Plagiarism and its Detection in Programming Languages

Sanjay Goel, Deepak Rao et. al.

Abstract

Program similarity checking is an important application of programming education fields. The increase of material now available in electronic form and improved access to this via the Internet is allowing, with greater ease than ever before, plagiarism that is either intentional or unintentional. Due to increased availability of On-line material, people checking for plagiarism are finding this task increasingly harder. Techniques for detecting plagiarism in programming languages are discussed in this report to provide the reader with a comprehensive introduction to this area.

Keywords

Source-code, Plagiarism, Reuse, Greedy String Tiling

1. Introduction

Plagiarism in programming assignments is an inevitable issue for most academics teaching programming. The Internet, the rising number of essay banks, and text-books are common sources used by students to obtain material, and these facilities make it easier for students to plagiarize. A recent article revealed that some students use the internet to hire expert coders to implement their programming assignments [4]. Bull *et al.* [1] and Culwin *et al*. [2] have carried out surveys on academics to determine the prevalence of plagiarism and have evaluated the performance of free-text plagiarism detection software and source-code plagiarism detection software respectively. The surveys have shown that both free-text and source-code plagiarism are significant problems in academic institutions, and the study by Bull *et al.* [1] indicated that 50% of the 293 academics that participated in their survey felt that in recent years there has been an increase in plagiarism.

A review of the current literature on source-code plagiarism in student assignments reveals that there is no commonly agreed description of what constitutes source-code plagiarism from the perspective of academics who teach programming on computer courses. Some definitions on source code plagiarism exist, but these appear to be very limited. For example, according to Faidhi and Robinson [3], plagiarism occurs when programming assignments are *"copied and transformed"* with very little effort from the students, whereas Joy and Luck define plagiarism as *"unacknowledged copying of documents and programs."*

Furthermore, a book on academic misconduct written by Decoo [5], discusses various issues surrounding academic plagiarism. Decoo briefly discusses software plagiarism and the level of user interface, content and source-code. Sutherland-Smith [6] carried out a survey to gather the views of 11 teachers in the faculty of Business and Law at South-Coast University in Australia. The findings reveal varied perceptions on plagiarism between academics teaching the same subject, and the author suggests that a *"collaborative, cross disciplinary re-thinking of plagiarism is needed."*

Computer systems for source code plagiarism detection have been in existence for over thirty years (Halstead 1977, Ottenstein 1977) and are now routinely used in many academic institutions. The first systems were based upon attribute counting algorithms, extracting various superficial metrics from source code submissions, for example a count of the use of a particular reserved word, and then flagging those pairs of submissions which were significantly close for tutors to inspect further. Later developments saw the introduction of structure metric systems where each submission is reduced to a series of identifiers, or tokens, representing, for example, a function call or a variable declaration. Pairs of tokenized submissions are then recursively searched for the longest common token sequences and the proportion of the submissions matched used as a similarity metric.

Structure metric systems have been shown to be more effective than attribute counting systems (Verco & Wise 1996) and all existing systems use those principles. The earliest attribute counting system, developed by Ottenstein (Ottenstein 1977), used Halstead's software science metrics (Halstead 1977), and primarily counted the uses of operators and operands within a piece of code. Such metrics were originally developed to allow automated marking of source code for style. Subsequent research has introduced new metrics and tried to find the most effective combinations (Robinson & Soffa 1980). Parker and Hamblen reviewed the metrics that seem to be successful for finding plagiarism in source code written in different programming languages (Parker & Hamblen 1989). Structure metric systems have developed from Donaldson, Lancaster and Sposato's early hybrid system (Donaldson et al 1981). More recent examples include Plague (Whale 1998) and YAP3 (Wise 1996) [8]. Joy and Luck showed how students could plagiarize and how they could be detected with reference to their own SHERLOCK system (Joy & Luck 1999).

Academic institutions have access to both their own locally developed systems, including TEAMHANDIN (Culwin & Naylor 1995) and Irving's Big Brother system (Irving 2000). They also have access to services provided over the Internet. Best known is Aiken's Measure of Software Similarity (MOSS) [7]. No technical details of MOSS are available, only an independent critique (Bowyer & Hall 1999) describing one approach for using the system. The JPlag [9] system, accessible on the Web and described from a technical perspective (Prechelt et al 2000 & 2001) offers similar functionality. JPlag takes as input a set of programs, compares these programs pair wise (computing for each pair a total similarity value and a set of similarity regions), and provides as output a set of HTML pages that allow for exploring and understanding the similarities found in detail. JPlag [9] works by converting each program into a stream of canonical tokens and then trying to cover one such token string by substrings taken from the other (string tiling). Aside from JPlag little research continues into source code plagiarism detection, since it is a well-understood problem involving the analysis of constrained text. Recent developments in structure metric systems employ sophisticated pattern matching algorithms, developed as part of the human genome project, to establish similarity between pairs of programs. Figure 1 is a tabular comparison on the basis of different features (Languages supported, Speed, and Accuracy etc.) of few of the existing plagiarism detection tool.

