Source Code Plagiarism Detection using Multi Layered Approach for C Language Programs

Kshitiz Gupta
B.Tech Scholar
National Institute of Technology
Kurukshetra

Ekta Sardana
B.Tech Scholar
National Institute of Technology
Kurukshetra

ABSTRACT

Source code plagiarism is a growing concern in academia. Programming assignments are used to evaluate students in programming courses. Therefore, checking programming assignments for plagiarism is essential. If a course consists of a large number of students, it is impractical for a human inspector to check each assignment, and while automated tools are available, none is accurate, robust and fast enough to detect plagiarism in the programming assignments. Thus, there is a prominent need for automated and accurate plagiarism detection tool.

Keywords

Plagiarism, source code, multilayered, data slicing, AST, structure based approach, exe comparison, attribute counting.

1. INTRODUCTION

Source code plagiarism detection is extremely useful and important for both the academia and industry. Students may plagiarize by copying code from various sources. Most prominent ones are friends, web and private tutors. In programming courses students are evaluated based on their performance in programming assignments. Therefore, detection and prevention of plagiarism at universities becomes more essential and thus there is a huge demand for accurate source code plagiarism detection systems. The most challenging aspect in code-plagiarism detection is the techniques that the implicated students tend to use to disguise the copied code in order to mislead the grader. Arwin, [3], lists the most common disguises; which are

1. changing formatting,
2. changing identifiers,
3. changing the order of operands in expressions,
4. changing data types,
5. replacing expressions by equivalents,
6. adding redundant statements,
7. changing the order of time-independent statements,
8. changing the structure of iteration statements,
9. changing the structure of selection statements,
10. replacing procedure calls by the procedure body,
11. introducing no structured statements,
12. combining original and copied program fragments,
13. The translation of source code from one language to another.

2. COMPARISON

Woo and Cho [4] have mentioned two methods for plagiarism detection. 1) Attribute Counting Method. 2) Structured Based Method In the attribute counting systems programs are depicted by various quantities such as number of operands, operators, variables, blanks, loop statements, control statements and conditional statements. Then the similarity between two programs is calculated by comparing their respective values. This approach has a disadvantage that it can be either very insensitive (two programs might share the same measures while they completely differ in the logic) or very sensitive as it ignores the program's structure [2]. It fails easily with the common disguises that students might use such as blanks insertion or deletion that does not affect the structure of the program, (see [5] for details).On the other hand, the structure metric systems, that were recently used, were shown to have high performance in detecting source-code plagiarism, [5]. These systems use one of four techniques: string matching, [9], abstract syntax tree (AST), [6], program dependence graph, [2], and tokenization, [8].Presently most of the source code plagiarism detection algorithms are based on the structured method [3], [4], [2]. In addition to that there are few attempts which are based on the attribute counting method [7], [10].

Faidhi and Robinson [11] have defined six levels of source code plagiarism. Level 0 being the lowest level of plagiarism and Level 6 being the most severe one. Level 0 represents the exact copying of someone else's program whereas in Level 6 program's logical flow is modified in order to achieve the same operation. Thus, the structural features of the modified program varies more severely from the original one as we move from level 0 to level 6. Moreover, Arwin and Tahaghoghi [3] have mentioned that structural based plagiarism detection techniques rely on the belief that the similarity of programs' structures can be used to estimate whether the programs are similar. Since structured properties of plagiarized programs vary largely from the original program, it becomes highly difficult to detect plagiarism at level four or higher. On the other hand plagiarism detection systems based on the attribute counting techniques do not take structural properties of source programs into account. Although, they are unaffected by structural based problems yet it has been proved that attribute counting techniques are not accurate enough [7], [10]. Therefore, we have proposed a new system which is based on the combination of both techniques i.e. structural and attribute counting in a layered based approach. Ethem Alpapaydin [12] has pointed out that, "There is no single learning algorithm that in any domain always induces most accurate learner". Further, he has mentioned that by combining multiple algorithms in a suitable way the prediction performance and accuracy can be improved. Therefore, instead of using just one learning algorithm, we have used multiple algorithms for finding plagiarism detection.

3. MULTI LAYERED APPROACH

The source code files are passed through 4 layers for effective and efficient plagiarism detection, having progressively rigorous ways of detecting plagiarism to avoid false positive and to avoid non plagiarized files to be detected in the cycle as early as possible.

