A Lexicon Driven Approach to Handwritten Word Recognition for Real-Time Applications

Gyeonghwan Kim and Venu Govindaraju

Abstract—A fast method of handwritten word recognition suitable for real time applications is presented in this paper. Preprocessing, segmentation and feature extraction are implemented using a chain code representation of the word contour. Dynamic matching between characters of a lexicon entry and segment(s) of the input word image is used to rank the lexicon entries in order of best match. Variable duration for each character is defined and used during the matching. Experimental results prove that our approach using the variable duration outperforms the method using fixed duration in terms of both accuracy and speed. Speed of the entire recognition process is about 200 msec on a single SPARC-10 platform and the recognition accuracy is 96.8 percent are achieved for lexicon size of 10, on a database of postal words captured at 212 dpi.

Index Terms—Handwritten word recognition, segmentation algorithm, variable duration, chain code representation, dynamic programming.

1 Introduction

Unconstrained handwritten word recognition by a computer program is a challenging task. It has several applications such as reading addresses on mail pieces [1], [2], [3], reading amounts on bank checks [4], [5], extracting census data on forms [6], [7], reading address blocks on tax forms [8], and routing FAX messages. The challenge stems mainly from the wide variety of writing styles. Handwritten words can be classified into three categories: *cursive*, *hand-printed* and *mixed* (Fig. 1). Moreover, programs have to deal with image degradation caused by the transmission media, inaccurate digitization and lack of temporal information.

Since a word is essentially a sequence of characters, a natural approach to word recognition is to segment the word into characters and recognize the individual characters using optical character recognizers (OCR). In most applications it is reasonable to assume that a lexicon is provided. The lexicon can be either static or generated dynamically. The task of address interpretation on a mailpiece is an example of an application where the lexicon is generated dynamically. The ZIP code on the address provides all possible city names or street names as the lexicon. Different ZIP codes provide different lexicons [2]. An example of an application with static lexicon is the reading of amounts on bank checks. The lexicon is fixed in this case to about 40 words.

Segmentation and recognition of cursive script have been adequately described in the literature [4], [9], [10], [11], [12], [13]. The task of segmentation is integrally coupled to the recognition methodology and is analogous to segmenting continuous speech [14]. In the case of hand-printed script (Fig. 1a), segmentation is a relatively simple task. In the case of cursive script (Fig. 1b) and mixed script (Fig. 1c), however, segmentation is relatively hard. Several approaches have been explored. Typically, words are first classified into a category of their type and subsequently different schemes are applied depending on the script type [15].

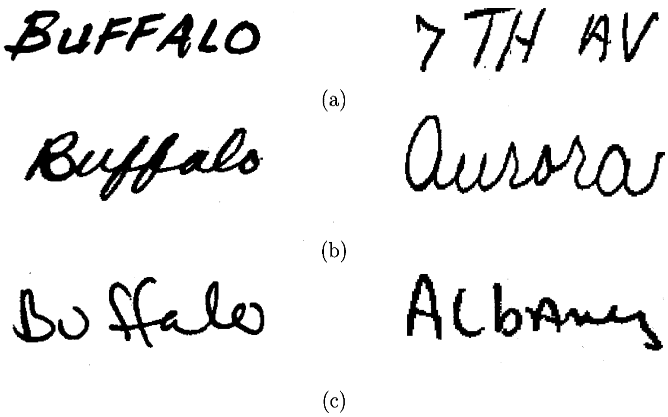


Fig. 1. Types of handwriting: (a) Hand-printed, alphanumeric. (b) Cursive. (c) Mixed.

In this paper, we present a method of word recognition that uses word models (as opposed to character models). The key concept underlying this approach is the early involvement of lexicon in the recognition process. The word image is compared with only words present in the lexicon thus eliminating any need for post processing (an essential step in traditional *segmentation-OCR-post processing paradigms*). Since lexicons are small in a majority of applications this is an attractive approach.

Also, the concept of variable duration, which is obtained from character segmentation statistics and used for determining the size of matching window during the recognition, is introduced in this paper. The variable duration

• *The authors are with the Center of Excellence for Document Analysis and Recognition (CEDAR), Department of Computer Science, State University of New York at Buffalo, 520 Lee Entrance, Amherst, New York 14228-2567. E-mail: gkim@cedar.buffalo.edu.*

Manuscript received June 15, 1995; revised Feb. 11, 1997. Recommended for acceptance by M. Mohiuddin.
For information on obtaining reprints of this article, please send e-mail to: transpami@computer.org, and reference IEEECS Log Number104068.0.

