An Application Oriented Arabic Morphological Analyzer

Dr. Riad Sonbol **Dr. Nada Ghneim** **Dr. Mohammed Said Desouki***

Abstract

Morphological analysis is an important step in natural language processing and its various applications. Each kind of these applications needs a certain balance between: performance, accuracy, and generality of solutions (i.e. getting all possible roots); while we focus on performance with a "good" accuracy in Information retrieval applications, we try to achieve high accuracy in systems like pos-tagger and machine translation, and both high accuracy and high generality in systems like language learning systems and Arabic lexical dictionaries. In this paper, we describe our approach to build a flexible and application oriented Arabic morphological analyzer; this approach is designed to satisfy various requirements of most applications which need morphological processing. It also provides a separate stage (Original Letters Detection Algorithm) which can be plugged easily in any Other morphological analyzer to improve its performance, and with no negative effect on its reliability.

Keywords: Arabic Natural Language Processing; Arabic Morphological Analysis; Stemming; Original Letters Detection Algorithms, Application oriented system.

* Informatics Department, HIAST - Damascus, Syria
riad.sonbol@ hiast.edu.sy, nada.ghneim@hiast.edu.sy, said.desouki@hiast.edu.sy

1. Introduction

The importance of Arabic language processing tools has dramatically increased in the last decade because of the huge increment of Arabic digital content on the internet, and in internet users who speak Arabic. This fact increases the importance of creating language processing tools that can process this content, and interact with these users in better ways.

Morphological analysis is an important step in Arabic language processing because of the complex morphological structure of Arabic where we have infixes along with prefixes and suffixes. In addition, each prefix or suffix may have its own syntactical tag; this means that we have to use the result of the morphological analysis stage in higher stages of Arabic processing like POS-tagging, syntactical analysis. For example, the Arabic word "كتبتُهَا" – which means *I wrote it*- consists of a verb "كتب", a subject "تُ", and an object "ها", so it is very useful to analyze the word to its "simplest" parts before we determine its syntactical tags.

Moreover, morphological analysis is a basic step in various applications including text mining, information retrieval (IR), machine translation, automatic summarization, and Arabic learning systems. This diversity in applications is reflected as a variety in morphological analyzer's requirements; each application needs a certain balance between:

high accuracy,

high performance ,

generality in solutions i.e. finding all possible roots for each word.

In IR applications, for example, the most important metric is performance. This does not mean that accuracy and generality are not important, but that we can accept intermediate values for these two metrics to improve the performance. On the other hand, it is very important to have a high accurate morphological analyzer in applications like machine translation. Moreover, there are many applications which need both generality and accuracy, like Arabic learning systems where we need to find all possible results, and only correct ones.

Therefore, the effort spent on creating a reliable, efficient, and "flexible" Arabic morphological analyzer is justified by its reuse in many of these applications.

In this paper, we will present an approach to build a high flexible Arabic morphological analyzer. Our morphological analyzer can be adjusted to satisfy the requirements of all kind of applications with simple parameterization.

In section 2 we will present an overview of previous Arabic Morphology processing approaches. Section 3 will provide a description of Original Letters Detection Algorithm which we consider the main axe of our morphological analyzer. In section 4, we describe our flexible morphological analyzer. We provide our methodology to evaluate the approach in section 5, and we finally conclude our study in section 6.

2.Overview of Previous Works

Several approaches have been proposed for Arabic stemming; many papers survey these techniques (Al-Sughaiyer et al. 2004; Larkey et al, 2001; Darwish, 2002; Al-Fedaghi et al., 1989).

Buckwalter's morphological analyzer follows a dictionary-based approach (Buckwalter, 2002), It divides the Arabic word into all possible three parts: prefix, stem, and suffix. Then the analyzer checks the correctness of each segmentation using three Arabic dictionaries (prefixes, stems, and suffixes), and three compatibility tables representing pairs of compatible morphological categories.

This system provides high reliable results which leads it to be one of the most useful analyzer in NLP tasks. However, it needs a huge size of manually entered data which produces many limitations in performance, and generality of solutions.

To avoid such limitations, a number of approaches depend on a strong linguistic base (Al-Bawab et al., 1994). These approaches give, in general, high accurate results, but need long time to construct and very strong linguistic base.

