International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 25, No. 7 (2011) 1009–1033

DOI: 10.1142/S0218001411008956

World Scientific
www.worldscientific.com

SEGMENTATION-FREE ONLINE ARABIC HANDWRITING RECOGNITION

FADI BIADSY

Computer Science, Columbia University
New York, NY 10027, USA
fadi@cs.columbia.edu

RAID SAABNI

Computer Science, Ben-Gurion University of The Negev
Beer-Sheva, 84105, Israel

Triangle Research & Development Center
Kafr Qara, 30075, Israel
saabni@cs.bgu.ac.il

JIHAD EL-SANA

Computer Science, Ben-Gurion University of The Negev
Beer-Sheva, 84105, Israel
el-sana@cs.bgu.ac.il

Arabic script is naturally cursive and unconstrained and, as a result, an automatic recognition of its handwriting is a challenging problem. The analysis of Arabic script is further complicated in comparison to Latin script due to obligatory dots/stokes that are placed above or below most letters. In this paper, we introduce a new approach that performs online Arabic word recognition on a continuous word-part level, while performing training on the letter level. In addition, we appropriately handle delayed strokes by first detecting them and then integrating them into the word-part body. Our current implementation is based on Hidden Markov Models (HMM) and correctly handles most of the Arabic script recognition difficulties. We have tested our implementation using various dictionaries and multiple writers and have achieved encouraging results for both writer-dependent and writer-independent recognition.

Keywords: Online handwriting recognition; Arabic; HMM.

1. Introduction

Keyboards and electronic mice may not endure as the prevalent means of human-computer interfacing. Devices such as digital tablets, hand-held computers, and mobile technology, provide significant opportunities for alternative interfaces that work in forms smaller than the traditional keyboard and mouse. In addition, the need for more natural human-computer interfaces becomes ever more important as

computer use reaches a larger number of people. Two such natural alternatives to typing are speech and handwriting, which are universal human communication methods. Both are potentially easier human-computer interfaces to learn by new users compared to keyboards. Although a handwriting interface expects users to be literate, it ensures a higher degree of privacy and confidentiality compared to speech.

Automatic handwriting recognition has been classified into two categories, *offline* and *online*, based on the presentation of the data to the system. *Offline handwriting recognition* approaches do not require immediate interaction with users. A scanned handwritten or printed text is fed to the system in a digital image format. In a typical *online handwriting recognition* approach, a special stylus is used to write on a digital device, such as a digital tablet. The digitized samples are fed to the system as a sequence of 2D-points in real-time, thus tracking additional temporal data not present in offline input.

In this paper, we extend the work[12,13] and introduce an online handwriting recognition system for Arabic script, which is used in various languages, such as Arabic, Farsi, Urdu, Pashto and Kurdish. Our approach performs the recognition on the continuous word-part level and the training on the letter level. Such a scheme avoids the segmentation of words into individual letters during the recognition process, which is often prone to errors, and substitutes the training for large set (the word-parts) by a small set (the letters). Figure 1 depicts the flow of our recognizer. Our approach accurately handles delayed strokes by first detecting them and then integrating them into the word-part body. The current implementation is based on Hidden Markov Models (HMM) and deals with many of the Arabic script recognition difficulties. We focus on word-level recognition of undiacritized (unvocalized) Arabic, and thus no sentence-level context is modeled. Arabic vocalic diacritics are most often ignored in writing and printing and, therefore, not addressed here. In this work, we treat the *shadda* (ω) as diacritic and thus it is not addressed in this work.

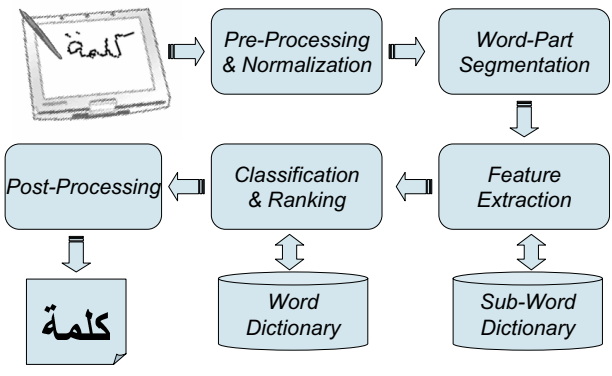


Fig. 1. The main stages of our online Arabic handwriting recognizer.

In the rest of the paper, we first explain the basic characteristics of the Arabic script followed by an overview of related work in handwriting recognition. Then, we discuss preprocessing and feature extraction, the recognition framework, and evaluation results. Finally, we draw some conclusions and suggest directions for future work.

2. Characteristics of the Arabic Script

Arabic script consists of 28 basic letters, 12 additional special letters, and 8 diacritics.[a] Arabic is written (machine printed and handwritten) in a cursive style from right to left. Most letters are written in four different letter shapes depending on their position in a word, e.g. the letter ع (Ain)[b] appears as ع (isolated), عـ (initial), ـعـ (medial), and ـع (final). Among the basic letters, six are disconnective — ا (Alef), د (Dal), ذ (Thal), ر (Reh), ز (Zain) and و (Waw). Disconnective letters do not connect to the following letter and have only two letter shapes each. The presence of these letters interrupts the continuity of the graphic form of a word. We denote connected letters in a word, as a *word-part*. If a word-part is composed of only one letter, this letter is in its isolated shape. For example, the Arabic word مرتفعات (mrtfEAt) "heights" consists of 7 letters (from right to left): م (Meem), ر (Reh), ت (Teh), ف (Feh), ع (Yeh), ا (Alef), and ت (Teh), which are realized initially مـ, finally ـر, initially رتـ, medially ـفـ, medially ـعـ, finally ـا, and isolated ت, respectively. This word has three word-parts (from right to left): مر, تفعا, and ت.

Arabic script is similar to Roman script in that it uses spaces and punctuation marks to separate words. However, certain characteristics relating to the obligatory dots and strokes of the Arabic script distinguish it from Roman script, making the recognition of words in Arabic script more difficult than in Roman script. First, most Arabic letters contain dots in addition to the letter body, such as ش (Sheen) which consists of س (Seen) letter body and three dots above it. In addition to dots, there are strokes that can attach to a letter body creating new letters such as ك, ط and لا. These dots and strokes are called *delayed strokes* since they are usually drawn last in a handwritten word-part/word. Second, eliminating, adding, or moving a dot or stroke could produce a completely different letter and, as a result, produce a word other than the one that was intended (see Table 1). Third, the number of possible variations of delayed stokes is greater than those in Roman script, as shown in Fig. 2. There are only three such strokes used for English: the cross in the letter t, the slash in x, and the dots in i and j.

Finally, in Arabic script a top-down writing style called *vertical ligatures* is very common — letters in a word may be written above their consequent letters. In this style, the position of letters cannot be predefined relative to the baseline of the word. This further complicates the recognition task, particularly in comparison with the

[a] The diacritics are not explored here, since they are almost never used in handwriting.

[b] All Arabic letters are transliterated in Buckwalter's Arabic transliteration format, without diacritics (refer to www.ldc.upenn.edu/myl/morph/buckwalter.html)

Table 1. Word (a_1) (EzAm) "lion" is a result of moving the dot to the left from word (a_2) (grAm) "love". Word (b_1) (Erb) "Arab" is a result of eliminating the dot above the first letter from word (b_2) (grb) "west". Additional case happens with the letter ش which can be confused with a set of two or three letters when the dots are misplaced. In (c_1) and (c_2) we can see a close example which is much more confusing with handwriting.

	a	b	c
1	عزام	عرب	شكر
2	غرام	غرب	تنكر

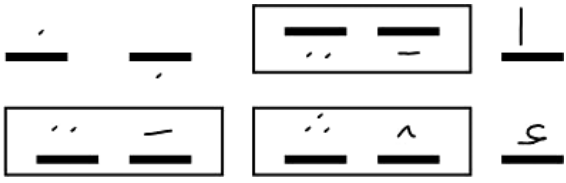


Fig. 2. Delayed strokes in Arabic script may appear under or above the letter body. The boxed pairs represent common variants (e.g. three dots are often written as a circumflex "hat"). These seven strokes appear in letters used in writing standard Arabic. Eleven additional strokes exist for writing additional letters in other languages (Urdu, Pashto, Farsi, etc.).

