2014

Issues Related to the Detection of Source Code Plagiarism in Students Assignments

Izzat M. Alsmadi
Texas A&M University-San Antonio, ialsmadi@tamusa.edu

I. AlHami

S. Kazakzeh

International Journal of Software Engineering and Its Applications
Vol.8, No.4 (2014), pp.23-34
http://dx.doi.org/10.14257/ijseia.2014.8.4.03

Issues Related to the Detection of Source Code Plagiarism in Students Assignments

Izzat Alsmadi[1], Ikdam AlHami[2] and Saif Kazakzeh[3]

[1] *Information systems Department, Prince Sultan University*
[2,3] *Computer science Department, Yarmouk University*
[1] *ialsmadi@cis.psu.edu.sa,* [2] *ikdam@yahoo.com,* [3] *xsaifahmadx@gmail.com*

Abstract

Detecting similarity or plagiarism in the academic research publications, source code, etc. has been a long time complex and time consuming task. Several algorithms, tools and websites exist that try to find plagiarism or possible plagiarism in those human creative products. In this paper we used source code plagiarism detection tools to assess the level of plagiarism in source codes. We also investigated issues related to accuracy and challenges in detecting possible plagiarism in students' assignments. In a second study, we evaluated some tools against detecting possible plagiarism in research papers. Results showed that such process or decision is not binary to make and that subjectivity is high. In addition, there is a need to tune plagiarism detection tools to give criticality or weights by users of those tools to categorize and classify different levels of seriousness for committing plagiarism.

Keywords: *Plagiarism, Code similarity, Documents similarity, string search, information retrieval, and search engines.*

1. Introduction

In the academic field, one of the major serious problems is the plagiarism problem. There are two major areas of possible plagiarism in the academia. Those include plagiarism in research papers, projects and publications. It also includes plagiarism that is especially applicable for students in the computer and information technology majors. This is the plagiarism in writing code or programs assigned by their instructors. Further, code plagiarism may take several possible forms. In some cases, students in the same class may copy assignments from each other. They may also get their code assignment from external public resources, especially the Internet. In some places, local companies may offer helping students partially or completely in those code projects. The Internet also includes several websites in which students can submit their code assignments and get help from experts through the web. In some cases, this may be offered for financial compensations, or it can be offered as part of blogs or websites of experts for free. This link:
(http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/onlinesites.html) that is updated by University of Ulster contains a list of several websites that help students (or any person or business for that matter) in their code assignments.

Teaching some computer major courses without tasks, assignments and experiments that include programming is ineffective. On the other hand, instructors struggle to make sure that their students actually performed the tasks themselves without a significant or complete help from others. The Internet and the availability of many websites that can offer help makes it harder for instructors to find possible plagiarism as they will not only look for possible

ISSN: 1738-9984 IJSEIA

plagiarism among students in their course; they have to search through a vast number of websites, blogs, posts, etc. It may be argued that instructors can solve this through asking for new or different tasks all the time. This can be impossible and time consuming for instructors in courses that are time consuming also in grading, looking for possible plagiarism, etc. especially when the number of students in such classes is large.

To help instructors in the speed and the accuracy of detecting possible plagiarism, several tools and websites are available: free, open source and commercial. In the following section, we will describe some of those tools.

1.1. Tools and Techniques to Detect Code Similarity

There are several examples of source code plagiarism tools. Focus in this section will be on: JPlag, SIM, and MOSS as a sample.

- JPlag

While it is not the first source code web-based plagiarism detection tool, nonetheless, evaluations of the tool showed that it is reliable, available for free and easy to use in comparison with many other similar tools (Prechelt *et al.*, 2002 [1]). The paper of (Faidhi, and Robinson 1987 [2]) discussed an earlier code plagiarism tool where the tool includes a large set of metrics to compare among the different codes to judge possible plagiarism.

YAP (Yet another Plague) tool of (Wise 1992 [3]) discussed also a source code plagiarism tool. Wise released several enhanced versions of the tool later on. YAP itself was an enhancement of an earlier tool called (Plague). User of YAP is allowed to set the cut off percentage to consider the occurrence of plagiarism in the code or not.

