Diacritic-Aware Arabic Word Matching

Mustafa Jarrar[a], Fadi A. Zaraket[b], Rami Asia[a]
[a]Birzeit University, West Bank, Palestine
[b]American University of Beirut, Lebanon

Abstract *Words in Arabic consist of letters and short vowel symbols called diacritics that are typically inscribed atop regular letters. Changing some diacritics may change both the syntax and semantics of a word; turning a word into another. These results in difficulties when matching two or more words solely based on basic string matching techniques. Typically, Arabic NLP applications resort to morphological analysis to battle ambiguity originating from this and other challenges. In this paper, we introduce the implication relationship algorithm (IRA) which takes two words with the same non-diacritic letters and decides whether they are the same or not. It compares the words and computes a distance metric between diacritics. Second, we introduce the morphology subsume algorithm (MSA) which computes a metric that measures how much one word is a morphological replacement of another word with the same non-diacritic letters. Both algorithms are sound. When each makes a full decision, its decision is always correct. However, MSA is incomplete as it cannot make a decision in a number of cases; which could be the case for expert human readers as they might require context to decide as well. Nevertheless, our experiments show that after several refinement iterations for IRA rules, IRA provides an answer for 100% of the word pairs given, and MSA provides an answer for about 95% of the words given. Both IRA and MSA distance metrics agree on 93% of the intersection. The high agreement value is evidence that Arabic NLP applications that do not directly need the morphological features may use the computationally-lighter IRA algorithm for disambiguation. We demonstrate this result with a lemma disambiguation case study.*

Keywords: *Arabic; NLP; Natural language processing; implication; diacritics; disambiguation.*

1 Introduction

Diacritics are distinguishing features of the Arabic language. Diacritics in Arabic have two main roles: (i) they provide a phonetic guide i.e. help readers recite/articulate the text correctly, and (ii) they disambiguate the intended meaning of otherwise ambiguous words. Table 1 shows a list of the most used diacritics of the modern standard Arabic (MSA) language. The *fatha* and *kasra* diacritics show as accents above and below the corresponding letter and indicate short ‘a’ and ‘i’ vowels, respectively. The *dhamma* diacritic shows as an accent with a small circle and denotes a short ‘o’ vowel. A *sukoon* shows a small circle atop and denotes a silent letter. A *shadda* is a gemination marker seen above a letter. It denotes stressing the letter such that the letter is pronounced twice: first as a silent letter and second with a non-sokoun diacritic. A *tanween* diacritic is an indefiniteness mark and shows as a double fatha, kasra, or dhamma diacritic. It denotes the letter spelled with the marked diacritic followed by a silent ‘n’ sound.

Table 1 Basic diacritic table in Buckwalter

Diacritic	Shape	Example
fatha (short a)	ـَ	رَسَمَ (rasama) drew
damma (short o)	ـُ	سُنبلَة (sonobulap) spike (of grain)
kasra (short y)	ـِ	سِهام (sihAm) arrows
sokoun (silent vowel)	ـْ	سِعْر (siEor) price
shadda (stress mark)	ـّ	هدَّد (had~ad) threatened
tanween-fatha	ـً	أبداً (abadAF) never
tanween-dhamma	ـٌ	قلمٌ (kalamON) pen
Tanween-kasra	ـٍ	شعبٍ($aEobIN) people

It is common practice for Arabs to write Arabic text without diacritics, which makes Arabic text highly ambiguous [1]. Ambiguity refers to the fact that the morphological, syntactic or semantic analysis of one word may lead to several possible various word matches. That is, two words with the same non-diacritic characters but with different and possibly omitted diacritic characters are not necessarily the same. While morphological analysis is key in current automated analysis techniques for Arabic text, it is known that morphological ambiguity is a `notorious' problem for the Arabic language [2].

The results of [5], cited in [4], state that non-diacriticized words exhibit 8.7 syntactical

ambiguity on the average which drops to 5.6 for diacriticized words. For instance, the word جزر (jzr) has different interpretations based on diacritization; e.g. جَزَر (jazar) means carrots, جُزُر (jozor) means islands, and جَزْر (jazr) means the fallback of the tide. The word جَزَرَ (jazara) is a past tense verb meaning "butchered".

The use of diacritics for disambiguation is not restricted to human readers. It applies also to automated tools such as morphological analysers. Some morphological analysers such as [6][1] use partial diacritics to resolve ambiguity, e.g. they filter solutions that are inconsistent with diacritics available in the input text.

Essential to the process of disambiguation is the comparison of two Arabic words with the same sequence of non-diacritic letters, but with different diacritics. A simple string comparison of two Arabic words such as word جزر (jzr) and جَزَر (jazar) returns a negative result due to the two additional diacritics, while readers identify them as the same word in a sentence. Typical tools that provide Arabic editing and searching services tend to ignore diacritics in search and matching. In analogy, imagine ignoring vowels in English. This means that tools need to consider the words جَزَر (jazar) carrots and جُزُر (jozor) islands as the same word.

In this paper, we present two algorithms that compare words with similar non-diacritic letters with possibly different diacritic letters, and study their accuracy. The *implication relationship algorithm* (IRA) checks whether one word w_1 implies another word w_2 with the same no-diacritic letters. It compares the difference and distance between diacritics to assist in testing whether w_1 and w_2 are the same word or not. The *morphology subsume algorithm* (MSA) checks whether word w_1 is morphologically superior to another word w_2 with similar non-diacritic letters. It computes a score metric that measures how much w_1 is a morphological replacement of w_2.