Roman script. Due to the holistic approach in our proposed recognition model, no restrictions were applied regarding the top-down writing style.

3. Related Work

For the last three decades Hidden Markov Models (HMM) were successfully used for automatic speech recognition. Due to the success of HMM in modeling sequential data it has also been adopted for modeling letters in handwriting recognition. Many variations of HMM models have been adapted and used in script recognition research. Discrete, continuous, and semi-continuous types were used with various topologies ranging from ergodic to left-to-right models with no state skipping. HMM-based algorithms were designed to handle letters, words, strokes or pseudo characters using one-dimensional, two-dimensional or planar Hidden Markov Models. Results were very encouraging in the handwritten case and appear to handle the cursiveness well.

Pechwitz and Märgner[31] presented an offline recognition system for Arabic handwriting using a semi-continuous HMM. They used a sliding window that moves from right to left to collect features directly from the normalized gray image pixels. Then they applied Loeve−Karhunen transformation to reduce the number of features in each frame. They used seven states model for each character shape. Tests were performed using the IFN/ENIT database of handwritten Arabic words and achieved 89% maximal recognition rate. Khorshed[22] used the Hidden Markov Model

Toolkit (HTK) to develop offline printed Arabic text recognition system. After decomposing the document image into text line images, a narrow sliding window was used to extract a set of simple statistical features. The system was applied to a data corpus which includes Arabic text of more than 600 A4-size sheets typewritten in multiple computer generated fonts and achieved 95% maximal recognition rate.

Mahmoud[25] used an HMM based system to recognize offline handwritten Arabic (Indian) numerals. Angle, distance, horizontal, and vertical span features were extracted from these numerals as units for training and testing the HMM. The number of states had been estimated by performing several experiments which show that the best results were achieved using an HMM model with ten states. The system achieved an average recognition rate of 97.99% on a large private database using 120 features presented as 12 observations of 10 features per digit. Benouareth *et al.*[11] presented an offline segmentation-free recognition system for unconstrained Arabic handwritten words using discrete HMM with explicit state duration. The explicit state duration modeling was used to improve the discriminating capacity of the HMM and enable the recognition of difficult patterns in an unconstrained Arabic handwriting. They used a new version of the Viterbi algorithm that takes into account explicit state duration to perform efficient training and testing tasks. A set of statistical and structural features were extracted from the word image using a sliding window approach based on vertical projection histogram. Experiments using the IFN/ENIT database achieved 90.2% average recognition rate.

Al-Hajj *et al.*[2] presented a segmentation-free system for offline recognition of cursive Arabic handwritten words. They used three HMM-based classifiers and the combination of their results is used to determine the recognized words. Sliding windows with different orientations were used to extract pixel-level features, such as pixel density distribution and local pixel configurations from the binary image. They tested different combination schemes and achieved a recognition rate of up to 90.96% on the IFN/ENIT database.

Khorsheed[21] presented a segmentation-free method for offline recognition of cursive handwritten Arabic script. Structural features were extracted from the skeleton of the words after segmentation to elementary strokes. These features are used to train a single hidden Markov Model. The HMM is composed of multiple character models where each model represents one letter from the alphabet. The proposed method achieved recognition rate of 72% and 87% after consulting with a word dictionary on samples extracted from a historical handwritten manuscript.[20] Al-Muhtaseb *et al.*[3] proposed a HMM-based system for offline recognition of Arabic printed texts. Sixteen features were generated from each vertical sliding strip from overlapping and nonoverlapping hierarchical windows. Eight different fonts were used for training and testing, yielding recognition rates around 99%. Dehghan *et al.*[14] presented a segmentation-free approach for offline handwritten Farsi/Arabic words recognition. A discrete HMM was used for the recognition process and

Kohonen self-organizing maps for vector quantization of the feature vectors. A sliding window on the histogram of the chain code directions was used to generate the feature vectors. The width of the sliding window was fixed to twice the stroke width and divided to five horizontal zones. Experiments carried out on test samples of 17,000 of 198 city names in Iran achieved 65.05% recognition rate.

Menasri *et al.*[27] presented a hybrid system based on HMM and neural network classification methods using explicit grapheme segmentation. Each letter-body class was represented by an HMM model and the neural network computes the observations probability distribution. Experiments using IFN/ENIT database result in 87% average recognition rate. Dots and diacritics were recognized independently and used as prior knowledge to eliminate and validate letters.

Some papers focused on the recognition of isolated forms of Arabic letters or digits only.[5,8–10,16,28,29] Recently, many attempts have developed algorithms to recognize the cursive form of the Arabic script. HMM-based systems received most of the attention, but other techniques were also used and proved to have satisfying results. Al-Emami and Usher[1] developed an online Arabic handwriting recognition system, based on decision-tree techniques. Their system was tested on 13 Arabic letter shapes. Alimi[6] developed an online writer-dependent system to recognize Arabic cursive words using a neuro-fuzzy approach. The system was tested using one writer on 100 replications of a single word. Al-Taani[4] used a structural approach to develop an online Arabic digit recognizer. Primitives representing specific strings were extracted from each digit. Then, grammars, constructing these strings, were used to identify digits. The system was tested using 100 different writers, and an average recognition rate of 95% was reported. Mezghani *et al.*[28] developed an online recognition system for isolated Arabic letters using Fourier descriptors and Kohonen maps. They reported a recognition rate of 86% on 7244 samples of 17 classes written by 17 writers. Fourier descriptors and tangents, extracted along the boundary, were used to represent the characters. Alimi and Ghorbel[5] developed an online recognition system for isolated Arabic characters using dynamic programming algorithms. They reported a recognition rate of 93% using different database sizes and replication of characters.

AraPen is an Arabic online handwriting recognition system, which was developed by Alsalakh and Safadi.[7] The system, which is based on Dynamic Time Warping (DTW), was designed to handle noncursive character recognition and adapted to the cursive case. In the noncursive case, the system was tested on a small corpus and achieved a recognition rate of 91%, after training with a specific writer's style. The recognition rates went down dramatically, to lower than 50%, when adapting the system to cursive scripts. Baghshah *et al.*[9] developed a system to recognize isolated Persian handwritten letters online. The system is based on a fuzzy logic approach and yields a recognition rate of 95% using the Razavi and Kabir database.[33] Halavati *et al.*[17] used visual features and fuzzy logic classifiers to develop a system for online recognition of Persian handwriting.

In general, previous work has viewed delayed strokes as features that add complexity to online handwriting recognition. Four methods were proposed to recognize words with delayed strokes:

- Delayed strokes were totally discarded from handwriting in the preprocessing phase.[6]
- Delayed strokes were detected in the preprocessing phase and then used in a post-processing phase.[19]
- The end of a word was connected to the delayed strokes with a special connecting stroke.[26] Adding the special stroke, which indicates that the pen was raised, results in a continuous-stroke sequence for the entire handwritten English sentence.
- Delayed strokes were treated as special characters in the alphabet,[19] i.e. a word with delayed strokes was given alternative spellings to accommodate different sequences where delayed strokes are drawn in different orders.

These four methods are not adequate for recognizing Arabic script. The first and second methods cannot be employed effectively since the number and location of dots define the letter itself. Eliminating delayed strokes causes tremendous ambiguity, particularly when the letter body is not written clearly. Furthermore, eliminating delayed strokes may lead to a similar shape that may represent different letters or a sequence of letters. For example, the letter (Seen) سـ has a shape similar to that of the three letters بتيـ $(b + t + y)$ (without dots) in some writing styles. The third and fourth methods also cannot be implemented, since Arabic words may contain many delayed strokes. These methods dramatically increase the hypothesis space, since words should be represented in all their handwriting permutations. For example, the word حقيقية (Hqyqyp) "truth" contains ten dots, six are above the word and four under it. Connecting the delayed strokes with the end of the word complicates the representation of the word and removing the delayed strokes (dots) require handling $3 \times 4 \times 5 \times 4 \times 2 = 480$ different representations.

