A Similarity Detection Platform for Programming Learning

Yuanyuan Li, Yu Sheng, Lei Xiao and Fu Wang
School of Information Science and Engineering, Central South University, Lushan South Road, Changsha, China

Keywords: Similarity Detection, Structure-Metric, GST Algorithm, Sub-Graph Isomorphism.

Abstract: Code similarity detection has been studied for several decades, which are prevailing categorized into attribute-counting and structure-metric. Due to the one fold validity of attribute-counting for full replication, mature systems usually use the GST string matching algorithm to detect code structure. However, the accuracy of GST is vulnerable to interference in code similarity detection. This paper presents a code similarity detection method combining string matching and sub-graph isomorphism. The similarity is calculated with the GST algorithm. Then according to the similarity, the system determines whether further processing with the sub-graph iIsomorphism algorithm is required. Extensive experimental results illustrate that our method significantly enhances the efficiency of string matching as well as the accuracy of code similarity detecting.

1 INTRODUCTION

The combination of information technology and education been increasingly applied to modern teaching. For the programming course in computer Science and Technology, we developed a learning platform to help students improve the ability of programming skills, and also help teachers to improve their teaching efficiency. With the help of our platform, teachers can assign and check homework, issue course news or organize examinations online, and the students can also complete their task online. And therefore plagiarism becomes a big headache of teachers. If teachers check each program manually, it will cost much time and effort, and if students modify the program slightly, the task of program check becomes more difficult.

To address this issue, code plagiarism checking has been studied widely, mainly focus on how to compute the similarity of two program code and determine whether plagiarism exisits. The attribute counting method in checking code plagiarism is firstly put forward by Halstead (M. H. Halstead, 1977), and using structure measurement techniques to calculate the code similarity was presented by Verco and Wise (K. L. Verco and M. J. Wise, 1996). Through investigation we find out that most mature anti-plagiarism system adopt the string matching method to compare the code structure (Donaldson et al., 1981; G. Whale, 1990; D. Gitchell and N. Tran, 1999; Michael J. Wise, 2003). The systems based on such method can run efficiently, and can be implemented easily; however the disadvantage is such systems can't make accurate detection in complex copying method. This paper studies the related algorithms and techniques, and designs a similarity detection method, which combines string matching algorithm and subgraph isomorphism algorithm.

2 CODE SIMILARITY DETECTION OVERVIEW

Code similarity means that the degree of similarity between one program and another program.

2.1 Code Plagiarism Description

Programming language course is a very practical course, and extensive programming exercises are necessary to improve students' programming ability. However some students copy other students' source code or just simply change the name of variables or functions, which lead to plagiarism. Plagiarism waste teachers' effort, and can not lend any help to improve the students' programming ability. Faidhi and Robinson (J. A. W. Faidhi and S. K. Robinson, 1987) divided code plagiarism into seven levels. L0: not make any modifications to the source code; L1: only modify the source code comments; L2: modify identifiers of the source code, such as the name of the functions,

Li Y., Sheng Y., Xiao L. and Wang F..
A Similarity Detection Platform for Programming Learning.
DOI: 10.5220/0005490304800485
In *Proceedings of the 7th International Conference on Computer Supported Education* (CSEDU-2015), pages 480-485
ISBN: 978-989-758-107-6

macros and variables ; L3: change the position of variables; L4: replace the function call with the function body, representing a decrease of function; L5: modify the program statements, such as i++ becomes i + = 1; L6: modify the program control logic.

2.2 Code Similarity Definition

Obviously, 100% means completely copying. The plagiarism relationship with two programs is measured by code similarity. The higher the similarity is, the greater the possibility of plagiarism. T. Yamamoto, M. Matsushita, T. Kamiya and K. Inoue (J. A. W. Faidhi and S. K. Robinson, 1987) give the definition of similarity of the two software systems. For two software systems A and B, A consists of the elements a1, a2, a3, ..., am, represented by the set: {a1, a2, a3, ..., am}.Similarly, B elements b1, b2, b3 , ... bn, represented by the collection {b1, b2, b3, ..., bn}. Here, a1, a2, a3, ..., am and b1, b2, b3, ..., bn can be the file or the line of a program in software systems A and B.

