DEVELOPMENT OF A SOFTWARE ON DISTANCE EDUCATION APPLICATIONS FOR COMPILATION AND PLAGIARISM DETECTION OF C PROGRAMMING LANGUAGE ASSIGNMENTS*

Uzaktan Eğitim Uygulamalarinda C Programlama Dili Ödevlerinin Derlenmesi Ve İntihal Tespiti İçin Bir Yazilim Geliştirilmesi

Mümine KAYA
Computer Engineering

Selma Ayşe ÖZEL
Computer Engineering

ABSTRACT

Rapid improvement of information technology has caused the distance learning types to vary and it has enabled the Internet based distance learning to be popular. In this study, a module for plagiarism detection of C programming language assignments on Moodle distance education software for Çukurova University Department of Computer Engineering was developed. This study uses Moodle, which has an utmost position to meet the information needs of students in distance learning programs. In this study, we have developed a new plagiarism detection algorithm for determining similarities between the programming assignments that are submitted by the students. Our source code plagiarism algorithm is in fact a hybrid algorithm which employs Greedy String Tiling algorithm similar to JPlag source code plagiarism software, and also computes word similarity, source line similarity, and comment line similarity similar to CodeMatch plagiarism detection software. The experimental analysis made over the real programming assignments of our Computer Engineering Department students showed that our hybrid system is successful in detecting source code similarities for C programming language assignments.

Key Words: Moodle, Plagiarism, Programming Assignments, Distance Education

ÖZET

Hızla gelişmekte olan bilgi teknolojisi, uzaktan öğretim türlerinin çeşitlenmesine ve Internete dayalı uzaktan öğretimin yaygınlaşmasına neden olmuştur. Bu çalışmada Çukurova Üniversitesi Bilgisayar Mühendisliği Bölümü için Moodle Uzaktan Eğitim yazılımına C programlama dili ödevlerinin intihal tespiti için bir eklenti geliştirildi. Bu çalışmada, öğretim sistemi içinde önemli bir yeri olan Moodle programı kullanılmıştır. Bu çalışmada öğrenciler tarafından gönderilen programlama ödevleri arasındaki benzerliği belirleyen yeni bir intihal tespit algoritması geliştirilmiştir. Önerilen kaynak kod intihal algoritması gerçekte JPlag'a Hırslı Metin Eşleme Algoritmasına benzeyen bir eşleme algoritması ile CodeMatch intihal tespit yazılımında kullanılan Kelime Eşleme, Kaynak Satırı Eşleme, Yorum Satırı Eşleme ve Anlamsal Satır Eşleme Algoritmalarının birleştirildiği bir hibrit algoritmadır. Bilgisayar Mühendisliği Bölümümüzün gerçek programlama ödevleri

* Yüksek Lisans Tezi-MSc. Thesis

üzerinde yapılan deneysel analiz, hibrit algoritmanın C programlama dili ödevleri için kaynak kodu benzerliklerini tespit etmede başarılı olduğunu göstermiştir.
Anahtar Kelimeler: Moodle, İntihal, Programlama Ödevleri, Uzaktan Eğitim

Introduction

Rapidly changing technology and evolving market conditions have changed the education system as well. More educational opportunities have emerged with fewer budgets. As a result of this, distance education tools have increased.

Distance education, which is also called distance learning, has come into being for centuries. Some recent definitions have focused on it as a new development, involving advanced technology. Such definitions have been too restrictive and failed to recognize the actual needs of distance education users. A better definition has been provided by Ian Mugridge (1991), as “it is a form of education in which there is normally a separation between teacher and learner and thus one in which other means - the printed and written word, the telephone, computer conferencing or teleconferencing, for example - are used to bridge the physical gap."

One of the most useful distance education tools is Moodle (http://moodle.org/), which is a free web application that educators can use it to create effective online learning sites, and students can use it to take all the information they need to be successful. It is a learning management system that enables lecturers to provide lecture notes, assignments, quizzes, discussion forums, etc. to students easily and effectively.

