The 7th International Conference on
Computer Science & Education (ICCSE 2012)
July 14-17, 2012. Melbourne, Australia

Source Code Plagiarism Detection and Performance Analysis Using Fingerprint Based Distance Measure Method

Sandhya Narayanan
Department of CSE
FISAT, Cochin, India
nairsands@gmail.com

Simi S
Department of CSE
FISAT, Cochin, India
Simi.surendran@gmail.com

Abstract— **In today's world, technological trends facilitate faster and easier communication between people any where in the world. Data search and exchange is one mouse click away from every people and this causes an exponential increase in the magnitude of plagiarism. Researches show that this is a serious problem in many academic institutions and research institutes. In this paper, we have developed an algorithm based on fingerprinting approach to identify the reuse of source codes in direct and indirect way. The results show that the proposed method is efficient in computation and resources and, provides effective measures to detect plagiarized programs. Effectiveness of the features used in the proposed methods is tested using a large database, including programs of different programming languages.**

Keywords- plagiarism, finger printing

I. Introduction

Plagiarism is the reproductions of others work with out acknowledging the source. It is a serious problem in many academic institutions and research institutes. Due to the widespread use of internet, students plagiarise most of the programming assignments from the web resources. In application development scenarios also, the programmers are utilizing the unreliable source code segments which may result in unnoticed bugs in software packages. A survey performed by Sheard, Dick, Markham, Macdonald & Walsh on a sample of students at Monash and Swinburne universities shows that 85.4% of 137 Monash University students and 69.3% of 150 Swinburne University students admitted to having engaged in academic dishonesty [1]. For all the above scenarios, it is difficult to manually detect the similarity between source codes. So we require an automated mechanism which will find out plagiarism effectively with minimum computation and resources.

Detecting plagiarism in software presents some problems due to the nature of programming. Many attributes of a set of non-plagiarised programs could exhibit similar properties. When we are detecting similarity between codes, we have to take account of different scenarios like redundancy in code , code produced from Code Generation Tools such as Net beans IDE, implementation of common algorithms etc. In this paper, we developed a plagiarism detection tool using finger printing based on distance measure approach. The system can detect the similarity between source codes written in same language or in different languages. It supports C, C++, Java, and Matlab and C# source codes. For inter language detection, we are using a low threshold level similarity score to find out the copying effect. A combination of similarity score and complexity measure gives a better performance to the system.

The paper is organized as follows: Section II and III describe the work related to the system and the design of the source code matching mechanism. Section IV includes the experimental results and performance evaluation followed by conclusion.

II. Related Work

several techniques to detect for both natural languages and programming languages. One of the simplest method to detect plagiarism is string pattern matching. This technique uses only the actual words or letters used in a document and not semantics [2]. String pattern matching techniques does not require knowledge of the language being analysed and do not take into account structural changes. An advantage of this being that such a system could be used to detect similarity between types of text document [3].

Another plagiarism detection tool is developed by Mozgovoy et al., called Fast Plagiarism Detection System (FDPS). FDPS aims to improve the speed of plagiarism detection by using an indexed data structure to store files [4]. Initially files are converted into tokens and tokenised files are stored in an indexed data structure. This allows for fast searching of files, using an algorithm similar to that in YAP3. In [5], the authors described a method for finding similar codes using conceptual similarity based measure. Each constructs of the source code is converted into its corresponding graphical structure. The system gives a better similarity measure but high complexity and the large amount parameters involved in the computation process. Christian Arwinand et.al proposed an approach [6], to detect plagiarism in multiple languages using intermediate program code produced by a compiler suite. They are able to handle cases where source code is copied from one language to another. Their test was valid only when using the GCC compiler suite for plagiarism detection involving programs written in the C and Java languages.

Authors of [7] considered program data for tracing similarities. Expression list of data dependency statement is used to detect copied programs. Also they are using data dependency matrix method to detect copied code. The system will give a better performance when there is high degree of dependency in source code, but, there is degradation in performance, if the dependency between lines of code is less.

III. System Design

The proposed system uses multiple phases to detect plagiarism effectively. The system will take source codes from the database and outputs the similarity score by passing through different phases. Following figure 1 shows the design of the system.