4. Preprocessing and Feature Extraction

In this section, we describe our geometric preprocessing, feature extraction, and our novel solution for delayed-strokes.

4.1. *Geometric preprocessing*

Acquired point sequences pass a geometric processing phase to minimize handwriting variations. We have used a low-pass filter algorithm[35] to reduce noise and remove imperfections caused by acquisition devices. To simplify the point sequences, in order to eliminate redundant points irrelevant for pattern classification, we applied the Douglas and Peucker algorithm.[15] To complete the preprocessing, we performed writing-speed normalization by resampling the point sequences.

4.2. *Feature extraction*

We extract three main features for each point in the processed point sequence: *local_angle*, *super-segment*, and *loop-presence*. The *local-angle* feature of p_i, which is denoted by *local-angle*$_i$, is defined as the angle between the segment $\overline{p_{i-1}, p_i}$ and the x-axis ($i > 1$), as shown in Fig. 3. The *super-segment* feature provides wider geometric information which relates each segment to its segment group. This feature is computed by further simplifying the processed point sequence to obtain a coarse representation, which are denoted the *skeleton points*. Every two consecutive skeleton points define a skeleton segment. The super-segment feature for point p_i, which temporally appears between two consecutive skeleton points, is defined as the angle between the skeleton segment and the x-axis (see Fig. 3). This feature is denoted by *super-seg-angle*$_i$. The *loop feature* is a global feature that indicates the presence of a loop in the processed point sequence. Global features capture information related to the global geometric shape of the whole word/letter. Three common global features have been used in previous work for handwriting recognition: loops, cusps, and crossings.[18] In this work, only the loop feature is used, since loops are obligatory in many Arabic letter shapes, e.g. (f) ف. In contrast, cusps and crossings are less common and vary among writers. Global features are not robust features by themselves for unconstrained script. However, the loop feature has greatly improved our recognition rate. We refer to this feature for point p_i as *is-loop*$_i$, which is 1 if p_i is in a loop, otherwise 0. Different people write the initial form of the Arabic Letters ح , خ and ج while performing a loop while others write them similar to the printed form with no loops. In this work, we restrict the writing style to the case with no performing loops. Extending this work to include such cases can be done by treating the different shapes as two different letters (with and without loop) representing the same letter for each one of these three letters.

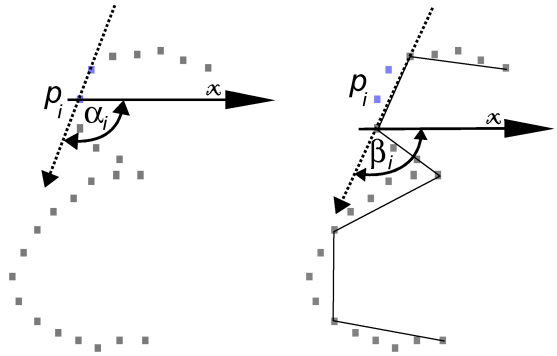


Fig. 3. The *local-angle*$_i$ and *super-segment*$_i$ features, denoted by α_i and β_i, respectively.

4.3. *Handling delayed strokes*

Delayed strokes are essential to distinguish between various Arabic letters. Thus, handling them correctly is vital for accurate recognition of the Arabic script. We have developed the *delayed-stroke projection* algorithm to integrate a delayed stroke within the appropriate word-part body. Our algorithm involves two steps, the detection of delayed strokes and the incorporation of delayed strokes within the right word-part body of the processed point sequence.

In Arabic scripts, delayed strokes are written above or below a word-part and could appear before, after, or within a word-part with respect to the horizontal axis, as shown in Fig. 2. Usually, delayed strokes are written immediately after completing the word-part body. This creates the general interleaved sequence wp_1, $ds_1, wp_2, ds_2, \ldots, wp_n, ds_n$ where wp_i is ith word-part and ds_i is the ith delayed-stroke set associated with wp_i. The delayed-stroke set can be empty for word-parts without delayed strokes. To detect the delayed strokes associated with a word-part, it is enough to determine whether a given processed point sequence forms a delayed stroke or not.

The detection of delayed strokes — dots and short-strokes — is performed based on their sequential order, location, and size. Dots are detected based on the size and shape of their bounding box with respect to the word-part. They usually tend to have nearly square bounding boxes. Valid non-dot delayed strokes are required to either fall within the horizontal boundary of the word-part or to appear before (on the right side of) the word-part. This restriction allows consecutive word-part bodies to overlap, as shown in Figs. 4(a) and 4(b), e.g. in (a) word-parts 1 and 2 overlap.

The trainers and evaluators were required to respect the rule that a word-part body should be written in a single continuous stroke followed by a number of delayed strokes. Upon the detection of a delayed stroke and distinguishing it from the word-part body, we perform the *delayed-stroke projection*, which is illustrated in Fig. 5 (with one letter). Our delayed-stroke projection algorithm starts by vertically projecting the first point of the delayed stroke q_1 into the letter body at point p_i. Then incorporating the delayed stroke into the letter body by inserting the delayed-stroke's point sequence, d_{ps}, into the letter-body's point sequence, l_{ps}, starting from p_i. Finally, the last point of the delayed stroke is connected to point p_{i+1}. The two newly added virtual segments that connect the delayed stroke with the letter body

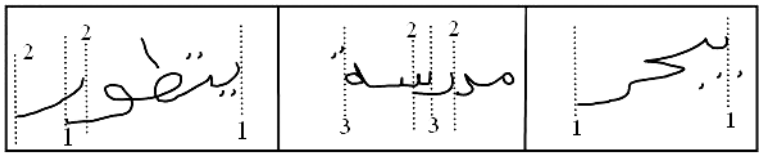


Fig. 4. Possible delayed stroke positions used for detection: (a) five delayed strokes for word-part 1; (b) two delayed strokes for word-part 3; (c) three delayed strokes for word-part 1.

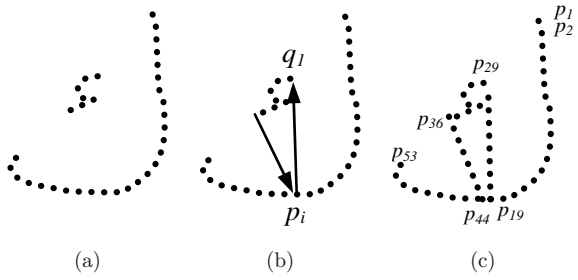


Fig. 5. (a) The projection of the delayed stroke ك in the letter ك (k); (b) the delayed stroke is projected to the letter body; (c) the newly generated PPS (p1 to p53).

are uniformly sampled (according to the average sampling rate of the word-part). We denote the points on the virtual segment as the *virtual points*.

Arabic letters usually appear within connected word-parts and not as isolated letters. Delayed-stroke projection is used to integrate a delayed stroke within a word-part body, as in the isolated-letter case (see Fig. 6). In the cases where a delayed stroke appears before or after the word-part body, as shown in Figs. 4(b) and 4(c), we connect the delayed stroke to the closest point of the word-part body.

4.4. *Feature-vector construction*

Due to the nature of our feature space, we have adopted the discrete Hidden Markov Model[c] (HMM) (for a tutorial in Hidden Markov Model, please refer to Ref. 32) for the recognition task. The input to this model is a sequence of discrete values, which are usually denoted as the observation sequence. Thus, a quantization process is required to convert the three-dimensional feature-vector sequence, extracted from a handwritten word-part, to a discrete observation sequence. In our current implementation, each observation o_i in an observation sequence is an integer value $[0 \cdots 259]$(which are represented using 9 *bits*). This sharp discretization is necessary to reduce the training samples for online Arabic handwriting systems. The lowest 8 *bits* are used to represent the 3D-feature vector — *local-angle*$_i$, *super-seg-angle*$_i$, and *is-loop*$_i$. The *local-angle*$_i$ and *super-seg-angle*$_i$, which are real angle values, are converted to 16 and 8 directions, respectively (similar to Ref. 23); and the feature *is-loop*$_i$ is a binary value (one bit). The *ninth bit* is used to mark virtual points and when it is

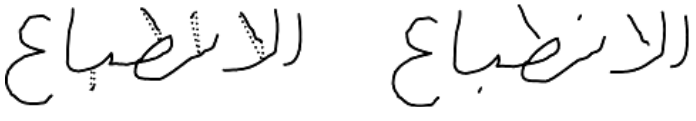


Fig. 6. Three delayed strokes are projected on the second and third word-part bodies for the handwritten word: الانطباع (AlAnTbAE) "the impression."

