Overview of Source Code Plagiarism in Programming Courses

Deniz Kılınç, Fatma Bozyiğit, Alp Kut, Muhammet Kaya

Abstract*: *Plagiarism of programming source codes is an undesirable situation in the many fields of software development world. Especially in educational field, it is obviously realized that plagiarism in programming courses increases consistently. The aim of this study is attempting to answer questions such as "which codes are similar?", "what similarity ratios are?" in order to prevent plagiarism among university students who attend programming courses. While developing the proposed methodology, N-gram similarity calculation method and Vector Space Model (VSM) were considered. Information Retrieval (IR) System and Cosine Normalization (CN) methods were utilized to calculate similarity ratios. Experimental study was performed on the dataset yielded by changing source code examples in different forms. The results obtained provide convincing evidence that the study is fit the purpose.

Index Terms:* *Plagiarism source code, n-gram, vector space model, cosine normalization.

I. INTRODUCTION

Plagiarized code is a source code example which its source cannot be understood in detail most of the times [1]. If a license of software allows using entire or some parts of source code, there is no problem while citing and using it. However, if citing a source code or appropriating is not allowed, this is out of line in terms of ethics. This issue is legally remarked in Intellectual and Artistic Works and while computer programs are included in the scope of works of science, the owner of source code is discussed as an author [2]. Plagiarism of source code is an important problem that can be faced every time, in everywhere. For example, using a source code of a program without permission which is developed specifically for a company is a common plagiarism situation. Another example can be seen in education area. Especially, programming course instructors indicate that source code theft issues pose a major problem while evaluating students' projects and home-works.

Based on continuing development of technology, applications in field of software increase correspondingly and plagiarism stands out as a big problem. There are many methods to understand whether code is stolen or not and how to prevent code theft. One of these methods is evaluating a software tool which finds similarity ratios among source codes. Already developed tools are available and have been using in many fields such as education. For example, Plague [3], JPlag [4] and YAP [5] applications are well-known tools. Many instructors in universities use these applications in order to check whether the assignments in programming courses are copy or not.

The necessary steps to solve plagiarism problems on source codes are much harder than natural language processing (NLP)[6]. The traditional method is extracting source code metrics before similarity check. However, there are some disadvantages in this traditional approach. For example, software metrics are programming language dependent. Metrics which are created to specify characteristics of Java programming language may not be appropriate for C or Pascal. Another difference is that the metric selection is not a trivial process and usually involves setting thresholds in order to eliminate metrics which aren't correlated to the classification model.

The aim of this study is to find the similarity ratios among source codes belonging to a programming course and attempt to decide whether or not two or more programs are plagiarized. To carry out our study, N-gram algorithm, Vector Space Model (VSM) [7] and Information Retrieval (IR) [8] system are utilized. Since N-gram algorithm is language independent and does not contain disadvantages of traditional methods, it has been selected in this study.

Firstly, optimal N value for an application is specified and documents are divided into N-grams. In this study, bi-gram and tri-gram methods are performed on datasets. After obtaining N-grams, a VSM is constructed where each document is represented as a vector. In VSM, a vector includes each N-gram frequencies of a document in data set. While calculating the weights of N-gram, IR approach is utilized. After counting the values of Term Frequency (TF) and Inverse Document Frequency (IDF), their values are placed into VSM. Finally, the similarity scores between documents are obtained by using Cosine Normalization (CN) [9] method.

The rest of paper is organized as follows: Section 2 detail plagiarism and how students create copy codes from the original one. Section 3 introduces the background of study. It includes related works and general information about N-gram algorithm, VSM. In Section 3, the details of study are explained. In Section 4, the datasets that have created for the application and experimental results are touched on. Section 5 concludes the paper and gives some information about future works.

II. PLAGIARISED SOURCE CODE SAMPLES/PARADIGMS

Plagiarized code is a modified and concerted version of original code which is taken without permission of code owner. Especially, in education area, some students, who attend

Manuscript Received on April 2015.

