# A Word Matching Algorithm in Handwritten Arabic Recognition Using Multiple-Sequence Weighted Edit Distances

**Gheith A. Abandah**[1] **and Fuad T. Jamour**[2]

[1] **Computer Engineering Department, The University of Jordan**
**Amman 11942, Jordan**

[2] **King Abdullah University of Science and Technology**
**Thuwal, Saudi Arabia**

**Abstract**

No satisfactory solutions are yet available for the offline recognition of handwritten cursive words, including the words of Arabic text. Word matching algorithms can greatly improve the OCR output when recognizing words of known and limited vocabulary. This paper describes the word matching algorithm used in the JU-OCR2 optical character recognition system of handwritten Arabic words. This system achieves state-of-the-art accuracy through multiple techniques including an efficient word matching algorithm. This algorithm reduces the average sequence error for the IfN/ENIT database of handwritten Arabic words from 32.3% to an average word error of just 5.0%. This algorithm is a weighted version of the edit distance algorithm. The weighted version has a 5.0% advantage over the plain edit distance algorithm. This algorithm selects the best match utilizing a set of multiple probable sequences from the sequence transcription stage. Using multiple sequences, instead of one, reduces the average error by 27.0% over the weighted edit distance algorithm. Compared with an algorithm used in a leading system, this algorithm offers 6.7% lower average word error for the main two test sets.

***Keywords:*** *Optical Character Recognition, Handwritten Arabic Words, Word Matching, Edit Distance.*

## 1. Introduction

Offline cursive handwriting recognition is a hard problem for which no satisfactory general solutions are yet available. Major challenges include the overlap and interconnection of neighboring characters, the huge variability in both quality and style of human handwriting, and the similarities of distinct character shapes [1, 2].

Arabic is the native language of more than 440 million people and its alphabet is used in around 27 languages, including Arabic, Persian, Kurdish, Urdu, and Jawi [3]. Arabic is always cursive in print and in handwriting. Despite decades of research, there is still a lack of accurate Arabic handwriting recognition systems [4].

Post processing is often used to improve the recognition accuracy of handwritten words [5, 6]. In limited vocabulary applications, a word matching algorithm is used to map the output words to the closest lexicon words.

The JU-OCR2 system for recognizing handwritten Arabic words achieves high accuracy using efficient segmentation, feature extraction, recurrent neural network (RNN), and word matching [7]. This paper focuses on the word matching stage that is used in JU-OCR2.

This paper describes the experimental method used to develop an efficient word matching algorithm for recognizing handwritten Arabic words. It describes this algorithm and evaluates its performance. The main contributions of this paper are summarized as follows:

- Multiple popular string distance measures are evaluated to select the most efficient measure.
- A weighted distance measure is developed to take into consideration the similarities among the Arabic letters.
- A novel accurate word matching algorithm is described that uses multiple-sequence weighted edit distances.

This paper is organized as follows: The rest of this section reviews the related work and briefly describes the JU-OCR2 recognition system. Section 2 describes our experimental setup including the samples used in this work. Section 3 describes and evaluates six popular string distance measures. Section 4 describes and evaluates a weighted distance measure appropriate for recognizing Arabic words including an optimization that leverages word prior probabilities. Section 5 presents and evaluates the final algorithm developed which uses multiple-sequence weighted edit distances. Section 6 concludes by discussing the accuracy of this algorithm and compares its accuracy with the accuracy of a leading algorithm.

## 1.1 Related Work

Navarro performed a detailed study on the problem of *approximate string matching* [5]. The general goal is to perform string matching of a pattern in a text where one or both of them have suffered some kind of corruption.

Many general algorithms have been developed for string matching [8, 9, 10, 11, 12, 13]. These algorithms are described and evaluated in Section 3.

Märgner, El Abed, and Pechwitz have organized a series of Arabic handwriting recognition competitions [4, 14, 15, 16, 17]. The purpose of these competitions is to advance the research and development of Arabic handwritten word recognition systems. These competitions use the IfN/ENIT database [18] and have had excellent participations from the research leaders in the area. The participations have shown remarkable progress over seven years. The following paragraphs review some of the best systems concentrating on their word matching algorithms.

The UOB system developed by Al-Hajj *et al.* won the first competition in ICDAR 2005. This system uses HMM character and word modeling [19]. Each word of the lexicon is built by concatenating the appropriate character models. Thus, no explicit word matching stage is needed.

The competitions organizers also participated with their ARAB-IFN system in ICDAR 2005. This system got the second place and is described in [20]. They also use HMM to recognize individual characters. However, their recognition process assigns to an unknown feature sequence a valid word from the lexicon. This is done by calculating the probability that the observation was produced by a state sequence for each word of the lexicon, and then selecting the sequence with the highest probability.

The Siemens system submitted by Alary *et al.* was the winner of the ICDAR 2007 competition. This system was adapted for Arabic script from the standard HMM-based Latin script word recognizer that is widely in use within Siemens AG for postal automation projects [21]. They use HMM to recognize individual characters and then they find the best word in the lexicon that comprises the words writing variants.

The MDLSTM system developed by Graves was the winner of the ICDAR 2009 competition [22]. This system holds the current record for recognition accuracy on the main competition test set. It has a connectionist temporal classification (CTC) token passing algorithm for word matching algorithm that is integrated in the RNN CTC output layer.