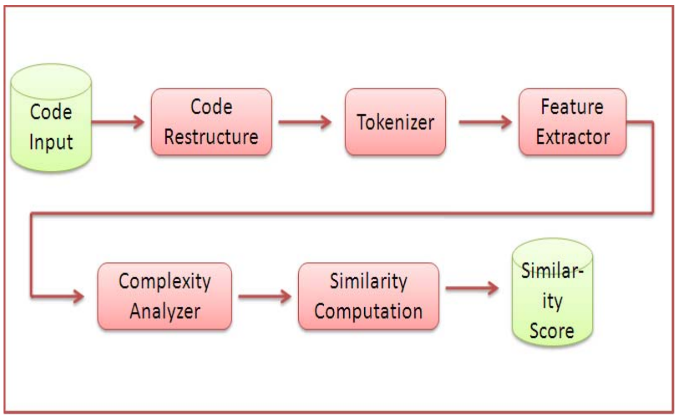


Figure 1. System Design

A. *Code Restructuring Phase*

The plagiarism checker will take C, C++, and Java and C # codes as input files. A predefined standard format of all these programming languages codes is stored for reference. In the code restructuring phase, both the input files are converted into the standard format depends on their file extension. The comment lines of the codes are eliminated first and then remove white spaces.

B. *Tokenization Phase*

In this phase, we are using a two level statement based feature extraction of the lines of code. The algorithm will check the language syntax properties to identify the statement usage. The keyword list of each programming language is known to the system. Depending on the keyword, the algorithm will find the identifiers used in the code. Then it will generate a signature of the identifiers which includes the count and other features associated with the identifier. This phase analyses the restructured code to identify the number of lines, number of characters used, and the number of inputs and outputs required for the program. It also separates code blocks, parameter passing combinations, and the features of different operations carried out in the program. The output of this phase is the detailed fingerprint of the source code.

C. *Complexity Analyzer*

To estimate the volume of the source code , we are using Halstead's Measure [8, 9] . The volume of the code is computed using the following equation.

$$V = (N_1 + N_2)\log_2(n_1 + n_2) \tag{1}$$

where n_1 is the number of unique operators, n_2 is the number of unique operands, N_1 is the number of operator occurrences and N_2 is the number of operand occurrences. The unique operators under consideration are summation, subtraction, division, multiplication, less than or greater than symbols etc .Number of occurrences of all these operators is taken and unique operands are variables. This computed volume is used as an extra feature to improve the similarity measure.

D. *Similarity Computation*

In this phase, we are computing the plagiarism percentage. Consider two programs A and B. Let Feature count of program A be $f_c(A)$, feature count of program B be $f_c(B)$, and number of features be n. If $f_c(A)$ is less than $f_c(B)$, then, compute the ratio

$$\delta_i = \frac{f_{C_i}(A)}{f_{C_i}(B)} \tag{2}$$

If $f_c(B)$ is less than $f_c(A)$ then,

$$\delta_i = \frac{f_{C_i}(B)}{f_{C_i}(A)} \tag{3}$$

δ is the ratio of the signatures of program A and program B. Using this feature ratio, a distance measure is computed. This is represented as Δ . Thus the distance measure based similarity score is

$$\Delta = \frac{\Sigma_{i=1}^{n} w_i \delta_i}{n} \tag{4}$$

Where W_i is the weight assigned to the different features of the program. These features are selected with respect to input programs. The similarity score will be between 0 and 1. To improve the similarity measure, we are using the volume feature. Let $V(A)$ be the volume of program A and V(B) be the volume of program B. If $V(A)$ is less than V(B) then the ratio to be calculated is

