Unsupervised Morphological Relatedness

Ahmed Khorsi
Computer Science Department
Al-Imam Mohammad Ibn Saud Islamic University
Riyadh, Kingdom of Saudi Arabia

Abeer Alsheddi
Computer Science Department
Al-Imam Mohammad Ibn Saud Islamic University
Riyadh, Kingdom of Saudi Arabia

***Abstract*—Assessment of the similarities between texts has been studied for decades from different perspectives and for several purposes. One interesting perspective is the morphology. This article reports the results on a study on the assessment of the morphological relatedness between natural language words. The main idea is to adapt a formal string alignment algorithm namely Needleman-Wunsch's to accommodate the statistical characteristics of the words in order to approximate how similar are the linguistic morphologies of the two words. The approach is unsupervised from end to end and the experiments show an nDCG reaching 87% and an r-precision reaching 81%.**

***Keywords*—Arabic Language; Computational Linguistics; Morphological Relatedness; Semitic Morphology; Unsupervised Learning**

I. Introduction

Expanding a query word to its variants is one of the challenges facing an Information Retrieval (IR) system in order to achieve a decent recall.

Take the Arabic word "كتاب" ([kitaAb]: book)[1]. An IR system seeking information related to this word in a collection of documents should pick all the documents in which occur the word itself or any of its variants such as "كاتب" ([kaAtib]: writer), "كتب" ([kutub]: books), "كتيب" ([kutayib]: small book),...etc.

The purpose is to capture the documents in which also occur words with meanings close to the query words. One way to do this is to exploit the fact that two words derived from the same morphological origin are likely to share the same broad meaning.

Experience shows that such technique depends on the type of the language morphology [2], [3]. In languages like English, a word is generally a concatenation of prefixes, stem and suffixes. For instance, the word "*unbreakable*" is composed of "*un*", "*break*" and "*able*". While the principle of decomposition can be applied to Arabic, words in Semitic languages, such as Arabic, are actually derived by combining two entities; each might be regarded as an origin: *root* and *pattern* [4]. For instance, the word "كاتب" ([kaAtib]: writer) is coined by combining the root "ك ت ب" ([k t b]) and the pattern "فاعل".

This means the normal form an IR system may reduce the Arabic query word to one of three different types, each expressing a different level of similarity.

- Stem: The extraction of the stem is simply the elimination of the prefixes and suffixes.
- Root: Specific to the Semitic languages, its extraction is more complicated than the extraction of the stem as it tries to identify the three, four or five core letters among all letters of the words [5]. The words derived from the same root have a common meaning broader than the one shared by words having a common stem.
- Pattern: The extraction of the pattern is the identification of the non-core letters and their positions among the core ones. The authors are not aware of any IR system that makes use of pattern as the normal form of words.

For instance the word "الكاتب" ([alkaAtib]: The writer) might be reduced to its stem "كاتب", to its root "ك ت ب" ([k t b]) or to the pattern "فاعل".

While the trend is to reduce the query words to the normal form then to match them against the stored normal forms, another approach [6] is to redesign the matching itself in such a way that it identifies words morphologically close to the query word by measuring the Morphological Relatedness (MR).

The present work is an attempt to enhance this approach [6] in order to improve the effectiveness of morphology-aware matching. Three major changes are introduced to the computation of the MR:

1. The words are first processed by an unsupervised morpho-segmenter which tries to remove the prefixes and suffixes.
2. The frequency is involved earlier in the computation of the Longest Common Subsequence (LCS). An alignment algorithm is adapted to take into account the frequencies of the letters in the computation of the cost.
3. The comparison is extended to *n*-grams.

Section II reviews the principle of string alignment that will be used to calculate the MR [7]. Section III overview works related to the idea of computing the similarity among natural words. Section IV introduces the proposed approach. Section V details the test and discusses the results.

II. Sequences Alignment

Sequence Alignment (SA) is the process of identifying the minimal number of edit operations required to transform one string of characters into another [8] [9]. In an edit operation, a character may undergo one of the following changes:

[1] In this paper, Arabic is represented in some or all of three variants according to context: "Arabic word" ([Buckwalter Arabic transliteration] [1]: English translation).

- Indel: The character is simply deleted or a new character is inserted.
- Substitution: The character is substituted by another.

Other operations might be defined on the basis of these. This study will limit this section to the simplest definitions of the underlying concepts. Each edit operation is assigned a cost.

