Санкт-Петербургский государственный университет

Кафедра системного программирования
Группа 21.М07-мм

Ван Тяньцзин

Средство автоматической проверки выполнения заданий по программированию с анализом качества кода и поиском плагиата

Toolkit for automatic checking of programming assignments completion with code quality analysis and plagiarism search

Отчёт по учебной (проектно-технологической) практике

Научный руководитель:
Доцент кафедры СП, к.ф.-м.н. Д.В. Луцив

Санкт-Петербург
2022

Abstract

In the field of source code plagiarism detection, how to detect the existence of common fragments or similar structure in source code and other source code is always a difficulty. The traditional method based on text comparison is particularly cumbersome for source code plagiarism detection. This thesis proposes a source code plagiarism detection scheme based on AST (Abstract Syntactic Tree) and Index information. The code block structure information generated by AST and the code block based HashIndex save the more complete source code information, and this form occupies less memory space, transmits faster, and retrieves faster than source code. The detection scheme can better protect the personal intellectual property rights and enterprise source code security and achieve the efficient plagiarism detection.

Keywords: Code plagiarism, Clone detection, Index

1. Introduction

The development of the Internet makes people's lives more and more convenient and faster, and the methods of obtaining information have become more and more abundant. People can search for huge amounts of information in a few seconds. However, everything has two sides. While the Internet brings convenience to people, it also provides great convenience to plagiarists.

Plagiarism is an unethical act of reusing someone else's work without explicitly identifying the original author (Hannabuss, 2001). In computer science education, the most common problem is source code plagiarism. This is a big problem that educators are eager to solve. Source code plagiarism has also received more and more attention in academia. Source code is available from a variety of sources. Such as the Internet, source code repositories, books, etc. In 2007, Nadelson proposed a study around a university academic misconduct problem (Nadelson, 2007). The study gathered the views of 72 academics on plagiarism and found 570 incidents of academic misconduct among bachelors and masters. Most incidents were "accidental and unintentional" plagiarism, involving 134 bachelors and 39 masters. Furthermore, academics have even suspected that theses submitted by students were plagiarized from the Internet.

In public utilities, source code plagiarism can cause many harms. Software developers acquire large amounts of code through the network. Many open-source codes have brought great convenience to this development mode. However, since open-source code adopts a variety of licenses, each of which has different constraints, blind reuse of software is likely to bring serious legal risks to enterprises. The long-running infringement lawsuit between Oracle and Google is about duplication of software code. Oracle sued Google in 2012 over software code plagiarism, and Oracle's charges against Google included 9 lines of code duplication. The lawsuit lasted for four years, and it didn't settle until 2016, although it ended with Oracle's failure. However, the legal risks caused by source code plagiarism will still cause huge losses to enterprises in the future [1].

The licenses of open-source software include MIT license, Apache license, GPL license, LGPL license, etc. [2]. Different license information gives code users different rights. For example, the GPL license has a typical contagious nature. GPL allows developers to freely use and modify open-source code. However, code modification and newly developed code based on the GPL license must also follow the GPL license to make the code public, and the modified and derived code is not allowed to be released and sold as closed-source commercial software. Due to the large number of enterprise software developers and codes, and the countless amount of open-source software, the manual detection of various licenses and plagiarism risks is almost impossible. Therefore, in order to help enterprises avoid potential legal risks, a method or system that can detect and report potential software source code plagiarism risks is also required.

The goal of this thesis is to develop a more efficient solution to source code plagiarism based on index information, which considers not only the similarity between codes, but also the relationship of program context. Pursue a source code plagiarism detection system that is faster in speed, shorter in time, and consumes less memory. The main tasks are:

(1) Code information extraction

(2) Local mapping table construction

(3) Building a global feature information table

(4) Upload code block information

(5) Plagiarism detection Toolkit

The structure of the thesis presents the existing methods and solutions for part 2. Part 3 is the strategy in which anti-plagiarism detection can be done. Part 4 is the solution proposed in this thesis. Part 5 is the concrete implementation of the solution proposed in this thesis. Part 6 is the implementation result and result analysis of the solution proposed in this thesis. Part 7 provides a summary and future work for the proposed solution, as well as the acknowledgments and references at the end of this thesis.

2. Related work and background

In terms of software plagiarism detection, many methods and systems have been researched and developed [3], and the more commonly used detection systems include JPlag [4], MOSS (Measure of Software Similarity) [5], YAP3 (the third version of Yet Another Plague) [6], GPLAG [8], etc. There are mainly two types of source code plagiarism detection methods: detection technology based on property statistics and detection technology based on structural measurement.

