Efficient Source Code Plagiarism Identification Based on Greedy String Tilling

Khurram Zeeshan Haider, Tabassam Nawaz, Sami ud Din, Ali Javed

47080, University of Engineering and Technology, Taxila, Pakistan

ABSTRACT
There is no doubt that use of computers and internet has given benefits in abundance but on the other hand with other harms it has made easier to produce plagiarized work. Truthfully plagiarism can be encountered in any field and should be condemned at every level. The focus of this paper is on source code plagiarism. In college and universities students copy the programming assignments of each other similarly the employees claim the code belong to their by reproducing it or committing some other means of plagiarism. The lot of work has been made in the prevention of source code plagiarism. This paper spotlights the work which already has been done and a new method has been proposed in this research. The proposed method is based on GST.

Key words:
GST- Greedy String Tiling, Plagiarism, Algorithm, Source code tokenization.

1. Understanding Plagiarism

Plagiarism can be understood as___ with intent or by mistake replicating (making a copy, replacing the words, translating, etc) work that was created by someone else without any recognition for the purpose to achieve academic promotion. Tolerating such duplication to occur may also comprise plagiarism.
Plagiarism in a work consists of: language (communication), thoughts, results, written material, graphic illustration, computer related programs, drawings, charts, graphics, artistic work, knowledge, teachings, on paper work, electronic work, or any other innovative and fresh work produced and presented by anyone else.

1.1 Self-Plagiarism

Self-plagiarism takes place when a research scholar reclaims whole or some portion of his/her personal work which was formerly evaluated for educational benefit and claims it as portion of another work without of any recognition of this fact. Some university rules concentrate on the matter of self-plagiarism. As, the Department of English at the University of Bristol has an undergraduate handbook with a section called "Advice on study skills". It is stated in their handbook (2005).

"The Department will also regard, for assessment purposes, the re-use of your own essays as 'self-plagiarism'. While you may return to the same subjects or works in essays for different units, or within a unit, to avoid self-plagiarism you must not only avoid the verbatim or near-verbatim re-use of previously submitted essays in part or whole, but also ensure that your return to the same subjects or works involves a rethinking of your ideas. Self-plagiarism is also a serious disciplinary matter."

1.2 Plagiarism methods

In essay assignments, research scholars are compulsory to acknowledge the source and belonging of the work that was not initially presented by them, they are advised to use quotation marks where applicable with suitable references. But some plagiarism practices in essays can be:

- Students/research scholars copy the work from different locations/sources without of proper referencing and citation.
- Students/research scholars summarize the work from one or more sources without of proper referencing and citation.
- Thieving other's material and claiming it as it was his/her own is a serious act of plagiarism.
- A common practice is of copying another scholar's work as a whole or some portion of it then submitting it with their name. It is also important to note that a student/research scholar who grants permission to someone else for reproducing his/her work is also committing plagiarism.
- Students/research scholars if pay someone to do work for them is again an act of plagiarism.
- Students/research scholars sometimes resubmit their own work which falls under self-plagiarism.
- When two or more Students/research scholars make assignment with their collective effort when they were advised not to do so.

1.3 Plagiarism method in source code

The assigned work related to source code also the Students/research scholars are always directed to mention the proper recognition if they use any code which is

Manuscript received December 5, 2010
Manuscript revised December 20, 2010

originally related to someone else. They should acknowledge the work in the coding as well as in the documentation of the assignment. Cosma and Joy (2006) have mentioned some common source-code plagiarism practices which are:

- Representation of source-code without any changing and proper acknowledgement.
- Taking some portion of the source-code which was authorized by other party without of any proper acknowledgement.
- Also converting/reproducing the whole of part of someone else's source-code in other but similar programming language.
- The use of code generating software without mentioning this.
- Students/research scholars if pay someone to do coding work for them is again an act of plagiarism.
- When two or more Students/research scholars make programming assignment with their collective effort when they were advised not to do so.

1.4 Tackling with the source-code plagiarism

The following is the checklist of some measures aimed for colleges/universities faculty members, students, research scholars and programming business companies to tackle with the source-code plagiarism:

- One should teach and train himself/herself about the plagiarism.
- Always follow the instructions and regulations of your institution on the plagiarism.
- Teach and train your students and subordinates in the organization who are directly involved with the programming tasks for example developers and programmers. Make it sure that your students and subordinates in the organization understand the consequences of this theft and steeling and unauthorized collaboration in preparing the source-code assignments.
- Teachers must offer the programming assignments and evaluations methodology which should be not easy to plagiarize and they should encourage the use of source-code plagiarism prevention tools such as "EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON GREEDY STRING TILLING"
- Convey your students and subordinates in the organizations that what they should consider as plagiarism and what is not the part of plagiarism.
- Be aware of the cheat sites where people plagiarize their work or hire someone else for their work which is again a plagiarism.
- You should properly document the conversation and communication with the students or subordinates in organization which would be useful in court.
- It is most important to always take action when you spot plagiarism.

2. Problem Statement

Source code plagiarism can be experienced in institutions or even at commercial business points. It is considered as against the law if the others work is copyright or bad mannered practice otherwise in the society. Manually identification of source code plagiarism in educational institutions is not an easy task; the reason behind this is possibility of same copied source code but with different patterns. Some diplomatic changing are commonly in practice for making the code look different from the copied one but result in the same output. It actually decreased the efficiency of plagiarism detection tool. It could be:

- Making variable names different
- Placement of functions/procedures at different position as compared to the original work
- toggling lines of codes

Those grounds become the key initiative of developing "EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON GREEDY STRING TILLING".

3. Aims & objectives of the proposed system

3.1 Clone finding

Clones are exact (identical), renamed (just name is different), gapped (some portions/patched have been used) [1, 2].

- Textual resemblance
- Token (It has proven to cover all above clones detection in this paper)

3.2 False positive clone finding

- Consecutive method declarations
- Consecutive method invocations
- Consecutive if-statements and if-else statements
- Consecutive case entries
- Consecutive variable declarations

Moss, Sherlock, Copy-Paste Detector (CPD) and JPlag are most common tools for source code plagiarism detection available [3]. Below is the table comprising of the comparisons among them in terms of ease, outcomes and technique.

	MOSS	SHERLOCK	CODEMATCH	COPY/ PASTE DETECTOR (CPD)	JPLAG
Start Year	1994	1994	2005	2003	1997
Cost	Free but a user account is needed	open sourced	It is a Commercial tool. Free on less than 1 M-byte size	open source	Free but a user account is needed
Open Source	NO	YES	NO	NO	NO
Safety	Sign in required	Executes at local machine	Executes at local machine	Executes at local machine	Sign in required
Speed	Fast	more files requires more time	more files requires more time	Fast	Fast
Service	Internet	Stand-alone	Stand-alone	Stand-alone	Web service
Interface	Graphic user interface	Graphic user interface	Graphic user interface	Graphic user interface	Graphic user interface
Necessities	A script for UNIX or Windows	JDK 1.4 or higher	---	JDK 1.4 or higher	Web browser, JRE, Java 1.5 or higher
outcome storage	Storage is at Remote server	Storage is at local machine	Storage is at local machine	Storage is at local machine	Storage is at local machine
Algorithms	Winnowing	Token matching	String matching	Greedy String Tiling	Greedy String Tiling

The following outcomes were discovered made from the comparison:

- Greedy String tiling is the best option for source code plagiarism detection. The work "EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON GREEDY STRING TILLING" is based upon it.
- There is a need for some better solution regarding plagiarism in terms of ease of use, reliability, fastness and trustworthy which are main features of the proposed system.
- Results must be more generic and understandable so that the source code plagiarism can be prevented so the results in the proposed system are more statistical and understandable.