Having two strings (words) at hand, the objective is first to calculate the minimal cost of edit operations required to transform the first string into the second one. Then, to identify what and where are those edit operations. The algorithm which will be focused on in this paper is due to Needleman, Saul and Wunsch, Christian [7]. Algorithm 1 depicts the steps to align two words A and B, where $cost_s$ denotes the cost of a substitution, $cost_{g_d}$ and $cost_{g_i}$ are a gap penalty pointing out aligned with a null. First, a two-dimensional matrix is built $M[0 \dots n, 0 \dots m]$, where *n* is the size of A and *m* is the size of B, and the rows are labeled with letters of A and the columns are labeled with letters of B. The extra row and column at index zero have been added to deal with the empty string. Second, all cells are filled with the similarity values starting from the top row and going to the bottom-right cell from left to right. Each cell in this matrix holds the similarity between two substrings of the two strings whose ends intersect at a given cell; that is, the cell $M[i, j]$ holds the similarity between substrings $A = a_1 \dots a_i$ and $B = b_1 \dots b_j$. Then the last cell $M[n, m]$ holds the similarity between strings A and B.

Algorithm 1: Needleman-Wunsch similarity

Input : Two strings $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_m$
Output: The Needleman-Wunsch similarity between two strings A and B

```
M: matrix[0...n, 0...m]
M[0,0] ← 0
for i ← 1 to n do
    M[i,0] ← M[i−1,0] + cost_gd
for j ← 1 to m do
    M[0,j] ← M[0,j−1] + cost_gi
for i ← 1 to n do
    for j ← 1 to m do
        M[i,j] ← max(M[i−1,j−1] + cost_s, M[i−1,j] + cost_gd, M[i,j−1] + cost_gi)
return M[n,m]
```

Example: Table 1 shows an example of applying Algorithm 1 to find the similarity between two words, *A*= "*winter*" and *B*= "*write*". $cost_s$ was supposed to be equal to equal to 1 when the two letters match, and the other costs are equal to -1. For instance, the value in the cell $M[4, 4]$ indicates that the similarity between "*wint*" and "*writ*" is 1.

$$\begin{aligned} M[4][4] &= \max(M[3,3] + cost_s, M[3,4] + cost_{g_d}, \\ &\qquad M[4,3] + cost_{g_i}) \\ &= \max(0+1, 0-1, -1-1) \\ &= 1 \end{aligned}$$

Table 1: Example of the Needleman-Wunsch similarity

		w	**r**	**i**	**t**	**e**
	0	-1	-2	-3	-4	-5
w	-1	1	0	-1	-2	-3
i	-2	0	0	1	0	-1
n	-3	-1	-1	0	0	-1
t	-4	-2	-2	-1	1	0
e	-5	-3	-3	-2	0	2
r	-6	-4	-2	-3	-1	**1**

So the value in the last cell $M[6, 5]$ means that the similarity between the words "*write*" and "*winter*" is 1.

$$\begin{aligned} M[6][5] &= \max(M[5,4] + cost_s, M[5,5] + cost_{g_d}, \\ &\qquad M[6,4] + cost_{g_i}) \\ &= \max(0-1, 2-1, -1-1) \\ &= 1 \end{aligned}$$

III. Related Works

Beside [6], the authors are not aware of any published work on the concept of the MR. This section overviews a number of approaches that make use of the concepts of edit distance in the context of Natural Language Processing (NLP).

Ghafour *et al.* [10] suggest to adapt the Levenshtein's distance [11] in the comparison of compare Arabic words. The cost of the operation captures three levels of similarity: phonetic, character form and keyboard wise similarities. Gomaa & Fahmy [12] proposed a system to automatically grade answers to an essay question. They tested different similarity measures, trying to achieve a maximum correlation value between the proposed system and human experts grades. The Needleman-Wunsch similarity [7] is one of the measures they tested. It achieves 26.5% of the correlation score.

Mustafa & Al-Radaideh [13] who investigated for a *n*-grams based comparison claim that the bigram based comparison is more effective than the trigram based comparison, and that the use of pure *n*-grams technique alone does not perform with Arabic words as well as it does with English words. In [14] Mustafa suggests to extend the comparison to non contiguous letters. For instance, the *n*-grams in the word $W = w_1 w_2 \dots w_n$ might be $\{w_1.w_2, w_1.w_3, \dots w_{n-2}.w_n, w_{n-1}.w_n\}$. Tested on 160,000 words, the author claims that this approach outperforms the classical one when using a rule-based stemming. The approach meets the balancing point of recall and precision at around 40%.

Reference [15] opted for the rule based approach to match Arabic words. It identifies the common letters between two words, compare their order and checks whether the uncommon letters are valid affixes or not. If these two conditions apply, a match is raised. This approach uses a predefined list of affixes. The authors tested 1,500 distinct words and claim they have achieved a 15% error rate at a 13% missing rate. The error rate measures how many erroneous hits are found among all the relevant variants. While the missing rate measures how many relevant variants are missing among all actual relevant variants in the dataset.

IV. A Two Steps Morphological Relatedness

To the best of the authors' knowledge, the concept of MR was introduced by Ahmed Khorsi [6] where he tried to substitute the classical normalize-then-match approach in the matching process used in the IR systems with a straight comparison that takes into account the core letters intended to carry the core meaning of the word and the non-core ones which are meant to carry the variation in the meaning. Basically, two challenges had to be addressed: 1. How to distinguish the core letters from the non-core ones 2. How to model the matching and mismatching, either within the core letters or the non-core letters.

