A Novel Code Stylometry-based Code Clone Detection Strategy

Wenyuan Dong
Beijing Key Lab
of Intelligent Telecomm.
Software and Multimedia
Beijing University of
Posts and Telecomm.
Beijing 100876, China
dwwwwy@bupt.edu.cn

Zhiyong Feng
Dongfang Middle School,
Yanzhou District, Jining City
Shandong 272000, China
sdyzfzy@163.com

Hua Wei
Beijing Key Lab
of Intelligent Telecomm.
Software and Multimedia
Beijing University of
Posts and Telecomm.
Beijing 100876, China
weihua2015@bupt.edu.cn

Hong Luo
Beijing Key Lab
of Intelligent Telecomm.
Software and Multimedia
Beijing University of
Posts and Telecomm.
Beijing 100876, China
luoh@bupt.edu.cn

***Abstract*—Program similarity-based detection can be used to discover potential code cloning issues. However, just based on the similarity of the program, it is difficult to accurately identify code clones with modified syntax and program similarity does not directly reflect the possibility of code plagiarism. In this paper, we propose a program similarity detection strategy with code stylometry matcher to solve this problem. Firstly, the code stylometry model based on the random forest classifier is used to identify a list of possible authors. If the code submitter is not in this list, we analyze the similarity between the submitted code and the program submitted by the author in the list to determine whether there is code cloning. In this way, we can effectively reduce the complexity of program similarity comparison. We propose a PBCS method based on parallel Bi-directional Long Short-Term(Bi-Lstm) Memory network for code similarity detection. This model effectively improves the accuracy of detection by supervising deep features extracted from AST and token. Experimental results show that the recall of the proposed method is 84%, which is improved compared to other methods.**

***Keywords*—code similarity; Bi-Lstm; code stylometry**

I. Introduction

With the development of information technology, plagiarism is becoming easier to implement and harder to prevent. In the assessment of programming coursework and online tests, there are also some students who copy code from others. Since the original author is not properly identified in the huge assignment set, students can easily reuse other people's source code by copy-paste-modify and avoid being detected. It makes computer-related teaching objectives more difficult to reach the expectation. In order to reduce the occurrence of code plagiarism in computer professional education and ensure the teaching quality of teachers, it is necessary to design a special tool for code clone detection.

There are many studies on code plagiarism, and they mainly recognize code plagiarism by detecting code similarity. The program similarity detection technology mainly includes two types: one is a method based on Attribute Counting (AC), and the other is a method based on Structural Metrics (SM) [1]. The method based on SM makes up for the problem that structural information is easily ignored in the attribute-based method. The method based on SM first matches the structural representations converted from the source code through corresponding matching algorithms. Then, we calculate the distance between the structures as the similarity between the programs. The performance of the algorithm varies with structural granularity. At present, there are some methods based on machine learning or deep learning is introduced to learn the deep features of different granularity of the code, which can effectively detect code clones.

However, it is difficult to accurately identify code clones with modified syntax and program similarity does not directly reflect the possibility of code plagiarism. In this paper, we propose a method associated with code styles to effectively detect program similarity. First, we determine the author of the code through code style matching. If a program's coding style is inconsistent with its the author's coding style, code plagiarism is likely to occur, and then program similarity calculations are used to further determine whether code plagiarism exists. Our code clone detection system with author's code style analyzing can improve the efficiency of code plagiarism detection.

In code style matching models, we use machine learning method to learn the code characteristics of multiple authors, including lexical, layout, and syntactic features of the code, and then use Random Forest Classification (RFC) to train code style matcher models. In code similarity detection, we use ASTs and Tokens as the source code representation, construct word vectors based on word embedding technology, and then propose a code similarity detection model PBCS based on parallel Bi-directional Long Short-Term Memory model.

The code clone detection can be used in the detection of plagiarism in students' program assignments, and also have a practical significance for software copyright identification. Experiments show that our method can more effectively and accurately detect similarities. The main contributions of this paper are summarized as follows:

978-1-7281-3129-0/20/$31.00 ©2020 IEEE

1)We design a author identification model based on code style, and realize code stylometry matcher based on random forest classifier (RFC) through the lexical, layout, and grammatical features of the code.

2)We propose a code clone detection model PBCS based on a parallel Bi-directional Long Short-Term Memory model. By constructing word vectors based on word embedding, we design and implement a program similarity detection system.

3)The proposed code style-based code similarity detection method can effectively detect code plagiarism. Through experiment on the public dataset OJClone, the accuracy of our proposed method is 94%, which is improved compared to other methods.

