I.J. Information Technology and Computer Science, **2012, 10, 80-89**
Published Online September 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2012.10.10

Detection of Plagiarism in Arabic Documents

Mohamed El Bachir Menai
Department of Computer Science, College of Computer and Information Sciences,
King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia
menai@ksu.edu.sa

Abstract— Many language-sensitive tools for detecting plagiarism in natural language documents have been developed, particularly for English. Language-independent tools exist as well, but are considered restrictive as they usually do not take into account specific language features. Detecting plagiarism in Arabic documents is particularly a challenging task because of the complex linguistic structure of Arabic. In this paper, we present a plagiarism detection tool for comparison of Arabic documents to identify potential similarities. The tool is based on a new comparison algorithm that uses heuristics to compare suspect documents at different hierarchical levels to avoid unnecessary comparisons. We evaluate its performance in terms of precision and recall on a large data set of Arabic documents, and show its capability in identifying direct and sophisticated copying, such as sentence reordering and synonym substitution. We also demonstrate its advantages over other plagiarism detection tools, including Turnitin, the well-known language-independent tool.

Index Terms— Plagiarism Detection, Similarity Detection, Arabic, Fingerprinting, Heuristic Algorithm

I. Introduction

The easy access to information through networks and particularly Internet, makes plagiarism an easy operation for students, and might make them taking grades without knowledge background. Several types of plagiarism exist, including direct copying of phrases or passages from a published text without citing the sources, plagiarism of ideas, sources, and authorship. There are other types of plagiarism, such as translating content to another language, presenting the same content with another media like images, video and text, and using program code without permission [1].

There are two main classes of methods used to reduce plagiarism [2]: plagiarism prevention methods and plagiarism detection methods. Plagiarism prevention methods include punishment routines and plagiarism drawback explanation procedures. These methods have a long term positive effect, but they require a long time to be implemented, since they rely on social cooperation between different universities and departments to reduce plagiarism [1]. Plagiarism detection methods include manual methods and software tools. They are easy to implement, but have a momentary positive effect. Both methods can be combined to reduce fraud and cheating. Although software tools are the most efficient approach to identify plagiarism, the final judgment should be done manually [3].

Plagiarism can be discovered in free text (written in natural languages) or in source code (written in programming languages) [2]. Detecting plagiarism in source code is relatively easy because there is neither ambiguity nor interference between words in programming languages. For example, renaming some variables in a source code or modifying the structure of the code can be detected without difficulty by several methods [4]. Plagiarism in free text is more difficult to identify [5], since every word may have many synonyms, and different meanings. Some plagiarism detection methods are language-independent, while other methods are language-sensitive (dedicated to one natural language).

Language-independent methods are based on evaluating text characteristics that are not inherent to a specific language, such as number of single characters and average length of a sentence [3]. Language-sensitive methods are based on evaluating text characteristics that are specific to one language. For example, counting the frequency of a special word in a particular language is a language-dependent attribute [3]. Stylometry-based methods can be used in language sensitive systems. They are inspired by authorship attribution methods and consist basically in classifying writing styles of authors to identify similarity. Content-based methods consist in analyzing specifications of texts in terms of logical structure to discover similarity.

In this paper, we present a plagiarism detection tool, APlag (Arabic Plagiarism detection), built around a content-based method. We describe its main components including its preprocessing stage and a heuristic algorithm for comparing documents at different logical levels (document, paragraph, and sentence levels). We evaluate it experimentally on a large set of Arabic documents and compare it with particularly Turnitin, a language-independent tool.

The rest of this paper is organized as follows. Section 2 presents related work in plagiarism detection methods and tools. Section 3 gives an overview of Arabic language characteristics and challenges. Section 4 details our approach for plagiarism detection, describes a heuristic algorithm for document comparison, and

presents APlag's design and implementation issues. Section 5 presents experimental results. Finally, our conclusions and some future research directions are drawn in Section 6.

II. Related Work in Plagiarism Detection

In the following sub-sections, we present some details on the main methods used for detecting plagiarism in free text.

2.1 Traditional Methods

Traditional plagiarism detection methods are mainly manual. Texts are compared to each other to detect copy-paste content, or to identify different writing styles within a document. The latter method is not applicable if an author has more than one writing style. Search engines can support such methods to check suspicious parts of a document that do not reflect the writing style or understanding level of an author.

Traditional methods include also compression-based techniques. Given two documents D_1 and D_2, let d_1 and d_2 represent their respective compressed files using a given file compression method. Let $a=d_1d_2$ represents the concatenation of d_1 and d_2. Consider now B the concatenation of D_1 and D_2, $B = D_1D_2$, and b its compressed file. If D_1 and D_2 are different, then a and b have the same size. If D_1 and D_2 contain some redundant parts, then the size of b is smaller than a [3].

Traditional methods are easy to apply, but usually require a long processing time and are not reliable, especially in case of large texts. Automatic tools are needed to help users to detect plagiarism quickly and precisely.

