The Journal of Systems and Software 83 (2010) 2478–2486

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Code analyzer for an online course management system

Jong Yih Kuo*, Fu Chu Huang

Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 14 December 2009
Received in revised form 12 May 2010
Accepted 15 July 2010
Available online 24 July 2010

Keywords:
Intelligent agent
Program similarity
Program plagiarism

ABSTRACT

The online course management system (OCMS) assists online instruction in various aspects, including testing, course discussion, assignment submission, and assignment grading. This paper proposes a plagiarism detection system whose design is integrated with an OCMS. Online assignment submission is prone to easy plagiarism, which can seriously influence the quality of learning. In the past, plagiarism was detected manually, making it very time-consuming. This research thus focuses on developing a system involving code standardization, textual analysis, structural analysis, and variable analysis for evaluating and comparing programming codes. An agent system serves as a daemon to analyze the program codes for OCMS. For textual analysis, the Fingerprinting Algorithm was used for text comparison. Structurally, a formal algebraic expression and a dynamic control structure tree (DCS Tree) were utilized to rebuild and evaluate the program structure. For variables, not only the relevant information for each variable was recorded, but also the programming structure was analyzed where the variables are positioned. By applying a similarity measuring method, a similarity value was produced for each program in the three aspects mentioned above. This research implements an Online Detection Plagiarism System (ODPS) providing a web-based user interface. This system can be applied independently for assignment analysis of Java programs. After three comparison experiments with other researches, the results demonstrated the ODPS has many advantages and good performance. Meanwhile, a combined approach is proven that it is better than a single approach for source codes of various styles.

1. Introduction

With growing attention being devoted to internet-assisted teaching, many online course management systems have been developed (Broder, 1997; Adaptive Brusilovsky, 2001; Perkowitz and Etzioni, 1997). Online course management systems support chiefly various activities in online teaching, including testing, course discussion, assignment submission, and assignment grading (Frasson and Aimeur, 1998). However, the coding style for programming often varies according to each programmer's personal habit, and is reflected in the use of different techniques in designing with the same programming language. If the similarity between the two samples is high, then it is possibly a case of plagiarism. To plagiarize a program, most people rewrite the original version, including part or all of its interface or contents, to avoid easy human identification.

If the differences between codes are compared manually, it will be very time-consuming. As the coding scale becomes larger and more complex, the success rate of identification also diminishes. In the past, purely textual or structural comparison methods have generally been employed to compare documents and programs, though these methods all have their limitations. This is in part because slight differences in textual comparison will cause the similarity rate to vary significantly. For example, "System.out.println("This is my first java program, hello world!");" and "String str = "This is my first java program, hello world!";" involve completely different programming source codes, but there appears to be many similarities. If it is only evaluated by textual comparison, the output similarity value may be greater than expected. In addition, both structural analysis and variable analysis are liable to identify two programs of the same structure, but for different purposes as having a high similarity match. To improve this situation, combining code standardization, textual analysis, structural analysis, and variable analysis was proposed for enhancing the comparison ability.

There are a number of methods for detecting plagiarism in assignments, and most methods focus only on how to improve similarity. Although increasing the similarity is useful for detecting plagiarism in assignments, how to make a correct detection is also very important. Using a single method will probably get an overly high or low similarity in different cases and it is very easy to make wrong estimation. In view of this, three analysis methods were combined to build an Online Detection Plagiarism System

* Corresponding author. Tel.: +886 2 27712171x4237; fax: +886 2 87732945.
E-mail addresses: jykuo@ntut.edu.tw (J.Y. Kuo), t7599002@ntut.edu.tw (F.C. Huang).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.07.037

(ODPS) plugged into an online course management system (OCMS) for detecting similarity. The ODPS serves as a daemon module for performing analysis. Pre-analyzing the sample through structural and variable analyses before textual analysis will increase the precision of textual analysis. For textual analysis, the Document Fingerprinting Algorithm is employed. For structural analysis, the formal algebraic expression and the DCS Tree structure are used to rebuild and examine the program composition. For variables, not only the information of each variable is a relevant record, but also the formation of variables for calculating similarity is also structurally analyzed. As for fingerprinting, the document is translated into k-grams, and finds a unique fingerprint representing this document. Structural analysis uses formal algebraic expression to simplify the code structure, and applies textual comparison to the structural comparison problem to increase the accuracy of output similarity. In terms of variable analysis, each variable in the code is focused, and both statistical and architectural analyses were performed. This allows the comparison process to be more flexible, and produces a more precise similarity output. Thus, by combining textual analysis, structural analysis, and variable analysis, programs that are likely to have been intentionally plagiarized can be distinguished.

