

# Diacritical processing for unconstrained on-line handwriting recognition using a forward search

Giovanni Seni, John Seybold\*

Motorola Lexicus Division  
3145 Porter Drive  
Palo Alto, CA 94304, USA  
giovanni@lexicus.mot.com

The date of receipt and acceptance will be inserted by the editor

**Abstract.** Out-of-order diacriticals introduce significant complexity to the design of an on-line handwriting recognizer, because they require some reordering of the time domain information. It is common in cursive writing to write the body of a ‘i’ or ‘t’ during the writing of the word, and then to return and dot or cross the letter once the word is complete. The difficulty arises because we have to look ahead, when scoring one of these letters, to find the mark occurring later in the writing stream that completes the letter. We should also remember that we have used this mark, so that we don’t use it again for a different letter, and we should also penalize a word if there are some marks that look like diacriticals that are not used. One approach to this problem is to scan the writing some distance into the future to identify candidate diacriticals, remove them in a preprocessing step, and associate them with the matching letters earlier in the word. If done as a preliminary operation, this approach is error-prone: marks that are not diacriticals may be incorrectly identified and removed, and true diacriticals may be skipped. This paper describes a novel extension to a forward search algorithm that provides a natural mechanism for considering alternative treatments of potential diacriticals, to see whether it is better to treat a given mark as a diacritical or not and directly compare the two outcomes by score.

**Key words:** Handwriting recognition – Cursive script recognition – Online recognition – Search algorithms

## 1 Introduction

Research in speech and handwriting recognition – the translation of a person’s voice or writing into computer readable text – has been ongoing since at least the 1960’s. The two problems share many common characteristics:

---

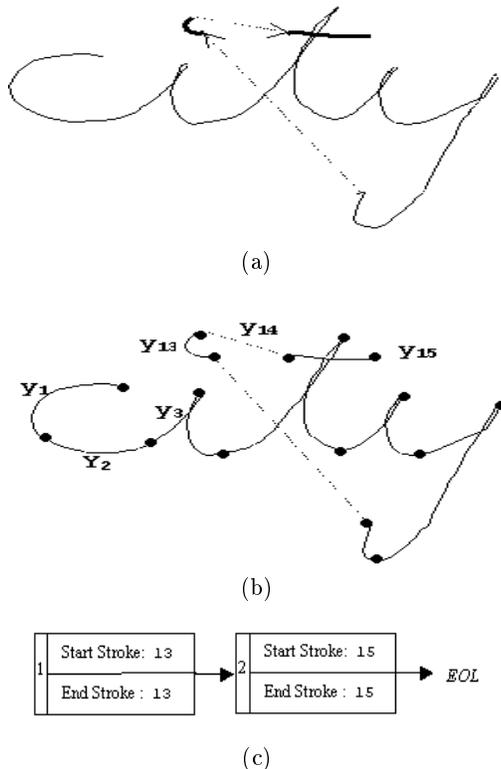
\* Dr. Seybold was with the Motorola Lexicus Division when the paper was submitted. He can be reached by e-mail: johnseybold@post.harvard.edu

processing of a signal with inherent temporal structure, recognition of a well-defined and finite set of symbols (i.e., characters and phonemes), composition of these basic symbols to form words and phrases, and presence of the coarticulation effect. Because of these similarities, techniques developed to solve one problem can often be applied to the other.

The temporal information of handwriting gets lost when the writing is scanned from a page (“off-line” recognition), but it is available when the writing is captured using a digitizing tablet (“on-line” recognition). Recognition in the off-line case is more difficult because it is necessary to deal with overlapping or touching characters, unintentional pen lifts, and different stroke widths, which sometimes significantly alter the topological pattern of characters in the input script. On the other hand, these problems have little or no influence on the dynamic pattern of motion in the writing. Therefore, on-line recognition systems commonly use time-based representation schemes.

Time-based representations are subject to a source of misinformation as well. For instance, the letter ‘E’ can be written using multiple pen trajectories generating temporal variations that are not apparent in its static representation. While such variations in trajectory can be relatively large in isolated characters, it is believed that the number of variations is limited when writing naturally run-on text. A second difficulty with time-based representations is that of delayed diacriticals. A *delayed diacritical* is a piece of ink used to complete a character but which does not immediately follow the first portion of that character. For example, the word “city” is usually written in cursive style with three ink parts: the first is the main body of the word, the second the dot for the ‘i’, and the third the cross of the ‘t’ (see Figure 1a). Delayed i-dots and t-crossings constitute a problem for time ordered representations because all the evidence for characters ‘i’ and ‘t’ may not be contiguous. Other problem characters are ‘f’ and ‘z’ which may contain a small horizontal bar crossing in the middle. In fact, any multi-component character will exacerbate the problem because each pen lift is a moment of free placement deci-

sions in the writer, mixing diacriticals with other isolated components. Therefore, when processing the writing in temporal order, it is necessary to somehow reassemble a character with the delayed ink before submitting it to the character recognizer for scoring. Furthermore, we should also remember that we have used a given piece of ink as a diacritical mark, so that we don't use it again for a different letter, and we should also penalize a word if there are some marks that look like diacriticals that are not used.



**Fig. 1.** Example image for the word “city” written in cursive style with delayed diacriticals. In (a) the order in which the body, i-dot, and t-cross of the word were written is shown. In (b) the same image has been divided into a sequence of basic ink units (i.e., “strokes”)  $Y = y_1, y_2, \dots, y_{15}$ . In (c) the corresponding list of (two) potential diacriticals.

In this paper we present an efficient extension to a forward search algorithm that provides a natural mechanism for considering alternative treatments of potential diacriticals, to see whether it is better to treat a given mark as a diacritical or not and directly compare the two outcomes by score. Our method also provides means for a strict accounting of the ink in the input image; that is, enforcing that every piece of ink in the input is used exactly once. The organization of the paper is as follows: Sect. 2 explains previously used methods for dealing with delayed diacriticals, Sect. 3 briefly talks about search algorithms, and Sect. 4 describes our proposed diacritical processing method. Sect. 5 presents the architecture of our system where the proposed method has been incor-

porated, Sect. 6 covers a simple search example, and in Sect. 7 we report the impact in our system of applying the method, followed by concluding remarks.

## 2 Previous approaches

Recognition systems using a time-based representation scheme start by converting an unknown input ink into a sequence of feature vectors,  $Y = y_1, \dots, y_T$  using a pre-processing module. Each of these vectors typically represents geometrical properties of a short contiguous ink fragment (see Figure 1b).

Removing delayed vectors from this sequence [3] or trying to reinsert them at a better position [4] [5] are two previously proposed procedures to address the problem of delayed diacriticals. For the sample image of Figure 1 such methods might reorder the original input vector sequence  $Y = y_1, \dots, y_t, \dots, y_{15}$  into  $Y = y_1, \dots, y_4, y_{13}, y_5, y_6, y_{15}, \dots, y_t, \dots, y_{11}$  so that all the ink for the letters ‘i’ and ‘t’ – namely  $y_3, y_4, y_{13}$  and  $y_5, y_6, y_{15}$  respectively, is contiguous. One difficulty with these approaches is that very often it is not obvious to what point of the word the delayed mark should be linked; particularly, because these marks are usually carelessly positioned, the closest point in the word may not correspond to the intended location. Furthermore, these approaches may end up discarding ink valuable for character disambiguation (say between ‘i’ and ‘e’ or between ‘t’ and ‘l’). Finally, because these approaches tend to require a ‘hard’ decision early in processing, any mistakes are likely to result in non-recoverable errors.

A third approach pairs each vector  $y_t$  in the input trajectory representation with an indication of whether or not those sections of the ink were later horizontally covered by a diacritical mark [6] [7]. Continuing with the example of Figure 1, using such an approach the input vector sequence for that image might be  $Y = y_1, \dots, \hat{y}_3, \hat{y}_4, \hat{y}_5, \hat{y}_6, \hat{y}_7, \dots, y_t, \dots, y_{15}$  where a ‘hat’ feature has been added to input vectors  $y_3$  through  $y_7$  to indicate that they are covered by a potential diacritical. One advantage of this approach is that it eliminates the ‘hard’ decision, and so is less likely to be brittle.

The problem with this approach is that it does not allow for a strict accounting of the ink. We would ideally like to combine the ‘soft’ characteristics of trying many possible rearrangements of delayed diacriticals and assignments to character bodies, with a strict accounting that allows us to make sure that all of the ink has been used correctly.