	JPlag	**Moss**	**Sherlock**	**CodeMatch**	**Copy/Paste Detector (CPD)**
System Started from	1997	1994	1994	2005	2003
Language (s) supported	Java, C#, C, C++, Scheme and natural language text.	C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS Assembly 8086, HCL2.	Java, C, C++, natural language text.	BASIC, C, C++, C#, Delphi, Flash ActionScript, Java, JavaScript, MASM, Pascal, Perl, PHP, PowerBuilder, Ruby, SQL, Verilog, VHDL.	Java, JSP, C, C++, Fortran and PHP code
Cost	Free but user must create an account	Free but user must create an account	Free and open sourced	Commercial tool, free on any code where the total of all files being examined is less than 1 megabyte	Free and open sourced
Service	Web service	Internet service	Standalone application	Standalone application	Standalone application
Interface	GUI	Web interface	GUI	GUI	GUI
Requirements	Web browser, Java Runtime Environment (JRE), Java 1.5 or higher	A submission script for either UNIX or Windows	JDK 1.4 or later		JDK 1.4 or later
Security	User id and e-mail needed	User id and e-mail needed	Runs locally	Runs locally	Runs locally
Submission Methods	Standalone Java software application that can be deployed over the network	Command line	Standalone Java application	Standalone application	Standalone application
Source Data	Files, or a directory with or without subdirectories	Files, or a directory with or without subdirectories	A directory containing subdirectories, each containing one or more zip archives or submissions	Files, or a directory with or without subdirectories	Files, or a directory with or without subdirectories
Option for excluding files	Yes	Yes	Yes	Yes	No

Speed	Fast	Fast	Large sets of files will require long amounts of time to analyze	Large sets of files will require long amounts of time to analyze	Fast
Result					
Result Output	HTML output for exploring the similar code fragments found	Both text and HTML reports	Interactive dialogues for comparing detected similarities and printing reports	HTML report, and spreadsheet showing statistical information about the files that were analyzed	Text report, xml file, cvs file.
Result storage	Local	Remote	Local	Local	Local
Overview results display method?	Histogram, statistics about the files that were analyzed	Ordered list	Matching pair tree	Ordered list	Ordered list
Display of results	Powerful graphical interface for presenting results	Powerful graphical interface for presenting results	Powerful graphical interface for presenting results	Graphical user interface presenting the results	Results displayed in a dialogue
Visualization of results	Cross linked listings, scroll bars		2/4 scroll bars, simultaneous scrolling	Table showing the similar lines of code detected among file pairs.	List of results showing the detected code fragments between file pairs.
Visual display of matched file pairs?	Yes	Yes	Yes	No	No
Display of suspicious code-fragments?	Highlights the suspicious source-code fragments	Highlights the suspicious source-code fragments	Highlights the suspicious source-code fragments	Returns suspicious lines of code, no option to view entire suspicious code fragments or the entire suspicious files	Returns duplicate lines of code
Metrics produced	Percentage similarity, token matches	Percentage similarity, token matches, lines matched	Percentage similarity, token matches	Percentage similarity, lines matched	Token matches, lines matched
Other					
Other features			Checkbox to mark the suspicious pairs		

Miscellaneous	Several detection parameter settings, including sensitivity	Self adjusting	Several detection parameter settings, including sensitivity	Some detection parameter settings	Some detection parameter settings
Algorithms	Greedy String Tiling	Winnowing algorithm	Token based matching algorithm for source-code, string matching for natural language texts	String matching	Greedy String Tiling

Figure 1: Table comparing the features offered by various tools that aid with the detection of source code plagiarism.

2. Methodology

The paper presents the design of an anti-plagiarism tool for local machine usage. Eventually the very basic sub-problem statement to be addressed is '*How to analyze the codes to be compared?*' The basic need is the correct analysis of the optimum parameters for inferring the uniqueness of the codes concerned. Each code to be analyzed is to be distinguished from the other code. If any two codes be having similar distinguishing features, they might be copied. Precisely, '*Which kind algorithm for comparison to choose and what are the critical issues of concern?'* are the next questions to be addressed. The algorithm must be optimum in both processing time and complexity. Further, the processing must be such that the false cases encountered may be minimal. These issues are discussed in the following subsections.