(1) Property Statistics Technology:

Only the properties of the program are statistically processed, regardless of the internal structure of the program. Reference [11] first used property statistics technology for source code plagiarism detection. Halstead counts 4 properties, namely N1 is the number of occurrences of the operator, N2 is the number of occurrences of the operand, n1 is the number of different operators, n2 is the number of different operands, then the source program is converted into a quaternion (n1, n2, N1, N2), when the quadruples of the two source programs are the same, it is determined as plagiarism. Different academics [12-14] have introduced many metrics, but none of these attempts have an obvious effect. Since property statistics technology measures the whole program, most of the structural information is ignored, only increasing measurement metrics does not substantially improve the detection results of the program. This method has a good effect on the code plagiarism detection of directly copied and pasted code. When the structure of the source program is modified or the code segment is added or deleted, the effect is not good, and there are many underreports.

(2) Structural measurement techniques:

According to the internal structure information of the program to determine whether two programs are similar, the structure measurement technology is a subset of the code clone detection, including ①Text-based method ② Token-based method ③AST (Abstract Syntax Tree) ④Program Dependency Graph (PDG), etc.

① The Text-based plagiarism detection starts from the text structure of the source code to detect the similarity of the program. A representative tool such as Dup [15] uses each line of the source code as a comparison unit and calculates the similarity of the code by building a suffix tree. The reference [10] uses a sliding window to segment code lines and proposes a more efficient detection method with index-base. Text-based reflects less syntactic and semantic information of programs and has lower detection accuracy.

② Token-based detection is to convert the source code into the form of Token sequence, More typical in the field of clonal detection is the CCFinder of reference [7]. In the field of plagiarism detection, JPlag and YAP3 are both plagiarism detection systems of this type. They first convert the source code into Token sequences, and then use the RKR-GST (Running Karp-Rabin Greedy String Tiling) algorithm to calculate the code similarity. Token-based ignores the syntactic and semantic nature of the program, and only has a good effect on code plagiarism detection at the lexical level.

③ The detection method based on the program dependence graph builds the dependency graph of the program by analyzing the syntactic structure of the source code, the calling relationship of functions, etc. and detects the similarity of the code by the subgraph isomorphism. For example, the GPLAG system uses the subgraph similarity methods to calculate the similarity of the source code. Although accurate, PDG cannot be applied to large-scale scenarios due to the large amount of computation.

④ The plagiarism detection method based on AST converts the source code into AST, which can compute the similarity by finding similar subtrees. For example, reference [16] directly finds similar subtrees on AST to calculate the similarity of the syntax tree. In the field of clone detection, reference [17] developed the tool DECKARD to detect clone codes by clustering using similar subtrees. Tree structure based on AST is more expensive and it is difficult to apply in detection scenarios with large software.

The thesis proposes a code detection method based on AST and index information. Generated by abstract syntax tree code block structure information and hash index based on code block, more fully save the source information, does not affect the detection effect, this form occupies less memory space, the transmission speed is faster, the retrieval speed is also faster and it can automatically reduce the false positive rate based on statistical information, the detection results are more accurate.

3. Source code plagiarism strategy

According to Jones's definition of code plagiarism: a plagiarized program or an exact copy of the source code, or modified code obtained by applying various modification strategies (Jones, 2001). Plagiarists often adopt deliberate modification strategies to counter code plagiarism detection. The following modification strategies are mainly divided into the following categories from easy to difficult:

(1) Complete plagiarism. Complete plagiarism is the most direct and simplest form of plagiarism. The plagiarists get the code from his classmate or the Internet and submit it directly as his own code without any modification. Such plagiarism is simple and direct, very easy to be detected. It is the lowest plagiarism strategy.

(2) Modify the code comments. In order to increase the readability of the program language, programmers usually add some comments to mark when writing programs. These annotations are semantically unrelated to the program code and are ignored by the compiler. However, plagiarists often try to evade detection by modifying these annotations. With the guarantee that the program function is not changed, it is easy to get a new code that is quite different from the original code in the text by simply changing the code annotation. Although they compile to the same result, the code varies widely.

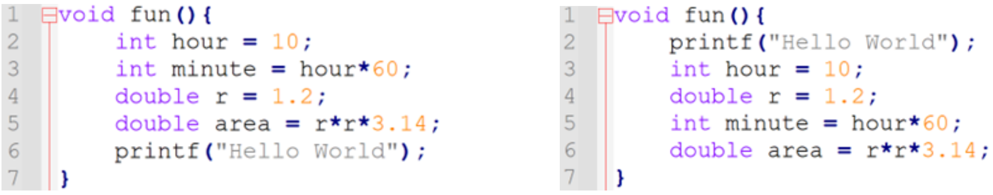
(3) Modification of code format. The C language uses a specific delimiter to separate declarations and separate code blocks (unlike the Python language, which is indented with spaces to denote code blocks), so there are almost no restrictions on the use of spaces. Everyone's coding style is different, and there are naturally many differences in the spaces included in the code. Plagiarists are likely to take advantage of this and drastically modify the style of the source code so that the modified code looks very different from the original. That is, although the two codes are quite different in text, the actual effect is the same.

