Online Arabic Handwriting Recognition Using Hidden Markov Models

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

Jihad El-Sana
Ben-Gurion University
Department of Computer Science, Beer-Sheva, 8105 Israel
el-sana@cs.bgu.ac.il

Nizar Habash
Columbia University
Center for Computational Learning Systems, New York, NY 10115, USA
habash@cs.columbia.edu

Abstract

Online handwriting recognition of Arabic script is a difficult problem since it is naturally both cursive and unconstrained. 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. This paper introduces a Hidden Markov Model (HMM) based system to provide solutions for most of the difficulties inherent in recognizing Arabic script including: letter connectivity, position-dependent letter shaping, and delayed strokes. This is the first HMM-based solution to online Arabic handwriting recognition. We report successful results for writer-dependent and writer-independent word 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 Tablet PC, 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 based on the presentation of the data to the system: *offline* and *online*. Offline handwriting recognition approaches do not require immediate interaction with the user. A scanned handwritten or printed text is fed to the system in a digital image format. In online handwriting recognition approaches, the user writes using a digital device (such as a digital tablet) utilizing a special stylus. 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 recognition.

In this paper, we introduce an online handwriting recognition system for the Arabic script, which is used by approximately one-seventh of the world's population to write a variety of languages such as Arabic, Farsi, Urdu, Pashto, and Kurdish[1]. We focus on word-level recognition of undiacritized (unvocalized) Arabic. No sentence-level context is modeled. As such, references to language modeling in this paper are over word parts not words. Arabic vocalic diacritics are most often ignored in writing and printing and, therefore, not addressed here.

We first explain basic characteristics of the Arabic script and overview 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[2]. 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 (E)[3] appears as ع (isolated), عـ (initial), ـعـ (medial), and ـع (final). Among the basic letters, six are *disconnectives*, i.e., they do not connect to the following letter: ا (A), د (d), ر (r), ذ (*), ز (z), و (w). Disconnectives have only two letter shapes each.

The presence of these letters causes the continuity of the graphic form of the word to be interrupted. We denote connected letters in a word as a *word part*[4]. If a word part is composed only of one letter, this letter will be in its isolated shape. For example, the Arabic word مرتفعات (mrtfEAt) 'heights' consists of 7 letters (from right to left): م (m) realized initially مـ, ر (r) realized finally ـر,

[1] We focus in this paper on the Arabic script as it is used for writing Modern Standard Arabic only.

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

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

[4] Formally, *wp* is defined: (initial • medial* • final) || isolated.

ت (t) realized initially تـ, ف (f) realized medially فـ, ع (E) realized medially عـ, ا (A) realized finally ا, and ت (t) realized in isolated shape ت. 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 markers 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 ($) which consists of س (s) 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 Figure 1. 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*.

Table 1: Word (a1) (EzAm) 'lion' differs from (b2) (grAm) 'love' in the position of the only dot in the word. Word (a2) (Erb) 'Arab' differs from word (b2) (grb) 'west' in the absence of one dot.

	a	*b*
1	عزام	غرام
2	عرب	غرب

Finally, in Arabic script, a top-down writing style is very common: letters in a word may be written above their consequent letters. In this style, the position of letters can not be predefined relative to the base line of the word. This further complicates the recognition task, particularly in comparison with the Roman script. In our proposed recognition model, no restrictions were applied regarding the top-down writing style.

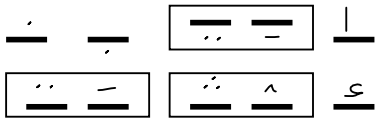

Figure 1: Delayed strokes in Arabic script under or above the letter body. The boxed pairs represent common variants (e.g., three dots are often written as a circumflex '^'). These seven strokes appear in letters used in writing standard Arabic. Eleven additional strokes exist for writing additional letters in other languages (Urdu, Pashto, Farsi, etc.)

3. Previous Work

Most of Arabic handwriting recognition in previous works focused on recognizing offline script [1]. Much of online recognition focused on isolated Arabic letters only [4][6][12]. As far as we could determine, there was little work that tackled the difficulties of online Arabic cursive handwriting recognition. Al-Emami and Usher [2] developed an online Arabic handwriting recognition system based on decision-tree techniques. The system was tested with 13 Arabic-letter shapes. Alimi [3] developed an online writer dependent system to recognize Arabic cursive words based on neuro-fuzzy approach. The system was tested by one writer on 100 replications of a single word.