Suppose we are able to calculate the matching between ai and bj (1 <= i <= m, 1 <= j <= n). The collection of all match (ai, bi) is is represented by Rs, the similarity S is defined as follows:

$$S(A,B) = \frac{|\{a_i \mid (a_i,b_i) \in R_s\}| + |\{b_i \mid (a_i,b_i) \in R_s\}|}{|A| + |B|} \quad (1)$$

As shown in Eq.(1), this definition indicates that the similarity between A and B is a ratio, which is obtained by the sum of A' size and B' size divided by the size of Rs. if Rs is small, then S will be smaller ,if the RS is the empty set, then S = 0. When A and B are the same, S = 1.

3 CODE SIMILARITY DETECTION METHOD

Code similarity detection methods are divided into two categories: attribute-counting technique and structure-metric technique. Attribute-counting technique is proposed and used in code detection firstly. Program code has its features, such as: the number of lines of code, the number of variables, operators, the number of control conditions and the number of cycles. Attribute-counting technique should figure out the number of the unique attributes of a program. Obviously different programs have different result of attributes statistic, and the result of the attributes statistics of the same or similar program code should be similar. Verco and Wise have proved that a anti-plagiarism detection system based on attribute-counting technique just be well work in the situation that the two programs are same or very similar, it does not work for the programmers with a little programming experience who could make several modifications to the source code. The structure-metric method is used to determine whether the two procedures are similar by comparing their structural information. It is well known that, for programmers, it is easy to change the attributes of a program, but the structure of the program is very difficult to change, otherwise it can't be called as plagiarism.

At present, the structure-metric method based on string matching algorithm is widely used in most anti-plagiarism systems. This method has two key points. The first point is how to analyze the structure of the program code and converse the code into a string. The second point is how to choose a string matching algorithm to compute the similarity.

3.1 Code Plagiarism Description

The string matching algorithm in the Anti-plagiarism detection system is used to calculate the similarity of the program code. The plagiarism refers to the situations that the students simply make some modifications to some of the variables, change the position of some functions; and therefore the string matching algorithm must be able to detect these cases. String matching must be possible to find the longest match, due to the fact that some short match exists even if there is no plagiarism. String matching algorithm for plagiarism detection is not simply find the position of the mode string, but to find the set of all exact matches in the two strings; the proportion of the size of exact matched strings to the size of total string can be used to determine the level of similarity. String matching algorithm should mark the location of the longest matches to facilitate the detection. There are many effective string matching algorithm such as: LCS (longest common substring), Levenshtein distance (Michael Gilleland, 2007), Heckel algorithm (Michael J. Wise, 1992), dynamic programming and GST. Given the features of anti-plagiarism system, in this paper, we use the GST algorithm which has a better accuracy. The processes of GST are as follows:

Step 1: defining MIN_MATCH_LEN, which should be equal to or greater than 1; the TILES is initialized as empty; S and T are not marked by default, which means the matching has not start yet.

Step 2: Find one or more maximum-matching string; initializing the maximum matching length max_match as MIN_MATCH_LEN; setting matches

to empty, and then repeatedly scanning and comparing the unmarked character in S and T. If the two characters are equal, then increase the matching length len. Repeating the process until the characters are not equal or the characters have been marked. If len equals to max, match,which means that we find a new match with the max_match size, and therefore we add the new match into matches. if len is greater than max_match which means the collections in the matches we found before are not the longest common strings, therefore we should reset the matches to empty, add the current maximum matching item into matches, and reset max_match to len. Repeating the process above until there are no unmarked character in S and T.

Step 3: If the matches generated in the second step is not empty, then add it to the set tiles and mark the characters in matches.

Step 4: Repeating the second and the third step. And the algorithm is ended.

3.2 Subgraph Isomorphism Algorithm

Subgraph isomorphism problem is an NP-hard problem (M. Garey and D. Johnson, 1979), however several decades of studies have shown that some optimization algorithm is relatively fast,such as the backtracking search algorithm (Evgeny B. Krissinel and Kim Henrick, 2004) proposed by E.B.Krissinel in the University of Cambridge, and only in rare cases such algorithm is slow. The process of the backtracking search algorithm is as follows:

```
function BackTrace() {
   if !Extendable(queue)
      return
  end if
   node vi = PickVertex()
    X = GetMatchedNodes(v);
  for all ui in X
     map.put(vi,ui);
    If Validate() then
        Marked(vi);
        Marked(ui);
        n=n>map.size()?n:map.size();
        UpdateQueue(vi);
        Backtrace();
    else
        map.remove(vi,ui);
    end if
   end for
 }
```

4 THE REALIZATION OF THE SIMILARITY DETECTION

The code similarity detection designed in this article works in the process shown in Fig.1. Preprocess the program code A and B and calculate the similarity with the GST algorithm. Then according to the similarity, the system determines whether further processing with the subgraph iIsomorphism algorithm is required. The algorithm finally return the similarity.