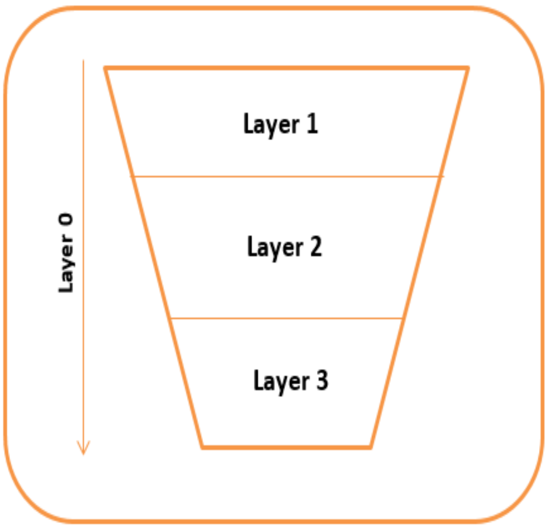


Figure 1

3.1 Layer 0

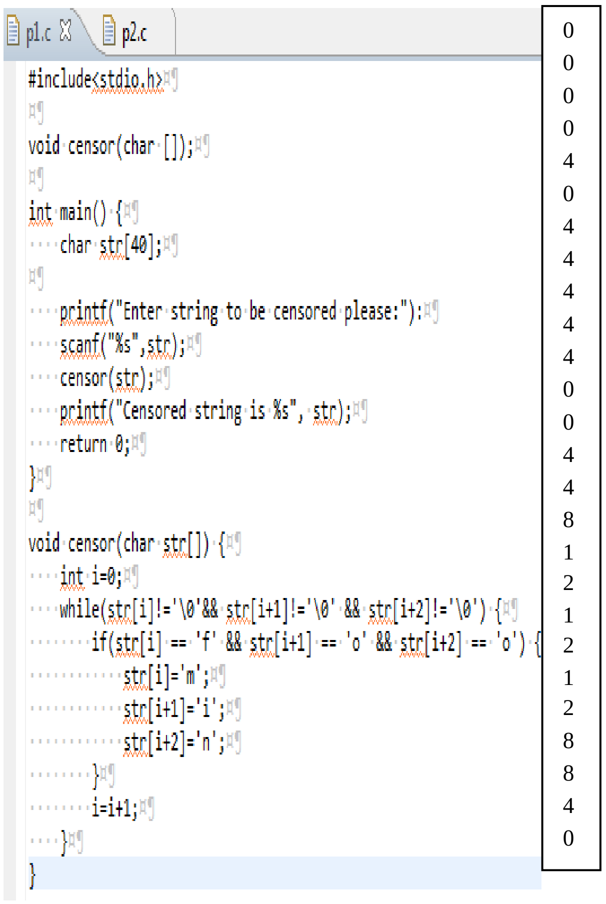
This layer is introduced to avoid unnecessary structural and attribute based comparison of obvious plagiarized source code files. In this layer, files which are exact copy paste are found through string matching algorithm and are marked as plagiarized.

3.2 Layer 1

This layer consists of 3 sub layers for indentation comparison, variable & operator count comparison and function signature comparison. There are different thresholds of amount of match for all the three. If the two programs under consideration crosses threshold of match for any of these 3 then the programs are considered to be similar and passed to layer 2 for confirmation. Note that all these layers have quite lower threshold so that plagiarized programs are not missed by these layers in most of the cases.