We evaluated both algorithms against a collection of word pairs collected from several Arabic dictionaries, resulting in 16,408 distinct words. Each word was paired with potentially similar words from the SAMA 3.1 database that the ALMOR analyser proposed to be potentially the same word. In the end, our evaluation was against a set of 35,203 distinct word pairs. Both algorithms are sound and exhibit high agreement. When either IRA or MSA makes a full decision, that decision is always correct. The MSA algorithms is not complete as it cannot make a decision in a small number of cases. Note that this also could be the case for expert human readers as they might require context to decide. Nevertheless, IRA reported determined answers for 100% of the pairs used in the experiment. The IRA algorithm was fine-tuned based on the feedback of an expert linguist who inspected the results manually. The MSA provides an answer for about 95% of the words given. The remaining 5% were missing in the lexicons of the morphological analyser we used. Both the IRA and the MSA distance metrics agreed on 93% of the intersection. The source code of both algorithms, as well as the datasets we used in the experiments, are accessible online*.

The high level of agreement is evidence that Arabic NLP applications that typically use morphological analysis as a necessary pre-processing step to battle ambiguity may use the much lighter IRA algorithm instead, in case the application did not directly need the morphological features. We demonstrate this claim with a lemma disambiguation case study.

The rest of this paper proceeds as follows. In Section 2 we provide necessary definitions and background. In Section 3, we present and discuss the IRA algorithm. In Section 4, we present and discuss the MSA algorithm. In Section 5, we present related work and compare it to IRA and MSA. We present the experimental setup and detail our results in Section 6 and we discuss the utility of our work for practical case studies in Section 7. We conclude and discuss future work in Section 8.

* (https://github.com/SinaInstitute/ImpliCheck) or (http://ontology.birzeit.edu/tools/verbmesh/DiacriticAwareMatching/index.html)

2 Related Work

We first review related work that stress the importance of considering diacritics for automated comprehension of Arabic text and for ambiguity reduction [4][5][10][1][17]. Then we review works that attempt to restore diacritics to Arabic text [13][14][15][16][18][20][21]. Then we review morphological resources that include diacritic information [3][6][1][8][9].

Diacritics and disambiguation. Much literature attested to the high ambiguity of un-vocalized text and the power of diacritics in ambiguity reduction. Programs that automatically add diacritics to Arabic text exist in the software industry (Sakhr, RDI). However, the commercial products are closed source. They use morphological analysis and syntax analysis to predict the diacritics. The work in [10] automatically adds diacritics for transcriptions of spoken Arabic text and employs existing acoustic information to predict the diacritics. The work in [1] introduces an ambiguity controlled morphological analyser that employs a rule based system and finite state machines. The work ignores diacritics when they exist in Arabic text because it claims that it only found insignificant meaningful diacritics when considering a large corpus. It then illustrates how diacritics may be used later on to filter vocalized solutions if needed by the application using the morphological analyser.

Both [10][1] quote from [5] that a dictionary word with no diacritics has on average 2.9 different possible vocalizations and that a sample text of 23 thousand words exhibited 11.6 different diacritic assignments per word on average. The same source reports that 74% of Arabic words have more than one possible vocalization. It also reports an average of 8.7 syntactical ambiguity for unvocalized words, which drops to 5.6 for vocalized words. This is evidence of the role of diacritics in disambiguation of word forms.

The work in [4] presents problems in DECORA-1 related to correction and error agreement detection in Arabic text. They introduce an enhancement, DECORA-2, that considers diacritics to help resolve the problems. The work in [4] claims that most Arabic text omits diacritics, few Arabic teaching books have partial diacritics, and only the Quran and teaching books for early stages are fully vocalized. Studies in [4] show that an Arabic word usually has six varying vocalizations.

The work in [17] takes an Arabic word, checks it against preset vocalization templates and returns the POS tags of the word. The approach works for a set of verbs and nouns and is reported to decrease ambiguity when diacritics that vocalize the last character of the pattern exist. This is evidence of both the utility of diacritics in disambiguation of POS tags and the importance of where to place the diacritic within the word.

Nevertheless, the study in [12] concludes that the presence of diacritics affects reading speed negatively while increasing text comprehension only when diacritics play a role in disambiguation. It advises that writers must provide diacritics economically when needed for disambiguation of the intended meaning.

Diacritic restoration. The vast majority of work on Arabic diacritics is concerned with restoring diacritics to Arabic text [13][14][15][16][18] [20] [21].

Among other morphological disambiguation tasks, the work in [18] explores two diacritics related tasks: DiacFull and DiacPart. DiacFull restores diacritics to all letters of the word and DiacPart restores them to all letters except the final letter. The work uses a linear optimization technique to select the best diacritization of the given word. They confirm two important hypotheses: 1) the use of lexeme features help in determining the best diacritics, 2) tuning the parameters of the optimization algorithm to the task at hand helps the disambiguation task. In our work, we manually fine-tuned weights that characterize an arbitrary distance between diacritics to reduce comparison errors and we arrived at a similar result to hypothesis (1).

The work in [20] is a follow-on to [18]. It adds diacritics to words in context of morphological disambiguation and tokenization.