This research examines previously developed online course management systems, because submitted assignments are easily copied. Programming courses are focused, and the processes such as code standardization, textual analysis, structural analysis, and variable analysis are applied to code evaluation. A hybrid approach is proposed by combining and advancing three methods including the Fingerprinting Algorithm (Schleimer et al., 2003), algebraic expression (Canfora et al., 1998) and variable statistics (Donaldson et al., 1981). The rest of the paper is organized as follows. Section 2 discusses the references used in our research. Section 3 presents the proposed analysis method for the codes. Section 4 presents the system design and testing results, and Section 5 contains conclusions.

2. Background and related work

2.1. Textual analysis

Schleimer et al. (2003) proposed winnowing as a method for textual comparison of web data. In this method, the tokens of document are first partitioned into k-grams. k signifies the number of characters in each k-gram block dictated by the user according to the length of the document. After partitioning, all k-grams are stored into a hash table for distribution. Then a partition is made according to the hashed results, creating "fingerprints", the smallest unit of comparison. In other words, fingerprints are sets of k-grams after implementing the hash algorithm. The compared document creates one or more fingerprints. The possibility of collision is minimized because the hash algorithm is utilized. To compare two documents, the same fingerprints mean the same original contents. The number of characters between two k-grams is called the "gap", usually in a size of one. When the documents are longer, it is advisable to use larger gap values to decrease the repetition of fingerprints, thus increasing system efficiency.

Winnowing algorithm: Set three variables, where n is the length of the target document with the white spaces removed. k is the length of a k-gram (in number of characters). g is the number of characters between two adjacent k-grams. Variables k and g are both user-specified, but note that $n \geq k$ must be satisfied. If $n < k$, then the system will not be able to produce multiple k-grams for comparison and the comparison result is produced only by chance. For example, consider a sentence such as "I am a cat" with n value of 7. If the value of k is set to be 8, there would be no k-gram. Therefore, it is important to choose an appropriate value of k. The document is then partitioned into k-grams. For example, when $k = 4$ and $g = 1$, if the target document is "I am a super cat" with the white spaces removed, then the derived k-grams in order are shown in Table 1. Next, these k-grams are processed with a hash function to retrieve their hash value. All hash values $(h_1, \ldots, h_n)$ are placed in a numerical sequence. The set of some hash values in this sequence is called the "window", with its size also being user-specified. If window size is represented by the variable w, then the range of every window is $(h_i, \ldots, h_{i+w-1})$, with i being the position of the first character in this window in relation to the hash sequence. For the character at position i, its corresponding window range is $1 \leq i \leq n - w + 1$. Last, by juxtaposing the hash values in every window, the section that is being plagiarized can be efficiently detected. This is because when a minimum value appears inside one window, then it is quite possible that this value is also apparent in an adjacent window. Therefore, only the minimum hash value in the first window is recorded for overlapping windows. Only the unrepeated minima are called fingerprints. At the same time, the positions of each fingerprint are also recorded as a basis for comparison, thus increasing the accuracy of analysis.

Table 1
k-grams of "I am a super cat".

iama	amas	masu
asup	supe	uper
perc	erca	rcat

Gitchell developed a plagiarism detection system to compare token sequences using a dynamic programming string alignment approach (Gitchell and Tran, 1998). This approach first assigns a score to each pair of characters in an alignment score. For example, a match score of 1, a mismatch score of −1, and a gap score of −2. The highest score of a block gives the score of an alignment. The score between two sequences is then defined as the maximum score among all alignments, which is easily calculated by dynamic programming techniques. With this definition a similarity measure between two sequences is defined as follows:

$$S = \frac{2 * \text{score(s, t)}}{\text{score(s, s)} + \text{score(t, t)}}$$

which is calculated from the individual score for each block, thus giving a similarity value between 0.0 and 1.0. The higher the value, the more similar the two sequences are.

Chen et al. (2004) proposed a metric according to Kolmogorov complexity (Li and Vitányi, 1997) for measuring the amount of shared information between two sequences. They use a compression algorithm involving the LZ data compression scheme (Ziv and Lempel, 1977) to approximate heuristically the Kolmogorov complexity. An information-based sequence distance is then defined as

$$d(x, y) \approx \frac{1 - (\text{Comp}(x) - \text{Comp}(x|y))}{\text{Comp}(xy)}$$

where x and y represent the strings, $d(x, y)$ represents the similarity value, and Comp(x) measures the amount of absolute information the sequence x contains. That is, Comp(x) is length, in number of bits, for the input x after being compressed by LZ data compression. Given another sequence, y, Comp($x|y$) measures the amount of information of x given y for free. By definition, Comp($x|y$) is the length of the shortest program that on input y prints x after being compressed by LZ data compression. The denominator Comp($x|y$) is the total amount of information in the compressed concatenated string xy.