Khoja and Garside (Khoja et al, 1999) developed an effective stemmer depending on

simpler linguistic rules, this approach (1) removes prefixes and suffixes, then (2) matches the remaining word against the patterns to extract the root, and finally (3) checks whether the extracted root is a valid root using an Arabic roots dictionary. Khoja stemmer is considered as a high performance stemmer, but it has some drawbacks such that removing prefixes and suffixes may lead to wrong solutions or a failure state. Moreover, it generates wrong roots for words which contain Ebdal cases like 'قَوَلَ' (*kawala)*. 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 that needs providing all possible solutions less effective.

3. Original Letters Detection Algorithm

The aim of Original Letters Detection Algorithm is to detect morphological information about each letter in an Arabic word to facilitate retrieving its root; this does not just include the detection of some original letters, i.e. the root letters, but also the detection of some additional letters, and the extraction of available morphological information about the rest of word letters.

We emphasize here that this algorithm is not a stemming algorithm, but an auxiliary algorithm to help Arabic stemming process to retrieve more accurate results in better performance. However, experiments showed that this algorithm can retrieve the root for more than one third of the Arabic words.

Original Letters Detection Algorithm provides in two stages very useful information, without using stored data. In the first stage, we will encode Arabic letters by their initial "morphological states" codes. In the second stage, we will apply "transformation rules" to change the current morphological states of letters to "better" and more precise ones to facilitate retrieving the root.

In next paragraphs, we will describe, in detail, the two stages of this algorithm.

3.1. First Stage (Initialization)

In the first stage, we start from the static property for each Arabic letter to determine its initial "morphological state". Each morphological state defines a limited number of possibilities for the concerned letter. In fact, we have two kinds of morphological states: deterministic, and ambiguous. In the deterministic cases, we know if the concerned letter belongs to the root or not, while we do not know this information in the ambiguous cases (we will see later that there are several levels of ambiguity).

For instance, we have a morphological state that says: the concerned letter surly belongs to the root. Of course, it is a deterministic case; we will call it O-Case. On the other hand, an ambiguous case says: if the concerned letter is an additional one, then it surly belongs to the prefixes side of the word, we will call this ambiguous state P-State. Another ambiguous case (the S-case) says: if the concerned letter is an additional one, then it surly belongs to the suffixes side of the word. And so on.

A complete description of all the defined morphological states is shown in the Table-1.

Table 1. Description of defined morphological states

State	Description
O	The concerned letter is surely part of the root.
A	The concerned letter is always considered as an additional letter.
P	The concerned letter can only be added in the prefix part.
S	The concerned letter can only be added in the suffix part.
T	The concerned letter can be added in both sides of the word, i.e. in the suffix part or in the prefix part.
U	The concerned letter can be added anywhere in the word.

After encoding each letter by its initial morphological state code (Table-2), we obtain an encoded word that can be more useful for morphological analysis. The root can be extracted directly in some cases, like when we have 3 Os (or more) in the encoded word, and in this case they represent root letters.

Table 2. The initial morphological state of Arabic letter

Arabic Letters	Initial State
ث، ج، ح، خ، د، ذ، ر، ز، ش، ص، ض، ط، ظ، ع، غ، ق	O
ة	A
ب، ف، س، ل	P
هـ	S
ك، م، ن	T
ت، و، ي، ا، أ	U

3.1.1. Example:

We can extract the root of the Arabic word 'التطبيقيّة' only by using this initial stage, but we can not do that in the case of the word 'المعهد' (Figure-1):

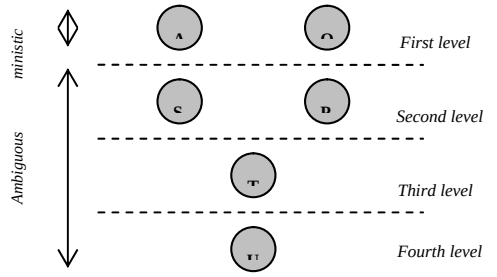

Figure 2. Classifying morphological states depending on their ambiguity

P and S states are from the second level of information, where we have two choices for the concerned letter: it is from the root letter, or it is an additional letter in one side of the word (the prefixes side for P state and the suffixes side for S state).