On the other hands, some students use Internet and these distance education tools to make plagiarism easily. Plagiarism is described by the Honor Council (http://orgs.odu.edu/hc/pages/What_is_the_Honor_Council.shtml) as "the act of passing off the ideas or writings of another as one's own." Plagiarism in computer code is also frequent since programming courses become popular in most of the undergraduate programs, and detection of source code plagiarism requires different tools than those found in textual document plagiarism. The aim of any plagiarism detection process is to ensure that copied material is detected, and that no student is unfairly accused of copying. Automated Plagiarism Detection Tools help to achieve this aim, but no tool is perfect, and no fully automatic plagiarism detection tools identify all cheaters. Some tools such as Moss (http://theory.stanford.edu/~aiken/moss/), JPlag (https://www.ipd.uni-karlsruhe.de/jplag/) and CodeMatch (http://www.safe-corp.biz/products_codematch. htm) are near perfect to detect source code plagiarism.

The reason for the selection of this study in this thesis is the opinion that distance education is used by many educational institutions more frequently in the near future. In this study, our aim is to design and implement an efficient source code plagiarism detection system that runs on our own server and within our Moodle (Cole, 2005) system since Moss (Aiken, 1998) and JPlag (Prechelt, 2000)

are run on their own remote servers and they are affected by network failures. CodeMatch is a paid software, and we need to pay $400 for 40 license; and it is no possible to directly include JPlag and CodeMatch into our Moodle system. To develop our own system, we examined the existing plagiarism detection software and observed that JPlag and CodeMatch perform better than Moss. So, our new algorithm is a combination of JPlag and CodeMatch such that, we employ Greedy String Tiling from JPlag, and Source Line Matching, Comment Line Matching, Word Matching, and Semantic Sequence Matching algorithms from CodeMatch (Zeidman, 2010). Also, our algorithm can directly process the homeworks submitted to Moodle in our own local server and it is not affected from the network failures.

Material and Method

Material

Moodle (Cole, 2005), Moss (Aiken, 1998), JPlag (Prechelt, 2000) and CodeMatch (Zeidman, 2010) systems are used in this study, because Moodle is a popular, free, and open-source software, and it is used as a distance education tool; Moss, JPlag and CodeMatch are the most widely used plagiarism detection tools, and they are used during the design and analysis of our plagiarism detection algorithm.

Method

The purpose of this study is to design and develop a system that educators can load lecture notes and homework as online over the Internet, and they can also control plagiarism of the submitted C programming language homeworks. Unlike other plagiarism softwares, this new algorithm is going to be used as a module on the Distance Education Application with Moodle (Cole, 2005). Therefore, the plugin which is prepared in this study is targeted to run at our local server. In this study, it is studied a new hybrid algorithm based on Greedy String Tiling Algorithm (GST) (Wise, 1993) which is also used in JPlag, and Source Line Matching (Zeidman, 2004), Comment Line Matching (Zeidman, 2004), Word Matching (Zeidman, 2004) and Semantic Sequence Matching Algorithms (Zeidman, 2004) of CodeMatch (Zeidman, 2010).

Our Hybrid Plagiarism Detection Algorithm

In this study a hybrid algorithm which is a combination of GST Based Source Line Matching, GST Based Comment Line Matching, Longest Common Subsequence (LCS) Based Semantic Sequence Matching, and Hash Based Word Matching Algorithm is proposed. This hybrid algorithm can be easily used to detect plagiarism on the programming assignments that are written in C programming languages. Each component of our hybrid algorithm is explained in more detail in the following subsections.

Preprocessing of Source Codes and Tokenization

All source codes to be compared are parsed firstly. Then whitespace and punctuations are removed from the files. After this preprocessing, remaining language statements are replaced by tokens. The tokens are created according to the syntax rules of the C programming language. The punctuations like '(', ')', '[', ']', '{', '}' are transformed into tokens. But the others are removed from the files. The keywords and identifiers are replaced with the appropriate tokens.

Greedy String Tiling Algorithm

Greedy String Tiling Algorithm (GST) is the basis algorithm in this study. The other comparison algorithms are based on the GST. GST (Wise, 1993) is selected as the basis algorithm in this study because it gives more accurate results and it detects more successful similarities when it is compared with the Winnowing Algorithm (Schleimer et al., 2003) used in Moss.