2.2 Content-based methods

Content-based methods rely on explicit comparisons of the document contents in a specific representation. Fingerprinting [6] is among the most popular techniques in this category. It consists to measure the similarity of two documents by comparing their fingerprints.

A fingerprint is a set of integers created by hashing subsets of a document to represent its key content. Techniques to generate fingerprints are mainly based on k-grams (a k-gram is a contiguous substring of length k) which serve as a basis for most fingerprint methods. Fingerprints are selected according to different schemes, including "ith hash", "0 mod p hash", and Winnowing method [7].

In the "ith hash" scheme, every ith hash of a document is selected. This method is easy to implement, but not robust in case of insertion, deletion or reordering. For example, if one letter is inserted into the text then the fingerprints will be shifted by one, which makes the altered and the original documents sharing no fingerprint. Consequently, the copy will not be detected [7].

In the "0 mod p" scheme, where p is an integer, hashes located at every "0 mod p" are selected. This method is also easy to implement, but weak in terms of plagiarism detection cases. Similar content is detected if its hashes are among the "0 mod p" selected ones [7].

Winnowing is a local fingerprinting algorithm developed by Schleimer, Wilkerson, and Aiken [7] to select fingerprints from hashes of k-grams. Winnowing is intended to be used in similarity detection algorithms to identify subtle matches of a certain length (small partial matches). Let t and k be the respective guarantee threshold and noise threshold. Two properties must be satisfied to find matches between two documents: (1) a match is detected if there is a substring match at least as long as the guarantee threshold t; (2) any match shorter than the noise threshold k is not detected.

Winnowing algorithm consists in the following steps [7]. Given a window size of t-k+1, each window w_i contains the hashes $h_i \ldots h_{i+w-1}$. A minimum hash value is selected from each window to be a fingerprint. If there is more than one hash with the minimum value, then the rightmost occurrence one is selected. All selected hashes represent the document fingerprint.

Latent Semantic Analysis (LSA)[1] [8] is a technique used to describe relationships between a set of documents and terms they contain. In this technique, words that are close in meaning are assumed to occur close together. A matrix is constructed in which rows represent words, and columns represent documents. Every document contains only subset of all words. Singular Value Decomposition (SVD), a factorization method of real or complex matrix, is used to reduce the number of columns while preserving the similarity structure among rows. This decomposition is time consuming because of the sparseness of the matrix. Words are compared by taking the cosine of the angle between the two vectors formed by any two rows. Values close to 1 represent very similar words, while values close to 0 represent very dissimilar words.

Stanford Copy Analysis Mechanism (SCAM) [9] is based on a registration copy detection scheme. Documents are registered in a repository and then compared with the pre-registered documents. The architecture of the copy detection server consists of a repository and a chunker. The chunking of a document breaks up a document into sentences, words or overlapping sentences. Documents are chunked before being registered. A new document must be chunked to the same unit before comparing it with pre-registered documents. Inverted index storage is used for sorting chunks of registered documents. Each entry of the chunk is a pointer to the documents in which that chunk occurs (posting). Each posting has two parts: document name and its related chunk occurrence number. A small unit of chunk increases the probability of finding

[1] LSA was patented in 1988 (US Patent 4,839,853) by S. Deerwester, S. Dumais, G. Furnas, R. Harshman, T. Landauer, K. Lochbaum and L. Streeter.

similarity between documents. The chunk unit in SCAM is a word. Documents are compared using the Relative Frequency Model (RFM) which consists mainly in computing a set of words that occur with the same frequency in two documents.

Ranking is an information retrieval method used to find the match between the query and documents. Search engines and other retrieval systems are based on this method [6]. A similarity measure is used to calculate match scores between a query and documents which are sorted decreasingly by their scores, and highly ranked documents are then returned. Various types of similarity measures for score matches exist. Hoad and Zobel [6] proposed several variations of a similarity measure based on the number of occurrences of similar words in the documents, such as document lengths, difference of word frequencies in the query and documents, and term weighting. Reported results [6] show that term weighting similarity measure is among the best ones, particularly when stop-words are removed and words are reduced to their root form. Examples of plagiarism detection tools built around content-based methods, include Turnitin [10], EVE2 [11], Wcopyfind [12], and CHECK [13].

2.3 Stylometry-based methods

Stylometry is a statistical approach used for authorship attribution. It is based on the assumption that every author has a unique style [3]. The writing style can be analyzed by using factors within the same document, or by comparing two documents of the same author. Plagiarism detection within the same document and without considering outside references is called intrinsic plagiarism detection [2]. Generally, it is performed by dividing the documents into parts like paragraphs and sentences. The style features are then extracted and analyzed. The main linguistic stylometric features are [14]:

- Text statistics which operate at the character level (number of commas, question marks, word lengths, etc).
- Syntactic features to measure writing style at the sentence level (sentence lengths, use of function words, etc.).
- Part-of-speech features to quantify the use of word classes (number of adjectives or pronouns, etc.).
- Closed-class word sets to count special words (number of stop words, foreign words, "difficult" words, etc.).
- Structural features which reflect text organization (paragraph lengths, chapter lengths, etc.).

Using these features, formulas can be derived to identify the writing style of an author [14]: writer specific and reader specific formulas. Writer specific formula is about the author himself. It includes vocabulary richness, complexity and understandability. Vocabulary richness formulas measure the number of different words in the document. Complexity and understandability formulas measure the understandability of the document and give it a score. Reader specific formula consists in determining the grading level of the document readers. Glatt [15] is an example of a plagiarism detection tool based on a stylometry technique.

Stylometry-based methods can be used in internal and external detection, but content-based methods can be used only in external detection. Moreover, if an author has more than one style, stylometry-based methods can detect false-positive plagiarism. Content-based methods are generally better than stylometry-based methods in terms of precision [16] and can give a proof of plagiarism by visualizing the results.

The most powerful plagiarism detection tools are language-sensitive ones that consider linguistic properties of a particular language [16]. Language-independent tools work on many languages, but give generally poor results.

To the best of our knowledge, APD [17] (Arabic Plagiarism Detection) is the only one existing plagiarism detection tool dedicated to Arabic. It is based on fingerprinting each submitted document by taking the least frequent 4-grams and comparing them to an intra-corpus collection of document fingerprints. Detection of similarities between documents is performed using an information retrieval technique based on fuzzy sets.

III. Arabic Language Characteristics

Arabic language belongs to the Afro-Asian language group It has much specificity which makes it very different from other Indo-European languages. Arabic language has twenty eight alphabet letters (ا، ب، ت ... ي). Three of them are long vowels (‘ا’,’و’,’ي’) and the remaining ones are consonant letters. Arabic letters change shape according to their position in the word, and can be elongated by using a special dash between two letters. Arabic writing is right to left, cursive, and does not include capitalization. Diacritization or vocalization in Arabic consists in adding a symbol (a diacritic) above or below letters to indicate the proper pronunciation and meaning of a word. The absence of diacritization in most of Arabic electronic and printed media poses a real challenge for Arabic language understanding. Arabic is a pro-drop language: it allows subject pronouns to drop, like in Italian, Spanish, and Chinese [18].

The language is highly inflectional. An Arabic word may be composed of a stem plus affixes (to refer to tense, gender, and/or number) and clitics (including some prepositions, conjunctions, determiners, and pronouns). Words are obtained by adding affixes to stems which are in turn obtained by adding affixes to roots. For instance, the word المكاتب, transliterated *al-*

makAtib and meaning *offices*, is derived from the stem مكتب, transliterated *maktab* and meaning *office*, which is derived from the root كتب, transliterated *katab* and meaning *to write*.

IV. Aplag – Arabic Plagiarism Detection

A plagiarism detection system for natural languages should satisfy the following properties [7]:

- Insensitivity to punctuation, extra whitespaces, capitalization, etc.
- Insensitivity to small matches (a match should be large enough to imply plagiarism).
- Insensitivity to permutations of the document content.

Our plagiarism detection tool, APlag, is built around a content-based method. It fulfills the three properties. The first property is handled by a preprocessing of any input text, including tokenization, stop-word removal, rooting and synonym replacement. APlag is based on fingerprinting *k*-grams. The second property is satisfied if *k* is sufficiently long to ignore common idioms of Arabic language. The third property is demonstrated by the performance results on the data set "Structure change" (see Section 5).

The main architecture of APlag is described in Figure 1. Its most important design issues are related to:

- Preprocessing: tokenization, stop-word removal, rooting, and synonym replacement.
- Fingerprinting: make use of *k*-grams, where *k* is a parameter chosen by the user.
- Document representation: for each document, create a document tree structure that describes its internal representation.
- Selection of a similarity metric: use of a similarity metric to find the longest match of two hash strings.