Deniz Kılınç, Celal Bayar University, Department of Software Engineering, Turkey.
Fatma Bozyiğit, Celal Bayar University, Department of Software Engineering, Turkey.
Alp Kut, Dokuz Eylul University, Department of Computer Engineering, Turkey.
Muhammet Kaya, Celal Bayar University, Department of Software Engineering, Turkey.

programming courses, copy all or part of a program from other students and submit the copy as their own work. When the copy code is inspected in detail, it can be obviously seen that most of students only change the specific points of program such as renaming variable names, adding comment lines, replacing code blocks etc.., while creating copy codes. Considering the situations that are mentioned above, the experimental dataset is generated in order to test the proposed study. This dataset includes different instance of programs which are obtained by changing a code sample at specific points. At this stage of the study, it is performed a work with a team consisting of five instructors. It is determined which parts of source code are generally modified by the students while creating copy code. Instructors highlight that they realize the most noticeable thing is renaming variables, functions and parameters when they inspect copy codes. In this part, renaming is the most efficient step while generating dataset that includes modified code examples to test study. For example, in Figure 1, both of the program codes calculate the factorial of a number. While left one is original code, the code in the right side is copy. At first glance, distinction in identifiers can be realized easily. It is clearly seen that the name of parameter "input" is changed as "inpt" and parameter "result" is changed as "rslt" while producing a new code from the original one. Also, when looking at similarities among source codes, regulating comment lines like removing, translating into another language etc… is efficient for decreasing similarity ratio. In Figure 1, it is also seen that the comment line which gives information about "factorial" method is removed at the right code.

```
class Calculation
{
//Recursive   function   which   continuously
called itself to calculate factorial

    int factorial(int input)
    {
        int result;

       if(input==0)
         return 1;

       result = factorial(n-1) * n;
       return result;
    }
}

public class Factorial
{
     public static void main(String args[])
     {
       Calculation     object_c     =     new
Calculation();

       int input = object_c.fact(4);

       System.out.println("The  factorial  of
the number is : " + input);
     }
}
```

Original Code

```
class Calculate
{
    int factor(int inpt)
    {
       int rslt;

       if(inpt==0)

           return 1;

       rslt = factorial(inpt-1)*inpt ;

       return rslt;
    }
}

public class Factor
{
     public static void main(String args[])
     {
       Calculate object_c = new Calculate();

       int inpt = object_c.fact(4);

       System.out.println("The   result:   "   +
inpt);
     }
}
```

Copy Code

Figure 1 Original and modified copy code comparison

As a result of inspections, the following steps are selected which are useful to generate copy source codes. These steps are;

1) Renaming identifiers,
2) Adding or removing blank lines,
3) Modifying the comment lines,
4) Changing parameter order in functions/methods,
5) Removal of functions/methods,
6) Adding or removing operator space.

III. BACKGROUND

A. Related Works

Joy and Luck (1999) dwell on plagiarism in assignments of programming courses [10]. They explain plagiarism as out of favor making copy of documents or source codes. In the study, it is claimed that if students in the programming course are in high number, detecting and controlling copies in assignments can be difficult. Also similarities among programs don't refer to plagiarism all the time. In the study, it is inspected that how it can be decided about the code is copy or not. Source codes are divided into tokens that take value as name, operator, loop etc.. After filtering out unnecessary information, incremental comparison step is completed and similarity ratios among codes are obtained. In incremental comparison step, pair of programs is compared five times; in their original form, with the white spaces removed, with all comments removed etc.. So they provide obtaining more consistent similarity results.

Jones (2001) indicates that plagiarism is an ethical problem can be faced always in the academic area [11]. It is also mentioned that trying to detect copy codes in programming courses is so difficult for educators in terms of presenting proofs about copies, wasting time and emotional burden because of charging a student as cheater. Jones develop an application to give evidences of plagiarism to students. So, objection of students and arguing between instructors and students can be terminated owing to results of application. In metric based system, physical profiles that include general

parameters such as number of lines, words and characters are created at first. Then, Helstead profiles that divide source code into tokens and store the frequencies of tokens, is evaluated. Lastly, two profiles are combined and distances of patterns of profiles from the each other computed.

