Journal of Electrical and Electronic Engineering
2018; 6(2): 53-58
http://www.sciencepublishinggroup.com/j/jeee
doi: 10.11648/j.jeee.20180602.13
ISSN: 2329-1613 (Print); ISSN: 2329-1605 (Online)

Research of Automatic Scoring of Student Programs Based on Static Analysis

Dongmei Yan, Xiangyuan Qi, Wenyue Yang

Department of Information Science and Technology, Tianjin University of Finance Economics, Tianjin, China

Email address:
297315708@qq.com (Dongmei Yan), 1344126822@qq.com (Xiangyuan Qi), 893848044@qq.com (Wenyue Yang)

To cite this article:
Dongmei Yan, Xiangyuan Qi, Wenyue Yang. Research of Automatic Scoring of Student Programs Based on Static Analysis. *Journal of Electrical and Electronic Engineering*. Vol. 6, No. 2, 2018, pp. 53-58. doi: 10.11648/j.jeee.20180602.13

Received: April 20, 2018; **Accepted**: May 17, 2018; **Published**: June 20, 2018

Abstract: In the student programming examination, the program must be automatically evaluated. It can be not give a reasonable score to the wrong program by comparing the output results of the dynamic evaluation method. Only by using the static analysis of the program can it give more accurate results. In this paper, the ratio of the length of the program feature vector and the Token sequence are introduced in two static analysis algorithms of attribute count and the longest common subsequence, and the optimum weight of various algorithms is determined by experiments. The experimental results show that the score given by the algorithm is very close to the teacher's score, which proves that the algorithm is an effective automatic scoring method.

Keywords: Automatic Scoring, Static Analysis, Attribute Counting, Longest Common Subsequence

1. Introduction

With the development of network communication and computer technology, more and more online automatic grading is used in the evaluation of programming test. At present, dynamic judgment is used in most of the domestic MOOC systems and some contest systems such as Online Judge. They compares the running results of the program to be tested and the correct program, so the judgment results can only be correct and incorrect. This scoring method can quickly judge the correctness of the program. However, if it is introduced into the online examination of programming courses in colleges and universities, it may not accurately reflect the students' programming thinking and programming level. So, it is more reasonable to use static analysis method to judge the student program. There are currently many static analysis algorithms, such as attribute counting method [1], longest common sequence (LCS) algorithm [2], token sequence [3], abstract syntax tree [4], system dependency graph [5] and so on. Among them, abstract syntax tree [6] and system dependency graph [7] have good performance in program static analysis, but their time and memory performance is usually unable to meet actual needs [8]. The attribute counting method and the LCS algorithm measure the similarity between the student program and the correct answer from the view of the program properties and structure respectively, and they are complementary to each other. At the same time, they meet the actual performance requirements.

In this paper, an online automatic scoring method based on the two static analysis algorithms which are attribute counting method and LCS algorithm is proposed. In the attribute counting method, the values of angle cosine and length ratio of the program feature vectors are calculated, and in the LCS algorithm, the subjective identifiers such as user-defined variables are processed by Token sequence, and the similarity value is calculated. Finally, the optimal weights of the above three values are determined by experiments.

2. Program Preprocessing

Programs typically contain comments, header introduction statements, macro definitions, blank lines, spaces, enter and Tab characters [9], and the comments include line comments and block comments. The preprocessing needs to remove these non-logical parts and obtain a program that contains only the logical parts.

It reads the program code in units of line, and if the first non-space character of this line code is '#', this line code is a header file introduction statement or macro definition. If this line of code contains the string '//', it contains line comments. Finally it scans this line code and get blank lines, spaces, enter

and Tab characters.

When scanning the program, it also need to remove block-level comments which are the contents in the middle of the string '/*' and '*/'. According to obtain the index values of the string /*' and '*/', the block-level comments are identified.

The preprocessed program is passed to the test procedure as a string. Attribute counting method and LCS longest public string method will take the preprocessed string as the analysis basis and carry out similarity analysis.