- SIM

This is a tool that is developed to detect code as well as text possible plagiarism, or even DNA string comparison (Gitchell and Tran 1999 [4]). The tool is original developed to compare C program codes. A similarity score algorithm is developed with a value between 0 and 1 based on the level of similarity between the subject codes.

- MOSS

This is also another popular free code plagiarism tool. It supports different operating systems. The tool divided the code into several finger prints and matching or similarity is evaluated based on the number of similar finger prints between the evaluated codes.

1.2. Techniques to Detect Documents Similarity

In this area, there are many methods to judge similarity between documents. A brute force approach will compare the subject document with investigated documents word by word. However, in most cases, such approach is time and resources' consuming. In addition, such approach can be easily fooled through editing a small number of words in the document. A more effective approach depends or is based on metrics related to the documents such as the number of statements, paragraphs, punctuation, *etc*. (Grier 1981 [5], Faidhi, and Robinson 1987 [2]). A similarity index is calculated to measure the amount of similarity between documents based on those metrics. Comparing the approach of taking the document word by word in comparison to statement or paragraph by graph for example can have several contradicting tradeoffs. On one side, word by word comparison can minimize the effect of changing one or a small number of words relative to the total document. However, this can be time consuming and word to word document similarity may not necessarily means possible

plagiarism especially if the algorithm did not take the position of the words into consideration. Documents' similarity can be classified in different categories. In one classification, they can be classified into: word based, keyword based, sentence based, *etc*. Sentence or paragraph by paragraph approach is also affected by several variances such as the difference in size between the compared documents and the amount of words edited in those statements or paragraphs.

Hashing algorithms are also used to measure documents similarity. Hashing algorithms are used originally in security to verify the integrity of an investigated disk drive and protected it from being tampered. Hashing can be calculated for a word, a paragraph, a page, or a whole document.

N-gram and Latent Semantic Analysis (LSA) approaches are also different algorithms used in documents' similarity. The main drive behind using N-gram in evaluating similarity between documents is that similar words will have a high percentage of N-grams in common. In most experiments, n is selected to be two or three. For example, using n-gram for the word "software" and n to be 3, will give the following outputs: ##S, #SO, SOF, OFT, FTW, TWA, WAR, SRE, RE#, and E## where # denotes a padding space. The number of possible bigrams is given by the equation: n+m-1, where n is the number of possible characters in the word or the string and m is the possible grams. In the previous example, n is 8, and m is 3 and hence the number of bigrams is 10. Several text similarity applications such as: information retrieval, natural language processing, OCR, spell checking, etc use n-gram in their text similarity decisions.

1.3. Semantic Similarity

Measuring semantic is usually a harder task in comparison with measuring words' similarity. In documents, semantic similarity between the two documents can be measured based on a similarity index that measures the number of similar words based on several possible algorithms. Statistical means such as vector space models can be also used to measure the amount of correlation between the two subject documents. A topological similarity method is usually used to measure similarities between ontological concepts. Examples of such methods include: edge-based, node-based, pair-wise, and group-wise techniques. In terms of tools, there are some popular tools that are experimented for semantic similarity. Examples of such tools include: Wordnet, MSR, UMLS, SenseBot, SenseLearner, GWSD, and FrameNet. Wordnet uses an extensive word-definition library or dictionary that can be queried for each word in the subject document.

2. Literature Review

2.1. Code Plagiarism

In this section, we will describe some papers related to plagiarism in general. Then in the second section, we will describe some of the papers dedicated to code plagiarism evaluation.

Manber presented approximate index concept to measure similarity between strings in different documents (Manber 1994 [6]). A tool called "Sif" is developed to find similar files in a large file system. He proposed the concept of approximate index to measure the similarity of character strings between documents, which was adopted later by many similar systems.