(4) Identifier renaming. In a program language, the identifier refers to the naming of the reserved word (Reserved word) used by the programmer during the programming process. For example, constant, variable and function are all identifiers. In a program, any identifier defined by the programmer can be replaced without affecting the program syntax itself. That is, the plagiarists can modify the text by modifying the variable name without affecting the original running result of the program.

(5) Adjust the order of code blocks. Unlike natural language, code does not emphasize the ordering between blocks. In general academic thesis, there are constraints on the sequence of chapters. For example, the abstract must be written at the front of the article, the reference must be written at the end, the introduction must be written in front of the text, the conclusion must be written at the end of the text, and the results must be written behind the experimental method. The code is written without these restrictions, and the entry of the program can be written in any position of the program file. Plagiarists can take advantage of this feature of the programming language to adjust the order of the code blocks without affecting the normal operation of the program.

(6) Adjust the order of the statements within the code block. A certain paragraph of natural language often requires a logical order, and the order of sentences cannot be adjusted arbitrarily. A code block often implements the functions of several modules, they have no logical sequence requirements , and they are independent of each other. Therefore, plagiarists will take advantage of this feature of the code and readjust the modules in the code block, making the code look messy, but the meaning of the program expression has not changed.

As shown in Figure 3-1, the two pictures are quite different only from the text, but the meanings of the two code blocks are the same, but the order of the modules in the code blocks is different. In the code on the left, the fun function implements the logic of the three modules in turn: knowing the number of hours to calculate the number of minutes, knowing the radius of the circle to calculate the area of the circle, and outputting "Hello World”. In the code on the right, the order of implementation of the three modules is disrupted inside the function.



```
void fun(){
    int hour = 10;
    int minute = hour*60;
    double r = 1.2;
    double area = r*r*3.14;
    printf("Hello World");
}
```

```
void fun(){
    printf("Hello World");
    int hour = 10;
    double r = 1.2;
    int minute = hour*60;
    double area = r*r*3.14;
}
```

Fig. 3-1 The original code and the code after reordering statements within code blocks

(7) Change the constant value. In the programming language, in order to facilitate the management and maintenance of the programming language, some constant names are usually stated at the front of the code, and the values of constants in the program cannot be changed. In

many cases, the constant values are not exclusively invariant, but are changeable and replaceable. Common modifications include modifying the size of the array, modifying the constant accuracy, and adjusting the range of the constant. In this way, the plagiarist will change the constant value as another plagiarism method, by modifying the constant value of the code to increase the difference of the code. Although the constant value of the code is different, it can express the same or similar meaning under the requirements of the title.

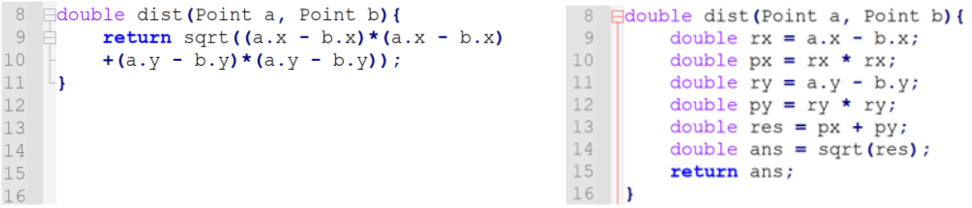
(8) Rewrite the expression. Like natural language, another common means of code plagiarism is to rewrite expressions. In a programming language, some expressions can be arbitrarily rewritten without affecting the meaning of the program expression. Common modifications include swapping variables, merging, or splitting expressions, etc.

(9) Data type replacement. In programming languages, many variable constants are often declared to store intermediate results. When declaring, the first thing to determine is the data type of the variable constant. Common data types in C language are int, short, long, float, double, bool, char, etc. The data type is not fixed and cannot be changed, and sometimes it can be replaced without affecting the meaning of the program expression. For example, many times int can be replaced by long or double, float can be replaced by double, bool can be replaced by int and so on. So, plagiarists may use this feature to modify the source code. Although there are differences between the codes, the meaning of the program expression is the same.

(10) Add redundant code. In natural language, each sentence of each paragraph has a specific meaning relative to the entire article, while in program code, not necessarily every line of code is valid, which means that there may be redundancy in the code. These codes will not affect the compilation and running of the program, nor will it affect the input and output of the program. Variables and constants that are not used in the program, expressions that do not change the value of any variables, functions that are not called, etc. are all redundant code. Therefore, plagiarists will choose to add some redundant code to the source code, making the modified code look very different from the source code.

