Code Similarity and Plagiarism Detection

Mehmed Mustafa

MatrNr: 21914377

Supervisor: Ella Albrecht

February 9, 2020

Contents

1 Introduction

In 2004 researchers reported that plagiarism is a serious threat to education and has started to increase in the past years [9]. Plagiarism can be described as an act of getting someone else's work and represent it as your own, of course by hiding it's source. In academia, mainly plain texts, such as reports, and source code assignments are plagiarized. A simple and good definition for source code plagiarism is given by Hage et al. "Source code plagiarism can be defined as trying to pass off (parts of) source code written by someone else as one's own (i.e., without indicating which parts are copied from which author)" [6]. There are several actions which may lead to source-code plagiarism. Copying source code from online sources is the most obvious one. One similar action is to get source codes from the internet and convert them from one programming language to another. There are some tools which generate source codes from diagrams, and thus a student can give a diagram as an input and get source codes as an output. Yet another action is to perform source code exchange. For example, if a student is able to complete one part of the assignment, and another student is able to complete the other part, then they could exchange their parts with each other. Paying someone else to complete an assignment is also an action which may be considered by students. According to a research [10] there are many reasons why students plagiarize. One reason is the believe that plagiarism will go unnoticed. This believe may motivate a weak student to get the assignment of another student with, or without his/her permission, and edit it before submitting it. One other reason, why students plagiarize, is working in groups on an assignment although they are supposed to work alone. Some students may even not know that it is not acceptable. It is also possible a poorly motivated student, which is not necessarily weak, to get a code of another student and edit it just in order to decrease the amount of work needed for the completion of the assignment. According to another research [13] reason for plagiarism is the desire for obtaining higher grades. The reasons behind this desire of students may be the need of proving self-worth to other students, or a pathological fear of failing. Belief for better job options may also be a reason for wanting higher grades. Sometimes companies, which are looking for new employees, are doing their preliminary selection depending on the grades of freshly graduated students, thinking that higher grades mean better knowledge. Which, of course, is not always the case. Culwin et al. [3] discuss why plagiarism is a problem. Firstly, students committing plagiarism and not getting punished for it, can influence other students and courage them to plagiarize. Secondly, plagiarism also affects negatively the qualification value awarded by an academic institution because students who plagiarize will not get the knowledge required for obtaining a degree. From a moral point of view, plagiarism is a form of lying, because people committing plagiarism are making others believe they are competent at something when they are not. Plagiarism can also negatively affect the creator of the plagiarized work by reducing the amount of respect they deserve or by making them lose money if this work generates an income for them. In order to reduce the impact of plagiarism on education something has to be done. One way to try to prevent plagiarism is to develop systems which will analyze and try to detect similarities among the different works. In this work, the focus is to overview plagiarism (obfuscation) methods used in academia and suggest tools for detecting them. Section 2 analyzes obfuscation methods for source code, Section 3 analyzes obfuscation methods for natural languages, Section 4 suggests plagiarism detection tools and Section 5 concludes.

2 Obfuscation Methods for Source Code Plagiarism

Novak et al. [11], after analyzing 72 articles, which mention some kind of obfuscation methods, they have specified 16 distinct obfuscation methods, which were classified into one of the 4 categories: Lexical, Structural, Advanced Structural and Logical. The first mentioned category, Lexical, is consisted of methods, which are easiest to be detected, while the last mentioned, Logical, is consisted of methods, which are hardest to be detected. We found this classification the best in comparison to other classifications which are either not complete or it is hard for deciding in which category to put some of the obfuscation methods.

2.1 Lexical changes

"Changes which could, in principle, be performed by a text editor. They do not require knowledge of the language sufficient to parse a program." [8] These obfuscation methods are easiest to be spotted, especially for tools which perform similarity comparisons once the code is compiled. Usually students with no to low knowledge are expected to perform lexical changes in order to try to hide their plagiarism. There are 6 obfuscation methods in this category:

1. Visual code formatting - a method by which a student would just change the amount of space, usually at the beginning of each line before the actual code, or insert new lines consistent only from whitespace characters. The opposite is also possible - removing the empty lines, which are usually set by programmers in order to improve readability;