(Manber 1994 [6]) described using a finger print (or what they called anchors) and a fixed number of characters as a baseline to search for plagiarism. In a similar approach and rather than considering a fixed number of characters where changing one character may affect the whole comparison, we decided to select 4 words as the baseline. An initial method is

developed to calculate the most frequent words in a paper and use them as an anchor. This is of course after removing all generic words, prepositions, and any other words that are expected to be seen in any paper (*i.e.*, abstract, keywords, "this paper", *etc*). For each occurrence of those frequent words, the algorithm will take 4 words starting from frequent words, and then look in all subject documents for possible matches.

We compared using the most frequent words as anchors in comparison to all documents words. Comparison will be based on two criteria: performance and plagiarism detection. If sufficient number of baselines (*i.e.*, 4-words statements are common to two files (under comparison) then this is a good enough evidence that the two files are similar in some way.

The tool we developed in this paper uses several different search algorithms. The first one searches for possible similar documents for the subject document through a directory of files. The other algorithm searches for similar documents through the Internet. Calculating similarity between documents does not require in many cases similarity in cosmetic attributes such as the file type, size, number of words, etc. He defined a checksum algorithm called "fingerprint" that is based on defining keywords in each document and parse a certain amount of characters starting from those keywords to calculate similarity. In those checksum, anchor words are used from which a certain number of characters is selected and compared among documents. Anchors are created through analyzing text from many different files and selecting a fixed set of representative strings. In somewhat similar approach, we used the most frequent words in the subject word to be our anchors from which the algorithm will start looking for possible plagiarism or sentences' match.

Some papers tried to tackle the performance problem of finding plagiarism in documents through using indexing (Mozgovoy *et al.*, 2005 [7]). Such concept is utilized also in search engines for fast document retrieval.

Detecting possible plagiarism in source code is another relevant subject to this paper. In principle, searching for similarities between two code projects is similar to that of documents. However, some cosmetic changes to a source code (*e.g.*, changing all variables, methods, classes' *etc*., names) can make the new code look different for a code plagiarism tool while in reality it is similar or identical. Based on this assumption (Baker 1993 [8]) defined two source codes to be similar if one can be obtained from the other by changing parameter, method, attributes, or classes' names. He presented several algorithms to identify similar source codes.

We will be contrasting our findings with those obtained using the shingle and finger print techniques (Manber 1994 [6], and Broder *et al.*, 1997 [9]). This technique depends on reducing each document to a series of numeric codes, such as hash codes, based on sequences of words. In the original paper, the authors suggested making each hash code of a group of 10 adjacent words, and moving the window by one word to create the next hash code. They then eliminate duplicates and, to reduce the number of values, save only those divisible by 25. If this is still too many, they save only the 400 smallest values. The advantage of using shingles to compare documents is that a simple set membership between two tables of integers can be computed very rapidly. Documents that match in all shingles are assumed to be identical and those that match nearly all shingles are closely related.

For code plagiarism, several papers are available focusing in this issue. Some papers discuss the development and evaluation of code plagiarism tools such as those mentioned earlier. Other papers focus on the experience of dealing with students' code plagiarism evaluation.

Several papers tried to compare between different source code analyses tools (*e.g.*, Jun-Peng et al 2003 [10], Maurer *et al.*, 2006 [11], Kustanto and Liem 2009 [12], Hage *et al.*, 2010 [13], *etc.*). There are several popular tools such as those described earlier that were the focus of such surveys or comparisons. There are two major criteria upon which such tools are

compared. Those are accuracy and speed or performance. In terms of accuracy, metrics are used to measure the ability of those tools to successfully or correctly detect the occurrence of plagiarism. In such scenarios, failures can occur when such tools assume plagiarism while it's not, or the opposite. Challenges arise in cases where it is difficult to judge plagiarism occurrence (*e.g.*, semantic plagiarism). In terms of performance, it is important for such tools to complete the process in a timely manner. Testing a code project against several other projects, line by line can take a significantly long time.

2.2. Plagiarism in Research Publications

A lot of works are conducted on the plagiarism process, tools, evaluations, *etc*. We will list only a sample of those in this section.