The algorithm to get rid of the line comments:

```
while ((line = br.readLine() ) != null)
{
   int index=line.indexOf("//");
   if (index>=0){
      //This line contains line comments
      line=line.substring(0, index);
   }
   if (line!=null&&line.length()>0){
      //Add this line to the code
      code=code+" "+line;
   }
}
```

The algorithm to get rid of the block comments:

```
int pre_index,next_index;
while((pre_index=code.indexOf("/*"))!=-1) {
   //There is only "/*" but not "*/"
   next_index=code.indexOf("*/");
   if(next_index<0){
      //Get rid of the part from "/*" to the end of the code
      str=str.substring(0,pre_index);
      break;
   } else{
      //Get rid of the part from "/*" to "*/"
      str=Attribute.remove(str,pre_index,next_index+2);
   }
}
```

3. Attribute Counting Method

Code variants include complete copy of code, replacement of function name and variable name, disordering of statements, type redefinition and so on. At present, software homology detection technology is mainly divided into binary programs and source code. The computer language has fewer components, and only a limited number of keywords, so the software homology detection technology based on code level has the advantages of simple implementation and high efficiency.

Attribute counting method is the earliest proposed method of program code detection. According to this method, the program source code is regarded as the same as general entities which have some characteristics, such as vocabulary, code lines, the number of variables and capacity, and so on. Attribute counting is the statistics of these special attributes of program source code. The number of these attributes statistics varies from program source code to program source code. But the same or similar program source code of these attributes statistics are not very different.

The attribute of program refers to the intrinsic nature of the program, which is extracted from the program code, and does not vary with the form of expression [10]. Attribute counting is the program attributes that can be measured by statistics, such as the number of program lines, the number of characters, and so on.

3.1. Selecting Program Properties

Attribute refers to the nature or characteristics of things that do not change with the form of expression. In the case of program code, the attributes are usually physical attributes and structural attributes. Physical properties include inherent features of program source code, such as unmeasurable programming languages, program functions, measurable lines of code, words, characters. Structural attributes refer to all operands and operators in a program, including variables, constants and symbols, which are collectively called identifiers. The attribute feature vector must have typical features that uniquely represent the source code of the program and it can be distinguished from other programs, so that the attributes which act as the elements in the vector need to be carefully selected. The first step is to select measurable physical properties, which are the most basic measures of program code to describe the size of source files. Since structural attributes describe a source file more accurately than physical attributes, structural attributes are also essential elements to form attribute feature vectors. To sum up, the elements of the attribute feature vector of a program code file are generally composed of physical attributes and structural attributes, in which the physical attributes include the number of lines of code, the total number of words, the total number of characters, and the structural attributes include the total number of identifiers, number of keywords, number of user-defined identifiers, operators, numeric constants, word strings and character constants.

By analyzing the program, select the type of the program identifier and the frequency of their occurrence to identify the program [11]. The identifiers in the program are divided into two categories: system predefined identifiers and user-defined identifiers. System identifiers include keywords, operators, and library function names. User identifiers include user defined variables, numbers, and user-defined function names. Through the above statistics, the attributes such as the total number and vocabulary of identifiers can be obtained.

3.2. Spliting Program

In order to get the type and number of various identifiers in the program, the program needs to be split. The program can be completely decomposed by searching, counting and removing the identifiers one by one. First, the system operators are stored in the database in the correct order, and the operators are separated according to this certain order, which can improve the efficiency. For example, the operator '++' should be separated first and then the operator '+' is separated. After counting and removing the operators, it replace the operators with spaces to prevent the other parts

from joining together. For example, the statement "int sum=a+b;", after removing the operator, will become "int sumab". If it is not replaced with a space, and then no longer can distinguish three attributes of sum, a, b. So the space replacement operator is necessary. And it counts the types and numbers of other identifiers in the same way. In the end, the values of all key attributes are obtained and combined to form an attribute vector, then it calculate the distance between attribute vectors to get the value of program similarity.

3.3. Computing the Distance of Attribute Vector

The feature vector model can be used as a vector expression of source code. Some feature attributes form space vector to represent source code information, and it transforms the source code into a structured data form which can be easily processed by computer. The distance of attribute vector includes two parts: the angle cosine of attribute vector and the ratio of length of attribute vector.

The angle cosine between the feature vectors is calculated, as shown by formula (1), where *m* represents the dimension of the vector.

$$simCos(V_i, V_j) = \cos\theta_{ij} = \frac{\sum_{k=1}^{m} V_{ki} \times V_{kj}}{\sqrt{\sum_{k=1}^{m} V_{ki}^2} \times \sqrt{\sum_{k=1}^{m} V_{kj}^2}} \quad (1)$$

When the angle between the two vectors is small, the cosine function value is larger, and the calculated similarity value is also larger, which is generally more than 0.9. Therefore, it is not comprehensive to identify the program similarity only by the spatial angle cosine of attribute vector.