The work in [21] describes a commercial product by RDI to automate diacritization of Arabic text. It uses a stochastic process to decide the most likely diacritic map. Since the map is not necessarily grammatically correct and may be out of the vocabulary, a second stochastic process uses more features of the word including morphological features to select the most likely compatible solution out of a set of diacritic based feature factorizations. This work is evidence of

how much partial diacritics can reduce sophisticated work needed later to disambiguate the Arabic text. With our MSA and IRA algorithms one can infer the diacritic that most reduces ambiguity and use it.

The work in [13] presents a system for Arabic language diacritization using Hidden Markov Models (HMMs). It represents each diacritic with an HMM and uses the context of the whole text to concatenate the HMM decisions and produce the final diacritic sequence.

The work in [14] presents a hybrid system for Arabic diacritization based on rule based and data driven techniques. It uses morphological features and out of vocabulary elimination techniques to reduce the solutions. They do better than [18] [20][21] by an absolute margin of 1.1% and still make an 11.4 full diacritization word error rate. This is evidence of how complex and sophisticated the diacritization process is.

The work in [15] presents a system to diacritize Arabic text automatically using a statistical language model and morph-syntactical language models. The work in [16] presents an evaluation of three commercial Arabic diacritization systems using fully diacritized text from the Quran and short poems from the period of the advent of Islam.

Existing morphological resources with diacritics. In addition to automatic diacritization tools, previous work includes tools that disambiguate based on input diacritics. Analysers such as Buckwalter [3] and SAMA [9], contain the diacritization of lexicon entries in addition to other annotations. However, they ignore the partial diacritics in the analysis phase and the analysis makes little benefit from lexicon vocalizations. MORPHO3 [1], Arabic Xerox [8] and MORPH2 [6] are other examples of morphological analysers with the same capability. They later filter morphological solutions based on their consistency with available diacritics.

In summary our related work, we find that the vast amount of work discussing the challenges imposed due to missing diacritics is evidence of the utility of our work. Sophisticated and advanced technologies coupled with advanced expert rules used to automate diacritization lack in accuracy and make significant errors (11.4 %). Morphological analysers avoid partial diacritics in analysis and defer the task for interested NLP applications to use them as filters later.

To conclude, we are the first to approach the diacritic placement problem as a word-to-word comparison problem. Leveraging our algorithms, NLP tools can reduce ambiguity by placing the diacritics that matter. Users at the entry level also can be encouraged to introduce the minimal number of diacritics that matter to reduce ambiguity.

3 The IRA Algorithm

Before delving into the details of how the algorithm works, we present a few definitions.

Definition 1 (Implication Relation): Given two Arabic words w_1 and w_2, an *implication relationship* denotes that w_1 implies w_2 *iff* both words have the same letters and every letter in w_1 has the same order and same (or less) diacritics as the corresponding letter in w_2. See the words (فَعَل) (فَعَل) in Table 1.1 as an example. If both w_1 implies w_2 and w_2 implies w_1, then the two words are called *compatible*; Otherwise, in case of no implication, the words are *incompatible*.

Implication Direction	Meaning	Example
0, -1	incompatible	فَعَل ، ثَعلَب
1	w_1 implies w_2	فعل ، فَعَل
2	w_2 implies w_1	فَعَل ، فعل
3	Compatible	فعَل ، فَعل

Table 3.1 Implication Direction with examples

Definition 2 (Distance Map): A *distance map* denotes a matrix of all possible pairs of Arabic diacritics and a distance value between them (see Table 3.2). The value represents a score of interchangeability. Two diacritics are interchangeable if either one can replace the other on a character in a word without changing the meaning of the word. A distance map tuple <d_1, d_2, *delta*> denotes the two diacritics by d_1 and d_2 and the distance score by *delta*. As will be discussed later, the IRA algorithm uses this distance map to calculate the distance between two words. For example, the distance between the diacritics *fatHa* and *kasra* is 1. When the *shadda* or *hamza* appear as either d_1 or d_2, and the compared diacritic is different, the distance equates 15 (but it can be also 4 depending on which letter has the diacritics, as will be explained later).

Table 3.3 Diacritic Pair Distance Hash Map

	Fatha	Dhamma	Kasra	Sukun	Fathatan	Kasratan	Dhammatan	Shadda	Hamza
No Diacritic	1	1	1	1	1	1	1	1	1
Fatha	0	1	1	1	1	1	1	15	15
Dhamma	1	0	1	1	1	1	1	15	15
Kasra	1	1	0	1	1	1	1	15	15
Sukun	1	1	1	0	1	1	1	15	15
Fathatan	1	1	1	1	0	1	1	15	15
Kasratan	1	1	1	1	1	0	1	15	15
Dhammatan	1	1	1	1	1	1	0	15	15
Shadda	15	15	15	15	15	15	15	0	15
Hamza	15	15	15	15	15	15	15	15	0

Definition 3 (Conflicting Diacritics): Two diacritics are called *conflicting diacritics* if they are distinct and appear on the same character of a given word. That is, given words w_1 and w_2, for a pair of diacritics d_1 and d_2, where d_1 is located in w_1 at the corresponding position of d_2 in w_2, if d_1 does not equal d_2, then d_1 and d_2 are conflicting diacritics and w_1 is incompatible with w_2.