El Tahir Ali *et al.*, 2011 [14] presented a survey of the most important plagiarism detection methods. They classified the detection tools based on the used methods to four classes: natural language text detection, index structure, external and cluster-based plagiarism detection tools. Natural language text copy detection is used for years and includes three methods. First is the grammar-based method which is appropriate for catching the text plagiarized without modifications. Second method is the semantic-based method which can work properly for non-partial plagiarized text as it is based on the vector space model. Grammar-semantic hybrid is the third method which is suitable with partial plagiarized text that also includes modifications. Ferret is an example on the use of a specific index structure that is based on the words trigrams. The external plagiarism detection method uses external corpus collections in order to compare any given document with it. The last effective method is clustering, which is used widely for text summarization and in reducing the search time. Fingerprint-based plagiarism is the main method that relies on clustering.

Most of the proposed plagiarism detection tools are not specific for a particular language despite the fact that the majority are developed for English language in the first place. Alzahrani *et al.*, 2009 have produced an Arabic plagiarized detection (APD) tool especially for working with Arabic language. Their tool detects and highlights the plagiarized text, and it was experimented and integrated within an e-learning system. Additionally, another Arabic plagiarism detection tool (APlag) was presented by Menai and Bagais 2011. APlag depends on fingerprints methods, and other characteristics of Arabic language. It has been experimented and the results present a better effectiveness compared to APD.

A recently published study by Kakkonen and Myller 2012 claimed that their novel plagiarism detection tool (AntiPlag) has performed better (with 95% accuracy) compared to four of the well-known commercial tools (*i.e.*, Turnitin, Eve2, SafeAssignment and Plagiarism-Finder). AntiPlag works with both local collections and web-based plagiarism detection. In general, there are many factors that should be considered when evaluating a plagiarism detection tools such as: accuracy, performance, and false alarm reduction, *etc*.

Another direction of using plagiarism detection tools is presented by Graven and MacKinnon 2007. Authors have studied the flexibility and richness of two advanced plagiarism detection tools (Turnitin and VALT/VAST). They wanted to address whether those tools provide a good enough detection to detect commonalities between texts that are not actually plagiarized but yet should be similar. Their evaluation depends on the idea of using such tools in an automated assessment process within a virtual learning environment (VLE). In the project, a student should create a narrative in order to pass to next levels in the learning process. Narratives are about conceptual elements that are defined in the project. The next step is decided, according to a predefined narrative sample as a solution, and depending on the plagiarism detection tools. Similarities should be detected if the student wrote a close solution to the predefined one. In this way an automatic assessment can be achieved to some

extent by using those tools. In some cases, there were a number of strong similarities in form of semantic or separated words. These results provide a proof that those tools still not useful or immature for developing automated assessment techniques or evaluation. Authors raised questions on how much can those tools detect smart plagiarism attempts, not only directly copying a text.

3. Experiments and Analysis

In an earlier paper (Alhami and Alsmadi 2011), we described our implementation of a tool for automatic grading for code homework. The tool is developed based on concept extraction to automatically grade each question in comparison with a typical answer for that question. Rather than looking for a specific answer, the typical answer, which is the baseline for each question that the grading process depends on, include keywords that are expected to exist in the answer.

This includes using JPlag code plagiarism detection tool to evaluate possible code plagiarism among students' assignments gathered from actual submitted home-works. In Plagiarism, the divided the levels of plagiarism into several levels based on the percentage of similarity between the evaluated codes.

Following is a description of the evaluation experiment along with results analysis. Several code assignments are submitted from students. Students were from 3 different sections.

- Task 1: First assignment for the first student section. Five students have submitted the assignments. Results showed that there is no clear plagiarism among student assignments and the percentage of similarity among all assignments in this section is limited to between 0% - 10%.
- Task 2: First assignment for the second student section. Six students have submitted the assignment. Two cases of plagiarism in the level: 40-50%, 17 cases between 10-20 % and the rest are in the range of less than 10%. Table 1 shows a summary of experiment for students' assignments possible plagiarism in this section. The table shows the similarity matrix among the different assignments that have a significant level of similarity.
- Task 3: First assignment for the third student section. Two students have submitted the assignment. Ranges of plagiarism are between 30 % and less. Table 2 shows a summary of this task results.
- Task 4: Second assignment for the first student section. Six students have submitted the assignment. Plagiarism levels vary between 60 % and below. This is an average level of plagiarism where it can indicate that students are actually copying from each other or from the same source. Table 3 shows a summary of those results.
- Task 5: Second assignment for the second student section. Eight students have submitted this assignment. In this case, serious plagiarism occurred with levels higher than 60 % (*i.e.*, 64.8 and 99.7 %). Summary of results is shown in Table 4. The first row represents a solid case of plagiarism between students (2009901087 and 2008901120).
- Task 6: Second assignment for the section three. Seven students have submitted this assignment. So far, this is the most serious case of plagiarism with several almost complete cases of plagiarism. Further, results showed that in some cases more than two students are copying from other. Results are shown in Table 5.
- Task 7: Third assignment for the first student section. Six students submitted the assignment. Table 6 shows the results with a medium level of plagiarism.

- Task 8: Third assignment for the second student section. Five students submitted this assignment in section 2. Results indicate a significant level of plagiarism among all students. This is somewhat a unique case in comparison to all previous assignments or cases. Table 7 summarizes the results for Task 8.
- Task 9: Third assignment for the third student section. Twelve students have submitted the assignment. Only 7 of those are displayed which showed possible plagiarism. Results in this section showed a significant, even complete, levels of plagiarism where some students are exactly using the code of others representing a solid case of plagiarism. Table 8 shows a summary of Task 9 results.
- Task 10: Fourth assignment for the first student section. Eight students submitted the assignments and results of five of them are showed for significant plagiarism. Results showed significant levels of plagiarism among student codes. Table 9 shows a summary of the results of Task 10.
- Task 11: Fourth assignment for the second student section. Only assignments of two students are evaluated. Table 10 shows a summary of the results.
- Task 12: Fifth assignment for the first student section. Six of ten submitted assignments are evaluated for possible plagiarism. There is a significant level of plagiarism in some of those assignments in comparison to the others. Table 11 shows a summary of the results.

Table 1. Assignment 1. Section 2: Results Summary

2008802022	->	2007901168 (45.0%)	2008902010 (13.2%)	2008802013 (12.5%)	2008801058 (11.8%)	2008902128 (11.3%)	2008901130 (10.7%)
2008902010	->	2008901130 (42.3%)	2008902190 (17.8%)	2008802013 (16.2%)	2008901123 (14.4%)	2007901168 (10.3%)	
2010902016	->	2008901130 (18.1%)	2008801058 (15.8%)	2008902128 (11.8%)	2008902190 (11.0%)		
2007901168	->	2008801058 (18.0%)	2008902128 (10.1%)	2008901130 (9.6%)			
2008902128	->	2008801058 (17.8%)					
2008902190	->	2008901130 (13.5%)					

Table 2. Assignment 1. Section 3: Results Summary

200990208	->	2010801051 (26.3%)	2008902008 (24.6%)	2008902089 (12.7%)
2010801051	->	2008902008 (22.9%)	2008902089 (11.7%)	

Table 3. Assignment 2. Section 1: Results Summary

2009901084	->	2008801045 (57.3%)	2009902129 (33.3%)	2009902139 (21.5%)	2008902049 (17.8%)	2009901106 (16.1%)		
2009902139	->	2009901127 (38.7%)	2009901006 (23.5%)	2009901106 (23.4%)	2008801045 (17.4%)	2008902047 (16.9%)	2008902049 (16.4%)	2009901007 (12.2%)
2009902129	->	2009902016 (34.8%)	2008801045 (30.8%)					
2009902142	->	2009901106 (29.2%)	2008902049 (22.6%)	2009902154 (17.9%)				
2009901106	->	2009901007 (16.5%)	2008902049 (13.8%)					
2009902154	->	2009902016 (15.7%)						

Table 4. Assignment 2. Section 2: Results Summary

2009901087	->	2008901120 (99.7%)	2008901123 (31.8%)	2008901067 (16.4%)	2009902104 (16.3%)
2008901067	->	2008802013 (64.8%)	2008902128 (47.8%)	2008901120 (16.5%)	2009901126 (13.6%)
2008902128	->	2008802013 (40.7%)	2009901126 (32.6%)	2008901123 (29.4%)	
2008901123	->	2008901120 (31.8%)	2009901126 (17.8%)		
2008902015	->	2008902110 (24.5%)	2008801035 (14.0%)		
2009901126	->	2008802013 (16.6%)	2008801035 (10.1%)		
2009902104	->	2008901120 (16.4%)	2008801035 (11.5%)		
2008902110	->	2008801035 (13.1%)			

Table 5. Assignment 2. Section 3: Results Summary

data 2008902035 (100.0%)	2008902093 (98.2%)	2008902070 (42.0%)	2008902089 (15.4%)	2008901151 (12.8%)
2009902027 (100.0%)	2008902070 (11.2%)	2010801051 (11.0%)		
2008902093 (98.2%)	2008902070 (42.0%)	2008902089 (15.4%)	2008901151 (12.8%)	
2008902093 (41.7%)	2008902089 (16.5%)	2008901151 (13.7%)	2009902027 (11.2%)	
2008902121 (23.4%)	2008902093 (12.7%)			
2008902089 (15.4%)				
2009902027 (11.0%)				

Table 6. Assignment 3. Section 1: Results Summary

2009901084	->	2008801045 (57.3%)	2009902129 (33.3%)	2009902139 (21.5%)	2008902049 (17.8%)	2009901106 (16.1%)		
2009902139	->	2009901127 (38.7%)	2009901006 (23.5%)	2009901106 (23.4%)	2008801045 (17.4%)	2008902047 (16.9%)	2008902049 (16.4%)	2009901007 (12.2%)
2009902129	->	2009902016 (34.8%)	2008801045 (30.8%)					
2009902142	->	2009901106 (29.2%)	2008902049 (22.6%)	2009902154 (17.9%)				
2009901106	->	2009901007 (16.5%)	2008902049 (13.8%)					
2009902154	->	2009902016 (15.7%)						

Table 7. Assignment 3. Section 2: Results Summary

hw2008902023.cpp (86.2%)	HW 2008902117.cpp (77.3%)	H.W2008902187.cpp (67.1%)	hw2008901040.cpp (51.0%)	HW 2008901034.cpp (50.0%)
HW 2008902117.cpp (72.8%)	hw2008902023.cpp (69.6%)	hw2008901040.cpp (59.3%)	HW 2008901034.cpp (58.0%)	
hw2008902023.cpp (63.4%)	hw2008901040.cpp (53.9%)	HW 2008901034.cpp (52.7%)		
HW 2008901034.cpp (53.1%)	hw2008902023.cpp (52.7%)			
HW 2008901034.cpp (51.7%)				

Table 8. Assignment 3. Section 3: Results Summary

s(12).cpp	s (11).cpp (100.0%)	s (6).cpp (98.2%)	s (4).cpp (42.0%)	s (9).cpp (12.8%)
s (5).cpp	s (10).cpp (100.0%)	s (8).cpp (11.5%)	s (4).cpp (11.2%)	s (1).cpp (11.0%)
s (11).cpp	s (6).cpp (98.2%)	s (4).cpp (42.0%)	s (9).cpp (12.8%)	
s (4).cpp	s (6).cpp (41.7%)	s (9).cpp (13.7%)	s (10).cpp (11.2%)	
s (9).cpp	s (2).cpp (23.4%)	s (6).cpp (12.7%)		
s (8).cpp	s (2).cpp (11.6%)	s (10).cpp (11.5%)	s (1).cpp (11.3%)	
s (10).cpp	s (1).cpp (11.0%)			