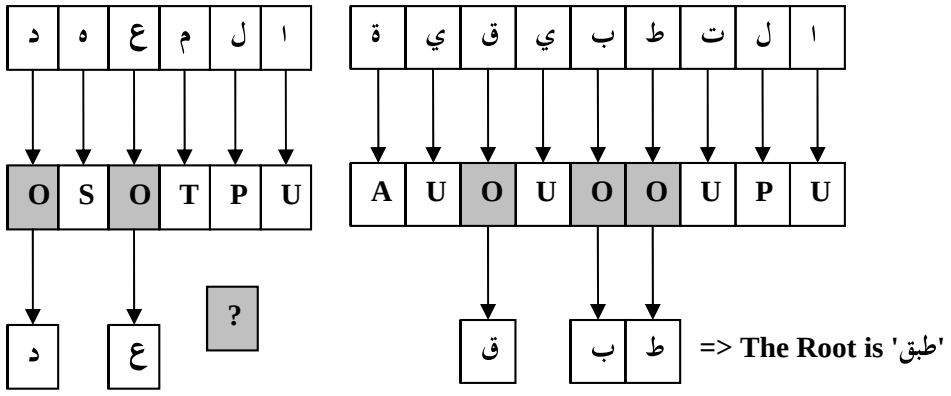

Figure 1 . Applying the initial step on the word 'التطبيقية' is sufficient to retrieve the root, but is not sufficient on the word 'المعهد'

3.2. Second Stage (Applying Transformation Rules)

In fact, the six morphological states are located at four levels of information. Each level has certain values of ambiguity:

O and A states are at the highest level of information (the deterministic level); because we can certainly determine that the concerned letter is from the root letters or not (zero-ambiguity).

T state is from the third level of information, where we have three choices for the concerned letter: it is from the root letter, it is additional letter in the prefixes side of the word, or it is from the suffixes side of the word.

U state is at the lowest level of information, where all choices are available: from the root letter, additional in the prefixes side,

additional in the suffixes side, or additional between the root letter.

In the second stage, we will apply "*transformation rules*" which consider the context of each letter in the word. The aim of these rules is to move word's letters each from its morphological state to a higher one with less ambiguity.

We use the following 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 'T' before 'P' to 'P'.

R5) Change each 'T' before 'O' to 'P'.

R6) Change each 'T' after 'S' to 'S'.

R7) Change each 'T' after 'O' to 'S'.

R8) Change the first letter to 'P' if it is not 'O' or 'A'.

R9) Change the last letter to 'S' if it is not 'O' or 'A'.

R10) *Cutting Rule:*

Let: n_r: Maximum length of the root.

n_o: Number of O letters in the encoded word.

n_p: Index of the first P letter.

n_s: Index of the last S letter.

So, for each letter at an index *i* where:

$$i \in [0, n_p] \cup [n_s, len] \quad \text{and}$$

$$\min(|i - n_p|, |i - n_s|) \geq (n_r - n_o)$$

Change it to 'A'.

These rules are concluded from the properties of each morphological state. 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.

[*] Note that when using the words "before" and "after" in the transformation rules, we consider the direction of Arabic reading (right to left).

3.2.1. Example:

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

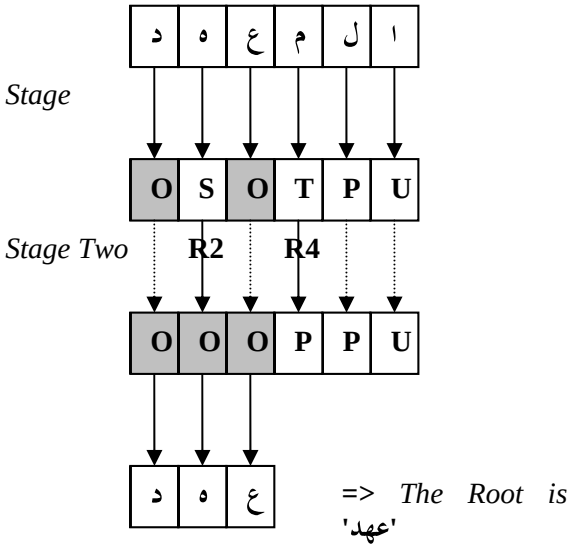

Figure 3. Applying Original Letters Detection Algorithm on the word 'المعهد'

3.3. Additional Improvement:

We add some improvements to the last stages by adding the *position conditions* (Sonbol et al., 2008) when processing some letters. In fact, there is a maximum index for each letter when it is situated in a prefix. For example, the letter *Baa* 'ب' can be part of a prefix only if it is situated in the first three letters like 'وبالحرف'. And it can not be part of the prefix if it is not situated 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').

