IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

Plagiarism Detection Tool for Programs

Yash Sanzgiri[1], Kevin Garda[2], Akshay Pujare[3]

Student, Dept of Information Technology, Fr Conceicao Rodrigues College of Engineering, Mumbai, India[1, 2, 3]

Abstract: Plagiarism is an act of Fraud which involves copying someone else's work and stating it as our own without giving proper credit to that person. Plagiarism was seen earlier in various fields such as literature, science etc. Nowadays Plagiarism can also be seen in Programs especially in colleges where programming assignments are performed. Plagiarism is an easy to do task, but very difficult to detect without proper tool support. This report presents an overview of the tool which would be developed for detecting plagiarism. The tool would help detect plagiarism efficiently and help the faculty grade the students effectively.

Keywords: Plagiarism, Program similarity, Karp Rabin, Tokenization.

I. INTRODUCTION

Plagiarism is an act of unethical behaviour, which involves reuse of someone else's work without explicitly acknowledging the original author. [1][6] A good example of plagiarism is source code plagiarism in programming languages courses, when a student submits a program whose part was copied from another student's program or the internet. In such a case, the plagiarized source code has been derived from another piece of source code, with a smallnumber of routine transformations. Plagiarism in university course assignments is an increasingly common problem. Several surveys showed that a high percentage of students have engaged in some form of academic dishonesty, particularly plagiarism.

Source code plagiarism is an easy to do task, usually, when students are solving the same problem by using the same programming language, the probability of their solutions looking same is high. Thus detecting Plagiarism is very tedious for the faculty members without proper tool support.

II. LITERATURE SURVEY

This section presents an overview of the existing development in plagiarism detection and also explains certain related concepts. There are three main categories of plagiarism detection approaches:[6][7] text-based, attribute oriented code-based and structure-oriented code-based. These approaches are used by source code similarity detection tools that can be divided into two categories: offline and online. Online source code similarity detection tools can check a document for fragmentsthat can be found through web search engines, while offline source code similarity detection tools check similarity between documents usually stored in a database.

A. MOSS

Moss (for a Measure of Software Similarity) [3] [4] [5] is an automatic system for determining the similarity of programs. The main application of Moss has been in detecting plagiarism in programming classes. MOSS was developed in 1994, and has been very effective in detecting similarity in programs. The algorithm behind moss is a significant improvement over other cheating detection algorithms.

MOSS is not a system to completely detect Plagiarism. It is still up to a human to go and look at the parts of the code that Moss highlights and make a decision about whether there is plagiarism or not. One way of thinking about what Moss provides is that it saves teachers and teaching staff a lot of time by pointing out the parts of programs that are worth a more detailed examination. But once someone has looked at those portions of the programs, it shouldn't matter whether the suspect code was first discovered by Moss or by a human; the case that there was plagiarism should stand on its own. In particular, it is a misuse of Moss [3] to rely solely on the similarity scores. These scores are useful for judging the relative amount of matching between different pairs of programs and for more easily seeing which pairs of programs stick out with unusual amounts of matching. But the scores are certainly not a proof of plagiarism. Someone must still look at the code. Moss is being provided as an Internet service. The service has been designed to be very easy to use–you supply a list of files to compare and Moss does the rest. The current Moss submission script is for Linux.

B. Karp Rabin Algorithm

Karp-Rabin Algorithm [1]is a string matching algorithm. It uses fingerprints to find occurrences of one string into another string. Karp-Rabin Algorithm reduces time of comparison of two sequences by assigning hash value to each string and word. Without hash value, it takes too much time for comparison like if there is a word W and input string is S then word is compared with every string and sub string in program and hence it consumes more time. Karp-Rabin [1] has introduced concept of Hash value to avoid time complexity O(m2). It assigns hash value by calculating to both word and string/substring. So hash of substring (S) matches with hash value of W then only we can say exact comparison is done.

```
functionNaiveSearch(string s[1..n], string pattern[1..m])
for i from 1 to n-m+1
for j from 1 to m
if s[i+j-1] pattern[j]
jump to next iteration of outer loop
return i
return not found
```

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

Karp-Robin Algorithm using Hashing:- [3]
functionRabinKarp(string s[1..n], string pattern[1..m])
hpattern := hash(pattern[1..m]); hs := hash(s[1..m])
for i from 1 to n-m+1
ifhs = hpattern
if s[i..i+m-1] = pattern[1..m]
return i
hs := hash(s[i+1..i+m])
return not found
has(s[1..m)
d:=1000;
p:=0;
t:=0;
h:=1;
q:=101;
for i from 0 to m
p=(d*p+ s[i..i+m-1])return p;