$$\delta = \frac{V(A)}{V(B)} \tag{5}$$

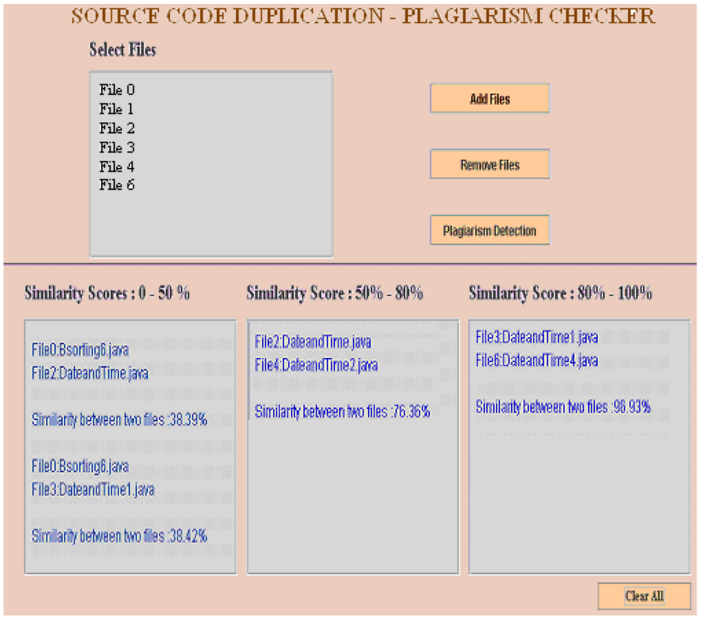
If $V(B)$ is less than $V(A)$ then ratio is

$$\delta = \frac{V(B)}{V(A)} \qquad (6)$$

Combination δ and delta gives a better similarity score.

IV. Experimental Results

The following figure 2 is the user interface for this application, finding near duplication in source code using fingerprint based distance measure approach. Here input files are added to the 'select files'list box on the click event of 'Add Files' button. To obtain the similarity scores on the output list box, click event of 'Plagiarism detection' button has to be activated. Thus on the output list boxes similarity scores of all the file pairs are displayed. Files will be compared and similar lines will be highlighted on the browser window.


SOURCE CODE DUPLICATION - PLAGIARISM CHECKER
Select Files
Add Files
Remove Files
Plagiarism Detection
Similarity Scores : 0 - 50 %
Similarity Score : 50% - 80%
Similarity Score : 80% - 100%
Clear All

```
File 0
File 1
File 2
File 3
File 4
File 6
```

```
File0:Bsorting6.java
File2:DateandTime.java

Similarity between two files :38.39%

File0:Bsorting6.java
File3:DateandTime1.java

Similarity between two files :38.42%
```

```
File2:DateandTime.java
File4:DateandTime2.java

Similarity between two files :76.36%
```

```
File3:DateandTime1.java
File6:DateandTime4.java

Similarity between two files :96.93%
```

Figure 2. GUI of Plagiarism Detector

Our plagiarism detection reliability measure is based on precision and recall. Precision represents the proportion of actual plagiarism pairs that have been detected in the program pairs detected overall. Recall represents the proportion of actual plagiarism pairs that have been detected in the total set of program pairs that are actually plagiarized[10].

Suppose we have a dataset with N program pairs and that the plagiarism detection algorithm labels program pairs either as plagiarized transcripts (positives) or as originals (negatives). Each pair in the test set is finally labelled as true positive TP, false positive FP, true negative TN or false negative FN. Then the total program pairs is the sum of all these components. We compute accuracy and recall [10] as

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (7)$$

$$precision = \frac{TP}{TP + FP} \qquad (8)$$

$$\mathrm{Re}\,call = \frac{TP}{TP + FN} \qquad (9)$$

In the below Tables 1, 2 and 3 shows the false positives, true positives, false negatives , true negatives, Accuracy, Precision and Recall rate of different set of programs.

TABLE I. Comparison of 20 Programs

20 programs	190 comparisons
TP	7
FP	2
FN	4
TN	177
Accuracy	96.84%
Precision	0.778
Recall	0.636

TABLE II. Comparison of 30 Programs

30 programs	435 Comparisons
TP	11
FP	3
FN	2
TN	420
Accuracy	98.85%
Precision	0.786
Recall	0.846

TABLE III. Comparison of 50 Programs

50 programs	1225 comparisons
TP	26
FP	7
FN	9
TN	1183
Accuracy	98.69%
Precision	0.788
Recall	0.743

Figure 3, shows the performance of the system based on the number of input programs to be tested and the total time taken for the completion of execution. We tested the system with different data sets. As the number of programs to be tested increases, it will not affect the similarity score between two programs. So the system can give almost steady accuracy in similarity score. Following figure 4 shows the effect of different data sets.

