International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Software Source Code Plagiarism Detection Using Latent Semantic Analysis

Bhramadeo Vishnu Deokate [1], Dinesh Bhagwan Hanchate[2]

[1]ME Second year Student Computer Engineering, Vidya Pratishthan's College of Engineering, Baramati, Pune, India

[2]Professor, Department of Computer Engineering, Vidya Pratishthan's College of Engineering, Baramati, Pune, India

Abstract: ***Plagiarism is a main problem in academia. Academics often use plagiarism detection system or tools to detect similar source-code files in a program. Similar files are detected, the system proceeds with the investigation process which involves detecting the similar source code fragments with in evidence for proving plagiarism. This system describes tools that can be integrated with existing plagiarism detection to improve plagiarism detection performance. The system can be implements new thing between source-code to gathering corpus files. The source code content indicates the relative importance given source-code fragments across files in a corpus. This system is done using the latent semantic analysis, retrieving data technique, and detecting plagairism is important within the specific files under relation the corpus.***

Keywords: Software plagiarism detection, dynamic code identification, Latent Semantic Analysis, cosine-similarity

1. Introduction

In the computer science field, there are number of probabilities that code theft problems are occurred. In the education system, students submit their projects work and the programming assignments, there may be possibility of duplication in source code. The manually plagiarism detection in the source code is a very difficult task. Mostly the people in computer science are using programming assignments of another one [7].

In the plagiarism detection process, there are two parts. In the first part, it generates a representation from a given program. The intermediate representation is used for evaluating the similarity between two programs or projects. A token sequence is often used by intermediate representation. Plagiarism detection system uses the token sequence. In the second part, system evaluates several techniques and methods are developed for identifying similar code software projects with similar codes [5].

In plagiarism is easy to do, but it is not easy to detect. Usually, when students solve the same problem by using the same programming language source code, there is a high possibility that their assignment solutions will be more similar. Strategies of source code modification that can be used to mask source code plagiarism. Examples of such strategies are renaming identifiers and combining several segments copied from different source code files. These modifications increase the difficulty in assignment plagiarism [9].

In source-code files, similarity may be suspicious or innocent. For example, similarities between source-code files are existed innocently due to source-code. The students given during classes, method suspiciously same source-code. Files share source-code fragments that are distinct in program logic, content and functionality from source-code fragments found in the corpus database [1].

2. Related Work

The plagiarism detection approaches mainly show potentially plagiarised source code pairs. A system determines which case pairs are likely to be plagiarised by analysing the similarity levels in the program code. If the similarity level between a case pair is high, the system indicates the case pair is suspicious and suggests to the user. This pair may require other investigation [8]

Table 1: Comparison of four plagiarism detection tool

Tools	JPLAG	SIM	MOSS	PLAGGIE
Open source	NO	YES	NO	YES
Local/Online	Web	Local	Web	Local
Codebase /File	Code base	File base	Code base	Code base
Language Support	6	5	23	1
Founded Year	1996	1989	1994	2002
Founded By	Guido Malpohl	Dick Grune	Aikenetal	Ahtiaine netal

Detecting Source Code Re-Use: It detects source code, reuses in the many programming languages or projects. The main target of this is to provide new technologies to detect source code duplication. Using these tools, it decides whether the source code has been reused or not. This tool compares two different source codes at the level of methods and functions written in different programming languages [5].

Plagiarism Detection Engine For Java Source Code: Here in this research article, authors Ameera Jadalla & Ashraf Elnagar developed PDE4Java model that is basically used for plagiarism detection in java source code. Method used in this research is data mining, clustering, N-Gram, tokenization. At the end of the research it is showing performance of system similarity pair in programing code [10].

3. Proposed System

This system is used to plagiarism detection in source code. In this section some previous work plagiarism detection techniques are reviewed.

Data Collection: It is challenging work to detect the duplication of source code or a nearly duplication of code. The researchers done on the basis of problem on theft code. The mainly datasets are using two different project and programs.

Extension work in Academia:
In academic institutions programming task is copying someone work else that is another student's work from such as books, program, and failing to provide of the source. The act of plagiarism is still regarding as whether it was intentional or unintentional.

Source-Code Similarity to Source-Code Plagiarism: Plagiarism detection in programming assignments is a task follows higher education system carry out, people reuse source-code. The particular source-code is not their own, once similarity between assignment is found. The education system proceeds with the task of similarity investigation between source codes. The investigation process involves comparing the detected source code files for plagiarism by examining their similar source-code fragments.

Resuse code in java project: In java source code, assignments for self-plagiarism occurs when student reuses function, methods and parts of code previously submitted for academic submission of it as part of other assignment without providing intimention of this work. In project programming modules where source-code reuse is self-plagiarism, may not be considered. Collusion occurs when more than one student used same source code, but educational assignment requires students to work individually [1].