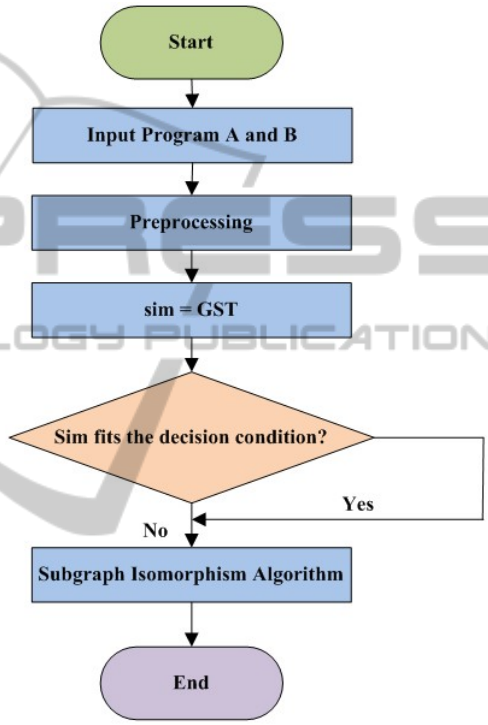

Figure 1: Similarity detection process.

4.1 Preprocessing

Due to the readability of the program code, there are always several comments or text prompts in the code. The programmer can just modify these comments or text prompts. And preprocessing module is used to filter out such useless information and avoid plagiarism level L1. Preprocessing module will first scan the source code and delete the comments and empty lines, as well as text prompts. Besides such useless information, preprocessing module also should delete the header files, because copycat add a lot of irrelevant header files will confuse similarity detection; mostly the same but a lot of program header file, for example, in C language, programmer will use these standard header files such as stdio.h, stdlib.h, file.h, math.h. For that even if there is no plagiarism the

header files could be the same, so the header file information is not necessary in similarity detection.

4.2 GST Algorithm Implementation

1. Tokenization

Tokenization means converse the source code to a tag string by lexical analysis, which can facilitate the string matching. The concrete realization of tokenization is changing the program code into an intermediate object called LangGrammerElem. LangGrammerElem is divided into four types: SINGLE, METHOD, LOOP and CONTROL. SINGLE elements include single-line elements, such as variable declarations, assignments and function calls. METHOD elements include functions, including the main function. Loop elements include for, while, do-while and other loop statement. Control elements include if, else, switch and other branch control statements. LangGrammerElem is a recursive structure because METHOD,LOOP and CONTROL elements may contain more than one SINGLE elements. Table 1 shows an example of tokenization of a C program.

Table 1: Example of Tokenization.

C Source Code	Tokenization
1 void main()	MAIN{
2 {	
3 int number[20],n,m,i;	DECLAREDECLARED ECLARE DECLARE
4 scanf("%d",&n);	METHOD_CALL
5 scanf("%d",&m);	METHOD_CALL
6 for(i=0;i<n;i++)	FOR{
7 scanf("%d,",&number[i]);	METHOD_CALL }
8 move(number,n,m);	METHOD_CALL
9 for(i=0;i<n-1;i++)	FOR{
10 printf("%d ",number[i]);	METHOD_CALL }
11 printf("%d",number[n-1]);	METHOD_CALL
12 }	}

2. Similarity Calculation

Similarity is using the result of string matching algorithm-GST. GST is using the BR brute force algorithm to compare two strings. However, we can use KMP algorithm to optimize it. Or we can replace GST with the RKR-GST algorithm which is better and also based on the the famous string matching algorithm,Karp-Rabin (Michael J. Wise, 1993).

