038.

[3] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data, pages 76–85, 2003.

[4] Wise MJ. YAP3: Improved detection of similarities in computer programs and other texts. In: Proceedings of the SIGCSE'96. 1996, 130~134.

[5] Alan Parker and James O. Hamblen. Computer algorithms for plagiarism detection. IEEE Transactions on Education, Volume 32, Issue 2, May 1989 :pp94 – 99.

[6] J. A. W. Faidhi and S. K. Robinson. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Computers & Education, Volume 11, Jan 1987:pp11-19.

[7] G. Whale, Identification of Program Similarity in Large Populations, The Computer Journal, Vol. 33, Number2, 1990.

[8] Michael J Wise.String similarity via Greedy String Tiling and Running Karp-Rabin Matching[M].An Unpublished Paper,1993.

[9] Paul Clough,"Plagiarism in natural and programming languages:an overview of current tools and technologies"

[10] Ira D. Baxter, Andrew Yahin, Leonardo Moura et al. Clone Detection Using Abstract Syntax Trees. Proceedings of the International Conference on Software Maintenance, 1998, 368-377.

[11] Lijie Zou and Michael W. Godfrey. Detecting merging and splitting using origin analysis. In Proceedings of the Working Conference on Reverse Engineering, 2003.

[12] Qiang Tu and Michael W. Godfrey. An integrated approach for studying architectural evolution. In Proceedings of the International Workshop on Program Comprehension, 2002.