GST Based Source Line Matching Algorithm

Greedy String Tiling Algorithm (Wise, 1993) is used for making comparison in the Source Line Matching Algorithm (Zeidman, 2004). The reason of integrating GST of JPlag to Source Line Matching Algorithm is that JPlag (Prechelt, 2000) and CodeMatch (Zeidman, 2010) are the most successful ones among the plagiarism detection tools. Our GST based source line matching algorithm (Zeidman, 2004) of CodeMatch works on only source lines of the source codes. For this purpose all comment lines are removed, and the tokenized source codes are compared with GST. In this algorithm minimum match length is taken as a value of 3. The similarity rate is produced and it is then weighted when all algorithms are finished to run.

GST Based Comment Line Matching Algorithm

Greedy String Tiling Algorithm (Wise, 1993) is used in the Comment Line Matching Algorithm (Zeidman, 2004). This algorithm compares only comment lines. The source code lines, punctuation and whitespace characters are removed from the file in the preprocessing step. In this algorithm minimum match length is taken as a value of 1. This algorithm produces a similarity score and it is then weighted when all algorithms are finished to run.

LCS Based Semantic Sequence Matching Algorithm

The Longest Common Subsequence Algorithm is used to find the longest sequences that are subsequences of two or more sequences. In this study Semantic Sequence Matching Algorithm (Zeidman, 2004) is used with Longest Common Subsequence Algorithm. It produces a similarity score and it is then weighted when all algorithms are finished to run.

Hash Based Word Matching Algorithm

In this study hash data structure of Perl scripting language is used for Word Matching Algorithm (Zeidman, 2004). Because in this way, accessing the searched data is more easy and faster. It reads all words, which are identifiers, from the first source code file to the hash. Then it reads all words, which consists of identifiers, in the second source code file and each identifier in the second file is searched from the hash. If the identifier is found in the hash, then a match occurs. The number of identifiers in each file is calculated. The similarity rate is then computed as the percentage of the matched identifiers between the two files. The similarity value is then weighted when all other similarity algorithms are finished to run.

Computation of the Total Match Score

Each score of each individual algorithm is weighted and the Total Match Score is calculated according to Equation 1, where WMA, SSMA, GBSMA, and GBCMA are the similarity rates computed by the Word Matching Algorithm, Semantic Sequence Matching Algorithm, GST Based Source Line Matching Algorithm, and GST Based Comment Line Matching Algorithm, respectively. α_1, α_2, α_3 and α_4 are the weights assigned to the score of each algorithm. These weights are adjusted experimentally to give the optimal results. After that each score is weighted to form the Total Match Score.

$$\text{TotalMatchScore} = \alpha_1.\text{WMA}+\alpha_2.\text{SSMA}+\alpha_3.\text{GBSMA}+\alpha_4.\text{GBCMA} \quad (3.2)$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad (3.3)$$

Research and Discussion

In this section experiments performed and their results are presented. Perl scripting language (http://www.perl.org/) was used for implementing our hybrid algorithm. The proposed method was compared with Moss and JPlag in three scenarios.

The first scenario contains 10 short assignments which are completely the same with each other. It has 22 lines and it has no comment lines.

The assignments are compared with Moss firstly. The similarity scores obtained by Moss for this scenario are presented in Table 1. F1 represents the assignment of the first user and F10 represents the assignment of the last user.

As shown in Table 1, although the source codes are the same files, Moss computed their similarities as 95%.

Table 1. Moss Results as a Table For Scenario 1

MOSS	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
F1	-	95%	95%	95%	95%	95%	95%	95%	95%	95%
F2	-	-	95%	95%	95%	95%	95%	95%	95%	95%
F3	-	-	-	95%	95%	95%	95%	95%	95%	95%
F4	-	-	-	-	95%	95%	95%	95%	95%	95%
F5	-	-	-	-	-	95%	95%	95%	95%	95%
F6	-	-	-	-	-	-	95%	95%	95%	95%
F7	-	-	-	-	-	-	-	95%	95%	95%
F8	-	-	-	-	-	-	-	-	95%	95%
F9	-	-	-	-	-	-	-	-	-	95%
F10	-	-	-	-	-	-	-	-	-	-

The assignments are then compared with JPlag. Java Web Start Application must be installed on a computer to run JPlag. As shown in Table 2, JPlag can find the similarity rates between the files as 100%.