Table-3 presents the maximum index in prefix and the minimum index in suffix for certain letters (P, S, and T letters). The symbol '*' indicates that it is not effective to put such condition for this letter.

Table 3. Statistics about the position of some letters in the word

The letter	The maximum index in prefixes	The minimum index in suffixes
Baa 'ب'	3	—
Lam 'ل'	5	—
Seen 'س'	4	—
Faa 'ف'	2	—
Haa 'ه'	—	3
Kaf 'ك'	3	*
Noon 'ن'	*	*
Meem '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 'فانسدّ' where the prefix is 'فانـ' and in 'سدّتهم' where the suffix is 'تهم'.

3.4. The Effectiveness of Applying Original Letters Detection Algorithm in finding roots:

To prove the effectiveness of Original Letters Detection Algorithm we will describe the distributions of Arabic words after applying this algorithm.

Our statistics on about 375000 words show that we can divide Arabic words after applying Original Letters Detection Algorithm to three parts (Figure-4):

30% of Arabic words are solved directly after applying Original letters detection algorithm,

30% of Arabic words represent particles, foreign words, and other static words,

40% of Arabic words are solved partially after applying Original letters detection algorithms, i.e. we found part of root's letters. Most of this part (about 25% of Arabic words) represents words that contain two original letters.

It is important here to emphasize that each original letter (O) adds more constraints on the process of root extraction; this means retrieving more accurate result.

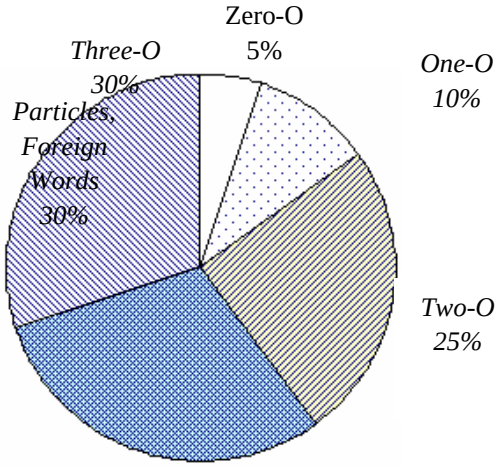

Figure 4-The Distribution of Arabic Words after Applying Original Letters Detection Algorithm

In addition, we evaluate Original Letters Detection Algorithm by checking the correctness of the Arabic words classified in the first part (where we find the root directly). The results (Table-4) show that we retrieve a correct root for about 99% of the words.

Table 4- Original Letters Detection Algorithm's Accuracy

No. of words	Accuracy of Tri-root words	Accuracy of NonTri-root words	Total Accuracy
3200	99.7%	68.7%	98.9%

After a study of errors states, we concluded that:

- **About 75% of errors are because of words from non tri-root.**
- **Some errors related to *Ebdal* problem.**

Depending on the last experiments, we can conclude that Original Letters Detection Algorithm is very useful to improve both the performance and the accuracy of a morphological analyzer.

4. Our Approach for an Arabic Morphological Analyzer

In this section, we will present our approach for a flexible and application oriented Arabic morphological analyzer. This flexibility is due to the Original Letters Detection Algorithm, and to other techniques that we use to achieve a balance between accuracy, performance, and generality.

4.1. Algorithm:

We summarize our approach by the following steps:

Step1: Check if the word is a particle or a foreign word using a dictionary of particles and common foreign words.

Step2: Apply normalization steps:

a. Remove diacritics, and the Shadda.

b. Replace all distinct forms of Hamza with (أ).

c. Replace Madda (آ) with Hamza and Alef (أا).

d. Replace Alef Maksura (ى) with Alef (ا).

Step3: Apply Original Letters Detection Algorithm.