[c]Using a multivariant mixture of Guassians (as emission probability density function) does not model properly the binary feature, the *is-loop* in our case.

on, the first two bits are used to describe the property of the corresponding virtual point. The observation values $[256 \cdots 259]$ (which correspond to the first two bits $00 \cdots 11$, respectively) are used to classify the virtual points using (a) the position of the delayed stroke (above or below the word-part), and (b) the direction of the virtual segment (up or down). These four observation values are crucial to distinguish between different letter shapes that have the same letter body, but differ on the position of their delayed strokes, e.g. ت (Teh) and ي (Yeh).

5. Recognition Framework

Our recognition framework uses discrete HMMs to represent each letter shape. To enhance word recognition, these letter-shape models are embedded in a network that represents a word-part dictionary. The recognition of word-parts is performed without explicit segmentation into letter shapes, and instead, the recognition is performed along paths that represent valid word-parts, similar to[18,26,30]. Our approach greatly utilizes the fact that Arabic words are composed of word-parts to improve the efficiency of the recognition framework. The next four sections describe, in more detail, the word-part dictionary, the letter-shape models, the word-part network, and the word recognizer.

5.1. *Word and word-part dictionaries*

To limit the search space, we utilize a dictionary of possible valid words. This ensures better recognition rates compared to systems that can recognize any arbitrary permutation of letters. The Arabic dictionary D is subdivided into a set of subdictionaries $D_1, D_2, \ldots, D_n$ based on the number of word-parts in each word. Subdictionary D_k includes all words that consist of k word-parts. For example, if a given dictionary D includes the words {انسان , التحدي , ثقافة , جامعة , رواية , فادي , محمد , محمود , معلم , هل , وسام }, D is divided into four subdictionaries as shown in Fig. 7. We refer to the word-part dictionary $WPD_{k,i}$ as the list of word-parts at the ith entry (starting from right) of the words in D_k. The word-part dictionaries for D_3 are shown in Fig. 8.

5.2. *Letter-shape models*

Each Arabic letter has one, two, three or four shapes that vary according to its position in the word. We have chosen to treat these letter shapes independently

$$
\begin{aligned}
D_1 &= \{\text{هل , معلم , محمد}\} \\
D_2 &= \{\text{محمود , جامعة , ثقافة}\} \\
D_3 &= \{\text{وسام, فادي , التحدي , انسان}\} \\
D_4 &= \{\text{رواية}\}
\end{aligned}
$$

Fig. 7. The subdivision of a dictionary into sub-dictionaries.

$$
\begin{aligned}
WPD_{3,1} &= \{\text{و , فا , ا}\} \\
WPD_{3,2} &= \{\text{سا, د ,لتحد ,نسا}\} \\
WPD_{3,3} &= \{\text{م, ي ,ن}\}
\end{aligned}
$$

Fig. 8. The word-part dictionaries for D_3 (from Fig. 7).

(i.e. as unique characters). For example, to each shape of the letter ه (Heh), we associate four letter-shape models ه , ھ , ھ , and ه corresponding to its isolated, initial, medial and final shape, respectively. The discrete left-to-right HMM without state skipping has been adopted to model each Arabic letter shape. We selected this basic topology because it has been effectively used in handwriting recognition.[18] Additionally, there is no sufficient evidence that more complicated topologies would necessarily achieve better recognition results.[18]

5.3. *Word-part network*

The letter shapes are embedded in a network that represents the word-part dictionary $WPD_{k,i}$. We optimize this network by grouping all shared suffixes, as shown in Fig. 9. Each node in this network represents a letter shape, and each path from the start node (root) to a leaf corresponds to a unique word-part in $WPD_{k,i}$. Each leaf, l, contains the word-part wp_j, which is represented by the path from the start node to the leaf l. We shall refer to this network as a *word-part network* and denote $WPN_{k,i}$ the word-part network that represents the word-part dictionary $WPD_{k,i}$. $WPN^*_{k,i}$ is $WPN_{k,i}$ where each node is replaced with its corresponding letter-shape model. Null transitions are used to connect consecutive letter-shape models in the network as shown in Fig. 10.

A word-part network can be constructed by either assigning the first or last letters (of word-parts) to the first level of the tree. Since Arabic word-parts always (except the last word-part in a word) end with one of the six disconnective letters, we assign the last letters to the first level in the word-part network. This fact guarantees that at least one letter is shared in each word-part, which reduces the size of WPN.

5.4. *Arabic word recognizer*

In Sec. 4, we discussed the generation of the observation sequences $O_s = [O_1, O_2, \ldots, O_k]$ from a given handwritten Arabic word, where $O_i = [o_{i,1}, o_{i,2}, \ldots, o_{i,T_i}]$

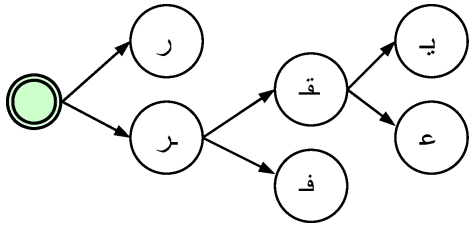


Fig. 9. A word-part dictionary structure, which encodes the word-parts ر, فر, عقر, and يقر.

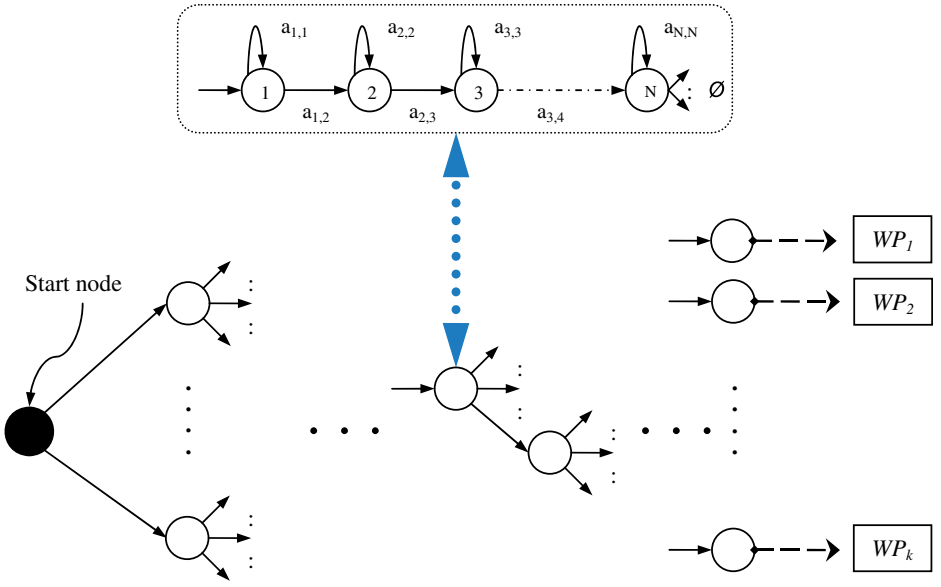


Fig. 10. A word-part network: for $i = 1, \ldots, k$, each path from the start node to a leaf represents a wp_i which is formally defined as $[final + medial^* + initial)|isolated]$.

is the observation sequence constructed from the handwritten word-part wp_i. In this section, we introduce our Arabic word recognizer which is based on our word-part network and uses the *Viterbi* algorithm to determine the recognized word-part.