The remainder of this paper is organized as follows. Section II presents background and related work. Section III overviews the whole system. Section IV presents the Code Stylometry Matcher Methodology. Section V proposes the Code Similarity Detection Methodology. Experiments and results are given in Section VI. The paper concludes with section VII.

II. Related Work

In terms of program similarity detection, many systems and supporting tools have appeared. A combination of attribute-based counting and structural-based measurement was used to detect program code similarity, such as the Moss system of Stanford University [2].

Shippey Thomas [3] proposed a method for calculating the similarity of abstract syntax trees based on N-gram algorithm.

Code clones can be categorized into four different types based on different levels of similarity [1]: Type-1: identical code fragments in addition to variations in comments and layout; Type-2: apart from Type-1 clones, identical code fragments except for different identifier names and literal values; Type-3: apart from Type-1 and -2 clones, syntactically similar code that differ at the statement level. The code fragments have statements added, modified and/or removed with respect to each other; Type-4: syntactically dissimilar code fragments that implement the same functionality.

Jiang $et\ al.$ [4] introduced the AST structure to calculate the structural similarity of code slices in Deckard.the syntactical information may be helpful in detecting Type-3 clones, it may still not be effective to detect the Type-4 clones.

Mateusz Pawlik and Nikolaus Augsten [5] proposed a more robust tree edit distance algorithm in calculating code fragments similarity. Socher $et\ al.$ [6] used deep learning to perform feature learning through recursive autoencoders, which represented each source code segment as an identifier and text type for code clone detection. But it needs to manually extract a large amount of features.

Sajnani $et\ al.$ [7] convert the source program into a token sequence and compare subsequences to detect cloning in SourcererCC. SourcererCC is typical lexicon-based approach which ignores the syntactical information. Thus, this method may be able to successfully detect the lexical clones and would be ineffective for detecting syntax clones in most cases. Although the syntactical information may be helpful in detecting Type-3 clones, it may still not be effective to detect the Type-4 clones.

III. System Overview

In this section, firstly, we introduce our system workflow. Then we introduce Code Stylometry Matcher strategy and Code Similarity Detection model.

A. System Workflow

Fig 1 shows the system workflow. First, we get large amounts of code submissions which contain program problem, author and code fragment from the online judge system in CodingGo website. Then, for any code fragment from the submissions, we can get its potential author list by the Code Stylometry Matching strategy. If the code submitter is in this list, we think it is created without plagiarism, and end this process. Otherwise, we analyze the similarity between the submitted code and the program submitted by the author in the list. Finally, we can know the all plagiarism probability in the submissions under the same problem.

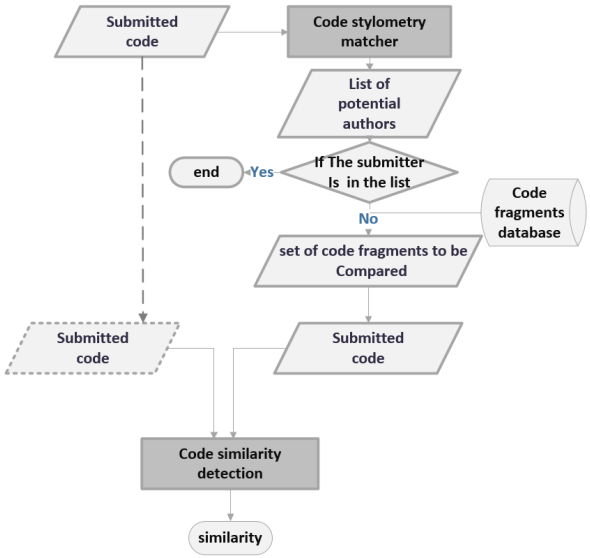

Fig. 1. System Workflow

B. Code Stylometry Matcher Strategy

Fig 2 shows that the Code Stylometry Matching strategy includes feature extraction, code style classification, and style matching process. Firstly, the source code is transformed into a vector representation through feature extraction. Then, the code style of the input code is identified by trained random forest classifier and we can obtain a list of potential authors as output. Finally, we compare whether the submitter is in the list. If not, select the submissions of the authors in the list on this problem from the database, and perform code similarity detection.

C. Code Similarity Detection Model

Fig 3 shows that the Code Similarity Detection workflow contains feature extraction, vector normalization, and a Bi-LSTM network model. The input source code pairs are converted into AST features and token features through feature extraction. Vector normalization is based on the word embedding technology to train AST features and token features, and convert the AST features and token features into corresponding vector representations. We use the PBCS model to calculate the similarity of the code pair under a different level of granularity.