The UPV-PRHLT system developed by Alkhoury *et al.* was the winner of the ICFHR 2010 competition. This system uses windows of raw, binary image pixels, which are directly fed into embedded Bernoulli HMMs [23]. It recognizes the most likely word using the lexicon image models that each is modeled as a BHMM (built from shared, embedded BHMMs at the character level).

The RWTH-OCR system developed by Dreuw *et al.* was the winner of the ICDAR 2011 competition. This system uses sliding window for HMM-based handwriting recognition [24]. It uses 120 glyph models for the possible Arabic letters presentation forms. A word is modeled by a sequence of glyph models.

All the HMM-based systems reviewed above do not have explicit word matching algorithms. They recognize words through their HMM models. Unlike these systems, JU-OCR2 has a sequence transcription stage that recognizes characters and does not have word models. Thus, word matching is needed as a post processing stage to improve the accuracy when a lexicon is available.

## 1.2 OCR-JU2 Overview

JU-OCR 2 is a system for recognizing handwritten Arabic words. It achieves high accuracy using efficient segmentation, feature extraction, and recurrent neural network (RNN). Figure 1 summarizes its five main stages: (*i*) sub-word segmentation, (*ii*) grapheme segmentation, (*iii*) feature extraction, (*iv*) sequence transcription, and (*v*) word matching.

This system uses a robust rule-based segmentation algorithm that uses special feature points identified in the word skeleton to segment the cursive words into graphemes. An efficient set of features extracted from the graphemes are fed to a tuned RNN that transcribes the sequences of feature vectors to character sequences. This RNN was implemented using the publically available RNNLIB library [25]. The used word matching algorithm gives ordered list of the ten most probable lexicon (dictionary) words for each word image. This paper concentrates on the word matching algorithm. For more details about JU-OCR2, refer to Ref. [7].

Word matching can greatly improve accuracy since many transcription mistakes that result in non-lexicon words are corrected. The example in Fig. 1 shows how word matching corrects the miss-transcribed last letter from **Lam** (ل) to the letter **Reh** (ر), drawn underlined.

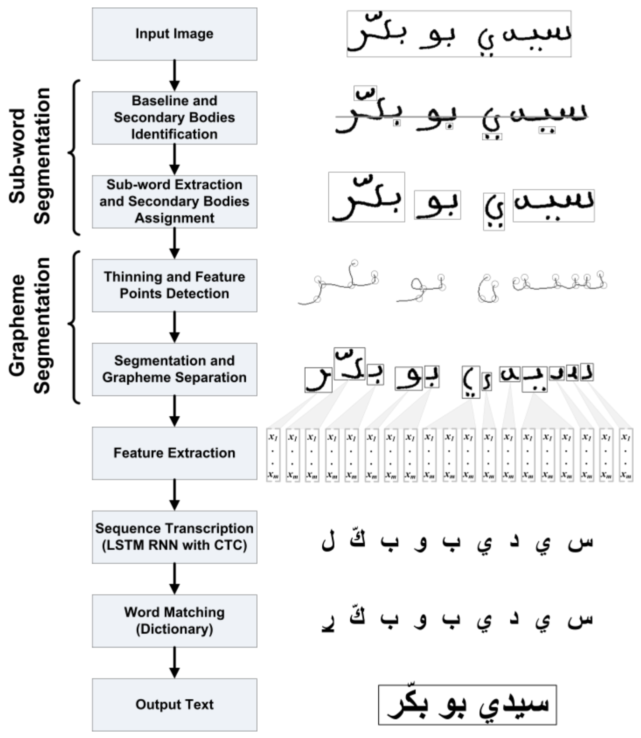
Input Image
Sub-word Segmentation
Baseline and Secondary Bodies Identification
Sub-word Extraction and Secondary Bodies Assignment
Grapheme Segmentation
Thinning and Feature Points Detection
Segmentation and Grapheme Separation
Feature Extraction
Sequence Transcription (LSTM RNN with CTC)
Word Matching (Dictionary)
Output Text
سيدي بو بكّر

Fig. 1 Processing stages of the JU-OCR2 Arabic handwriting recognition system.

## 2. Experimental Setup

This section describes the samples used in developing and evaluating the word matching algorithm. We also report here the base recognition accuracy without using word matching.

### 2.1 Samples

Our experiments are based on the IfN/ENIT database of handwritten Arabic words [18]. This database is used by more than 110 research groups in about 35 countries [4]. The database version used is v2.0p1e and consists of 32,492 Arabic words handwritten by more than 1,000 writers. These words are 937 Tunisian town/village names. This database is divided into five training sets and two test sets. The numbers of names, parts of Arabic words (PAWs), and characters in the seven sets are shown in Table 1.

In the rest of this paper, we show results of experiments carried out by training the system using some of these sets and testing using some other set. We refer to each experiment by its training sets followed by its test set, separated by a hyphen. For example, *abcd-e* experiment indicates that the training sets are *a* through *d* and the test set is *e*.

Table 1: The IfN/ENIT database of handwritten Arabic words

| | Set | Names | PAWs | Characters |
|---|---|---|---|---|
| Training sets | *a* | 6,537 | 28,298 | 51,984 |
| | *b* | 6,710 | 29,220 | 53,862 |
| | *c* | 6,477 | 28,391 | 52,155 |
| | *d* | 6,735 | 29,511 | 54,166 |
| | *e* | 6,033 | 22,640 | 45,169 |
| Test sets | *f* | 8,671 | 32,918 | 64,781 |
| | *s* | 1,573 | 6,109 | 11,922 |

We randomly select 90% of the training sets samples for training the RNN and the rest 10% for validation.