Culwin and Thomas (2001) mention about plagiarism problem that increases in academic institutions [12]. They elaborate why students steal the information and show it as their own while they do assignments. They perform a study to dissolve plagiarism. So, their study helps instructors to understand which assignment is copy. The study consist of four stage; collection, detection, confirmation and investigation. At collection stage, students use web form submissions, so collection of assignment is completed via Web. After collecting the data, detection stage is started and similarity ratios among documents are obtained. At confirmation step, instructors should check whether the similarity results are consistent. Because, two students whom similarity ratio is high, may use same web site while doing assignments, so they cannot be charged with plagiarism. The process is terminated with investigation step and students who will be punished are determined. It is briefly pointed that revealing proof of copy is so necessary to eliminate plagiarism in the study.

Frantzeskou (2007) indicates that to solve authorship disputes in software area, not only finding similarities among programs is enough, but also identifying source code authors is necessary [13]. So, she developed SCAP Method to specify owner of source code. The author underscores that SCAP method is effective on all programming language. Also, it is claimed that SCAP Method can work with simple profile examples that include a few code lines and a few examples of profile is enough to get good results. In SCAP, after finding N-gram frequencies, Simplified Profile Intersection (SPI) value is counted. Value of SPI measures the intersection of source code documents and gives a similarity ratio.

B. N-gram Algorithm

N-gram algorithm obtains a substring combination and finds repeat ratios of this substring in a character array which will be compared with other strings to find similarity [14]. Besides using in fields of natural language processing, owing to technological development, N-gram algorithms have started to be used in programming languages. The algorithm inspects documents to categorize and to find similarities. N-gram algorithm is recognized as one of the simplest and best efficient method that finds similarity among strings.

An N-gram algorithm starts to work with dividing a text into substrings has length of N that is specified by the user. When reached to N-1th element of string, process is terminated. If value of N is one, it is called uni-gram. If value of N is two, it called bi-gram. If value of N is three, it is called tri-gram. For example, to explain tri-gram, the results in Table 1 can be shown.

N-Gram	Frequencies
STR	1
TRI	1
RIN	1
ING	1

Table 1. Tri-grams in STRING word and frequencies of substrings

As shown in Table 1, at fourth character of STRING, grouping process is terminated so, N-gram algorithm is completed. Substrings in specified text and frequencies of them are held to compare two or more documents.

At indexing step, the documents are partitioned into N-grams, and then each N-grams word is added to lists correspondingly. Figure 1 explains the indexing step briefly. At search step, the query is also partitioned into N-grams, and for each of them corresponding lists are scanned using the metric.

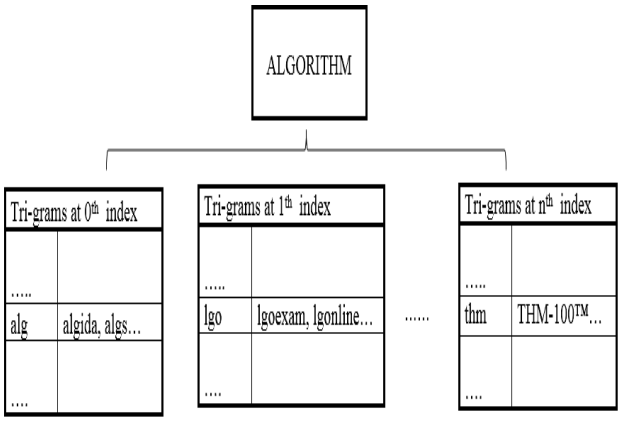

Figure 2 Tri-grams in "ALGORITHM"

Briefly, in this algorithm, N-gram frequencies of two documents are compared and distances between them are measured. The distance variable takes value between 0 and 1. While the value of distance closes to 1, it is deduced that similarity ratio increases. Otherwise, this ratio decreases

In this study, the reason of choosing N-gram algorithm is providing language independent structure and obtaining accurate results while finding similarities among source code documents. Value of N is specified as three and tri-grams in each source codes are compared separately to obtain similarity ratios.

C. Vector Space Model

The representation of a set of documents as vectors in a common vector space is known as the vector space model and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering [15]. In this model, each dimension shows a separate term. All terms in vector have a weight that is represented as "*w*". According to query result, if a document contains a term, value of weight is counted and takes a value different from 0. In Figure 3, x and y show two documents that will be compared and w_1, w_2 and w_3 indicate the weight of terms in documents.