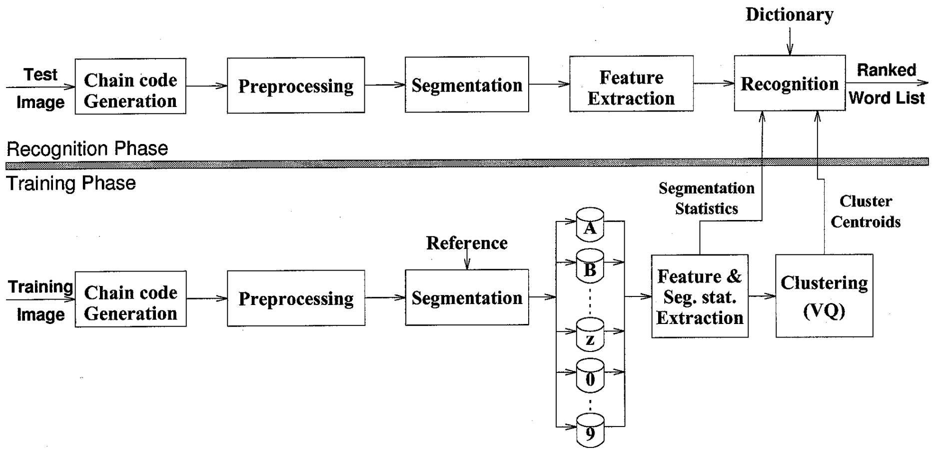


Fig. 2. Word recognition methodology.

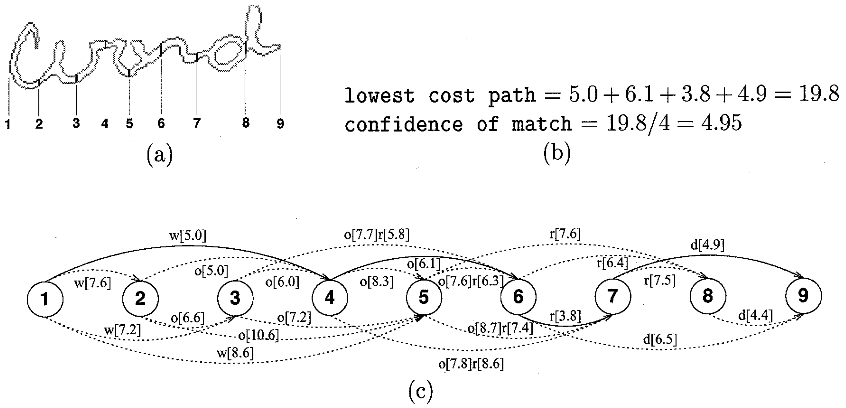


Fig. 3. Matching between a sample image and one lexicon entry "word." (a) Segmentation points. (b) Confidence of match. (c) Matching paths and confidences.

maximizes the efficiency of the lexicon driven approaches of the handwritten word recognition in terms of both speed as well as recognition accuracy.

In addition, we present a robust image handling scheme for all types of script. Chain code representation of contours of word images is used for efficient image processing.

Section 2 describes the overall methodology of the recognition system. Section 3 describes the chain code representation, and preprocessing operations. Section 4 describes segmentation strategies for partitioning a word image into characters. Section 5 describes feature extraction. Section 6 outlines the recognition and training strategies. Section 7 is about experiments and results. Section 8 provides a summary of the work presented in this paper.

2 Methodology

The objective is to develop a fast handwritten word recognition system for real time applications that accepts all types of script described in Fig. 1. Fig. 2 illustrates the methodology that has been developed. There are two processing phases: *training* and *recognition*. Input images go through the steps of chain code generation, preprocessing, segmentation and feature extraction in both phases.

1) *Chain code generation* step converts the binary image input into a chain code representation by coding the boundary contours of components in the image while preserving the positional and directional information of adjacent pixels [16]. An array is defined for efficient representation and manipulation of data. Single pixel components (speckle noise) are detected and removed. Subsequent image handling steps work with chain code data.
2) *Preprocessing* step includes noise removal, slant correction and smoothing [9], [17]. Noise introduced by digitizing devices and transmission media, is eliminated by comparing the size of connected components with an estimate of average stroke width. Slant angle is estimated by averaging orientation angles of "vertical" strokes and shifting the x-coordinates of components accordingly. Smoothing removes jaggedness of the contours (some introduced during slant correction).
3) *Segmentation* step returns the segmentation points to be used for grouping one or more segment(s) to form meaningful characters. The segmentation points are determined using a combination of ligatures and concavity features on the contour. Average stroke width

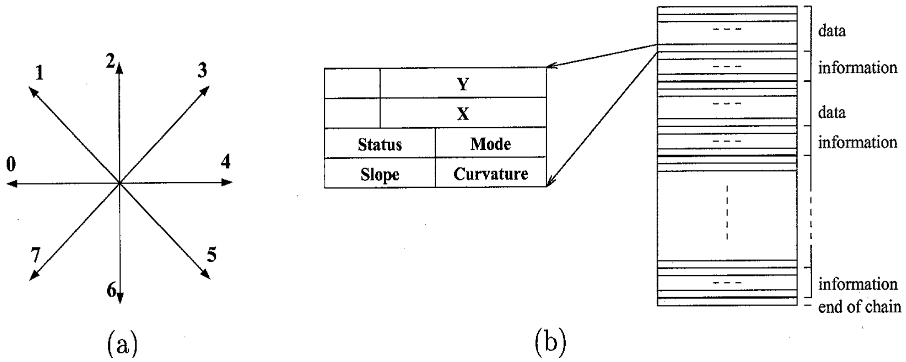


Fig. 4. Chain code representation. (a) Slope. (b) Structure.

of an image is estimated and used in an adaptive fashion to determine the features. The number of segmentation points is kept to a minimum while ensuring that a segmentation point exists to split touching characters and the maximum number of segmentation points per character is four.

4) *Feature extraction* step generates feature vectors for combination of segments that are hypothesized to be characters. Global and local features are defined and extracted from 3×3 subimages of the segment(s) represented by their chain code.
5) *Training phase* uses a training set consisting of binary word images different from the test set. The reference of character segmentation points in the input image are determined manually. Segmentation statistics such as how often a particular character is split into how many segments (1 … 4) is stored. Feature vectors are extracted for the manually referenced character segments and used by a clustering algorithm to find cluster centroids of characters to be used in recognition.
6) *Recognition phase* uses segmentation statistics, character cluster centroids, a dictionary, and feature vectors derived from the test image. A dynamic matching scheme is used to compare features of a segment or a combination of consecutive segments with the cluster of centroids of a character in a lexicon entry. This procedure is used to rank the lexicon entries. Fig. 3 illustrates the recognition procedure. A sample word image, "*word*," is split by the segmentation algorithm (Fig. 3a). The example illustrates the matching scheme between the image of Fig. 3a and one lexicon entry "*word*." In Fig. 3c, arcs between adjacent nodes represent the possibility of grouping the image segments (Fig. 3a) between those segmentation points as a character hypothesis. For example, the arc from node 2 to node 5 represents the confidence of match between cluster centroid of "*o*" (from training) and the feature vector extracted from the point two to five in the image. The idea of taking the comparison range limiting, which is based on the statistics obtained during the training phase, is introduced to avoid unnecessary computation during the recognition phase.

3 PREPROCESSING

We have adopted the chain code method of image representation [18], [19] which allows a compact representation and reduction of data and hence processing time [20]. Chain code is a linear structure that results from quantization of the trajectory traced by the centers of adjacent boundary elements in an image array. Each data node in the structure represents one of eight grid nodes that surround the previous data node. Fig. 4a shows the slope convention used.

For efficient manipulation, certain properties of the chain code contour are stored in an array (Fig. 4b). *Data* fields in the array contain positional and slope information of each component of the traced contour. Properties stored in the *information* field are: The coordinates of bounding box of the contour, number of components in the corresponding data fields, area of the closed contour, and a flag which indicates whether the contour is interior or exterior. Chain code representation of an entire image consists of n such contour arrays cascaded, where n is the number of closed contours. Properties of a component are defined by assigning different numbers in the status and mode fields. Selection of a contour is achieved by searching the information field.