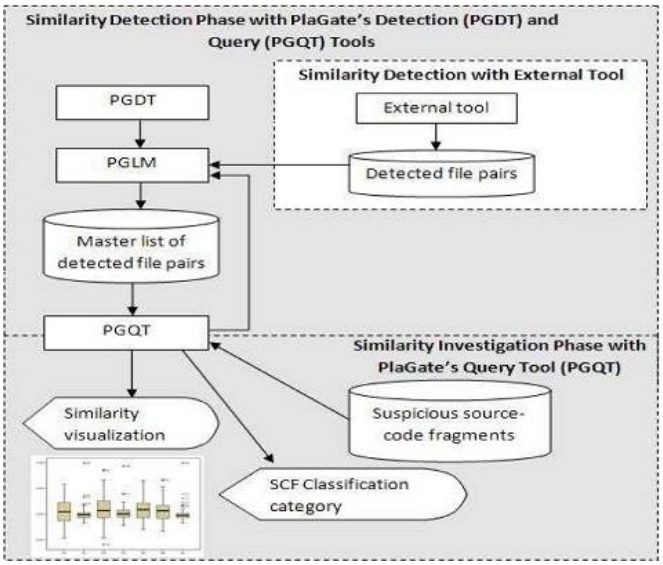

Figure 1: System Architecture.

The similarity detection functionality and steps of system described as follows: [Fig. 1]

1) Similarity detection is performing these tools.
2) External tool outputs make a list detected file pairs based on a given threshold value.
3) PlaGate's List Management component (PGLM) works to stores the detected similar files in the master list.
4) Detecting similar file compars with plagate detection tool in master list and similar file pairs.
5) Plagate list management deletes file pairs occurring more than one from the master list.
6) Plagate query tool takes as input first query file from each file pair and store in the master list.
7) Plagate query tool store a query file and detects file pairs based on a given threshold value.
8) Including updates the master list the detected file by PGQT and displaying similar file pairs.

4. Implementation Details

4.1 Latent Semantic Analysis

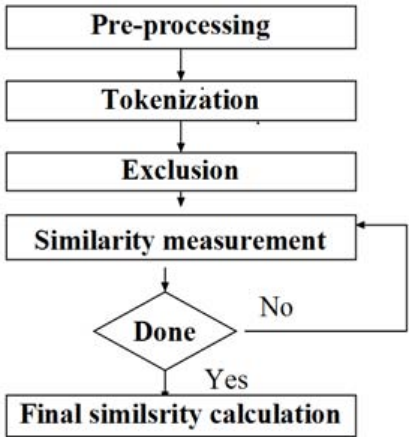

Figure 2: Similarity detection approach

LSA-Processing: Latent Semantic Analysis is automatic statistical, mathematical technique and methods for inferring and extracting connection of contextual usage into programs. It is neither artificial intelligence nor traditional natural language processing program. It uses not humanly constructed dictionaries, semantic word, syntactic parsers, morphologies, knowledge bases, grammars and takes as input raw texts. Parsed convert words defined as unique character strings base separated into passages such as paragraphs or sentences [2].

1) Remove token.
2) Remove comment.
3) Remove all the numbers.
4) Remove non-frequent tokens.
5) Create union set of all token from each source code file.
6) Creating term by file matrix.
7) Cosine similarity between two vectors.
8) Collecting union of suspicius file and union of innocent file.
9) Creating a report genration.

Applying Latent Semantic Analysis to a Source-Code Corpus

Latent Semantic Analysis (LSA) is a variant of the information retrieval. The application of LSA for information retrieval dates back to 1988. LSA is also known as Latent Semantic Indexing (LSI), and the term LSI is used for tasks releting the indexing or releted data, where as term LSA is used for tasks concerned with everyone else, such as automatic essay grading text, source code summarisation in a corpus.

4.2Mathematical Model

Input Set:

S= $\{s_i ; 0 < i < n\}$ -set of source code file
Where,
n=no of source code files

2. Q= $\{q_i ; 0 < i < n\}$ -set of token in quer Source code files.

Preprocessing Sets:

1. F= $\{f_i ; 0 < i < n\}$ - set of fragments
Where,
n=no of fragments
$\exists f_i \mid f_i \in S$

2. T= $\{ t_i ; 0 < i < n\}$ - set of tokens
Where,
n=number of tokens
$\exists\ t_i \mid ti \in F$

3. M= $\{(t_i , s_j); 0 < i < n , 0 < j < m \}$ - set of file term pair
Where,
n =no of token/terms
m=no of file

$$\exists(t_i , s_i) \mid (t_i , s_i) \begin{cases} \text{true: } ti \in si \text{ and } \mid ti \mid > 0_t \\ false, otherwise \end{cases}$$