As the core letters in a word might not be contiguous, the matching made use of the computation of the Longest Common Subsequence (LCS) [16]. As its name suggests, it extracts the longest sequence of letters, either contiguous or not, but in same order shared by the two words. As the LCS does not guarantee that the common letters are all core ones, the formula to calculate the MR tries to exploit the fact that the non-core letters are usually more frequently used than the core ones. The words in a collection of documents found to have the highest MRs with the word at hand were considered the most morphologically related and the most probable to carry a meaning very close to the meaning carried by the word at hand as shown in Algorithm 2. The MR measure is:

Algorithm 2: Top five morphological relatedness

input : A word w
output: Five words have the highest MRs with w
foreach *word* w_i *in a corpus* **do**
  $mr_i \leftarrow \text{MR}_{(w,w_i)}$
  $\text{MR}_{all} \leftarrow \text{MR}_{all} \cup \{mr_i\}$
  $\text{W}_{all} \leftarrow \text{W}_{all} \cup \{w_i\}$
$\text{MR}_{top} \leftarrow$ top five values from MR_{all}
return W_{top}

$$MR_{(w_1,w_2)} = \sum_{i=1}^{|LCS_{(w_1,w_2)}|} log\left(\frac{1}{freq(LCS_{(w_1,w_2)}[i])}\right) - \sum_{i=1}^{|\overline{LCS_{(w_1,w_2)}}|} log\left(\frac{1}{freq(\overline{LCS_{(w_1,w_2)}}[i])}\right) \quad (1)$$

Where w_1 and w_2 are two strings whose will be tested, $|w_1|$ is the length of w_1 and $w_1[i]$ is the i^{th} letter in w_1. $LCS_{(w_1,w_2)}$ is the LCS between w_1 and w_2, and $\overline{LCS_{(w_1,w_2)}}$ is LCS's complement (i.e., it contains all letters that are not included in $LCS_{(w_1,w_2)}$). $freq(a)$ is the frequency (count) of the letter a in a corpus.

Tested on more than 200,000 words, such simple approach could achieve 82% nDCG and 78% R-precision when the five (05) highest MRs are picked.

Based on the analysis of results and the lacks of identification in the original work [6], the present work introduces three major changes to the concept of MR computation:

1. Stemming: To avoid the interference of (pre/suf)fix letters in the computation of the MR, an unsupervised morphological segmentation is applied beforehand to extract the stems on which the actual computation of the MR is applied.
2. Alignment cost: The LCS extraction in the original approach does not make any distinction between letters. In an attempt to involve the frequency factor early in the process, the cost model of an alignment is adapted to accommodate the frequencies.
3. *N*-grams: The investigation of this study is extended to the effect of making the comparison unit *n*-grams of letters rather than single letters.

The first step relies on a morphological segmentation of words, which is also suggested in a separate work. The following paragraphs summarize its main traits.

A. Morphological Segmentation

The objective of this section is to take a quick look at the step introduced before the actual computation of the MR. This step aims at identifying the prefix, stem and suffix of a word, as experience has shown that an unsupervised morphological segmentation is feasible and could reach acceptable performance [17]–[28]. The following paragraphs describe how the unsupervised learning and segmentation of natural words are approached.

Let the word be "*unbreakable*", which is formed by concatenating the prefix "*un*", the stem "*break*" and the suffix "*able*". The vocabulary suggests such segmentation should contain other words with different combinations of prefixes, stems and suffixes (e.g. "*rebreakable*", "*unbreaking*"...etc.), which makes the occurrence of "*un*" have a weak dependence on the occurrence of "*break*", whose occurrence is also relatively independent from the occurrence of the suffix "*able*". On the other hand, it is obvious, but worth mentioning, that each morpheme is not separable, either from its first letter or its last one. The proposed approach is all about exploiting this fact: a morpheme depends on only the letters of which it is formed. To address the challenge of how to assess the dependence among the letters of a word, probabilistic dependence was employed [29].

1) Segmentation Algorithm Algorithm 3 iterates over the word, letter by letter, and, for every position, computes two dependencies: 1. the dependency of the prefix on its last letter; 2. the dependency of the suffix on its first letter, where the prefix starts (inclusive) at the current letter and the suffix ends (inclusive) at it. The difference between the two values then points to which of the two fragments (i.e. the prefix or the suffix) is more attached to the current letter. The algorithm keeps going until it encounters a change of the direction of the highest dependency. If, in the immediately preceding position, the prefix depends on the letter more than the suffix does and, in the current position, the suffix depends on the letter more than the prefix does, a cutting point is marked between the previous and the current letter.

2) Computation of the Dependence By definition, the concept of dependence that used is symmetric [29]. In this context: "*the string depends on the letter*" means "*the letter depends on the string*" and vice versa.