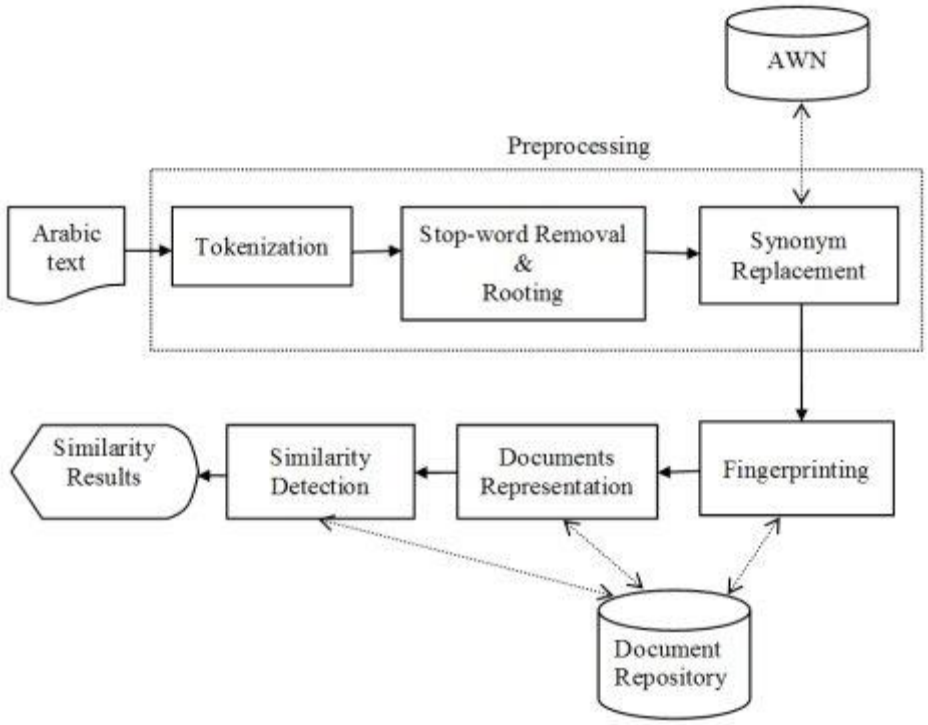

Fig. 1: Main architecture of APlag

4.1 Preprocessing

Most of content-based detection methods assume a preprocessing phase in which stop-words are removed and words are reduced to their root form. The following steps are performed to transform an Arabic text into structured and formatted representation, which is more convenient for the plagiarism detection process.

- Tokenization: input text is broken up into tokens (words).
- Stop-word removal: since stop-words are used in any text, they are considered as unimportant differences between documents. They are removed in order to get more significant results by reducing number of false-positives.
- Rooting: morphological variants are reduced to their root form. Khoja's stemmer [19] is used to reduce words to their root by removing the longest suffix and prefix, and then matching the remaining word with verbal and noun patterns.

- Synonym replacement: words are converted to their most frequent synonyms, which may help to detect advanced forms of hidden plagiarism. Word synonyms are retrieved from Arabic WordNet (AWN) [20]. The first synonym in the list of synonyms of a given word is considered as the most frequent one.

Figure 2 presents an example of preprocessing steps of a sentence in APlag.

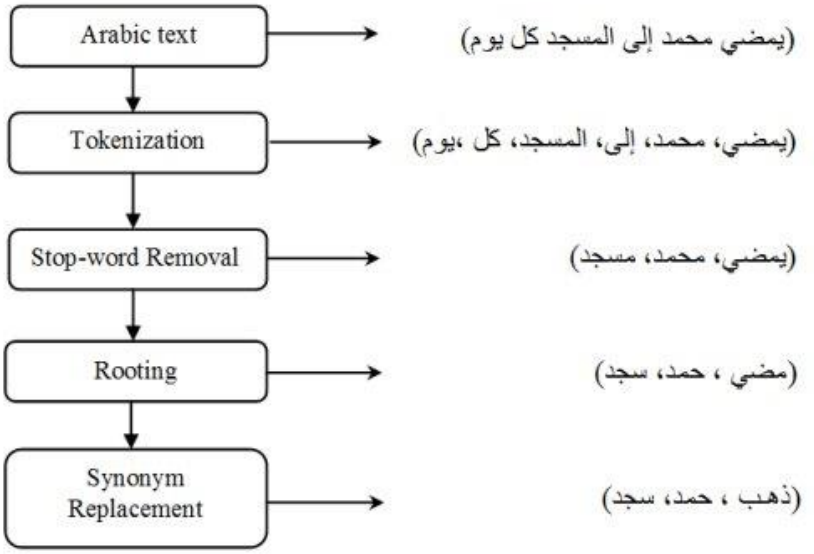

Fig. 2: An example of preprocessing steps of an Arabic text

4.2 Fingerprinting and similarity metrics

To extract fingerprints of a document, we first determine the chunking method that consists in cutting up the text into smaller pieces [21]. A sentence or a word can be used as a unit of chunk. In case of sentence-based chunking, the document is divided into chunks based on chunk parameter n, which group every sequence of n sentences into a chunk. For example, given a document containing the sentences $s_1\ s_2\ s_3\ s_4\ s_5$, if n=3 then the chunks are $s_1\ s_2\ s_3$, $s_2\ s_3\ s_4$, $s_3\ s_4\ s_5$. In a word-based chunking, the document is divided into chunks based on chunk parameter n, which group every sequence of n words into a chunk. For example, given a document containing the words $w_1\ w_2\ w_3\ w_4\ w_5$, if n=3 then the chunks are $w_1\ w_2\ w_3$, $w_2\ w_3\ w_4$, $w_3\ w_4\ w_5$. Word-based chunking gives higher precision in detecting similarity than sentence-based chunking. APlag is based on a word-based chunking method: in every sentence of a document, words are first chunked and then hashed using a hash function.

It is important to select a hash function that minimizes collisions due to mapping different chunks to the same hash. For example, it is easy to implement a hash function that maps each chunk to the sum of the integer values of chunk characters. However, this is not an accurate hash function because the chunks with the same characters in different order have the same hash values (collisions). In our implementation, we use the BKDR (comes from Brian Kernighan and Dennis Ritchie) [22] hash function for chunk hashing. This function returns the sum of multiplications of each character by a special value (named seed and usually equal to 31). Seed value should be a prime number to guarantee the uniqueness of hash values.