Definition 4 (Words Matching): The matching between two Arabic words is defined as a tuple <w_1, w_2, *implication direction, implication distance, conflicts, verdict*>. The w_1 and w_2 elements are the two words to be compared. The *implication direction* is a number denoting the relationship between the two words Table 3.1. The *implication distance* depicts the overall similarity of the diacritization implication between the two words, which we compute based on the distance map. The *conflict* elements denote the number of conflicting diacritics between the two words. Finally, *verdict* takes one of the values: 'Same', or 'Different', to state whether w_1 and w_2 are matching.

3.1 Description of IRA Algorithm

The IRA algorithm takes two words as input and produces the matching tuple defined by definition 4. For each input word, IRA generates two arrays, one for the letters (each letter receiving a cell) and one for diacritics (each diacritic in a cell). The words are then checked to find if they contain the same letters. If so, then for each pair of corresponding letters, an implication value and a distance is assigned. Figure 3.1 illustrates an example of comparing two words (فَعَلَ) and (فعل).

Implication between letter pairs is determined as follows. When both letters have exactly the same diacritics (see Table 3.1), a score of 3 is assigned to the corresponding position in the implication array. If the pair has conflicting diacritics, a score of 0 is assigned to the pair in the array's corresponding position. If the first letter in the pair is missing a diacritic present on the second letter, then an implication direction of 1 is assigned to the pair in the array's corresponding position. If the second letter in the pair is missing a diacritic present in the first one, an implication score of 2 is assigned to the pair in the array's corresponding position.

Implication between two words is determined as follows. Once an implication value is assigned for each pair of letters, the implication array is observed. If *all* entries in the implication array contain a value of 3, then an overall implication value of 3 is returned. If all entries in the array contain a value of either 1 or 3, then an overall implication of 1 is returned. If *all* entries contain a value of either 2 or 3, then an overall implication of 2 is returned. Though, if there exists at least one 0 or both 1 and 2 are in the array, then an overall implication of 0 is returned.

The implication distance between two words is determined as the following. While generating the implication array, the algorithm loops through the diacritic arrays. For each pair of diacritics, the algorithm returns a distance from the distance map and adds it to the overall distance value.

Once the algorithm returns both the implication direction and the distance values, it returns a verdict of "Same" if the overall implication direction equals 3, 2, or 1. Otherwise, when the overall implication is equal to 0, there is no clear implication between the two words and the verdict is "Different". Note that the distance value between two words does not impact the implication value or the verdict. The distance is returned by the algorithm as an extra measure and only to indicate how much two words are close to each other's diacritics.

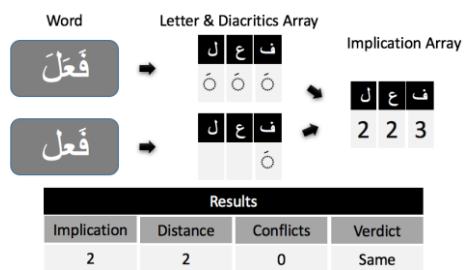

Figure 3.1 Implication Example

[*] Due to the IRA algorithm's full description being slightly lengthy, the full details of the IRA algorithm are presented and diagrammed in http://www.jarrar.info/publications/JZR16_IRA.pdf

3.2 Special Cases

In this subsection we specify the special cases that our algorithm fine-tune to increase the accuracy of the results. We learned of these special cases and how to handle them through intensive manual comparisons and investigations between similar words in our dataset. These special cases are those of: diacritics on the last letter of a word being compared, the *hamza*, the *shadda*, and the *sokoon*.

Diacritics on the ***last letter of a word*** are a special case, as differences in diacritization do not change the meaning of the word. Considering this, the algorithm neglects the diacritics on the last letter of the words being compared.

The case of the ***shadda*** is also special. We use different weights for calculating word distance depending on the position of *shadda* in a word. When on the first letter of a word, the distance of a pair including the *shadda* is 4 if the diacritics are not the same, as people tend to ignore *shadda* in the beginning of a word most the time. In the case of shadda on any other letter in the middle of a word, this distance increases to 15.

This distance changes because the *shadda* plays a much larger role when found in the middle of the word, as opposed to the beginning. When on the first letter of a word, it does not singlehandedly determine whether two words are the same, whereas it can certainly determine whether two words are the same or not when found in the middle of a word.

The case of the ***hamza*** is similar to that of the *shadda*. When found atop the first letter of a word, it is considered a diacritic, rather than a letter. As found in the case of the *shadda*, when the *hamza* is on the first letter of a word, the distance of the diacritic pair is 4 if the diacritics are not the same. In all other cases, the *hamza* is considered a letter and treated as such. In other words, the *hamza* is treated as and considered a letter only when it appears after the first letter in a word.

The ***sokoon*** is a peculiar case. This is because it carries no sound. Because of this, it is usually neglected in writing, where a letter that is not diacritized is usually taken as a letter diacritized with a *sokoon*. As a result, in the Diacritic Map, the *sokoon* has a weight of 0 when alone and not compared to another diacritic. Otherwise, the weight differs depending on the other diacritic.

As stated earlier, our algorithm is always sound over all the dictionary words it covers. We established soundness by iterating few times over the data set and refining for the non-deterministic results after each iteration. The refinement was based on input from a linguistic expert who carefully evaluated whether all critical cases are valid results, and what the correct result should have been. The refinement included changes to distance map entries and modifications of the special cases.

The linguistic rules implemented in IRA also play a vital role in determining the weights within the Diacritic Map. Especially in the cases of the *shadda* and *hamza*, the heavy weights of the diacritics were largely influenced by the linguistic properties these diacritics have.