The dependency of a letter `a` on the prefix α`:` will be called the forward dependency, and it is denoted `fdep(u)`

where u=αa is the prefix appended with the letter under consideration a. The dependency of the letter a on the suffix β: will be called the backward dependency, and it is denoted bdep(v) where v=aβ is the suffix headed by the letter under processing a. The beginning and the end of a word are marked by respectively # and $. The forward dependency is then:

$$fdep(u) = \frac{P(\alpha \to a)}{P(a)} \tag{2}$$

and

$$bdep(u) = \frac{P(a \to \beta)}{P(a)} \tag{3}$$

Where $P(\alpha \to a)$ is the conditional probability:

$$P(\alpha \to a) = \frac{Count(\alpha a)}{Count(\alpha)} \tag{4}$$

and $P(a \to \beta)$ is the conditional probability:

$$P(a \to \beta) = \frac{Count(a\beta)}{Count(\beta)} \tag{5}$$

then

$$fdep(u) = \frac{Count(\alpha a)}{Count(\alpha)P(a)} \tag{6}$$

and

$$bdep(u) = \frac{Count(a\beta)}{Count(\beta)P(a)} \tag{7}$$

Where the probability of a letter a: P(a) is approximated by its normalized frequency in a corpus.

$$P(a) = \frac{Count(a)}{\sum_{b \in \mathcal{A}} Count(b)} \tag{8}$$

Where $\mathcal{A}$ is the alphabet. Count(α) expresses how often an *n*-gram α occurs in the corpus.

Algorithm 3: Morphological Segmentation

input : A word $w = a_0 a_1 \dots a_n$
output: Cutting Points
foreach a_i *where* $0 < i \leqslant n$ **do**
  if *fdep(*$\#\dots a_{i-1}$*)-bdep(*$a_{i-1}\dots a_n\$$*)* > 0 **and** *fdep(*$\#a_0\dots a_i$*)-bdep(*$a_i\dots a_n\$$*)* < 0 **then**
    add i to the cutting points

Example: Table 2 is a simulation of Algorithm 3 on the word "*unbreakable*". The second letter "*n*" depends on the prefix "*#u*" more than it does on the suffix "*breakable$*", where the third letter "*b*" depends on the suffix "*reakable$*" more than it does on the prefix "*#un*". This change of the dependence direction makes the point "*un|breakable*" a cutting point. The same logic applies to the seventh letter "*k*" and the eighth letter "*a*".

B. Morphological Relatedness

An MR assessment is expected to fulfil two assumptions.

- The longer the shared sequences are, the higher the relatedness should be.
- Words sharing core letters are much more related to each other than are words sharing only non-core letters.

The following shows that an adaptation of a string alignment algorithm might be the answer.

1) Relatedness Algorithm Algorithm 4 is an adaptation of the Algorithm 1, where a word A contains α *n*-grams and a word B contains β *n*-grams. Instead of talking about a cost, the term gain will be used which fits well with the aforementioned assumptions.

Algorithm 4: Morphological Relatedness

Input : Two words $A = a_1 a_2 \dots a_\alpha$ and $B = b_1 b_2 \dots b_\beta$
Output: The Needleman-Wunsch similarity between two words A and B

M: matrix$[0 \dots \alpha, 0 \dots \beta]$
$M[0,0] \leftarrow 0$
for $i \leftarrow 1$ *to* α **do**
  $M[i,0] \leftarrow M[i-1,0] + gain_{del}(a_i)$
for $j \leftarrow 1$ *to* β **do**
  $M[0,j] \leftarrow M[0,j-1] + gain_{ins}(b_j)$
for $i \leftarrow 1$ *to* α **do**
  for $j \leftarrow 1$ *to* β **do**
    if $a_i = b_j$ **then**
      $M[i,j] \leftarrow \max(M[i-1,j-1]+gain_{match}(a_i), M[i-1,j]+gain_{del}(a_i), M[i,j-1]+gain_{ins}(a_i))$
    else
      $M[i,j] \leftarrow \max(M[i-1,j-1]+gain_{subs}(a_i,b_j), M[i-1,j]+gain_{del}(a_i), M[i,j-1]+gain_{ins}(b_j))$
return $M[\alpha,\beta]$

2) Computation of the Relatedness The following describes how is the gain computed:

1. $gain_{match}(a)$ = + $\frac{1}{freq(a)}$: When two letters match.
2. $gain_{subs}(a,b)$ = - $\frac{1}{freq(a)}$: In case of substitution, the letter a is the one with the lowest frequency.
3. $gain_{del}(a)$ = - $\frac{1}{freq(a)}$: In case of deletion, the letter a is the deleted letter.
4. $gain_{ins}(a)$ = - $\frac{1}{freq(a)}$: In case of insertion, the letter a is the inserted letter.

where $freq(a)$ is the normalized frequency of the letter a:

$$freq(a) = \frac{Count(a)}{\sum Count(.)} \tag{9}$$

where . denotes any letter of the alphabet.