(11) Reorganization of expressions. The modification of expressions is a kind of meter reading method that is difficult to detect, especially when merging or splitting expressions, many detection tools cannot accurately detect them. Recombining expressions greatly increases the differences between codes, whether in terms of text or from the logical structure of the program, but they are not affecting the results of program execution. In fact, for a problem, because of different coding styles of different students, some students will use one expression to solve the problem, while some students may split the expression into multiple expressions to represent. So, merging and splitting expressions is a plagiarism method that highly simulates coding, which is why detection tools are difficult to detect.

Figure 3-2 shows the original code and the code after restructuring expression. The two codes seem to be very different, but in fact the code on the right just splits the expression in the code on the left into multiple expressions. The meaning of the code expression is the same, both define a Euclidean distance function between the two points.



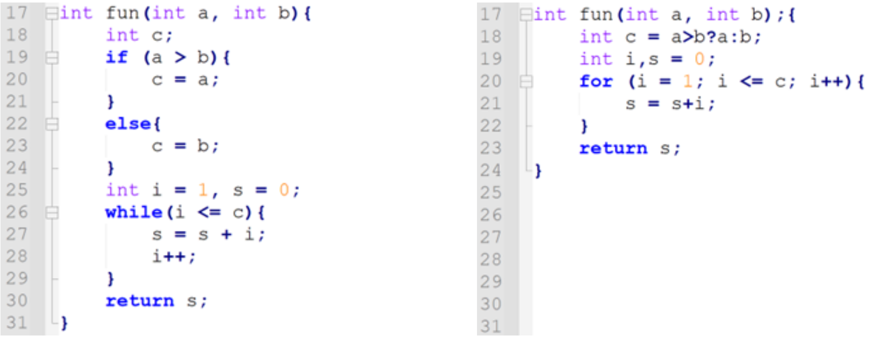
```
double dist(Point a, Point b){
    return sqrt((a.x - b.x)*(a.x - b.x)
    +(a.y - b.y)*(a.y - b.y));
}
```

```
double dist(Point a, Point b){
    double rx = a.x - b.x;
    double px = rx * rx;
    double ry = a.y - b.y;
    double py = ry * ry;
    double res = px + py;
    double ans = sqrt(res);
    return ans;
}
```

Fig. 3-2 The original code and the code after restructuring expressions.

(12) Code structure equivalent replacement, code structure equivalent replacement is a more "advanced" plagiarism method. Common modification methods include equivalent replacement of conditional structures, such as reciprocal judgment conditions, changing if conditions into trinocular operator, adding redundancy; the equivalent replacement of loop structures, such as rewriting for loops into while loops, rewrite the “while” loop into a do...while loop, etc. The code after the equivalent replacement of the code structure is often quite different from the original code, and it is difficult to be detected by the detection tool. Even if it is manually checked, it is difficult to determine whether there is plagiarized code.

Figure 3-3 lists two code segments. The two code segments are quite different. It is difficult to detect that there is code plagiarism, but the meanings of the codes are the same. The code on the left is done using the if...else... conditional structure, while the code on the right is done using the trinocular operator. The accumulation part of the loop variable i, the code on the left is completed by the while loop, and the code on the right is completed by the for loop.



```
int fun(int a, int b){
    int c;
    if (a > b){
        c = a;
    }
    else{
        c = b;
    }
    int i = 1, s = 0;
    while(i <= c){
        s = s + i;
        i++;
    }
    return s;
}
```

```
int fun(int a, int b);{
    int c = a>b?a:b;
    int i,s = 0;
    for (i = 1; i <= c; i++){
        s = s+i;
    }
    return s;
}
```

Fig. 3-3 The original code and the code after replacing equivalent structures

4. Methods overview

The method in this thesis includes two parts: local and service. The server side is responsible for building the code source, receiving detection requests and code feature information from the local, and performing code plagiarism detection calculations; the local side is responsible for extracting the feature information of the code to be detected, including code block structure information and hash index information, and reporting the information to the server side, and localized interpretation of the repeated information sent by the server. The method structure is shown in Figure 4-1.