In a recognition process, we are required to find the word $W = [wp_1, \ldots, wp_k]$, where wp_i is the word-part i, in a given subdictionary D_k that maximizes the posterior probability in Eq. (1). For simplicity, we assume that all word-parts are statistically independent.

$$P(W|O_s) = \prod^{k} P(wp_i|O_i) \tag{1}$$

where,

$$P(wp_i|O_i) = P(O_i|wp_i)P(wp_i)/P(O_i) \tag{2}$$

We also assume that all word-parts in the subdictionary occur with equal probability. As a result, $P(wp)$ is the same for all word-parts and estimating the probability is reduced to maximize $P(O_i|wp_i)$, which can be computed efficiently using the *Viterbi* algorithm on the given $WPN^*_{k,i}$. The *Viterbi* algorithm computes $\delta_t(S)$ which refers to the highest likelihood along a single path at time t, that accounts for the first t observation and ends in state S.[32] Specifically, we are only interested in the accumulated maximum likelihood in leaf states at time T_i $(= |O_i|)$, given $WPN^*_{k,i}$ and O_i (for $1 < i < k$), which is computed using Eq. (3), where q is a leaf state in $WPN^*_{k,i}$, wp is its corresponding word-part, and δ^i_{Ti} is the result of applying the *Viterbi*

algorithm on $WPN^*_{k,i}$. The search for the word W in D_k is performed by applying Eq. (4), where W is the recognized word in text format.

$$P(O_i|wp, WPN^*_{k,j}) = \delta^i_{Ti}(q) \tag{3}$$

$$W = \underset{w=[wp_1,wp_2,\ldots,wp_k]\in D_k}{\operatorname{argmax}} \prod_{i=1}^{k} P(O_i|wp_i, WPN^*_{k,i}) \tag{4}$$

6. Model Training

Training data is created by asking Arabic-literate trainers to handwrite (using a digital tablet) a list of predetermined words. The trainers are also asked to manually specify demarcation points, which are points along the word-part curve and a vertical handler, that separate letter shapes such that all delayed strokes of a letter shape are horizontally aligned between the letter shape's demarcation points, as shown in Fig. 11. As the training process progresses, the system tries to guess the demarcation points based on the accumulated training. Then the trainer can update and correct the demarcation position for incorrect guesses. The details of the specific training data used in our evaluation are discussed in Sec. 7.

The words in the training data are split into letter-shape samples. In order to avoid improper samples, each letter-shape sample is tested to determine if it satisfies the predetermined letter-shape well-formedness rules, e.g. number and placement of dots/strokes above or below the letter body. The Baum−Welch training algorithm is used to determine the HMM parameters, $\lambda = (A, B, \pi)$, for each letter-shape model. Before the training process, the initial state distribution $\pi = \{\Pi_i\}$ is initialized to: $\pi_1 = 1$ and $\pi_i = 0$ for $1 < i < N$ (where N is the number of states in the model). The transition probability matrix $A = a_{i,j}$ is initialized to $a_{i,i} = 0.5$, and $a_{i,i+1} = 0.5$ for

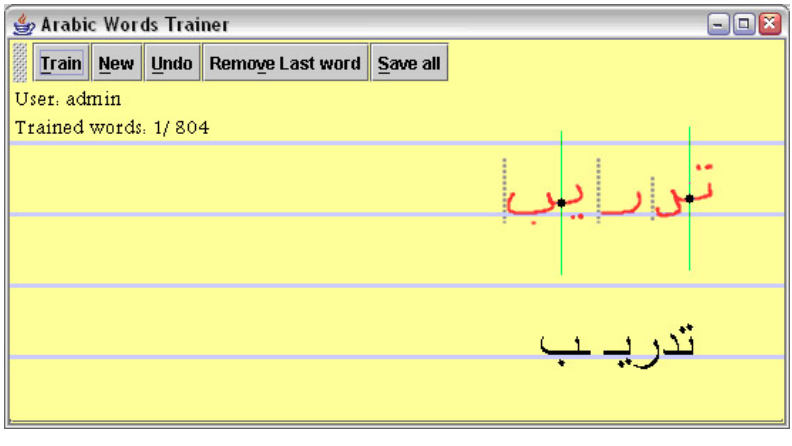


Fig. 11. The trainer is required to manually specify demarcation points that separate letter shapes for letter level training.

$i < N$ and $a_{i,j} = 0$ (where, $i \neq j$ and $i \neq j+1$ for $j < N$, and $a_{N,N} = 1$). The observation matrix B is initialized to reflect a uniform distribution. We have empirically chosen the number of states for each letter-shape model based on the geometric complexity of the letter shape. In our system, the number of states varies from 5 to 11. For example, we assigned 11 states for the isolated letter shape ش (Sheen); and five states to the isolated letter shape ا (Alef).

7. Optimization

Time and space complexity play major roles in application efficiency. In interactive applications, such as online script recognition, the system response time is obviously very crucial. Nevertheless, high recognition rates are the most important aspect of these systems.

Segmenting a Latin word into individual letters is an easy task for the non-cursive handwriting and challenging one for cursive writing. In Arabic scripts, such segmentation is often difficult for printed and handwritten words. Several reasons make the segmentation of Arabic words more complicated than cursive Latin words. In Latin cursive writing, the letters are similar to their isolated equivalence and they are usually connected using additional ligatures, which are often not part of the letter body. In addition, Arabic script is not restricted to writing horizontally (along a base line) and allow some letters to appear in vertical orders.

An alternative holistic approach, is the segmentation free scheme. However, these approaches usually require huge databases to store the basic models and high time complexity to search for the right candidates. Each connected component (word-part) is treated as one component to be classified. As a result, a distinct different model is constructed, trained, and saved for each word-part. The number of possible word-parts determines the complexity of the system. Fortunately, in the Arabic language this number is not as huge as one would think. We have processed a collection of Arabic words and found that the number of unique word-parts in Arabic language is around 47,000 (Table 2) and the majority of these word-parts include four and five letters. These properties are used for further optimizations that include fixing observation-sequence length, and purifying the statistical post-processing phase. Most words in Arabic, more than 90% (as seen in Tables 3 and 4), have

Table 2. The number of valid Arabic word-parts as a function of their size (the number of letters). Note that the one letter length word-parts include few additional letters, such as أ , ة .

#Letters	#Word-parts	#Letters	#Word-parts
1	31	6	5532
2	653	7	1253
3	7273	8	198
4	18540	9	19
5	14236	10	3

Table 3. The number of dots (above/below) and loops define the *DotLoop* property for each Arabic letter. These properties may change as letters change location — beginning, middle, and end — in a word.

DotLoop Property	List of Letters
0 loops	أ , ي, ن, ل, ك, غ, ع, ش, س, ز, ر, ذ, د, خ, ح, ج, ث, ت, ب, ا
1 loop	و, ه, م, ق, ف, غـ, عـ, ظ, ط, ض, ص
0 dots	و, ه, م, ل, ك, ع, ط, ص, س, ر, د, ح, ا
1 dot above	ن, ف, غ, ظ, ض, ز, ذ, خ
2 dots above	ة, ق, ت
3 dots above	ش, ث
1 dot below	ب, ج
2 dots below	ي
optional loops	ه, هـ, ھ, جـ, حـ, خـ, ج, ح, خ

additional strokes and/or loops. Since these features are determined in the feature extraction step, they are utilized to accelerate the classification process by reducing the search space. Based on the results in Table 4, an optimization step is performed as a preprocessing step to reduce the number of models to be tested using the number, position, and order of the additional strokes. As shown in Table 4, determining the additional strokes of a written component representing a word-part reduces the size of a class of candidates to less than 500 on average which accelerates the system responses, see Table 8.

In our approach, we perform training on the letter level. The models of these letter are combined into a word-part dictionary network, which also represents the models of the word-parts. This network is used to assist and verify word-part recognition and guides the combination of recognized word-parts into words. Such an approach

Table 4. This table includes four matrices that correspond to $0, 1, 2$ and 3 loops. In each matrix, the cell in column i and row j include the number of word-parts that have i dots above it and j dots below it.