3.2.1 Layer 1: Indentation based Approach

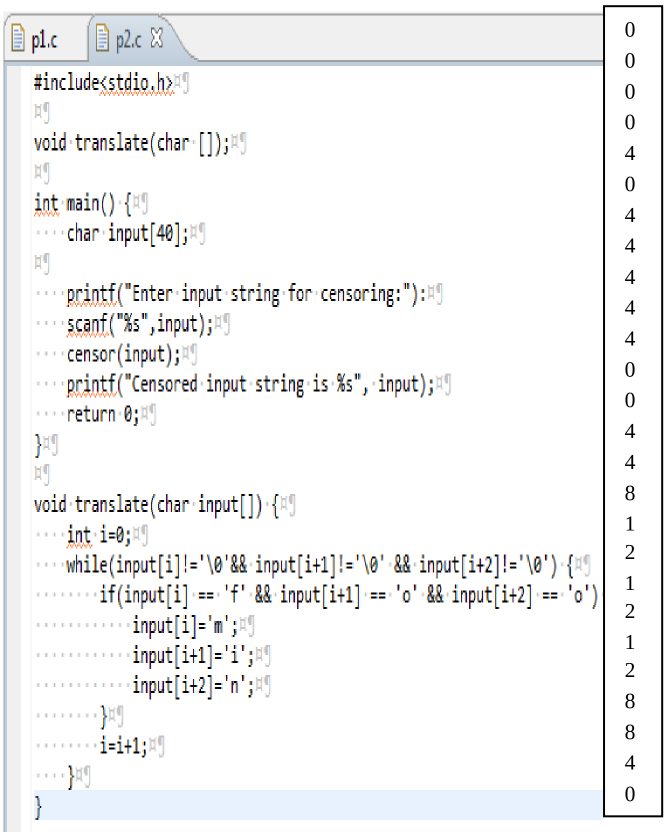
One of the significant approaches of identifying some one's coding style is to check the indentation, spacing and alignment of the program. Most of the current tools available like JPlag, MOSS etc. remove spacing and indentation while parsing the file in the first step which can otherwise be very helpful. Since it is un-common for different programmers to put similar spacing for their loops and logical conditions, checking for indentation gives us an insight if two programs are copied or not. It is a structure based approach in which first we determine the indentation of the files and then we determine the amount of indentation match between these two files using Longest Common Subsequence approach and if it exceeds a given threshold we mark the files to have plagiarism. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 80%.



```
#include<stdio.h>

void censor(char []);

int main() {
    char str[40];

    printf("Enter string to be censored please:");
    scanf("%s",str);
    censor(str);
    printf("Censored string is %s", str);
    return 0;
}

void censor(char str[]) {
    int i=0;
    while(str[i]!='\0'&& str[i+1]!='\0' && str[i+2]!='\0') {
        if(str[i] == 'f' && str[i+1] == 'o' && str[i+2] == 'o') {
            str[i]='m';
            str[i+1]='i';
            str[i+2]='n';
        }
        i=i+1;
    }
}
```

```
0
0
0
0
4
0
4
4
4
4
4
0
0
4
4
8
1
2
1
2
8
8
4
0
```

Figure 2



```
#include<stdio.h>

void translate(char []);

int main() {
    char input[40];

    printf("Enter input string for censoring:");
    scanf("%s",input);
    censor(input);
    printf("Censored input string is %s", input);
    return 0;
}

void translate(char input[]) {
    int i=0;
    while(input[i]!='\0'&& input[i+1]!='\0' && input[i+2]!='\0') {
        if(input[i] == 'f' && input[i+1] == 'o' && input[i+2] == 'o')
            input[i]='m';
            input[i+1]='i';
            input[i+2]='n';
        }
        i=i+1;
    }
}
```

```
0
0
0
0
4
0
4
4
4
4
4
0
0
4
4
8
1
2
1
2
1
2
8
8
4
0
```

Figure 3

3.2.2 Layer 1: Variable and Operator Count Approach

It is an attribute based approach. It is observed that many students who perform plagiarism tend to rename variables in program. Assuming that the number of a particular type of variable remains same, a vector comparison of two programs is performed and if the two vectors match above threshold value, the programs are marked as suspects of plagiarism. The elements that are taken into account while comparing the vectors are:

- Variables
- n-D arrays
- n-D pointers
- Operators for ex. +, -, ++

The following data types are taken into consideration while computing the number of above elements:

- Int
- Float
- Char
- Short
- Long
- Double

Two hash maps are generated containing details of variable count for specific data types and operator usage counts within the two programs. These hash map are compared with each other to determine the number of same key value pairs and if these are greater than a certain threshold, then the programs are marked as plagiarized by this layer. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 90%. Following examples show two programs, their generated vectors and the result of their comparison:

Vector Figure2	Vector Figure3
==: 3	==: 3
!=: 3	!=: 3
char: 0	char: 0
+: 6	+: 6
++: 1	++: 1
=: 5	=: 5
int : 1	int : 1
char 1D: 1	char 1D: 1

Figure 4

3.2.3 Layer 1: Function Signature Approach

It is an attribute based approach. It is usually required by the programmers to make their own functions to perform several repetitive tasks in programming. The functions contains of elements: Return Type and Parameters. These two elements constitute the signature of a function, thus comparing two function on the basis of their signatures i.e. return type, number of parameters, type of parameters we can have a fair idea if two functions are similar and hence this information can be used further to identify if the two programs are similar or plagiarized or not. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 75%.

