A Software Plagiarism Detection Approach based on Winnowing through Extracting Code Skeletons

Osman Araz, Muhammet Çeneli
Bilgisayar Mühendisliği Bölümü
Yıldız Teknik Üniversitesi, 34220 Istanbul, Türkiye
arazosman@outlook.com, muhammetceneli@gmail.com

***Özetçe* —Eğitim kurumlarındaki yazılım üzerine verilen ödevlerde, kod intihali sıklıkla karşılaşılan bir sorundur. Bu soruna çözüm aramak için günümüze dek onlarca yöntem geliştirilmiştir. Bu makalede, günümüzde kod intihali tespiti için en sık olarak kullanılan Measure Of Software Similarity (MOSS) isimli yöntem incelenecek ve bu yöntem üzerinde, döküman içeriklerinin sadeleştirilmesi yoluyla başarım oranının arttırılması hedeflenecektir.**

Anahtar Kelimeler—yazılım intihali, kod intihali, Measure of Software Similarity, MOSS, Karp-Rabin algoritması, winnowing algoritması, döküman parmak izleri

***Abstract*—Software plagiarism is a common problem in educational institutions for software assignments. To offer a solution to this problem, dozens of methods have been developed. In this article, Measure Of Software Similarity (MOSS) algorithm, which is the most commonly used method for software plagiarism detection, will be focused with a method for simplification of document contents, thus it is aimed to increase the success rate.**

Keywords—code plagiarism detection, document plagiarism detection, Measure of Software Similarity, MOSS, Karp-Rabin algorithm, winnowing algorithm, document fingerprints

I. Introduction

During the spring 1990 semester in MIT, in the *Introduction to Computers and Problem Solving* course, the professor conducted an experiment without the students being aware. For this experiment, students were given a programming assignment and any copy was strictly forbidden. As a result, it was determined that 80 of the 239 students were copying. This result shows how high the plagiarism in programming assignments can be.[1]

The common plagiarism techniques are used in programming assignments are listed below:

- Changing program name.
- Changing comments on code.
- Changing variable and function names.
- Changing spaces and indents.
- Changing the order of the variables and functions.
- Changing types of loops and types of conditional statements.
- Adding or removing redundant elements.

A well-designed software plagiarism detection method should handle the tricks listed above.

II. Extracting the Code Skeleton

A plagiarized code might not be seem similar to the original code at first sight. A meticulous student may change all variable and function names and order of them, changes comments and indentation in the code, adds or removes redundant elements etc. To ignore these differences, the code must be simplified to a specific form.

To perform it, we used some rules to transform code to its skeleton. (We will call the simplified code as *skeleton* in this article.) These rules can be adopted to any programming language, but we will go with *C* programming language for now.

A. Step 1: Remove redundant elements from the source code.

- Comment blocks: Comment blocks have no effect to the compiled code, therefore these are should not be in the code skeleton.
- Arbitrary spaces and indents: The code skeleton should not have spaces. It's because that the code skeleton will be split into pieces, which named (and will be described later in this article) as *fingerprint*s.
- Contents of strings: Most of string contents are sake for inform the user. So, these are also should not be in the code skeleton.

B. Step 2: Extract the code skeleton.

In this step, we make all the variable names same, as like as function names, loop types and conditional statement types.

- Variable and function names: We will re-factor all the variables as `_var_`, all the functions as `_fnc_` and all the function parameters as `_par_`. For example, the statement `ans = 5 * abs(-2)` will be transformed to `_var_=_var_*_fnc_(_par_)`.
- Loop types: We will re-factor all the loop types, such as `for`, `while` and `do-while`, as `_lop_`. For example both of the statements `while (i < 10)` and `for (i = 0; i < 10; ++i)` will be transformed to `_lop_(_var_<_var_)`.
- Conditional statement types: We will re-factor all the conditional statement types, such as `if`, `else if`, `else` and `switch-case` as

cnd. For example, both of the statements `if (x == 0) printf("0"); else printf("?");` and `switch (x) { case 0: printf("0"); break; default: printf("?"); }` will be transformed to `_cnd_(_var_==_var_)_fnc_(_par_); _cnd_()_fnc_(_par_ );`

While extracting the code skeleton, variable and function declarations are ignored. Declarations have no effect to the similarity. The key similarity criterion are assignments, mathematical operations, function calls, and usage of loops and conditional statements.

III. Document Fingerprinting

Fingerprinting is one of the most commonly used method of plagiarism detection. A fingerprint is a contiguous sub-string of a document. The specialized version of fingerprinting method is called as *k-grams*. A k-gram is a fixed sized (`k`) fingerprint. The main idea behind the document fingerprinting is splitting the document into k-grams and putting them into a hash table. To calculate similarity rate of two documents, we use these hash tables.

For text `a game of thrones`, after removing redundant elements, the text will become as `agameofthrones`. 5-grams of the text will be `agame`, `gameo`, `ameof`, `meoft`, `eofth`, `ofthr`, `fthro`, `thron`, `hrone` and `rones`.