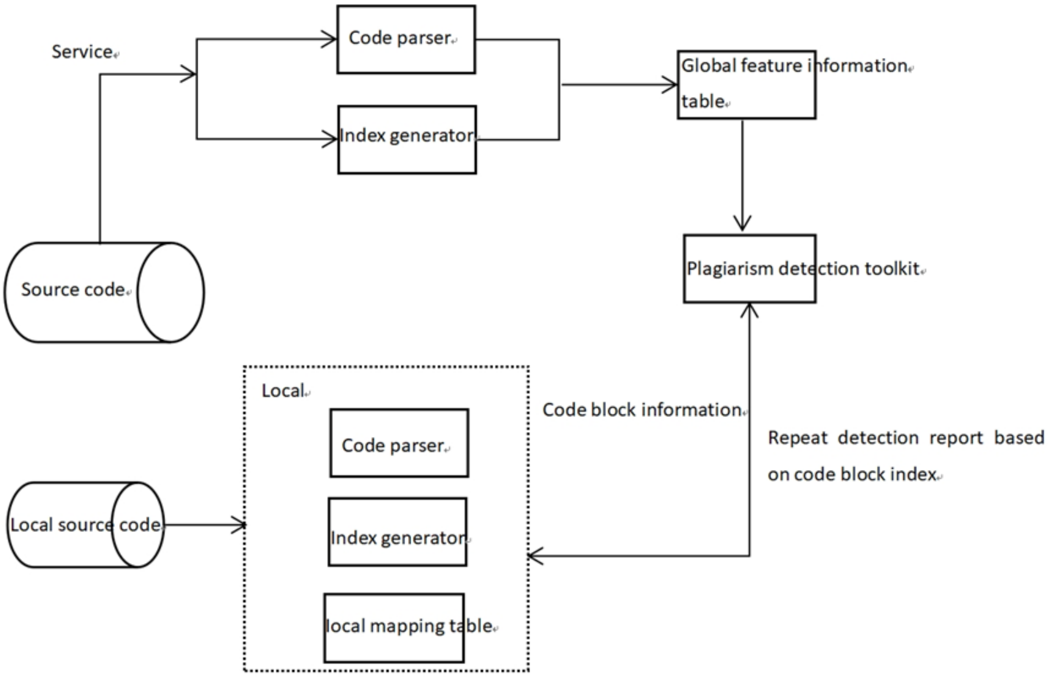


Fig. 4-1 Plagiarism Detection Method Structure

4. 1 Code information extraction

Code feature extraction is a common module for the server and local. Responsible for extracting code structure and hash index features with two subunits: code structure resolution and index information generation.

(1) Code structure analysis.

The program source code is parsed into the code block structure through AST, and then the code block is used as the basic unit of subsequent analysis and detection. Define the parsed source code structure as a five-tuple (fileID, segmentID, parentSegmentID, segmentInfo, hashIndex). The fileID and segmentID are the unique identifiers of the digital file and the code block, respectively. A code block can be uniquely identified by file ID and segmentID, while hiding the details of the code. parentSegmentID is the ID of the parent node of the code block , which is used to find segmentID, segmentInfo is other information in the code block, including startLine, endLine, textLength, sequence, startCol, etc. startLine is the starting line of the code block in the file, endLine is the ending line of the code block in the file, textLength is the text length information of the code block, startCol is the starting column of the code block, sequence is the order of different code blocks in the same file, etc. Figure 4-2 is an example of the code block structure of an abstract syntax tree. For example, in the code block N 4 (122, 3, 2...), 122 is the fileID of the code block, 3 is the segmentID, and 2 is the parentSegmentID of the code block.