## 3 Search algorithms

Current large vocabulary speech and handwriting recognition systems are firmly based on the principles of statistical pattern recognition. The recognition task is typically enunciated as that of finding the most probable word, or word sequence,  $\hat{W}$  for a given input image or utterance  $Y$ . That is,

$$\hat{W} = \arg \max_W P(W|Y) \quad (1)$$

Finding  $\hat{W}$  is a *search* or *decoding* problem and two main approaches are possible: *depth-first* and *breadth-first* [2]. The strategy followed by depth-first search is to pursue a particular hypothesis deeper and deeper until the end of the input is reached or a maximum accumulated score is exceeded. Examples of depth-first decoding are *stack-decoders* and *A\*-decoders* [1].

Breadth-first search is so named because all hypotheses are pursued in parallel, in a forward “time synchronous” manner. That is, the algorithm explores all hypotheses for time step  $t$  – i.e., for input  $y_1, \dots, y_t$ , before exploring any hypothesis for time step  $t + 1$ . In general, a breadth-first search proceeds by expanding a tree of theories (usually derived from a dictionary). A theory consists of a record of the string to that point in time, and a current character hypothesis to be scored. Each theory thus encodes an interpretation of all of the ink processed to that point by the search. As the search proceeds, each theory spawns new theories corresponding to possible legal extensions of its string.

Breadth-first decoding exploits Bellman’s optimality principle and is often referred to as *Viterbi-decoding*. The exponential growth of theories is limited by competitive mechanisms, usually based on selecting for extension some set of best-scoring theories [8]. A range of qualities below the most likely theory is defined, yielding a beam of surviving theories. Another criterion is to allow a maximum number of theories to survive, determining the maximum beam width directly. In our system, on the order of a few hundred theories are maintained simultaneously.

## 4 Diacritical accounting

The use of a *beam search* gives us a natural mechanism for considering alternative treatments of diacriticals and diacritical-containing characters. All we need to do is to generate theories that embody different diacritical treatments and score them, letting the normal competitive mechanisms determine which theories survive to propagate.

The main issue then becomes how to account for the ink contained in hypothetical diacriticals. Specifically, we would like to ensure that *a*) if a given piece of ink is used as a diacritical mark, e.g. as the dot of an ‘i’, it is not used again later as either a diacritical mark or a separate character, and *b*) if a piece of ink is not used as a diacritical, it must be used as another character, or, conversely, the theory must be penalized for not using it.

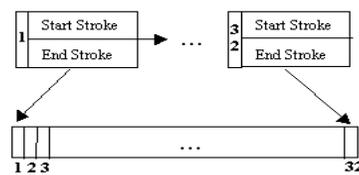
### 4.1 Isolated word recognition

We introduce the implementation of this idea by first considering the case of a single word that is completely written before recognition starts. The first step involves the identification of likely diacritical marks by scanning the ink. Good candidates for diacritical marks are short,

simple pieces of ink. We build a list of these potential diacriticals (see Figure 1c). At recognition time, as we traverse the ink with the search, we process the diacritical-containing letters by scoring them twice. One variant is scored without adding any diacritical. The other variant is scored by selecting the most appropriate diacritical mark from the list and presenting the ink for the body of the character together with the diacritical to the character recognizer for scoring.

We use a fuzzy logic method to identify the most appropriate diacritical. For example, if we are looking for a t-cross, we will develop a score for each potential diacritical in the list using a weighted sum of various measurements, including the distance of the midpoint of the diacritical from the midpoint of the character body, the extent to which the diacritical is above the center line of the writing, the extent to which it is a short, horizontal line that is wider than it is tall, and so on [9]. We take the diacritical that scores highest as the diacritical to attach to the character hypothesis for scoring.

Once we have scored the character both ways (with and without a diacritical) we take whichever variant scored highest and use that as the score for the theory. Now we must account correctly for the use (or non-use) of the diacritical. What we need to do is to add a means for recording the diacriticals that have been used to this point, and to carry this information forward as we propagate the theory to subsequent letters. We accomplish this by adding a 32-bit bit-field to each theory. Each bit in the field corresponds to an entry in the list of diacriticals (see Figure 2). The contents of the bit field are copied into each new theory as it is created by propagation, so that each theory can know what diacriticals have been used by the theories that preceded it.