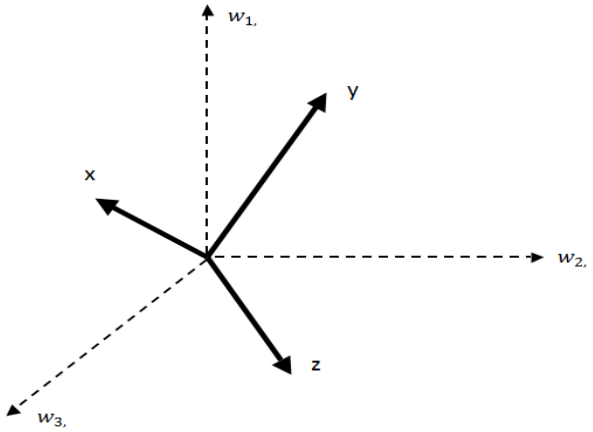

Figure 3 Representation documents in VSM

Weights of terms can be calculated by $TF \times IDF$ method. Term Frequency (TF) represents frequency of a term in the document. Inverse Document Frequency (IDF) gives information about the number of times that term occurs in all documents of collection. Equation of (TF) and (IDF) are shown in Equation (1) and Equation (2).

$$TF(t) = \frac{Number\ \ of\ time\ s\ term\ \ t\ appears\ \ in\ a\ document}{\text{Total number of terms in the document}}. \quad (1)$$

$$IDF(t) = \log_e \frac{Total\ \ Number\ \ of\ documents}{Number\ \ of\ documents\ \ wit\ h\ term\ \ t\ in\ it} \quad (2)$$

After finding the weights of all terms in vector, some vector operations are used to compare documents to specify how they are similar. Generally, the cosine of the angle between documents is calculated. This method is called Cosine Normalization. Equation (3) shows formula of Cosine Normalization.

$$sim_{d1,d2} = \cos Q = \frac{d1\ d2}{IId1II\ IId2II} \quad (3)$$

In this study, tri-grams in source code document are taken as terms and placed into VSM. Weights of tri-grams in source code documents are calculated by $TF \times IDF$ and values of terms are set. After weighting process, the cosine normalization is calculated and similarity ratios of source code documents are obtained.

IV. EXPERIMENTAL STUDY

A. Dataset

In this part of the study, 63 different code examples are generated by modifying the original code example as reported in the third section of paper. The documents which are created to test the study and to view similarity scores are called according to their alteration style. Table 2 shows all acronyms of documents names and explanations.

Table 2. Information of documents in dataset

Document Name	Explanation
D_{BL}	Adding/Removing Blank Line
D_{RI}	Renaming Identifiers
D_{PO}	Changing Parameter Order
D_{OS}	Adding/Removing Operator Space
D_{RF}	Relocation of Functions
D_{MC}	Modifying Comment Lines
D_{C1}	Adding/Removing Blank Line + Renaming Identifiers
D_{C2}	Adding/Removing Blank Line + Changing Parameter Order
D_{C3}	Adding/Removing Blank Line + Adding/Removing Operator Space
D_{C4}	Adding/Removing Blank Line + Relocation of Functions
D_{C5}	Adding/Removing Blank Line + Modifying Comment Lines
D_{C6}	Adding/Removing Blank Line + Renaming Identifiers + Changing Parameter Order
D_{C7}	Adding/Removing Blank Line Renaming Identifiers + Adding/Removing Operator Space
D_{C8}	Adding/Removing Blank Line + Renaming Identifiers + Relocation of Functions
D_{C9}	Adding/Removing Blank Line + Renaming Identifiers + Modifying Comment Lines
…	
…	
D_{C57}	Adding/Removing Blank Line + Renaming + Parameter Order + Operator Space + Functions + Comment Lines

In Table 2, first six documents are obtained by adding or removing blank line, changing parameter orders in methods, adding or removing spaces between operators, relocation of functions and modifying comments. Other 57 ones are generated by combining of the six alteration steps. According to the Table 2, D_{BL} represents the documents which are formed with additional blank lines among original source code lines.