4. S'= $\{s'_i ; _0 < i < n\}$ - set of suspicius file
Where,
n=no of suspicius files

$$\exists s'i \mid s'i \begin{cases} true, cosine(s', Q) \geq \mathbb{0} \\ false, otherwise \end{cases}$$

$\mathbb{0}$ =Thershold

5. I = $\{(i_j;0 <I <n\}$ - set of innocent file
Where,
n=no of innocent file

$$\exists\ i_j \mid i_j \begin{cases} \text{true: } \mid () \mid > thr \\ false, otherwise \end{cases}$$

Thr=$\mathbb{0}$

Cosine(S', Q) = $\sum S'ij * Qj \ / \ \sum\sqrt{(S'ij)^2} . \sum\sqrt{(Q_j)^2}$

Output Sets:

R = $\{(Q',S'_i,fj ;0<n;0<j<m)\}$ - Set of fragments
Where,
n-no of suspius file
m=noof matching frgments

$$\exists((Q',S'i,fj) \begin{cases} true, fj > 0 \\ false, otherwise \end{cases}$$

Q'=Query source file
S'=ith source code file
F_j=j^{th} fragments

Let Y be the system, mathamatically represent Y used by set theory,

Y= {S,Q;F,T,M,S',I;R}

5. Output Design and Results Analysis

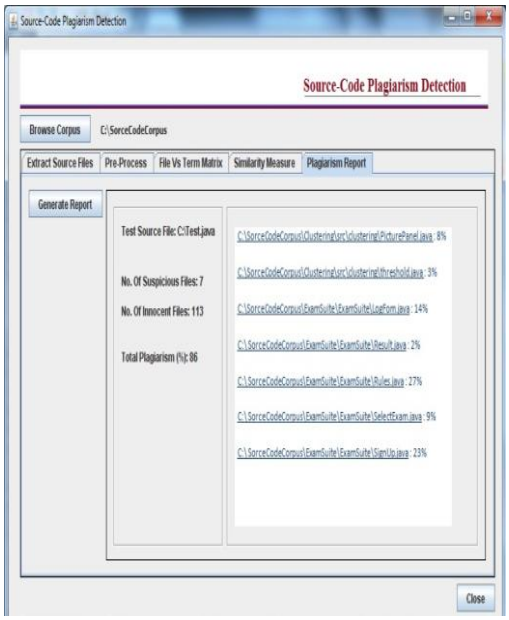

Figure 3: Output Design

Similarity between two java source code files is investigate the fragments searching within the files for common characteristics. Investigation involve, scrutinizing similar source-code content after judging whether the suspicious file or innocent file. If the significant rate amount of suspicious, file similarity is found then the files under investigation becomes considered as suspicious file.

5.1 Result Discussion

The files are present in the total number of files in a corpus. Numbers of terms are found in an entire source-code corpus after performing preprocessing. Number of matching or suspicious file pairs, the total number of file pairs that are detected in a corpus and finalised as suspicious. System reads and parses the files to compare them. Files are excluded from the comparison process by this system.The displaying query source file and similar detection files .precisions are calculted detected files and corpus file[Table.2].

Table 2: Results Analysis

#Source File In Corpus	#Query Source File	#Source Files With Plagirism	Detected File	Precision
235	7	6	4	0.67
400	16	14	13	0.92
650	20	17	16	0.94
460	25	22	21	0.95
550	22	20	19	0.95

5.2 Analysis Graph

In graph table, the existing system source code plagiarism is nearly less accuracy, but the proposed system exactly detects plagiarism and its accuracy is greater than existing system. The matching or suspicious file shows individually. Total

number of file to displaying that detected in a corpus and finalised as suspicious. System imports good results and accuracy in assignment.

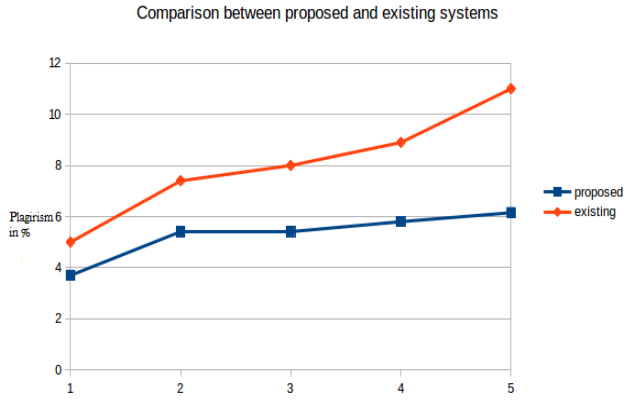

Figure 4: Analysis graph

6. Conclusion

LSA used in plagiarism detection system, source code and project has been introduced. The problem of the source-code plagiarism becomes more complicated task with the availability of the internet and the growing more web sites. To detect programing plagiarism, an efficient work to optimize the speed and the accuracy of the detection process is important as well as required. This is a difficult and challenging task which is an extension of this work.