2. Comments modification - this method can include partly or fully adding, modifying or removing comments. Usually comments added by this method will be placed at places where, actually, no comment is required and the code is obvious. For example, placing "Check if grade is greater than 50" comment for this line of code "if(grade > 50)". Although this may be a clear sign which invites to have a closer look at the code, changing the comments by paraphrasing or completely deleting them has no such signs;

3. Modifying program output - changing the output messages or reformatting the output shape of objects, such as tables, or changing the *Graphical User Interface* (GUI) by modifying the content labels or re-positioning of labels, radio buttons and so on. If modifying of the GUI, however, depends on languages like HTML or CSS this modification can be classified as another obfuscation method - "reordering independent lines of code" which belongs to the "Structural changes" category;

4. Identifier rename - modification of identifier names, like variable names, class names, function names, constant names and so on. Sometimes students may not change all of the identifier names, which will lead to inconsistencies in the programming style;

5. Translation of program parts - this obfuscation method could be seen as a special case of the previous three methods, because translation to another language could be done for identifier names, program output and comments. For example, translating from English to another language which is native for the student performing plagiarism;

6. Changing constant values - sometimes constants within a program could be changed without affecting the logic or output of the program. For instance, the , which is 3,14 could be written with higher precision - 3,1415. Although this method may look similar to identifier rename method, it may require more knowledge to be performed. In the given example, a student may also have to change the output precision to make output look similar;

```
1    int main()
2    {
3        const int SEED_RANDOMIZER = 7;
4
5        int number, guess;
6        number = RNG(SEED_RANDOMIZER);
7
8        printf("Please  enter your number guess:");
9        scanf("%d", &guess);
10
11       /* Check if the number matches user's guess */
12       if(number == guess)
13           printf("Congratulations, you won the game!\n");
14       else
15           printf("Wrong guess! You lost the game!\n");
16
17       return (0);
18   }
```

Figure 1: Original source code

```
1    int main(){
2        const int NIVO_SLUCHAINOST = 12;
3
4        int chislo, dogatka;
5        chislo = RNG(NIVO_SLUCHAINOST);
6
7        printf("Моля въведете число за отгатване:");
8        scanf("%d", &dogatka);
9
10       /* Проверява дали числото съвпада с докатката */
11       if(chislo == dogatka){
12           printf("Поздравления, ти спечели играта!\n");
13       }
14       else{
15           printf("Лош опит! Ти загуби играта!");
16       }
17
18       return (0);
19   }
```

Figure 2: Plagiarized source code with lexical changes

Figure 1 and Figure 2 are examples respectively for original and plagiarized source code using lexical changes. All of the methods described above are used in the plagiarized code. Translation of program parts method covers identifier rename, modifying program output and comments modification.

2.2 Structural changes

"Requires the sort of knowledge of a program that would be necessary to parse it. It is highly language-dependent." [8] In comparison to lexical changes, which don't require any programming knowledge, structural changes require some knowledge about the connection of different lines of code. There are 4 obfuscation methods in this directory:

1. Reordering independent lines of code - all kinds of line reordering which do not change the working flow of a program. Such modifications could be achieved by changing the order of variable declarations or by reordering the position of function declarations or inner classes;

2. Adding redundant lines of code - could be achieved easily by adding codes to unreachable sections of the program or lines, such as "b=b;" anywhere. Fortunately, any *Integrated development environment* (IDE) would warn if there are lines of code which are unreachable or redundant, so this method is not very useful to be plagiarized with, unless the source codes are only automatically analyzed by tool which does not support the feature of the current generation development environments;

3. Splitting up lines of code - the idea behind this method is to make the code seem longer than the original version. It could be performed on single line by splitting it to many lines, or splitting one class to many smaller classes, or splitting one function into many smaller functions and etc.;

4. Merging lines of code - this method could be seen as the opposite of splitting. Smaller classes, functions or files could be combined into one bigger. Simplest version of this method is to combine declaration or initialization of variables in less lines;

```c
int main()
{
    double ia0, ia1;    /* Coefficients of the input polynomial */
    double p0;       /* P(0) of the output polynomial */
    double ra0, ra1, ra2; /* Coefficients of the output polynomial */

    /* Get the 1st degree input polynomial */
    printf("Enter coefficients from higher to lower order:\n");
    scanf("%lf%lf", &ia1, &ia0);

    /* Get the constant value for the output polynomial */
    printf("Enter p(0) for the output polynomial:\n");
    scanf("%lf", &p0);

    /* Calculate the 2nd degree output polynomial */
    calculate2ndDegree(&ra2, &ra1, &ra0, ia1, ia0, p0);

    printf("Result: %5.3fx^2 + %5.3fx + %5.3f\n", ra2, ra1, ra0);

    return (0);
}
```