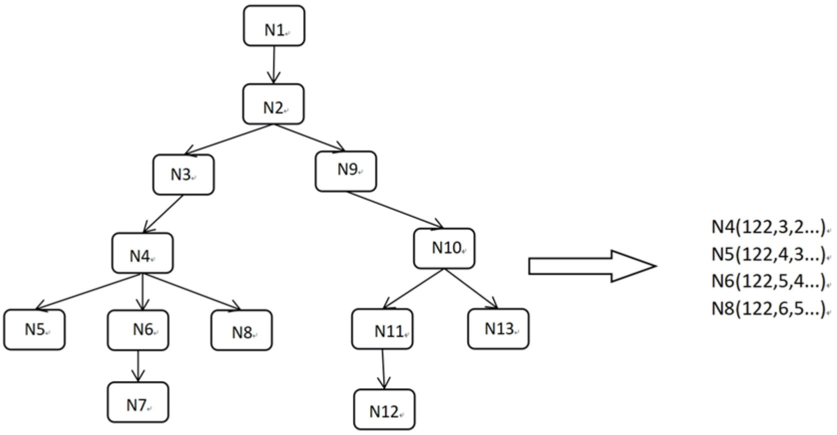


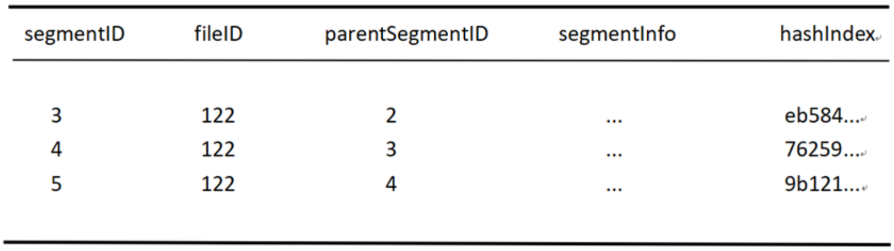
Fig. 4-2 Code block structure

(2) Code index generation

Based on code structure analysis, an index is generated for each code block, and the characteristics of a code block are uniquely represented by the index. Code index generation includes two steps of normalization and hash value calculation. Normalization is a common strategy for code duplication detection. By removing spaces, replacing variable names and numbers with consistent symbols, etc., the influence of different factors such as variable names and code format on the detection effect is eliminated. The MD 5 hash algorithm was used on the normalized code to generate one 128-bit hash value for each node [18]. Finally, the obtained hash index is written to the hashIndex of the code block information.

4. 2 Local mapping table construction

The local mapping table maintains the correspondence between the code blocks uploaded to the server and the local readable code. When the server returns the repeated information expressed by the index information, it can be parsed into the readable code form locally, as shown in Table 4-1. Among them, each line represents a code block, and the file ID and segmentID of the code block are used as indexes. parentSegmentID and segmentInfo reflect the corresponding information with the original code, and hashIndex represents the hash index feature information of the code block (see Table 4-1).



segmentID	fileID	parentSegmentID	segmentInfo	hashIndex
3	122	2	...	eb584....
4	122	3	...	76259....
5	122	4	...	9b121....

Tab. 4-1 Local mapping table

4. 3 Building a global feature information table

To perform feature matching with the code to be detected, the server also needs to maintain a global feature information table. The server converts the project source code into a global index table. The global feature information table not only needs to maintain the structural characteristics and index characteristics of all the code blocks, but also maintains the large-scale software project information and project license information.

4. 4 Upload code block information

Upload code block information to the server for code duplication detection. Before uploading, remove the original information in the code block information, such as the start column, end column, and code block occurrence order in segmentInfo. Keep the start line, end line, code block length information, version information, and then upload the code block information to the server. The server receives the code block information transmitted locally and uses the plagiarism detection tool to detect.

4.5 Plagiarism detection toolkit

The server determines the repetition of the code block and whether it is plagiarized based on the locally uploaded code block structure information. If plagiarism is found, the duplicate information of plagiarism will be sent to the local for analysis.

To this end, the uploaded code block information is compared with the global index table and is detected according to the duplicate detection rules. Several reference rules are given below.

Rule 1: Simple Index Duplication

Simple index repetition means that the index in the index table is matched with the index in the global index table. If the index value is the same and the number of lines of code is greater than 5, it is regarded as a repetition.

Rule 1: Detection process

```
For each index value S.hash in the index table S
  For each index value S1 in the global index table S1.hash
  If S.hash and S 1. hash values are equal do
  If S1.startLine—S1.endLine is greater than 5 do
    Store S.segmentID in index table S2
  End for
End for
```

Among them, S is the index table received by the server, which contains the start line sartLine, the end line endLine, the index value hash, the code block segmentID, and the index value detected by rule 1 will be stored in the index table S2.

Rule 2: Regional distribution of duplicates

Plagiarism detection using only the hash value of the code block is flawed. For example, when a person makes a slight modification to a code segment, the hash value changes, so that plagiarism cannot be detected. However, the code block information in this thesis contains structural information that can detect such plagiarism based on statistical features. Based on simple index repetition, a new rule is defined to perform statistical detection on the child node tree under the same parent node. If the number of child nodes with the same hash value exceeds a certain proportion, the parent node is a repeat node.

Rule 2: Detection process

```
For each index value S.hash in the index table S
   For each index value S1 in the global index table S1.hash
      If S.hash and S1.hash values are equal do
         If S1.startLine—S1.endLine is greater than 5 do
            for code block parentSegmentID corresponding to S.hash and S1.hash respectively
         If in addition to S.hash, there are still more than half of the child nodes with the same hash value and the same hash value, S.parentSegmentID is stored in the index table S2
         Else stores S.segmentID in index table S2
   End for
 End for
```

Rule 3: Global statistical characteristics of duplicates

In code plagiarism, whether the code has unique characteristics is often an important indicator. For example, a piece of code that, although repeated, is a widespread programming pattern rather than a unique feature of a project, and it cannot be marked as plagiarism. Based on the above, filtering is performed based on the global statistical characteristics of duplicates. If the index value is the same as the index value of multiple items in the global index table, for example, 30 is the critical value. If it exceeds 30, it is regarded as normal code reuse, otherwise, it is regarded as duplicate.