As for the delayed strokes, previously work viewed them as features that added complexity to online handwriting recognition. Four methods were proposed to recognize words with delayed strokes. In the first method, delayed strokes were totally discarded from handwriting in the preprocessing phase [3]. In the second, delayed strokes were detected in the preprocessing phase and then used in a postprocessing phase [8]. In the third method, the end of a word was connected to the delayed strokes with a special connecting stroke. This special stroke, which indicated that the pen was raised, resulted in a continuous stroke sequence for the entire handwritten English sentence [11]. Finally, delayed strokes were treated as special characters in the alphabet. So, a word with delayed strokes was given alternative spellings to accommodate different sequences where delayed strokes are drawn in different orders [7].

These four methods are not adequate for the task of recognizing Arabic script. The first and second methods could not be employed effectively since the information that makes letters different from others is the number and position where the dots are located. Eliminating delayed strokes will cause a tremendous ambiguity, particularly when the letter body is not written clearly. Furthermore, some Arabic letters have a similar shape of composition with some letters, such as: the letter (s) سـ has a similar shape to the three letter shapes بتيـ (b + t + y) (without dots). The third and fourth methods also cannot be implemented, since Arabic words may contain many delayed strokes. These methods will dramatically increase the hypothesis space, since words should be represented in all of their handwriting permutations. For example: the word حقيقية (Hqyqyp) 'real' contains 10 dots, thus, 10! representations would be required.

4. Preprocessing and Feature Extraction

In this section, we describe our approach in terms of geometric preprocessing, feature extraction, and our novel solution to the delayed-stroke problem.

4.1. Geometric Preprocessing

At this stage, the acquired point sequences pass a geometrical processing phase to minimize handwriting variations. We have used a low-pass filter algorithm [15] to reduce noise and remove imperfections caused by acquisition devices. Then, Douglas and Peucker's algorithm [5] was adopted to simplify the point sequences by using a tolerance t_1 (determined

empirically) in order to eliminate redundant points irrelevant for pattern classification. In the final step, we performed writing-speed normalization by re-sampling the consequent point sequences.

4.2. Feature Extraction

In our current implementation, we extract three features from the point sequence[1] *(PS)*, for each point: *local-angle*, *super-segment,* and *loop-presence*.

The local-angle feature: This local feature is the angle between each vector $(v=p_{i-1}p_i)$ in *PS*, and the *X-axis*, where $i > 1$. The local-angle feature of p_i is denoted by $local\text{-}angle_i$.

The super-segment feature: This novel feature provides wider geometric information which relates each segment to its segment group. The feature is computed by first applying Douglas and Peucker's algorithm with tolerance $t_2 > t_1$, on *PS* to obtain the *skeleton points* (the remaining vertices after applying Douglas and Peucker's algorithm)[2]. Every two consecutive skeleton points define a *skeleton vector*. The super-segment feature, for every point p_i, which temporally appears between the vector's skeleton points, is defined as the angle between the skeleton vector and the *X-axis*. This feature is denoted by $super\text{-}seg\text{-}angle_i$.

The loop feature: This is a global feature that indicates the presence of a loop in *PS*. Global features capture information about the global geometric shape of the whole word/letter. Three common global features were used in previous work in handwriting recognition: *loops*, *cusps* and *crossings* [7]. In this work, only the loop feature is used, since loops are obligatory in many Arabic-letter shapes, e.g., (f) فـ. 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 denote this feature for point p_i as $is\text{-}loop_i$, where $is\text{-}loop_i = 1$ if p_i is in a loop, otherwise *0*.

4.3. Delayed-Stroke Handling

Delayed strokes are essential to distinguishing among various Arabic letters. Thus, handling delayed strokes correctly is vital for appropriate recognition of the Arabic script. We have developed the *delayed-stroke projection* algorithm as a novel method to handle delayed strokes. Our algorithm involves two steps, the detection of delayed strokes and the incorporation of delayed strokes in the word-part body *PS*.