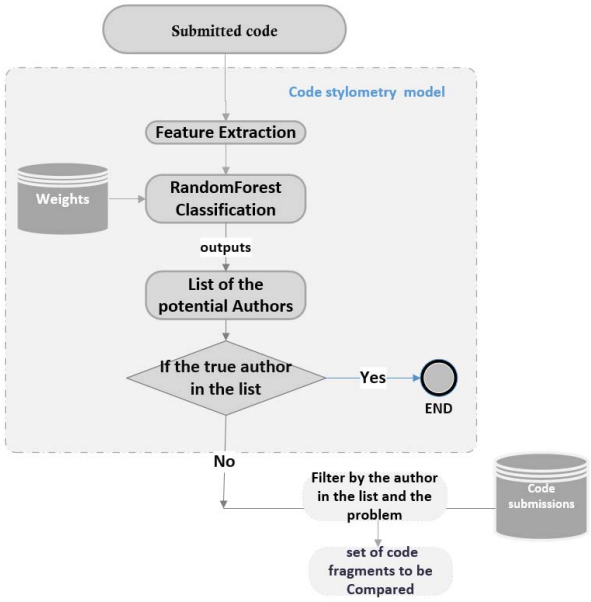

Fig. 2. Code Stylometry Matcher

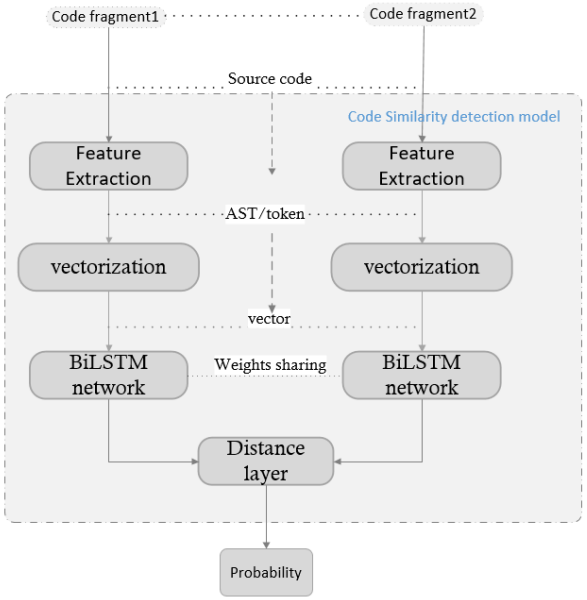

Fig. 3. Code Similarity Detection

IV. Code Stylometry Matcher Methodology

In this section, we introduce the code stylometry matcher methodology in detail. It contains three processes: feature extraction, random forest classification, and matching potential authors.

A. Feature Extraction

We mainly extract data from three aspects of source code: lexical features, layout features, and syntactic features. We extract the lexical features with personal characteristics from the source program, such as identifiers, keywords, functions, and function parameters. Layout characteristics can clearly reflect people's programming habits. Lexical and layout features can be extracted from the source program without going through the parser stage. Here, we can call the lexical analyzer Lexer to convert the source program into a token stream, and then analyze the token stream to extract the corresponding lexical features and layout features. For syntax features, we analyze the source program and transform it into an abstract syntax tree, and then analyze the abstract syntax tree to obtain its syntax features.

Lexical Features: Table I summarizes of the lexical features used in our Code Stylometry Matcher strategy.

TABLE I
Lexical Features

Features	Description
Ln(keyword/length)	Log of the number of occurrences of keyword divided by file length in characters
Ln(Operator/length)	Log of the number of operators divided by file length in characters
Ln(Token/length)	Log of the number of tokens divided by file length in characters
Ln(Literals/length)	Log of the number of string, character divided by file length in characters
Ln(numFunct/length)	Log of the number of function divided by file length in characters
Ln(lengthFunc/length)	Log of the number of line in function divided by file length in characters
numParameter	the average number of Parameters in function

Layout Features: Table II summarizes of the layout features used in our Code Stylometry Matcher strategy.

TABLE II
Layout Features

Features	Description
Indentation	A boolean representing the use of space or tab for indentation
Connection	A boolean representing the use of space in string connection
Space	Log of the number of space divided by file length in characters
Empty line	Log of the number of empty line divided by file length in characters
Comment length	Log of the number of comments divided by file length in characters
Line length	the average length of line

Syntactic Features: Table III summarizes of the syntactic features used in our Code Stylometry Matcher strategy.