Rule 3: Detection process

```
For each index value S.hash in the index table S
   For each index value S1 in the global index table S1.hash
      If S.hash and S1.hash values are equal do
         If S1.startLine—S1.endLine is greater than 5 do
         For count the number of S.hash
```

```
        If count(S.hash)<30
        Store S.segmentID in index table S2
  End for
End for
```

Among them, S is the index table received by the server, the table contains the start line startLine, the end line endline, the index value hash, and the index value detected by rule 3 will be stored in the index table S2.

After finding the duplicate segment according to the duplicate detection rules, the license information of the file where the duplicate segment is in the source code of the large-scale project is detected. If the license information of the project where the duplicate code is located allows the user to modify and republish it, it is deemed that there is no plagiarism, otherwise is considered plagiarism. For example, the GPL license information does not allow its derivative code to be used for non-open-source commercial purposes, and if it directly reuses its project code, it will cause plagiarism.

4.6 Generating plagiarism reports

The server transmits the detected repeating sequence of code blocks back to the local. The received plagiarism detection report is parsed locally, and the local code is restored according to the code block ID for display.

5. System implementation

Plagiarism detection is mainly responsible for extracting the index features of software code, reporting to the server, and localizing the repeated information sent by the server. The server is mainly responsible for duplicate detection, as shown in Figure 5-1.

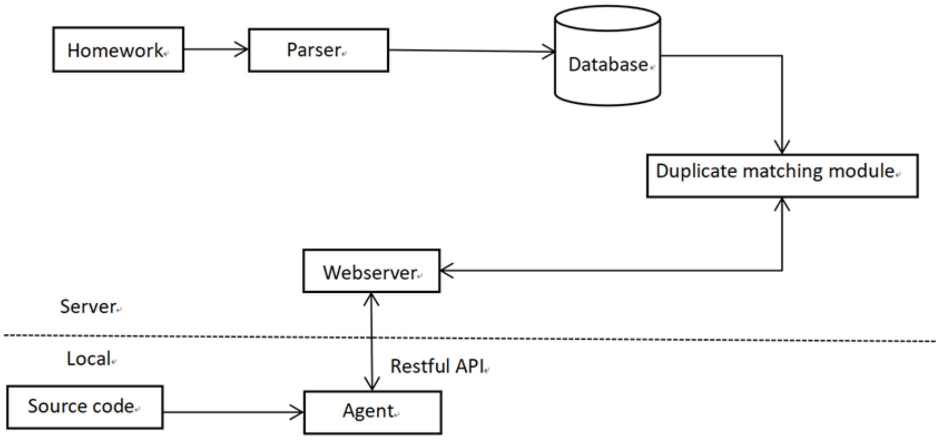


Fig. 5-1 Plagiarism detection system structure

Figure 5-1 The parser works both locally and on the server. The local is the project agent, and the server is the project parser. The main function is to generate an index table from the

source code through the syntax tree and index. When the local agent receives the duplicate from the server, it converts it into readable code form.

The project database mainly stores the scraped code on the server. Due to the large number of open-source projects, even if converted into an index form, it still takes up a lot of space, and the memory is not enough to store. At this time, the project is persisted and stored in the database.

The repeated matching module mainly performs repeated matching between the code block index information transmitted by the local project and the global index table on the server and uses the duplicate matching method to detect whether the repeat is repeated or not.

6. Experiments and results

To verify the plagiarism detection scheme proposed in this thesis, the homework of 50 students was selected as the experimental subjects. The experimental environment is the operating system Win10 64-bit, CPU Intel Core i 7 -10710U, and memory 16 G.

Experimental steps:

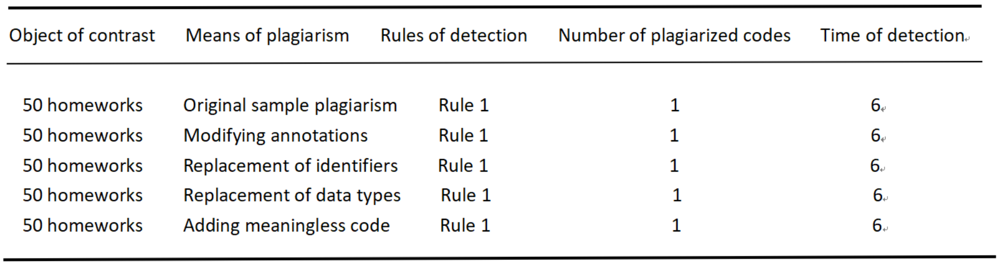
(1) Select the homework of some students as the source of the detection code.

(2) Build the open-source project code base, start the server, capture the students' homework information, and cache it locally, then parse the project, and store the parsed code blocks in the global index table.

(3) Select the duplicate detection rule, select rule 1, rule 2, and rule 3 respectively, and then start the detection to detect students' homework.