The ratio of the length between the attribute vectors is also a quantitative index of the program similarity. For example, the vector V_1(1, 1, 1) and vector V_2(2, 2, 2) are in the same angle with the vector V_3 in space, but the vector length is different, so the programs they represent are not similar. In order to control the similarity between 0 and 1, the larger length of the vector is the denominator when it calculates the ratio of the length between the attribute vectors, which is shown in formula (2).

$$simLen(V_i, V_j) = \min\left(\frac{\sqrt{\sum_{k=1}^{m} V_{ki}^2}}{\sqrt{\sum_{k=1}^{m} V_{kj}^2}}, \frac{\sqrt{\sum_{k=1}^{m} V_{kj}^2}}{\sqrt{\sum_{k=1}^{m} V_{ki}^2}}\right) \quad (2)$$

4. LCS Algorithm

LCS is the longest common subsequence, which is the same character sequence with the longest length after two given strings are deleted zero or more characters, respectively. For example, the longest common subsequence of the string "abcfg" and "bffg" is "bfg". The principle is that the string of program statements is dynamically matched to find similar programs, and it is also known as structural measurement. By using LCS algorithm to identify similar codes, the source code can be used as the detection object, which can avoid the dependence on the compiler of the program, and can be compared intuitively with the similar code at the same time.

The LCS algorithm calculates the code similarity from the view of program text and structure. At first, the source program is preprocessed, then it is transformed into Token sequence by lexical analysis. Finally, it figured out the longest common subsequence of Token sequence.

4.1. Common Subsequence

The location of the characters in the subsequence can be discontinuous in the original string, but the order of characters must be consistent with the characters in the original string [8]. In general, there are multiple common subsequences. The mathematical definition of the common subsequence of strings X and Y is like the formulas (3) - (8):

$$sub(X) = sub(x_0 x_1 \cdots x_{m-1}) = x_{i0} x_{i1} \cdots x_{ik} \quad (3)$$

$$sub(Y) = sub(y_0 y_1 \cdots y_{n-1}) = y_{j0} y_{j1} \cdots y_{jk} \quad (4)$$

In formulas (3) and (4), the following conditions are satisfied:

$$k < \min(m, n) \quad (5)$$

The character subscripts in the subsequence sub(X) satisfy the following conditions:

$$0 \le i0 < i1 < \ldots < ik \le m - 1 \quad (6)$$

The character subscripts in the subsequence sub(Y) satisfy the following conditions:

$$0 \le j0 < j1 < \ldots < jk \le n - 1 \quad (7)$$

If the characters which have the same subscript in the sub(X) and sub(Y) are equal, the sub(X) or sub(Y) is a common subsequence of X and Y, that is:

$$x_{it} = y_{jt}, \quad 0 \le t \le k \quad (8)$$

The longest common subsequence is longest in all common subsequences, and it may also exist more than one.

4.2. Program Token

Lexical analysis is the first step in the compilation process. The main task of lexical analysis of program is to recognize words one by one from the source program according to the lexical rules of language. The source program in string form is converted into word form, and each word is converted into their internal representation, which is called token, and the lexical rule is checked. For a complete lexical analysis module, it mainly includes three parts: input, processing, output. the part of input is advanced language source program. the part of output is Token List, and Token is an internal representation of a word, usually composed of two parts: word category and semantic information. This category is used to distinguish different kinds of words, usually can be encoded with integers,

and semantic information is also dependent on the convenience of future processing (as the content of words).

Due to the program contains user-defined identifiers, if the input strings are the standard program and student program, the accuracy of the result will be not high. So before the detection, the source programs need to be transformed into the Token sequences, and the subjective codes of program statements are eliminated such as user-defined identifiers. For example, the program statement "*s++ = 0" is transformed into the Token sequence "<*>, <*var*, s>, <++>, <=>, <*num*, 0>". In the Token sequence, *var* is user-defined variables and strings, *num* is numbers, *func* is the name of user-defined function. After processing the program statement, the sequence of "* *var* = *num*" was obtained. In order to further improve the accuracy of the result, the sequence "* 1 = 2" can be obtained by using the number 1-3 to replace *var*, *num* and *func*. After the above processing, the standard program and the student program become two new program strings, and the accuracy of the detection result is improved when the input becomes the two new program strings.