4. Methodology

Two phases have been adopted for the research:

4.1 Phase 1:

Parsing & Pre-tokenization stages:

- Read source code
- Remove strings
- Remove comments
- Separate statements
- Generate tokens which include[4]:
    a. Identifiers or Symbols (variables, types, functions, and labels)
    b. Keywords (these are language reserve words e.g. for, while, if etc for C++ language)
    c. Literals (these are constants)
    d. Operators (e.g. +, -, / etc)
    e. Punctuators (these have syntactic and semantic significance for the compiler e.g. and, not, bitand, xor etc for C++ language)

4.2 Phase 2:

Greedy String Tiling (an algorithm) application stages:

- Input Tokens
- Apply Greedy String Tiling (The algorithm have been explained later in this paper)
- Obtain Results

Some other results are included which are to detect "false positive clone detection". Method is to count the token if found consecutive e.g. for "consecutive if" the psudocode is:

```
GETCONSECTIVEIF( TOKENFILE )
{
   LOOP INDEX = 1 TO TOKENFILE.LENGTH
   {
         IF( TOKENFILE[INDEX] = “if” THEN
         ADD 1 IN  CONSECTIVE_IF
   }
RETURN CONSECTIVE_IF
```

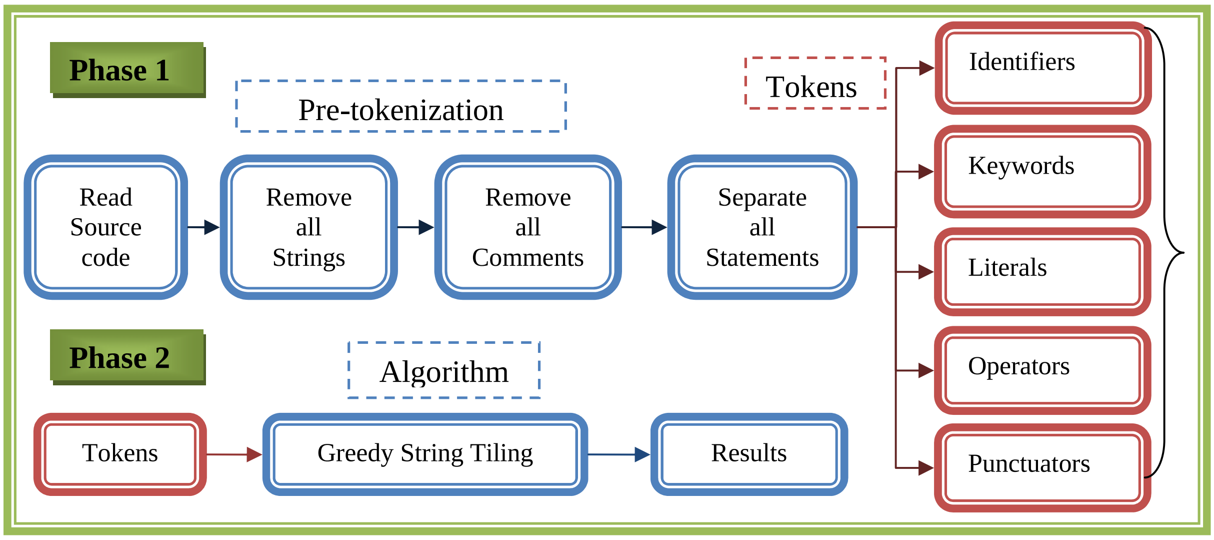

Figure 1: Tokenizer and GST structure

4.3 Other Statistics

- INCLUSION
  $Fn_{inc}(T_1, T_2) = M(T_1 \cap T_2) / Min(M(T_1), M(T_2))$
- COVERAGE
  $Fn_{cov}(T_1, T_2) = M(T_1 \cap T_2) / Max(M(T_1), M(T_2))$
- SIMILARITY
  $Fn_{sim}(T_1, T_2) = M(T_1 \cap T_2) / M(T_1 \cup T_2)$