Down/Up	0	1	2	3	4	0	1	2	3	4
			No loops					One loops		
0	589	746	761	552	242	1154	973	1054	472	195
1	600	623	748	408	192	1520	851	1130	379	192
2	696	597	642	390	151	1693	854	1158	337	191
3	764	546	614	320	128	1563	573	908	217	123
4	499	298	305	99	42	1062	281	585	116	66
			Two loops					Three loops		
0	813	382	449	119	43	213	53	68	8	6
1	1190	375	673	115	64	361	57	112	9	6
2	1424	425	783	118	84	535	80	204	16	19
3	1078	194	588	59	50	319	34	139	10	12
4	801	117	423	52	37	224	33	87	6	7

avoids training for all valid word-parts in the language, but manages to recognize word-parts and words at high rates even though most of them were not part of the samples training the system.

Table 4 shows that the word-part dictionary is divided into several disjoint classes based on the number of loops and dots above or below a word-part. Each class has less than 2000 words and the average number is less than 500 words.

To recognize a given word w, we can use an improved algorithm that utilizes the properties of Arabic script.

```
For each w in Text
 For each word-part (wp) in w
 above_dots = CountUpperDots(wp)
 below_dots = CountLowerUpperDots(wp)
 loops = CountLoopsDots(wp)
 ReducedDict = ReduceDictionary(above_dots, below_dots, loops)
 Classify(wp, ReducedDict)
```

To reduce the dictionary search space, we index its words using the number of loops and dots above and below each word. Such indexing reduces the search space to be less than 500 word-parts in average, instead of the entire word-part dictionary. In cases where the loop feature is not consistent, the average number would increase to be three times when loops feature is ignored which is still affordable.

The calculation presented in this paper used around four million words from different books and websites. It is obvious that this dataset does not include every word in the Arabic languages. Nevertheless, we believe it is enough to describe the general distribution of Arabic language word-parts.

8. Results and Discussion

Several tests and experiments had been carried out to test and validate the different parts of our system. Four classes of experiments were performed to measure the effect and validate the following parameters: (1) the datasets for training and evaluation, (2) the proposed feature set, (3) the HMM based classification method, and (4) the optimization process based on additional strokes and loops.

On the contrary to the English language where databases for online handwriting are publicly available for many years, there is no standard reference dataset for training and/or evaluating online handwriting recognition systems for Arabic script. We were not able to access any Arabic handwritten corpus to be used to properly evaluate our approach. Furthermore, it was not possible to attain any online Arabic handwriting recognition system to accurately compare with our results. Therefore, we constructed our own datasets (Manual Database).

Manual Database: Four trainers were guided to write 800 selected words and mark the boundaries of the letter shapes. The words were selected to cover all Arabic letter

shapes with almost uniform distribution. In the evaluation stage, ten writers (the four trainers and an additional six new writers) were asked to write 280 words not in the training dataset. The evaluation set included 2358 words in total.[d] The overlap of trainers participating in the creation of training and evaluation data is intended to help us evaluate writer-dependence, as well as writer-independence. The trainers and evaluators were asked to write in their own writing style, but respect the rule that a word-part body should be written in a single continuous stroke followed by a number of delayed strokes. We evaluated our system using five different dictionary sizes: 5K, 10K, 20K, 30K, and 40K words selected from the Arabic Treebank,[24] twenty random articles from Al-Arabi Magazine, and ten random articles from the website of the news channel Aljazeera. The 280-evaluation words were present in all dictionary sizes. The purpose of the various dictionary sizes is to test our system's performance under different ambiguous conditions.

Synthetic Database: Developing and maintaining large comprehensive databases which include the many different shapes of each word in the dictionary is extremely expensive and difficult. In Ref. 34, we present an efficient system that automatically generates prototypes for each word-part in a given dictionary using multiple appearance of each letter shape. A set of generated prototypes (fonts) have been collected from 14 different writers imitating handwriting styles for each character in each position. The 800 words, written by the four trainers in the Manual Database were separated to characters and added to the generated prototypes. These sets of prototypes were used to generate all shapes for writing each word-part in any given dictionary. Principal component analysis (PCA) followed by k-means clustering are performed to select the minimal number of shapes representing the wide variation of

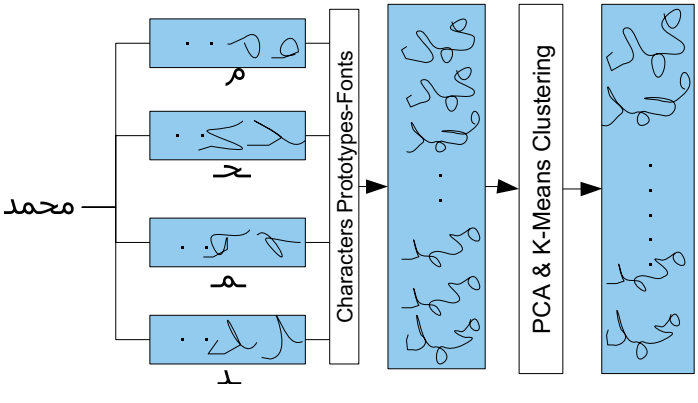


Fig. 12. A diagram flow sample for generating a compact set of shapes for a given word-part. In the first step, we use the collected prototypes, concatenate them to multiple shape of the given word-part and reduce redundancy in the second and third steps respectively.

[d]Not all volunteers finished the testing task, and some word samples were omitted due to being incomplete.

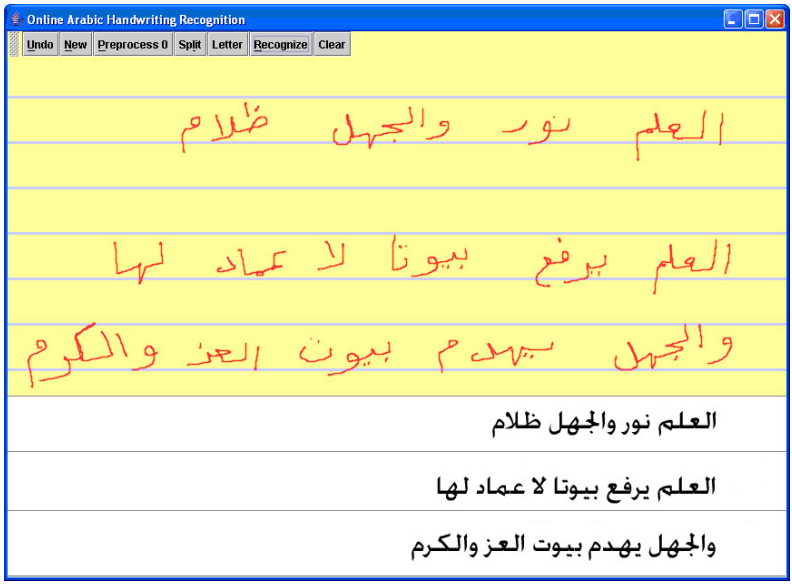


Fig. 13. Several handwritten sentences (top) and their correct recognition (bottom).

handwriting styles for a word-part, see Fig. 12. To validate results using our Manual Database, we have used the methods presented in Ref. 34 to synthetically generate the same 280 words we have in the Manual Database. In total, the evaluation set included 3282 multiple shapes of the same 280 words synthetically generated and 1200 shapes written by the four trainers. Even though the prototype sets were generated based on the 18 different writers, the variety of the generated words is richer due to mixing different styles of writers. The trainers and evaluators were asked to respect the rule that a word-part body should be written in a single continuous stroke followed by a number of delayed strokes. As in the Manual Database, we have evaluated our system using five different dictionary sizes: 5K, 10K, 20K, 30K, and 40K words selected from the Arabic Treebank.[24]

Table 5. Writer-dependent (WD) and writer-independent (WI) average word recognition rates for two tests including 2358 of the Manual Database and 4482 words from the Synthetic Database. Results using the synthetic database to extract the same words as in the manual database are in lines three and four.