Figure 3: Original source code

Figure 3 and Figure 4 are examples respectively for original and plagiarized source code using structural changes. For example, reordering independent lines of code and merging lines of code methods are used to transform lines 3-5 in Figure 3 to lines 3-4 in Figure 4. Also the function call on line 16 in Figure 3 is substituted with the implementation of that function and its independent lines of code are distributed to lines 10 and 16-17 in Figure 4. There are also lexical changes, like comments modification.

```
1    int main()
2    {
3        double ra0, ra1, ra2;
4        double p0, ia0, ia1;
5
6        /* Get the constant value for the output polynomial */
7        printf("Enter p(0) for the output polynomial:\n");
8        scanf("%lf", &p0);
9
10       ra0 = p0;
11
12       /* Get the 1st degree input polynomial */
13       printf("Enter coefficients from higher to lower order:\n");
14       scanf("%lf%lf", &ia1, &ia0);
15
16       ra2 = ia1 * 0.5;
17       ra1 = ia0;
18
19       printf("Result: %5.3fx^2 + %5.3fx + %5.3f\n", ra2, ra1, ra0);
20
21       return (0);
22   }
```

Figure 4: Plagiarized source code with structural changes

2.3 Advanced structural changes

"Subcategory of structural changes that require more knowledge of program possibilities and relations between equivalent statements in a specific programming language." [11] This category is the advanced version of the previous one. Students which are able to perform these obfuscation methods have some knowledge about modifiers, control structures like if, switch, for or while and different data structures.

1. Changing of statement specification - this method could be further divided in future sub methods - altering modifiers, changing data types and changing operations and operands. Altering modifiers could be performed, for example, by changing "private" variables or functions of a class to "public". Changing "float" to "double", "int" to "long int" or "arrays" to "array lists" are cases of changing data types. Modifying operands or operations in logical expressions is a bit more complicated than the previous two sub methods. For example, changing "(x == y)" to "!(x != y)" or "(x <= 5) to "(6 > x)"

2. Replacing control structures with equivalents - "if" statements changed to "switch" or vice-versa is an example for this method. Other possible changes are "for" to "while" or "do...while";

5

```
1    int x = 0, y = 0, z = 2;
2    for(int i=0; i<=9; ++i){
3        if(x == y){
4            z = z + 9;
5        }else{
6            ++z;
7        }
8    }
9
10
11   int i = 0, x = 0, y = 0, z = 2;
12   while(10 > i){
13       if(x != y) ++z;
14       else z = z + 9;
15   }
```

Figure 5: Example of original and plagiarized code snippets

Figure 5 shows two snippets of code, the first 8 lines are the original source code, the lines after line 10 are plagiarized using advanced structural changes. The difference between the two snippets is the exchange of looping control structure - "for" is substituted with "while" and changes in statement specifications - "x == y" in the original code is exchanged with "x != y" in the plagiarized code. Of course, the lines which are executed depending on the value of the boolean expression are also swapped to keep the functionality of the code and the output same (lines 4 and 6 are swapped in lines 13 and 14). The boolean expressions controlling the two loops are also exchanged - "i <= 9" is substituted with its equal "10 > i". These two advanced structural methods combined with lexical and structural obfuscation methods could work even better for hiding plagiarism. For example, if the variable names are changed and there are some additional comments, the two snippets of code, at first glance, will look completely different.