2.1 The Algorithm

The most important step is the selection of the algorithm for comparison of the codes. There are a lot of algorithms proposed for the purpose. A few of them are worth mentioning here.

Greedy String Tilling Algorithm **[10]**

Winnowing based approach **[11]**

Metrics based approach **[12]**

'*Which approach to choose from, for code comparison?*' is the next concern. All the above mentioned approaches, but greedy string tilling based approach, basically are either computationally expensive or are not suitable for the concerned usage. The greedy string tilling algorithm, on the other hand is preferred. String tilling is a powerful technique that is used in various domains like Document Matching and Rigorous String Operations. The algorithm works on the basis of the comparison of the tokens identified by the partial parsing of the code files. The tokens are created on the basis of the language requirements like tokens for keywords, punctuations, constants etc in the language. Comparison on the basis of such tokens is fast and accurate.

2.2 The Tokens

It is worthwhile now to discuss ‘*How the tokens would be created?*’ The tokens are created according to the language concerned. For instance, the C language tokens include keywords like ‘void’, ‘int’ etc. Separate tokens are required for these keywords individually. The punctuations like ‘#’, ‘;’ etc are also to be assigned the relevant tokens. A few examples of the formed tokens are depicted in Figure 2.

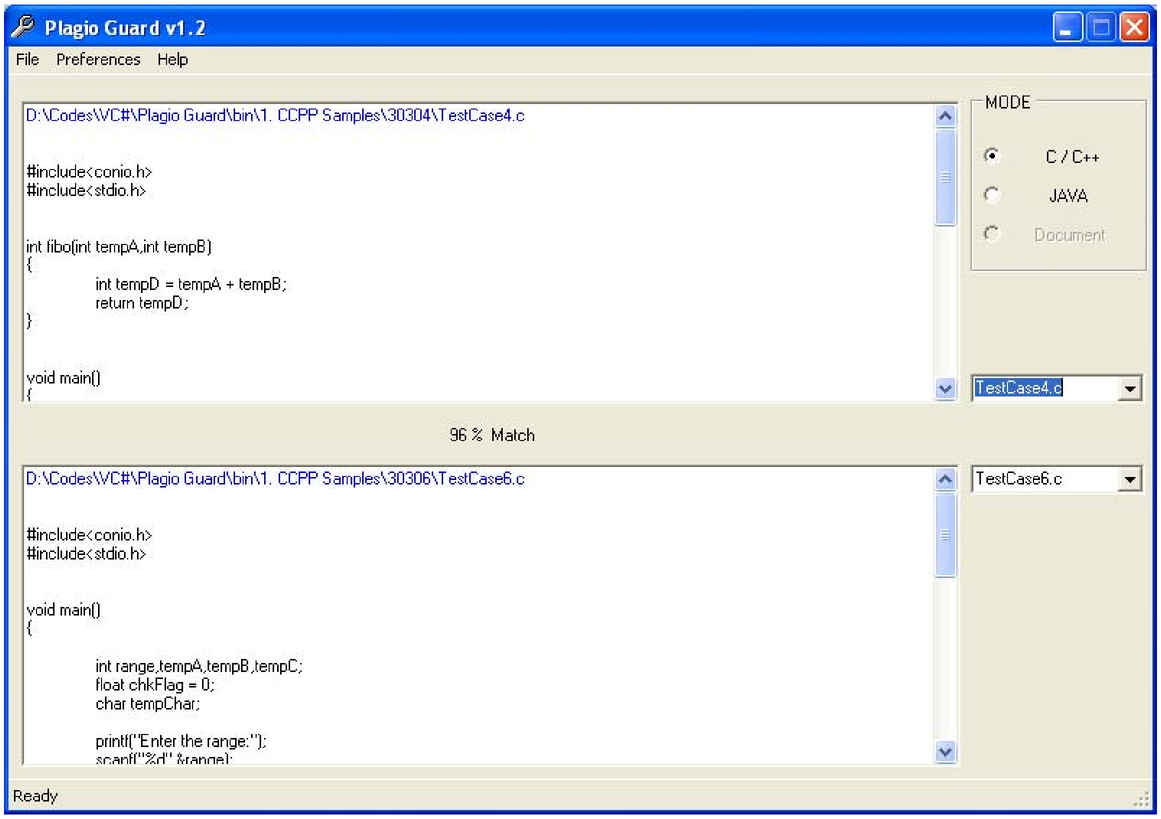
Keywords Parsing

```
       * KeywordS
       * --------
* auto            K_AUTO
* double          K_DUBL
* int              K_INT
* struct          K_STRT
* break           K_BREK
* else            K_ELSE
* long            K_LONG
* switch          K_SWCH
* case            K_CASE
* enum            K_ENUM
* register        K_REGS
* typedef         K_TDEF
* char            K_CHAR
* extern          K_EXTN
* return          K_RETN
* union           K_UNIN
* const           K_COST
* float           K_FLOT
* short           K_SHRT
* unsigned        K_USGN
* continue        K_CONT
* for              K_FOR
* signed          K_SIGN
* void            K_VOID
* default         K_DFLT
* goto            K_GOTO
* sizeof          K_SZOF
* volatile        K_VLTL
 * do              K_DO
 * if              K_IF
* static          K_STAT
* while           K_WHIL
```