Database Size	5K	10K	20K	30K	40K
Manual Database					
WD	98.44%	97.94%	96.86%	95.90%	95.44%
WI	98.49%	97.78%	96.54%	95.12%	94.44%
Synthetic Database					
WD	92.14%	91.15%	89.61%	88.33%	85.17%
WI	91.44%	91.01%	89.64%	88.22%	88.11%

Table 6. Writer-dependent (WD) and writer-independent (WI) average word recognition rates using our geometric features and pixel based features from the sliding window technique.

Database Size	5K	10K	20K	30K	40K
Geometric Features					
WD	98.44%	97.94%	96.86%	95.90%	95.44%
WI	98.49%	97.78%	96.54%	95.12%	94.44%
Sliding Win Features					
WD	91.21%	91.15%	90.61%	88.63%	88.21%
WI	92.11%	91.31%	90.22%	90.11%	86.78%

Table 5 shows the recognition rates of our system using our Manual Database and the Synthetic Database for training and testing. To evaluate the Writer Dependent (WD) case, we have used 1200 valid shapes written by the four trainers. In both cases (Synthetic and Manual) the same 1200 shapes were used for (WD) evaluation, even though the dependency in the synthetic case is implicitly embedded by the prototypes generation process.

To evaluate the effectiveness of the presented set of features we carried out several experiments, while keeping the same system and training data sets but replacing the geometric features set by a pixel based features extracted from consecutive rectangular windows shifted from right to left. Pechwitz and Märgner[31] used the sliding-window pixel based features to recognize written Arabic words using the IFN/ENIT benchmark database. To adjust the online strokes to the offline case we thicken the one pixel width stroke to three pixels width before applying the sliding window technique. We extract the features from the binary image instead of the gray one. The feature extraction is directly based on an image representation of the script using pixel values as basic features. A rectangular window with three columns width is shifted from right to left across the generated script image and generates a feature vector which is the concatenation of the three columns. In order to reduce the number of features a Loeve−Karhunen Transformation is performed on the pixel values of each feature vector, see Ref. 31 for more details. Using the Manual Database, recognition rates in Table 6 shows that the system based on geometric features outperforms the pixel based one.

We designed a system based on Dynamic Time Warping (DTW) technique for the matching and classification process. This system uses the same feature set extracted from the word-parts to build the collection of prototypes for matching. The words from the training sets were used to build the sets of prototypes and datasets of feature vectors were saved for each word instead of the HMM model. Table 7 shows the results of the two classifiers.

High recognition rates are the most important aspect in our work, even though the system response time is obviously very important. In these experiments we intend to show the applicability of the suggested optimization to improve recognition rates and reduce the time response.

Table 7. Results of our system compared to a system with the same database and feature sets but using a different classifier based on DTW matching technique.

Database Size	5K	10K	20K	30K	40K
HMM classifier					
WD	98.44%	97.94%	96.86%	95.90%	95.44%
WI	98.49%	97.78%	96.54%	95.12%	94.44%
DTW classifier					
WD	91.24%	90.21%	90.12%	89.23%	88.27%
WI	96.18%	96.11%	93.32%	90.18%	87.22%

Table 8. The improvement in response time and recognition rates that results from using the dots and loops for optimization.

Dictionary Size	5K	20K	40K
Writer-independent			
Time response reduction	−62.11%	−75.31%	−78.45%
Improvement recognition	+0.91%	+1.63%	2.87%
Writer-dependent			
Time response reduction	−64.33%	−76.18%	−78.98%
Improvement recognition	+1.12%	+2.32%	3.14%

Reducing time response was not the main task in this experimental test. Therefore, results are presented as improvement percentage to validate the effectiveness of the optimization independent of the time efficiency of the system. Table 8 shows that using the number of dots above and under their corresponding word-part body,

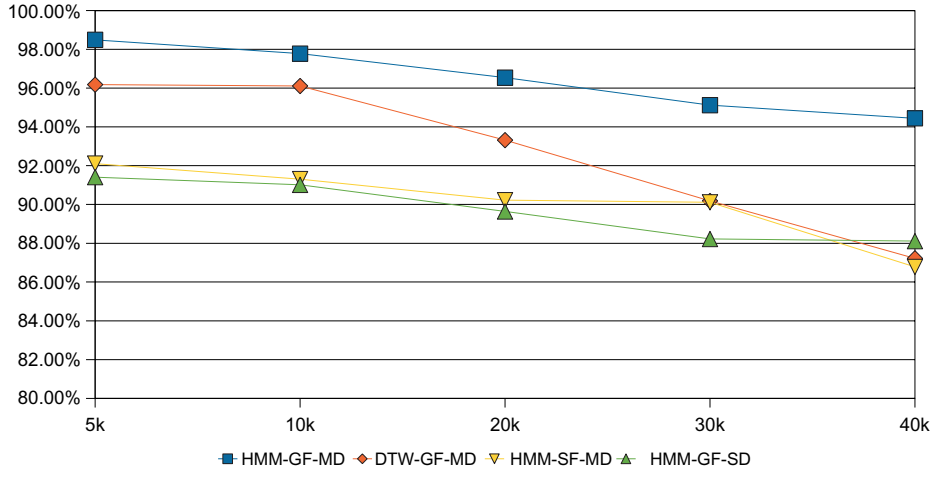


Fig. 14. This graph shows the results of recognition rates comparing the different systems. The compared systems are our proposed system and other three systems change one factor each time. The factors we changed are: SD = Synthetic Database, SF = Sliding Window features, and DTW = Dynamic Time Warping classifier.

combined with the number of loops to filter candidates list in a preprocessing step can reduce the search space and improve recognition rates and response time. It is obvious that this factor becomes more important when the target database is large.

Overall, we achieved good results given that we used a relatively small training set. The differences between the writer-independent and writer-dependent recognition rates are less than 2%, with all tested dictionary sizes. This implies that the features, model and delayed-stroke algorithm, we introduced, are adequate for writer-independent handwriting recognition. The performance degrades as the dictionary size increases. The degradation in word-part recognition is at a lower rate than word recognition, suggesting that the recognition failure is tied to specific word-parts. Most of the recognition errors arose in word-parts that have similar shapes, such as ب/يا and د/ر. Therefore, the current features are not sufficient for adequately distinguishing between such word-parts.

9. Conclusion and Future Work

This paper introduced an HMM-based system with novel components to provide solutions for most of the inherent difficulties in recognizing Arabic script — letter connectivity, position-dependent letter shaping and delayed strokes. An evaluation of the system shows that the used features and letter models are adequate for writer-independent handwriting recognition at high rates. Our solution for delayed strokes can also be utilized to recognize scripts that include diacritical marks (e.g. French, German, Spanish, etc.).

In the future, we plan to increase the system's robustness to handle cases where delayed strokes are written before the completion of a word-part. We also plan to reduce the number of errors described in Sec. 8, using geometric-computation techniques and a more sophisticated post-processing phase. Moreover, we plan on exploring sentence-level language modeling to improve word recognition.[26]

References

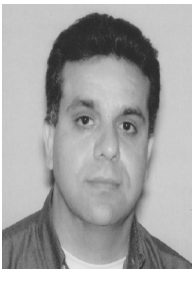
1. S. Al-Emami and M. Usher, On-line recognition of handwritten Arabic characters, *IEEE Trans. Patt. Anal. Mach. Intell.* **12**(7) (1990) 704–710.
2. R. Al-Hajj, C. Mokbel and L. Likforman-Sulem, Combination of HMM-based classifiers for the recognition of Arabic handwritten words, In *ICDAR 2007. Ninth Int. Conf. Document Analysis and Recognition*, Vol. 2 (2007), pp. 959–963.
3. H. A. Al-Muhtaseb, S. A. Mahmoud and R. S. Qahwaji, Recognition of off-line printed Arabic text using hidden markov models, *Sign. Process.* **88**(12) (2008) 2902–2912.
4. A. T. AL-Taani, An efficient feature extraction algorithm for the recognition of handwritten Arabic digits, *Int. J. Comput. Intell.* **2**(2) (2005).
5. A. M. Alimi and O. A. Ghorbel, The analysis of error in an on-line recognition system of Arabic handwritten characters, In *ICDAR'95: Proc. Third Int. Conf. Document Analysis and Recognition (Vol. 2)*, (IEEE Computer Society, Washington, DC, USA, 1995), p. 890.