As will be discussed in the evaluation section, and in the use case section, the IRA algorithm is 100% sound, and 100% complete w.r.t. to the used dataset that makes it fairly comprehensive. We used the algorithm to match between Arabic verbs in the ALMOR's dataset and 37 other lexicons. These lexicons represent different levels of diacritization, ranging from fully diacritized to non-diacritized words.

4 The MSA Algorithm

The MSA algorithm takes two Arabic words w_1 and w_2 and returns a score between 0 and 1 that denotes how much w_1 can morphologically subsume w_2. Before presenting the how the algorithm works we must define a few notions.

Definition (Morpheme) A *morpheme* is the smallest unit of morphological structure of a given word. For Arabic, a morpheme is either an *affix* or a *stem*. The *affix* is either a *prefix* or a *suffix*. The stem could be a *root* or an *inflection* of the root. A word is the concatenation of *connecting* prefix, stem and suffix morphemes where the prefix and the suffix could be empty strings. Morpheme connectivity is a predefined relation. For example, the prefix يَ *ya* connects to the stem لعب *l'b* (play) to form the word يلعب *yal'b* (is playing). Each morpheme is associated with morphological features such as part of speech (POS), transliteration, lemma and gloss.

Definition (Morphological analysis) The *morphological analysis* of a given word *w* is the set of all possible morpheme concatenations that

form *w*. A word may have more than one morphological *solution* and one solution may have more than one set of features associated with it. Table 4.1 illustrates three different solutions for the word أحمدهم (Ahmdhm). The first solution differs than the other two in diacritics, morpheme segmentation and translation. The other two agree in diacritics and morpheme segmentation however, they differ in meaning.

Translation		Suffix	Stem	Prefix
Did he praise them?	aAhamidahom	هم	حَمِد	أ
Their Ahmad	aAhmadahom	هم	أَحْمَد	
The best of them	aAhmadahom	هم	أَحْمَد	

Table 4.1 Three morphological solutions for أحمدهم

Definition (Morphology subsume relation) We say word w_1 morphologically subsumes word w_2 if the morphological analysis of w_1 returns a set of solutions including the set of solutions returned by the morphological analysis of w_2. For example, the words أحمدهم without diacritics, or with one fatha on the first letter, morphologically subsumes أَحَمِدَهم and أَحْمَدَهم.

Definition (Morphology distance metric) The morphological distance metric between two words w_1 and w_2 measures how much w_1 can morphologically disambiguate w_2. If w_1 and w_2 fully match in discretized and non-discretized characters, then the distance is 0; which denotes similarity. If w_1 and w_2 have different non-diacritic characters, then the metric is 1; which is the maximum distance and the words are deemed different. In case the words have the same non-discretized characters, the metric is calculated as $1 - |M2 - M1| / |M2|$ where M1 and M2 are the sets of morphological solutions of w_1 and w_2 respectively, and |M| denotes the cardinality of M. Intuitively, the distance is a ratio measure of how many solutions of M2 can be eliminated by the diacritics of M1.

4.1 Description of MSA Algorithm

As explained earlier, this algorithm takes two Arabic words w_1 and w_2 and returns a score that denotes how much w_1 can morphologically subsume w_2. The algorithm (See **MSAMetric** in Figure 4.1) makes use first of a diacritic consistency algorithm **isDiacriticConsistent** shown in Figure 4.2.

Algorithm **isDiacriticConsistent** takes the two words and makes sure the sequence of non-diacritic characters is an exact match and the partial diacritics included in the two words are consistent. Two diacritics are considered consistent if they are not conflicting diacritics. For example, consider the Arabic word شاهد with two possible full diacritizations شَاهَد *(Shahad) watched* and شَاهِد *(Shahid) witness*. The partially diacritized words شَاهد and شاهِد are diacritic consistent as there is no conflict in diacritic assignment between them. However, the partially diacrized words شَاهَد and شاهِد are not diacritic consistent since the fatha and the kasra on the third letter are conflicting.

If **isDiacriticConsistent** returned false, the MSA algorithm reports that the two words are distinct by returning a score of 1. Otherwise, the MSA algorithm uses an existing morphological analyser [22] to compute the morphological solutions M1 and M2 of w_1 and w_2, respectively. It also computes M, the morphological solutions of *w*, the non-diacriticized form of both w_1 and w_2. The MSA computes M1Minus, the complement of M1 in M, or intuitively the solutions that are eliminated due to the diacritics present in w_1. Then, the algorithm computes M12 by subtraction M1Minus from M2. Intuitively, this is the set of solutions that would have been eliminated from M2 by the diacritics of w_1. The metric finally computes and returns the ratio of the eliminated solutions to the solutions of w_2.

Consider the illustrative example in Table 4.2. The entries in Table 4.2 show five different morphological solutions (|M| = 5) of the non-vocalized Arabic word بن (*w*). The same word with a kasra on the first letter بِن (w_1) has four solutions (|M1| = 4). The same word with a shadda on the second letter بنّ (w_2) has two solutions (|M2| = 2). The set M1Minus has only one element **بن**. The set M12 eliminates **بِن** and has consequently one solution **بِن.** Intuitively, since w1 and w2 are diacritic consisting and have no conflicting diacritics, the kasra on w_1 implies half the solutions of w_2 while the shadda of w_2 implies one fourth the solutions of w_1.