### 2.2 Raw Sequence Error

Figure 2 shows the sequence error of the seven sets and the average sequence error without any application of word matching. The *sequence error* is the ratio of sequences (words) that have one or more character recognition errors.

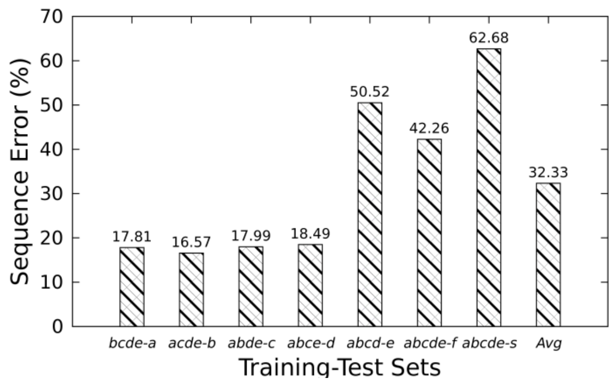

Fig. 2 The sequence error for the seven IfN/ENIT sample sets. For each experiment, the label shows the training sets followed by the test set, separated by a hyphen.

Note that the five training sets (*a* through *e*) are each tested using an RNN trained using the rest four training sets. Whereas, the two test sets (*f* and *s*) are tested using an RNN trained using the five training sets.

The sequence error for the seven sets ranges from 16.57% to 62.68%, reflecting large differences among these sets. Due to differences in numbers of writers per set and writing quality, set *s*, is hardest to recognize, followed by sets *e* and *f*. Whereas sets *a* through *d* are easier to recognize. In fact, set *s* is particularly different from the other sets as it was collected from writers of another country.

## 3. Distance Measures

The output of the word matching algorithm is the lexicon word that is in some sense "closest" to the sequence transcriber's output $y$. More formally, if the lexicon is a set of words $S$, the word matching stage outputs the word $o \in S$ where

$$o = \arg \min_{s \in S} d(y, s) \tag{1}$$

The function $d(y,s)$ returns a *distance measure* between two sequences, where a lower value indicates that the two sequences are closer. We have experimented with six popular distance functions to find the most suitable function: Hamming $d_h$, Jaro $d_j$, Jaro-Winkler $d_w$, Ratcliff/Obershelp $d_r$, Levenshtein $d_l$, and Damerau-Levenshtein $d_d$. These distance measures are described and evaluated in the following subsections.

### 3.1 Hamming Distance

Hamming distance $d_h$ [8] between two strings of equal length is the number of mismatching letters between the two string. When the strings are not of equal length, the difference between their lengths is added to the Hamming distance between the shorter string and a prefix of length $|s|$ of the longer string, where $|s|$ is the length of the shorter string. For example, the Hamming distance between SOURCE and SUORCE is 2.

Figure 3 shows the word error for the five IfN/ENIT training sets using Hamming distance. The figure also shows the average word error of these five sets. The *word error*, similar to the sequence error, is the ratio of words that have one or more character recognition errors. However, we reserve using the word error as a metric to evaluate the output of the word matching stage. Consistent with the sequence errors, the word error of set *e* is higher than the word errors of sets *a* through *d*.

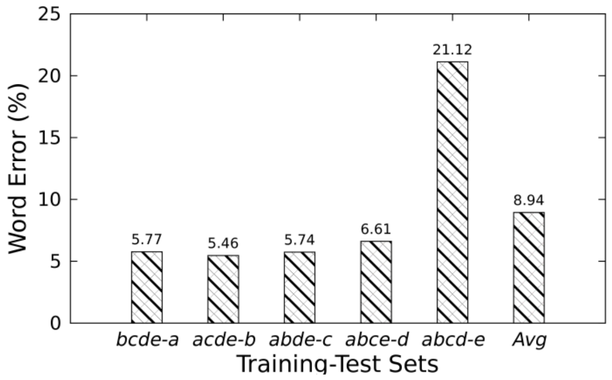

Fig. 3 The word error for the five training sets using Hamming distance $d_h$.

### 3.2 Jaro Distance

Jaro distance $d_j$ is described in Ref. [9]. Let

$$MD = \left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1 \tag{2}$$

be the *match distance* between strings $s_1$ and $s_2$, which is the maximum allowed distance between two equal letters in the two strings to be considered a match. The *number of matches* $m$ is the number of equal letters that are within $MD$ distance from each other. The *number of transpositions* $t$ is the number of matching letters that are not at the same position in their respective strings divided by two. Jaro string similarity is defined to be:

$$s_j = \begin{cases} 0, & \text{if } m \text{ is zero} \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right), & \text{otherwise} \end{cases} \tag{3}$$

Jaro similarity is between 0 and 1, 0 being very different strings and 1 being exact match. In our work, our convention is that smaller distance means more similar strings, so what we actual use is $d_j = 1 - s_j$. For example, the Jaro distance between SOURCE and SUORCE is $1 - 0.944$. In this example, the values of $MD$, $m$, $|s_1|$, $|s_2|$, and $t$ are: 2, 6, 6, 6, and 1, respectively.

Figure 4 shows the word error for the five sets using Jaro distance. Jaro distance is much better than Hamming distance for these samples. Its word error is less than half the word error of the Hamming distance.

