

Arabic Morphological Analysis: a New Approach

Riad Sonbol
Informatics Department, HIAST
Damascus, SYRIA
rsonbol@gmail.com

Nada Ghneim
Informatics Department, HIAST
Damascus, SYRIA
nghnem@scs-net.org

Mohammed Said Desouki
Informatics Department, HIAST
Damascus, SYRIA
msdos@scs-net.org

Abstract—Morphological analysis is an important step in the process of Arabic language processing which became a must in today's IT development. We present here a new approach for Arabic root extraction. Our algorithm outperforms most other morphological analysis algorithms; we achieved more reliable results with better performance and minimal storage. Our Morphological analyzer provides different levels of reliability and performance to support the needs of different applications. It also provides a separate stage which can be plugged easily in any other Morphological analyzer to improve its performance, and with no negative effect on its reliability.

Keywords- *Morphological Analyzer; O-Letters Detection Algorithm; Arabic Language Processing; Root Extraction.*

I. INTRODUCTION

Morphological analysis is an important step in Arabic language processing because of the complex morphological structure of Arabic. Moreover, morphological analysis is a basic step in almost all Natural Language Processing (NLP) applications including text mining, information retrieval (IR), machine translation, and automatic summarization. Therefore, the time spent on creating an "intelligent" reliable and efficient Arabic morphological analyzer is justified by its reuse in many of these applications.

Our aim in this paper is to present a new root-extraction algorithm which outperforms other approaches in both reliability and performance. We try to provide an approach that can be used in both IR and NLP applications in an efficient way.

In section 2 we will present an overview on the previous work in Arabic Morphology. Section 3 will provide a description of our approach for morphological analysis. In section 4, we provide some experiments to evaluate the approach. We conclude our study in section 5.

II. PREVIOUS WORK

Several approaches have been proposed for Arabic stemming, the most common approaches are: light stemmers and Root-based stemmers.

Most light stemmers [1] [4] [6] are based on the same idea: delete the most frequent prefixes and suffixes to achieve the stem. These stemmers do not interest in producing the Arabic root, because they are directed to serve IR applications.

Root-based stemmers try to find the correct morphological analysis. Many morphological analyzers have been developed like Khoja stemmer [5], ISRI stemmer [7], and Buckwalter Morphological analyzer [3].

Khoja and Garside produce an effective stemmer, this stemmer removes prefixes and suffixes, then matches the remaining word against the patterns to extract the root. Finally it checks whether the extracted root is a valid root using a list of Arabic roots. Khoja stemmer is considered as a high performance stemmer. However, this stemmer makes some problems especially when removing the prefixes and suffixes which lead to wrong solutions or a failure in stemming operation. Also it generates wrong roots for words which contain *Ebdal* case like 'اصطَلح'. This stemmer gives one solution for each word, so it ignores other possible solutions. This fact makes the use of this stemmer in NLP applications less effective, because in such applications we need to provide all possible solutions. For example, the Arabic word 'نقول' has two possible roots: 'قول' (*say*) and 'نقل' (*transfer*). But according to Khoja stemmer the root is always 'نقل' (*transfer*).

Taghva, Elkhoury, and Coombs propose the ISRI stemmer similar to Khoja one, but without a root dictionary. This stemmer performs equivalently to the Khoja one as well as light stemmers in IR task.

Buckwalter's morphological analyzer follows another approach, the stemmer produces all possible morphological analyses by using three Arabic lexicon files: prefixes dictionary (299 entries), suffixes dictionary (618 entries), and stems dictionary (82158 entries), and three compatibility tables representing pairs of compatible morphological categories (prefixes-suffixes, prefixes-stem, and stem-suffixes). It provides high reliable results which leads it to be one of the most useful analyzer in NLP tasks. But it is not a suitable choice for IR tasks, where the performance is an important metric.

III. OUR APPROACH

A. Main Idea:

The principal idea in our algorithm is based on the encoding of Arabic letters by a new code that: (1) preserves morphologically useful information, (2) and simplifies its capturing toward retrieving the root.