	Transliteration	Semantics	POS
بُنّ	bun~	Coffee	NOUN
بِن	bin/ben	Name of a person (like Benjamin)	NOUN_PROP
بِن	bin	Son	Noun
بِنَّ	bin+na	They (female plural) appear	VERB_PERFECT+ PVSUFF_SUBJ:3F
بِن	bi+n	with Noon (name of a person)	PREP+NOUN_PROP

Table 4.2: Morphological solutions for بن

```
Double MSAMetric (Word w1 , Word w2 )
  if (isDiacriticConsistent(w1, w2) == False)
      return 1;
  // both w1 and w2 are consistent w is w1 and w2
  //without diacritics
  Word w = removeDiacritics(w1);
  // compute the morphological solutions
  Solutions M = analyser(w);
  Solutions M1 = analyser(w1);
  Solutions M2 = analyser (w2);
  // M1Minus are the solutions that w1 diacritics
  // eliminated from M to form M1
  Solutions M1Minus = M – M1;
  // M12 are the solutions w1 diacritics (that are not in
  // conflict with w2) would have eliminated from w2
  Solutions M12 = M2 – M1minus;
return 1 – M12.size / M2.size;
```

Figure 4.1 MSAMetric Algorithm

```
bool isDiacriticConsistent(Word w1 , Word w2 )
  int i1 =0, i2 =0;
  Diacritics d1, d2 ;
   while (i1 < w1.size && i2 < w2.size)
      if (!equals(w1[i1], w2[i2]))
        return false;
      i1 ++; i2 ++;
      d1.clear(); d2.clear();
      while (i1 < w1.size && isDiacritic(w1[i1]))
          d1.add(w1[i1]);
          i1 ++;
       endwhile // traverse w1 till next non-diacritic
       while (i2 < w2.size && isDiacritic(w2[i2]))
          d2.add(w2[i2]);
          i2 ++;
        endwhile // traverse w2 till next non-diacritic
        if (!isConsistent(d1 ,d2))
          return false;
  endwhile //traverse all of w1 and w2
return (i1 == w1.size && i2 == w2.size);
```

Figure 4.2 isDiacriticConsistent Algorithm

5 Evaluation and Discussion

This section presents the evaluation of both, the IRA and MSA algorithms, whether they are sound and complete, which was experimented over a large dataset of about 36K pairs of words. First, we define the notion of soundness and completeness, then we present our dataset, and discuss the results.

Definition (Soundness) As known in logic, we define the soundness of our algorithms as whether the results of the algorithm are correct or not. That is, if the algorithm judges two words to be same, or to be different, and this answer is *always correct* then the algorithm is called *sound*. As will be described later, our matching algorithms are both sound.

Definition (Completeness) As is also known in logic, we define the completeness of our algorithms as whether the algorithm is always able to judge whether two words to be same or different. If so, then the algorithm is called *complete*. As will be described later, our matching algorithm is not complete. However, our experiment below demonstrates that it provides 97% ground coverage.

5.1 Experiment Setup and preparation.

To evaluate the soundness and completeness of the algorithms we need to prepare a large dataset of pairs of words, run both algorithms and evaluate their results. To do this, we have collected 16,408 distinct Arabic verb words extracted from 38 different dictionaries. Afterwards, we used ALMOR [22] to retrieve possible matches of the collected 16,408 words. The retrieved matching words by ALMOR generated 35,203 pairs to compare.

The dictionaries we collected are at different levels of diacritization. For instance, the Al-Maany, Al-Waseet, Al-Ramooz, and the Dictionary of Scientific, Technical and Engineering Terms are each filled with highly diacritized verbs. Dictionaries of medium-level diacritization included, for example, the Biological Lexicon of Biology and Agricultural Sciences, the Dictionary of Economic Terms, and the Dictionary of Statistics. Dictionaries providing no diacritics on their words include the Hydrology Glossary, the Lexicon of Chemicals and Pharmaceuticals, the Lexicon of Education and Psychology, and the Historical and Geological Lexicon. A high level of diacritization entails that all or most diacritics of a word are written on the word. A medium level of diacritization provides a word with few diacritics present on the word. Selecting dictionaries with different levels of diacritization is important for our experiment as this provides a granular scope of the performance and extent of the capabilities of our algorithm.

The collected verb words were run through the ALMOR Arabic word analyser using the SAMA 3.0 database. ALMOR retrieved highly diacritized words that were *similar* (in appearance and spelling) to the input words. For each input word (of the 16,408 words), ALMOR returned a fully discretized that is possibly the same word as the input word. That is, the output of this process is a set of pairs where the first word that is partially discretized, and the second

word, from ALMOR, is fully discretized. In this process, after giving ALMOR our 16,408 words, it produced 35,203 distinct pairs of words. A sample of the dictionaries and the amount of pairs that we were able to retrieve from each dictionary is presented in **Error! Reference source not found.**.

The 35,203 pairs of words were input to our both matching algorithms to find whether each pair is equal or not. The results then given to a linguist to manually verify the algorithms' decisions.