Normalization operations are performed to adjust skew,

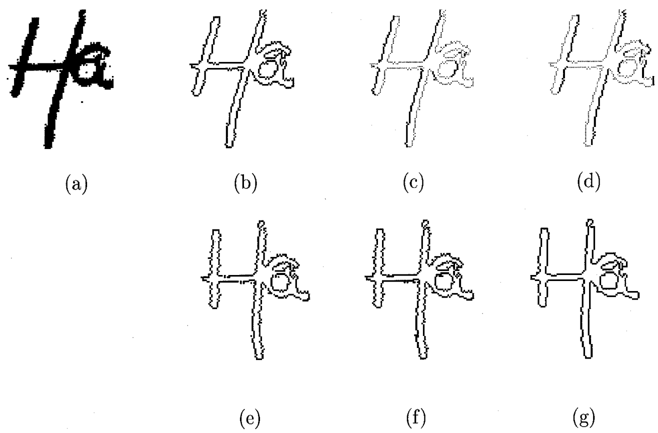


Fig. 5. Slant normalization using chain code. (a) Original image. (b) Chain code. (c) and (d) Downward and upward vertical lines, respectively (counterclockwise tracing). (e) Slant correction based on the angle estimated from lines in (c) and (d). (f) Connecting broken chain code. (g) Result of smoothing.

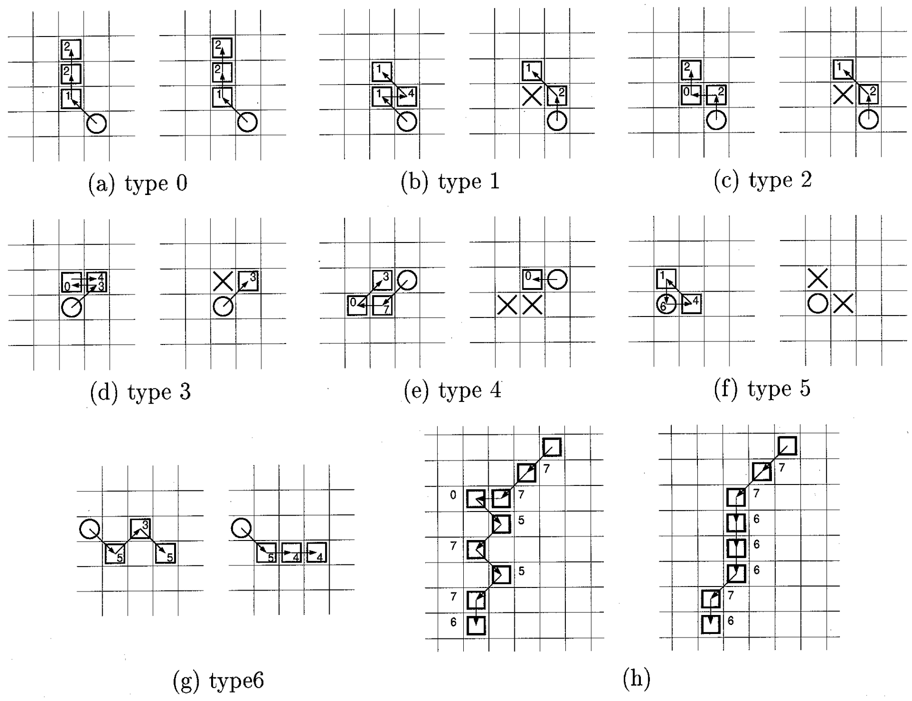


Fig. 6. (a - g) Types of smoothing operations—for each type, left sequence turns into right sequence after the smoothing operation (× represents elimination of the components and ○ represents starting point). (h) Shows how the algorithm is working for a sequence of components.

slant, stroke width, and size [10], [11], [17]. Digital capture of images can introduce noise from scanning devices and transmission media. Furthermore, script often has slant and skew which needs to be corrected before segmentation and recognition can begin. Smoothing operations are used to eliminate the artifacts introduced during image capture and slant normalization.

This section describes three preprocessing stages applied to chain code data:

- slant angle correction,
- smoothing, and
- average stroke width computation.