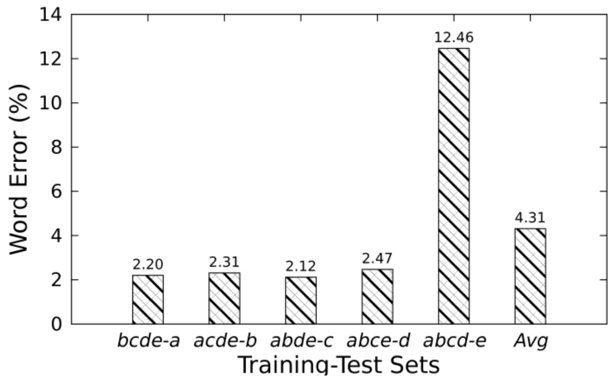

Fig. 4 The word error for the five training sets using Jaro distance $d_j$.

### 3.3 Jaro-Winkler Distance

Jaro-Winkler distance $d_w$ [10] is based on Jaro distance with a slight modification to give equal prefixes some weight. Let $l$ be the length of the common prefix between the two strings, and $p$ be a parameter between 0 and 0.25.

If $l$ exceeds 4, it is limited to 4 to keep the Jaro-Winkler similarity between 0 and 1. Jaro-Winkler string similarity is defined as:

$$s_w = s_j + l \times p \times (1 - s_j) \tag{4}$$

Again, we use $d_w = 1 - s_w$ in our work to get similar strings to have smaller distances. For example, the Jaro-Winkler distance between SOURCE and SUORCE is $1 - 0.944 + 1 \times 0.1 \times (1 - 0.944) = 1 - 0.950$; $p$ is set to 0.1 here.

Figure 5 shows the word error for the five sets using Jaro-Winkler distance. Comparing with Jaro distance, set *e* benefits form the Winkler optimization. The four other sets get higher word errors. However, the word error averages for Jaro and Jaro-Winkler distances are identical.

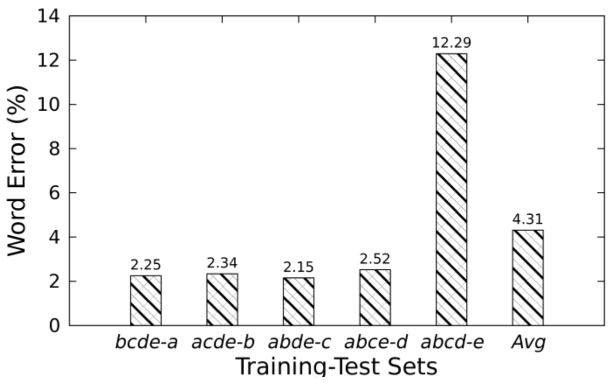

Fig. 5 The word error for the five training sets using Jaro-Winkler distance $d_w$.

## 3.4 Ratcliff/Obershelp Distance

In Ratcliff/Obershelp pattern matching algorithm, the similarity of two strings $s_r$ is found as the doubled number of *matching characters* divided by the total number of characters in the two strings [11]. Matching characters are those in the longest common subsequence plus, recursively, matching characters in the unmatched region on either side of the longest common subsequence.

We compute Ratcliff/Obershelp distance as $d_r = 1 - s_r$. For example, the Ratcliff/Obershelp distance between SOURCE and SUORCE is $1 - 2 \times (1 + 1 + 3)/(6 + 6) = 1 - 0.833$.

Figure 6 shows the word error for the five sets using Ratcliff/Obershelp distance. This distance measure is better than $d_w$ by 5.8% on average.

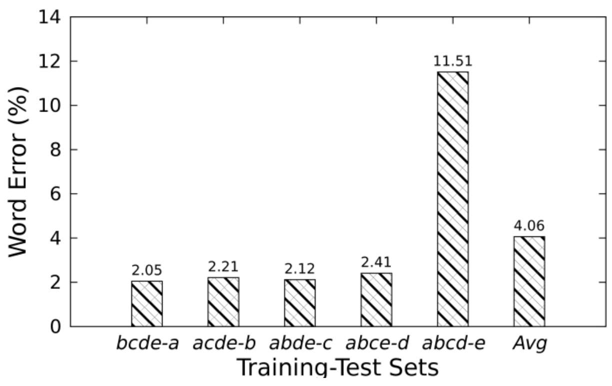

Fig. 6 The word error for the five training sets using Ratcliff/Obershelp distance $d_r$.

## 3.5 Levenshtein Distance

The Levenshtein distance is commonly known as the *edit distance* [12]. The edit distance between sequence $y$ and word $s$ is the minimum total number of insertions, deletions, and changes required to change pattern $y$ to word $s$. Let $d_{ij}$ be the edit distance between a prefix from sequence $y$ of length $i$ and a prefix from sequence $s$ of length $j$, which is defined by the recurrence

$$\begin{gathered} d_{i0} = i \text{ for } 0 \le i \le m = |y| \\ d_{0j} = j \text{ for } 0 \le j \le n = |s| \\ d_{ij} = min\begin{pmatrix} d_{i-1,j} + 1, \\ d_{i,j-1} + 1, \\ d_{i-1,j-1} + \text{diff}(y_i, s_j) \end{pmatrix} \\ \text{diff}(y_i, s_j) = \begin{cases} 0 & \text{if } y_i = s_j \\ 1 & \text{if } y_i \ne s_j \end{cases} \end{gathered} \tag{5}$$

The edit distance equals the edit distance of the entire two sequences $d_l = d_{mn}$.