Elaboration of terms used in above functions:

- Fn_{inc} = Inclusion function
  Inclusion defines the measure of information of one file into another. If $Fn_{inc}(T_1, T_2) = 1$ then it means the smaller work is included into larger work.
- Fn_{cov} = Coverage function
  It calculates the coverage of the larger work by the smaller work
- Fn_{sim} = Similarity function
  It measures the amount of similarity in both files. If $Fn_{sim}(T_1, T_2) = 1$ then it depicts the both files are identical according to the token matrix M.

Sr. #	*Term or Symbol*	*Elaboration*
1	$M(T_1)$	Matrix of the number of tokens in first file
2	$M(T_2)$	Matrix of the number of tokens in second file
3	$\cap$	intersection
4	$\cup$	union
5	$M(T_1 \cap T_2)$	Matrix of the number of common tokens in both files
6	$M(T_1 \cup T_2)$	Matrix of the union of all tokens in both files (same token cannot be repeated)
7	$Min(M(T_1), M(T_2))$	Minimum in $M(T_1)$ and $M(T_2)$
8	$Max(M(T_1), M(T_2))$	Maximum in $M(T_1)$ and $M(T_2)$

5. Greedy String Tiling

Psuducode for GST in proposed methodology is:

```
SEARCH-PIECE P := FIRST-SEARCH-PIECE
TERMINATE := FALSE
DO AGAIN //loop
        $P_{MAX}$ := SCANPATTERN(P)
        IF $P_{MAX} > 2 \times P$ THEN P := $P_{MAX}$
        ELSE
                MARKPIECES(P) /* CREATE TILES */
                IF P > 2 × MIN_MATCH_PIECE THEN P := P DIV 2
                ELSE IF P > MIN_MATCH_PIECE THEN P := MIN_MATCH_PIECE
        ELSE TERMINATE := TRUE
UNTIL TERMINATE
```

Here; P_{MAX} = maximum match piece (The match is as long as it could be until the end of pattern came across or it's already marked)

MARKPIECES(P) ___ Its tiling process when tile is marked from P_{MAX} (maximum match), it becomes unavailable for future comparisons.

It's important to note that the proposed system is fully effective against the source code plagiarism when someone tries the following techniques:

- Making variable names different as compared to the original work
- Placement of functions/procedures at different position as compared to the original work
- shuffling the lines of codes (LOC)

6. Experimental Results

Sr. #	*No. of tokens*	*Time in ms*
1	52	0.071428571
2	2735	0.024163569
3	1394	0.04779607
4	723	0.059612321
5	1729	0.041887945
6	1326	0.048977695
7	890	0.056658258
8	1108	0.052817977
9	1922	0.038490773

Nine different source code files of C++ language were experimented for the plagiarism detection (On a Intel Core 2 Duo 2GHz with 2 GB RAM). First the code was tokenized as discussed earlier (figure 1) and then GST was applied. It showed results in terms of similarity in amazingly less time when applied to the scenario presented in figure 1. The data was recorded and entered in columns and rows of Microsoft Excel for the chart to view the trend of the tokens of a source code file over time.

The experimental values of tokens are evaluated by category using vertical bars with the time in milliseconds. The histogram shows the variety in number of tokens (generated from the source code files) which have been taken into account for comparison purpose to detect plagiarism by its vertical rectangles.