Experiment 1: Verify the detection ability of simple plagiarism rules (rule 1)

Selecting the homework of 50 students. According to the code plagiarism method proposed in the reference [18], code obfuscation operations are performed on the get() method , such as copying as it is, modifying annotation information, replacing identifier names, replacing data types, adding meaningless code, etc. Select Rule 1 for plagiarism detection. The experimental results are shown in Table 6-1. Using the plagiarism detection method of Rule 1, it only takes 6 seconds to detect all plagiarism codes. The experimental test report after the identifier name is replaced is shown in Table 6-1.



Object of contrast	Means of plagiarism	Rules of detection	Number of plagiarized codes	Time of detection
50 homeworks	Original sample plagiarism	Rule 1	1	6
50 homeworks	Modifying annotations	Rule 1	1	6
50 homeworks	Replacement of identifiers	Rule 1	1	6
50 homeworks	Replacement of data types	Rule 1	1	6
50 homeworks	Adding meaningless code	Rule 1	1	6

Tab. 6-1 Rule 1 Plagiarism detection report

The results of experiments 2, 3 and 4 need to be further verified.

7. Summary

This thesis proposes a source code plagiarism detection scheme based on index information, which realizes efficient plagiarism detection and has good customizability. The detection scheme not only considers the similarity between codes, but also considers the context of the program. The detection scheme can better protect personal intellectual property rights and enterprise source code security, and the detection time is relatively fast, achieving the expected results. However, there is still some room for improvement in this plagiarism detection tool. At present, only a few programming language source code plagiarism detections have been implemented, and more programming languages still need to be covered in the future. In addition, the scale of the test dataset in this experiment is small, and the scale will be expanded in the next step, and the effectiveness of the plagiarism detection tool needs to be further verified.

References

[1]https://www.infoworld.com/article/3049149/oracle-seeks-93-billion-for-googles-use-of-java-in-android.html

[2]https://www.gnu.org/licenses/license-list.html

[3]TIAN Zhen-zhou, LIU Ting, ZHENG Qing-hua, et al. Software plagiarism detection: a survey [J]. Journal of Cyber Security, 2016(3): 52-76.

[4]Prechelt, Lutz et al. “Finding Plagiarisms among a Set of Programs with JPlag.” *J. Univers. Comput. Sci.* 8 (2002): 1016-.

[5]M. Wise, “YAP3: Improved Detection of Similarities in Computer Program and Other Texts,” Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, Philadelphia, 15-18 February 1996, pp. 130-134.]

[6]S. Livieri, Y. Higo, M. Matushita and K. Inoue, "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder," *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 106-115, doi: 10.1109/ICSE.2007.97.

[7]Liu C , Chen C , Han J , et al. GPLAG: detection of software plagiarism by program dependence graph analysis[C]// Knowledge Discovery and Data Mining. ACM, 2006.

[8]Shi Q Q , Meng F J , Zhang L P , et al. Survey of research on code clone technique[J]. Application Research of Computers, 2013, 30(6):1617-1623.

[9]Hummel B , Juergens E , Heinemann L , et al. [IEEE 2010 IEEE 26th International Conference on Software Maintenance (ICSM) - Timi oara, Romania (2010.09.12-2010.09.18)] 2010 IEEE International Conference on Software Maintenance - Index-based code clone detection: incremental, distributed, scalable[J]. 2010:1-9.

[10]Halstead M H. Elements of Software Science (Operating and programming systems series)[M]. Elsevier Science Inc., 1977.

[11]Ottenstein K J. An algorithmic approach to the detection and prevention of plagiarism[J]. ACM Sigcse Bulletin, 1976, 8(4): 30-41.

[12]Grier S. A tool that detects plagiarism in Pascal programs[J]. ACM Sigcse Bulletin, 1981, 13(1): 15-20.

[13]Verco K L, Wise M J. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems[C]//ACM International Conference Proceeding Series. 1996, 1: 81-88.

[14]Baker B S. A program for identifying duplicated code[J]. Computing Science and Statistics, 1993: 49-49.

[15]Kim Y C, Cho Y Y, Moon J B. A Plagiarism Detection System Using A Syntax-Tree[C]//International Conference on Computational Intelligence. 2004, 1: 23-26.

[16]Jiang L, Misherghi G, Su Z, et al. Deckard: Scalable and accurate tree-based detection of code clones[C]//29th International Conference on Software Engineering (ICSE'07). IEEE, 2007: 96-105.

[17]Rivest R. The MD5 message-digest algorithm[R]. 1992.]

[18]Jones E L. Metrics based plagiarism monitoring[J]. Journal of Computing Sciences in Colleges, 2001, 16(4): 253-261.