Many similarity metrics exist for fingerprint comparison, including Levenshtein distance [23], Longest Common Substring (LCS), and Running Karp-Rabin Matching and Greedy String Tiling (RKR-GST) [23]. The Levenshtein distance measures the minimum number of operations: insertions, deletions, or substitutions to transform one string to another. For example, the Levenshtein distance between "Saturday" and "Sunday" is three. The Longest Common Substring (LCS) consists in finding the common longest substring in two strings. For example, the common longest substring in "Saturday" and "Sunday" is "day". RKR-GST [24] is used for comparing amino acid bio-sequences. It consists in tiling one string with matching substrings of a second string. RKR is an improvement technique to speed up the GST algorithm. A hash value is created for each substring of length s of the pattern string and for each substring of length s of the text string. Each of these hash values of the pattern string is compared with the hash values of the text string. If the pattern and text hash values are equal, then there are matches between the corresponding pattern and text substrings.

A key issue in similarity detection is to choose the adequate metric. For plagiarism detection, Levenstein distance and LCS are more suitable, since plagiarism involves modification of a text (insertion, removal …). In APlag, we choose to use LCS, because it is based on the concept of similarity rather than distance.

4.3 Text comparison heuristics

A tree representation is created for each document to describe its logical structure. The root represents the document itself, the second level represents the paragraphs, and the leaf nodes contain the sentences. This representation is similar to the one used in CHECK [13]. It is intended to avoid unnecessary comparisons between several documents. Trees are then explored top-down and compared first at document level, then at paragraph level and finally at sentence level.

We define a heuristic algorithm for each level of the tree: Algorithm 1 (document level), Algorithm 2 (paragraph level), and Algorithm 3 (sentence level).

At document level, two documents are compared according to their common hashes and a fixed threshold. If the number of hashes in the intersection subset is greater than the threshold, then there is a potential similarity between both documents. In that case, the comparison process continues at paragraph level, otherwise no similarity is detected and the process is stopped. If a possible similarity is detected at paragraph level, then the process continues at sentence level, otherwise the process terminates. If there is a possible similarity between two sentences, then it is measured using LCS metric. If the length of the longest common sequence is greater than the length of the minimum sentence multiplied by a threshold, then similar strings are identified in both sentences, otherwise the process continues with the next sentence.

Algorithm 1: Document level heuristic

Input : *DocA, DocB // Two input documents*

Output: *similarity*

Begin

DocMinSize = min (|DocA|, |DocB|)

DocIntersectionSize = |DocA ∩ DocB|

If *(DocIntersectionSize >= DocMinSize*DocThreshold)*

Then

//Possible similarity

//Check similarity at paragraph level

similarity = true

Else

similarity = false

End

Algorithm 2: Paragraph level heuristic

Input : *ParA, ParB // Two input paragraphs*

Output: *similarity*

Begin

ParMinSize = min (|ParA|, |ParB|)

ParIntersectionSize = |ParA ∩ ParB|

If *(ParIntersectionSize >= ParMinSize*ParThreshold)*

Then

//Possible similarity

//Check similarity at sentence level

similarity = true

Else

similarity = false

End

Algorithm 3: Sentence level heuristic

Input : *SenA, SenB*

Output: *similarity, similar substrings in SenA and SenB*

Begin

SenMinSize = min(|SenA|, |SenB|)

SenIntersectionSize = |SenA ∩ SenB|

If *(SenIntersectionSize >= SenMinSize*SenThreshold)*

Then

LongestCommonSeq = LCS (SenA, SenB)

If *(|LongestCommonSeq| >= SenMinSize*SimilarityThreshold)*

Then

//Similarity detected

//Determine similar

//substrings

similarity = true

Else

similarity = false

Else

similarity = false

End

One important step in the heuristic algorithms consists in calculating the intersection of two given sets. Its computation by enumerating all matching hashes is time consuming and conflicts with our initial goal of adopting the document tree representation to reduce the number of comparisons. We propose to approximate the intersection between two sets of hashes by adding a string of bits to each node in each level of a document and use it to estimate the intersection as follows. The bit

values in a string of length *m* are calculated by the modulus (%) of the hashes to *m*. The results of this operation represent the hash positions in the bit string.

For example, given a document *A* containing the hashes 2435, 6786, 2234, and 4673. To obtain a bit string of length 10, the following operations are performed:

2435 % 10 = 5

6786 % 10 = 6

2234 % 10 = 4

4673 % 10 = 3

The results 5,6,4,3 represent the positions of the bits to set to 1 in the bit string. The remaining bits are set to 0.

0 0 0 1 1 1 1 0 0 0

3 4 5 6

The number of bits set to 1 resulting from a Boolean AND operation of two bit strings represents the size of their intersection. For example, given two documents *A* and *B* represented by their respective bit strings 0001111000 and 1001010010, the size of the intersection between *A* and *B* is 2.

There are two options for associating bit strings to the tree nodes:

- Bit strings are associated to the leaves only (sentences) and concatenated in higher levels (paragraphs and document),
- Bit strings are duplicated at each level and then associated to each node of the tree.