2.4   Logical changes

"Changes that except for structural changes also change the logic (flow) of a program and require certain amount of programming skills and knowledge about the application being developed to be performed correctly." [11] This category contains the most advanced techniques for hiding plagiarism and usually these are hard to be detected by a human or a plagiarism tools. If a student is able to use some of these methods it means that he or she has enough knowledge about programming or about the target programming language. There are 4 methods in this category:

1. Simplifying the code - removing the complex or unneeded parts of the copied source code. For example, students may get an assignment in which they have to implement Dijkstra's algorithm for finding the shortest paths between nodes in a graph. There are a lot of applications using this algorithm one or another way. The students who

want to plagiarize could just copy the algorithm part and leave out other details of the application;

2. Translation of program from other programming language - converting source code written in one programming language to another. Usually code needed for an assignment or part of it could be found on the Internet, but written in another programming language and since the student performing plagiarism will also need to change some of the logic within the copied program this method is not only structural;

3. Changing the logic - this method requires the most knowledge from the student's side and it is the most complicated method. Sometimes even if the student has all the programming knowledge he or she needs to complete an assignment, he or she may lack an idea how to design and implement the application or project. One other case may be when a student has completed his assignment, but created variations of his or her source code to be able to sell it to other students which are lacking skills to complete the assignment alone;

4. Combining copied and original code - this obfuscation method could be either easy or very complicated to be performed depending on the level of mixing between the copied code and the original code of the student. Some logical changes may be needed for a successful mix;

3   Obfuscation Methods for Natural Languages

Detecting plagiarism in natural language texts is just as important as the detection of plagiarism in source codes, because students, in addition to their assignments, may have to write reports or research papers as well. However, plagiarism is not limited only to students and may be performed even by teaching assistants. There are some obfuscation methods, which are used to surpass plagiarism detection tools. Some of these methods may have already been known and the tools patched to be able to detect them.

3.1   Substitution methods - whitespace, alphabet or words

We have classified the substitution methods into three subgroups, because their main and similar part is the substitution - whitespace substitutions [5], latin-greek-cyrillic alphabet letters substitutions [5] or word(synonym) substitutions [12]. These 3 methods are similar to each other. The whitespace substitution method is the action of replacing white space characters (spaces, tabs, feeds...) with another character and making the font of that character white. In the eyes of the instructor, this white font character will look like a white space character. On the other hand, the plagiarism detection tool, since there will be no white space characters, will see the whole text as a single word. Then when the tool compares it to the same text, but with white space characters, it will detect completely different content, although it is not. An easy way to detect this method is to check the words occurring inside a text and try to match them with words inside a dictionary. If there are some random white font characters placed here and there, the detection tool will fail to match them and if the

matching percentage is below some level, it may be an indication that something is wrong with the submitted work.
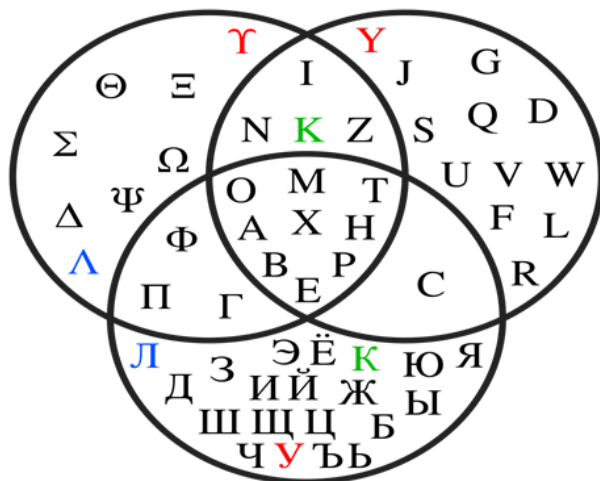

Figure 6: Similar looking letters in the Greek, Cyrillic and Latin alphabets. [14].

The second method is similar to the first, but this time the substitution happens between letters, which have the same appearance, but different *American Standard Code for Information Interchange* (ASCII) code value. Figure 6 shows a Venn diagram of letters, which have the same appearance in different alphabets - Greek, Cyrillic and Latin. This obfuscation method can be countered the same way as the first one - by trying to match words to dictionary words. The last obfuscation method of this subsection is the substitution of words with their synonyms without changing the structure of sentences. This method is harder to be detected in comparison to previous two methods, because a simple word match will not help. However, the tools could still be able to label words with their synonyms as similar, if there is a synonym dictionary available for the tool.

