ISSN: 1813-4890

Code Plagiarism Detection System Based on Programming Courses - Taking "Python Programming" as an Example

Jun Pan, Shihua Liu and Lili Shi

College of Artificial Intelligence, Wenzhou Polytechnic, Wenzhou, Zhejiang 325035, China

Abstract

With the development of information technology, resources on the internet have become very rich, and obtaining program resources from the internet has become increasingly convenient and fast, whether in school or vocational training. Plagiarism in programming assignments or exams has always been a serious problem. This project aims to design a code plagiarism detection system that can calculate the similarity between multiple source code files, quickly, efficiently, and accurately identify files suspected of plagiarism, provide similarity values, and intelligently detect some complex plagiarism techniques, such as replacing variable names and changing statement order, etc.

Keywords

Programming, similarity, Python, code.

1. Introduction

The issue of plagiarism in programming assignments and exams has been a serious problem in the process of computer-related studies. Even in some universities with good academic integrity, it is quite common for students to copy each other's programming assignments. According to surveys by foreign educational institutions, as many as 85.4% of students have experienced copying programming assignments. As programming courses can be challenging for students who are new to programming, they may resort to searching for related programs online or copying programs from their classmates and submit them directly, or even making some changes before submission. When teachers are grading assignments or exams, they need to manually check each program, and they must compare each pair of programs. With a large number of students, this process can be time-consuming and energy-intensive, and the judgment may not be accurate due to low precision.

Currently, in the teaching management of university teachers, an effective code plagiarism detection system is particularly important for programming exams and assignments. Whether two programs have a plagiarism relationship is measured by code similarity. The higher the similarity, the greater the possibility of plagiarism. This code plagiarism detection system should be able to measure the similarity of program codes, in order to assist teachers in grading assignments or exams and to facilitate better teaching process management, greatly improving efficiency. This project aims to design a smart code plagiarism detection system that can help teachers quickly match code source files that may have plagiarism, and also help with software copyright identification. It works better for complex code.

2. Current research status at home and abroad

As early as the 1970s, scholars abroad began researching the similarity detection of program code. In 1976, Halstead first proposed using the attribute count method to calculate the similarity between program codes [1]. In 1977, based on Halstead's research, Otteiistein designed the first program code plagiarism detection system based on attribute counting

method, which was suitable for FORTRAN programming language [2]. In 1996, Verco and Wise conducted extensive testing on Grier's Accuse system. After analyzing their test results in detail, Verco and Wise pointed out that in the statistical process, simply increasing the program attributes counted did not significantly improve the accuracy of the detection results. In addition, research scholars have also studied the use of neural networks for similarity detection.

Domestically, research on program code similarity detection did not start as early as in foreign countries. Compared with foreign countries, China's research on program code similarity detection started relatively late. However, Inner Mongolia Normal University's College of Computer Science has made breakthrough progress in program code similarity detection technology. Based on attribute counting technology and structural measurement technology, several program code similarity detection systems have been implemented. Yin Danping from Beijing University of Posts and Telecommunications proposed a code similarity detection and plagiarism detection system based on CNN, which combines the field of deep learning [3]. Liang Hanyuan from Harbin Institute of Technology designed a C language code similarity detection method based on AST and graph attention network [4].

3. Project Research Content and Implementation

Due to the limitations of commonly used algorithms such as sequence alignment, it is not effective in detecting some complex plagiarism techniques, such as variable name substitution and changing the order of statements, which can result in many erroneous judgments. Additionally, processing Python source code is also crucial, as code comments themselves have no actual impact, and computing similarity on them can cause redundancy and affect the accuracy of similarity judgment. Therefore, this project first reads the files to be compared in a loop and conducts redundant preprocessing of the code. Then, it uses the generated abstract syntax tree to traverse and generate a syntax word list. Finally, it uses the Smith-Waterman algorithm to compare the similarity of the syntax word list, outputs the similarity value, sorts it, and outputs the file pair with the highest similarity for reference by the user. The overall implementation approach is as follows:

3.1. Reading and preprocessing of source code files

First, put the source code files to be detected in a folder (usually there are several dozen source files for a class's homework or exam collection). Then use Python to read all file paths in the current directory, and traverse all paths to read 2 files that are not repeated.

For the files read, preprocess their content, such as deleting comment statements, blank lines, and some spaces. Remove redundant components of the code, such as some header files, macro definition statements, etc., and unify the format for the next stage of processing and detection.