In the Arabic script, delayed strokes are written above or below the word part and could appear before, after, or within the word-part with respect to the horizontal axis as shown in Figure 2. Typically, 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 i^{th} word part and ds_i is the i^{th} delayed stroke set associated with wp_i. The delayed stroke set can be empty for word parts with stroke-less letters. Therefore, to detect delayed strokes associated with a word part, it is enough to determine whether a given *PS* forms a delayed stroke or not. The detection also groups each word-part body with its delayed strokes in a word.

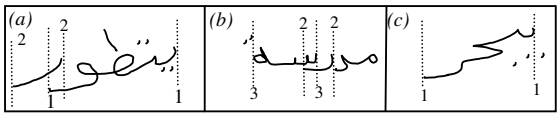

Figure 2: Possible delayed-stroke positions used for the detection mechanism: (a) five delayed strokes for word part 1; (b) two delayed strokes for word part 3; (c) three delayed strokes for word part 1.

The detection of delayed strokes is performed based on the location and size of the strokes, in addition to the time order of the written strokes. Recall that delayed strokes are either dots or short stroke sequences. Dots are detected based on the size and shape of their bounding box with respect to the word part. Dots 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 for overlapping consecutive word-part bodies, as shown in Figure 2 *(a* and *b)* – e.g., in *a*, word-part 1 and 2 overlap.

At this stage, we know which point sequence is a delayed stroke and which is a word-part body. The next step is the projection procedure, which we illustrate in Figure 3 (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. The incorporation of the delayed stroke into the letter body is performed by inserting the delayed stroke *PS* into the letter body *PS* starting from p_i. The last point of the delayed stroke is connected to point p_{i+1}. The two newly added *virtual vectors* that connect the delayed stroke with the letter body are sampled in a uniform manner with a predefined number of points, denoted as *virtual points*. Then, we generate a new *PS* for the letter. The new sequence includes all points starting from the first point of the letter body to point p_i, then to q_1, to the last point of the delayed stroke, to p_{i+1}, and finally to the last point of the letter body.

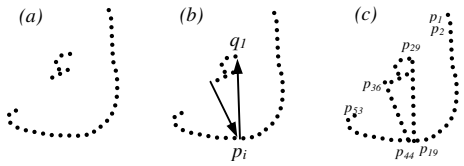

Figure 3: The projection of the delayed-stroke ء in the letter ك (k); *(b)* the delayed stroke is projected to the letter body; *(c)* the new generated *PS* (p_1 to p_{53}).

1 From now on, we use point sequence to denote the preprocessed point sequence.

2 Here, t1 is the tolerance utilized in the preprocessing phase and t2 was determined empirically.

Of course, Arabic letters usually appear as a part of connected word parts and not as isolated letters. We handle this case by projecting the starting point of each delayed stroke into the word-part body and integrating it as in the isolated letter case (see Figure 4). For the cases where the delayed strokes appear before or after the word-part body, as shown in Figure 2 (*b* and *c*), we connect the delayed stroke to the closest point of the word-part body. Our solution for delayed strokes can also be utilized for the task of recognizing scripts that include diacritic markers (e.g., French, German, Spanish, etc.)

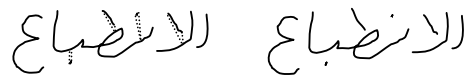

Figure 4: Three delayed strokes are projected in the second and third word-part bodies for the handwritten word: الانطباع (AlAnTbAE) 'the impression'.

4.4. Feature-Vector Construction

Since we use a discrete Hidden Markov Model (HMM) (for more details on HMM see [14]) for the recognition task, the "input" (observation sequence) to this type of model is a sequence of discrete values. Thus, a quantization process is required to convert the 3D feature-vector sequence, extracted from a handwritten word part, to a discrete observation sequence. In our current implementation, each observation o_i in this observation sequence is an integer value [0...259]. The necessity of such sharp discretization stems from the lack of training samples for online Arabic handwriting systems. The values [0…255] are used to represent the 3D-feature vector. The features *local-angle*$_i$ and *super-seg-angle*$_i$ are real angle values, converted to 16 and 8 directions, respectively. This treatment is similar to [9]. The feature *is-loop*$_i$ is binary (one bit). The values [256…259] are utilized to represent the virtual points using (a) the position of the delayed stroke (above or below the word part), and (b) the direction of the virtual vector (up or down). These four observation symbols are crucial to distinguish letter-shapes that have the same letter body but differ on the position of their delayed strokes, e.g., تـ (t) and يـ (y).

