Plagiarism Detection using Free-Text Fingerprint Analysis

Mohamed Elkhidir, Mohannad M. Ibrahim, Tarig A. Khalid, Shawgi Ibrahim, Mohamed Awadalla
Department of Electrical & Electronic Engineering
University of Khartoum
Khartoum, Sudan
Mohamed.elkhidir@uofk.edu, mmibrahim@uofk.edu, t.a.khalid@ieee.org, shawgichan@gmail.com, mohata094@gmail.com

***Abstract*—Plagiarism generally defined as using other people's ideas or work and representing it as one's own original work. Free-text plagiarism detection is an application based on analyzing the texts contained in researches, thesis, scientific reports and also literary products, these analyzed data will be used to compare a group of documents to find out how much these documents are similar. This paper proposes a Free Text Plagiarism Detection Software (FTPDS); which is a software tool that uses documents' fingerprints to detect the likelihood that the documents are plagiarized from each other. The system is able to detect plagiarism between two given documents, given document and group of local documents, and between given document and online available documents. Agile software methodology was used to develop the software and some open source libraries were manipulated and used to search the internet and read PDF documents respectively. The speed of the detection process, the inaccurate detection of the same file and the lag of online search and downloading are stated as future work aspects. Source in this paper means the suspected document which we want to detect the amount of plagiarized data contained in it. The target is the document which is probably the document where the author plagiarized the data from it and claimed that he\she owns that data.**

***Index Terms*—Plagiarism, Detection, FTPS, fingerprint, gram, source, target**

I. Introduction

In plagiarism detection there are two major categories for algorithms to detect the plagiarized data; algorithms that analyze grammar structure of the text and algorithms that analyze the text fingerprints. Plagiarism detection presents many problems in itself, as plagiarism does not always contain an obvious copying of paragraphs. There are situations where plagiarism may involve the copying of smaller chunks of content, which could further be transformed effectively to make it extremely difficult to detect. It is also possible that copied text can be translated into another language [1].

The most important types of plagiarism detection are the *Free-text* and the *Source-code* detection. There are many methodologies used to detect free-text plagiarism. The usual approach for detecting plagiarism splits a document into a (large) set of '*fingerprints'*. A set of fingerprint contains pieces of text that may overlaps with one another. A fingerprint is then used as a query to search the web or a database, in order to estimate the degree of plagiarism [1].

Stylometry is an attempt to analyze writing styles based on text similarity patterns. A particular text can be compared with the typical writing style of an individual based on his or her past works. Alternatively, the text in a single paragraph can be compared with the overall style of writing as found throughout a paper. *Stylometry* is able detect plagiarism without the need for an external corpus of documents. It can be applied to detect essential patterns within documents that capture style parameters that include syntactic forms, text structure as well as the usage of key terms [1].

A home-grown plagiarism detection method built on top of *Google's* search *API* has surprisingly produced superior results as compared to leading software packages in the industry such as *Turnitin*® and *Mydropbox*®. This is mainly due to Google's indexing of many more web sites as compared to these plagiarism detection tools. The limitation of employing *Google's* free *API*, has however restricted their system to *1000* queries a day. As Google does not license their Search engine, they maintain the exclusive control to numerous potential applications. This will also increase the reliance of publishers on these global search engines [1].

Grammar-Analysis of Authors is an approach based on the assumption that authors use a recognizable and distinguishable grammar to construct sentences. The main idea is to analyze the grammar of text documents and to find irregularities within the syntax of sentences, regardless of the usage of concrete words. If suspicious sentences are found by computing the pq-gram distance of grammar trees and by utilizing a Gaussian normal distribution, the algorithm tries to select and combine those sentences into potentially plagiarized sections. The parameters and thresholds needed by the algorithm are optimized by using genetic algorithms [2].

Along with these tools to detect plagiarism in free-text documents there are many tools to detect source code plagiarism. *SIM* is used to detect plagiarism of code written in *Java*, *C*, *Pascal*, *Modula*-2, *Lisp*, *Miranda* . *SIM* is also used to check similarity between plain text files. *MOSS* is available free to use in academics and it is accessible as an online service. *MOSS* support *Ada* programs, *Java*, *C*, *C++,* plain text and Pascal [3]. *Plague* is one of the earliest structure-oriented systems which only support programs written in C [3]. "*JPlag*" system is used to find pairs of similar programs among a given set of programs. It has been successfully used in practice for detecting plagiarisms among student *Java* program submissions. It also provides support to the languages *C, C++* and *Scheme* and it is widely available as a web service [4].

II. Methodolgy

We use a *Three-stage* plagiarism detection process. The stages are *Documents Analysis*, *Data Comparison* and *Results Reporting* respectively. The stages are presented in Fig. 1.

978-1-4799-9907-1/15/$31.00 ©2015 IEEE

A. Documents Analysis

We use an algorithm called *Winnowing Algorithm* in stage one to analyze the document. Winnowing algorithm is an instance of document fingerprinting: a document is summarized by a small set of character sequences called fingerprints which can be efficiently used to find copies of parts of a document in a large document collection [5].

A *gram* is a group of sequential words, letter or sentences. The letter *K* defines the length of each gram. A *hash* value is a unique value given to an input; no two inputs can have the same hash value unless they are the same input. The process of fingerprints extraction is shown in Fig. 2.

At this stage, only portable document format ".pdf" is accepted as an input for the software. The input document may contain plain text, images, tables and graphs. The software will extract only the plaint text and then it will be converted to arrays of strings to be analyzed.

B. Documents Comparison

In this phase we will use the fingerprints generated from the *Source* and *Target* documents to compare them to each other and determine how they are identical.

There are many algorithms tested to search and compare the finger prints. *Linear Search* and *Binary Search* algorithms are examples of algorithms used in search and comparison processes. The comparison process relies mainly here on the duplication of sequence of letters or words in the *source* and the *target* documents. For example if there a 21 letters appears sequentially in *source* and *target* that means hit according to

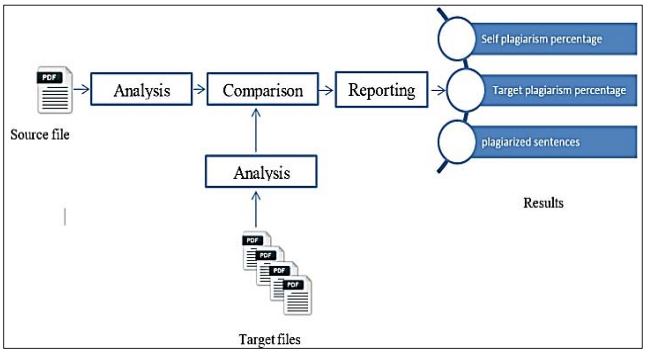

Fig. 1. Plagiarism Detection Software Overview

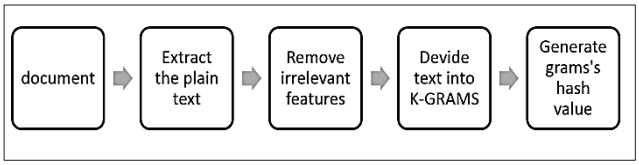

Fig. 2. Document's fingerprints extraction process.

the used algorithm here.

C. Results Reporting

The plagiarism detection software reports the processed results from the comparison stage in three different ways; to help the investigator to identify the best indicator. Although the current computer based plagiarism detectors are very advanced, there is still a need for human judgment upon the indicators represented by these detection tools. The first report generated represents the plagiarism percentage that is detected in Source document against the Target. In other words how, much of the *Source* document data have been plagiarized from the *Target* document. The second report shows the percentage of plagiarized data from the *Source* compared to the size of data in itself. The third report represents the plagiarized data itself and compares it side by side with the *Target* document's identical or similar data.

III. Design & Implementation

In this section, we present the implementation details of the FTPD software. We also present the steps carried on to achieve the final version of the software and the GUI designed for the system.

A. Software Implementation

We use Java technology platforms to implement the FTPD. All of the functions are implemented from scratch except the library used for PDF reading, which is an open source library called *IText*. *GOOGLE API* is used for online detection.

1) ITexet

IText® is an open source library that allows the user to create and manipulate *PDF* documents. It enables developers looking to enhance web and other applications with dynamic PDF document generation and/or manipulation [6]. *IText* has a class called "*PdfReader*" that takes a file path as an input of its constructer and then returns every page's text in that file using a function called " *getTextFromPage*".

2) Remove Irrelevant Features and Extract the GRAMS

We build a function called "*sequenceOfKGrams*" that takes as an input the text extracted from *PDF* document, it removes the white spaces and punctuation marks from the text using a built-in *Java* function called "*String.replaceAll()".*

After removing the irrelevant feature from the text it will be decomposed to *GRAMS*, each *GRAM* has *K* letters in it. These *GRAMS* will be used to generate the unique fingerprint of the document.

3) Generating Document's Fingerprints

We build a function called "*hashGenerator*" to generate a unique number from sequence of letters which is represented here by *GRAMS*. This function uses a built-in java function called "*String.hashCode()*" to generate the unique number. The function is independent of the time and dependent on the order if the letters. Therefore, if two typical sequences are received by the function, they'll both be assigned identical numbers.

4) Fingerprints Comparison

We built a function called "*compareFingerPrints*" to compare two document's fingerprints at every time. It takes only two document's fingerprints in the iteration and compares them using linear search algorithm then produce an output that represents the number of identical fingerprints in the two documents. The time consumed in the comparison process was one of the main factor that affects the detection process so to obtain the results in appropriate amount of time compared to

manual detection and other detection tools two algorithms combined together to produce the results in the desired time.

First algorithm is quick sort algorithm which is used to sort the documents' fingerprints. The second algorithm is binary search which is used in searching the sorted fingerprints to find match for given fingerprint. Those two algorithms combined together led to appropriate average time in plagiarism detection process compared to online tools and manual tools.

5) Online Search

We use *GOOGLE API*, which is an open source library provided by Google used to search in the internet and obtain the URL which locates the *Target* files. The URLs are then used to download the *Target* files to be used in the detection process.

6) Results Reporting

In this phase the main function uses the returned number of duplicated fingerprints and the duplicated fingerprints array in order to show the result to the end user. The number of duplicated fingerprints is used to calculate the two percentages of plagiarism. The first percentage defines the amount data taken from the *Target* document and the second percentage define the amount of data that is plagiarized in the *Source* document. The fingerprints array is used to retrieve the original sentences to be presented to the user.

B. Graphical User Interface

The main window describes briefly the three different available features in the software; the *file to file, local database*, and *online search comparison*. Fig. 3 and Fig. 4 show the main window of the FTPDS and the *online detection interface* respectively.

The *offline* or *local database* feature allows the user to analyze files; the program will check for similarities in a local database and give a chart to display the results. Fig. 5 shows a graphical representation when a *Source* document is checked for plagiarism in a local database.

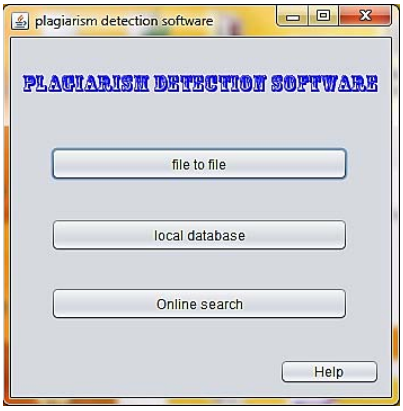

Fig. 3. FTPDS main interface

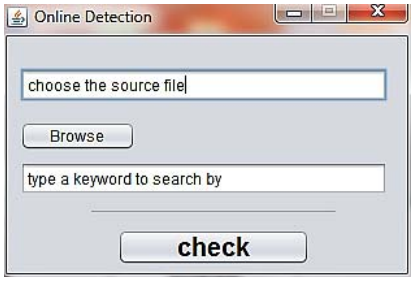

Fig. 4. FTPDS online detection interface

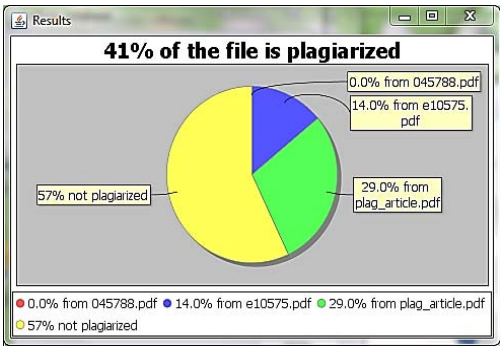

Fig. 5. Local database detection

IV. RESULTS

We choose a number of plagiarized essays for the detection process [7]. The first step is to detect the plagiarism at each essay *manually*; by reading both the *Source* and *Target* documents and then determining the plagiarism percentage at the source files manually. The second step is using our FTPDS software for the process. The last step is using free online tools to detect the plagiarism at the essays and compare the results with our software.

The tables from Table. I to Table. VIII show the results for each of the essays separately. The online free tools used for comparison are *COPYSCAP* (Online tool 1) and *TOOLS4NOOBS* (Online *tool 2).*

One of the paper's objectives is to determine the most accurate k-gram selection to produce the most accurate results compared to manual detection. Generally, the results shown here are used as an indicator to help students and teachers to detect plagiarism in free-text documents.

TABLE I. ESSAY 1 DETECTION USING FTPDS

	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Plagiarism percentage	98%	88%	66%	51%	42%	35%
Average time	0.01s	0.002s	0.002s	0.003s	0.003	0.004s

TABLE II. ESSAY 1 DETECTION USING OTHER TOOLS

	Manual detection	Online tool-1	Online tool-2
Plagiarism percentage	55%	45%	72.29%
Average time	4mins	0.05s	0.04s

TABLE III. ESSAY 2 DETECTION USING FTPDS

	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Plagiarism percentage	99.6%	96.2%	90.2%	85.0%	82.7%	80.8%
Average time	0.05s	0.002s	0.002s	0.003s	0.004s	0.005s

TABLE IV. ESSAY 2 DETECTION USING OTHER TOOLS

	Manual detection	Online tool-1	Online tool-2
Plagiarism percentage	92%	91%	96%
Average time	5mins	0.04s	0.03s

TABLE V. ESSAY 3 DETECTION USING FTPDS

	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Plagiarism percentage	99.6%	98.5%	91.6%	86.8%	83.7%	81.4%
Average time	0.05s	0.002s	0.002s	0.003s	0.004s	0.005s

TABLE VI. ESSAY 3 DETECTION USING OTHER TOOLS

	Manual detection	Online tool-1	Online tool-2
Plagiarism percentage	70%	44%	51%
Average time	6mins	0.045s	0.02s

TABLE VII. ESSAY 4 DETECTION USING FTPDS

	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Plagiarism percentage	99.4%	95.6%	91.9%	88.8%	86.8%	83.8%
Average time	0.05s	0.002s	0.001s	0.002s	0.003s	0.006s

TABLE VIII. ESSAY 4 DETECTION USING OTHER TOOLS

	Manual detection	Online tool-1	Online tool-2
Plagiarism percentage	70%	44%	51%
Average time	4mins	0.045	0.02

V. DISCUSSION

In the case that the same file is inputted as *Source* and *Target* document, the plagiarism percentage may not be exactly *100%*. This is due to the used algorithm; it discards the duplicated fingerprints in the same file and counts them only once. So when calculating the percentage it may be less than *100%*. The practical experiment and its results aimed to help in *K*-gram selection. From the Results section, it is obvious that very small *K* values lead to false positive results. Increasing K to large values also leads to false negative results. It was found that the most accurate range for *K* values when using documents fingerprints analysis is in the range from *3 to 6*. One of the aspects to notice when developing a plagiarism detection tool is the time consumed to produce the results. In our tool we use *quick sort algorithm* to sort data and then the *binary search algorithm* to search in that data. The results are served in an appropriate average time compared to the other tools used in this experiment.

The system suffers from some limitations; the algorithms for detecting graphs and tables plagiarism were not implemented in the software. Also, the number of checked documents in online search is limited by the speed of the connection and available memory. Furthermore, there is no suitable method for online search and comparison without downloading the documents to the local disks found.

VI. CONCLUSION

There are many features that can be added to the FTPDS: Graphs and tables plagiarism detection algorithms, Solving the problem of duplicated fingerprints, Source-code plagiarism detection algorithms implementation and Enhancing the online search mechanism and number of downloadable documents.

REFERENCES

[1] Shams, Khalid. Plagiarism Detection using Semantic Analysis. Dhaka: BRAC University, 2010.

[2] Michael Tschuggnall, G¨ unther Specht. Detecting Plagiarism in Text Documents through. s.l.: Technikerstraße Innsbruck.

[3] Ahmad, Ahmad Gull Liaqat & Aijaz. Plagiarism Detection in Java Code. s.l. : Linnaeus University, 2011.

[4] Sabri, Khair Eddin Muawiyah. Reverse Software Engineering and Reengineering to Detect Plagiarism in Java Programs. s.l. : University of Jordan, 2003.

[5] Sorokina, Daria. Plagiarism Detection in arXiv. s.l. : Cornell University, Ithaca, NY, USA.

[6] http://itextpdf.com/. http://itextpdf.com/. [Online] 9 19, 2014.

[7] http://gethelp.library.upenn.edu/guides/engineering/ee/plagiarize.html#examples. [Online].