The first option saves the memory usage, but it is time consuming. Conversely the second option is memory demanding, but it is less time consuming. We choose to implement the second option in order to preserve the interactivity of the tool by guarantying a reasonable response-time.

V. Experimental Evaluation

We implemented a prototype of APlag in Java and evaluated its performance on a handmade data test set of 300 Arabic documents of about 800 words each. We extracted 20 documents from different books available on Alwaraq website [25]. We generated 3 data sets from the original documents as follows:

- **Data set: *Synonym***

  5 candidate documents were generated from each original document by replacing randomly 50% of the total number of words in each document with one of their synonyms. Stop-words were not considered.

- **Data set: *Structure change***

  5 candidate documents were generated from each original document by changing the structure of randomly selected sentences. The number of generated sentences represents 50% of the total number of sentences.

- **Data set: *All data***

  5 candidate documents were generated from each original document by copying randomly selected sentences (40% of the total number of sentences), replacing selected words with one of their synonyms (20% of the total number of words), and changing the structure of selected sentences (40% of the total number of sentences).

The data sets *Synonym* and *Structure change* were used to evaluate the performance of APlag in detecting hidden plagiarism. The data set *All data* served to measure APlag's overall performance in detecting hidden plagiarism and exact copy of parts of texts.

Three variants of APlag were tested to measure the impact of stop-word removal, rooting, and synonym replacement:

- SWR: only stop-word removal is applied to the input texts.
- SWR+Rooting: stop-word removal and rooting are applied to the input texts.
- SWR+Rooting+Synonym: stop-word removal, rooting, and synonym replacement are applied to the input texts.

The chunk parameter was set to 3. The document threshold *DocThreshold* was set to 0.1 assuming that documents describing different subjects have an intersection less than 10% of the minimum document size. The paragraph threshold *ParThreshold*, sentence threshold *SenThreshold*, and similarity threshold *SimilarityThreshold* were set to 0.2, 0.1, and 0.5, respectively. Performance results were measured using *Recall* (1) and *Precision* (2) metrics.

$$Recall = \frac{number\ of\ plagiarized\ sequences\ identified}{total\ number\ of\ plagiarized\ sequences} \times 100 \quad (1)$$

$$Precision = \frac{number\ of\ plagiarized\ sequences\ identified}{total\ number\ of\ sequences\ identified} \times 100 \quad (2)$$

Figures 3 and 4 show respective mean precision $Mean(Precision)$ and mean recall $Mean(Recall)$ obtained by APlag's variants on the 3 data sets. The results obtained can be summarized as follows:

- SWR does not detect hidden plagiarism (synonym replacement and structure change). Its overall performance on all data sets is weak ($Mean(Precision) = 53\%$, $Mean(Recall) = 37\%$).
- SWR+Rooting does not detect synonym exchanges, but it can identify changed sentence structure with high precision and recall ($Mean(Precision) = 95\%$, $Mean(Recall) = 72\%$). This shows that reducing words to their root can enhance the performance of the plagiarism detection.
- SWR+Rooting+Synonym is the best performing APlag's variant achieving $Mean(Precision) = 97\%$ and $Mean(Recall) = 94\%$. Synonym replacement is detected with $Mean(Precision) = 96\%$, while sentence structure change is detected with $Mean(Precision) = 93\%$.