Table 2. JPlag Results as a Table For Scenario 1

JPLAG	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
F1	-	100%	100%	100%	100%	100%	100%	100%	100%	100%
F2	-	-	100%	100%	100%	100%	100%	100%	100%	100%
F3	-	-	-	100%	100%	100%	100%	100%	100%	100%
F4	-	-	-	-	100%	100%	100%	100%	100%	100%
F5	-	-	-	-	-	100%	100%	100%	100%	100%
F6	-	-	-	-	-	-	100%	100%	100%	100%
F7	-	-	-	-	-	-	-	100%	100%	100%
F8	-	-	-	-	-	-	-	-	100%	100%
F9	-	-	-	-	-	-	-	-	-	100%
F10	-	-	-	-	-	-	-	-	-	-

Finally the assignments are compared with our hybrid algorithm which runs under Moodle. Table 3 summarizes the results of our hybrid algorithm.

Table 3. Hybrid Results as a Table For Scenario 1

HYBRID	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
F1	-	100%	100%	100%	100%	100%	100%	100%	100%	100%
F2	-	-	100%	100%	100%	100%	100%	100%	100%	100%
F3	-	-	-	100%	100%	100%	100%	100%	100%	100%
F4	-	-	-	-	100%	100%	100%	100%	100%	100%
F5	-	-	-	-	-	100%	100%	100%	100%	100%
F6	-	-	-	-	-	-	100%	100%	100%	100%
F7	-	-	-	-	-	-	-	100%	100%	100%
F8	-	-	-	-	-	-	-	-	100%	100%
F9	-	-	-	-	-	-	-	-	-	100%
F10	-	-	-	-	-	-	-	-	-	-

According to Table 1, 2 and 3, for completely the same source files JPlag and our hybrid algorithm compute similarity rate as 100%, however Moss compute similarity rate as 95% although all the source codes are the same. For this scenario, our hybrid algorithm and JPlag are more successful than Moss.

All the algorithms (i.e., Moss, JPlag, and Hybrid) are compared with respect to processing time for the first scenario and results are presented in Table 4. As shown in the table, for this scenario our hybrid algorithm is the fastest.

Table 4. The Comparison of The Processing Time For Scenario 1

MOSS	JPLAG	HYBRID
3 seconds	4 seconds	1 second

The second scenario is that 10 long assignments which are completely the same with each other are compared with each other. The source file has 285 lines and it has comment lines.

The assignments are compared with Moss primarily. Although the source codes are the same files, Moss computed their similarities as 99%

The assignments are then compared with JPlag. For completely the same source files, JPlag can find the similarity rates between these source files as 100%.

Finally the assignments are compared with our hybrid algorithm which runs under Moodle. Similarity rate computed by our hybrid algorithm is 100% for the same source files.

According to the results, JPlag and our hybrid algorithm compute similarity rate as 100% for the same source codes. For this scenario, similarity rate computed by Moss is increased to 99%. For this scenario, our hybrid algorithm and JPlag are able to detect 100% similarity. As the length of the source files increases, similarity score of Moss improves.

All assignments are compared with regard to processing time of Moss, JPlag and this Hybrid Algorithm in a Table 5. According to Table 5, Moss is the fastest and JPlag is slower than Moss. Our hybrid algorithm is the slowest among Moss and JPlag for this scenario.

Table 5. Compare The Processing Time For Scenario 2

MOSS	JPLAG	HYBRID
6 seconds	20 seconds	113 seconds

The third scenario is that 25 long source codes which are different submissions of the same assignment compared with each other. The source codes are compared with Moss, JPlag and The Hybrid Algorithm. Source codes have between 71 and 775 lines and they have comment lines.