IV. Karp-Rabin Algorithm

To hash a string, it's needed to multiply every character of the string with a value. Let the string size be `l`, then the number of k-grams of the string will be `l-k+1`. In a straight approach, we need $(l-k+1)\times k$ multiplying operations to calculate hash keys of all k-grams, which could be very expensive for long k-grams. Hash function of the straight approach is given below:

$$hash(c_1...c_k) = c_1 * R^{k-1} + c_2 * R^{k-2} + ... + c_k$$

Karp-Rabin is a string searching algorithm that uses hashing. Firstly, it calculates hash key of the first k-gram. Afterwards, for every k-gram, it uses previously calculated key to calculate the current key. This approach prevents same multiply operations for countless times. Therefore, there will be only $k + (l-k) = l$ multiplying operations. Hash function of the Karp-Rabin approach is given below:

$$hash(c_2...c_{k+1}) = (hash(c_1...c_k) - c_1 * R^{k-1}) + c_{k+1}$$

V. Winnowing Algorithm

When we calculate the similarity rate of two documents, we use their fingerprints. For numerous documents, it could be very expensive to use all of the fingerprints. A simple but incorrect approach is, selecting every i^{th} fingerprint. This approach is weak when the plagiarized document has insertions, deletions are reordering. A clever approach should not be based on positions of fingerprints.

Winnowing algorithm[2] is a selection algorithm for fingerprints. This algorithm uses a fixed-size (`w`) window which slides on the hash keys of k-grams. In each window, the minimum hash key will be selected, if the key not selected before. If there is more than one minimum hash key, the rightmost key will be selected.

Assume that the hash keys of fingerprints of a string are calculated as `77 74 42 17 98 50 17 98 13 88 11 39`. For window size 4, steps of sliding window are listed below:

(77 74 42 **17**)	(74 42 17 98)	(42 17 98 50)
(17 98 50 **17**)	(98 50 17 98)	(50 17 98 **13**)
(17 98 13 88)	(98 13 88 **11**)	(13 88 11 39)

Selected fingerprints are shown in bold-face: `17, 17, 13, 11`. The intuition behind choosing the minimum hash key is that the minimum hash key in a window is likely to be the minimum hash key of the next window.

VI. Calculating the Similarity Rate

Similarity rate of two documents is calculated as counting the same fingerprints of documents and then divide it to the total number of fingerprints for each document. It means that there will be a pair of similarity rate. One is the similarity rate of the first document over the second document, and the other is vice versa. The mathematical notations are given below:

$$sim(A,B) = \frac{|f(A) \cap f(B)|}{|f(A)|}$$

$$sim(A,B) = \frac{|f(A) \cap f(B)|}{|f(B)|}$$

$sim(A,B)$ refers to similarity rate of document A over document B. $f()$ function refers to set of fingerprints.

The advantage of bidirectional similarity rate is that it can handle the situation even the document sizes are very different. If the small document plagiarized the large document, then the similarity of the small document over the large document will be high, but not vice versa.

VII. Performance Analysis

There are two main parameters for this work. First is that the length of fingerprints, which is the `k` of k-grams. And the second is that the window size, which is a winnowing concept.

A. Length of Fingerprints

Changing the length of fingerprints changes the precision. Bigger length means bigger threshold. The similarity rate mostly decreases with the increase in fingerprint length. It has not much effect on the time complexity of the system.

The following table shows the test results with different fingerprint lengths on various data sets. Each data set was

tested on four different fingerprint lengths of 5, 25, 50 and 100. Elapsed times are showed in seconds and the window size specified as 5.

Table 1 Elapsed Times for Different Fingerprint Lengths

Dataset	Size	k = 5	k = 25	k = 50	k = 100
c-simple	5	00.048	00.044	00.046	00.078
c-large	1000	09.946	10.730	11.482	09.758
txt	100	00.272	00.304	00.303	00.293
txt-large	1000	24.017	27.973	27.665	26.370

B. Window Size

Window size is used for optimization while selecting fingerprints. It can highly reduce the number of selected fingerprints, and so elapsed time of the program.

The following table shows the test results with different window sizes on various data sets. Each data set was tested on four different window sizes of 1, 5, 25 and 50. Elapsed times are showed in seconds and the fingerprint length specified as 25.

Table 2 Elapsed Times for Different Window Sizes

Dataset	Size	w = 1	w = 5	w = 25	w = 50
c-simple	5	00.053	00.050	00.058	00.103
c-large	1000	21.669	12.242	08.813	08.223
txt	100	00.926	00.312	00.092	00.064
txt-large	1000	86.962	29.720	06.526	03.430

The table shows that the effect of the winnowing algorithm increases as the size of data set grows.

VIII. Experiments

Due to measure the performance rate of this work, there will be some comparisons with the MOSS system, which is the most commonly used method for software plagiarism detection.

We created five simple `C` codes and test it on our work and the MOSS system. There are four plagiarized code over the original code. One have changed variable and function names, one have changed spaces and indentations, one have changed loop type (`for` loop to `while` loop) and the other have changed conditional statement type (`if-else` statement to `switch-case` statement).

The results are shown below:

Table 3 Comparison of this work and the MOSS system

Code 1	Code 2	Work		MOSS	
code.c	names.c	100%	100%	95%	95%
code.c	spaces.c	100%	100%	95%	95%
code.c	conditions.c	100%	100%	83%	81%
code.c	loops.c	74%	66%	28%	28%

For both of the systems, the first rate refers to the similarity of first code over the second code, and the second rate vice versa.

Changing variable and function names and spaces and indentations have similar effect to both of the systems. But when the loop type is changed, the performance of MOSS system dramatically decreased. These results shows that extracting code skeletons may be very helpful.

IX. Conclusions

The general overview and the steps of this work can be summarized as;

1) Removing redundant elements.
2) Extracting the code skeleton.
3) Hashing fingerprints of the skeleton with Karp-Rabin algorithm.
4) Selecting the fingerprints with winnowing algorithm.
5) Calculating the similarity rate with using the selected fingerprints.

As shown before, extracting the code skeletons may be very helpful for software plagiarism detection. It prevents tricks like changing variable and function names, changing spaces and indentations, changing loop and conditional statement types etc.

References

[1] N. R. Wagner, "Plagiarism by student programmers," 2000.

[2] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting."