ELSEVIER

Pattern Recognition Letters 16 (1995) 999-1009

Pattern Recognition Letters

Word-level recognition of small sets of hand-written words

A. Eliaz [1], D. Geiger *

Computer Science Department, Technion IIT, Haifa 32000, Israel

Received 10 June 1994; revised 26 April 1995

Abstract

A system is presented for off-line word-level recognition of small sets of hand-written words. The system performs well on a variety of tasks such as recognizing numerals hand-written in English (one through ten), Hebrew characters, and Pascal's reserved words. Only small changes are needed in the parameters of the system when switching from one application to another. The heart of the proposed system is the use of dynamic programming for matching between a word in question and a prestored model of a word. The matching algorithm which uses line segments as its input is similar to Lee and Chen's algorithm for matching Chinese characters. We vary the segmentation algorithm showing that regardless of the segmentation algorithm chosen, the average recognition rate usually exceeds 95%. Different thinning algorithms are also shown not to reduce the stated recognition rate and a rejection scheme further enhances it.

1. Introduction

Off-line recognition of script handwriting is a hard problem for which a satisfying solution has yet to be found. The fact that spacing and sizes vary in script handwriting, letters are connected in different fashions, and the many possible ways of writing each word, are among the reasons for this difficulty. Most recognition systems to date for script handwriting use temporal information, such as the order of strokes, measured while the author writes the text (Mantas, 1986; Tappert et al., 1990). Such techniques, which are called on-line, are useful in applications where a special input device can be supplied to the user. For example, input pads of some commercially available computers.

There are two approaches for off-line script handwriting recognition. One approach is dividing each input word into individual letters and identifying each letter separately and the second is to identify each word as a whole entity. The advantage of the first approach is its generality. Each word in western languages is composed of a small set of letters for which recognition procedures have been devised. The difficulty in this approach is finding the correct partition of a word into its composing letters. The second approach which we call, *word-level* recognition, suffers from the fact that the number of possible words in a language is enormous. Consequently, rough methods are applied in order to form small sets of candidate identifications.

Word-level recognition is most appropriate for applications where the input dictionary is limited, e.g., numerals hand-written in English on bank checks. In this correspondence, we describe an off-line word-level recognition system for script handwritten words that are selected from a small set of possible words. Although the system was originally developed for recognizing numerals (one through ten) hand-written in English, we have tested it also on two other applica-

* Corresponding author. Email: dang@csd.cs.technion.ac.il.

[1] Current affiliation: IBM Research Center, Haifa, Israel.

SSDI 0167-8655(95)00065-8

tions and found that the system works well after minor adjustments. We applied the system to recognize letters of the Hebrew alphabet as well as Pascal's reserved words and obtained recognition rates in most experiments that exceed 95%. Section 5 summarizes the experimental results.

Our recognition system constructs for each word in a given dictionary several word-models reflecting different styles of writing the word. Each word-model is a set of segments characterized by its end-points' coordinates. After scanning several representative examples for each word in the dictionary, preprocessing algorithms that include normalization, thinning and segmentation yield word-models which we call *templates*. Preprocessing is described in Section 2. At the recognition phase a given word is preprocessed, its word-model is formed and then compared to all templates using a *similarity score* defined in Section 3. The identity of the most similar template is selected as the identity of the given word. The basis of this recognition algorithm is the observation that variations in the segments' features among word models belonging to the same word are smaller than between word-models originating from distinct words. Its success hinges upon a careful selection of the similarity score.

When comparing two segments of two word-models, the similarity score takes into account parameters of the individual segments consisting of position and slope as well as parameters of the segments that reside in proximity to the two segments being compared. It is the second factor that causes the recognition algorithm to perform well in different applications. This score is based on Lee and Chen's work which used a close variant of it to match Chinese characters (Lee and Chen, 1992). As demonstrated by Section 5, this similarity score performs well under a variety of preprocessing algorithms and recognition tasks. Its success is not limited to Chinese characters.

2. The preprocessing phase

Even for a small collection of, say, thirty handwritten words, using the entire bitmap produced by optical scanners as the input to a matching algorithm consumes too many computational resources. Consequently, bitmaps are usually processed so that each word is described with fewer parameters. We represent each word by a set of segments called a word-model.

Definition 1. A *word-model* M is a set of segments each represented by two pairs, (x_1, y_1) and (x_2, y_2), which are the coordinates of its end-points. The number of segments in M is denoted by $|M|$.

Given an input bitmap resulting from an image scan, we use the following four steps to produce a word model: normalization, thinning, segmentation and merging. These steps are described in the sequel.

Normalization is the process of equating the size of all given bitmaps. The average size of the length and width of the bitmaps in the training group defines W, the normalized width corresponding to a length of 300 pixels. Every input bitmap P, of size $m \times n$, is transformed into a normalized bitmap of size $W \times 300$ using:

$$Q[i][j] = P[\lfloor i \cdot m/W \rfloor][\lfloor j \cdot n/300 \rfloor] \qquad (1)$$

for each $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, 299$.

The thinning algorithm we used is a standard 4-pass procedure from (Rosenfeld and Kak, 1990).The binary bitmaps are scanned in four directions: up-down, left-right, down-up, right-left. In each pass, border pixels which are *simple* and have more than one neighbor are erased. The process iterates until no more pixels can be erased. The resulting bitmap constitutes a *skeleton* of the original picture, that is, a 1-pixel wide picture consisting of a subset of the original black pixels. Most of the pixels of the skeleton have only two adjacent pixels. We call these pixels *regular* and all other pixels which have either more or less than two adjacent pixels are called *non-regular*.

The segmentation algorithm transforms paths of regular pixels between two non-regular pixels into a set of segments. For a given path, starting from pixel v with coordinates (v_x, v_y) to pixel u with coordinates (u_x, u_y), the segmentation algorithm follows the skeleton from one pixel to its neighbor in the direction of u. Each step to an adjacent pixel has a unit length and each step to one of the diagonally adjacent pixels has a length of $\sqrt{2}$. When the length being accumulated exceeds 12, the algorithm generates a new segment, by joining the current pixel with the end pixel of the last generated segment. The process stops when it reaches pixel u. This segmentation procedure is repeated for all paths that exist between

pairs of non-regular pixels until they all generate sets of segments.

Due to the thinning algorithm, occasionally several non-regular pixels are formed in close proximity. The merging algorithm searches such pixels around each non-regular pixel. This is done by setting up frames of growing odd sizes around each non-regular pixel until no further non-regular pixels are found. The word model is then changed as follows. Every segment which had one endpoint outside the frame and the other endpoint inside the frame is replaced with a segment having the same endpoint outside the frame as before and having the central pixel as its endpoint inside the frame. Short segments that were entirely inside the frame are eliminated from the word-model.

In Section 4 we describe variations of the segmentation and thinning algorithms and examine their influence on the recognition rate.

3. The matching algorithm

The matching algorithm finds the best match between a set of prestored word-models and a word-model U of an unknown word. The algorithm measures the similarity between U and each stored word-model and selects the model that is most similar. Similarity is a function

$$S : \textit{word-models} \times \textit{word-models} \rightarrow [0, 1)$$

that given two word-models produces a number between 0 and 1 called the *similarity score*. Two word-models are identical when their similarity score is 0 while close to 1 similarity scores indicate major differences.

The similarity function S is defined procedurally. This function compares pairs of segments taken from two given word-models. When comparing two segments, S takes into account parameters of the individual segments consisting of position and slope as well as parameters of the segments that reside in proximity to the two segments being compared. The first score is called *individual pairwise similarity* and the second is called the *neighborhood pairwise similarity*. The average of the individual and neighborhood pairwise similarities defines the *pairwise similarity*. Finally, the average of the pairwise similarity over all pairs of segments yields the similarity between U and M. A detailed description of this procedure is provided below.

Let U and M be two word models. For each pair of segments $u_i \in U$ and $m_j \in M$ the *individual pairwise similarity* is defined by the following equation:

$$IS_{ij} = \alpha \cdot \left(\frac{d_{UM}(i,j)}{\sqrt{2} \cdot w - d_{UM}(i,j)} \right) + (1 - \alpha) \cdot \left(\frac{|sd_{UM}(i,j)|}{\pi - |sd_{UM}(i,j)|} \right) \quad (2)$$

where $d_{UM}(i,j)$ is the distance between i and j's center-points, sd_{UM} is the absolute value of the slopes differences of the two segments i and j, ranging from 0 to $\pi/2$, and $\alpha \in [0, 1]$ is a mixing parameter.

Note that IS_{ij} is in $[0, 1)$, obtains a zero value for identical segments and approaches 1 as sd_{UM} or $d_{UM}(i,j)$ or both increase. The slopes and center points are computed from the coordinates (x_1, y_1) and (x_2, y_2) of each segment using standard equations. The coordinates system is the same for both U and M segments due to normalization.

While IS_{ij} involves only properties of the segments themselves, the neighborhood similarity adds information about the segment's surroundings in the word model.

Definition 2. In a word-model A, the *w-neighborhood* $N(k)$ of a segment $a_k \in A$ is the set of segments that have their center-point located in a $w \times w$ window around a_k center-point (excluding a_k itself).

Each segment a_ℓ in a w-neighborhood $N(k)$ of segment a_k is described by a *characterizing triplet* (x, y, z) representing the relationship between segments a_k and a_ℓ where x is the visibility angle of a_ℓ from a_k, ranging from 0 to 2π, y is the distance between the center-points of a_k and a_ℓ, ranging from 0

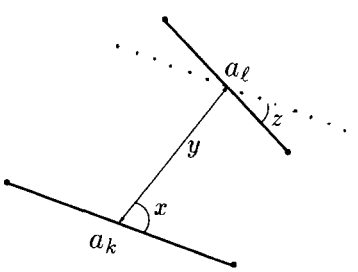

Fig. 1. A triplet from $N(k)$.

to $\sqrt{2}w/2$, and z is the slope difference of a_k and a_ℓ, ranging from $-\pi/2$ to $\pi/2$ (see Fig. 1).

Furthermore, for the purpose of computing similarity between two w-neighborhoods it suffices to compute characterizing triplets using a small finite number of values to each parameter. A *discrete characterizing triplet* (α, β, γ) is defined by the following equations:

$$\alpha = round(8x/\pi) \pmod{16}$$

$$\beta = trunc(10y/(\sqrt{2}w))$$

$$\gamma = round(8z/\pi + 8) \pmod{8}$$

Consequently, α, β, and γ can assume only 16, 10, and 8 integer values respectively. As a result, we gain a substantial speedup using discrete triplets for comparing two w-neighborhoods. Moreover, we hardly loose in recognition rate because distinct segments within the same w-neighborhood are rarely described by the same discrete triplet.

The *neighborhood similarity* between two w-neighborhoods $N(i)$ and $N(j)$ of segments u_i and m_j, is denoted by NS_{ij} and defined recursively in the sequel. Suppose $N(i)$ and $N(j)$ are sorted lexicographically according to α, β, γ. This means sorting the two w-neighborhoods $N(t)$ ($t = i$ or j) in a *counterclockwise* order, with the distance from t taken as a secondary key and the orientation relative to t as the least significant key. Denote the w-neighborhood of $N(t)$ by

$$(N(t)[k]_\alpha, N(t)[k]_\beta, N(t)[k]_\gamma)$$

for $k = 1, 2, \ldots, |N(t)|$ where $|N(t)|$ is the number of segments in $N(t)$. Let F_k and G_l stand for the sets of the first k and l triplets in $N(i)$ and $N(j)$ respectively. Note that according to our definitions

$$F_{|N(i)|} = N(i), \quad G_{|N(j)|} = N(j), \quad F_0 = G_0 = \emptyset.$$

In order to calculate NS_{ij}, we use the edit-distance introduced in (Wagner and Fischer, 1974):

$$\begin{aligned}\Delta(F_k, G_\ell) = \min\{&\Delta(F_{k-1}, G_{\ell-1}) + d_{k,\ell},\\ &\Delta(F_{k-1}, G_\ell) + 1,\\ &\Delta(F_k, G_{\ell-1}) + 1\}\end{aligned} \tag{3}$$

with

$$\Delta(\emptyset, \emptyset) = 0, \qquad \Delta(F_k, \emptyset) = k, \qquad \Delta(\emptyset, G_\ell) = \ell$$

for $k, \ell > 0$ and:

$$\begin{aligned}d_{k,\ell} = \frac{1}{3} \cdot \Bigg(&\left| \frac{N(i)[k]_\alpha - N(j)[\ell]_\alpha}{16} \right| \\ &+ \left| \frac{N(i)[k]_\beta - N(j)[\ell]_\beta}{10} \right| \\ &+ \left| \frac{N(i)[k]_\gamma - N(j)[\ell]_\gamma}{8} \right| \Bigg).\end{aligned} \tag{4}$$

NS_{ij} is computed using:

$$NS_{ij}(\emptyset, \emptyset) = 0, \tag{5}$$

$$NS_{ij} = \frac{\Delta(N(i), N(j))}{\max\{|N(i), |N(j)|\}}. \tag{6}$$

Note that NS_{ij} is a real number in the interval $[0,1)$. For exactly equal neighborhoods, $NS_{ij} = 0$ and for notable distinct neighborhoods NS_{ij} gets close to 1. The implementation of $\Delta(N(i), N(j))$ is done using the standard approach of dynamic programming.

Algorithm 1 (*Finding* $\Delta(N(i), N(j))$).

Input:

- $N(i)$, the neighborhood of segment $u_i \in U$ represented by n triplets ($n = |N(i)|$) of the form:

  $$(N(i)[k]_\alpha, N(i)[k]_\beta, N(i)[k]_\gamma),$$
  $$k = 1, 2, \ldots, n$$

- $N(j)$, the neighborhood of segment $m_j \in M$ represented by m triplets ($m = |N(j)|$) of the same form.

Output: $\Delta(N(i), N(j))$, the edit-distance between $N(i)$ and $N(j)$.

```
1   for k := 0 to n do δ[k][0] = k
2   for l := 1 to m do δ[0][l] = l
3   for k := 1 to n
4     for l := 1 to m
4.1     d1 = δ[k − 1][l − 1] + d_{k,l}
4.2     d2 = δ[k][l − 1] + 1
4.3     d3 = δ[k − 1][l] + 1
4.4     δ[k][l] = min(d1, d2, d3)
5   Δ(N(i), N(j)) = δ[n][m]
```

The value of $\Delta[k][l]/\max(k, l)$ gives the similarity between the first k segments in $N(i)$ and the first l segments in $N(j)$, calculated taking into account only these $k + l$ segments from the two ordered lists. It is computed using the values in the three adjacent cells

$\Delta[k-1][l]$, $\Delta[k][l-1]$ and $\Delta[k-1][l-1]$ which are obtained by earlier iterations.

The algorithm can be viewed as a minimal cost matching of the two neighborhood lists, segment by segment. It follows a path in Δ, from its upper-left corner, representing the initial two empty lists, to its lower-right one, representing the completion of $N(i)$ and $N(j)$, while maintaining two partly-built lists and adding segments to them as it advances.

The *pairwise similarity* is then given by

$$S_{ij} = \beta \cdot IS_{ij} + (1-\beta) \cdot NS_{ij} \tag{7}$$

where β is another mixing parameter of the algorithm.

We denote by $N_M(i)$ the set of segments from M which have their center-points located in a $w \times w$ window around u_i center-point, on the common coordinate system of M and U. Similarly, $N_U(j)$ denotes the set of segments from U which have their center-points located in a $w \times w$ window around m_j. The total similarity between the word-models U and M is defined by

$$S(U,M) = \frac{1}{|U|+|M|}\left(\sum_{u_i \in U, N_M(i) \neq \emptyset} \min_{m_j \in N_M(i)} S_{ij} + \sum_{m_j \in M, N_U(j) \neq \emptyset} \min_{u_i \in N_U(j)} S_{ij} + |A|\right) \tag{8}$$

where

$$A = \{u_i \in U \mid N_M(i) = \emptyset\} \cup \{m_j \in M \mid N_U(j) = \emptyset\}.$$

This formula matches each segment u_i in U with the most similar segment in $N_M(i)$ and each segment m_j in M with the most similar segment in $N_U(j)$. If either $N_M(i)$ or $N_U(j)$ is empty, a maximal penalty is added to the similarity score through the term $|A|$.

The matching algorithm just presented differs from the algorithm of (Lee and Chen, 1992) in Eqs. (4), (7) and (8). These changes were necessary in order to obtain the level of recognition rates reported in Section 5. We show, somewhat contrary to Lee and Chen's observations, that the matching algorithm is the single most important factor for achieving high recognition rates.

4. Alternative preprocessing steps

Our primary goal in experimentation has been to isolate the component of the recognition system that is most influential on the recognition rate. To do so, we have applied several thinning and segmentation algorithms to the same input in the preprocessing stage and examined the recognition rate. We noted that while each preprocessing algorithm generates different word models, there were only slight differences between the neighborhoods of segments generated from the same area. Consequently, as we shall see in the next section, recognition rates are not very sensitive to the preprocessing stages and are mainly determined by the neighborhood similarity function.

In this section, we describe two thinning algorithms (Naccache and Shinghal, 1984; Deutsch, 1972), denoted by **T2** and **T3**, respectively, and 3 segmentation algorithms (Dunham, 1986; Sklansky and Gonzalez, 1980; Reumann and Witkam, 1974), denoted by **S2**, **S3** and **S4**, that we use in the experiments. Together with **T1** and **S1**, the thinning and segmentation algorithms presented in Section 2, we have all together experimented with 3×4 different preprocessing possibilities applied to the same database.

Thinning algorithms which are also often called skeletonization algorithms, usually consist of an iterative process involving deletion of black pixels along the edges of the pattern, provided that their deletion does not remove endpoints and maintains the connectivity of the pattern. The process repeats until no more pixels are deleted. The remaining pixels constitute a skeleton of the original pattern. Thinning algorithms differ in the conditions by which pixels are deleted. Let N_i, $i = 1, \ldots, 8$, denote the ith neighbor of a pixel in counterclockwise order (see Fig. 2).

Algorithm **T2** compares the neighborhood of each pixel to several pre-defined 3×3 patterns and classifies each pixel as safe or deletable. A pixel is *safe* if its neighborhood matches one of the patterns shown in Fig. 2 where $\emptyset$ stands for either a black or white pixel. Each iteration of the algorithm consists of two scans. In the first one, every pixel that is not safe and for which either $N_4 = 0$ or $N_0 = 0$ is removed from the pattern. Similarly, in the second scan, every pixel that is not safe and for which either $N_2 = 0$ or $N_6 = 0$ is removed.

Algorithm **T3** deletes pixels by using the sum

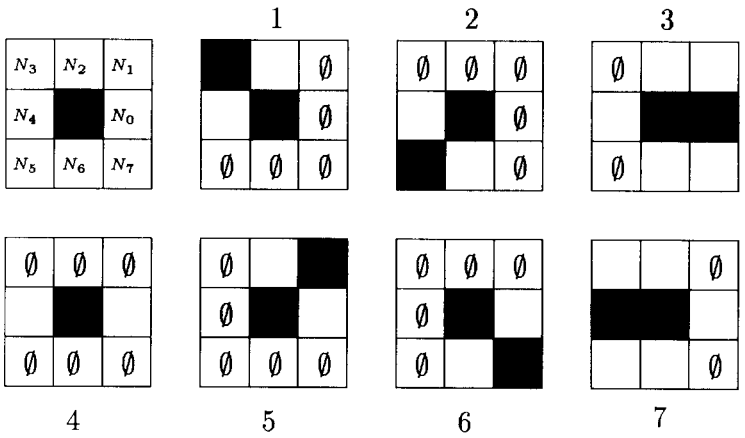

Fig. 2. The 3 × 3 patterns used in **T2**.

$\sum_{i=0}^{7} |N_{i+1} - N_i|$ where $N_8 \equiv N_0$ which is computed for each pixel and is called its crossing number. A black pixel with zero, two or more black neighbors and with a crossing number equal to zero or two (i.e., when there exists exactly one black component in its neighborhood) is deleted if its deletion does not break the connectivity of the pattern. A pixel is also deleted under additional conditions as specified in (Naccache and Shinghal, 1984).

Segmentation algorithms usually operate as follows: for every curve P between two non-regular pixels in the skeleton, they find a sequence of pixels which includes both endpoints of P such that the polygon approximation Q of P, formed by joining successive pixels by a straight segment, lies within a distance ε from P. Segmentation algorithms, which are also called data reduction (or vectorization) algorithms, differ in the distance score they use.

Algorithm **S2** uses a uniform norm as its distance score. The distance $d(p_j, q)$ between a pixel p_j on P and a segment $q = (p_i, p_k)$ from Q where p_j lies between pixels p_i and p_k is defined to be the Euclidean distance between p_j and q as long as p_j lies in a strip formed by drawing perpendicular lines at p_i and p_k and otherwise it is the Euclidean distance to the closest endpoint among p_i and p_k (Dunham, 1986). The deviation $e(q)$ of a segment $q = (p_i, p_k)$ from p_j is $\max_{i \leqslant j \leqslant k} d(p_j, (p_i, p_k))$ and the distance between P and Q is the largest deviation among the segments on Q, i.e., $D(P, Q) = \max_{q \in Q} e(q)$. For each curve between two non-regular pixels p_0 and p_u the algorithm finds a set of pixels such that the curve is approximated with the *minimal* number of segments. The algorithm achieves minimality by using the recursive formula

$$F(u) = 1 + \min_{v \in B} F(v)$$

where $B = \{v \mid 0 \leqslant v \leqslant u, e((p_v, p_u)) \leqslant \varepsilon\}$ and $F(u)$ is the minimal number of segments needed to approximate the path on P between p_0 and p_u. The set Q is found by going backwards from pixel p_n to a pixel p_v which realizes the minimum above for $u = n$, inserting p_v into Q, and continuing backtracking in this fashion until p_0 is placed into Q.

Algorithm **S3** determines Q by following the curve P from p_0 to p_n and generating segments as it progresses. Let p_i be the endpoint of the currently generated segment. The algorithm follows the curve P until a pixel p_k is reached such that at least one pixel on the path between p_i and p_k is distanced further than ε from the segment (p_i, p_k). When such a pixel is found, the algorithm generates a new segment (p_i, p_{k-1}).

Algorithm **S4** also determines Q by following the curve P from p_0 to p_n and generating segments as it progresses but it uses a different criterion than **S3**. Given p_i, the endpoint of the currently generated segment, it forms a *critical line* passing through p_i and p_{i+1}. This line estimates the tangent to P at p_i. The algorithm now follows the curve starting at p_{i+1} until a pixel p_k is found which is at a distance higher than ε from the critical line. Pixel p_{k-1} is inserted into Q. The algorithm stops when p_n is reached. We have used $\varepsilon = 2.0$ in our implementation of **S2**, **S3** and **S4**.

Finally, since the similarity score does not take into

Table 1
Results for the evaluation of the first 20 samples of *one* as compared to *one*, ..., *ten*

Sample	*one*	*two*	*three*	*four*	*five*	*six*	*seven*	*eight*	*nine*	*ten*
1	0.299	0.471	0.386	0.559	0.469	0.553	0.416	0.510	0.447	0.494
2	0.282	0.465	0.404	0.580	0.473	0.543	0.428	0.543	0.466	0.510
3	0.299	0.434	0.417	0.571	0.559	0.562	0.497	0.519	0.412	0.503
4	0.267	0.429	0.456	0.681	0.523	0.604	0.454	0.495	0.406	0.515
5	0.204	0.409	0.432	0.626	0.567	0.592	0.405	0.482	0.422	0.537
6	0.235	0.461	0.423	0.577	0.548	0.543	0.436	0.502	0.380	0.520
7	0.332	0.524	0.385	0.577	0.651	0.666	0.539	0.622	0.430	0.540
8	0.221	0.412	0.410	0.609	0.522	0.584	0.389	0.500	0.431	0.511
9	0.303	0.475	0.434	0.563	0.462	0.535	0.477	0.472	0.449	0.497
10	0.282	0.506	0.439	0.613	0.541	0.561	0.516	0.576	0.453	0.529
11	0.251	0.428	0.443	0.631	0.593	0.610	0.433	0.584	0.403	0.577
12	0.322	0.454	0.407	0.516	0.452	0.548	0.448	0.467	0.414	0.467
13	0.333	0.530	0.407	0.517	0.593	0.609	0.461	0.575	0.468	0.513
14	0.250	0.485	0.428	0.649	0.599	0.587	0.434	0.509	0.433	0.585
15	0.305	0.468	0.392	0.576	0.525	0.635	0.464	0.551	0.499	0.605
**16*	0.369	0.554	0.367	0.563	0.510	0.654	0.564	0.600	0.441	0.533
17	0.297	0.518	0.482	0.704	0.662	0.679	0.572	0.605	0.559	0.656
18	0.318	0.485	0.435	0.543	0.539	0.583	0.457	0.503	0.517	0.481
19	0.331	0.497	0.457	0.628	0.575	0.635	0.536	0.619	0.464	0.521
20	0.262	0.470	0.427	0.635	0.530	0.584	0.478	0.491	0.451	0.521

account segment lengths, we added a *split and stretch* stage before classification. Each segment with length higher than 16 units is cut into several segments as equally sized as possible and each having a length less than 16. To avoid excessive fragmentation, every pair of neighboring segments (sharing a common endpoint) is transformed into one segment by connecting their disjoint endpoints, if the angle between them is higher than $7\pi/8$ and the sum of their length is less than 16 units.

5. Experimental results

We implemented the system in C and ran it on a SUN SPARC workstation. The software consists of two main modules: an experimental tool for viewing and manually processing samples and a program for testing the performance of the similarity matching algorithm. The samples' pages were scanned on a DEST scanner using a resolution of 300 dpi.

A database of 600 off-line cursive script word images was built by asking a single person to write 60 samples of the following 10 words: *one*, *two*, *three*, ..., *ten*. Each group of 60 samples from the same word was divided into a *training group* of 15 samples and a *test group* of 45 samples. A session consists of a training and a testing stage. In the training stage, we have selected a template out of the 15 samples from each word as the representative of its class, by identifying the one which achieves a minimal sum of similarities when compared to the rest of its group. We then repeated this experiment with four additional writers. We have tested the algorithm by comparing each sample not used in the training stage with the 10 template models M_j found in the training stage. Let U_i, $i = 1, \ldots, 450$, be a sample from the testing stage. For each U_i we compute S_i^j, $j = 1, \ldots, 10$, representing the similarity between U_i and group j. Similarities were computed using a square window of size 60×60 pixels, and $\alpha = \beta = 0.5$. The identity of U_i is the identity of the most similar template.

Table 1 presents the results of testing 20 instances of the word *one*. This table reflects the variations among the similarity scores computed by the algorithm and is provided so that some raw data becomes available to the reader aside of summarized statistics. The overall results of the first experiment are summarized in Table 2, where **T** stands for the different thinning algorithms and **S** for the segmentation ones. The total recognition rates vary from 92.89% to 97.33% for the given database, depending on the chosen preprocessing algorithms. The data in this table suggests that these recognition rates could be vastly improved if the word

Table 2
Overall results for the primary test

T	Group	S1 (%)	S2 (%)	S3 (%)	S4 (%)
1	*one*	95.56	100.00	97.78	97.78
	two	62.22	71.11	91.11	88.89
	three	100.00	100.00	100.00	100.00
	four	95.56	100.00	97.78	95.56
	five	95.56	91.11	91.11	97.78
	six	93.33	95.56	95.56	91.11
	seven	97.78	97.78	100.00	100.00
	eight	100.00	100.00	100.00	100.00
	nine	100.00	100.00	100.00	100.00
	ten	100.00	97.78	100.00	100.00
	total	94.00	95.33	97.33	97.11
2	*one*	93.33	91.11	95.56	95.56
	two	57.78	73.33	82.22	88.89
	three	100.00	100.00	100.00	100.00
	four	100.00	100.00	100.00	100.00
	five	97.78	95.56	86.67	93.33
	six	97.78	93.33	95.56	91.11
	seven	97.78	100.00	97.78	100.00
	eight	100.00	100.00	100.00	100.00
	nine	100.00	95.56	100.00	100.00
	ten	100.00	95.56	97.78	100.00
	total	94.44	94.44	95.56	96.89
3	*one*	93.33	97.78	95.56	97.78
	two	84.44	82.22	84.44	55.56
	three	100.00	100.00	100.00	100.00
	four	100.00	100.00	100.00	97.78
	five	91.11	91.11	97.78	91.11
	six	93.33	88.89	91.11	88.89
	seven	97.78	100.00	97.78	100.00
	eight	100.00	100.00	100.00	100.00
	nine	100.00	97.78	100.00	100.00
	ten	100.00	97.78	100.00	97.78
	total	96.00	95.56	96.67	92.89

two is removed from the dictionary. Indeed when we removed it in a repeated experiment, the recognition rates were between 96.79% and 98.52%. The problem with recognizing the word *two* is mainly due to the fact that its model is often too close to either *three* or *ten*. This result suggests the self-evident observation that for this method to work well high variations among templates are needed. Note that the preprocessing steps have only a marginal influence on the recognition rates. We repeated this experiment with four additional writers fixing the preprocessing algorithms to be **T3** and **S3**. The recognition rates obtained were: 96.67%, 98.67%, 99.11%, 99.33% (the word *two* included).

Examples of the four writing styles are given in Fig. 3. These four words were correctly identified when compared to prestored models of the first writer. In fact all sevens were identified correctly because their model is rather long and unique. Generally, the system didn't perform well when a word written in one handwriting style was compared to prestored words created using another handwriting style.

In Fig. 4, we provide several examples of the word *two*. The upper line consists of the actual bitmaps and the lower line consists of the corresponding word-models. The first three columns contain prestored words. The forth column contains a wrong classification. The system has identified this word as a *ten* having a score of 0.32 while the second option was the correct one with a score of 0.36. All other options got a score higher than 0.4. The word in the fifth column was identified correctly with a score of 0.22 while the second option was ten with a score of 0.29. As one can see from this example, it is hard to guess a priori which words will be correctly identified.

To improve recognition rates we added a *rejection scheme* to the training stage of the classification process. After selecting the best template for each word, we evaluate the quality of the resulting templates and fix a threshold rejection value accordingly as follows. From the 150 models used in the training stage 10 models are selected as templates. We then test the 140 remaining models vs. the 10 templates, giving 10 similarity scores $S_i^{(j)}$ per model. Let g_i and h_i be the two minimal scores of model U_i, and let s_i be the standard deviation among the ten scores. We choose *not* to classify U_i if $(h_i - g_i)/s_i$ is smaller than a certain threshold value Θ. When this criterion is met, the minimal score is too close to the next best score, i.e. it is not distinct enough from alternative identities. Otherwise we give U_i the identity of g_i. The threshold value is repeatedly raised by increments of 0.001, starting with 0 (no rejection), until the overall recognition percentage of the 140 models surpasses 99%. The obtained Θ value is used in the testing stage for rejecting unknown samples using the same scheme.

Table 3 presents the rejection and recognition rates for different combinations of thinning and segmentation algorithms. The recognition rate is the percentage of correct identifications among the samples in the testing stage that have not been rejected. The Θ values learned in the training stage of each session are shown as well. Using **T2** and **S3** the recognition rate was 100%, however, the rejection rate was quite

Table 3
Overall results with rejection

T	Group	S1		S2		S3		S4	
		Rej (%)	Rec (%)	Rej (%)	Rec (%)	Rej (%)	Rec (%)	Rej (%)	Rec (%)
1	Θ	0.283		0.167		0.057		0.103	
	one	15.56	92.11	6.67	100.00	4.44	100.00	0.00	97.78
	two	44.44	72.00	46.67	87.50	6.67	92.86	20.00	91.67
	three	0.00	100.00	4.44	100.00	0.00	100.00	0.00	100.00
	four	8.89	100.00	0.00	100.00	2.22	97.73	2.22	97.73
	five	13.33	100.00	11.11	95.00	4.44	90.70	0.00	97.78
	six	6.67	95.24	4.44	97.67	0.00	95.56	4.44	93.02
	seven	20.00	100.00	2.22	97.72	2.22	100.00	2.22	100.00
	eight	6.67	100.00	0.00	100.00	0.00	100.00	0.00	100.00
	nine	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
	ten	0.00	100.00	0.00	97.78	0.00	100.00	0.00	100.00
	total	11.56	96.99	7.56	98.08	2.00	97.73	2.89	97.94
2	Θ	0.320		0.081		0.381		0.252	
	one	17.78	97.30	6.67	90.91	17.78	100.00	11.11	100.00
	two	42.22	80.77	46.67	75.00	51.11	100.00	44.44	100.00
	three	0.00	100.00	4.44	100.00	2.22	100.00	2.22	100.00
	four	8.89	100.00	0.00	100.00	8.89	100.00	4.44	100.00
	five	8.89	100.00	11.11	97.62	44.44	100.00	11.11	97.50
	six	4.44	97.67	4.44	95.45	4.44	100.00	6.67	97.62
	seven	24.44	100.00	2.22	100.00	26.67	100.00	6.67	100.00
	eight	2.22	100.00	0.00	100.00	13.33	100.00	13.33	100.00
	nine	8.89	100.00	0.00	100.00	22.22	100.00	17.78	100.00
	ten	2.22	100.00	0.00	100.00	8.89	100.00	11.11	100.00
	total	12.00	98.23	3.78	96.07	20.00	100.00	12.89	99.49
3	Θ	0.265		0.265		0.284		0.361	
	one	8.89	100.00	17.78	100.00	17.78	100.00	11.11	100.00
	two	40.00	90.62	60.00	94.44	51.11	92.59	60.00	77.78
	three	2.22	100.00	0.00	100.00	2.22	100.00	2.22	100.00
	four	11.11	100.00	6.67	100.00	8.89	100.00	13.33	100.00
	five	6.67	95.12	17.78	100.00	44.44	100.00	26.67	100.00
	six	11.11	95.35	2.22	90.91	4.44	100.00	13.33	100.00
	seven	8.89	100.00	8.89	100.00	26.67	100.00	20.00	100.00
	eight	8.89	100.00	0.00	100.00	13.33	100.00	8.89	100.00
	nine	2.22	100.00	26.67	100.00	22.22	100.00	4.44	100.00
	ten	0.00	100.00	2.22	100.00	2.22	100.00	4.44	100.00
	total	10.44	98.26	13.33	98.72	10.22	99.51	17.11	98.93

Table 4
Additional writers using **T3** and **S3**

Writer	1		2		3		4	
	Rej (%)	Rec (%)	Rej (%)	Rec (%)	Rej (%)	Rec (%)	Rej (%)	Rec (%)
Θ	0.000		0.070		0.227		0.265	
one	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
two	0.00	97.78	7.69	91.67	26.67	100.00	40.00	100.00
three	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
four	0.00	93.33	0.00	100.00	6.67	100.00	0.00	100.00
five	0.00	100.00	0.00	100.00	40.00	100.00	6.67	92.86
six	0.00	100.00	0.00	100.00	13.33	100.00	0.00	100.00
seven	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
eight	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
nine	0.00	95.56	0.00	100.00	0.00	93.33	6.67	100.00
ten	0.00	100.00	0.00	100.00	26.67	100.00	0.00	100.00
total	0.00	98.67	0.77	99.23	11.33	99.25	5.33	99.30

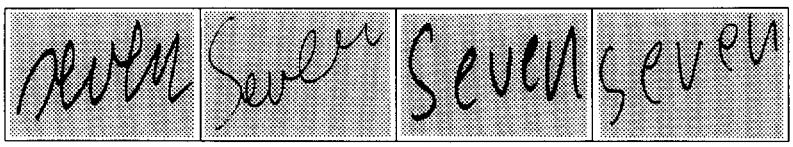

Fig. 3. Four different writers.

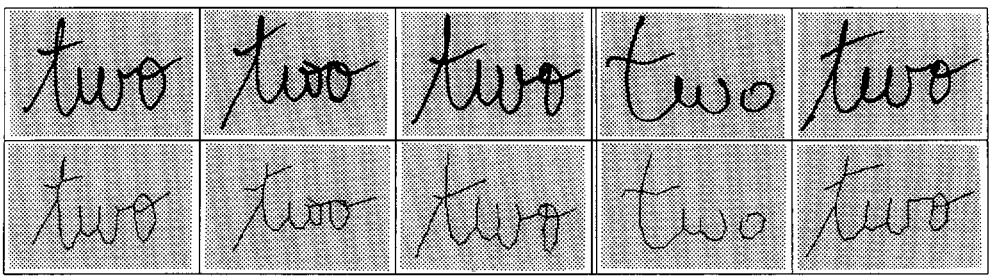

Fig. 4. Examples of the word *two* taken from a single writer.

Table 5
Results for Pascal words ($\Theta = 0$)

begin	100.00
end	100.00
for	97.78
if	97.78
program	100.00
readln	97.78
repeat	100.00
until	100.00
while	97.78
writeln	100.00
total	99.11

high (20%). Using **T3** and **S3** the recognition rate was 99.51% and the rejection rate was 10.22%. These two combinations of preprocessing algorithms yielded the best results. Nevertheless, the data indicates that the rejection and recognition rates are influenced only moderately by the choice of the preprocessing algorithms. We tested the system with four additional users employing **T3** and **S3** as the preprocessing algorithms and without changing its parameters (α, β, and w). Recognition rates varied between 98.67% and 99.3% and rejection rates varied between 0% and 11.33%. Table 4 presents the rejection and recognition rates for each user and for each word.

To verify the versatility of the recognition system we tested two additional dictionaries: a set of 10 Pascal's reserved words and a set of 20 Hebrew characters. Pascal words were found by the system to have quite distinct word models and so the chosen Θ parameter was 0. Furthermore, the algorithm made only one recognition error for each of the following words (for, if, readln, while) and no recognition error in all other words. The data in this experiment came from a single writer. When the Hebrew characters where tested, the recognition rates were quite high except for the character *daled* that resembles the character *tsadik*. However, rejection rates were unacceptably high since the word models of each character contained too few segments (usually less than 20) because they represent a

Table 6
Results for Hebrew letters ($\Theta = 0.363$)

aleph	2.22	100.00
bet	31.11	100.00
gimel	2.22	100.00
daled	95.56	0.00
he	82.22	87.50
zain	31.11	70.96
chet	2.22	100.00
tet	37.78	100.00
caf	33.33	100.00
lamed	31.11	100.00
mem	2.22	100.00
nun	15.56	100.00
samech	2.22	100.00
ayn	11.11	100.00
pe	11.11	100.00
tsadik	2.22	100.00
kuf	0.00	100.00
resh	97.79	100.00
shin	0.00	95.56
taf	2.22	100.00
total	20.39	99.03

Table 7
Errors out of 450 words as a function of α and β

β / α	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.0	2	3	3	4	6	6	7	9	12	18	26
0.1	2	3	5	6	10	11	16	19	25	28	25
0.2	2	2	3	6	7	11	14	19	23	27	25
0.3	2	2	3	6	6	11	14	17	20	23	22
0.4	2	2	4	5	6	9	12	16	18	21	19
0.5	2	2	3	4	6	6	9	12	17	20	14
0.6	2	2	2	4	6	6	6	10	13	19	13
0.7	2	2	2	4	5	6	6	7	11	14	10
0.8	2	1	2	2	4	5	6	6	7	9	11
0.9	2	1	2	2	2	4	5	5	7	9	11
1.0	2	1	1	2	2	2	2	2	2	7	12

single simple character rather than a full word. Thus, the recognition algorithm is more successful when the words to recognize contain sufficient number of segments (usually over 50) and they are sufficiently distinct of each other. These results are shown in Table 5 and Table 6.

Finally, we tested the relative importance of the neighborhood similarity vs. the individual similarity by repeatedly changing β and we tested the relative importance of segments' distance and segment's slope difference (which determine the individual similarities) by repeatedly changing α. In this experiment we used **T3** and **S3** as preprocessing algorithms and we used no rejection scheme. Table 7 shows the results. In particular note that best recognition rates were obtained when the similarity score is biased towards the neighborhood similarity ($\beta = 0.1$). In fact, the system works quite well if individual similarity is ignored all together. When β is set to 1, the data shows that segment's distance is the main factor in the quality of individual similarity and that the segments' slopes can almost be ignored.

Acknowledgment

The first author wishes to thank R. Bar-Yehuda for his advice during the early stages of this research.

References

Deutsch, E.S. (1972). Thinning algorithms on rectangular, hexagonal, and triangular arrays. *Comm. ACM* 15 (9), 827–837.

Dunham, J.G. (1986). Optimum uniform piecewise linear approximation of planar curves. *IEEE Trans. Pattern Anal. Mach. Intell.* 8 (1), 67–75.

Lee, H.-J. and B. Chen (1992). Recognition of handwritten Chinese characters via short line segments. *Pattern Recognition* 25, 543–552.

Mantas (1986). An overview of character recognition methodologies, *Pattern Recognition* 19, 425–430.

Naccache, N.J. and R. Shinghal (1984). SPTA: A proposed algorithm for thinning binary patterns. *IEEE Trans. Syst. Man Cybernet.* 14 (3), 409–418.

Rosenfeld, A. and A.C. Kak (1990). *Digital Picture Analysis*, Vol. 2, 231–233.

Reumann, K. and A.P.M. Witkam (1974). Optimizing curve segmentation in computer graphics. *Internat. Computer Symposium*, 467–472.

Sklansky, J. and V. Gonzalez (1980). Fast polygonal approximation of digitized curves. *Pattern Recognition* 12, 327–331.

Tappert, C.C., C.Y. Suen and T. Wakhara (1990). The state of the art in on-line handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 12 (8), 787–808.

Wagner, R.A. and M.J. Fischer (1974). The string to string correction problem, *J. ACM* 21 (1), 168–173.