4.3 Decision-making Process

Using the GST algorithm mentioned above we can get a similarity of two program codes. In this paper, to get a more precise result, we design a decision-making module to decide whether we should use the subgraph isomorphism algorithm to detect the similarity. In our system, users can configure the similarity threshold max and min. For example, we set max to 0.9 and min to 0.5. If the similarity got by GST algorithm is greater than max(0.9), we can make a conclusion that there is plagiarism; in contrast, if the similarity is less than min(0.5), we can assume that there is no plagiarism. Otherwise, if the similarity is between max and min, this situation is suspected of plagiarism and therefore further detecting via the subgraph isomorphism algorithm is necessary. The setting of similarity threshold max and min are considered as follows, on one hand we should try to ensure the accuracy of the detection of plagiarism, the other is that we should assures the high efficiency of the system. For example, if min is too high the accuracy of the detection would be decreased. And if min is too small the efficiency of the system would be reduced. Generally, max=0.9 and min=0.5 is relatively modest according to the accuracy and efficiency of the system.

4.4 Subgraph Isomorphism Algorithm Implementation

Subgraph isomorphism algorithm implementation includes two parts. The first part convert the structure of the program into a dependency graph. The second is the subgraph isomorphism calculation.

1. The Dependency Graph Generation

For the programmers, no matter how they modify a program they will not change the output of the program. If they change the results, such plagiarism does not make any sense. And the output is determined by the program's data and its structure. The program's data, namely the variables in the program, can be directly assigned a value. Also it is indirectly assigned by another variable, which is a dependency relationship of the variables. The structure of a program includes sequential process, branching process and loop. Program dependence graph (PDG) can fully represent the data and the structure of a program, and therefore we use PDG in our system. In PDG, the node represents a programming statement, and an edge represents a data dependency and control flow.

Table 2: Types of program dependence graph node.

Type	Description
Declare	Declaration of variables

Assign	Assignment of variables,such as =,+=,++,--
Control	if,else,while,for,do-while,switch
Jump	goto,break
Call	Function call
Return	return
Case	Case or default in switch

The nodes in PDG are divided into the types shown in Table 2. The PDG edges are divided into two types: control dependency edges and data dependency edges. Control dependency edges represent controlling relationship, such as if, else or while control. Data dependency edges represent that there are data dependencies between nodes.

The following program is the source code for summing.

```
int i;
int sum = 0;
for(i=0;i<=100;i++) {
   sum+=I;
}
```

And Fig. 2 shows the PDG after conversion.

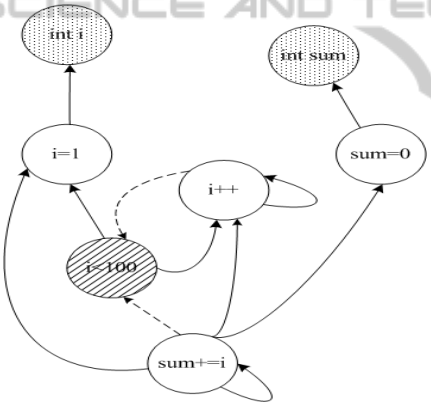

Figure 2: Program Dependence Graph.

2. Similarity Calculation

Using subgraph isomorphism algorithm on PDG and we can get the maximum common subgraph, and then we can calculate the ratio of nodes number in the maximum common subgraph(represented with T) to that in the pattern graph(represented with P). And the similarity sim=|T|/|P|.

5 EXPERIMENTAL ANALYSIS

In this study, we tested our system by two groups of program codes. The first group contains five questions fetched from the programming language platform. And each question is finished independently by nine students. During the similarity detection test, to any question, there are 9 source codes, and we compared these source codes in pairs. That is, to one question, there are 9X(9-1) /2 = 36 compare and a total of five questions generated 36X5 = 180 comparisons. In our experiments we set plagiarism threshold value max = 0.9, min = 0.5. The detection results are shown in Fig. 3.

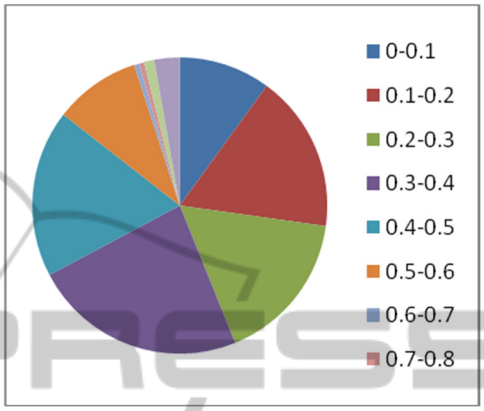

Figure 3: Experimental results.