We can divide our algorithm into two main stages: In the first stage, we will try to extract all possible roots without using any dictionary. The second stage is designed to improve the reliability, and to solve some special problems such as *Ebdal*, and *Ealal*.

B. Algorithm Detail:

Stage1. Extracting Possible Roots:

1) Normalization:

Before stemming, we normalize the Arabic word by applying the following steps:

- Remove diacritics, and the *Shadda*.
- Replace all distinct forms of *Hamza* with (أ).
- Replace *Madda* (آ) with *Hamza* and *Alef* (أ).
- Replace *Alef Maksura* (ى) with *Alef* (أ).

2) Encoding:

In this step, we will encode the Arabic letters in our new coding. This coding is based on six symbols {O, P, S, PS, U, A} representing six groups of letters each of which shares certain characteristics:

- O:** Original letters. These letters are **surely part of the root**. They are:
{ت، ج، ح، خ، د، ذ، ر، ز، ش، ص، ض، ط، ظ، ع، غ، ق}.
- P:** Prefix letters. These letters **can** be added only in the prefix part. They are:
{ب، ف، س، ل}
- S:** Suffix letters. These letters **can** be added only in the suffix part. They are:
{ه} only *Haa*
- PS:** Prefix-Suffix letters. These letters **can** be only added in both sides of the word i.e. in the suffix part or in the prefix part. They are:
{ك، م، ن}
- U:** Uncertain letters. These letters **can** be added anywhere in the word. They are:
{ت، و، ي، ا، أ}
- A:** Added letters. These letters are **always** considered additional letters. They are:
{ة} only *Taa Marbuta*.

We add some improvements to the last coding by adding the position conditions when processing some letters.

There is a maximum index for each letter when it is found in a prefix. For example, the letter *Baa* 'ب' **can** be part of a prefix only if it is found in the first three letters like 'ويالحرف'. And it **can not** be part of the prefix if it is not found in the first three letters like 'والبحر'.

We can apply the same idea for suffixes by using the minimum index in suffixes for each letter, but we found that this idea is not efficient for S letters (except *Haa* 'ه').

Table1 presents the maximum index in prefix and the minimum index in suffix for certain letters (P, S, and PS letters). The symbol '*' indicate that it is not effective to put such condition for this letter.

Table 1- statistics about the position of some letters in the word

The letter	The maximum index in prefixes	The minimum index in suffixes
<i>Baa</i> 'ب'	3	–
<i>Lam</i> 'ل'	5	–
<i>Seen</i> 'س'	4	–
<i>Faa</i> 'ف'	2	–
<i>Haa</i> 'ه'	–	3
<i>Kaf</i> 'ك'	3	*
<i>Noon</i> 'ن'	*	*
<i>Meem</i> 'م'	*	*

In addition, the maximum length for any prefix or suffix is ($len-2$) where len is the length of the word. For example, this maximum is satisfied in 'فانسد' where the prefix is 'فانـ' and in 'تتهم' where the suffix is 'تـهم'.

So, **the position rules** are:

- We consider the letter *Baa* a *Prefixes letter* if it is found in the first three letters, else it is an *Original letters*. For example, *Baa* is an original letter in the Arabic word 'والبيوت'.
- We consider the letter *Faa* 'ف' a *Prefixes letter* if it is found in the first two letters, else it is an *Original letters*. For example, *Faa* is an original letter in the Arabic word 'الفتح'.
- We consider the letter *Seen* 'س' a *Prefixes letter* if the next letter is one of "أنيت" letters (*Hamza, Noon, Yaa, and Taa*), else it is an *Original letters*. For example, *Seen* is an original letter in the Arabic word 'سيحاب'.
- We consider the letter *Lam* 'ل' a *Prefixes letter* if it is found in the first five letters, else it is an *Original letters*. For example, *Lam* is an original letter in the Arabic word 'الشاملون'.
- We consider the letter *Haa* 'ه' a *Suffix letter* if it is found in the last three letters, else it is an *Original letters*. For example, *Haa* is an original letter in the Arabic word 'يهاجر'.
- We consider the letter *Kaf* 'ك' a *Prefix letter* if it is found in the first three letters, else it is a *Suffix letters*. For example, *Kaf* is not a prefix letter in the Arabic word 'أهلك', we consider it as suffix letter.