Step4: Generate a bank of solutions which consist of each sequence of letters satisfying the following conditions:

a. Contains all Original letters (letters in the state O).

b. Does not contain any Additional letters (letters in the state A).

c. The pre-string is valid. We define the pre-string as the string of letters that are situated before the first root's letter in the word (which is called Faa AL-Fel). For example, in the word 'المكاتب' we consider 'الـ' as a pre-String, while the classical prefix is only 'الـ'.

d. The suf-string is valid. We define the suf-string as the string of letters that are found after the last root's letter in the word (which is called Lam AL-Fel). We consider a string as a valid suf-string if it satisfies the following conditions:

i. There is no letters in the state P.

ii. If we have the letter Meem 'م' in the suf-string, it should be one of the following suf-string: {تم، كم، هم، ما}.

iii. If we have the letter Taa Marbuta 'ة' in the suffix, it should be one of the following suffixes: {ة ، ية، اتية ، انية}

iv. If we have the letter Hamza in the suffix, the previous letter of Hamza should be Alef 'ا'.

Step5: Generate solutions that represent shadda case, elimination case, and non-tri roots **(optional step)**.

Step6: Correcting solutions in the bank of solution: we can make a balance between the three metrics (reliability, performance, generality) by applying the next **optional steps**:

a. Pattern existence test using a list of available patterns.

b. Root existence test using a list of available roots.

c. Apply Ebdal and Ealal rules: we do this step only for invalid roots to check if it is invalid because of a special case.

d. Derivation test: in this test we try to derive the original word from the root using Arabic derivation rules. To achieve this, we use SARF system (Al-Bawab et al., 2007). However, this test affects clearly the performance of our morphological analyzer, but it still an **optional check** to provide the possibility of using this system in high accurate applications.

4.2. Controlling Technique:

To control the balance point (accuracy, performance, generality) we use the next two techniques:

Adding some parameters to control the different modules of the system. We add 11 parameters (Table-5).

Ranking the solutions by its "accuracy"

.Table 5- Morphological Analyzer's parameters

ID	Parameter	Value's type	Description
1	*Stp*	Boolean	Do StopWord Test or not.
2	*Frg*	Boolean	Do ForignWords Test or not.
3	*Rot*	Boolean	Do RootExistance Test or not.
4	*Pat*	Boolean	Do Patterns Test or not.
5	*Ebd*	Boolean	Do Ebdal Test or not.
6	*Eal*	Boolean	Do Ealal Test or not.
7	*Shd*	Boolean	Do Shadda Test or not.
8	*Elm*	Boolean	Do Eliminating Test or not.
9	*Drv*	Boolean	Do Derivation Test or not
10	*Bst*	Boolean	Satisfying the best solution or not.
11	*RML*	Integer (3, 4, or 5)	ROOT_MAX_ LENGTH

5. Evaluation:

The main goal of this evaluation is to prove the flexibility of our approach, i.e. to prove that it can satisfy the needs of most applications. To do this, we will describe our test corpora, choose evaluation metrics, and choose some states for our morphological analyzer (providing that each state represents certain values of morphological analyzers' parameters). For each state, we will evaluate chosen metrics on the test corpora (or on a part of it).

5.1. Evaluation metrics:

We choose our metrics to predict how our morphological analyzer acts in different kinds of applications. These metrics include performance, accuracy, and contextual metrics:

Speed of processing: we are interested here in the performance comparison between our morphological analyzer in its different states and other analyzers, and in the relation between the performance and other metrics.

Accuracy of the first solution.

Accuracy of all solutions.

Contextual correctness of the first solution.

Existence of the contextual correct solution in the solutions.

5.2. Test corpora:

We conducted our experiments using three different corpuses:

The first corpus consists of a lists of word-root pairs (167162 pairs) extracted from HIAST Arabic lexical database (Al-Attar et al., 2007) 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 the same effectiveness on all morphological categories.

The second corpus is a collection of 585 Arabic articles covering different categories (politics, economy, culture, science and technology, and sport). This corpus consists of more than 375000 words. We will use this corpus to evaluate the first metric (speed of processing).

The third corpus is a manually verified sample consists of five articles (more than 2000 words). We will use this corpus to evaluate the contextual metrics (fourth and fifth ones).

5.3. Chosen states for the morphological analyzer:

As we mentioned later, we have ten parameters to control the balance point between performance, accuracy, and generality. In this section, we will choose the most important states that represent our system's capability to cover the needs of most NLP applications.