Figure 7 shows the word error for the five sets using the edit distance. This distance measure is much better than the previous distances and is better than $d_r$ by 10.6% on average.

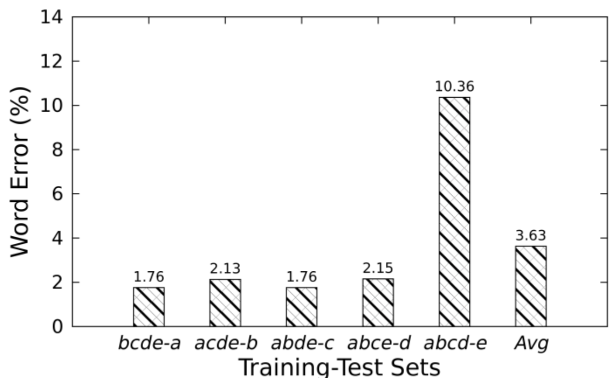

Fig. 7 The word error for the five training sets using the edit distance $d_l$.

## 3.6 Damerau-Levenshtein Distance

Damerau-Levenshtein distance $d_d$ is based on Levenshtein distance [13]. The only difference is that adjacent letter transpositions are allowed as transformation operations in addition to insertions, deletions, and changes. For example, while the Lavenshtein distance between SOURCE and SUORCE is 2 (one insertion and one deletion), the Damerau-Levenshtein distance between these two strings is 1 (one transposition of the "U" and the "O").

Figure 8 shows the word error for the five sets using Damerau-Levenshtein distance. This distance measure has an average word error equal to that of the edit distance. However, $d_l$ gives lower error rate for sets *a*, *b*, and *c;* and $d_d$ gives better rate for set *e*.

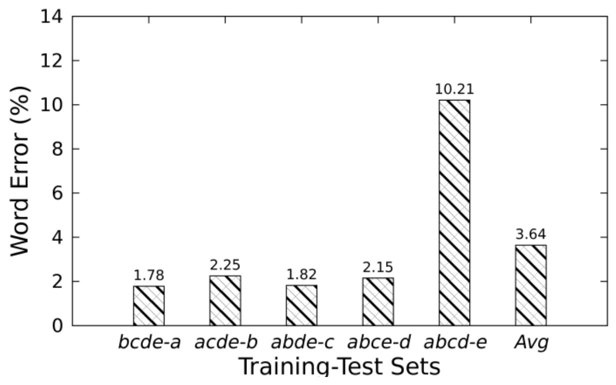

Fig. 8 The word error for the five training sets using Damerau-Levenshtein distance $d_d$.

Among these six distances, the best two are the edit distance and Damerau-Levenshtein distance. We adopted these two distance measures for further improvement as described below.

## 4. Weighted Distance Measures

We have noticed that many recognition errors include output letters that have similar shapes to their respective target letters, as is the example in Fig. 1 where **Lam** (ل) is the output for the target **Reh** (ر). This has motivated us to experiment with *weighted* edit distance measures [26].

We developed weighted versions for the edit distance $d_l$ and Damerau-Levenshtein distance $d_d$ called $WED$ and $WDL$, respectively. In these two versions, the function $\text{diff}(y_i, s_j)$ of Eq. (5) is modified to return weights ranging from 0 to 1 instead of returning 0 or 1. The return value reflects how different the two letters are. This function uses a *lookup table* that was created based on the Arabic letter shape similarities. For example, as the letters **Beh** (ب) and **Teh** (ت) have similar shapes, they have low diff value. And as letters **Alef** (ا) and **Seen** (س) are dissimilar, they have high diff value.

We have built the lookup table through two techniques:

**Weights 1**: The weights are manually estimated based on our expert knowledge of the similarities among the Arabic letters.

**Weights 2**: The weights are computed from the confusion matrix of the recognition errors of the training sets [27]. The change weight (penalty) for letters $a$ and $b$ is computed as:

$$\text{diff}(a,b) = \begin{cases} 0 & \text{if } a = b \\ 1 - \text{freq}(a,b) & \text{if } a \neq b \end{cases} \qquad (6)$$

where $\text{freq}(a,b)$ is the frequency of output $b$ for letter $a$ of the confusion matrix of the recognition errors.

Figure 9 shows the word error for the five sets using $WED$ and $WDL$ word matching algorithm with the Weights 1 technique. This figures illustrates that the weighted distance modification improves the word error by averages of 5.0% and 5.8% for $WED$ and $WDL$, respectively, compared with $d_l$ and $d_d$. The following section gives an evaluation for the Weights 1 and Weights 2 techniques.

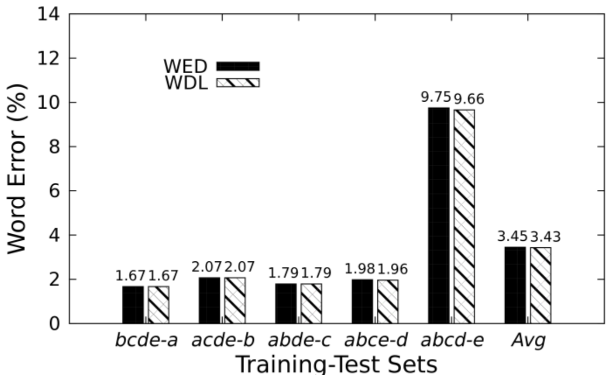

Fig. 9 The word error for the five training sets using weighted edit distance ($WED$) and weighted Damerau-Levenshtein distance ($WDL$) with Weights 1.