5. The 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 segmentation of word parts into letter-shapes and their recognition are performed simultaneously in an integrated process, similar to [7][11][13]. 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 constrain the space of search, 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 *sub-dictionaries* $\{D_1, D_2,..., D_n\}$ based on the number of word parts in each word. Sub-dictionary D_k includes all words that consist of *k* word parts. For example, if a given dictionary *D* includes the words {انسان , التحدي , ثقافة , وسام , هل , معلم , محمود , محمد , فادي , رواية , جامعة}. *D* is divided into the following four sub-dictionaries:

- D_1 = { هل , معلم , محمد }
- D_2 = {محمود , جامعة , ثقافة}
- D_3 = {وسام , فادي , التحدي , انسان}
- D_4 = {رواية}

We refer to the *word-part dictionary* $WPD_{k,i}$ as the list of word parts located in index *i* (starting from right in a word) of the words in D_k. The word-part dictionaries for D_3 presented above are the following:

- $WPD_{3,1}$ = {و , فا , ا}
- $WPD_{3,2}$ = {سا , د , لتحد , نسا}
- $WPD_{3,3}$ = {م , ي , ن}

5.2. Letter-Shape Models

Each Arabic letter has two or four shapes that vary depending on its position in the word. We have chosen to treat these letter shapes independently (i.e., as unique characters). For example, associated with the letter (h) are *four* letter-shape models for ه, هـ, ـهـ, and ـه 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 [7]. Additionally, there is no sufficient evidence that more complicated topologies would necessarily achieve better recognition results [7].

5.3. Word-Part Network

The letter-shape 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 Figure 5. Each node in this network represents a letter shape, and each path from the start node to a leaf corresponds to a unique word part in $WPD_{k,i}$. Each leaf contains the word-part text wp_j representing the path from the start node to this leaf.[1] We shall refer to this network as a *word-part network*. We denote $WPN_{k,i}$ as the word-part network that represents the word-part dictionary $WPD_{k,i}$. $WPN^*_{k,i}$ is $WPN_{k,i}$ with each node

[1] For wp_j, $1 \le j \le k$ (= the size of the word-part dictionary)

replaced with its corresponding letter-shape model. Null transitions are used to connect consecutive letter-shape models in the network.

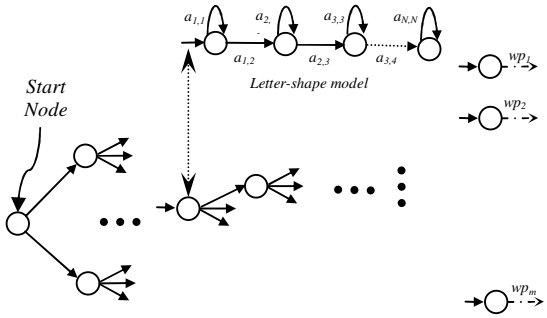

Figure 5: A word-part network – each path from the start node to a leaf represents a *wp* which is formally defined: *(final•medial*•initial) || isolated.*

A word-part network can be built starting either by placing the first or last letters (of word parts) in the first level of the tree. We decided to use the last letters to be in the first level in the word-part network because Arabic words always (except for the last word part in a word) end with one of the six disconnective letters. This fact guarantees that at least one letter is shared in each word part, which leads to a reduction in the size of the *WPN*.