Dictionary	Diacritization Level	Words Used	Word Pairs Generated
Al-Ramooz	High	8,734	19,296
Al-Waseet	High	8,033	17,721
Al-Ma`any	High	4,351	9,750
Al-Mustalahat	High	2,263	4,527
Pharmaceutical Dictionary	None	572	1,211
Nubian Dictionary	None	561	1,175
Philosophical Dictionary	None	133	307
Statistical Dictionary	Medium	45	79

Table 5.1 Sample of the dictionaries experimented on

5.2 Results and Discussion

After running **the IRA algorithm**, we found that it was able to judge all of 35,203 word pairs correctly. That is, the experiment yielded a success rate of 100% in terms of reaching a conclusive result. These results are also confirmed not only through the *manual evaluation* by an expert linguist, but also by the second MSA algorithm, as will be explained later. It is worth noting that the manual evaluation happened in two iterations. During the first iteration, the IRA was able to only correctly judge 97.6% of pairs, but then we learned and identified some special cases (the cases of *Shadda* and *Hamza* at the first letter of a word, discussed in section 3.2) and fine-tuned the IRA algorithm accordingly. As a result, we claim that the IRA algorithm is sound.

Although the IRA was able to correctly judge 100% the 35,203 word pairs, claiming that it is complete is a tricky case. This is because we cannot be absolutely sure, by nature of this problem, whether there might be any existing case where the IRA cannot judge. However, since we have evaluated the algorithm on a large number of dictionary entries that cover lots of the ground, we claim that it is *fairly comprehensive*. That is, we claim that IRA was *complete for the given dataset*, which itself represent a vast majority of language entries.

In case the IRA need to be used in extreme or very sensitive applications -for example in case one likes to consider two words with *Hamza* or *Shadda* difference to be different words- then we recommend to us the distance value computed by the algorithm. In case the distance value is 15 or more, it indicates that that the two words have *Hamza* or *Shadda differences*. However, based on comparing the 35k word pairs, we believe that *Hamza* or *Shadda* on the first letters do not have any impact on the decision.

After running **the MSA algorithm** on the 35,203 word pairs that we verified manually by an expet linguist, the MSA was able judge 95% of the pairs. The other 5% could not be judged because ALMOR did not return any solution. That is, the comprehensiveness of the MSA depends on the morphological analyser used and the comprehensiveness of its database, which is 95% in our case.

The soundness of MSA was 93%. This was calculated by comparing the judgments of the MSA with the judgments of the IRA that we verified manually, and in 93% of cases both algorithms agree.

Based on the above results, we would like to remark the IRA algorithm is computationally lighter than the MSA algorithm and does not require computation of morphological features to produce a highly accurate comparison result. This is evident in the high level of agreement between both algorithms. Therefore, it is advisable that Arabic text processing, including NLP applications that do not require morphological preprocessing, use the IRA algorithm instead of the MSA algorithm for word-to-word comparisons. Examples of such applications are those needing either word frequency or word colocation computations.

The MSA algorithm should be used in case morphological solutions and features are anyway needed and computed in the NLP application. Example for such applications are POS tagging and syntactic analysis.

6 Use Case (Lemma Disambiguation)

This section presents a use case to demonstrate the utility of our algorithm. In our VerbMesh project, which aimed to build a large graph of

Arabic verbs, we needed to link verbs found in hundreds of Arabic dictionaries that we currently possess at Birzeit University. Basic string-matching techniques alone do not suffice to match and link two verbs, as verbs with same spelling but different linguistic properties lead to incorrect and undesired matching. Instead, we used lemmas to match between verbs. Verbs having the same lemma are considered same entries. In other words, same verbs across dictionaries are linked with their lemmas, although they might be diacritized differently.

To assign a lemma to each verb in every dictionary, we used the ALMOR morphological analyser, utilizing the SAMA 3.0 database. The major challenge we faced in this approach was that ALMOR returned too many lemmas for each dictionary entry. For example, ALMOR returns 13 different lemmas for the entry نَحُلَ, which are {نَحَلَ , نَحَلُ , نَحِلَ , حَلِيَ , حَلَى , حَلُوَ , نَحُلُ , حالُ , حَلَّ , أَحَلَّ , أَحالَ , حُلَّ}. An undiacritized word can bring an extra level of ambiguity. An example of such a case can be seen when inputting the word سلب into ALMOR, where the resulting lemmas are {سَلَبَ, سَلَبُ}. This is because ALMOR takes a word as input, segments it, and uses the stem to find other words with a similar stem and their lemmas. As a result, multiple lemmas are returned for each ALMOR's word input. Therefore, a further disambiguation of lemmas was needed, which is where our algorithm plays an essential role.

By using our IRA matching algorithm, after ALMOR's lemmatization, we were able to filter and disambiguate the lemmas provided by ALMOR, and keep only the correct lemmas. Taking each verb-lemma pair as input, our algorithms decided whether each pair had an implication relationship. For instance, after lemmatizing the 8,731 verbs found in Al-Waseet (No.3 in Table 6.1), 17,721 word pairs were returned by ALMOR; among them incorrect lemmas. After using our matching algorithm to filter out those the incorrect lemmas (i.e., with different incompatible diacritics), we were able to reduce the amount of matching pairs to 4,766 correctly matched pairs. In the case of the Al-Ramooz dictionary (No. 2), which includes 9,866 verbs, ALMOR returned 19,296 different pairs of words. After running the resulting pairs through the IRA algorithm, we were left with 4,575 pairs considered correctly matched.