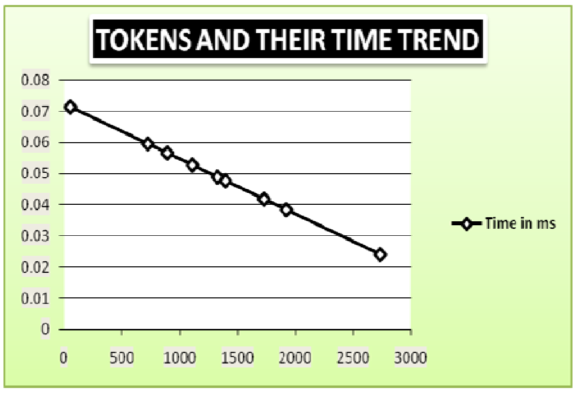

Figure 2: Tokens and their time trend

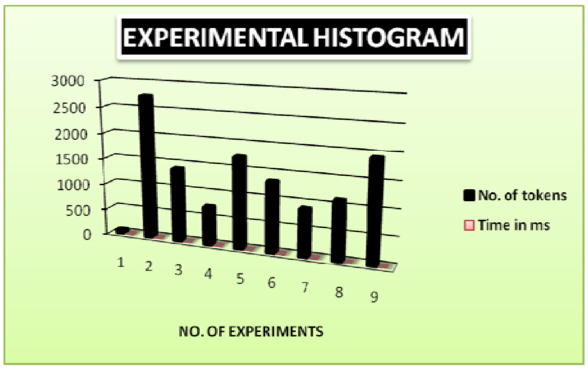

Figure 3: Experimental histogram

Sr. #	*Findings*	*Results*
1	Inclusion, coverage and similarity	Between 0 and 1
2	Consecutive: • method declarations • method invocations • if-statements and if-else statements • case entries • variable declarations	The count of consecutive occurrence(s).
3	Similarity index (from GST)	Percentage

7. Conclusion

The conclusion is summarized in the following points:

1. The proposed technique is effective if someone makes variable names / function names different etc, tries to defeat by placing function names below or higher as compared to the original code or toggle the LOCs.

2. The proposed technique possesses all features which other tools have and some additional features like showing the false positive clone, inclusion, similarity and coverage.
3. Tokens and their time trend shows it promising fast.
4. Summarizing the all features we can claim that it is a fine contribution towards source code plagiarism prevention.
5.

References

[1] Heejung Kim, Yungbum Jung, Sunghun Kim and Kwangkeun Yi, Clone Detection by Comparing Abstract Memory States, ROSAEC Research On Software Analysis For Error Free Computing, ROSAEC MEMO 2010-008 March 5, 2010

[2] E. Kodhai, S. Kanmani, A. Kamatchi, R. Radhika and B. Vijaya Saranya, CLONEMANAGER: A TOOL FOR DETECTION OF TYPE1 AND TYPE2 CODE CLONES, Information Processing and Management International Conference on Recent Trends in Business Administration and Information Processing, BAIP 2010, Trivandrum, Kerala, India, March 26-27, 2010. Proceedings

[3] Vaughn M. Segers, James, An Online System for Plagiarism Detection, University of the Western Cape, Private Bag X17 Bellville, 7535, South Africa 2008

[4] Oege de Moor, Michael Schwartzbach, Compiler Construction: 18th International Conference, 2009

[5] Jeong-Hoon Ji, Su-Hyun Park, Gyun Woo*, and Hwan-Gue Cho, Generating Pylogenetic Tree of Homogeneous Source Code in a Plagiarism Detection System, International Journal of Control, Automation, and Systems, vol. 6, no. 6, pp. 809-817, December 2008

[6] Michel Chilowicz, Étienne Duris, and Gilles Roussel, Finding Similarities in Source Code Through Factorization, Electronic Notes in Theoretical Computer Science, Volume 238, Issue 5, 10 October 2009, Pages 47-62

[7] Aleksi Ahtiainen, Sami Surakka1, Mikko Rahikainen, 2007, Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises, Proceedings, Koli Calling

[8] Department of English, University of Bristol (2005). *Undergraduate Handbook 2005-06*. Retrieved February 20, 2006.