Figure 2: A list of C Language Keywords with the corresponding tokens.

2.3 The Graphical User Interface (GUI) for the software

Ones the algorithm is finalized, the actual implementation of the code is to follow up next. The application to be designed must be such that it takes the input source code files from the chosen directory, analyzes them and then presents the results in an appealing way. The demonstration of the results must be such that, the user (the teachers / instructors) could visualize the results correctly with a clear picture of the situation. A typical GUI for the software is shown in the Figure 3. These screenshots are from the presented anti-plagiarism tool namely Plagio Guard, that uses similar kind of the approach for plagiarism detection.



MODE: [✓ C / C++] JAVA Document

TestCase4.c

```
D:\Codes\VC#\Plagio Guard\bin\1. CCPP Samples\30304\TestCase4.c


#include<conio.h>
#include<stdio.h>


int fibo(int tempA,int tempB)
{
        int tempD = tempA + tempB;
        return tempD;
}


void main()
{
```

96 % Match

TestCase6.c

```
D:\Codes\VC#\Plagio Guard\bin\1. CCPP Samples\30306\TestCase6.c


#include<conio.h>
#include<stdio.h>


void main()
{

        int range,tempA,tempB,tempC;
        float chkFlag = 0;
        char tempChar;

        printf("Enter the range:");
        scanf("%d",&range);
```

Ready

Figure 3: Plagio Guard: An Anti-Plagiarism Tool for Programming Languages. The upper code is the one whose replica is found and the below one is the replicated code. The label in between shows the percentage matches.

Conclusion

The application Plagio Guard clearly demonstrates the befitted applicability of the proposed solution for detection of plagiarized codes. The algorithm proposed is independent of the applicability in terms of the programming language, the data to be compared etc. The same solution approach could be applied to the code comparison for other languages like assembly (MASM assembly language) etc and also for the textual comparison for detecting essay plagiarism or in general document plagiarism.

References

[1] Bull J., et al., Technical review of plagiarism detection software report, CAA, University of Luton, (2001).

[2]Culwin F. MacLeod A. & Lancaster T., Source code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools, Technical Report No. SBU-CISM-01-02, South Bank University.

[3] Faidhi and Robinson, An empirical approach for detecting program similarity within a university programming environment, Computers and Education, vol. 11, no. 1, pp. 11-19, (1987).

[4] Gomes L., Some Students Use Net To Hire Experts to Do Their School Work. The Wall Street Journal [Online], (2006). http://online.wsj.com/public/us

[5] Decoo W., Crisis on campus: Confronting academic misconduct. MIT Press Cambridge, England (2002). ISBN:0-262-04201-0

[6] Sutherland-Smith W., Pandora's box: academic perceptions of student plagiarism in writing. Journal of English for Academic Purposes, vol.4, pp. 83-95, (2005).

[7] Alex Aiken. MOSS (Measure Of Software Similarity) plagiarism detection system. http://www.cs.berkeley.edu/˜moss/ (as of April 2000) and personal communication, 1998. University of Berkeley, CA.

[8] Michael J. Wise (1996). YAP3: Improved Detection of Similarities in Computer Program and Other Texts.

[9] Lutz Prechelt, Guido Malpohl and Michael Philippsen (2000). JPlag: Finding Plagiarisms among a Set of Programs. http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf.

[10] Michael J. Wise (1993). String Similarity via Greedy String Tiling and Running Karp−Rabin Matching.

[11] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken, "Winnowing: Local Algorithms for Document Fingerprinting", SIGMOD 2003, June 9-12, 2003, San Diego, CA.

[12] Edward L. Jones, "Metrics Based Plagiarism Monitoring", JSCS (16, 4) May 2001, Consortium for Computing in Small Colleges.