Table 9. Assignment 4. Section 1: Results Summary

s (4).cpp	s (3).cpp (86.2%)	s (6).cpp (77.3%)	s(7).cpp (67.1%)	s (1).cpp (51.0%)	s (5).cpp (50.0%)
s(7).cpp	s (6).cpp (72.8%)	s (3).cpp (69.6%)	s (1).cpp (59.3%)	s (5).cpp (58.0%)	
s (6).cpp	s (3).cpp (63.4%)	s (1).cpp (53.9%)	s (5).cpp (52.7%)		
s (1).cpp	s (5).cpp (53.1%)	s (3).cpp (52.7%)			
s (5).cpp	s (3).cpp (51.7%)				

Table 10. Assignment 4. Section 2: Results Summary

s (2).cpp	s (1).cpp (43.0%)	s(4).cpp (22.7%)
s(4).cpp	s (1).cpp (18.7%)	

Table 11. Assignment 4. Section 3: Results Summary

s(7)	s(10) 82	s(6) 32	s(4) 22	s(2) 21	s(10) 20
s(9)	s(1) 68	s(2) 18	s(10) 17		
s(10)	s(6) 36	s(4) 24	s(2) 24	s (1) 22	
s (2)	s(6) 32	s(8) 22	s(3) 15		
s (5)	s(4) 26	s(3) 17			
s (8)	s(6) 18				

Upon manual review of the students assignments we found out that plagiarism detected by the tool can be classified under the following categories:

1. In some cases, the plagiarism detection is (false alarm) where the tool by mistake decided that some similar use of variable or method declarations is a possible plagiarism. We know that in programming or code, there are some parts that can be identical between all assignments and those are part of the programming language built-in names that will be the same in all tasks if they are used.
2. On the other side, manual detection of students' code assignments showed that some students are clever in a since that they can mislead code plagiarism tools. This is as they change variable and method names while in reality the majority of the code among the different assignments is the same. However, such semantic type of plagiarism is still a challenge for all types of plagiarism detection tools.
3. On the third level of manual code plagiarism observation, our observations showed that code plagiarism tools that can be a useful effective tool for instructors for initial location of possible high plagiarism levels. While some percentage of error in plagiarism detection can be noticed, on the other hand, they are able to give initial indicators of plagiarism especially in cases where such plagiarism is high and obvious. Such task can be tedious and time consuming to perform manually.

4. Literature Evaluating Plagiarism in Research Papers

We have conducted a comparative study as a preliminary experiment. The study evaluated three plagiarism detection tools (*Plagiarisma*, *Dustball*, and *DupliCkecker*) that are free and web-based, Table 12. Based on a case study assembled for this purpose, tools are evaluated and compared mainly in their ability to predict plagiarism occurrences and reducing false alarms. Simple tests are conducted by preparing two different documents as test cases for the tools. The tests revealed that *Plagiarisma* was the most reliable and accurate tool for detection with issues only with performance of efficiency. *Dustball* and *DupliChecker*, ranked second and third, respectively, and both of them have significant problems related to the reliability of detecting or missing plagiarism cases.

Table 12. Plagiarism Detection Tools Characteristics and Features

	Plagiarisma	***Dustball***	***DupliChecker***
Website	http://www.plagiarisma.net	http://www.dustball.com/cs/plagiarism.checker/	http://www.duplichecker.com
Provide a Premium Membership	Yes	Yes	No
Provide Desktop Software	Yes	No	No
Need Registration	Yes	No	No. But, one can register for free to do unlimited searches per day
Ability To Upload Files	Yes, for free and premium users	Yes, only for premium users	Yes
Possibility to create PDF reports	Yes	No	No
Search Engines	Google, Babylon, Yahoo	Google	Google, Yahoo, MSN [7]
Restrictions	Characters per query; max 5000 unless user is registered (for free). Some options however are only available for premium users with paid registrations.	A delay to start the detection process for non-premium users	Max 2000 words per search, non-registered users can do 3 searches per day

5. Conclusion

In this paper, we evaluated the use of a code and research plagiarism detection tools for possible detection of code plagiarism in students' assignments. Such task can be tedious and time consuming to be performed by instructors manually. In addition, there are two major categories of possible source of plagiarism. Those are the Internet and students' team mates. In code plagiarism tools, there are two major criteria that are used to evaluate the performance of such tools. Those are accuracy and speed or performance. In most cases, those two quality attributes conflict with each other.