[9] Sanjay Goel, Deepak Rao et. al, Plagiarism and its Detection in Programming Languages

[10] Maxim Mozgovoy, Sergey Karakovskiy, and Vitaly Klyuev, October 10 – 13, 2007, Fast and Reliable Plagiarism Detection System, Milwaukee, WI 37th ASEE/IEEE Frontiers in Education Conference

[11] Bob Zeidman, Tools and algorithms for finding plagiarism in source code, Dr. Dobb's Journal July, 2004

[12] Samuel Mann and Zelda Frew, Similarity and originality in code: plagiarism and normal variation in student assignments, Department of Information Technology, Otago Polytechnic, Dunedin, New Zealand

[13] Christian Arwin, S.M.M. Tahaghoghi, Plagiarism Detection across Programming Languages, School of Computer Science and Information Technology RMIT University, GPO Box 2476V, Melbourne 3001, Australia.

[14] Sebastian Niezgoda and Thomas P. Way, SNITCH: A Software Tool for Detecting Cut and Paste Plagiarism, Applied Computing Technology Laboratory Department of Computing Sciences, Villanova University, Villanova, PA 19085

[15] Fintan Culwin, Anna MacLeod, Thomas Lancaster, SOURCE CODE PLAGIARISM IN UK HE COMPUTING SCHOOLS, School of Computing, South Bank University, Borough Road

[16] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken, Winnowing: Local Algorithms for Document Fingerprinting, University of Illinois, Chicago; Computer Science Division, UC Berkeley

[17] Peter Vamplew and Julian Dermoudy, An Anti-Plagiarism Editor for Software Development Courses, School of Computing, University of Tasmania, Private Bag 100, Hobart 7001, Tasmania

[18] J.-H. Ji, G. Woo, S.-H. Park, and H.-G. Cho, “An intelligent system for detecting source code plagiarism using a probabilistic graph model,” Proc. of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM Posters 2007, pp. 55-69, July. 2007.

[19] J.-H. Ji, S.-H. Park, G. Woo, and H.-G. Cho, “Evolution analysis of homogenous source code and its application to plagiarism detection,” Proceedings of the FBIT2007, pp. 813-818, October 2007.

[20] Paul Clough July 2000, Plagiarism in natural and programming languages: an overview of current tools and technologies, Department of Computer Science, University of Sheffield

Khurram Zeeshan Haider completed his Masters in Computer Science from Punjab University College of Information Technology, Lahore, Pakistan. He is student of MS (Software Engineering) at UET Taxila. His areas of interest are Programming Languages, Compiler Optimizations, Greedy Algorithms, Networking, Database Management Systems (DBMS), Software Development and Software Quality Assurance.

Tabassam Nawaz completed his MS Computer Engineering in 2005 from CASE (Center for Advance Studies in Engineering), Islamabad, Pakistan and subsequently he completed his Ph.D in 2008. He has published number of papers in different Journals. His research areas include Software Engineering, Programming Languages, Data Structure, Computer Graphics, Networks and Digital Image Processing. He is the Chairman of Software Engineering Department at University of Engineering & Technology Taxila, Pakistan.

Sami ud Din completed his M.Sc in Electrical Engineering, specialized in Computer Engineering in 2005 from University of Engineering and Technology Taxila, Pakistan and subsequently he completed his Ph.D degree in 2009. He has published number of papers in different Journals. The field of his interest includes design and development of real time imaging and video systems, image security and steganography. He currently holds a position of a senior researcher in an R & D Organization. He has also been teaching at UET Taxila as a visiting faculty member.

Ali Javed received his MS degree in Computer Engineering from the University of Engineering & Technology Taxila, Pakistan in February, 2010. He has received B.Sc. degree in Software Engineering from University of Engineering & Technology Taxila, Pakistan, in September, 2007. His areas of interest are Digital Image Processing, Computer vision, Video Summarization, Machine Learning, Software Design and Software testing. He is serving as a Lecturer in Software Engineering Department at University of Engineering & Technology Taxila, Pakistan since September, 2007.