3.1 Slant Angle Correction

Vertical and near vertical lines are extracted by tracing chain code components using a pair of one dimensional filters. Because the chain code has one dimensional structure, vertical lines having opposite directions are extracted using the filters. Each filter is a five element array of different weights as shown in Algorithm 3.1. A convolution operation between the filter and five consecutive components is applied iteratively by sliding the filter one component at a time.

Coordinates of the start and end points of each vertical line extracted provides the slant angle. Global slant angle is the average of all the angles of the lines, weighted by their length in the vertical direction since the longer line gives more accurate angle than the shorter one. Fig. 5c and d represent upward and downward vertical lines detected by the algorithm.

Algorithm 3.1.

```
v_up[8] = {2, 1, 0, 1, 2, 3, 4, 3}
v_down[8] = {2, 3, 4, 3, 2, 1, 0, 1}
for i = strat_component
to end_component step 1
      sum_slope_up = 0;
      sum_slope_down = 0;
      for j = i - 2 to i + 2 step 1
            sum_slope_up = sum_slope_up
            + v_up[j.slope];
            sum_slope_down = sum_slope_down
            + v_down[j.slope];
      end
      if sum_slope_up < THRESH_V
      then i.status = MARK_UP fi
      if sum_slope_down < THRESH_V
      then i.status = MARK_DN fi
end
```

Tangent of the estimated global slant angle (θ) is used to correct for slant. Equation (1) shows the adjustment made to the x-coordinates.

$$
\begin{aligned}
x_n &= x - y\tan(\theta) \\
y_n &= y
\end{aligned} \tag{1}
$$

Quantization error incurred during the correction process introduces “jigs” and broken points in the horizontal direction (Fig. 5e). An interpolation method is applied to connect the broken points (Fig. 5f). The slant normalization algorithm assumes that the slant angle is between –45° and 45°.

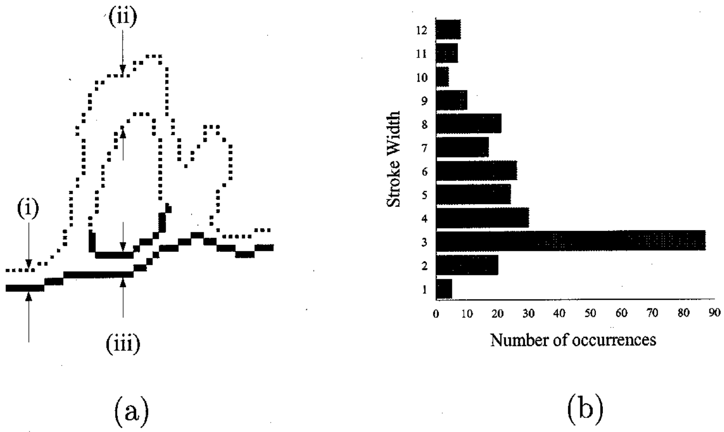


Fig. 7. Estimating average stroke width. (a) Measuring *y*-distances for each *x*-coordinate. (b) Analyzing occurrences of stroke widths.

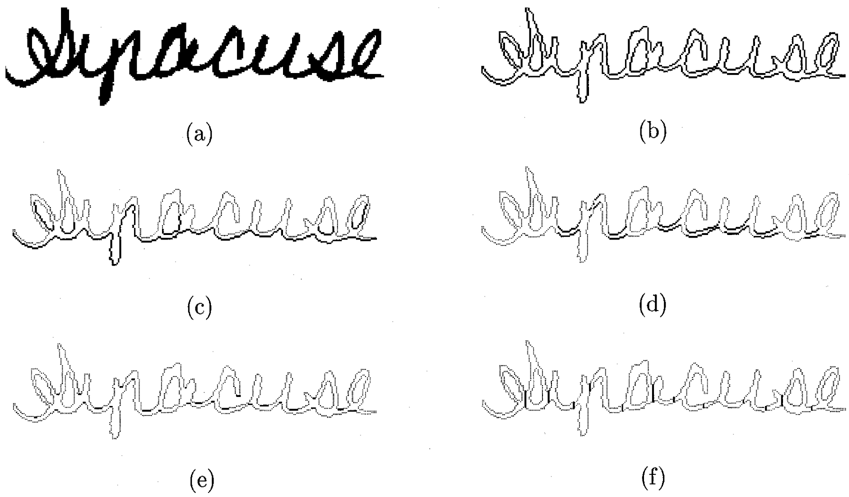


Fig. 8. Segmentation. (a) Original test image. (b) Chain code representation after slant normalization. (c) Splitting upper and lower contours. (d) Ligatures based on the average stroke width. (e) Concavities/convexities. (f) Segmentation points.

3.2 Smoothing

Smoothing includes elimination of small blobs on the contour and illegal combinations of chain code slopes. A sliding three-component one dimensional window is applied over all components. All combinations of three adjacent chain code slopes are analyzed and classified into only seven types based on the treatment warranted.

- Type 0: Needs no correction,
- Type 1: Remove first component and adjust slope of second component,
- Type 2: Remove second component and adjust slope of third component,
- Type 3: Remove second and third components,
- Type 4: Remove first and second components and adjust slope of third component,
- Type 5: Remove all three components, and
- Type 6: Adjust slopes of second and third component.

Fig. 6 shows examples of each case. Depending on the type, components can be removed from the chain code structure, and coordinates and slopes of components can be updated to maintain the relation between adjacent pixels. Fig. 6h shows components and the corresponding slopes of a sequence of components before (left) and after (right) applying the algorithm. Table 1 describes steps in the smoothing process.

TABLE 1
STEPS OF SMOOTHING PROCEDURES FOR FIG. 6h

slopes	type	action
$\boxed{770}57576$	0	no modification is needed
$7\boxed{705}7576$	2	2nd component is removed and slope of 3rd component is updated to 6
$7\boxed{767}576$	0	no modification is needed (type classification is performed using the updated component and the following component)
$77\boxed{675}76$	6	slopes of 2nd and 3rd components need to be updated to 6
$776\boxed{667}6$	0	no modification is needed. (type classification is performed based on the updated slopes)
$7766\boxed{676}$	0	no modification is needed

Box represents window for classification.

3.3 Computation of Average Stroke Width

Stroke width can vary locally depending on writing devices and paper within a script. It is reasonable, therefore, to talk of average stroke width and it can be used for the other image processing procedures in adaptive manner (e.g., detection of small components as noise). To estimate average stroke width, chain code contours of a word image are divided horizontally (Fig. 8c). By tracing contours from the left-most point to the right-most point, following distances are computed for each *x*-coordinate (Fig. 7a):

1) distance between upper and lower trace of the outer contour
2) distance between the upper trace of the inner contour and upper trace of the outer contour
3) distance between lower trace of the inner contour and lower trace of the outer contour

The histogram in Fig. 7b shows the number of occurrences of each distance value for the image of Fig. 8. The peak of the histogram gives a good estimate of the average stroke width (2).

$$N_d = \text{number of occurrences of distance value } d$$
$$P_d = \arg\max_d (N_d) \qquad (2)$$

To account for the fluctuation in pen movement, actual average stroke width is estimated by analyzing the shape of the histogram (Algorithm 3.2). For example, if the shape is dull, then the actual stroke width is greater than P_d by considering neighbor stroke widths until the condition is satisfied.

Algorithm 3.2.

```
d = P_d;
while (d < MAX_STROKE_WIDTH ∧ N_d > P_d/2)
  d = d + 1;
end
actual_stroke_width = d + 1;
```

4 Segmentation

A segmentation algorithm should be general and robust to handle various styles of writings and thickness of strokes. We make the following assumptions:

1) the number of segments per character must be at most four, and
2) all touching characters should be separated.

Handwriting is generated from the movement of a pen from left to right along an axis in the horizontal direction, giving rise to what are called ligatures. The ligatures are strong candidates for segmentation points in cursive scripts. Ligatures are extracted as follows. If the distance between *y*-coordinates of the upper half and lower half of the outer contour for a *x*-coordinate is less than or equal to the average stroke width, then the *x*-coordinate is marked as an element of a ligature. This procedure is repeated for subsequent *x*-coordinates. Ligatures in the extremities are eliminated to reduce the number of potential segmentation points. Fig. 8d shows ligatures obtained for an example word image.