Function Signature Figure 2	Function Signature Figure 3
F1:	F2:
Return Type: Int	Return Type: Int
Params : 0	Params : 0
F2:	F2:
Return Type: void	Return Type: void
Params, char []	Params, char []

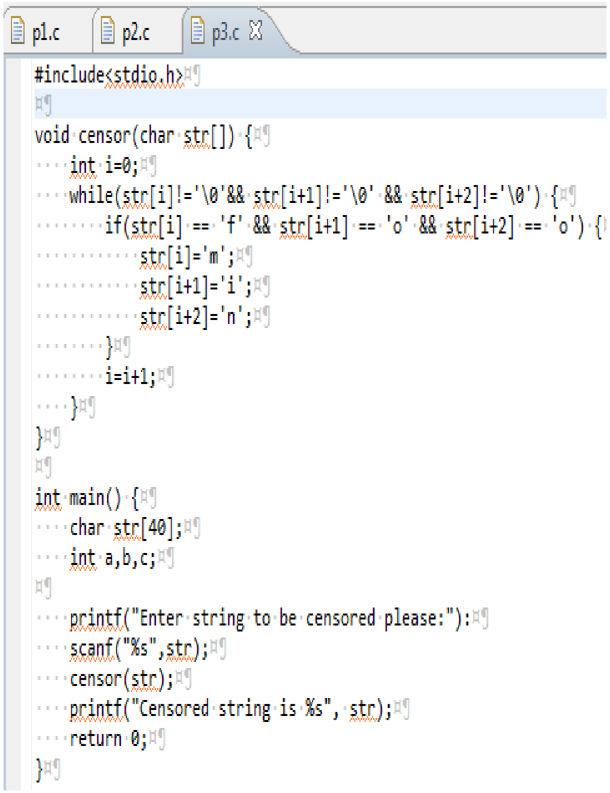
Figure 5

3.3 Layer 2

In this layer all the programs in a set of plagiarized programs formed by layer1 are compared against each other. If after comparison they are again determined as plagiarized they are passed to layer3 for further investigation else they are separated into different sets.

3.3.1 Layer 2: Exe Comparison

The exe files generated after compilation of C programs contains the memory map, address pointers and other details of the c program in binary format, and this binary format is almost same for the plagiarized files. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 85%.



```c
#include<stdio.h>

void censor(char str[]) {
    int i=0;
    while(str[i]!='\0'&& str[i+1]!='\0' && str[i+2]!='\0') {
        if(str[i] == 'f' && str[i+1] == 'o' && str[i+2] == 'o') {
            str[i]='m';
            str[i+1]='i';
            str[i+2]='n';
        }
        i=i+1;
    }
}

int main() {
    char str[40];
    int a,b,c;

    printf("Enter string to be censored please:"):
    scanf("%s",str);
    censor(str);
    printf("Censored string is %s", str);
    return 0;
}
```

Figure 6

Below is the exe difference between two plagiarized programs in figure 2 above and figure 6.

