PLAGIARISM DETECTION IN PROGRAM CODES USING MAPPING TECHNIQUE

Jakub Kašpar

Master Degree Program (2), FEEC BUT

E-mail: xkaspa36@stud.feec.vutbr.cz

Supervised by: Martin Vítek

E-mail: vitek@feec.vutbr.cz

Abstract: The aim of this paper is to introduce the problem of plagiarism and propose a method for plagiarism detection in program codes. In the first part of this paper the basic definition of plagiarism is described. Further in the paper the principle of preprocessing and localization process for signs of plagiarism is introduced. The last part of this paper presents an algorithm for comparison of the detected signs to get the best results possible. The detector was tested on student projects from the BTBIO study program.

Keywords: plagiarism, detection, preprocessing, signs, mapping

1. INTRODUCTION

One of the most serious problems in the academic field, mostly among students, is plagiarism. There are two types of plagiarism. The first one is plagiarism of scientific papers, publications, books, project or even ideas. The second type is plagiarism of program codes. Plagiarism basically means using someone else's work as your own, without acknowledging the original author. In this work a method for plagiarism detection in program codes is introduced. The proposed method does not declare exactly what is and what is not a plagiarism, but alerts about a probability of two program codes being similar or even the same. Focus on full automatization of the program is also a high priority, so that it could process a large set of data in the shortest time possible. The whole algorithm consists of two main parts, each of them being further divided into smaller elements, as it is shown in the Figure 1.[1]

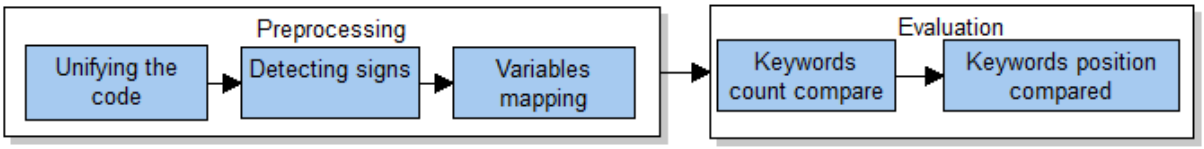

Figure 1: Block diagram

2. PREPROCESSING

Program codes come in a large variety of forms and different formats and often contain a lot of redundant information. Before the beginning of whole sign detection procedure, it is necessary to eliminate all the useless data, such as lines with code commentary, empty lines, and all the blank space in front of and behind the code. Setting all letters to lower case is also helpful for further processing. With these adjustments all processed codes are in a unified format. Without the preprocessing step, adding even minor format changes to the code would lead to inability to detect the similarities. [1]

2.1. SIGNS DETECTION

The next step before the final similarity evaluation of two compared codes is the detection of significant signs. These signs comprise of a large number of keywords typical for the used program environment. In case of the detector proposed in this paper the program environment is Matlab. The keywords were chosen from the most commonly used functions such as *for, while, if,* and also very specific functions, typical for signals and image processing and bioinformatics.

The proposed program scans separately all lines of the preprocessed code and searches for the keywords. Apart from the final count of detected words in the code, the exact position of these words is also recorded. This is very important for further evaluation, as even if the amount of keywords in two codes is the same, it is not necessarily a sign of plagiarism since those keywords can occur in unrelated positions in the code and be used to fulfill a different purpose. Important signs are also the size of the code in bytes and number of rows and characters.

The keywords are searched in different ways. The first group of keywords are the functions, which can be placed only at the beginning of a line e.g. *for* function. For the purpose of finding the keywords the standard *find* function is used. But in case of this kind of functions the position in the code must be also checked along with detection of a *space* sign behind the function. This procedure makes sure, that the detected keyword is actual function and not only part of some variable e.g. *forcount*. It is important that detection of these keywords is done before the space deletion process.

The second group consist of the functions which are usually the first function in a row and have parenthesis behind them e.g. *imshow*. The detection process is very similar to the one used in the first group with exception of the *space* sign detection. There are also keywords like *figure* which can be used in code with or without parenthesis. These words are searched for by combination of both previously described methods.