(a)

Model	Start Time	Start Score	Start Diacs	Score	Diac
Exit Score					
Exit Used Diacs					

(b)

**Fig. 2.** Mapping between the list of potential diacriticals and a bit-field, (a) for every entry in the list there is a bit in the bit-field (up to 32 entries). In (b), typical structure of a “theory” in a forward-search extended to include diacritical related fields. *StartDiacs* is the bit-field inherited from the parent theory, *Diac* is the bit-field of diacriticals used by this theory’s model, and *ExitUsedDiacs* is the logic OR of the two, which is to be passed to descendant theories. Dashed fields are computed but not really stored.

Continuing the example, if the best t-cross was entry number 2 in the diacritical list, we would set the second bit in the bit field of the 't' theory making use of this diacritical. This method of accounting provides extremely compact storage for the used diacritical information; it allows us to test very efficiently for the conditions pertaining to our rule that says that ink must be used exactly once, and it allows us to refuse to score (or give a very bad score to) a character that uses ink that overlaps any used diacritical.

Enforcing that a given diacritical is not used more than once is done during the step where we identify the best matching diacritical for a given character; we can refuse to consider any diacriticals that have already been used, simply by checking the appropriate bit in the bit-field attached to the theory (this is a binary AND operation, which is ordinarily a single clock cycle instruction). One possible exception to this rule is that of a double 't' written with a single crossing. This case can be handled by allowing 't' theories to score "borrowing" the diacritical from its predecessor, when such predecessor happens to be a 't' theory as well.

We can also enforce that all of the ink is used in some way as follows. When we score a word-ending theory, we can check that all subsequent ink has been used as a diacritical. This is often the case when diacriticals are written at the end of the word – the last character ends, but there is still some ink in the original writing order that should have been used up in the course of processing the various t's, i's, x's, and apostrophes in the body of the word. To the extent that this ink has not been used, or worse, does not appear at all in the list of diacriticals, we should view the theory with suspicion and penalize it.

Finally, we need to enforce that character theories don't use ink that overlaps diacriticals already used by parent theories. We do this by computing the "intersection" between the input vectors associated with the given theory,  $y_{StartTime}, \dots, y_t$ , and the input vectors associated with each of the diacriticals marked in the bit-field attached to the theory. When a conflict is detected (i.e., a non-empty intersection) the theory is not scored or is given a very bad score. However, when such a conflict is at the beginning of the theory's input vector range (e.g., from  $y_{StartTime}$  to  $y_{t_{conflict}}$  for  $t_{conflict} < t$ ), the theory can be scored using the ink corresponding to the non-conflicting region. In this way we effectively skip over delayed diacriticals that are written in the middle of the word.

#### 4.2 Continuous recognition

This is fine, as far as it goes, but what happens when we try to recognize a larger piece of text? Eventually the number of potential diacriticals will exceed the size of our bit-field, forcing us to increase the size of the structure if we are to track the diacriticals accurately. Fortunately, we can use the time synchronous nature of the forward search to solve this problem. The key insight is that once the search point has advanced beyond where the diacrit-

ical was actually written, we can forget about it – that is, it must either have been used earlier as a diacritical, or more recently as part of a character. In either case, however, it has been accounted for and we no longer need to track it. So, we can turn our list of diacriticals into a ring, in which we replace the oldest diacriticals once the search point has advanced beyond them.

The easiest approach is to find the first 32 diacriticals in writing order, and then stop our scan. As we search past diacriticals in the list, we replace them in order by restarting our diacritical scan where we left off before. This approach gives us the ability to account efficiently and accurately for every potential diacritical in the writing, subject to two limitations. First, delayed diacriticals must be written after the character in which they appear – otherwise we cannot discard them once the search point passes them, as they would be needed to complete a character in the future.

In practice, this rules out writing in which a t-cross is written before the word, and then the word is written underneath it to link up. This is a degenerate case, and we have not observed any instances in our large data sets. Note that this restriction applies only to delayed (perhaps the correct term here is anticipatory) diacriticals – it is perfectly acceptable to draw the t-cross first and then immediately draw the body of the 't'. In this case, the normal left-to-right flow of the writing is preserved and the search will automatically create a character hypothesis containing all of the relevant ink.