4.3. Calculate the Length of the LCS

The length of the two program strings is *m* and *n* respectively. The array *len(m+1, n+1)* records the length of all the longest common subsequences, The array *sign(m+1, n+1)* records the order number of the sub-problem from which *len(i, j)* is obtained. The first row and column of the two arrays are initialized to zero owing to the need of this algorithm. The problem of the algorithm can be divided into three categories, and the recursive formula is as follows where *strA(i)* is the *i* character of the string *strA* and *strB(j)* is the *j* character of the string *strB*:

$$sign(i,j)=\begin{cases}1, & strA(i)=strB(j);\\ 2, & strA(i)\neq strB(j),\quad len(i,j-1)>len(i-1,j);\\ 3, & strA(i)\neq strB(j),\quad len(i,j-1)<=len(i-1,j).\end{cases} \tag{9}$$

The problem is decomposed into three sub-problems. The length of the longest common subsequence is computed by recursive calls, as shown in formula (10):

$$len(i,j)=\begin{cases}len(i-1,j-1)+1, & sign(i,j)=1;\\ len(i,j-1), & sign(i,j)=2;\\ len(i-1,j), & sign(i,j)=3.\end{cases} \tag{10}$$

As mentioned above, if *strA(i)* and *strB(j)* are the same, *len(i,j)* is the length of the longest common subsequence of *strA(1...i-1)* and *strB(1...j-1)* plus 1, and it is marked as the first sub-problem. If the two characters are different, the length of the longest common subsequence is the maximum value of *len(i,j-1)* and *len(i-1,j)*; and if *len(i,j-1)* is greater than *len(i-1,j)*, it is marked as the second sub-problem, otherwise it is marked as the third sub-problem.

4.4. Calculate Similarity

The concept of similarity comes from the requirement of source code similarity measurement. Similarity is used to quantify the degree of correlation between two source codes. It is usually represented by a numeric *sim* ($0\leq sim\leq 100\%$). The larger the value, the higher the degree of similarity between the two source codes. On the contrary, the smaller the value, the lower the degree of similarity between the two source codes. It calculate the similarity between the standard code and the detected code [12], as shown in formula (11):

$$simLcs=\frac{2*length(LCS)}{m+n} \tag{11}$$

Where *m* and *n* represent the length of the two program strings, and *length (LCS)* represents the length of the longest common subsequence.

5. Weights of the Algorithms

In this study, it calculate the similarity of the codes by using the attribute counting method and the LCS algorithm, in which the calculation result of the attribute counting method includes the angle of the attribute vector and the length ratio of the attribute vector, so there are three results: the cosine value of the angle between attribute vectors *simCos,* the length ratio between attribute vectors *simLen* and the LCS algorithm similarity *simLCS*. The three kinds of static evaluation methods have their own advantages and disadvantages. In order to judge the score of the program more accurately and get more precise automatic evaluation results, the weights need to be reasonably distributed among the three similarity values. The method is shown in the formula (12):

$$sim=\omega_1\times simCos+\omega_2\times simLen+\omega_3\times simLCS \tag{12}$$

The reasonable weight distribution is based on the fitting degree of automatic evaluation and teacher evaluation. It follows the steps:

1) Collect simple codes which are composed of 173 student codes and 173 reference codes provided by the teacher, and the similarity between student codes and teachers' standard codes was calculated by the three methods. Considering the flexibility of programming, the teacher's reference code has many versions, and the student program's similarity calculation takes the maximum of the similarity degree compared with the multiple reference programs, and it obtains $S_{system}(i)$ in the formula (13).

2) The teacher marks the student program code, in which the score adopts the percentile system. And it takes the score as the reference.

3) An enumeration algorithm is used to distribute the weights in the similarity values of the three algorithms in units of 1%. After selecting a certain weight combination, it calculates the automatic evaluation scores of 173 student programs, which are also rated on a percent basis. Finally, the absolute error standard deviation is calculated. The formula is as follows:

$$AE=\left|S_{system}(i)-S_{teacher}\right|,\quad 1\leq i\leq n \tag{13}$$

$$MAE=\frac{1}{n}\sum_{i=1}^{n}AE \tag{14}$$

$$DAE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(AE - MAE)^2} \quad (15)$$

where S_{system} is the automatic evaluation result of the system, $S_{teacher}$ is the teacher's evaluation result, n is the number of the programs, and there are 173 samples in this experiment, AE is absolute error, MAE is mean absolute error and DAE is absolute error standard deviation. MAE can better reflect the actual situation of the prediction error. DAE can further reflects the distribution of prediction error.