To name morphological analyzers' states, we follow the form *Rn(+)(a|b|c)* where:

- **Rn means that we set the values of the first nth parameters (according to the numbers in Table-5) to true. For**

example, R3 means that the values of the parameters 1, 2, and 3 are true.

- **We add '+' to the form if we set Bst parameter to true.**
- **We add a letter to represent the value of the root maximum length RML (a for 3, b for 4, and c for 5), if we do not add anything RML=3.**

Table-6 shows the states we choose in the evaluation and their names; we represent the values of *Bst* and *RML* in the first two rows, and the first eight columns for the first eight parameters where we shadow the field only if it is true.

Table 6- The most important states in the morphological analyzer

3	5	**RML**								
True	False	**Bst**								
R0+	R0c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R1+	R1c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R2+	R2c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R3+	R3c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R4+	R4c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R5+	R5c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R6+	R6c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R7+	R7c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R8+	R8c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp
R9+	R9c	Drv	Elm	Shd	Ela	Ebd	Pat	Rot	Frg	Stp

5.4. Results:

a- Performance, Accuracy, and Generality of solutions:

We evaluate our morphological analyzer using the last three corpora. Figure-5 shows the most important balance points we achieved using our analyzer.

The Horizontal axe represents the processing speed, while the vertical axe represents the accuracy. Filled circles with **bold** names represent *Rn* states in which we try to find all possible solutions, where empty circles represent *Rn+* states in which we get the best solution. In this way we represent the accuracy, performance, and generality (for more detailed results: see Table-7, and Table-8).

In addition, we place some previous related works (Buckwalter and Khoja) in the same figure to compare all approaches.

In figure 5, we can see clearly that our approach provides different balance points which can support the needs of most applications. It provide states like (R0+, R1+, R2+, R3+, R4+) which have the advantage of *high performance* and **outperform the performance of most others stemmers**.

States R7+, R6+, R5+ are *high performance and high accurate* balance points. Their accuracy (about 97%) can be compared to high accurate rule-based stemmers like khoja one, but **we achieve this accuracy with four times better performance**.

Rn states outperform *Rn+* ones in *generality*. We did not notice much difference in accuracy in states R0, R1, R2, R3, R4 where the generality affects mainly the performance (as we do not cover solutions that represent elimination, shadda, and Ealal cases).

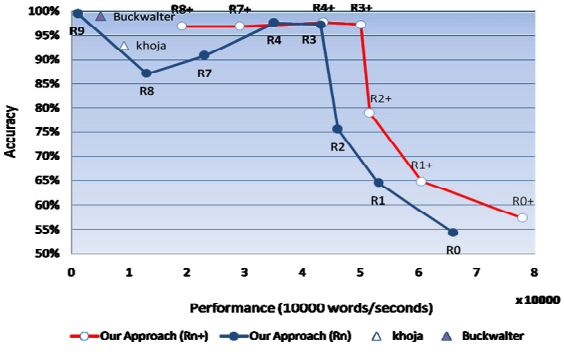

Figure 5- Most important balance points in our morphological analyzer and its position according to Koja and Buckwalter analyzers.

R5+, R6+, R7+ solve these problems without great effect on the performance, and still outperform other stemmers, because in these states we look for the best solution which is not one of these three difficult cases for most Arabic words.

In addition, we provide R5, R6, R7, R8 in which we try to **include all right solutions** even those representing elimination and shadda, which affects the accuracy. For example, the accuracy of R7 (where we solve **all** special cases) is about 87%. We can use these states for learning systems or lexical dictionaries where the stored data in these systems help to correct the result and raise the accuracy. But the main advantages here are: **the high performance (comparing to dictionary-based system) and the achievement of all correct solutions.**

On the other hand, we provide state R9 which focus mainly on **high accuracy and high generality** (we can conclude all the correct solutions and only the correct solutions with an accuracy exceeding 99%) i.e. we delete most wrong solutions which are concluded at state R8. This state represents the best choice for applications like learning systems, corpus building, machine translation and other "critical" systems.