The second limitation is equally minor – the writer must complete all of the characters in a word by writing their diacriticals before advancing too far down the line of writing. If the writer writes an entire sentence before dotting an 'i' in the first word, the diacritical list may fill up before the dot is encountered. Generally, people complete each word as it is written before going on to the next word, and it seems not an unreasonable restriction on a real system to require this.

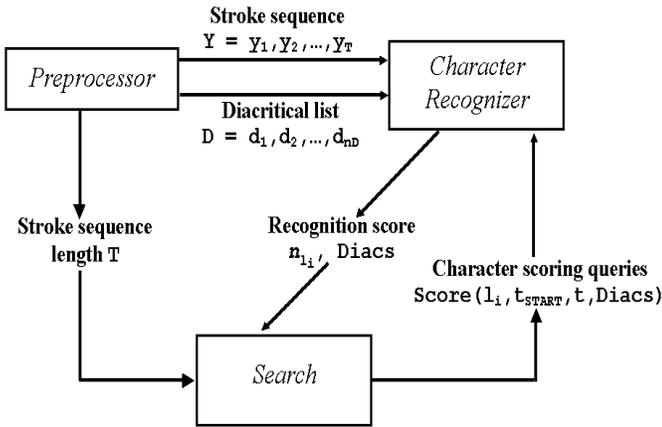
## 5 System architecture

In Figure 3 we present a diagram that describes the architecture of our run-on handwriting recognition system where this diacritical accounting method has been incorporated. The system is composed of three modules: *Preprocessor*, *Character Recognition*, and *Search*. The Preprocessor performs some low-level operations on the raw ink such as re-sampling, filtering, smoothing, and size normalization [10]. It then segments the cleaned and normalized ink into a sequence of strokes<sup>1</sup> (e.g.,  $Y = y_1, y_2, \dots, y_T$ ) and identifies potential diacriticals (e.g.,  $D = d_1, d_2, \dots, d_{n_D}$ ).

The stroke sequence  $Y$  and diacritical list  $D$  are made available to the Character Recognizer module. The length of the stroke sequence is passed on to the Search module, which carries on  $T$  score-propagate steps. At each time step, the observed input  $y_t$  is compared against a set of open theories, and a probability of that theory

<sup>1</sup> A stroke is a short contiguous ink fragment.

generating the input is multiplied into the theory's ongoing score. At any time, the likelihood of a theory is a product of  $n$  probabilities, one for each time step. The theories can be directly compared, and some subset may be selected for further evaluation. Each theory has the possibility of propagating to generate a set of new theories which encode possible completions of the utterance from the current point in time. By carefully controlling the rate of new theory propagation and the destruction of obsolete theories, the exponential growth of the search space is controlled.



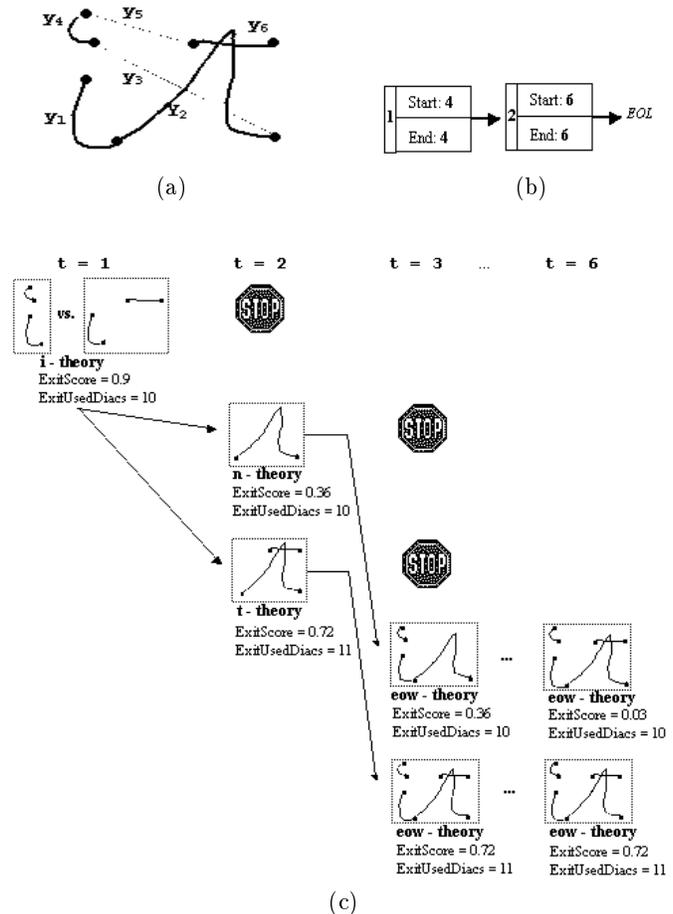
**Fig. 3.** System architecture for handwriting recognition of run-on text with efficient diacritical accounting procedures. There are three main modules in the system: *Preprocessor*, *Character Recognizer*, and *Search*. The *Preprocessor* decomposes input ink into a sequence of strokes and generates a list of potential diacriticals. The *Character Recognizer* computes scores in response to character recognition queries. The *Search* uses a dictionary to generate interpretation strings by combining character scores.