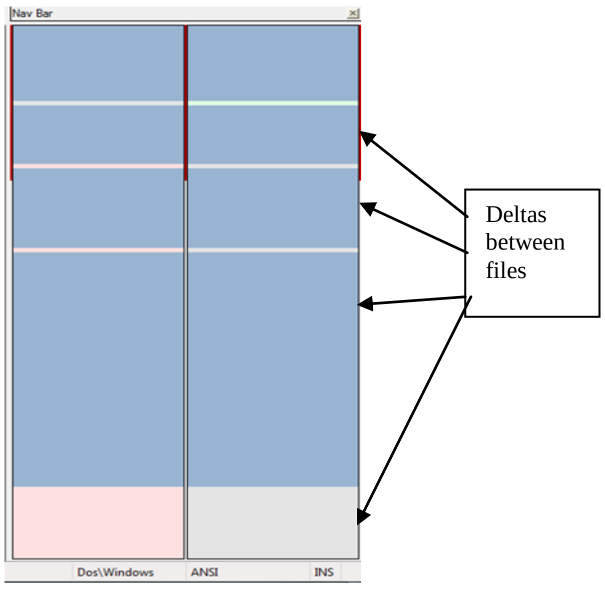
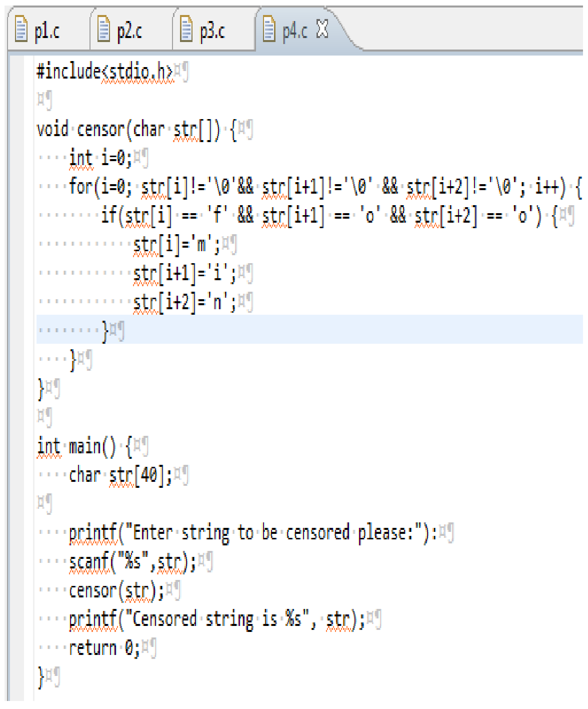


Figure 7

3.3.2 Layer 2: Keyword Sequence Comparison

Usually student's rename the variable and even introduce extra variables to avoid memory based detection, however, overall logic of the program is decided by keywords of the language and sequence of these keywords and function calls. This is a structure based approach in which we compare the relative ordering of keywords and function calls in the two programs by comparing the longest common subsequence of keywords and considers them as being plagiarized if the sequence matches above a certain threshold. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 78%.



```
#include<stdio.h>

void censor(char str[]) {
    int i=0;
    for(i=0; str[i]!='\0'&& str[i+1]!='\0' && str[i+2]!='\0'; i++) {
        if(str[i] == 'f' && str[i+1] == 'o' && str[i+2] == 'o') {
            str[i]='m';
            str[i+1]='i';
            str[i+2]='n';
        }
    }
}

int main() {
    char str[40];

    printf("Enter string to be censored please:"):
    scanf("%s",str);
    censor(str);
    printf("Censored string is %s", str);
    return 0;
}
```

Figure 8

The keywords for programs in figure 6 and 8 and longest common subsequence for them is given below.

Keywords for Figure 6

void

char

int

while

if

int

main

char

int

return

printf

scanf

censor

return

Keywords for Figure 8

void

char

int

for

if

int

main

char

return

printf

scanf

censor

return

Longest common subsequence of the 2 programs is same as keyword sequence for figure 8. And the amount of keyword matching for these 2 programs is 86%

Figure 9

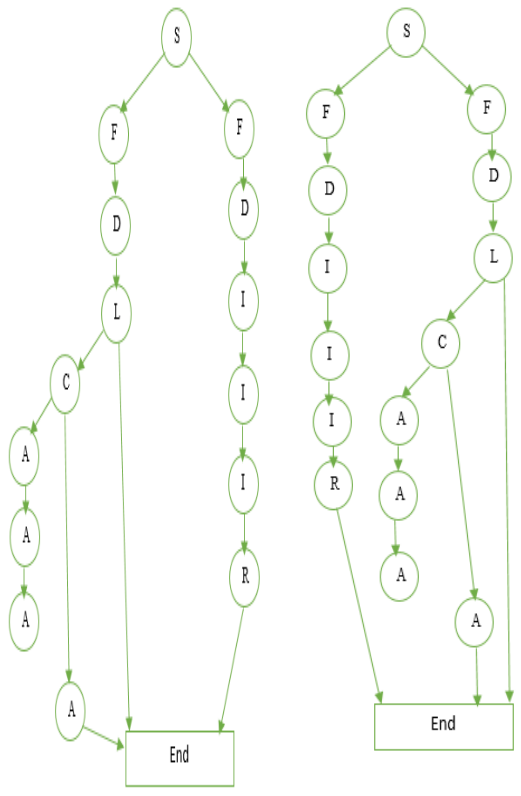
3.4 Layer 3

This is the last layer for detection of plagiarized programs and the most rigorous one. In this layer a similar procedure is followed as in layer 2 but with the most effective approaches of plagiarism detection. It comprises of advanced checks such as Data Slicing for checking the flow of prominent variables in the two programs and generating Abstract Syntax tree for their comparison.

3.4.1 Layer 3: AST Generation and Comparison

This is an structure based approach in which we generate Abstract Syntax Tree of given programs and compares pre-order traversal and in order traversal of all sub trees (representing the functions or inner details of for loop) with the other, which can detect the re-ordering or re-positioning of the code.

An abstract syntax tree is an n-ary tree representing abstract syntactic structure of a program where each node represents a statement in source code. The node is 'abstract' in the sense that it hides details that appear in the real syntax. Each node in the tree provides us with a detail of the type of statement that occurs at that position in the program flow. For ex. An if condition node in a program is represented by 'C' and a loop node is represented by 'L', then two strings representing preorder and inorder traversal of each sub tree in both programs is generated and all these set of inorder and preorder traversals are compared. Each of these sub tree traversals if matches with any of the other sub tree traversals is marked as visited so that it is compared again. If the number of sub trees out of the total number of sub trees matches above a certain threshold then these trees are considered as plagiarized. Threshold based on around 25 experiments conducted on 20 plagiarized programs is 92%. Below are the ast's generated for figure 3 and figure 8.



S - Start Node, F - Function Node, D - Declaration Node

L - Loop Node, C - Condition Node, A - Assignment Node

I - Function Invocation Node, R - Return Node

Figure 10

As can be seen apart from reorganization of sub trees the sub trees are similar and hence despite structural reorganizations, both codes can be easily determined as plagiarized.

3.4.2 Layer 3: Data Slicing

This is an attribute based approach, in which both programs are parsed to determine all the variables and their usage in the whole program. The use of these variables includes many type of expression like. Assignment ('A'), Increment ('I'), Binary Left Shift (L). Then 'X' most frequently used variables are selected from both programs and their usage string are compared through LCS (Longest Common Subsequence) technique. If the LCS length matches above a certain threshold for all 'X' variables, then those are considered as plagiarized. In case of variable with same name getting used in global scope and local scope, these two are considered different variables and updated accordingly. The following method was used for calculating X:

$X = 3$ for $v > 10$,

$X = v/2$ for $v > 2$ and $v <= 10$

$X = v$ for $v <= 2$

Where v belongs to Z+

Threshold based on around 25 experiments conducted on 20 plagiarized programs is 92%. Data Slicing based detection for figure 6 and 8 above is shown below.

Figure 6 prominent variable usage	Figure 8 prominent variable usage
Censor str : CCCCCCAAA	Censor str : CCCCCCAAA
i : DUJUJUUJUJUUJUJUJU	i : DAUJUJUUJUJUUJUJUJUI
str: DUUU	str: DUUU

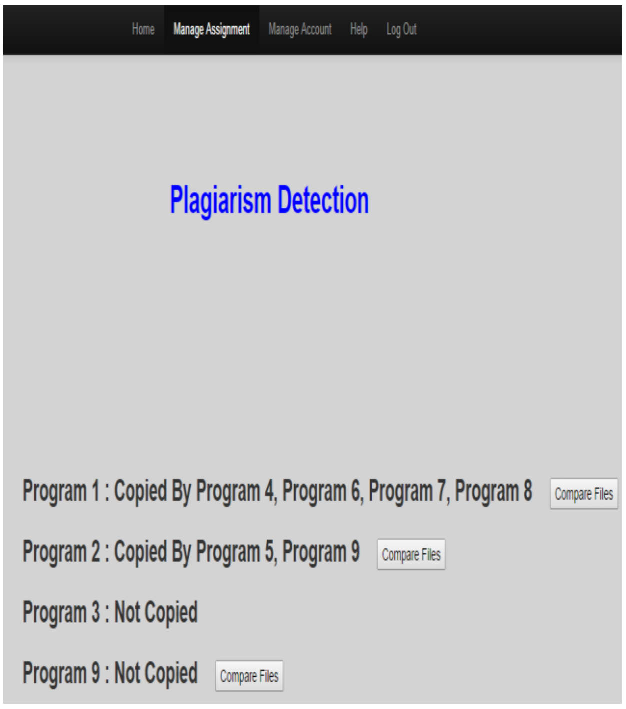
C – Condition, A – Assignment, D - Declaration