The algorithm pseudo code for determining the weight is as follows. In all variables, minStdDeviation is the smallest standard deviation. w1, w2 and w3 is the weight of attribute cosine, attribute length and LCS respectively.

```
minStdDeviation = MAX;
for（w1=0; w1<=100; w1++）
   for(w2=0.00; w2<=100-w1; w2++) {
      w3 = 100-w1-w2;
      for (i=0; i<173; i++) {
         scoreSystem[i]=w1*(SimCos[i])+w2*(SimLen[i])
                  + w3*(SimLCS[i]);
         AE[i] = abs(scoreTeacher[i]-scoreSystem[i]);
      }
      DAE = CountStandardDeviation(AE, 173);
      if (DAE < minStdDeviation) {
         Record this combination of weight;
         minStdDeviation = DAE;
      }
   }
```

4) It takes the minimum value of the absolute error standard deviations of all weight combinations and get the most reasonable weight values. As shown in Table 1, in all of 5151 weight combinations, the maximum absolute error standard deviation is 11.26, and the minimum absolute error standard deviation is 5.60. Finally, the cosine value of the angle between attribute vectors *simCos*, the length ratio between attribute vectors *simLen* and the LCS algorithm similarity *simLCS* is determined, which are 39, 17 and 44 respectively.

Table 1. *The DAE of the weight combination.*

Sequence number	Weight of Attribute Cosine	Weight of Attribute Length	Weight of LCS	DAE
1	39	17	44	5.6027976
2	39	18	43	5.6044927
3	40	16	44	5.6045443
4	40	15	45	5.6069386
5	39	16	45	5.6069387
6	38	19	43	5.6080240
7	40	17	43	5.6083072
8	39	19	42	5.6087979
9	38	18	44	5.6089869
10	38	20	42	5.6106077
……	……	……	……	……
5149	0	2	98	11.0691197
5142	0	5	95	10.7972424
5143	2	0	98	10.8692535
5150	0	1	99	11.1625563
5151	0	0	100	11.2563356

6. Experimental Result

In accordance with the above algorithm, we use Java language to develop the "C/C++ programming online exercise and examination system". The flow chart of the program evaluation is shown in Figure 1.

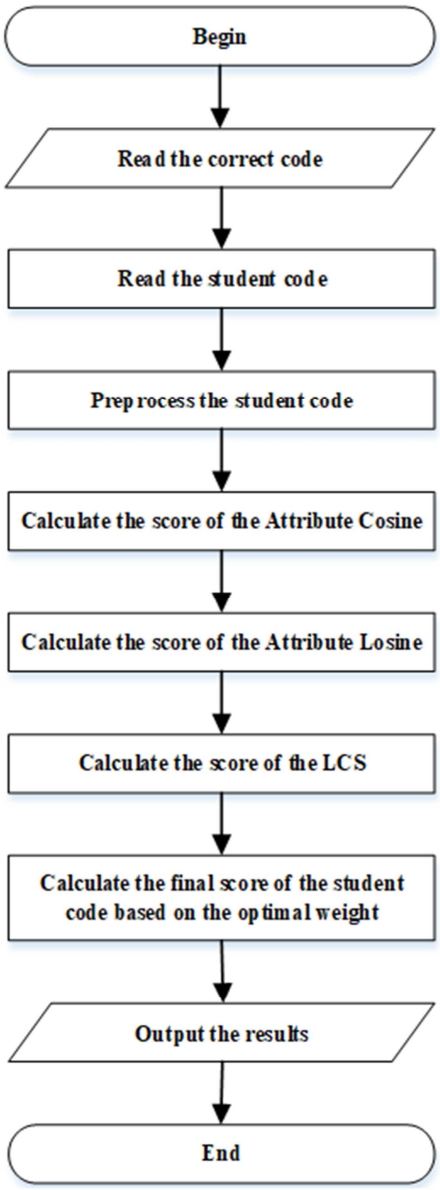

Figure 1. *Flow chart of the program evaluation.*

Then one class of Department of Computer Science in TJUFE carried out the mid-term examination with the system. We selected 100 student programs randomly and got the score of the system evaluation. At the same time, teachers are requested to make an artificial judgement on these 100 programs. Figure 2 compares the automatic scoring of the 100 codes with the teacher's score. It can be seen that the absolute error of the system automatic score and the teacher's score is less than 10%. This shows that the algorithm in this paper has higher accuracy in automatic program evaluation.