3.2   Usage of "invisible" characters

This method was derived by taking inspiration from the three methods above. In the Unicode character set, which is the world standard for text and can be seen as an extension of ASCII, there are some special kind of characters which are invisible and are used as formatting symbols for languages like Korean and Chinese. One example is the Unicode Character with code U+3164 named Hangul Filler [1]. Such symbols may be used inside sentences which are plagiarized and between words of those sentences. We have tested this obfuscation method by creating 2 files with exactly the same sentence as content. Then used "invisible" characters in one of the files. Then tested on two online plagiarism detection tools. Both of them failed to detect the plagiarized sentence with exactly the same content. One of the online tools was copyleaks [2] and it gave 0% similarity results in all of its categories: identical, minor changes and related meaning. The second tool was diffchecker [4]. Figure 7 shows the results of diffchecker, which failed as well.

Figure 7: Online plagiarism detection tool diffchecker fails to detect plagiarized content [4].

4 Available Tools for detecting Source Code Plagiarism

First research papers mentioning some kind of source-code plagiarism detection are dating back to the 1970s. However, the interest in this topic has grown in the last 15 years. This statement is supported with the fact that the number of source-code detection tools has started to increase at the beginning of 21st century. Check out Figure 8.

It shows the number of tools developed per year between 1980 and 2015. If instructors use such tools for preliminary check of assignments, and then observe only works which were labeled as "highly similar", it will be easier to detect plagiarism and the motivation of students, who plagiarize or think to plagiarize, will be decreased. However, using the source-code plagiarism detection tools may not be enough for a final conclusion. Sometimes the tools may produce false results and instructors should always approach these results critically. One research paper states the following: "It is certainly safe to say that neither the detection system described in this paper nor any other detection system will find all occurrences of plagiarism. There an inherent tradeoff between a highly discriminatory system, which overlooks some instances of cheating, and a less discriminatory one which flags many dissimilar programs." [7]. Although the increased number of plagiarism detection tools in the past years, it is still hard to decide, which tools are good and for what. Novak et al. [11] give an overview of "top-five" plagiarism detection tools. They have used two criteria to rank the tools. The first criterion is the number of mentions. The second criterion is the comparison number. A tool should have been compared to other tools at least four times to take place in the ranking. More a tool is mentioned in research papers and compared to other tools better the ranking is.

Table 1 shows the "top-five" tools and gives information whether they are open source,
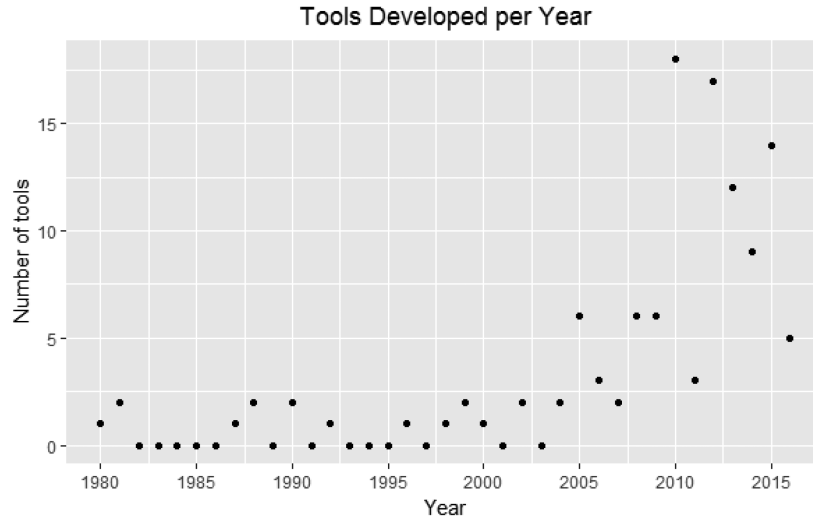
Figure 8: Number of tools developed per year. [11].

have GUI and are available offline. This ranking is not surprising, because all of the tools ranked high are free for use and most of them are open source. Hence, they are more likely to be compared and used. The selection criteria leaves out tools which are not open to the public or are developed by authors and discussed only in their research paper without comparisons to other tools, which makes the quality of the developed tool questionable. If a tool is not available for usage, although it could be the best tool ever, it has no use for researchers and instructors. Also offline availability of a tool could be a useful feature, since some universities may have privacy policy which forbids sending data of students to servers outside of the campus, and thus making tools available only online impossible for use.