D_{RI} demonstrates the code obtained from original source code by changing identifiers' names. D_{PO} shows the a code document yielded by changing the location of the parameters of methods in original code. For example, let's look at the Figure2 includes a method that finds the minimum between two numbers. While "minFunction" method in original code includes parameters "param1" and "param2" sequentially, the copy code this parameter order relocated.

```
public static int minFunction(int param1, int param2)
{
int minimum;
if (param1> param2)
min = param2;
else
min = param1;

return min;
}
```

Original Code

```
public static int minFunction(int param2, int param1)
{
int minimum;
if (param2 > param1)
min = param1;
else
min = param2;

return min;
}
```

Copy Code

Figure 4 An example of changing parameter order in methods

D_{OS} implies spacing out before or after operators. For example, it can be easily realized in Figure 3. While code block at the left side of Figure don't include any space before or after operator, at the left it can be easily recognized that there are spaces between operators.

```
i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Original Code

```
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Copy Code

Figure 5 An example of adding operator space

D_{RF} indicates changing of place of functions or methods in code lines. For example if a function starts at 57th code lines in original code, the student who attempt to plagiarize can move it 3rd code lines to show code as different. In the opinion of instructors, this is one of the most common methods among students while attempting to copy a code. D_{MC} is obtained by modifying comment lines as contraction or rewriting in different language.

B. Experimental Results

After creating dataset as mentioned in the previous section, Tri-gram and VSM Tri-gram similarity ratios are obtained and the results are showed in Table 3. The results of similarity ratios are between 0 and 1. When this value approximate to 1 from 0, it can be understood that the similarity is higher between two compared documents.

Document(Original)	Tri-Gram Result	VSM Tri-Gram Result
D_O	1	1
D_{BL}	0.96	0.98
D_{RI}	0.86	0.88
D_{CP}	0.94	0.97
D_{OS}	0.81	0.96
D_{RF}	0.93	0.94
D_{MC}	0.95	0.92
D_{C1}	0.82	0.90
D_{C2}	0.94	0.93
D_{C3}	0.80	0.94
D_{C4}	0.90	0.92
D_{C5}	0.94	0.91
D_{C6}	0.76	0.84
D_{C7}	0.72	0.83
D_{C8}	0.74	0.81
D_{C9}	0.78	0.80
D_{C10}	0.69	0.77
…	…	…
…	…	…
D_{C6}	0.69	0.66

Table 3. Compare Results of Tri-grams

After the codes in datasets are inspected by the instructors it is obviously seen that VSM results are more consistent. For example, according to the instructors, adding space before or after operators is not effective while attempting to change source code. However, when it is looked at the Table 2, it can be realized that the similarity ratio between D_{OS} and original document is low significantly beside VSM Tri-gram result. Even the ratio of D_{OS} is smaller than D_{C1} which consist of adding/removing blank line and renaming identifiers. The other point the instructors especially indicate that modifying comment is more effective than adding or removing blank lines among the code lines. However in the tri-gram results, D_{MC} and D_{BL} has nearly same similarity score when they compare to original code and it is obviously seen that the difference in similarity values of D_{BL} and D_{MC} are more coherent in VSM Tri-Gram.

As mentioned in Section 2, instructors claim that students generally choose only one alteration type while trying to change source code. Commonly used alteration types are leaving/removing blank lines among code lines, changing identifier names, adding/removing comment lines and replacing of code blocks of methods. In accordance with the experimental test results, doing one of these changing is not efficient while decreasing similarity ratio. If student combines all of six steps that mentioned in Section 2, the similarity among original code and copy code decreases significantly. However, this is difficult as creating new code, so instructors claim that students don't exert effort and waste their time to combine more than three steps.

In Figure 3, an application GUI of the study is shown that give information about bigram, trigram and VSM Tri-gram similarity scores between D_{BL} and other 62 documents.