U - Usage, J – Addition, I - Increment

Figure 11

4. JUXTAPOSE PLAGIARIZED FILES

Figure 13 shows the group of plagiarized files. To see the differences and compare two plagiarized files, click on "compare files" button. For details refer figure14



Plagiarism Detection

Program 1 : Copied By Program 4, Program 6, Program 7, Program 8

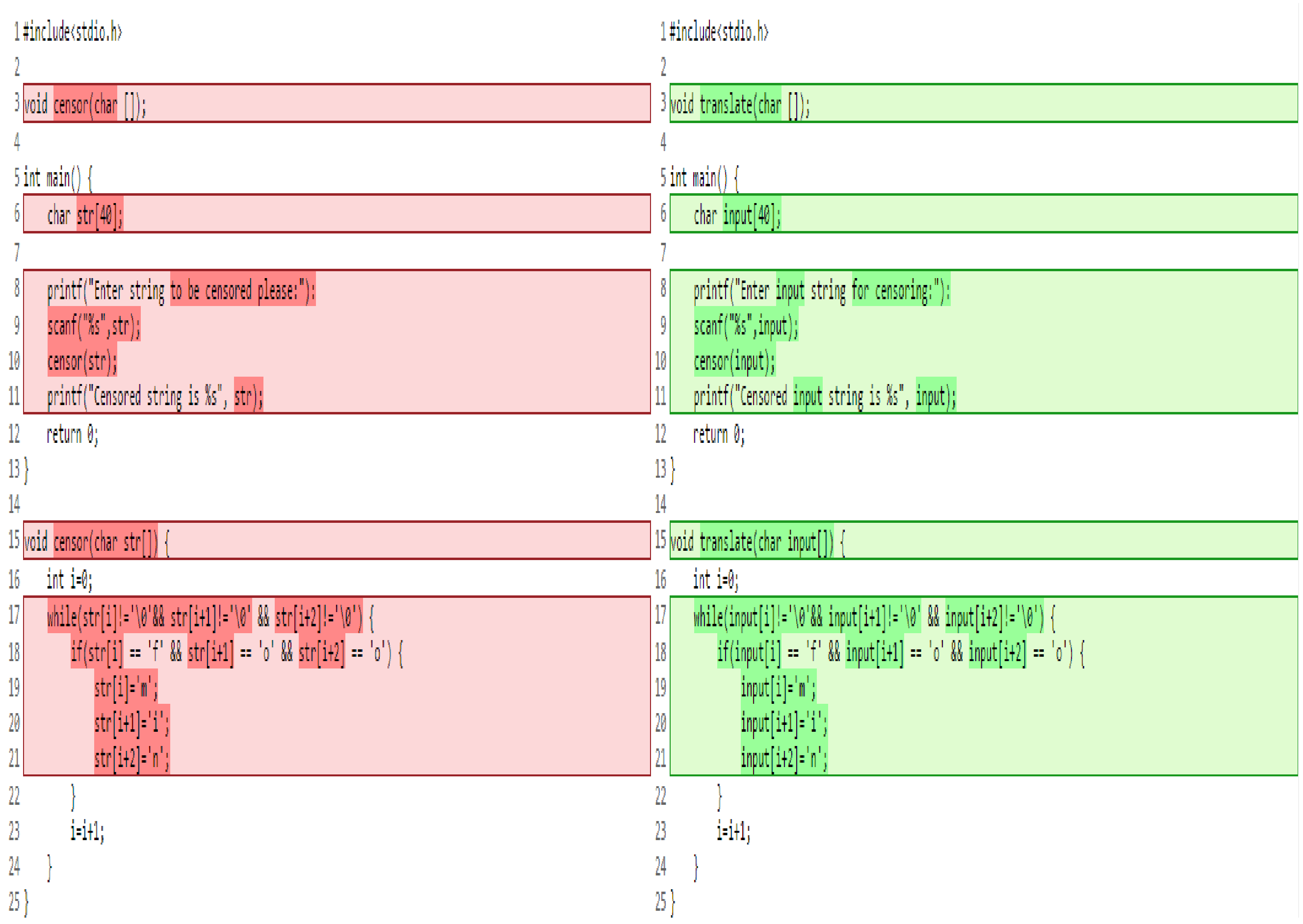
Program 2 : Copied By Program 5, Program 9