- We consider **any Prefixes letter** when it is not founded in the first (len-2) letters, an *Original letters*. For example, *Baa* is an original letter in the Arabic word 'فريه' (note that the first position rule can not discover such case).
- We consider **any Suffix letter** when it is not founded in the last (len-2) letters, an *Original letters*. For example, *Haa* is an original letter in the Arabic word 'يهرب' (note that the fifth position rule can not discover such case).

So, after applying the last coding, we have an encoded word which is more useful for morphological analysis. It can find the root directly in some cases, because if we have 3 O-Letters (or more) in the encoded word, then these letters are considered root letters, and we can terminate the process here. Else, we have to apply the next step.

For example; we can extract the root of the Arabic word 'التطبيقية' only by using this step (Figure-1), but we can not do that in the case of 'المعهد' (Figure-2):

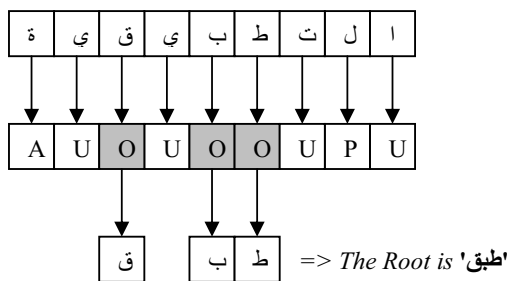


Figure 1- applying Encoding step on 'التطبيقية'

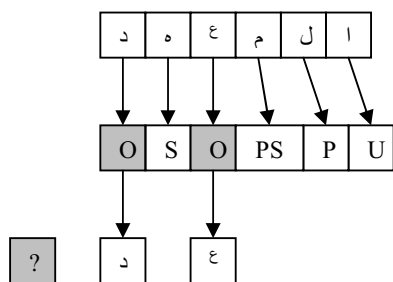


Figure 2- applying Encoding step on 'المعهد'

3) Apply Transformation Rule:

In this step, we will apply **transformation rules** between groups to obtain maximum number of original letters we can discover. Note that when using the words "before" and "after" in the transformation rules, we consider the way of Arabic reading (right to left).

Transformation Rules:

- R1) Change each 'P' after 'O' to 'O'.
- R2) Change each 'S' before 'O' to 'O'.
- R3) Change each 'P' after 'S' to 'O', and each 'S' before 'P' to 'O'.
- R4) Change each 'PS' before 'P' to 'P'.
- R5) Change each 'PS' before 'O' to 'P'.
- R6) Change each 'PS' after 'S' to 'S'.
- R7) Change each 'PS' after 'O' to 'S'.

These rules are concluded from the properties of our encoding. For example, we can not have P-Letter after O-Letter, because if so, all letters before this P should be part of the prefix. So it is a contradiction, because we have O-Letter before it.

As the previous step, if we have 3 O-Letters in the encoded word, then these letters are considered root letters, and the process will terminate here, else we apply the next step.

We will call the last two steps (Encoding and Apply transformation Rules) **O-Letters Detection Algorithm**.

For example, we can extract the root of 'المعهد' immediately after this step (Figure-3).

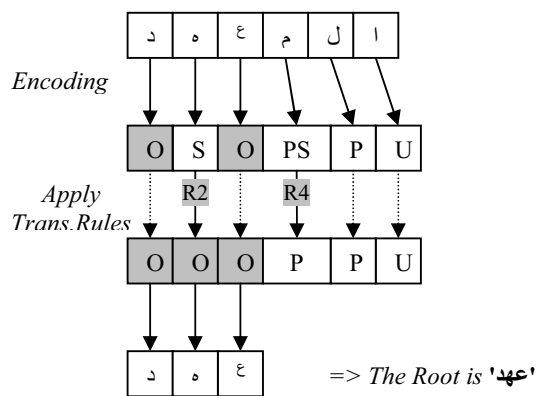


Figure 3- Applying O-Letters Detection Algorithm on 'المعهد'

4) Extract Possible Roots:

We have to apply this step only if we have less than 3 O-Letters. In this step, we use the idea of traditional algorithms, but with the aid of our encoded word. Traditional algorithms store lists of Arabic prefixes, suffixes, and patterns. These algorithms delete prefixes and suffixes then use the pattern to extract the root from the reminder.