Table 7- The Accuracy of the different states

The State	The Accuracy of Rn+ (Finding the best solution only)	The Accuracy of Rnc (Finding all possible solutions)
R0	57.30%	54.30%
R1	64.70%	64.45%
R2	78.90%	75.50%
R3	97.30%	97.30%
R4	97.70%	97.60%
R5	97.20%	94.80%
R6	96.80%	94.80%
R7	96.90%	90.80%
R8	96.90%	87.10%
R9	99.90%	99.10%

Table 8- The Performance of the different states (words/seconds)

The State	The Performance of Rn+ (Finding the best solution only)	The Performance of Rnc (Finding all possible solutions)
R0	78000	66000
R1	60500	53000
R2	51500	46000
R3	50000	43000
R4	43500	35000
R5	43000	34000
R6	39000	31000
R7	29000	23000
R8	19000	13000
R9	9000	1000

b- Contextual Correctness:

Moreover, we are interested in **contextual evaluation** which is very important in most applications. Table-9 shows that the contextual correctness of the first solution in most states of our analyzer is more than 96%. These values do not differ a lot from the last values in accuracy test because we found that more than 99.6% of correct analyses are contextually correct analysis.

In addition, we have tested the existence of the contextual correct solution in the output. The results are presented in the Table -1.

Table 9- The Contextual Correctness of the Different States

The State	The Contextual Correctness of Rn+ (The Contextual Correctness for the best solution)	The Contextual Correctness of Rnc (Finding the Contextual Correct solution in the solutions)
R0	56.90%	97.50%
R1	64.10%	97.50%
R2	78.88%	97.50%
R3	97.10%	97.50%
R4	97.50%	97.50%
R5	97.00%	97.57%
R6	96.60%	99.20%
R7	96.70%	99.67%
R8	96.60%	100.00%

c-Number of solutions & Failure percentage:

To complete our evaluation, we do some statistics about the number of solutions and the failure percentage in each case. The failure in R3 state is about 8%, because we do not process any special cases. Such state is useful in IR applications where we can accept these values. The failure is decreased gradually with the processing of special cases. We get approximately zero-failure in R7 and R8 states where we test special cases.

On the other hand, the average number of solutions (without diacritics) is not exceeding 3 solutions even when we look for a maximum percentage of generality, contextual correctness and accuracy (R8). We consider this result as an advantage because we do not need a big number of solutions to get the correct ones.

Table 10- Statistics About the Number Of Retrieved Solutions (without diacritics)

The State	Number of Solutions for a Word					Average number of solutions
	zero	one	two	three	more	
R3	8%	83%	9%	1%	0%	1.11
R4	8%	84%	8%	0%	0%	1.09
R5	8%	84%	8%	0%	0%	1.09
R6	1%	78%	15%	6%	0%	1.27
R7	0%	61%	23%	12%	3%	1.59
R8	0%	56%	6%	8%	31%	2.53

6. Conclusion

In this paper, we propose Original Letters Detection Algorithm as an auxiliary algorithm to improve the accuracy and the performance of any stemmer. Using this algorithm, we present an Application Oriented Arabic morphological analyzer which can satisfy the needs of most kinds of applications including those which need high performance, high accuracy, or generality in solutions.

7. Future Works

Our future work will focus on the use of this morphological analyzer in IR applications to evaluate its performance in the relevant tasks. In addition, we started to build an Arabic Part of Speech Tagger, where using a good morphological analyzer helps getting better results.

Appendix: sample outputs from Buckwalter system, Khoja system, and our system.

1- The input is "ساقاه" /sakah/

Bukwalter

INPUT STRING: ساقاه
LOOK-UP WORD: sAqAh
SOLUTION 1: (sAqAhu) [sAqaY_1] sAqA/VERB_PERFECT+(null)/PVSUFF_SUBJ:3MS+hu/PVSUFF_DO:3MS
(GLOSS): + give to drink + he/it <verb> it/him
SOLUTION 2: (sAqAhu) [sAq-u_1] sAq/VERB_PERFECT+A/PVSUFF_SUBJ:3MD+hu/PVSUFF_DO:3MS
(GLOSS): + drive/lead/direct + they (both) <verb> it/him
SOLUTION 3: (sAqAhu) [sAq_1] sAq/NOUN+A/NSUFF_MASC_DU_NOM+hu/POSS_PRON_3MS
(GLOSS): + leg + its/his two