Tool Name	Mentions	Comparisons	Open Source	GUI	Available Offline
JPlag	43	37	YES	YES	YES
MOSS	38	29	NO	YES	NO
Sherlock-Warwick	9	4	YES	YES	YES
Plaggie	7	6	YES	YES	YES
SIM-Grune	6	4	YES	NO	YES

Table 1: Top 5 plagiarism detection tools suggested by [11].
JPlag - https://jplag.ipd.kit.edu
MOSS - https://theory.stanford.edu/∼aiken/moss/
Sherlock-Warwick - http://warwick.ac.uk/iasgroup/software/sherlock
Plaggie - https://www.cs.hut.fi/Software/Plaggie
SIM-Grune - https://dickgrune.com/Programs/similarity_tester

5   Conclusion

Most of the students are aware of the wide availability of resource on internet and they can be very creative when it comes to hiding plagiarism. Students can find solutions to their assignments and use different techniques, obfuscation methods, which may trick the instructors. Instructors should be aware enough about these techniques, so they can have some hints when trying to decide if a work is plagiarized or not. Of course, when there is a big amount of students, taking particular course, it becomes harder for the instructor to spend enough time for trying to detect plagiarism manually. Sometimes students are aware of this fact, and they feel more confident to plagiarize. If the process of detecting source-code plagiarism is automatized then students will think twice before committing source-code plagiarism. However, instructors should not depend completely on the results produced by plagiarism detection tools because sometimes they may be misleading. In order to reduce plagiarism maximally instructors could also take some additional steps. For example, they could try to avoid giving assignments that have general solutions. Moreover, they could try to give different assignments each semester, which will greatly reduce plagiarism between students from different semesters. Additionally, instructors could choose assignments that allow several interpretations.

Abbreviations and Acronyms

ASCII *American Standard Code for Information Interchange*

IDE *Integrated development environment*

GUI *Graphical User Interface*

SCP *Source Code Plagiarism*

List of Tables

List of Figures

References

[1] Compart. Unicode character u+3164. Available online: `https://www.compart.com/en/unicode/U+3164`, last retrieved 01.02.2020.

[2] Copyleaks. Plagiarism detection tool - copyleaks. Available online: `https://copyleaks.com/compare`, last retrieved 02.02.2020.

[3] Fintan Culwin and Thomas Lancaster. Plagiarism issues for higher education. *VINE*, 31, 06 2001.

[4] DiffChecker. Plagiarism detection tool - diffchecker. Available online: `https://www.diffchecker.com/`, last retrieved 02.02.2020.

[5] ESSAY GUARD. How to cheat plagiarism detection software. Available online: `https://essayguard.com/how-to-cheat-plagiarism-detection-software`, last retrieved 02.02.2020.

[6] Jurriaan Hage, Peter Rademaker, and Nikè Vugt. Plagiarism detection for java: a tool comparison. pages 33–46, 04 2011.

[7] Ann-Marie Lancaster John L. Donaldson and Paula H. Sposato. A plagiarism detection system. *ACM Transactions on Computing Education*, 19:21–25, 02 1981.

[8] M. Joy and M. Luck. Plagiarism in programming assignments. *IEEE Trans. Educ.*, 42(2):137–162, 1999.

[9] Hammond M. Cyber plagiarism: are fe students getting away with words. plagiarism: Prevention, practice and policies 2004 conference;. pages 257–264, 06 2004.

[10] Schiller R M. A plagiarism detection system. *Journal of the American Dietetic Association*, 105(7):1058–1062, 04 2011.

[11] Matija Novak, Mike Joy, and Dragutin Kermek. Source-code similarity detection and detection tools used in academia: A systematic review. *ACM Transactions on Computing Education*, 19:1–37, 05 2019.

[12] QUORA. Is there a way to fool turnitin? Available online: `https://www.quora.com/Is-there-a-way-to-fool-Turnitin`, last retrieved 02.02.2020.

[13] Bennett R. Factors associated with student plagiarism in a post-1992 university. *Assessment and Evaluation in Higher Education 2005*, 30(2):137–162, 2005.

[14] Wikipedia. Letter (alphabet). Available online: `https://en.wikipedia.org/wiki/Letter_(alphabet)`, last retrieved 01.02.2020.