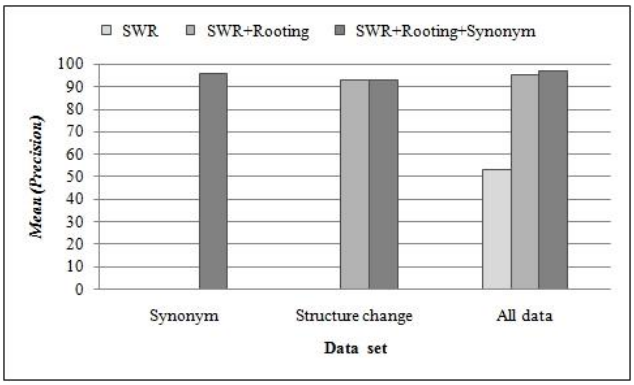

Fig. 3: Mean precision of APLag for each data set

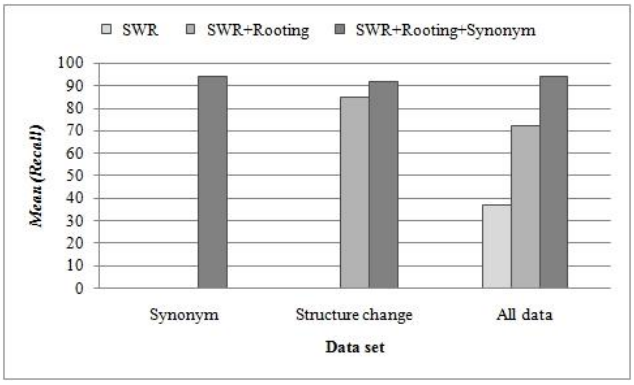

Fig. 4: Mean recall of APLag for each data set

Turnitin was used as a comparative baseline for APlag. It was set to exclude small matches by less than 1%. The performance results of Turnitin are returned in terms of Originality Similarity Index (OSI): percentage of matched words the tool was able to find for the tested document. For that reason, OSI is also estimated for APlag. Figure 5 shows the mean of the originality similarity index, $Mean(OSI)$, given by APlag and Turnitin for each data set. Turnitin was not able to detect any synonym replacement, but its performance is close to APlag's one in detecting changes in text structure: $Mean(OSI) = 40\%$ for APlag and $Mean(OSI) = 35\%$ for Turnitin. Overall, APlag outperformed Turnitin: $Mean(OSI) = 90\%$ for APlag and $Mean(OSI) = 67\%$ for Turnitin. Although Turnitin is worldwide used, its results for detecting similarities in our data sets are not competitive. This indicates that language-independent tools could be actually inefficient on specific languages, such as Arabic.

Table 1 reports comparison results of APlag (SWR+Rooting variant) and APD obtained in a preliminary previous study [26]. It shows $Mean(Recall)$, its standard deviation $\sigma(Recall)$, $Mean(Precision)$, and its standard deviation $\sigma(Precision)$. The results were obtained on only 12 documents (we have not been able to continue experiments with APD because it is no longer available online). The results of APD are close to those of APlag variant without synonym processing.

Overall, APlag results outperform those of Turnitin on the same data sets. However, no conclusion can be drawn regarding its competitiveness with APD, since the number of documents tested is not significant.

APlag's performance is dependent on Khoja's stemmer and synonyms retrieved from AWN. According to the comparative evaluation study of Arabic language morphological analyzers and stemmers [27], Khoja's stemmer achieves the highest accuracy then the tri-literal root extraction algorithm [28] and the Buckwalter morphological analyzer [29]. So, we do not expect to increase the performance of APlag by using other stemmers. However, using other synonym databases might impact its performance.

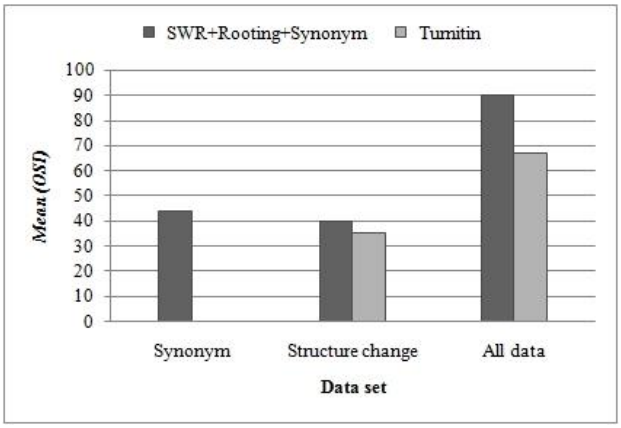

Fig. 5: Mean originality similarity index for APlag and Turnitin

Table 1: Comparison results of APlag (SWR+Rooting) and APD

	APlag	APD
$Mean(Recall)$ (%)	100	84.8
$\sigma(Recall)$ (%)	5	---
$Mean(Precision)$ (%)	93	90
$\sigma(Precision)$ (%)	2	---

VI. Conclusion and Future Work

We have presented APlag, a prototype of a plagiarism detector for Arabic documents in which some hidden forms of plagiarism can be detected, such as sentence structure change and synonym replacement. We have described its main components, in particular heuristic algorithms for comparing fingerprints of Arabic documents at different logical levels (document, paragraph, and sentence) to pass up redundant comparisons.

Finally, we have presented and discussed a series of experiments to demonstrate its effectiveness on a large set of Arabic documents. The results indicate that APlag has the capability to detect precisely exact copy, change in sentence structure, and synonym replacement. Comparison with Turnitin, one of the most used plagiarism detection tool, indicates that APlag compares favorably in terms of quality of results. Additional testing of other synonym databases and different parameters, such as thresholds and chunk values would be useful to further optimize the tool.

An improvement would be to include paraphrasing detection and an archive of submitted files to check against new submissions.

Acknowledgments

This paper is an extended version of a conference paper presented at the 6th International Conference on Computer Science & Education (ICCSE 2011) [26]. Only some preliminary results of this work have been communicated in that conference.

References

[1] Lukashenko R., Graudina V., Grundespenkis J. Computer-based plagiarism detection methods and tools: an overview [C]. In: Proceedings of the International Conference on Computer Systems and Technologies, Bulgaria, 2007, 14-15.

[2] Maurer H., Kappe F., Zaka B. Plagiarism – A survey [J]. Journal of Universal Computer Science, 2006, 12(8): 1050-1084.

[3] Gruner G., Naven S. Tool support for plagiarism detection in text documents [C]. In: Proceedings of the ACM symposium on Applied Computing, Santa Fe, New Mexico, 2005, 13-17.

[4] Menai M.B., Al-Hassoun N.S. Similarity detection in Java programming assignments [C]. In: Proceedings of the 5th International Conference on Computer Science & Education, Hefei, China, 2010, 356-361.