Program 3 : Not Copied

Program 9 : Not Copied

Figure 12

To compare these plagiarized files manually, those files can be viewed in diff mode too.



```
#include<stdio.h>

void censor(char []);

int main() {
   char str[40];

   printf("Enter string to be censored please:");
   scanf("%s",str);
   censor(str);
   printf("Censored string is %s", str);
   return 0;
}

void censor(char str[]) {
   int i=0;
   while(str[i]!='\0'&& str[i+1]!='\0' && str[i+2]!='\0') {
      if(str[i] == 'f' && str[i+1] == 'o' && str[i+2] == 'o') {
         str[i]='m';
         str[i+1]='i';
         str[i+2]='n';
      }
      i=i+1;
   }
}
```

```
#include<stdio.h>

void translate(char []);

int main() {
   char input[40];

   printf("Enter input string for censoring:");
   scanf("%s",input);
   censor(input);
   printf("Censored input string is %s", input);
   return 0;
}

void translate(char input[]) {
   int i=0;
   while(input[i]!='\0'&& input[i+1]!='\0' && input[i+2]!='\0') {
      if(input[i] == 'f' && input[i+1] == 'o' && input[i+2] == 'o') {
         input[i]='m';
         input[i+1]='i';
         input[i+2]='n';
      }
      i=i+1;
   }
}
```

Fig 13: Diff between two plagiarized files

5. CONCLUSION AND FUTURE WORK

Plagiarism in source code submissions is a serious problem that has motivated researchers to find effective automated detectors. This paper proposed a layered based approach which inculcates the advantages of both structure based techniques and attribute counting techniques. The layered architecture helps in detecting non plagiarized files quickly in earlier stages. Thus providing a more efficient and accurate solution. This approach has been currently applied and verified in detecting plagiarism in C program files.

This approach can also be used for checking plagiarism amongst programs of different languages such as C++, Java in the future by adding their respective grammars. Currently our application's grammar takes into consideration various constructs and tokens from C language which can be extended to include grammatical constructs from other languages as mentioned above to detect plagiarism in those languages.

6. ACKNOWLEDGMENTS

Our thanks to the Dr. J.K. Chhabra, Associate Professor, Head Of Department, National Institute of Technology, Kurukshetra for guiding us throughout the project, motivating us and teaching us about the relevant methodologies and best practices.

7. REFERENCES

[1] J. Zobel, "Uni Cheats Racket: A case study in plagiarism investigation," Proceedings of the Sixth Conference on Australasian Computing Education, vol. 30, 2004, pp. 357–365. Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 881. Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[3] Christian Arwin and S.M.M. Tahaghoghi. Plagiarism detection across programming languages. Proceedings of the 29th Australasian Computer Science Conference, 48:277–286, 2006.

[4] J.H. Ji, G. Woo, and H.G. Cho, "A source code linearization technique for detecting plagiarized programs," ACM SIGCSE Bulletin, vol. 39, 2007, p. 77. Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[5] Kristina L. Verco and Michael J. Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. Proceedings of the First Australian Conference on Computer Science Education, pages 81–88, 1996.

[6] Young-Chul Kim, Yong-Yoon Cho, and Jong-Bae Moon. A plagiarism detection system using a syntax- tree. International Conference on Computational Intelligence 1:23–26, 2004

[7] S. Engels, V. Lakshmanan, and M. Craig, "Plagiarism detection using feature-based neural networks,"

Proceedings of the 38th SIGCSE technical symposium on Computer science education, 2007, p. 38.

[8] Michael Philippsen Lutz Prechelt, Guido Malpohl. Finding plagiarism among a set of programs with JPlag. Journal of Universal Computer Science, 8(11):1016–1038, 2002.

[9] Lefteris Moussiades and Athena Vakali. PDetect: A clustering approach for detecting plagiarism in source code datasets. The Computer Journal, 48(6):651–661, 2005.

[10] R.C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software forensics," Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, p. 2089.

[11] J.A.W. Faidhi and S.K. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," Computers & Education, vol. 11, 1987, pp. 11–19.

[12] E. Alpaydin, Introduction to Machine Learning, Second Edition, The MIT Press, 2010.