5.4. Arabic-Word Recognizer

In section 4, we computed the observation sequences $Os = [O_1, O_2,\ldots, O_k]$ from a given handwritten Arabic word, where $O_i = [o_{i,1}, o_{i,2},\ldots,o_{i,Ti}]$ is an observation sequence constructed from the handwritten word-part i. In this section, we introduce an Arabic word recognizer using $WPN^*_{k,i}$, for $1 \le i \le k$, and the *Viterbi* algorithm, given Os. The recognition task is to find the word $W = [wp_1, wp_2,\ldots,wp_k]$ (wp_i is word-part i in W) in a given sub-dictionary D_k which maximizes the posterior probability:

$$P(W \mid Os) = \prod_{i=1}^{k} P(wp_i \mid O_i) \quad (1)$$

$$where,\ P(wp_i \mid O_i) = P(O_i \mid wp_i)P(wp_i)/P(O_i) \quad (2)$$

For simplicity, assuming all word parts in the sub-dictionary occur with equal probability and since $P(O_i)$, is the same for all word parts, the problem is reduced to maximize $P(O_i|wp_i)$, which can be computed efficiently by Viterbi algorithm given $WPN^*_{k,i}$. The Viterbi algorithm computes $\delta_t(S)$ which refers to the highest probability along a single path at time t, which accounts for the first t observations and ends in state S [14]. Particularly, we are only interested in the accumulated maximum likelihood in the leaf states at time T_i (= $|O_i|$), given $WPN^*_{k,i}$ and O_i (for $1 \le i \le k$). Therefore,

$$P(O_i \mid wp, WPN^*_{k,i}) = \delta^i_{Ti}(q) \quad (3)$$

where q is a leaf state in $WPN^*_{k,i}$, $q.text = wp$, and δ^i is the result of applying Viterbi algorithm on $WPN^*_{k,i}$ given O_i. Now, we search for the word W in D_k as follows:

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

Here, W is the recognized word text.

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 that separate letter shapes such that all delayed strokes of a letter shape are horizontally between the letter shape's demarcation points. The details of the specific training data used in our evaluation are discussed in section 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 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 for training the HMM parameters, $\lambda = (A, B, \pi)$ for each letter-shape model. The initial state distribution $\pi = \{\pi_i\}$ is initialized to: $\pi_1 = 1$ and $\pi_i = 0$ for $1 < i \le N$ where N is the number of states in the model. The transition probability matrix $A = \{a_{i,j}\}$ is initialized to $a_{i,i} = a_{i,i+1} = 0.5$, for $1 \le i < N$; $a_{i,j} = 0$ where, $i \ne j$ and $j \ne i+1$ for $1 \le i < N$ 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 states. For example: we assigned 11 states for the isolated letter shape ش (\$), and 5 states to the isolated letter shape ا (A).

7. Evaluation

There is no standard reference data set for training and/or testing online handwriting recognition systems for Arabic script. Therefore, we constructed our own sets as follows. For training, four trainers were asked to write 800 selected words each and mark the boundaries of the letter shapes as described in the previous section. The words were selected to cover all Arabic letter shapes with almost uniform distribution. For testing, ten testers (the four trainers, in addition to six new volunteers) were asked to write 280 words not in the training data. The test set included 2,358 words in total[1]. The overlap of trainers participating in the creation of training and test data is intended to help us evaluate writer-dependence in addition to writer-independence. The trainers and testers 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

[1] Not all volunteers finished the testing task, and some word samples were omitted due to being incomplete. Thus, on average, the number of words per tester was 236. The average number of word parts per word is 2.64.

delayed strokes. We tested our system with five different dictionary sizes: 5K, 10K, 20K, 30K, and 40K words selected from Arabic Treebank [10], twenty random articles from Al-Arabi Magazine[1], and ten random articles from the website of the news channel Aljazeera[2]. The 280 test words were present in all dictionary sizes. The purpose of the various dictionary sizes is to test our system's performance under different ambiguity conditions. We present our results in terms of two metrics: *word recognition rate* (Table 2) and *word-part recognition rate* (Table 3).

Overall, we get good results given that we used a relatively small training set. The difference between the writer-independent and writer-dependent recognition rates is less than 2%, with all tested dictionary sizes. This implies that the features, model, and delayed stroke algorithm we introduced are adequate for the task of writer-independent handwriting recognition.

The performance degrades as expected as ambiguity (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. In fact, recognition errors are mostly of very close looking word-parts such as با / ب and د / ر. The current features used are not good at distinguishing these word parts adequately.