By aid of the encoded word, we achieve more reliable results without storing all these lists as:

- We replace the list of classical prefixes by a new adequate list (78 entries). We consider a prefix each string of letters that can be found before the first original letter in the word (which called *Faa AL-Feal*). For example, in the Arabic word 'المكاتب' we consider 'الم' as a prefix, while the classical prefix is only 'ال';
- We replace the list of suffixes by 4 equivalent conditions:
 - The encoded suffix contains only symbols: S, PS, or U.
I.e., there is no O or P in it.
 - If we have the letter *Meem* 'م' in the suffix, it should be one of the following suffixes:
{تم، كم، هم، ما}
 - If we have the letter *Taa Marbuta* 'ة' in the suffix, it should be one of the following suffixes:
{ة، ية، اتية، انية}
 - If we have the letter *Hamza* in the suffix, the previous letter of *Hamza* should be *Alef*'ا'.

For example, in the Arabic word 'التطبيقية' (Figure-1) the available suffixes are: 'ة' and 'ية'.

- We replace the list of patterns for triple roots by 6 equivalent forms (Table-2) using our coding.

Table 2- Encoded Patterns

The New Form	Equivalent Arabic Pattern
OOO	فعل
OOUO	فعال، فعيل، فعول،...
OOOU	فاعل، فيعل، فوعل، فتعل(افتعل-مفتعل)، ..
OOUOU	فاعيل، فاعول،..
OOUOO (uncommon)	فواعل.
OOUUO (uncommon)	فعالل.

Stage2. Improving The Reliability:

The aim of this stage is to enhance the reliability by deleting the wrong solutions, and adding missed ones (which represent special cases). In all steps, we consider the prefix, suffix, and encoded pattern conditions which are described in the last stage.

1) Root Existence Test:

In this test, we check the existence of the concluded root in the list of Arabic roots. If it does not exist, we should apply the next step (step 2) on it, and then delete this wrong root. We can implement this step in O(1) test

using a suitable data structure (two-dimension array) to store the roots.

2) Adding Special Solutions (Ebdal & Ealal):

We apply this step when we have a wrong root to check if it is wrong because of a special case. We process two special cases: *Ebdal* and *Ealal*.

We indicate that we have *Ebdal* problem according to a list of Arabic rules which is known in Arabic language references [10][11]. But we can not do that to solve *Ealal* problem because most of *Ealal* rules depend on the diacritics. So, we solve this problem as following:

When we have a weak letter (*Alef*, *Yaa*, and *Waw*) we replace this letter with the two other letters and check if the result is a valid root. If so, we add this root to the possible roots.

3) Shadda Test:

We try to duplicate letters when we have two letters after deleting a possible prefix and a possible suffix. We add the result to available roots if it is a valid root.

4) Non-Tri Roots Test:

We add these roots by applying the procedure which was shown in the previous stage (stage1-step4) but by using suitable Non-tri roots patterns. In this step, there is no difference whether we use the traditional patterns, or encoded ones. We will achieve the same results in the same efficiency, because of the reduced number of these patterns.

IV. EXPERIMENTS AND DISCUSSION

A. Corpus:

We conducted our experiments using two different corporas.

The first corpus consists of lists of word-root pairs (167162 pairs) extracted from HIAST Arabic lexical database [2] which covers the morphological categories in Arabic (verbs, nouns, infinitives, plural of nouns, analogous adjectives, exaggeration forms of active participle, non-standard plural ...etc). Because of this variety, this corpus has an important role to determine if a morphological analyzer acts with all morphological category in the same effectiveness.