[5] Mozgovoy M., Kakkonen T., Sutinen E. Using natural language parsers in plagiarism detection [C]. In: Proceedings of the SLaTE Workshop on Speech and Language Technology in Education, Farmington, Pennsylvania, USA, 2007.

[6] Hoad C., Zobel J. Methods for identifying versioned and plagiarized documents [J]. Journal of the American Society for Information Science and Technology, 2003, 54(3): 203-215.

[7] Schleimer S., Wilkerson D., Aiken A. Winnowing: local algorithms for document fingerprinting [C]. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 2003, 9-12.

[8] Dumais S.T. Latent Semantic Analysis [J]. Annual Review of Information Science and Technology, 2005: 38-188, doi:10.1002/aris. 1440380105.

[9] Shivakumar N., Garcia-Molina H. SCAM: a copy detection mechanism for digital documents [C]. In: Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries, Austin, Texas, USA, June 1995.

[10] http://www.turnitin.com, visited: 10 Feb. 2012.

[11] http://www.canexus.com/eve/, visited: 15 Jan. 2012.

[12] http://plagiarism.phys.virginia.edu/Wsoftware.html, visited: 15 Jan. 2012.

[13] Si A., Leong H., Lau R. CHECK: a document plagiarism detection system [C]. In: Proceedings of ACM Symposium for Applied Computing, Feb. 1997, 70-77.

[14] Eissen S., Stein B., Kulig M. Plagiarism detection without reference collection [C]. In: Proceedings of the 30th Annual Conference of the German Classification Society, Berlin: Freie university, 8–10 Mar. 2006, 359-366.

[15] http://www.plagiarism.com/self.detect.htm, visited: 15 Jan. 2012.

[16] Lancaster T., Culwin F. Classifications of plagiarism detection engines [J]. ITALICS, 2005, 4(2).

[17] Alzahrani S.M., Salim N. Statement-based fuzzy-set IR versus fingerprints matching for plagiarism detection in Arabic documents [C]. In: Proceedings of the 5th Postgraduate Annual Research Seminar (PARS 09), Johor Bahru, Malaysia, 2009.

[18] Farghaly A., Shaalan K. Arabic natural language processing: challenges and solutions [J]. ACM Transactions on Asian Language Information Processing, 2009, 8 (14): 1-22.

[19] Khoja S. Stemming Arabic Text [R]. 1999. http://zeus.cs.pacificu.edu/shereen/research.htm

[20] Black W., Elkateb S., Rodriguez H., Alkhalifa M., Vossen P., Pease A., Fellbaum C. Introducing the Arabic WordNet project [C]. In: Proceedings of the 3rd International WordNet Conference, Masaryk University, Brno, 2006, 295-300.

[21] Pataki M. Plagiarism detection and document chunking methods [C]. In: Proceedings of the 12th International WWW Conference, Budapest, Hungaria, May 20-24, 2003.

[22] Kernighan B.W., Ritchie D.M. The C Programming Language [B]. 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1988.

[23] Levenshtein V.I. Binary codes with correction for deletions and insertions of the symbol 1 [J]. Probl. Peredachi Inf., 1965, 1(1), 12–25.

[24] Karp R.M., Rabin M.O. Efficient randomized pattern-matching algorithms [J]. IBM Journal of Research and Development, 1987, 31(2): 249-260.

[25] http://www.alwaraq.net, visited: 2 Feb. 2012.

[26] Menai M.B., Bagais M. APlag: a plagiarism checker for Arabic texts [C]. In: Proceedings of the 6th International Conference on Computer Science & Education (ICCSE 2011), Singapore, Aug. 3-5, 2011, 1379-1383.

[27] Sawalha M., Atwell E. Comparative evaluation of Arabic language morphological analysers and stemmers [C]. In: Proceedings of 22nd International Conference on Computational Linguistics (COLING 2008), Manchester, UK, Aug. 2008, 107-110.

[28] Al-Serhan H., Al Shalabi R., Kannan G. New approach for extracting Arabic roots [C]. In: Proceedings of the International Arab Conference on Information Technology (ACIT'2003), Potland, Oregon, USA, 2003, 42-59.

[29] Buckwalter T. Issues in Arabic orthography and morphology analysis [C]. In: Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages (Semitic'04), Geneva, Switzerland, 2004, 31-34.

Mohamed El Bachir Menai received a Ph.D. degree in computer science from Mentouri University of Constantine, Algeria, and University of Paris VIII, France, in 2005. He received also a "Habilitation universitaire" in computer science from Mentouri University of Constantine, in 2007 (it is the highest academic qualification in Algeria and France). He is currently associate professor in the department of computer science at King Saud University. His main interests include satisfiability problems, evolutionary computing, machine learning, and natural language processing.

How to cite this paper: Mohamed El Bachir Menai,"Detection of Plagiarism in Arabic Documents", International Journal of Information Technology and Computer Science(IJITCS), vol.4, no.10, pp.80-89, 2012. DOI: 10.5815/ijitcs.2012.10.10