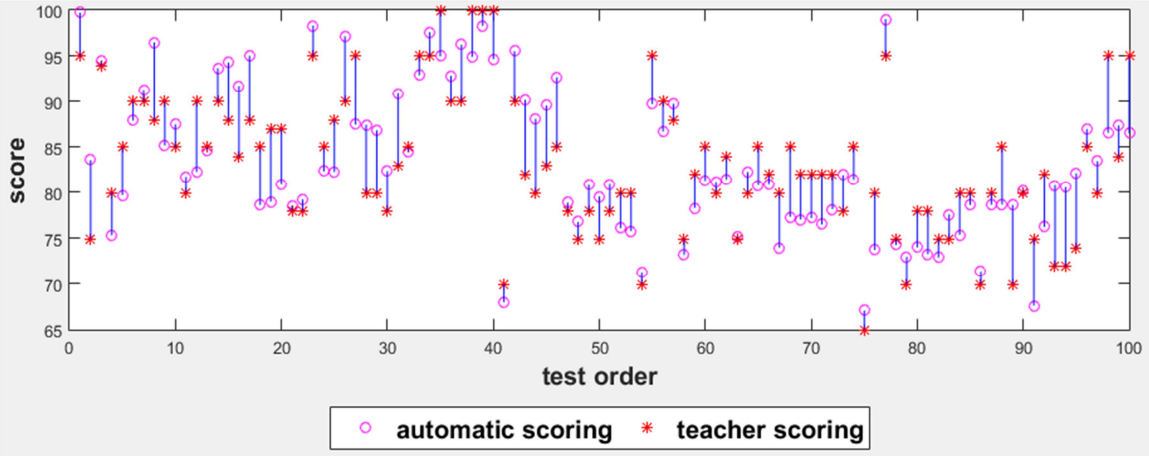

Figure 2. *Comparison of system automatic score and teacher score.*

7. Conclusion

The program automatic scoring method based on static analysis makes up for the deficiency of dynamic evaluation method and has important theoretical and practical value. Meanwhile it improves the attribute counting method and LCS algorithm. This method is not only applicable to the evaluation of C/C++ program, but also can be used to judge other programming languages such as Java.

However, it is important to note that the evaluation of the program code is subjective, and the criteria of evaluation of each school may be different. So the optimal weight may be different, and some test data should be trained to find the appropriate optimal weight. On the other hand, the correct code of each test question can also be given more. The score of student program can be selected the highest similarity with these correct codes.

Acknowledgements

This thesis is supported by the undergraduate research training program of Tianjin University of Finance Economics. Project name is automatic scoring method for programming questions based on semantic understanding.

References

[1] Basit H A, Puglisi S J, Smyth W F, "Efficient token based clone detection with flexible tokenization," Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2007[C]. Dubrovnik: ACM, 2007, pp. 513-516.

[2] YU Hai-ying, "Research and Implementation of Program Code Similarity Measurement", Computer Engineering, 2010, 36(4), pp. 45.

[3] GU Ping, ZHANG Feng, ZHOU Hai-tao, "Method of Program Source Code Similarity Measurement, Computer Engineering, 2012, 38(6), pp. 37.

[4] CUI Shuning, WU Ning, YE Dan, "Grade C++ code judge with constructing abstract syntax tree model," Journal of Computer Application, 2015, 35(S1), pp. 183.

[5] Wang Tiantian, "Research on Program Recognition Approach Based on Structural Semantic Similarity," unpublished.

[6] ZHANG Li-ping, LIU Dong-sheng, LI Yan-chen, "AST- based code plagiarism detection method," Application Research of Computer, 2011, 28(12), pp. 4616.

[7] Kim J, Choi H, Yun H, "Measuring Source Code Similarity by Finding Similar Subgraph with an Incremental Genetic Algorithm," Proceedings of the Genetic and Evolutionary Computation Conference 2016, Denver: ACM, 2016, pp. 925-932.

[8] LIU Yun-long, "Token-based structured code matching homology detection technology," Application Research of Computers, 2014, 31(6), pp. 1841.

[9] LIU Yun-long, "A Homology Detection Technology Based on Improved Edit Distance and LCS," Journal of Beijing Institute of Technology, 2017, 37(2), pp. 168.

[10] ZHANG Peng, "Research and Realization of Recognition Method on C Program Similar code," unpublished.

[11] Jones E L, "Metrics based plagiarism monitoring," Proceedings of the sixth annual CCSC northeastern conference on the journal of computing in small colleges, 2001[C]. Middlebury: Consortium for Computing Sciences in Colleges, 2001, pp. 253-261.

[12] ZHANG Jiujie, WANG Chunhui, ZHANG Liping, "Clone code detection based on Levenshtein distance of token," Journal of Computer Application, 2015, 35(12), pp. 3536.