No	Dictionary	Verbs	Initial Pairs	Correct Pairs
1	قاموس المعاني	4,425	9,750	2,504
2	معجم الرامووز الوسيط للأفعال	9,866	19,296	4,575
3	المعجم الوسيط في تصريف الافعال	8,731	17,721	4,766
4	قاموس المصطلحات العلمية والفنية والهندسية	7,021	4,523	915
5	القاموس النوبي	1,118	1,175	299
6	قاموس الصيدلة	1,101	1,211	316
7	قاموس ومصطلحات اليونسيف	244	245	63
8	قاموس الخدمة الاجتماعية الطبية	254	271	68
9	القاموس الرياضي	276	213	35

Table 6.1 Matching Algorithm Lemma Reduction

Table 6.1 provides a sample of the dictionaries we matched, which provide an evidence of not only the algorithm's success, but also of the *added utility* when used on top of other programs, such as morphological analysers. On average, the amount of word pairs originally proposed by ALMOR were reduced by 74.8% using the IRA algorithm. The remaining 25.2% of the word pairs are all pairs that are certain matches. This ensures the words being linked are, with certainty, the same words under the same lexeme. That is, the matching algorithm can be used with ALMOR or MADA in order to provide further filtration of the analysis and bring more desired results.

7 Conclusion and future work

In this paper, we presented two algorithms that compare a pair of Arabic words with the same non-diacritic characters but with different diacritics. The IRA algorithms encodes expert knowledge in a set of expert rules and reports an implication direction, an implication measure and a matching verdict. The MSA algorithm computes the morphological subsumption relation of one word with respect to the other. We evaluated both algorithms and experiments show that both algorithms are sound and agree on 93% on their recall intersection.

Future plans for the presented algorithms include further testing with an even larger dataset than the over 35,000 pairs used, and using word forms such as nouns and adjectives. Using larger data sets for testing will allow for even more granular fine-tuning of the IRA algorithm's criteria for word-matching. Also planned is the expansion of the algorithms to include taking multi-word phrases as input and returning sets of words considered equal.

Acknowledgement

This research was partially funded by Birzeit University (VerbMesh project) and partially by Google's Faculty Research Award to Prof. Jarrar. The authors are also very thankful to Mohammad Dwaikat, Faeq Rimawi and Reema Taha for helping in the implementation and in the manual evaluation of the results.

8 References

[1] M. A. Attia, "Handling Arabic morphological and syntactic ambiguity within the lfg framework with a view to machine translation," Ph.D. dissertation, University of Manchester, 2008.

[2] G. A. Kiraz, "Arabic computational morphology in the west," Conference and Exhibition on Multilingual Computing, 1998, pp. 101–110.

[3] T. Buckwalter, "Buckwalter Arabic morphological analyser version 1.0," LDC catalog number LDC2002L49, Tech. Rep.

[4] M. Boujelben, C. Aloulou, and L. Hadrich Belguith, "Toward a robust detection/correction system for the agreement errors in non-voweled Arabic texts," in Proc. ACIT 2008.

[5] F. Debili, H. Achour, and E. Souissi, "De letiquetage grammatical a la voyellation automatique de larabe," Tech. Rep. 2002.

[6] N. Chaaben Kammoun, L. Hadrich Belguith, and A. Ben Hamadou, "The morph2 new version: A robust morphological analyser for Arabic texts," ser. JADT 2010, June.

[8] K. R. Beesley, "Finite-state morphological analysis and generation of Arabic at xerox research: Status and plans," in ALP Workshop , France, 2001, pp. 1–8.

[9] S. Kulick, A. Bies, and M. Maamouri, "Consistent and flexible integration of morphological annotation in the Arabic treebank," in Proc. LREC'2010, Malta,

[10] D. Vergyri and K. Kirchhoff, "Automatic diacritization of Arabic for acoustic modeling in speech recognition," in Proc. of CAASL, ACL, 2004, pp. 66–73

[12] A. M. Seraye, "The role of short vowels and context in the reading of Arabic, comprehension and word recognition of highly skilled readers," Ph.D. Thesis 2004.

[13] M. Khorsheed, "An HMM-based system to diacritize Arabic text," SEA, V. 5, 2013.

[14] A. Said, M. El-Sharqwi, A. Chalabi, and E. Kamal, "A hybrid approach for Arabic diacritization," in NLPIS Systems. Springer, 2013, pp. 53–64.

[15] A. M. Hattab and A. K. Hussain, "Hybrid statistical and morphosyntactical Arabic language diacritizing system." Journal of Academic Research, vol. 4, no. 4, 2012.

[16] A. O. Bahanshal and H. S. Al-Khalifa, "A first approach to the evaluation of Arabic diacritization systems," (ICDIM), IEEE, 2012, pp. 155–158.

[17] S. Alqrainy, H. AlSerhan, and A. Ayesh, "Pattern-based algorithm for part-of-speech tagging Arabic text," ICCES 2008. pp. 119–124.

[18] Arabic Morphological Tagging, Diacritization, and Lemmatization Using Lexeme Models and Feature Ranking. Columbus, Ohio: ACL, 06/2008.

[20] N. Habash, O. Rambow, and R. Roth, "Mada+tokan: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization," in Proc. ICALRT, MEDAR, 2009.

[21] M. Rashwan, M. Al-Badrashiny, M. Attia, S. Abdou, and A. Rafea,"A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features," IEEE TASLP, vol. 19, no. 1, pp. 166–175, 2011.

[22] N. Habash, "Arabic Morphological Representations for Machine Translation", book chapter, Vol. 38, pp 263-285