3.2. Generate Abstract Syntax Trees

AST, or Abstract Syntax Trees, is a tree-like representation of the abstract syntactic structure of source code, where each node on the tree represents a structure in the source code. Typically, in the process of translating and compiling source code, a parser is able to construct a parse tree, which is then transformed into an AST [5]. For example, the following function definition can ultimately be parsed into a syntax tree:

ISSN: 1813-4890

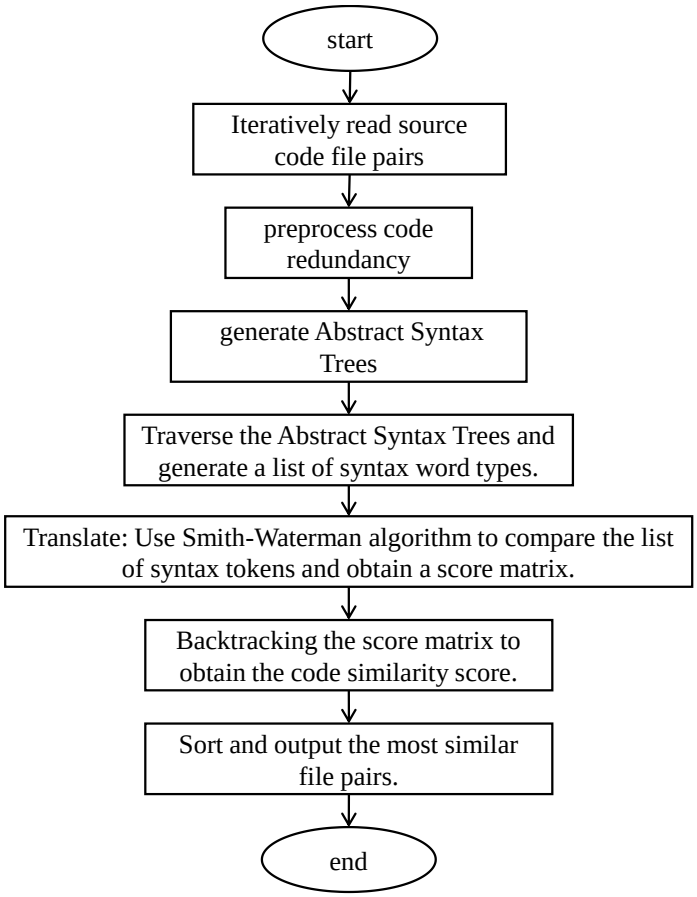

Fig. 1 Implementation approach

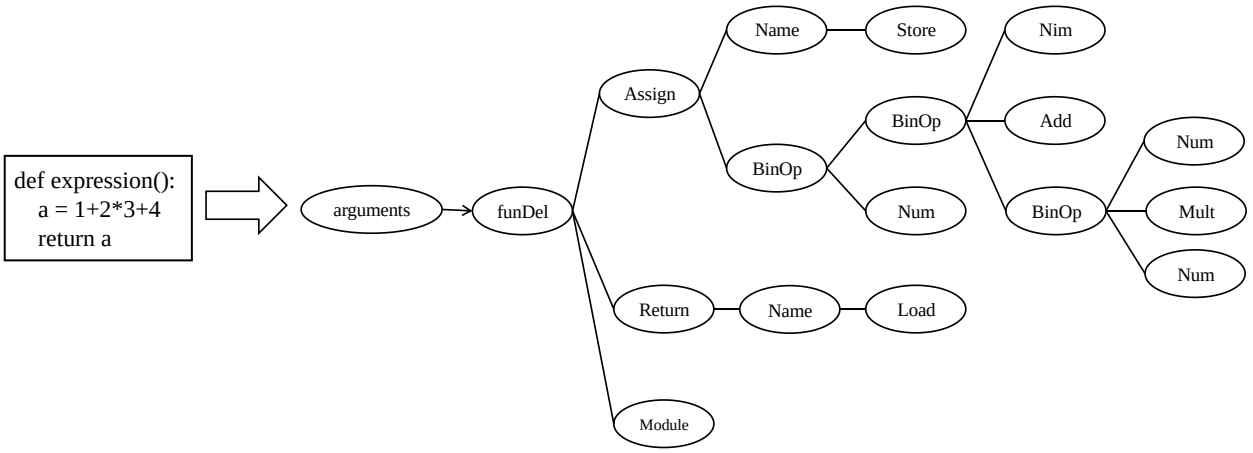

Fig. 2 A AST sample

AST-based plagiarism detection mostly involves traversing and comparing each subtree of the two ASTs to obtain their similarity score. Since the order of subtree comparison does not affect the comparison results, this method can effectively handle plagiarism techniques such as identifier renaming and code reordering [6].

3.3. Traverse the Abstract Syntax Trees and generate a list of syntax word types.

The next step is to traverse the structure of the syntax tree and output the syntax structure of the entire code in order for subsequent comparisons. The NodeVisitor class in the ast library based on Python is used to traverse the syntax tree generated earlier recursively visiting nodes. In the process of judgment, it is necessary to determine whether an object is a known type, i.e., whether it is a list or an AST. If it is a list, it is further traversed, if it is an AST, the visit function

ISSN: 1813-4890

is used to traverse it. If it is neither a list nor an AST, no operation is performed. Taking the syntax tree of version 3.2 as an example, the final output is a list of syntax word types as follows:
arguments,Store,Name,Num,Add,Num,Mult,Num,BinOp,BinOp,Add,Num,BinOp,Assign,Load,Name,Return,FunctionDef,Module

3.4. Comparing the syntax word type lists to obtain a score matrix.

After obtaining the syntax token lists of two source code files, the similarity of syntax tokens will be compared using the Smith-Waterman algorithm. The Smith-Waterman algorithm is a local sequence alignment algorithm used to find similar regions between two nucleotide or protein sequences. Its purpose is not to perform a full sequence comparison, but to find highly similar segments between two sequences.