The second corpus is a collection of 585 Arabic articles from different categories (policy, economy, culture, science and technology, and sport). This corpus consists of 377793 words.

B. Experiments:

a)Evaluating O-Letters Detection algorithm:

We found that by applying O-Letters Detection algorithm (which is: Encoding, and then applying Transform Rules) we achieve the root directly if we retrieve three O-Letters (or more). Here, we are going to find the probability of such case, and measure the accuracy of these solutions and the performance of this algorithm.

We apply the algorithm on the last two corpus and we find that we detect a satisfied number of O-Letters to retrieve the root in more than 30% of the Arabic word. (Table-3):

Table 3- The probability for each possible output for O-Letters Detection Algorithm

Corpus	No O-Letters	One O-Letter	Two O-Letters	Three O-Letters (or more)
Dictionary	3.2%	16.6%	41.3%	38.9%
Articles	8.5%	22.1%	36.9%	32.6%

To evaluate the accuracy of O-Letters Detection algorithm, we do two evaluations (Table-4):

In the first one, we use the word-root pairs to check the behavior of this algorithm with each morphological category. In general, the accuracy of these results is more than 95%.

In the second evaluation, we use the second corpus which represents Arabic texts. We find that the accuracy is about 99%.

We studied the errors in each case, and we conclude that:

- About 75% of errors are because of words from non tri-root. This result is normal, because when we have three O-Letters we conclude the root even if it is not tri-root.
- Some errors are a result of *Ebdal* problem.
- The rest errors are because of words from rarely pattern like 'فاععل'.

Table 4- O-Letters Detection Algorithm's Accuracy

Corpus	No. of words	Accuracy of Tri-root words	Accuracy of NonTri-root words	Total
Dictionary	167162	98.8%	59.8%	95.7%
Articles (manual evaluation)	3217	99.7%	68.7%	98.9%

The Performance:

This algorithm is very effective. Its performance is more than 55000 words/sec.

b)Evaluating The Reliability OF The Complete Approach:

We evaluate our approach using the last two corpus, we infer that for about 85% of words, all concluded morphological analyses are right. And for less than 1% of words all concluded analyses are wrong. In most of other cases, we have right analyses more then wrong analyses. On general, the accuracy is about 96%-98% (Table-5), and the average number of concluded analyses for a word is 1.4 (Figure-4).

Table 5- Our Approach's reliability

Corpus	No. of words	Failure (No Solution)	Accuracy
Dictionary	167162	0.4%	98.1%
Articles (manual evaluation)	1408	1.3%	95.5%

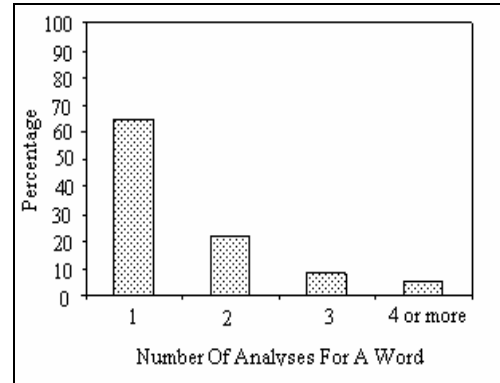


Figure 4- Number of analyses for each word

According to these results, this stemmer is considered a height reliable stemmer. We mention that khoja stemmer's accuracy is about 96% [5].

c)Evaluating The Performance OF The Complete Approach:

We compare the approach with two famous approaches: Khoja Stemmer, and Buckwalter Morphological analyzer. We apply the three algorithms on five graduated length files. We choose these files as length as we can ignore the effect of wasted time (initialization, reading stored data, opening files ...etc) to emulate the situation of IR.

We can see that our approach's performance is significantly outperforming other's performance.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a new approach in morphological analyses. The experiments clearly showed the effectiveness of the approach. The key idea in this efficiency is O-Letters Detection algorithm which reduces the number of false analyses by putting more constraints in the possible analyses.