### 4.1 Leveraging Prior Probabilities

The word prior probabilities $P(s)$ in the training set $s \in S$ are not the same. Therefore, we experimented leveraging these probabilities by modifying Eq. (1) as follows:

$$o = \arg\min_{s \in S} d(y,s) \times P(s) \qquad (7)$$

Here the matching criterion selects the word that has the minimum product of distance and prior probability. Figure 10 shows the word error for the five sets using two versions of the weighted edit distance algorithm: One that does not use word probabilities ($WED$) and one that does ($WEDp$). We also use Weights 1 in this comparison. Although $WED$ is on average better than $WEDp$, $WEDp$ gives better results with the difficult set *e*.

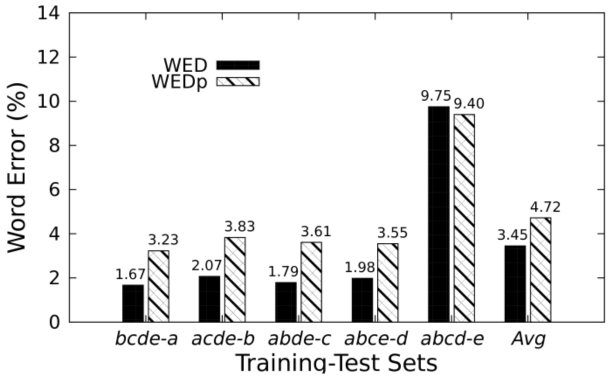

Fig. 10 The word error for the five training sets using weighted edit distance without/with word prior probabilities ($WED$/$WEDp$) with Weights 1.

The $WDL$ word matching has similar behavior with respect to using the word prior probabilities as $WED$.

## 5. Multiple-Sequence Weighted Edit Distances

The RNN of the sequence transcription stage outputs the sequence $y$ that has highest probability $p(y|\boldsymbol{x})$ given the observed input features sequence $\boldsymbol{x}$. The RNN can optionally output the set $Y$ of the $n$ most probable output sequences ($y \in Y$, of highest $p(y|\boldsymbol{x})$). We exploit these multiple output sequences through the following equation:

$$D(Y,s) = \sum_{y \in Y} (1 - p(y|\boldsymbol{x})) \times d(y,s) \qquad (8)$$

where $D(Y,s)$ is the composite distance of the multiple sequences $Y$ from word $s$ and $p(y|\boldsymbol{x})$ is the output conditional probability. We use $(1 - p(y|\boldsymbol{x}))$ instead of $p(y|\boldsymbol{x})$ to get higher distances for lower probabilities.

The distance $d(y,s)$ can be any of the above distances. However, we only experimented with the two best distance measures in this approach; the edit distances $d_l$ and $d_d$. In this *multiple-sequence weighted edit distances* approach, $D(Y,s)$ replaces $d(y,s)$ in Eqs. (1) and (7). We refer to the variant word matching algorithm that uses $d_l$ by $WED(n)$ and the one that uses $d_d$ by $WDL(n)$.

Figure 11 shows the average word error of the five sets when using $WED(n)$ word matching algorithm as a function of the number of RNN output sequences $n$. The figure shows the results for the two weight types.

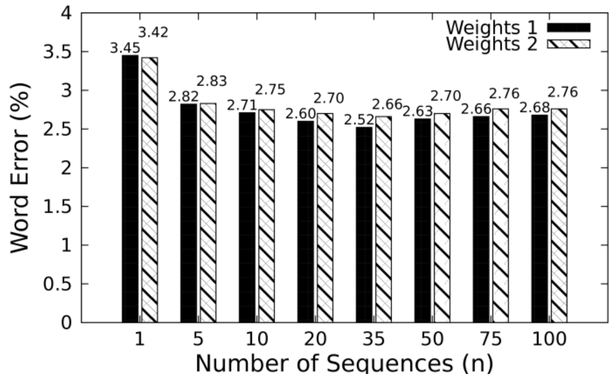

Fig. 11 The average word error of the five training sets using multiple-sequence weighted edit distances ($WED(n)$) as function of the number of sequences $n$ for the two weight types.

The word error decreases as $n$ increases from 1 to 35 and increases for $n > 35$, indicating that considering multiple output sequences improves accuracy. However, when too many output sequences are considered, the accuracy worsens due to including too far sequences.

As for the comparison between the two weight types, Weights 2 is better than Weights 1 only when $n = 1$. This is expected as the weights in Weights 2 are directly related to the recognition errors of the first output sequence. Weights 1 is superior for $n \geq 5$, indicating that the expertly-selected weights perform better beyond the first output sequence. As the best result is achieved with $n = 35$ and Weights 1, we adopt $n = 35$ and Weights 1 in the final algorithm.

Figure 12 shows a comparison between $WED(n)$ and $WDL(n)$ using Weights 1. Although for some $n$ values, $WDL(n)$ has lower error than $WED(n)$, lowest error is obtained at $n = 35$ with $WED(n)$. This indicates that the adjacent letter transposition optimization of the Damerau-Levenshtein distance that is useful in correcting typing mistakes is not very useful in correcting OCR errors.