C. Similarity Measurement used by MOSS
MOSS uses the following method for calculating the similarity between 2 programs.It uses the percent similarity between the 2 programs based on the number of tokens.[5]

Match=(same-diff)/minle-(maxle-minle)/maxle;
Percentage match=max(0,match)*100
Case1:Nothing is copied.
maxfile=500
minle=400
same=500 tokens
diff=420 tokens
Match=(500-420)/400-(500-400)/500 =0
Percentage match=max(0,0)*100=0Case 2:Something is copied
maxfile=500
minfile=400
same=300 tokens
diff=20 tokens
match=(300-20)/400-(500-400)/500=0.5
Percentage match=max(0,0.5)*100=50

III.PROPOSED RECOMMENDATION OF THE SYSTEM

Plagiarism can be done using various techniques.In order to detect Plagiarism effectively the following 12 cases must be detected.

Program 1

```
void main() {
   sum();

}
 void sum() {
float num1=5.0;
float num2=4.0;

 float num3 = num1 + num2;
if(true)
{
for(int i=1;i<=5;i++)
{
  num3=num3+i;
}

}
print(num3);
}
```

Program 2

```
void main() {
   int num3;
float num2=4.0;
float num1=5.0;
   num3 = (int)(num2 + num1);
if(!(false))
{
int i=1;
a1:num3=num3+i;
if(i<=4)
{
i++;
goto a1;
}

}
print(num3);
}
```

Fig.1. Example of a sample program for explaining the test cases

It is typically presented in order of sophistication, least to greatest. [2] Each test case is depicted in the programs. The programs look different but are similar and copied. The following figure gives an example of 2 programs which look different but are copied.

- Comments or formatting white spaces - Programs can be made to look different just by adding some white spaces or mere comments
- Changing identifiers – The names of the variables have been changed from num3 to a1 in program 2
- Order of operands in expressions – In program 1 the sum is represented as num1 +num2 but in program 2 it is represented as num2 + num1
- Changing data types – In the above program float has been changed to int
- Replacing expressions with semantically identical equivalents -(for example, true with !false)
- Redundant statements or variables-Many a times in order to make the program look different some statements are unnecessarily repeated, which have no effect on the final output
- Order of independent statements-These include declaration statements and other such statements which are independent with respect to some statements.
- Changing the structure of iteration statements- For example in the above example for loop is relaced with a if else structure
- Changing the structure of conditional statements – At times the structure of the conditional statements can be changed without affecting the overall meaning
- Replacing procedure calls with procedure bodies- For example in the program sum function can either be called
- Introducing non-structured statements such as GOTOs- For example the program above the GOTO statement is not needed
- Combining original and copied program fragments- A common type used by plagiarists, where usually the copied code is taken from the internet and changes are made such that it resembles unique

IV. PROPOSED SYSTEM FLOW

The Project would have the following flow of execution to facilitate maximum similarity detection

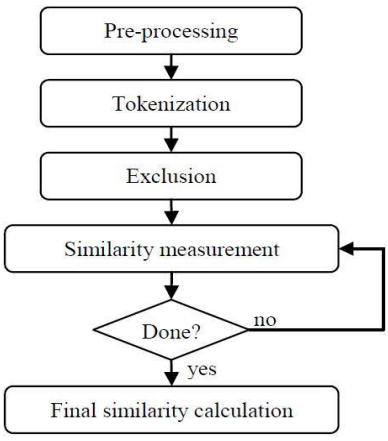

Fig2. Flow of the Project

DOI 10.17148/IJARCCE.2015.41107

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

A. Pre Processing

The first phase of source code similarity detection makes the detection process robust to the following simple source code transformations: addition, modification, or deletion of comments, changing the order of variables, as well as addition of some redundant statements. During this step, all comments are removed from the original source code... Combined variable declarations are split into a sequence of individual declarations. The fully qualified names are replaced by simple names, while package declarations and import statements are removed from the original source code. Also, in this phase, variables in statements are grouped by type. The following gives an example of the pre processing stage [7]

```
---- original source code ----
importjava.lang.*;
importjava.util.*;
/*
* block comment
*/
public class A {
/* single-line comment */
public static void main(String[] args) {
java.lang.Integer i1 = 1, i2 = 2;
java.lang.String s = "";
Long l = 5l; /* trailing comment */
java.lang.Integer i3 = i1 + i2;
// end-of-line comment
}}
---- source code after pre-processing ----
public class A {
public static void main(String[] args) {
Integer i1=1;
Integer i2=2;
String s = "";
Long l = 5l;
Integer i3 = (i1 + i2);
}}
```