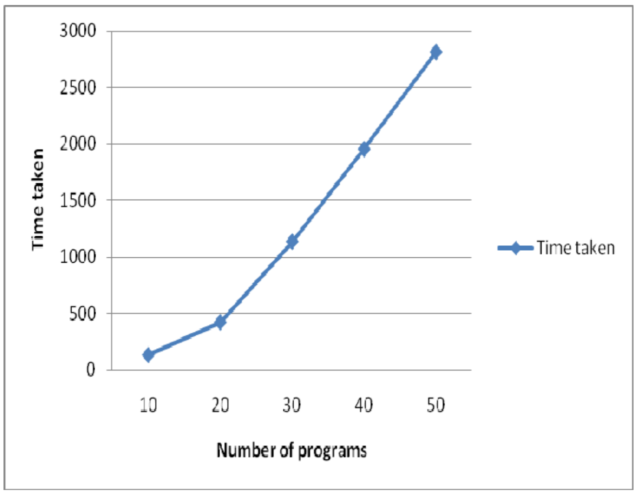


Figure 3. Graph of Number of programs- Time taken

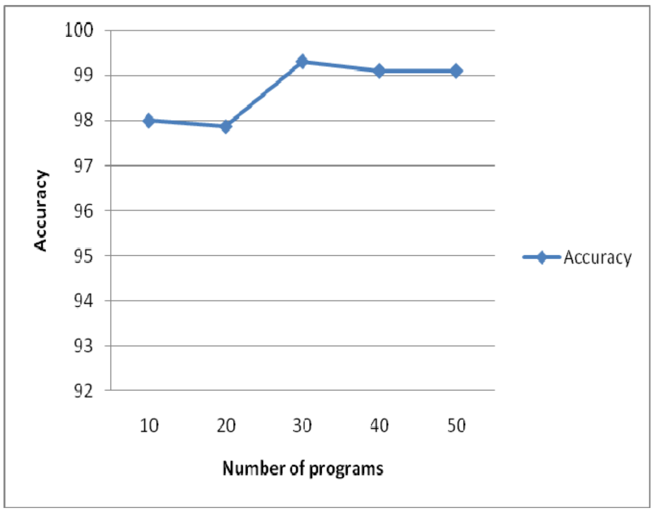


Figure 4. Number of programs vs Accuracy graph

V. Conclusion

Source code plagiarism is a serious problem in most of the academic institutions and research institutes. We have implemented a fingerprint based approach to detect source code plagiarism. It is possible to detect similarity between codes written in programming languages C, C++, Java, Matlab and C#. The algorithm is tested with large data sets and its performance is analyzed. In future, we are planning to incorporate more programming languages and to improve the efficiency of inter language similarity score

References

[1] S. Burrows, S. M. M. Tahaghoghi, & J. Zobel, Efficient and effective plagiarism detection for large code repositories," in 'G. Abraham and B.I.P. Rubinstein Editors, Proceedings of the Second Australian Undergraduate Students' Computing Conference (AUSCC04)', pp. 8–15 June 2004.

[2] J.I. Maletic, A. Marcus, "Using latent semantic analysis to identify similarities in source code to support program understanding," in: Proceedings of the 12th International Conference on 2000

[3] Marcus, J.I. Maletic, "Identification of high-level concept clones in source code, in: Proceedings of the 16th International Conference on Automated Software Engineering (ASE 2001), pp.107–114, May 2001.

[4] Richard M. Karp and Michael O. Rabin. "Pattern-matching algorithms," IBM Journal of Research and Development.

[5] GiladMishne and Maarten de Rijke "Source Code Retrievalusing Conceptual Similarity",

[6] Christian Arwin S.M.M. Tahaghoghi "Plagiarism Detection across Programming Languages ", Twenty-Ninth Australasian Computer ScienceConference (ACSC2006), Hobart, Tasmania, Australia, January 2006

[7] SeemaKolkur, Madhavi M. Naik (Samant) "A Tool to Detect Program Plagiarism using Expression List method" International Transactions on Applied Sciences and Technology (ITAST) Vol 1 No 1 May, 2011

[8] M. Halstead. Natural laws controlling algorithm structure. ACM SIGPLAN Notices,7(2):19–26, 1972.

[9] M. Halstead. Elements of Software Science (Operating and programming systems series). Elsevier Science, New York, USA, 1977.

[10] L. Prechelt, G. Malpohl, and M. Philippsen. JPlag " Finding plagiarisms among a set of programs," Technical report 2000-1, Fakultat fur Informatik, University Karlsruhe, Germany, 2000.