Probabilities are generated by sending character scoring queries to the Character Recognizer. These queries specify the letter associated with the theory in question  $l_i$ , the time step when the theory was created  $t_{START}$ , the current time step  $t$ , and a bit-field with bits on for already used diacriticals  $Diacs$ . The recognizer responds with a score  $n_{l_i}$  and an updated diacriticals bit-field  $Diacs$  if an unused diacritical mark was claimed by the given character hypothesis. At the end of the  $T$  time steps, the system outputs a ranked list of possible interpretation strings.

## 6 Search example

In Figure 4 we present a complete search example with delayed diacriticals. In this example, input ink for the word “it” has been divided into six strokes and two potential diacriticals have been identified.

The example further assumes a dictionary of two words only – namely, “in” and “it”. Accordingly, the decoding process starts ( $t = 1$ ) by creating a theory for the letter ‘i’. A request to score this character hypothesis is



**Fig. 4.** Search example with delayed diacriticals. In (a), example image for the word “it” shown divided into a sequence of six strokes. Also shown (b) is the associated list of potential diacriticals. In (c), a search tree for a dictionary of two words only (“in”, “it”). A ‘STOP’ mark next to a theory means that such theory is killed at that time step. Model “eow” stands for end-of-word.

sent to a character recognizer which tries to score the letter ‘i’ with each of the two available potential diacriticals. The first diacritical is preferred over the second one and a score of 0.9 is returned for the hypothesized ‘i’ letter. The character recognizer also returns an indication of having used the first diacritical in the diacritical list. That is,  $ExitScore = 0.9$  and  $ExitUsedDiacs = 10$ . This ‘i’-theory then propagates into two descendant theories, one for the letter ‘n’ and one for the letter ‘t’. Therefore, at time step two ( $t = 2$ ) there are three active theories.

Note that the newly created theories will inherit from the ‘i’-theory its bit-field of used diacriticals as well as its score. At this time step the ‘i’-theory scores poorly and hence it is removed from further consideration. On the other hand, the ‘n’-theory and the ‘t’-theory score reasonably well ( $ExitScore$  equal to 0.36 and 0.72 respectively) and thus propagate into end-of-word theories (“eow”). At the end of time step two the  $ExitUsedDiacs$  bit-field in the ‘t’-theory indicates that the character recognizer used the second potential diacritical from the list.

This diacritical was the correct one but also the only one available since diacritical number one was already used by a parent theory.

At time step three ( $t = 3$ ), the ‘n’-theory and the ‘t’-theory score badly and thus are deactivated; the “eow”-theories score well and continue active. At the last time step ( $t = 6$ ), the “eow”-theory corresponding to the word “in” scores poorly because it leaves the ink in one of the two potential diacriticals unaccounted for. The “eow”-theory for “it”, however, scores well because the two diacriticals were already used at previous time steps.

## 7 Results and discussion

The diacritical accounting procedure described above has been incorporated into a system for unconstrained on-line handwriting recognition. This system has been ported to the WindowsCE Palm-size PC platform, where it currently runs on real time at the word level. Tested on a disjoint database of 10,000 unconstrained words from 62 different writers, and using a lexicon of 25K words, the extended system achieved a 15% error reduction in the top output string. On a second disjoint test set of 1000 sentences from 121 writers, and using the same lexicon, the system achieved a 12% error reduction in (top-choice) word error rate.