6. A. L. M. Alimi, An evolutionary neuro-fuzzy approach to recognize on-line Arabic handwriting, *Proc. Fourth Int. Conf. Document Analysis and Recognition*, Vol. 1 (1997), pp. 382–386.
7. B. Alsallakh and H. Safadi, Arapen: An Arabic online handwriting recognition system, *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, Vol. 1 (2006), pp. 1844–1849.
8. A. Amin, Machine recognition of handwritten Arabic word by the irac ii system, *Proc. 7th Joint on Pattern Recognition* (1982), pp. 35–37.
9. M. S. Baghshah, S. B. Shouraki and S. Kasaei, A novel fuzzy approach to recognition of online Persian handwriting, In *ISDA'05: Proc. 5th Int. Conf. Intelligent Systems Design and Applications* (IEEE Computer Society, Washington, DC, USA, 2005), pp. 268–273.
10. H. Beigi, K. Nathan, G. Clary and J. Subrahmonia, Size normalization in online unconstrained handwriting recognition, *The IEEE Int. Conf. Image Processing*, Vol. I, (November 13–16, 1994), pp. 169–172.
11. A. Benouareth, A. Ennaji and M. Sellami, Arabic handwritten word recognition using HMMs with explicit state duration, *EURASIP J. Adv. Sign. Process.* **13** (2008).
12. F. Biadsy, Online Arabic handwriting recognition, M.Sc. Thesis, Ben Gurion University of the Negev, 2005.
13. F. Biadsy, J. El-Sana and N. Habash, Online Arabic handwriting recognition using hidden markov models, *IWFHR'10 2006*, France, (2006).
14. M. Dehghana, K. Faeza, M. Ahmadi and M. Shridhar, Handwritten Farsi (Arabic) word recognition: A holistic approach using discrete HMM, *Patt. Recogn.* **34**(5) (2001) 1057–1065.
15. D. Douglas and T. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *The Canadian Cartographer* **10**(2) (1973) 112–122.
16. T. S. El-Sheikh and S. G. El-Taweel, Real-time Arabic handwritten character recognition, *Patt. Recogn.* **23**(12) (1990) 1323–1332.
17. R. Halavati, M. Jamzad and M. Soleymani, A novel approach to Persian online hand writing recognition, *Trans. Engin. Technol.* **6**(1305–5313.) (2005).
18. J. Hu, S. G. Lim and M. K. Brown, Writer independent on-line handwriting recognition using an HMM approach, *Patt. Recogn.* **33**(1) (2000) 133–147.
19. J. Hu, S. C. Oh, J. H. Kim and Y. B. Kwon, Unconstrained handwritten word recognition with interconnected hidden Markov models, In *Proc. Third Int. Workshop on Frontiers in Handwriting Recognition* (1993), pp. 455–560.
20. M. S. Khorsheed, *Automatic Recognition of Words in Arabic Manuscripts*, Ph.D. thesis, University of Cambridge, 2000.
21. M. S. Khorsheed, Recognising handwritten Arabic manuscripts using a single hidden Markov model, *Patt. Recogn. Lett.* **24**(14) (2003) 2235–2242.
22. M. S. Khorsheed. Offline recognition of omnifont Arabic text using the HMM toolkit (HTK), *Patt. Recogn. Lett.* **28**(12) (2007) 1563–1571.
23. J. J. Lee, J. Kim and H. Kim, Data driven design of HMM topology for on-line handwriting recognition, *The 7th Int. Workshop on Frontiers in Handwriting Recognition*, (World Scientific Publishing Company, 2001), pp. 107–121.
24. M. Maamouri, Developing an Arabic treebank: Methods, guidelines, procedures, and tools, *Proc. Workshop on Computational Approaches to Arabic Script-Based Languages* (*COLING*) (2004).
25. S. I. Mahmoud, Recognition of writer-independent off-line handwritten Arabic (Indian) numerals using hidden Markov models, *Sign. Process.* **88**(4) (2008) 844–857.

26. J. Makhoul, T. Starner, R. Schwartz and G. Chou, On-line cursive handwriting recognition using speech recognition methods, *Proc. IEEE ICASSP'94* (IEEE, 1994), pp. V125−V128.
27. F. Menasri, N. Vincent, M. Cheriet and E. Augustin, Shape-based alphabet for off-line Arabic handwriting recognition, *ICDAR'07: Proc. Ninth Int. Conf. Document Analysis and Recognition* (IEEE Computer Society, Washington, DC, USA, 2007), pp. 969−973.
28. N. Mezghani, M. Cheriet and A. Mitiche, Combination of pruned Kohonen maps for on-line Arabic characters recognition, *ICDAR'03: Proc. Seventh Int. Conf. on Document Analysis and Recognition* (IEEE Computer Society, Washington, DC, USA, 2003), p. 900.
29. N. Mezghani, A. Mitiche and M. Cheriet, On-line recognition of handwritten Arabic characters using a Kohonen neural network, *IWFHR'02: Proc. Eighth Int. Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, (IEEE Computer Society, Washington, DC, USA, 2002), p. 490.
30. S. C. Oh, J. Y. Ha and J. H. Kim, Context-dependent search in interconnected hidden Markov model for unconstrained handwriting recognition, *Patt. Recogn.* **28**(11) (1995) 1693−1704.
31. M. Pechwitz and V. Märgner, HMM based approach for handwritten Arabic word recognition using the IFN/ENIT-database, *ICDAR'03: Proc. Seventh Int. Conf. Document Analysis and Recognition* (IEEE Computer Society, Washington, DC, USA, 2003), p. 890.
32. L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, *Readings in Speech Recognition* (1990), pp. 267−296.
33. S. M. Razavi and E. Kabir, A data base for online Persian handwritten recognition, *6th Conf. Intell. Syst.* (In Farsi, 2004).
34. R. Saabni and J. El-Sana, Efficient generation of comprehensive database for online Arabic script recognition, *ICDAR'09: Proc. 2009 10th Int. Conf. Document Analysis and Recognition* (IEEE Computer Society, Washington, DC, USA, 2009), pp. 1231−1235.
35. L. Schomaker, Using stroke or character-based self-organizing maps in the recognition of on-line, connected cursive script, *Patt. Recogn.* **26**(3) (1993) 443−450.

Fadi Biadsy received his B.Sc. in mathematics and computer science, July 2002, and his M.Sc. Summa cum Laude in computer science, July 2005, both from Ben-Gurion University. He received his Ph.D. in computer science from Columbia University in the City of New York, April 2011.

Dr. Biadsy is a Research Scientist at Google/ Research, NY, working as part of the speech processing team on automatic speech recognition and voice search. His research interests are in spoken language processing, particularly automatic speech recognition, voice search, and automatic classification of speaker characteristics.



Dr. Raid Saabni is a senior researcher at the triangle Research & Development center and on a Post Doc. fellowship at the Department of Electrical Engineering, Tel Aviv University. He received his B.Sc. in mathematics and computer science in 1989 and his M.Sc. and PhD in computer science from Ben-Gurion University in the Negev in 2006 and 2010 respectively.

His research interests are in historical document image analysis, handwriting recognition, image retrieval and image processing.

Dr. Jihad El-Sana is a senior lecturer at the Department of Computer Science, Ben Gurion University of the Negev. He received his B.Sc. and M.Sc. in computer science from BGU. In 1995 he won a Fulbright Scholarship for Israeli Arabs, for doctoral studies in the US. In 1999 he earned a Ph.D. in computer science from the State University of New York, Stony Brook.

El-Sana heads the department's Visual Media Lab, which hosts various research projects in computer graphics, image processing, augmented reality, computational geometry, and document image analysis. El-Sana has published over 50 papers in leading conferences and scientific journals. He is also member of the IAPR, IEEE, and Euro-Graphics societies.