Table 2: Example of morphological segmentation

i	u	$fdep(u)$	v	$bdep(v)$	Difference	Direction
1	#**u**	0. 46	**u**nbreakable$	30.97	-30. 51	↓
2	#u**n**	6. 36	**n**breakable$	5. 84	0. 52	↑
3	#un**b**	1. 35	**b**reakable$	46. 70	-45. 35	↓
4	#unb**r**	1. 94	**r**eakable$	10. 45	-8. 51	↓
5	#unbr**e**	3. 11	**e**akable$	5. 01	-1. 90	↓
6	#unbre**a**	6. 79	**a**kable$	1. 68	5. 11	↑
7	#unbrea**k**	34. 71	**k**able$	1. 50	33. 21	↑
8	#unbreak**a**	5. 09	**a**ble$	7. 38	-2. 29	↓
9	#unbreaka**b**	46. 70	**b**le$	9. 89	36. 81	↑
10	#unbreakab**l**	9. 72	**l**e$	2. 76	6. 96	↑
11	#unbreakable	10. 02	**e**$	1. 15	8. 87	↑

Table 3: Example of the computation of morphological relatedness

		ك	ا	ت	ب
	0	-38.450	-46.760	-62.036	-85.728
ك	-38.450	38.450	30.140	14.864	-8.828
ت	-53.727	23.173	23.174	45.417	21.725
ا	-62.036	14.864	31.483	37.107	21.725
ب	-85.728	-8.828	7.791	13.415	**60.799**

Example: The MR between two words A= "كتاب" ([kitaAb]: book) and B= "كاتب" ([kaAtib]: writer) is calculated as shown in Table 3, where the frequencies used in this example are shown in Table 4. To fill each cell in the matrix, the maximum value among adjacent cells plus the gain of the underlying operation is picked. The three adjacent cells are those on upper left corner side ($M[i-1, j-1]$), up side ($M[i-1, j]$) and left side ($M[i, j-1]$). The resulting value at the last cell $M[4][4]$ indicates the MR between the two words "كتاب" and "كاتب".

The value in the cell $M[2][2]$ is the relatedness between the two substrings "كت" and "كا". It corresponds to a substitution of the letter "ا" ([A]: 1^{st} Arabic letter) with "ت" ([t]: 3^{rd} Arabic letter). The gain $gain_{subs}$("ا","ت") used to evaluate the cell $M[2][2]$ value is the inverse of the frequency of the letter "ت", which is higher than the inverse of the frequency of the letter "ا". The gain $gain_{del}$("ت") uses the frequency of the deleted letter "ت". The last gain $gain_{ins}$("ا") is the inverse of the frequency of the inserted letter "ا". The value of the cell $M[2][2]$ is calculated as follows:

$$\begin{aligned} M[2][2] &= \max(M[1,1] + gain_{subs}(\text{"ت"}, \text{"ا"}), M[1,2] \\ &\quad + gain_{del}(\text{"ت"}), M[2,1] + gain_{ins}(\text{"ا"})) \\ &= \max(38.45 - 15.27, 30.14 - 15.27, 23.17 - 8.31) \\ &= 23.17 \end{aligned}$$

As the cell corresponds to a match the gain is a positive used to evaluate the value of $M[2][3]$ is the inverse of the frequency of the matched letter "ت" ([t]: 3^{rd} Arabic letter) for the three operations: matched, deletion and insertion. Because the cell $M[2][3]$ corresponds to the letter "ت" at both $M[2]$ and $M[3]$. $M[2][3]$ is calculated as follows:

$$\begin{aligned} M[2][3] &= \max(M[1,2] + gain_{match}(\text{"ت"}), M[1,3] \\ &\quad + gain_{del}(\text{"ت"}), M[2,2] + gain_{ins}(\text{"ت"})) \\ &= \max(30.14 + 15.27, 23.17 - 15.27, 14.86 - 15.27) \\ &= 45.41 \end{aligned}$$

Table 4: A sample of frequencies of Arabic letters

Letter	Count	Frequency	1/Frequency
$\sum$.	3,546,432	-	-
ا	426,784	0.120	8.309
ب	149,688	0.042	23.692
ت	232,146	0.065	15.276
ك	92,234	0.026	38.450

V. Tests and Results

This section firsts describes the test settings then discusses the results.

A. Test Dataset

The morphological segmentation is fed with a corpus of plain classical Arabic texts[2]. It contains around 1M distinct (122M in total) words with an average size of 6.22 letters per word. The morphological segmentation step produced 596,356 distinct (1,116,919 in total) stems with an average size of 5.94 letters per stem. The resulting stems were then used to feed the computation morphological relatedness.