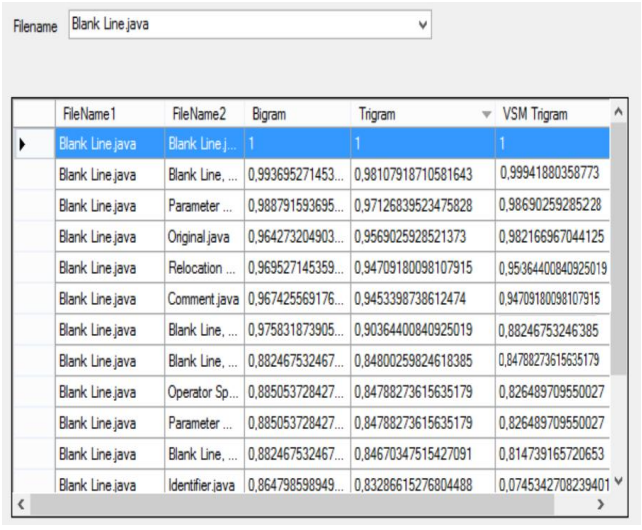

Figure 6 Interface of comparing results between D_{BL} and other documents

V. CONCLUSION

Plagiarism in programming courses is a growing problem in education. The aim of this study is attempting to find similar source codes. Although N-gram analysis is a well-known technique in NLP, it has been utilizing in source code analysis for a while. In this study, the reason of selecting N-gram method is providing language independency. In our previous study, we utilized only bi-gram and tri-gram methods to check the source codes are copy or not [14]. In this study, additional to n-grams, VSM is constructed and weights of tri-grams of all documents are placed into document matrices separately. Then, CSM scores are obtained between matrices in VSM. When acquired tri-gram and VSM tri-gram results are compared, it is determined by instructors that VSM tri-gram results give more accurate outcomes.

In future, we would like to integrate Word Net to our proposed method that provides finding similarities among source code. Another future direction of proposed study is generating a system that enables to build up greater datasets to test the study.

REFERENCES

1. A. Parker and J. Hamblen, “Computer algorithms for plagiarism detection”, IEEE Trans. Education, vol. 32, May 1989, pp. 94–99.
2. Intellectual and Artistic Works Law. Available: http://mevzuat.meb.gov.tr/html/7981_5846.html
3. G. Whale, “Plague: Plagiarism Detection Using Program Structure”, Dept. Comput. Sci., Univ. New South Wales, Kensington, Australia, Tech. Rep. 8805,1988.
4. G. Malpohl. JPlag: Detecting Software Plagiarism. Available: http://www.ipd.uka.de:2222/index.html
5. “YAP3: Improved detection of similarities in computer program and othertexts,” In Proc. 27th SCGCSE Tech. Symp., Philadelphia, PA, 1996, pp.130–134
6. R. Dale, H. Mois, H. Somers, “Handbook of NLP”, Marcel Dekker, 2000.
7. G. Salton, “The SMART Retrieval System – Experiments in Automatic Document Processing”, NJ, Englewood Cliffs: Prentice-Hall, 1971.
8. G. Salton, A. Wong, C.S. Yang. “A vector space model for information retrieval”, Journal of the American Society for Information Science, 1975 , 18(11):613-620.
9. C.D. Manning, P. Raghavan., H. Schütze, “An Introduction to Information Retrieval”, Cambridge University Press, 2009.
10. M. Joy and M. Luck, “Plagiarism in programming assignments,” IEEE Trans. Educ., vol. 42, no. 1, Feb. 1999, pp. 129–133.
11. E. L. Jones, “Metrics based plagiarism monitoring “, In: 6th Annual CCSC Northeastern Conference, Middlebury, Vermont, April 20-21, 2001.
12. F. Culwin, T. Lancaster, "Plagiarism issues for higher education", VINE, Vol. 31 Iss 2, 2001, pp. 36 – 41.
13. G. Frantzeskou, E. Stamatatos, S. Gritzalis, and C. E. Chaski. “Identifying authorship by byte-level n-grams: The source code author profile (SCAP)”, Journal of Digital Evidence, 2007.
14. Bozyigit, D. Kılınç, A. Kut, M. Kaya, “Bulanık Mantık Algoritmaları Kullanarak Kaynak Kod Benzerliği Bulma”,In: AB15, 2015, submitted for publication.
15. Vector Space Model. Available: http://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html