The second The second group only contains one question, namely "Find out all the intimacy numbers less than 3000", and then the source code was modified in plagiarism way by 12 students respectively. Also we provide ten modifying references to the students as follows: (1) completely copy the original program; (2) modify the annotation; (3) alter the program format and blank lines; (4) change the name of variable; (5) adjust the location of the code statement; (6) adjust the variable declaration position; (7) change the location of the operand or operator in the expression; (8) change the data type; (9) add redundant code; (10) replace the control structure with a equivalent way. Finally only 2 of the 12 plagiarism samples had the similarity less than 0.9, 4 codes went through GST similarity calculation, and the rest 8 codes tested by subgraph isomorphism algorithm. We found that the subgraph isomorphism algorithm was more accuracy in program structure modifications than GST.

Similarity calculation based on string matching is widely used in the plagiarism detection system. However, it cannot effectively detect plagiarisms if adding numerous useless code or changing the code positions, due to the characteristics of the string matching algorithm. JPlag tried to achieve a high detection accuracy for the structure modification with a string matching algorithm and failed finally. From the course of experiment above, we came to the conclusion that the Similarity Calculation based on subgraph isomorphism algorithm was more accurate than that based on string matching algorithm. String matching

algorithm needs to find a matching set, which is greater than the minimum matching length, and reducing the minimum match length can increase the similarity of the GST algorithm. But if a minimum match length was too small, it would cause suspicion of plagiarism for some code without coping. This system used the GST and subgraph isomorphism algorithm to calculate the similarity, achieving a better accuracy compared to JPlag etc. for most copying means, and the efficiency was also close to the string matching algorithm.

6 CONCLUSIONS

In this paper, we use the well-known string matching algorithm-GST and subgraph isomorphism algorithm in the similarity detection system, and these classic algorithms were applied to practical applications. The detection processes were completed by four steps. We tested our system by two experimental procedures, of which the program source codes were submitted by real students. The first result shows that the code similarity detection system runs faster, with low accuracy. The second testing result demonstrates that the system could detect the all the plagiarism level defined by Faidhi and Robinson with high accuracy of nearly 90%.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (61202494), The teaching reform project of Hunan Province.

REFERENCES

Donaldson, L. John, Ann-Marie Laricaster and H. Paula Sposato. A Plagiarism Detection System. Twelfth SIGCSE Teachnical Symposium, St. Louis, Missouri, 1981:21-25.

G. Whale. Identification of Program Similarity in Large Populations [J]. The Computer Journal, 1990, 33(2):140-146.

D. Gitchell and N. Tran. Sim: A Utility for Detecting Similarity in Computer Programs [C]. In Proceedings of the 30th SIGCSE Technical Symposium, March 1999.

Michael J. Wise. YAP3: Improved Detection of Similarities in Computer Program and other Texts [J]. Department of Computer, University of Sydney, 2003.

M. H. Halstead. Elements of Software Science [J]. Elsevier computer science library, New York, 1977 (17):5-7.

K. L. Verco, M. J. Wise. Software for Detecting Suspected Plagiarism: Companng Structure and Attribute-Counting Systems [J]. Computer Science, University of Sydney, 1996:3-5.

J. A. W. Faidhi and S. K. Robinson. An Empmcal Approach for Detecting Program Similarity within a University Programming Environment [J]. Computers and Education, 1987, 11(1):1-19.

Michael Gilleland. Levenshtein Distance, in Three Flavors [J]. http://www.Merriampark.com/ld.htm.2007-4-18.

Michael J. Wise. Detection of Similarities in Student Programs: YAP'ing May Be Preferable to Plague'hag [J]. SIGSCI Technical Symposium, Kansas City, USA, March 5-6, 1992:268-271.

Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search [J]. Software-Practice and Experience 2004(34):591-607(DOI: 0.1002/spe.588).

Michael J. Wise. String Similarity Via Greedy String Tiling and Running Karp Rabin Matching [J]. Department of Computer Science, University of Sydney. December 1993.

M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness [J]. Freeman, 1979.