Table 2: Writer dependent (WD) and writer independent (WI) average word recognition rates for 2,358 words written by ten testers (all values are in percentage.)

	5K	10K	20K	30K	40K
WD	96.47	95.50	92.86	90.84	89.75
WI	96.28	95.21	92.55	89.68	88.01

Table 3: Writer dependent (WD) and writer independent (WI) average word-part recognition rates for 6,220 word parts written by ten testers (all values are in percentage.)

	5K	10K	20K	30K	40K
WD	98.44	97.94	96.86	95.90	95.44
WI	98.49	97.78	96.54	95.12	94.40

There are no previous results on Arabic online handwriting recognition that we can compare to, since previous work on Arabic online recognition was limited in test size and/or letter coverage [2][3].

8. Conclusion and Future Work

This paper introduced an HMM based system with novel components to provide solutions for most of the difficulties inherent in recognizing Arabic script: letter connectivity, position-dependent letter shaping, and delayed strokes. An evaluation of the system shows the features and model selected to be adequate for the task of writer-independent handwriting recognition at a high rate of word recognition.

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 the previous section using geometric-computation techniques in an additional postprocessing phase. Moreover, we plan on exploring sentence-level language modeling to improve word recognition [11]. And finally, we plan to explore morphologically driven models of Arabic words to improve the dictionary's efficiency and coverage.

[1] www.alarabimag.com
[2] www.aljazeera.net

References

[1] Badr Al-Badr and Sabri A. Mahmoud. Survey and bibliography of Arabic optical text recognition. *Signal Processing*, 41(1):49–77, 1995.

[2] Samir Al-Emami and Mike Usher. On-line recognition of handwritten Arabic characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):704–710, July 1990.

[3] Adel M. Alimi. An evolutionary neuro-fuzzy approach to recognize on-line Arabic handwriting. In *Proceedings of the 4th International Conference Document Analysis and Recognition*, pages 382–386, 1997.

[4] Adnan Amin. Machine recognition of handwritten Arabic word by the IRAC II system. In *Proceedings of the 7th Joint on Pattern Recognition*, pages 35–37, Munich, Germany, October 1982.

[5] 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):112–122, 1973.

[6] T.S. El-Sheikh and S.G. El-Taweel. Real-time Arabic handwritten character recognition. Pattern Recognition, 23(12):132301332, 1990.

[7] Jianying Hu, Sok Gek Lim, and Michael K. Brown. Writer independent on-line handwriting recognition using an HMM approach. *Pattern Recognition*, 33(1):133–147, 2000.

[8] Jianying Hu, S.C. Oh, J.H. Kim, and Y.B. Kwon. Unconstrained handwritten word recognition with interconnected Hidden Markov Models. In *proceedings Third Int. Workshop on Frontiers in Handwriting Recognition*, pages 455-560, Buffalo, USA, May 1993.

[9] Jay J. Lee, Jahwan Kim, and Jin H. Kim. Data-driven design of HMM topology for online handwriting recognition, pages 107–121, 2002.

[10] Mohamed Maamouri, Ann Bies, Hubert Jin, and TimBuckwalter. Arabic treebank: Part 1 v 2.0. In *Linguistic Data Consortium, LDC Catalog No.: LDC2003T06*, 2003.

[11] John Makhoul, Thad Starnert, Richard Schwartz, and George Chou. On-line cursive handwriting recognition using speech recognition methods. In *Proceeding of IEEE ICASSP'94 Adelaide*, pages v125–v128, Adelaide, Australia, April 1994.

[12] N. Mezghani, A. Mitiche, and M. Cheriet. On-line recognition of handwritten Arabic characters using a kohonen neural network. In *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition*, page 490, Washington DC, 2002.

[13] Se-Chang Oh, Jin-Young Ha, and Jin H. Kim. Context dependent search in interconnected Hidden Markov Model for unconstrained handwriting recognition. *Pattern Recognition*, 28(11):1693–1704, 1995.

[14] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *IEE Proceedings on Readings in Speech Recognition*, 77(2):257–286, 1989.

[15] L. Schomaker. Using stroke- or character based self-organizing maps in the recognition of online, connected cursive script. Pattern Recognition, 26(3):443–450, 1993.