The third group of keywords are the functions, which can be situated anywhere in the code. The *find* function is again used for detection. Problematic are the functions, which are part of the name of some other function. Good example of this is the *fft* function which is part of the inverse *ifft* function. In order to prevent the false detection in these kind cases, the sign in front of the third group functions are checked and the right function is determined. In the detected keywords a check is carried out whether these keywords are not in the string format, which would mean that they are not used as functions. This process is done by detecting the number and positions of quotes in the row and determining whether or not the keyword is between the active pair.

The last keyword to detect is *end*. This step is done by simple string comparison, since there should not be any other signs on the same line after the preprocessing

2.2. VARIABLES DETECTION

One of the most common ways to hide plagiarism in program codes is to change the names of the used variables. In order to detect these changes, all variables in the code are found and their positions recorded. This process creates maps with coordinates of the variables in all the processed codes which are later compared to each other. Additionally a vector of distances between neighboring variables is created from these coordinates, which is later used for more general comparison.

The variables are detected with the use of information about positions of the first two groups of keywords. The declared variables have to be on every row apart from these positions.

3. EVALUATION

The final part of proposed method is evaluation of the detected signs. The signs from two different codes are compared together and weighted. Weights are not set on constant value, but are instead calculated for every code and sign independently. Every comparison of two signs can take the value from 0 to 1 and so do the weights. The following equations are used for calculation of probability of plagiarism (1, 2) and for the determination of weights (3, 4).

$$P = \frac{\sum_{i=1}^{N} P_i \cdot V_i}{\sum_{i=1}^{N} V_i} \cdot (1 - V_p) \quad (1)$$

$$P_i = \frac{A_i}{B_i} \quad (2)$$

$$V_i = P_{mi} \cdot \frac{4 \cdot P_{bi} + 5 \cdot P_{ri} + P_{ci}}{10} \quad (3)$$

$$V_p = \frac{C_{mi}}{C_i} \quad (4)$$

Here P_i is ratio of two compared signs, A_i and B_i are counts of i-th sign in compared codes, V_i is the weight for compared sign, P_m stands for comparison of the position maps of the sign and is further described below, P_{bi}, P_{ri} and P_{ci} are the ratios of the code sizes in bytes, numbers of rows and numbers of characters respectively. V_p is the weight for the whole comparison, C_{mi} is number of keywords used only in one of the two codes and C_i is number of all keywords in both codes. P represents the probability of plagiarism.

The comparison of the two position maps represented by variable P_m is done by use of the dynamic time warping (DTW) method on the both position vectors, which are used for the calculation of correlation coefficient to compare the similarity. The DTW method makes the shorter of the compared vectors the same length as is the length of the other vector by adding additional values to it, while preserving the initial similarity.

The detector was tested on a dataset consisting of school projects from the APRG course. 100 different codes were divided in to three groups by the topic of the project. All codes in groups were compared to each other and every pair with the higher comparison value than threshold was marked as suspicious from plagiarism. The results of testing are in the Table 1.

Statistics\Projects	Numeric Integrating	Genetics	Ciphers and games
SE	100 %	100 %	100 %
+P	85 %	100 %	77 %
Number of codes	40	21	39
Known plagiarism	4	0	7
Newly discovered plagiarism	7	0	3

Table 1: Results of testing of the proposed detector

Overall 1731 comparisons were made in the total time under 8 seconds. The main problem which caused the false detections were codes, that were made by the same person on top of which the problems were almost identical e.g. numeric integration with rectangle and trapeze method. The sensitivity (SE) results are based on the known cases of plagiarism in the database. Positive predictive value (+P) is based on both known and newly discovered cases of plagiarism.

4. CONCLUSION

The final result of this paper is novel software for detection of plagiarism in program codes. The software is based on similarity comparison of used functions and their positions. Results of the testing has SE 100 % and +P 87 % on average. One of the key advantages of the software are adaptive weights, which bring better results than constant weights. Furthermore the software is fully automatic therefore it is possible to evaluate large set of data in very short time.

REFERENCES

[1] ALSMADI, Izzat, Ikdam ALHAMI a Saif KAZAKZEH. Issues Related to the Detection of Source Code Plagiarism in Students Assignments. *International Journal of Software Engineering and Its Applications* (Vol.8, No.4 (2014), pp.23-34): [cit. 2015-12-30].