B. Performance Metrics

1) Morphological Segmentation Three samples of 100 words each are randomly picked out of the whole segmented corpus. The results of these approaches are evaluated manually by using three metrics: recall, precision and F-measure of the cutting points:

- Recall measures how many correct points are found among all the existing correct points.

$$Recall = \frac{\textit{correct points in the result}}{\textit{all correct points in the dataset}} \quad (10)$$

- Precision measures how many points are actually correct among all the points the algorithm found.

$$Precision = \frac{\textit{correct points in the result}}{\textit{all found points in the result}} \quad (11)$$

- F-measure measures the average of the recall and the precision.

$$F - measure = 2 * \frac{Recall * Precision}{Recall + Precision} \quad (12)$$

[2]https://sourceforge.net/projects/classical-arabic-corpus

2) MR Computation Evaluation The MR computation is ran over the whole set of stems resulting from the previous phase. For every stem, the ten others stems with the highest MRs are picked. Then two metrics are used to assess the performance of the MR computation.

- Normalized Discounted Cumulative Gain (nDCG) measures the number of related stems returned in the results and their order among all existing related stems. The higher is the nDCG, the more related and the better ordered are the results. Related stem means a stem which is derived from the same root. For this purpose, the binary value $rel(s, s_i)$ is defined, which is set to 1 when s and s_i are derived from the same root (relevant to each other) [6] and set to 0 otherwise.

$$nDCG(s) = \frac{DCG(s)}{IDCG} \tag{13}$$

where:

$$DCG(s) = rel(s, s_1) + \sum_{i=2}^{k} \frac{rel(s, s_i)}{log_2(i)} \tag{14}$$

$$IDCG(s) = 1 + \sum_{i=2}^{k} \frac{1}{log_2(i)} \tag{15}$$

- R-precision measures the number of related stems among all stems that appear at the k^{th} position of the returned stems. The higher is the R-precision, the better the performance is. This study supposes k is equal to 10.

$$R - precision = \frac{\sum_{i=1}^{k} rel(s, s_i)}{k} \tag{16}$$

C. N-grams vs Letters

The generalization of the approach to make the unit of comparison *n*-grams rather than letters is investigated. Three cases are tested: unigram (one letter as the original version), bigram (The unit of comparison is two letters) and trigram (three letter).

D. Noise

Typos are known to be a source of errors and a single kind of typo may influence the performance of the whole system. The study seeks to go deeper and investigate the extent of the influence of the misspellings on the performance of the MR computation. The study first runs the computation on a test set without any filtration (normal), then runs it on a test set with simple normalization rules that eliminate the effect of a number of common mistakes (Typos-free):

1. "ء" ([´]) confused with "ا" ([A]).
2. "ى" ([Y]) confused with "ي" ([y]).
3. "ـهـ" ([h]) confused with "ة" ([p]).

Table 5: Morphological segmentation results

	Raw	Nonempty affix	Thresholded
Precision	81.21±2.76	78.73±4.97	89.36±5.18
Recall	48.11±1.32	74.96±2.44	78.69±2.30
F-measure	60.62±1.47	76.80±2.54	83.69±3.49

E. Test Process

1) Morphological Segmentation The three metrics are recorded on each of the three sampling settings:

- Raw: Results sample is picked randomly with no restriction.
- Nonempty affix: Samples are picked randomly only among words for which morphological segmentation has carried out at least one cutting point. For a number of words, the segmentation simply did not identify any cutting point. For most of these cases, it was because of the scarcity of the stem or stem.affix remaining combination. The study then tries to assess the impact of such cases on the performance of the segmentation and the accuracy of the identified cutting.
- Thresholded: An additional filter is applied to the *nonempty affix* sample, where a segmentation is picked only if the affix reappears in more than 1,000 other segmentations.

The results of the raw sample are the lowest on Table 5. The two obvious causes might be the irregularities in Arabic morphology and the typos in the test set. The latter is confirmed by results obtained when the sample is restricted to the words with a relatively high frequency (thresholded) as shown on Fig. 1.

Another reason which affects the performance is the lack of a dataset. The proposed approach works wholly with the unsupervised method and depends only on the count of words in the corpus. However, it is difficult to include all possible derivatives in the corpus. For example, the derivative "الإجحافات" ([Al◦<ij◦HaAfaAt])the prejudices appears one time in the dataset; that is, there is no other derivative that appears in the dataset without the prefix "ال", for instance. Indeed, the word "إجحافات" would be more dependent on the prefix "ال" and the proposed algorithm does not learn that the prefix "ال" can be cut off from the word "الإجحافات". This problem will be removed if the word "إجحافات" is added into the dataset. The approach then finds the segmentation position "ال"|"إجحافات".

Running the morphological segmentation over the whole corpus resulted in around 1,805,231 morphemes and 596,358 distinct morphemes with an average size of 5.94 letters per morpheme.

It is worth noting that the objective of the study is to address the classical Arabic (CA) words. To the best of the authors' knowledge, there is no suitable gold standard for CA. The study should build itself a set of words and then proceed to the manual segmentation of three different randomly picked samples for each setting.