Given the two sequences to be compared: $A=a_1a_2...a_n$, $B=b_1b_2...b_n$.

A score matrix H is created, which is of size n+1 rows and m+1 columns, and is initialized to 0.

Set s(a,b) as the score of similarity between the elements that form the sequence, and W_k as the gap penalty for a length k. Penalty rules are set as follows:

$$s(a_i, b_j) = \begin{cases} 1, & a_i = b_j \\ -1, & a_i \neq b_j \end{cases}$$

Initialize the matrix based on the following formula:

$H[i-1,j-1]+s(a_i,b_j)$ represents the similarity score of aligning ai and b_j.

$H[i-k,j]-W_k$ represents the score of deleting a gap of length k ending with a_i.

$H[i,j-l]-W_l$ represents the score of deleting a gap of length l ending with b_j.

$$H_{ij} = \max \begin{cases} H_{i-1,j-1} + s(a_i, b_j) \\ H_{i-1,j} - W_1 \\ H_{i,j-1} - W_1 \\ 0 \end{cases}$$

0 indicates that ai and bj have no similarity at this position.

The algorithm can be simplified by using the constant gap penalty model of the Smith Waterman algorithm, in which the gap penalty value is fixed.

3.5. Backtracking the score matrix to obtain the similarity score of the two codes.

Starting from the highest scoring element in the matrix H, backtrack to the previous position based on the source of the score, repeatedly until an element with a score of 0 is encountered. First, obtain the position of the highest scoring element in the matrix. If ai=bj, backtrack to the upper left cell; if ai≠bj, backtrack to the cell with the maximum value among the upper left, upper, and left cells. If there are cells with the same maximum value, the priority is in the order of upper left, upper, and left. Based on the backtracking path, write out the matching string.

If backtracking reaches the upper-left cell, add a_i to the match string A1 and add bj to the match string B1;

If backtracking reaches the upper cell, add a_i to the match string A1 and add "" to the match string B1;

If backtracking reaches the left cell, add_to the match string A1 and add bj to the match string B1.

Actually, the operation above is to start from the maximum element and find the position of the element that produces it. If this backtracking path is saved when generating these elements, it would be more efficient to rewrite the calculation of the score matrix while saving the path. It only needs to find the coordinates of the maximum value, then find the coordinates

corresponding to that coordinate, backtrack to find the visited coordinates, record the coordinates along the way until visiting a value of 0.

3.6. Output the alignment result of similar code pairs

Finally, all the above functions are integrated together. In order to provide more evidence for reference, for each compared code file pair, the matching result of the syntax tree needs to be traced back to the corresponding block of the source code, so that users can further compare and refer to the plagiarized segments.

4. Conclusion

For a more reasonable test, we collected forty final exam codes from students for verification and found that even if the students substituted variables, plagiarism could still be detected. However, it was discovered that this study was more suitable for code containing complex grammar structures, as for simple code, most people have similar basic structures, resulting in a high chance of false positives when judging. Therefore, the results of this study should only be used as a reference when judging plagiarism.

Acknowledgements

This article is supported by the "14th Five-Year" Teaching Construction and Teaching Reform Research Project of Wenzhou Vocational and Technical College, with project number WZYYB 202223.
The article was also supported by the Project of the First Batch of Ideological and Political Demonstration Courses in Zhejiang Province(ZheJiaoHan [2021]No.47),the Course name: Relational Database Application (No.37).

References

[1] Cheng Jinhong. Research on Measurement of Program Code Similarity [D]. Hohhot: Inner Mongolia Normal University, 2007:12.

[2] Hu Zhengjun. Research and Application of Program Code Similarity Detection Method[D]. Hunan: Central South University, 2012:31-14.

[3] Yin Danping. Research on Code Similarity Detection Based on CNN and Code Checking System [D]. Beijing University of Posts and Telecommunications, 2018.

[4] Liang Hanyuan. Research on C Language Code Similarity Detection Method Based on AST and Graph Attention Network [D]. Harbin University of Science and Technology, 2022. DOI: 10.27063/d.cnki.ghlgu.2022.000635.

[5] Ouyang Peng. Research on Software Defect Prediction Method Based on Program Abstract Syntax Tree[D]. South China University of Technology, 2021. DOI: 10.27151/d.cnki.ghnlu.2021.002921.

[6] Wang Guolong, Jin Dahai, Gong Yunzhan. Research on Error Handling Technology of Abstract Syntax Tree Generation Based on JavaCC[J]. Computer Measurement and Control, 2022,30(02):151-159. DOI: 10.16526/j.cnki.11-4762/tp.2022.02.022.