Table 6. The Comparison of The Similarity Rate For Scenario 3

FILE1	FILE2	MOSS	JPLAG	HYBRID
odev4_2008638001.c	odev4_2008638012.c	15 %	39.6 %	55.98 %
odev4_2009638006.c	odev4_2009638010.c	9 %	28.1 %	45.33 %
odev4_2009638019.c	odev4_2009638060.c	98 %	99.0 %	99.20 %
odev4_2009638027.c	odev4_2009638045.c	0 %	38.7 %	45.55 %
odev4_2009638045.c	odev4_2009639050.c	79.0 %	75.7 %	82.54 %
odev4_2009638045.c	odev4_2009638048.c	99.0 %	100.00%	100.00%
odev4_2009638048.c	odev4_2009639050.c	79.0 %	75.7 %	82.54 %
odev4_2009638053.c	odev4_2009638054.c	9 %	37.4 %	49.86 %
odev4_2009638060.c	odev4_2009639400.c	0 %	60.4 %	74.67 %
odev4_2009639012.c	odev4_2009639039.c	97 %	100.00%	100.00%

The all assignments are compared with regard to processing time of Moss, JPlag and Hybrid Algorithm in Table 7. Although hybrid algorithm is more successful in detecting plagiarism, it needs more time for processing with respect to Moss and JPlag.

According to Table 7, our hybrid algorithm is slower than Moss and JPlag for this scenario.

Table 7. The Comparison of The Processing Time For Scenario 3

MOSS	JPLAG	HYBRID
30 seconds	70 seconds	4134 seconds

Conclusion

In this thesis, a hybrid string matching algorithm which is Greedy String Tiling, Source Line Matching, Comment Line Matching, Word Matching and Semantic Sequence Matching Based Plagiarism Detection Algorithm was developed the source codes written in C programming language. In our system, all these matching algorithms produced a match score. Each match score of each individual algorithm was weighted and a Total Match Score was calculated to give the optimal results.

Experimental evaluation shows that, the proposed algorithm is effective on detecting similarities. It runs fast for short source codes; however it needs more time for long source codes with respect to other plagiarism detection software.

As future work, we plan to decrease the processing time of our algorithm, so we can use it effectively. In future, we also plan to include support for other programming languages into our hybrid algorithm.

References

AIKEN A., 1998. MOSS (Measure Of Software Similarity) Plagiarism Detection System, http://www.cs.berkeley.edu/?moss/.

CODEMATCH, 2011. http://www.safe-corp.biz/products_codematch.htm.

COLE, J., 2005. Using Moodle: Teaching With The Popular Open Source Course Management System, O'Reilly Community Press, Sebastopol, CA.

HONOR COUNCIL, 2011. http://orgs.odu.edu/hc/pages/What_is_the_Honor_ Council.shtml.

JPLAG, 2011. https://www.ipd.uni-karlsruhe.de/jplag/.

MOODLE, 2011. http://moodle.org/.

MOSS, 2011. http://theory.stanford.edu/~aiken/moss/.

MUGRIDGE, I., 1991. Distance Education And The Teaching Of Science, Impact of Science on Society 41 4, 313-320.

PRECHELT L., MALPOHL G., and PHILIPPSEN M., 2000, JPlag: Finding Plagiarisms Among A Set Of Programs. Technical Report 20001, Fakultät für Informatik, Universität Karlsruhe, Germany, March 2000. ftp.ira.uka.de.

SCHLEIMER, S., WILKERSON, D. S. and AIKEN, A., 2003. Winnowing: Local Algorithms For Document Fingerprinting, In Proceedings ACM SIGMOD International Conference On Management of Data, pp. 76-85.

WISE, M. J., 1993. String Similarity via Greedy String Tiling and Running Karp-Rabin Matching, ftp://ftp.cs.su.oz.au/michaelw/doc/RKR_GST.ps.

ZEIDMAN, B., 2004. Detecting Source-Code Plagiarism - Tools and Algorithms for Finding Plagiarism in Source Code, Dr Dobb's Journal, http://drdobbs.com/architecture-and-design/184405734.

ZEIDMAN, R. M., 2010, Detecting Plagiarism In Computer Source Code, United States Patent Application 20100325614 A1.