The first stage in the approach can be considered a light stemmer. We expect that this stage outperforms other light stemmers in both performance and reliability. We did not compare it directly to light stemmer, but we compare it with Khoja stemmer.

The second stage is divided into several unrelated steps to provide extra levels of reliability. This stage raises the reliability to an important level which outperforms most others Morphological analyzers.

Our future work will be focused on four points:

- Evaluating our approach in IR task.
- Trying to find better solution for *Ealal* problem Which is considered the main reason for wrong solutions in the approach.
- Start building IR tools depending on this stemmer.
- Complete the work in the process of Arabic language processing. We plan to build an Arabic POS tagger as a next step.

RERERANCES

- [1] M. Aljlal and O. Frieder. 2002. On Arabic Search: Improving the retrieval effectiveness via a light stemming approach. In Proceedings of CIKM'02, VA, USA.
- [2] S. Attar, M. Bawab, and O. Dakkak. Arabic Lexical database. HIAST, Damascus. 2007.
- [3] T. Buckwalter. Buckwalter Arabic Morphological Analyzer Version 1.0. <http://www ldc.upenn.edu/Catalog/CatologEntry.jsp?catologId=LDC2002L49>
- [4] A. Chen, and F. Gey. Building an Arabic Stemmer for Information Retrieval, In TREC 2002.
- [5] S. Khoja. APT: An Automatic Arabic Part-of-Speech Tagger. PhD Thesis. Lancaster University, Computing Department. 2003.
- [6] L. S. Larkey, L. Ballesteros and M.E.Connell. 2002. Improving stemming for Arabic information retrieval: Light Stemming and co-occurrence analysis. In SIGIR 2002, Tampere, Finland: ACM, 2002.
- [7] K. Taghva, R. Elkhoury, J. Coombs. Arabic Stemming Without A Root Dictionary. 2005.
- [8] N. Thabet, Stemming the Qur'an. 2004.
- [10] A. Hassan, AL-Nahw al-wafi.
- [11] S. Al-afagani, Al-wajiz fi al-sarf al-arabi. <http://www.islamguiden.com/arabi/>

Table 6- Comparing the performance for Buckwalter, Khoja, and our approach

No. Of Words	Buckwalter (word/sec)	Khoja (word/sec)	Our Approach (word/sec)
19000	1496	3125	15289
78000	2956	8765	20781
186000	4473	9672	22111
262000	4969	9459	22649
350000	5469	9872	22803

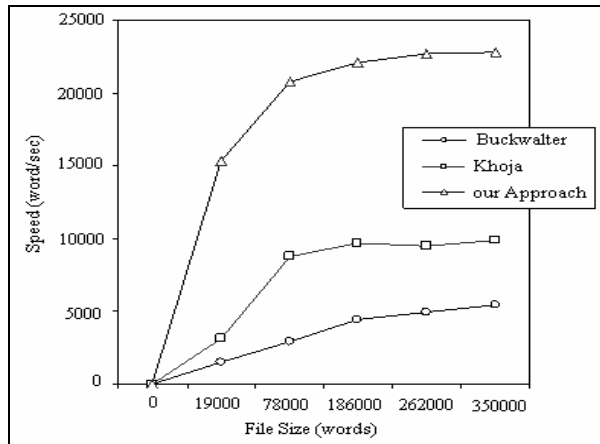


Figure 5- Comparing the performance for Buckwalter, Khoja, and our Approach

V. THE BENEFIT OF O-LETTERS DETECTION ALGORITHM

Our tests showed that O-Letters Detection algorithm produces high reliable results (about 99%) in very effective performance (55000 words/sec). It is faster than any other stemmer by 3 to 10 times.

So, we can improve the performance and the reliability of other stemmers by plugging this component in them as following:

For each word (w):

- Apply O-Letters Detection on this word.
- If we have 3 O-Letters or more so we extract the root directly which is the O-Letters.
- Else: Apply a Morphological analyzer algorithm.

We can apply the same procedure for light stemmers.