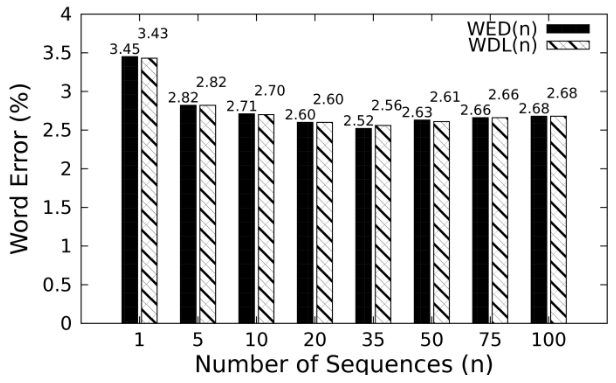

Fig. 12 The average word error of the five training sets using multiple-sequence weighted edit distances ($WED(n)$) and multiple-sequence weighted Damerau-Levenshtein distances ($WDL(n)$) as functions of the number of sequences $n$.

## 6. Discussion and Conclusions

The previous sections summarize our work in developing an accurate word matching algorithm for recognizing handwritten Arabic words. We noticed that the best distance measure for this application is Levenshtein's edit distance $d_l$. This distance reduces the average sequence error of the five training sets from 24.28% to an average word error of 3.63% for the five training sets. This demonstrates that word matching is very effective in this application.

The weighted edit distance $WED$ improvement reduces the average word error to 3.45%. This is a 5.0% improvement over the plain edit distance and using Weights 1 that were expertly selected. The multiple-sequence $WED(n)$ improvement at $n = 35$ further reduces this average word error to 2.52%. This is a significant 27.0% improvement over $WED$.

We noticed that the best algorithm for sets *a* through *d* is $WED(35)$. However, the best algorithm for set *e* is the variant that uses the word prior probabilities $WEDp(n)$. These best results for the five training sets are shown in Fig. 13. When testing these two algorithms with the two test sets, we found that $WEDp(n)$ also gives best results. The best word errors for sets *f* and *s* are also shown in Fig. 13 and are 7.54% and 15.20%, respectively. Although the sequence error of set *e* is larger than that of set *f*, set *e* has a lower word error. This indicates that set *e* benefits more than set *f* from word matching.

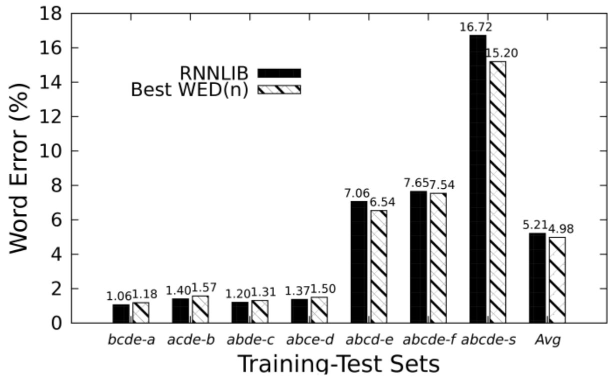

Fig. 13 The word error for the seven sets using the RNNLIB word matching and the best multiple-sequence weighted edit distances ($WED(n)$ or $WEDp(n)$).

Figure 13 also shows a comparison between our best results and the RNNLIB word matching algorithm [25]. This algorithm was used in the MDLSTM system which is the winner of the ICDAR 2009 competition and the system that holds the record score on set *f* [4]. This algorithm is integrated in the CTC output layer and is the CTC token passing algorithm described in Refs. [28, 29]. Through this algorithm the CTC labeling is constrained to only the sequences of the complete dictionary words. For any word that has variant spellings, this algorithm sums the probabilities of all the word's variants to find the word probability.

Although RNNLIB has better performance on sets *a* through *d*, our algorithm gives better average error and better results for the difficult test sets. For the important test sets *f* and s, our algorithm is better than RNNLIB word matching by an average 6.7%.

**Acknowledgment**

This work was supported by the Deanship of the Scientific Research in the University of Jordan. We would like to thank Alex Graves for making the RNNLIB publically available [25], for giving us a copy of the latest RNNLIB version, and for his help in using it. We would like also to thank Haikal El Abed for giving us copies of sets *f* and *s* of the IfN/ENIT database.