B. Tokenization

Tokenization [7] is the process of converting the source code into tokens. This technique is very popular and used by many source code plagiarism detection systems. The tokens are chosen in such a way that they characterize the meaning of a program, which is difficult to change by a plagiarist. For example, blank spaces should never produce a token. A simple tokenization algorithm can substitute all identifiers and values with tokens IDENTIFIER and VALUE, respectively. Our softwares tokenization algorithm substitutes identifiers with the appropriate tokens. These tokens are chosen based on the identifier type. For example, all identifiers of numeric types, i.e. all identifiers of byte, short ,int, long, double along with their corresponding wrapper classes (Byte, Short, Integer, Long, Float and Double, respectively), are substituted with the numeric token. Also, their values are substituted with the numeric token.

Example:

```
java.lang.Integer a = 10;
java.lang.String x = "Hello";
Long l = 15l
<NUMERIC_TYPE><IDENTIFIER> = <NUMERIC_VALUE>
<STRING_TYPE><IDENTIFIER> = <STRING_VALUE>
<NUMERIC_TYPE><IDENTIFIER> = <NUMERIC_VALUE>
```

C. Exclusion

At times the source code can be shared between programmers. For example, students can use some common base source code given by their teachers. In this phase the shared code will be removed from the inputs. Some similarity detection systems, like JP lag, MOSS and Plaggie, allow the user to give such legitimately shared source code (i.e., template code) that will be ignored during the similarity detection phase .

D. Similarity Measurement

In this phase the similarity between the codes is calculated. We are using a modified version of the Karp Rabin algorithm [1] to determine the similarity.

E. Final Similarity Calculation

The fifth phase in this similarity detection process is the final similarity calculation. This calculation is based on similarity Measure values obtained from the similarity detection algorithms, and their weight factors.

V. CONCLUSION

The aim of our project is to develop a tool for checking the similarity in student' programming assignments. The tool will take the student programs as inputs from the faculty and check for similarity between them. In addition to this, the tool will also compare the programs with the internet programs. The current version of the tool is made for checking similarity between programs written in C language only. Thus the tool will also consist of some default programs which would be used for comparing with the student programs. The output of the tool will givea percent match between the programs used for testing. Accordingly, for this we studied related plagiarism tools such as MOSS. We have also done an exhaustive study of reference papers based on Plagiarism and Program similarity to identify areas of improvement. We also studied algorithms like Karp Rabin algorithm which is a string matching algorithm which would be implemented in the project

ACKNOWLEDGMENT

We have great pleasure in presenting the report on "Plagiarism Detection Tool for Programs". We take this opportunity to express my sincere thanks towards the staff of Fr C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We would specially like to thank Prof Prajakta Dhamanskar, CRCE Mumbai, for her encouragement and guidance.

REFERENCES

[1] P. S. Sonawane Kiran Shivaji, "Plagiarism detection by using karp-rabin and string matching algorithm together," International Journal

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

of Computer Applications, vol. 11 6- no 23, pp. 0975– 8887, April 2015.

[2] G. Whale., "Identification of program similarity in large populations." ." In: The ComputerJournal 33.2 (1990), pp. 140–146.

[3] A. Aiken. A system for detecting software plagiarism. [Online]. Available: https: //theory.stanford.edu/~aiken/moss/

[4] A. A. Saul Schleimer, Daniel Wilkerson, "Winnowing: Local algorithms for document fin-gerprinting."

[5] K. W. Bowyer and L. O. Hall., "Experience using moss to detect cheating on programming assignments in: Frontiers in education conference, 1999. 29th annual. vol. 3. ieee. 1999, 13b3âĂƷ18." Frontiers in Education Conference, vol. 3, 1999.

[6] Vreda, "[automated assessment of programming assignments]," 2013.

[7] D. G. Zoran Djuric, "A source code similarity system for plagiarism detection."

DOI 10.17148/IJARCCE.2015.41107