7. Acknowledgements

This paper would not have been written without the valuable advices and encouragement of Prof. D.B. Hanchate, guide of ME Dissertation work. We thank to Prof. P. M. Patil and Prof. S. S. Nandgaonkar, Head of Department and Hon'ble principal Prof. V. U. Deshmukh, for their valuable support and for giving an opportunity to work on this paper.

References

[1] Yoon-Chan Jhi, Xinran Wang, Xiaoqi Jia, Sencun Zhu,Peng Liu, Dinghao Wu, "Program Characterization Using Runtime Values and Its Application to Software Plagiarism Detection," IEEE TRAN, ON Software Engineering, VOL. 10.1109/TSE.2015.2418777.
[2] B. S. Baker, "On finding duplication and near-duplication in large software systems," in Proceedings of 2nd Working Conference on Reverse Engineering (WCRE '95), 1995, pp. 86–95.
[3] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees." in Int. Conf. on Software Maintenance, 1998.
[4] K. Kontogiannis, M. Galler, and R. DeMori, "Detecting code similarity using patterns." in Working Notes of 3rd Workshop on AI and Software Engineering, 1995.
[5] J. Krinke, "Identifying similar code with program dependence graphs." in Proceedings of Eighth Working Conference on Reverse Engineering (WCRE '01), 2001, pp. 301–309.
[6] T. Kamiya, S. Kusumoto, and K. Inoue., "CCFinder: a multilin-guistic token-based code clone detection system for large scale source code." IEEE Transactions on Software Engineering, vol. 28,no. 7, pp. 654–670, 2002.
[7] M. Gabel, L. Jiang, and Z. Su, "Scalable detection of semantic clones," in Proceedings of the 30th International Conference on Software Engineering (ICSE'08), 2008, pp. 321–330.
[8] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone related bugs," in Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering, ser. ESECFSE '07, 2007, pp. 55–64.
[9] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in Proceedings of the 29th International Conference on Software Engineering (ICSE '07), 2007, pp. 96–105.
[10] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," The Univeristy of Auckland, Tech. Rep.148, Jul. 1997.
[11] C. S. Collberg, C. Thomborson, and D. Low, "Manufacturing cheap, resilient, and stealthy opaque constructs," in Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98), 1998, pp. 184–196.
[12] C. Collberg, G. Myles, and A. Huntwork, "Sandmark–a tool for software protection research," IEEE Security and Privacy, vol. 1, no. 4, pp. 40–49, 2003.
[13] M. Madou, L. Van Put, and K. De Bosschere, "Loco: An interactive code (de)obfuscation tool," in Proceedings of the 2006 ACM SIG-PLAN symposium on Partial evaluation and semantics-based program manipulation (PEPM '06), 2006, pp. 140–144.
[14] H. Tamada, K. Okamoto, M. Nakamura, and A. Monden, "Dynamic software birthmarks to detect the theft of Windows applications," in Int'l Symp. On Future Software Technology (ISFST), October 2004.
[15] F. Zhang, Y. Jhi, D. Wu, P. Liu, and S. Zhu, "A first step towards algorithm plagiarism detection," in Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA '12).ACM, 2012, pp. 111–121.
[16] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection," in Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014), November 2014.

Author Profile

Mr. Bhramadeo Vishnu Deokate received B.E in Information Technology from VPCOE Baramati, Savitribai Phule Pune University in 2014. Now pursuing Master of Engineering in Computer Engineering at Vidya Pratishthan's college of engineering, Baramati, Savitribai Phule Pune University. Email:deokatebv@gmail.com.

Prof. Dinesh Bhagwan Hanchate received B.E in Computer Engineering from Walchand College of Engineering, Sangli (1995), Lecturer in Gangamai College Of Engineering, Dhule (1995-96), Lecturer in S.S.V.P.S. B.S.D. College Of Engineering, Dhule In Computer & IT dept. (1996-2005), M.Tech. Computer from Dr.Babasaheb Ambedkar Technological University, Lonere (2002-05), Currently Asst. Prof. Computer Engineering, in Vidya pratishthan's College Of Engineering, Baramati, currently doing research at SGGS Institute of Technology and Engg, Nanded affiliated to SRTMU, under the guidance of Dr. Bichkar R.S.G.H. Raisoni College of Engineering and Management,Wagholi,,Pune.
Email:dineshbhanchate@gmail.com.