It is difficult to convey the significance of the proposed diacritical accounting procedure based on this result only since we did not establish how many delayed diacriticals were present in the above mentioned test databases. However, we can estimate the number of total diacriticals from the “truth” labels. In the test set of isolated words there were 5852 labels which contained at least one of the characters ‘i’, ‘t’, ‘f’, ‘x’, ‘j’, or ‘z’. If only 25%<sup>2</sup> of the corresponding images contained a delayed diacritical, then the reported 15% error reduction was derived from error correction in 1463 images, or about 14% of the test set. This would seem to indicate that delayed diacriticals constitute a high source of errors, and that our proposed diacritical processing method is having a significant impact on the performance of the system.

Finally, we would like to comment on a further limitation of the approach we have described which may not be immediately apparent that relates to the “admissibility” of the search. An *admissible search* is one in which, if no pruning were to take place, every possible interpretation of the input would be generated. By taking the best scoring of the two hypotheses (one with, and one without a diacritical), we have made the search inadmissible – that is, we have made a hard decision early on without knowing the later impact that decision will have, in effect ruling out some hypotheses from ever being considered. It would be better in principle to generate multiple alternatives, with and without diacriticals, and propagate them forward independently. Whether this would have a

material impact on accuracy is not clear. In the current implementation we have chosen not to do this, primarily for code complexity and speed reasons.

## References

1. D. Paul. Algorithms for an Optimal  $A^*$  Search and Linearizing the Search in the Stack Decoder. In *IEEE Conf. on Acoustics, Speech and Signal Processing*, Canada, 1991.
2. T. Cormen, C. Leiserson, R. Rivest. Introduction to Algorithms. MIT Press. 1990.
3. L. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line, connected cursive script. In *Pattern Recognition*, 26(3):443-450, 1993.
4. C.C. Tappert. A divide-and-conquer cursive script recognizer. Research Report, IBM Watson Research Center, 1988.
5. P. Morasso *et al.* Recognition experiments of cursive dynamic handwriting with self-organizing networks. In *Pattern Recognition*, 26(3):451-460, 1993.
6. M. Schenkel, I. Guyon and D. Henderson. On-line cursive script recognition using time-delay neural networks and hidden markov models. In *IEEE Conf. on Acoustics, Speech and Signal Processing*, Australia, 1994.
7. G. Seni, R. Srihari, and N. Nasrabadi. Large vocabulary recognition of on-line handwritten cursive words. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6), June, 1996.
8. J.J. Odell, V. Valtchev, P.C. Woodland, S.J. Young. A one pass decoder design for large vocabulary recognition. In *Proc. Human Language Technology Workshop*, Plainsboro, NJ, March, 1994.
9. G. Seni and E. Cohen. External word segmentation of offline handwritten text lines. In *Pattern Recognition*, 27(1), January, 1994.
10. W. Guerfali and R. Plamondon. Normalizing and restoring on-Line handwriting. In *Pattern Recognition*, 26(3):419-431, 1993.

**Giovanni Seni** received a Ph.D. in Computer Science from State University of New York at Buffalo (SUNY at Buffalo) in 1995, where he studied on a Fulbright scholarship. His research was on the use of convolutional neural networks for recognizing on-line cursive words in a large vocabulary setting. He was a Graduate Research Assistant at the Center of Excellence for Document Analysis and Recognition (CEDAR), during 1992-1995. Dr. Seni left CEDAR to join the Motorola Lexicus Division in Palo Alto, California, where he serves as co-principal investigator in the unconstrained handwriting recognition project. He is also technical lead for other related technologies such as ink compression and scribble matching. His present research interests are in pen based systems, computer vision, pattern recognition, and neural networks.

<sup>2</sup> This is an approximation to the frequency of delayed diacriticals based on visually sampling a couple of hundred images.

**John Seybold** received a D.Phil in Psychology from Oxford University, where he studied on a Rhodes scholarship. His research was on the application of Hopfield networks to spoken word recognition. In 1992, he joined Lexicus, a startup company developing cursive handwriting recognition software, where he worked on two generations of recognition systems. He has focused particularly on the design and implementation of high speed breadth-first searches for integrating recognition information. He has also worked on Chinese character recognition, and has been awarded several patents.