TABLE III
SYNTACTIC FEATURES

Features	Description
Height	AST tree height
Average height	Average AST tree height
Node	The number of nodes
Leave	The number of leaves
Average Node height	Average AST Node height for 22 important node type
Number of each node	The number of Node for 22 important node type

In order to facilitate extraction features, we use the ANTLR syntax generator. This tool can generate the corresponding lexical analyzer Lexer, parser Parser and specific ParseTreeWalker according to the Grammer file of the grammar rules we have written in advance. Lexer, a lexical analyzer, converts the source program into a token stream. Then, the token stream is input to the parser Parser, which can be transformed into an abstract syntax tree after parsing. Finally, the ParserTreeWalker is used to traverse the abstract syntax tree to obtain detailed syntax information.

B. Random Forest Classification

The random forest algorithm [] is a Bagging algorithm based on a decision tree. Random forests are ensemble learners built from collections of decision trees, each of which is grown by randomly sampling N training samples with replacement, and then generates k classification trees according to the self-help sample set to form random forest. The classification results of the data depend on the scores formed by the classification tree[8].

Based on the output of random forest classification, we train the classification model with a large number of features and analyze the style of source code and obtain a list of potential authors.

C. Matching the potential author

we obtain the list of potential authors, and we can know whether the submitter is on the list or not. If the submitter is not on the list, we think there may be more likely to occur code clone. Then we select submissions of authors in the list from this program problem in the database. We output the code fragments set which is to compare the code similarity with the input code fragment.

V. CODE SIMILARITY DETECTION METHODOLOGY

In this section, we introduce the Code Similarity Detection methodology in detail. It contains four processes: feature extraction, vector normalization, Bi-LSTM network model and parallel Bi-LSTM network model.

A. Feature Extraction

We extract Tokens and ASTs from source code. The Tokens in source code generally include constants, keywords, identifiers, strings, special symbols, and operators. The majority of code tokens are identifiers defined by user. The code fragment Ci can be represented as tokens sequence:

$$Ci \rightarrow R_{token} = \sum_{i}^{n} r_{token} i$$

The information AST in source code include the AST node type, and the code fragment Ci can be represented as AST node sequence:

$$Ci \rightarrow R_{ast} = \sum_{i}^{m} r_{ast} i$$

B. Vector Normalization

Word embedding is often used in natural language processing. It can map each word or phrase to a high-dimensional vector on the real number field. We train the source code corpus to get the embedding vector of tokens. Then the code fragment Ci can be represented as:

$$Ci \rightarrow V_{i_{token}} = \sum_{i}^{n} v_{i_{token}}$$

We train the source code corpus to get the embedding vector of tokens. Then the code fragment Ci can be represented as:

$$Ci \rightarrow V_{i_{ast}} = \sum_{i}^{n} v_{i_{ast}}$$

C. Network Model Based Bi-LSTM

To detect code clones between code pairs, we use two neural networks to process a pair of code fragments in parallel as shown in the Fig 4. Two neural networks share the same parameter set. We compute the cosine similarity of the two vectors in the Distance layer. Sij is the cosine similarity of the vector Vi and Vj converted from Ci and Cj :

$$S_{i,j} = CosineSimilarity(V_i, V_j) = \frac{V_i \cdot V_j}{\| V_i \| \| V_j \|}$$

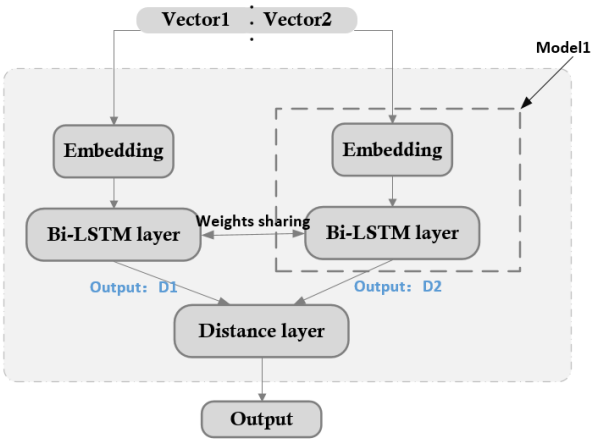

Fig. 4. Bi-LSTM based network model

D. Parallel Bi-LSTM-based Network Model

As shown in Fig 5, we build a parallel Bi-LSTM network model contains two Bi-LSTM based network model in Fig 4. to train token features and AST features of the code, respectively. Finally, the prediction result P1 (AST) is obtained based on the training of the AST feature, and the prediction result P2 (Token) is obtained based on the training of the token feature. The output of combining the two features is:

$$P = \alpha P_i + (1-\alpha)P_j (0 \le \alpha \le 1)$$

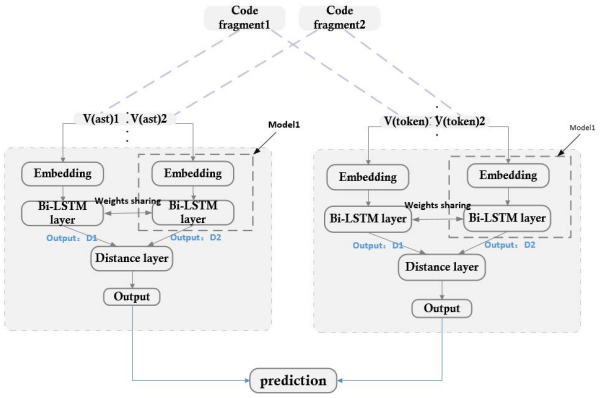

Fig. 5. parallel Bi-LSTM based network model

VI. Experiment

A. Dataset

The dataset of our experiment comes from the public data set OJClone and the Online judge system on CodingOJ website. In the experiment, we selected 20 programming problem, each of which selects 100 source code files. In OJClone, two source codes under the same programming problem are marked clone pairs and belong to at least a Type-3 clone [1]. The CodingOJ dataset is the data of the online programming evaluation system. We selected the 30 users who submit the largest number of answers and performed similarity calculation experiments based on code style classification.We mainly retain the main text information of the review. The statistical information of the dataset is shown in Table IV.

TABLE IV
The statistical information of dataset

aspect	dataset	train	test
code similarity detect	OJClone	250000	160000
code stylometry matcher	CodingOJ	600	1500

B. Comparison With Baseline Methods

In the comparison experiment,we used two different algorithms to compare with our code clone detection method. **SourcererCC**: a popular lexical-based clone detector. **Deckard**: a popular syntaticial-based clone detection tool. For comparison, three benchmarks are considered in our experiments, which are briefly summarized as follows: precision(P), recall(R), F1 value.(TP is True Positive, TN is True Negtive, FP is False Positive, FN is False Negtive) The precision reflect the correct number of samples determined to be positive:

$$P = \frac{TP}{TP+FP}$$

The Recall reflect the correct number of positive samples:

$$R = \frac{TP}{TP+FN}$$

If both P and R are required, we can compare the F1 score:

$$F1 = \frac{2P*R}{P+R}$$

The results of the comparative experiment are shown in Table V.

TABLE V
Experimental results of OJClone Dataset

Methods	Evaluation Metrics		
	precision (p)	Recall(R)	F1-score(F1)
Deckard	0.98	0.25	0.15
SourcererCC	0.44	0.74	0.16
PBCS	0.94	0.82	0.84

From the experimental results of the code clone detection on OJClone dataset in Table V, we can see that our model has improved recall from 25%, and 74% to 82%. Compared to Deckard, SourcererCC, our model is better at detecting code clone detection. SourcererCC's performance in JAVA is better in code clone detection.

TABLE VI
Experimental results of CodingOJ Dataset

method	Number percentage of code clones	time cost(ms)
PBCS	0.345	565ms
PBCS + CSM	0.267	237ms

To verify the efficiency of our proposed method, We take 30 programmers and select 20 submissions for each one of them for training code stylometry matcher, and we select 50 submissions per programmer as test data the experiment PBCS+code stylometry matcher(CSM). The results in Table VI show the percent of code pairs detected in CodingOJ dataset and the timecost. The results show that our PBCS model with code stylometry matcher is more efficient.

VII. Conclusion

In this paper, based on code stylometry matcher, we propose a novel code clone detection model. The main idea is to detect code clones by a parallel Bi-Lstm network joint code stylometry matcher. Among them, when performing code similarity detection, we combine important information of AST and Token features extracted from source code. The experimental results show that our model achieves better performance on the dataset.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.61672109, No.61772085, and No.61877005.

REFERENCES

[1] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY*, vol. 115, 2007.

[2] S. Schleimer, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data 2003*. ACM Press, 2003, pp. 76–85.

[3] T. Shippey, D. Bowes, and T. Hall, "Automatically identifying code features for software defect prediction: Using ast n-grams," *Information and Software Technology*, vol. 106, pp. 142 – 160, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584918302052

[4] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *IN ICSE*, 2007, pp. 96–105.

[5] M. Pawlik and N. Augsten, "Rted: A robust algorithm for the tree edit distance." *PVLDB*, vol. 5, no. 4, pp. 334–345, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/pvldb/pvldb5.html#PawlikA11

[6] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[7] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1157–1168. [Online]. Available: https://doi.org/10.1145/2884781.2884877

[8] A. C. Islam, R. E. Harang, A. Liu, A. Narayanan, C. R. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing programmers via code stylometry," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, 2015, pp. 255–270. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/caliskan-islam