While code plagiarism evaluation for students' assignments showed that code plagiarism tools may show false alarms in many cases, however, results showed also that such tools can be very helpful in initial investigation for possible plagiarism and they can be very effective useful tools for instructors in this field.

References

[1] L. Prechelt, M. Guido and M. Phlippsen, "JPlag: Finding plagiarisms among a set of programs", Journal of Universal Computer Science, vol. 8, no. 11, **(2000)**.
[2] J. Faidhi and S. K. Robinson, "An empirical approach for detecting program similarity within a university programming environment", Computers & Education, vol. 11, no. 1, **(1987)**, pp. 11-19.
[3] M. Wise, "Detection of similarities in student programs: YAP'ing may be preferable to Plague'ing". ACM SIGSCE Bulletin (Proc. of 23rd SIGCSE Technical Symp.), vol. 24, no. 1, **(1992)**, pp. 268-271.
[4] D. Gitchell and N. Tran, "Sim: a utility for detecting similarity in computer programs", The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, New Orleans, Louisiana, United States, **(1999)** March 24-28, pp. 266-270.
[5] S. Grier, "A tool that detects plagiarism in Pascal programs", ACM SIGCSE Bulletin, vol. 13, no. 1, **(1981)**, pp. 15-20.

[6] U. Manber, "Finding similar files in a large file system[C/OL]", Proceedings of the Winter USENIX Conference, **(1994)**, **(2006)**, pp. 1-10.
[7] M. Mozgovoy, K. Fredriksson, D. White, M. Joy and E. Sutinen, "Fast plagiarism detection system", Lecture Notes in Computer Science, vol. 3772, **(2005)**, pp. 267-270.
[8] B. Baker, "A theory of parameterized patern matching: Algorithms and applications", 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, **(1993)**, pp. 71-80.
[9] A. Broder, Z. Glassman, C. Steven, M. Manasse and G. Zweig, "Syntactic Clustering of the Web", Proceedings of the Sixth WWW Conference. Santa Clara, CA, **(1997)**.
[10] B. Jun-Peng, S. Jun-Yi, L. Xiao-Dong and S. Qin-Bao, "A Survey on Natural Language Text Copy Detection", Journal of Software, vol. 14, no. 10, **(2003)**, pp. 1753-1760.
[11] H. Maurer, F. Kappe and B. Zaka, "Plagiarism, a survey", Journal of universal computer science, vol. 12, no. 8, **(2006)**.
[12] C. Kustanto and I. Liem, "Automatic Source Code Plagiarism Detection", SNPD, **(2009)**, pp. 481-486.
[13] J. Hage, P. Rademaker and N. Vugt, "A comparison of plagiarism detection tools, Technical Report", UU-CS-2010-015, Department of Information and Computing Sciences Utrecht University, Utrecht, The Netherlands, **(2010)**.
[14] A. El Tahir, H. Abdulla and V. Snasel, "Survey of Plagiarism Detection Methods", 2011 Fifth Asia Modelling Symposium, Manila, Philippines, May 24-May 26.
[15] I. Alhami and I. Alsmadi, "Automatic code homework grading based on concept extraction", International Journal of Software Engineering and Its Applications IJSEIA (http://www.sersc.org/journals/IJSEIA/), vol. 5, no. 4, **(2011)**.

Authors

Izzat Alsmadi, is an associate professor in software engineering at Prince Sultan University in Saudi Arabia. He has his master and phd in software engineering from NDSU, USA. His research focus is mainly in software engineering, metrics and testing

Ikdam AlHami, is a lecturer in the computer science department at Yarmouk University in Jordan. His main research interests are in programming and algorithms

Saif Kazakzeh, is a current master student in software engineering at Yarmouk University in Jordan. His main research interests are in software engineering and natural language processing.