## References

[1] N. Arica, and F. Yarman-Vural, "Optical Character Recognition for Cursive Handwriting", IEEE Trans Pattern Anal Mach Intell, Vol. 24, No. 6, 2002, pp. 801-813.
[2] H. Lee, and B. Verma, "Binary Segmentation Algorithm for English Cursive Handwriting Recognition", Pattern Recognition, Vol. 45, No. 4, 2012, pp.1306-1317.
[3] M. P. Lewis (ed.), Ethnologue: Languages of the World, SIL International, Dallas, TX, 2009.
[4] V. Märgner, and H. El Abed, "ICDAR 2011 - Arabic Handwriting Recognition Competition", in Int'l Conf. Document Analysis and Recognition, 2011, pp. 1444-1448.
[5] G. Navarro, "A Guided Tour to Approximate String Matching", ACM Computing Surveys, Vol. 33, No. 1, 2001, pp. 31-88.
[6] V. SaiKrishna, A. Rasool, and N. Khare, "String Matching and its Applications in Diversified Fields", Int'l Journal of Computer Science Issues, Vol. 9, No. 1, 2012, pp. 219-226.
[7] G. Abandah, F. Jamour, and E. Qaralleh, "Recognizing Handwritten Arabic Words Using Grapheme Segmentation and Recurrent Neural Networks", Int'l J. Document Analysis and Recognition, Springer, 2014, DOI: 10.1007/s10032-014-0218-7.
[8] R. W. Hamming, "Error Detecting and Error Correcting Codes", Bell System Technical Journal, Vol. 29, No. 2, 1950, pp. 147-160.
[9] M. A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida", Journal of the American Statistical Association, Vol. 84, No. 406, 1989, pp. 414-420.
[10] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage", in Section on Survey Research Methods, American Statistical Assn., 1990, pp. 354-359.
[11] J. Ratcliff, and D. Metzener, "Pattern Matching: The Gestalt Approach", Dr. Dobb's Journal, Vol. 13, No. 7, 1988, pp. 46-72.
[12] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals", Soviet Physics Doklady, Vol. 10, No. 8, 1966, pp. 707-710.
[13] F. J. Damerau, "A Technique for Computer Detection and Correction of Spelling Errors", Communications of the ACM, Vol. 7, No. 3, 1964, pp. 171-176.
[14] V. Märgner, M. Pechwitz, and H. El Abed, "ICDAR 2005 - Arabic Handwriting Recognition Competition", in Int'l Conf. Document Analysis and Recognition, 2005, pp. 70-74.
[15] V. Märgner, and H. El Abed, "ICDAR 2007 - Arabic Handwriting Recognition Competition", in Int'l Conf. Document Analysis and Recognition, 2007, pp. 1274-1278.
[16] V. Märgner, and H. El Abed, "ICDAR 2009 - Arabic Handwriting Recognition Competition", in Int'l Conf. Document Analysis and Recognition, 2009, pp. 1383-1387.
[17] V. Märgner, and H. El Abed, "ICFHR 2010 - Arabic Handwriting Recognition Competition", in Int'l Conf. Frontiers in Handwriting Recognition, 2010, pp. 709-714.
[18] M. Pechwitz, S. S. Maddouri, V. Märgner, N. Ellouze, and H. Amiri, "IFN/ENIT - Database of Handwritten Arabic Words", in 7th Colloque Int'l Francophone sur l'Ecrit et le Document, 2002, pp. 129-136.
[19] R. Al-Hajj, L. Likforman-Sulem, and C. Mokbel, "Arabic Handwriting Recognition Using Baseline Dependent Features and Hidden Markov Modeling", in Int'l Conf. Document Analysis and Recognition, 2005, pp. 893-897.
[20] M. Pechwitz, H. El Abed, and V. Märgner, "Handwritten Arabic Word Recognition Using the IFN/ENIT-Database", in V. Märgner, H. El Abed (eds.), Guide to OCR for Arabic Scripts, Springer, London, 2012, pp. 169-213.
[21] M. P. Schambach, J. Rottland, and T. Alary, "How to Convert a Latin Handwriting Recognition System to Arabic", in Int'l Conf. Frontiers in Handwriting Recognition, 2008.
[22] A. Graves, and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks", in Advances in Neural Information Processing Systems, 2009, vol. 22, pp. 545-552.
[23] I. Alkhoury, A. Giménez, and A. Juan, "Arabic Handwriting Recognition Using Bernoulli HMMs", in V. Märgner, H. El Abed (eds.), Guide to OCR for Arabic Scripts, Springer, London, 2012, pp. 255-272.
[24] P. Dreuw, D. Rybach, G. Heigold, and H. Ney, "RWTH OCR: A Large Vocabulary Optical Character Recognition System for Arabic Scripts", in V. Märgner, H. El Abed (eds.), Guide to OCR for Arabic Scripts, Springer, London, 2012, pp. 215-254.
[25] A. Graves, "RNNLIB: A Recurrent Neural Network Library for Sequence Learning Problems", http://sourceforge.net/projects/rnnl/
[26] E. Ristad, and P. Yianilos, "Learning String-Edit Distance", IEEE Trans Pattern Anal Mach Intell, Vol. 20, No. 5, 1998, pp. 522-532.
[27] K. Audhkhasi, and A. Verma, "Keyword Search Using Modified Minimum Edit Distance Measure", in IEEE Int'l Conf. Acoustics, Speech and Signal Processing, 2007. Vol. 4, pp. 929-932.
[28] A. Graves, S. Fernández, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks", Advances in Neural Information Processing Systems, Vol. 20, No. 1-8, 2008, pp. 577-584.
[29] A. Graves, "Offline Arabic Handwriting Recognition with Multidimensional Recurrent Neural Networks", in V. Märgner, H. El Abed (eds.), Guide to OCR for Arabic Scripts, Springer, London, 2012, pp. 297-313.

**Gheith A. Abandah** has received MSE and PhD in Computer Science and Engineering from the University of Michigan in 1995 and 1998. He has been responsible for several funded projects in the areas of Arabic text recognition and electronic voting. He has more than 15 years of industrial experience in localization, military electronics, product and project development and deployment. He is an associate professor with the University of Jordan.

**Fuad T. Jamour** received a BS degree in Computer Engineering from the University of Jordan in 2011 with top honors. He has worked on handwritten Arabic word recognition as an undergraduate research assistant and also in his BS project. He is currently a graduate student in King Abdullah University of Science and Technology (KAUST), and his interests include data management, cloud computing, and pattern recognition.