Khoja

سقي ساقاه ROOT

Our approach (R9)

vocalized	pre	type	pattern	root	state	suf	details
سَاقاه		اسم جامد		سوق		اه	
سَاقَّاه		اسم فاعل	فَاعِل	سقق	مرفوع	اه	مضاف إلى معرفة/مذكر /مثنى
سَاقَاه		فعل ماضٍ	فعَل يفعُل	سوق	مبني	اه	لازم ومتعدي/مبني للمعلوم/ثلاثي مجرد/أجوف واوي/مسند للضمير هما للمذكر
سَاقَاه		فعل ماضٍ	فاعَل يُفاعِل	سقي	مبني	ه	متعدٍ/مبني للمعلوم/ثلاثي مزيد/ناقص يائي/مسند للضمير هو

2- The input is "أصطف" /astaf/

Bukwalter

INPUT STRING: أصطف
LOOK-UP WORD: >STf
SOLUTION 1: (>aSoTaf~) [{iSoTaf~_1] >a/IV1S+SoTaf~/VERB_IMPERFECT
(GLOSS): I + get in line/stand in formation +
SOLUTION 2: (>aSoTaf) [{iSoTafaY_1] >a/IV1S+SoTaf/VERB_IMPERFECT
(GLOSS): I + choose/select +

Khoja

اصطف اصطف NOT STEMMED

Our approach (R9)

vocalized	pre	type	pattern	root	state	suf	details
أَصْطَفُّ		فعل مضارع	افْتَعَل يَفْتَعِل	صفف	مرفوع		لازم/مبني للمعلوم/ثلاثي مزيد/مضعّف/مسند للضمير أنا
أَصْطَفَّ		فعل مضارع	افْتَعَل يَفْتَعِل	صفف	مجزوم		لازم/مبني للمعلوم/ثلاثي مزيد/مضعّف/مسند للضمير أنا
أَصْطَفَّ		فعل مضارع	افْتَعَل يَفْتَعِل	صفف	منصوب		لازم/مبني للمعلوم/ثلاثي مزيد/مضعّف/مسند للضمير أنا
أَصْطَفِ		فعل مضارع	افْتَعَل يَفْتَعِل	صفو	مجزوم		متعدٍ/مبني للمعلوم/ثلاثي مزيد/ناقص واوي/مسند للضمير أنا

References

Al-Attar. S, Al-Bawab. M, and Al-Dakkak, O.(2007) *Arabic Lexical databas*. ANLP, ICTIS 2007, fes, Morocco.

Al-Bawab, M., Mrayati, M., Alam, Y.M., & Al-Tayyan, M.H. (1994). *A computerized morpho-syntactic system of Arabic.* The Arabian Journal of Science and Engineering, 19, 461–480. Published by KFUPM, Dhahran, Saudi Arabia.

Al-Bawab M., mohtasseb H., Issa K., *Sarf - Arabic Morphology System*.

http://sourceforge.net/projects/sarf/.

Al-Fedaghi, S. S., & Al- Anzi, F. S. (1989). *A new algorithm to generate root-pattern forms.* In Proceedings of the 11th National Computer Conference (pp.391–400). Published by KFUPM, Dhahran, Saudi Arabia.

Al-Sughaiyer, I. and Al-Kharashi, I. (2004). *Arabic morphological analysis techniques: A comprehensive survey.* Journal of the American Society for Information Science and Technology, 55(3):189–213.

Buckwalter, T. (2002). *Buckwalter Arabic Morphological Analyzer* Version 1.0, Linguistic Data Consortium (LDC) catalog number LDC2002L49 and ISBN 1-58563-257-0.

Darwish, K. (2002). *Building a shallow Arabic morphological analyzer in one day.* In Proceedings of the Association for Computational Linguistics (ACL-02), 40th Anniversary Meeting (pp.47–54), University of Pennsylvania, Philadelphia

Khoja, S., & Garside, R. (1999). *Stemming Arabic text*. Computing Department, Lancaster University, United Kingdom,

http://www.comp.lancs.ac.uk/computing/users/khoja/stemmer.ps.

Larkey, L.S. and Connell, M. E. (2001) *Arabic information retrieval at UMass in TREC-10*. In TREC 2001. Gaithersburg: NIST.

Sonbol. R, Ghneim, N. and Desouki, M.S. (2008). *Arabic Morphological Analysis: a New Approach.* 3d International Conference on Information & Communication Technologies: from Theory to Applications - ICTTA'08. Damascus, Syria.

Received 13/4/2010