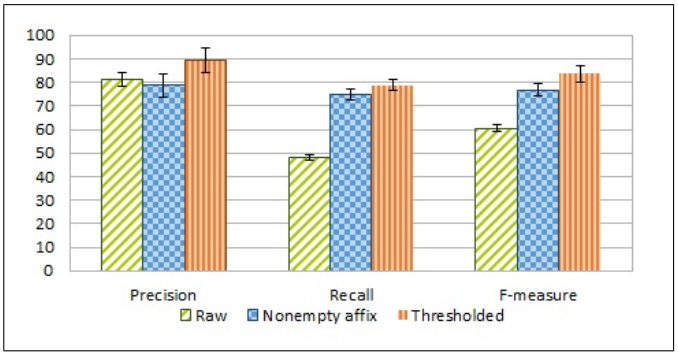

Figure 1: Results of morphological segmentation

Table 6: Morphological relatedness results

		Unigram	Bigram	Trigram
Normal	nDCG	80.38±1.86	84.26±1.11	81.06±1.02
	R-precision	79.33±1.10	81.03 ±0.75	77.90±0.60
Typos-free	nDCG	86.59 ±0.94	87.47 ±0.66	83.01 ±0.15
	R-precision	83.43±1.11	81.63 ±1.25	79.03 ±1.53

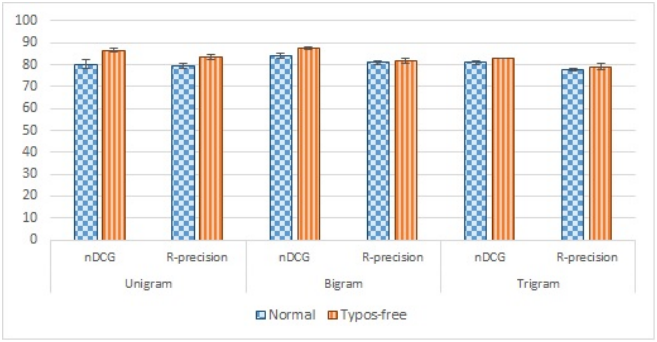

Figure 2: Results of morphological relatedness

2) Morphological Relatedness Obviously, checking the whole test set manually is impractical. Thus, the study opts for a sampling approach to approximate the performance. Three samples of 100 words each are randomly picked out of the whole corpus. Along with the nDGCG and r-precision computed for every sample, the *average* and the *standard deviation* are recorded. This process is repeated for the different settings explained earlier.

Values in Table 6 and Fig. 2 clearly indicate a high performance of around 80% in all cases. The highest value of nDCG occurs with *bigram*s in the *typos-free* case. This confirms the utility of such simple typos handling. This also suggests that the performance might be enhanced further if a heavier typos filtration is applied.

The combination of letters in bigrams and trigrams shows positive and negative effects. The positive effect is the lowering of the frequencies, which increases the influence of the core (root) letters and widens the differences between the frequencies of the *n*-grams formed of core letters and the frequencies of the *n*-grams formed of non-core letters *n*-grams. A good distinction is then made between the two classes of letters. The negative effect occurs when the combination becomes less common and fails to capture the distinct classes. Instead, it may mix up letters from both core and non-core letters. The results show that the bigram is a good compromise. The standard deviation is a good indication that the results are reliable.

The changes introduced to the original version [6] were fruitful and the performance increased considerably. It is worth mentioning that the evaluation reported in [6] was on the five top results. The proposed approach is in the top ten and the values are still higher. Of course, one of the factors that boosted the performance is the stemming. However, given that the stemming was also totally unsupervised, this is another proof that an end to end unsupervised method can handle a complex morphology language such as Arabic.

VI. Conclusion and Future Work

The concept of the Morphological Relatedness seems promising in the area of the unsupervised processing of languages. Even more, it shows a good handling of a complex language, such as classical Arabic. The purpose of the work reported in this article was to enhance the computation of the MR originally suggested in [6] without falling into the trap of human supervision. The study is able to overcome the problem of the long prefixes and suffixes by introducing an unsupervised morpho-segmentation. The study also handles the unclear boundaries between the frequencies by extending the comparison to bigrams.

The study commits itself to keep any processing human independent and as generic as possible. The open issues are diverse, and the explorable are numerous. A few of them are listed:

- Does the morphological relatedness perform well when generalized to upper levels such as the morphology of partially structured texts?
- Can the computation of the MR be sped up by using an index?
- Can $MR(w_1, w_2)$ be derived from $MR(w_1, w_3)$ and $MR(w_2, w_3)$?

Acknowledgement

This work was supported by King Abdulaziz City for Science and Technology (KACST) Project number AT-200-34.

References

[1] N. Habash, A. Soudi, and T. Buckwalter, "On Arabic transliteration," in *Arabic Computational Morphology*, ser. Text, Speech and Language Technology, A. Soudi, A. v. d. Bosch, and G. Neumann, Eds. Springer Netherlands, 2007, no. 38, pp. 15–22.

[2] I. Al-Sughaiyer and I. Al-Kharashi, "Arabic morphological analysis techniques: A comprehensive survey," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 3, pp. 189–213, Feb. 2004.

[3] M. Popovie and P. Willett, "The effectiveness of stemming for natural-language access to Slovene textual data," *Journal of the American Society for Information Science*, vol. 43, no. 5, pp. 384–390, Jun. 1992.

[4] S. Wintner, "Morphological processing of semitic languages," in *Natural Language Processing of Semitic Languages*, I. Zitouni, Ed. Berlin, Germany: Springer Berlin Heidelberg, 2014, pp. 43-66.

[5] P. Nugues, *An Introduction to Language Processing with Perl and Prolog*. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2006, pp. 1, 7, 23, 112, 123-129.

[6] A. Khorsi, "On morphological relatedness," *Natural Language Engineering*, vol. 19, no. 4, pp. 537–555, Oct. 2013.

[7] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, Mar. 1970.

[8] C. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*, online ed. Cambridge, England: Cambridge University Press, 2008, pp. 58-60.

[9] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on Strings*, 1st ed. Cambridge University Press, Apr. 2007.

[10] H. Abdel Ghafour, A. El-Bastawissy, and A. F. Heggazy, "AEDA: Arabic edit distance algorithm towards a new approach for Arabic name matching," in *Proceedings International Conference on Computer Engineering & Systems (ICCES)*, Cairo, Egypt, Dec. 2011, pp. 307–311.

[11] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707 – 710, Feb. 1966.

[12] W. Gomaa and A. Fahmy, "Short answer grading using string similarity and corpus-based similarity," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 3, no. 11, pp. 115–121, Nov. 2012.

[13] S. Mustafa and Q. Al-Radaideh, "Using n-grams for Arabic text searching," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 11, pp. 1002–1007, Sep. 2004.

[14] S. Mustafa, "Character contiguity in n-gram-based word matching: The case for Arabic text searching," *Information Processing & Management*, vol. 41, no. 4, pp. 819–827, Jul. 2005.

[15] ——, "Word-oriented approximate string matching using occurrence heuristic tables: A heuristic for searching Arabic text," *Journal of the American Society for Information Science and Technology*, vol. 56, no. 14, pp. 1504–1511, Dec. 2005.

[16] D. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, Jun. 1975.

[17] F. Peng and D. Schuurmans, "A hierarchical EM approach to word segmentation," in *In 6th Natural Language Processing Pacific Rim Symposium (NLPRS2001) Shai Fine, Yoram Singer, and Naftali Tishby.1998*, 2001, pp. 475–480.

[18] M. Melucci and N. Orio, "A novel method for stemmer generation based on hidden markov models," in *Proceedings of the twelfth international conference on Information and knowledge management*. ACM, 2003, pp. 131–138.

[19] K. ur Rehman and I. Hussain, "Unsupervised morphemes segmentation," 2005.

[20] M. Creutz and K. Lagus, "Unsupervised models for morpheme segmentation and morphology learning," *ACM Transactions on Speech and Language Processing*, vol. 4, no. 1, pp. 1–34, Jan. 2007.

[21] S. Keshava and E. Pitler, "A segmentation approach to morpheme analysis," in *Working Notes for the CLEF Worksh 2007*, Hungary, 2007, pp. 1–4.

[22] D. Bernhard, "Simple morpheme labelling in unsupervised morpheme analysis," in *Advances in Multilingual and Multimodal Information Retrieval*. Springer, 2008, pp. 873–880.

[23] S. Bordag, "Unsupervised and knowledge-free morpheme segmentation and analysis," in *Advances in Multilingual and Multimodal Information Retrieval*. Springer, 2008, pp. 881–891.

[24] O. Eroglu, H. Kardes, and M. Torun, "Unsupervised segmentation of words into morphemes," 2009.

[25] H. Poon, C. Cherry, and K. Toutanova, "Unsupervised morphological segmentation with log-linear models," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 209–217.

[26] H. Hammarström and L. Borin, "Unsupervised learning of morphology," *Computational Linguistics*, vol. 37, no. 2, pp. 309–350, 2011.

[27] J. Naradowsky and K. Toutanova, "Unsupervised bilingual morpheme segmentation and alignment with context-rich hidden semi-Markov models," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 895–904.

[28] B. Can and S. Manandhar, "Methods and algorithms for unsupervised learning of morphology," in *Computational Linguistics and Intelligent Text Processing*. Springer, 2014, pp. 177–205.

[29] R. Falk and M. Bar-Hillel, "Probabilistic dependence between events," *The Two-Year College Mathematics Journal*, vol. 14, no. 3, pp. 240–247, Jun. 1983.