On the other hand, segmentation points between discretely written (as opposed to cursive) touching characters cannot be hypothesized by ligatures alone. Alternatively, concavity features in the upper contour and convexities in the lower contour are used in conjunction with ligatures (Fig. 8e). Heuristics are applied to reduce the number of potential segmentation points (e.g., if ligatures and concavity features overlap, concavity features are ignored, and if a concavity and a convexity are overlapped in a *x*-coordinate, a segmentation point is assigned in the *x*-coordinate). The final segmentation points are stored with the corresponding segment information which consists of a number of components such as indices of related contours and coordinates of the bounding box of the segment.

Segmentation statistics of each character represent the possible ways in which a training character image can be split into segments. Equation (3) gives the computational form for computing the duration probability (*dur*(*j, i*)).

$$dur(j, i) = \Pr(j|K_i) = \frac{\text{No. of times that training character } (K_i) \text{ is segmented into j segments}}{\text{no. of training characters of } (K_i)} \qquad (3)$$

where K_i = `0, 1, ⋯,9, A(a), B(b), …, Z(z)`, $i = 1, 2, \ldots, 36$, and $j = 1, 2, 3, 4$. Table 2 shows the segmentation statistics which obtained by applying our segmentation scheme to training word images. It can be seen that almost 82 percent of characters of `0 (zero)` are not split, and 18 percent of them are split into two.

Our assumption allows at most four segments per character during the segmentation phase. However, statistics reveal that the upper limit of four segments is valid for only a few characters (such as "*m*" and "*w*"), and that for most characters it is less than four. In contrast to [21], where the statistics are used as the transition probability between character segments, this information is used to advantage in speeding up the matching step of the recognition phase and improving recognition accuracy. Confusions are reduced by matching within the window size controlled by the statistics, number of characters in a lexicon entry, and number of segments of the word image.

5 Features

We have designed a feature extractor that converts chain code images into feature vectors in a simple and fast manner. The same feature extraction procedure is used in both the training and recognition phases as shown in Fig. 2. In the training phase, extracted features are provided to a clustering procedure so that patterns of similar shapes can be represented by a code word of the trained code book [22]. Also segmentation statistics are obtained during the training phase. In the recognition phase, features of segment(s) of a test image are compared to the code words to find the best match.

5.1 Chain Code Features

Seventy-four chain code based features are used. Two global features—aspect ratio and stroke ratio of the entire template (a segment or a combined segment)—are used (4). Each segment is divided into nine subimages (3×3) and local features are collected from each subimage. Distribution of the eight directional slopes for each sub-image form the 72 local feature vectors ($8 \times 3 \times 3$) as shown in (5).

TABLE 2
SEGMENTATION STATISTICS

	Number of Segments					Number of Segments			
	1	2	3	4		1	2	3	4
0	0.81646	0.17655	0.00662	0.00000	*i*	0.94677	0.05015	0.00307	0.00000
1	0.99882	0.00118	0.00000	0.00000	*j*	0.54902	0.31372	0.13725	0.00000
2	0.36199	0.58329	0.05080	0.00390	k	0.18085	0.62234	0.17819	0.01861
3	0.62680	0.35355	0.01848	0.00115	l	0.86100	0.12050	0.01800	0.00050
4	0.09486	0.77207	0.12768	0.00477	m	0.00000	0.08317	0.70406	0.21276
5	0.44688	0.43248	0.11035	0.00891	n	0.03304	0.75373	0.20456	0.00865
6	0.07897	0.90418	0.01684	0.00000	o	0.77521	0.20979	0.01498	0.00000
7	0.63985	0.32673	0.03341	0.00000	p	0.42901	0.50925	0.05864	0.00308
8	0.75980	0.23124	0.00825	0.00068	q	0.55555	0.22222	0.22222	0.00000
9	0.75553	0.23855	0.00517	0.00073	r	0.64878	0.31138	0.03577	0.00406
a	0.50623	0.44725	0.04574	0.00075	s	0.66912	0.28571	0.04331	0.00184
b	0.36559	0.44802	0.16845	0.01792	t	0.69116	0.27010	0.03078	0.00794
c	0.83636	0.14965	0.01398	0.00000	u	0.01084	0.95664	0.03116	0.00135
d	0.35555	0.52888	0.10666	0.00888	v	0.05166	0.89298	0.05535	0.00000
e	0.79811	0.17643	0.02378	0.00165	w	0.00000	0.02443	0.86278	0.11278
f	0.61696	0.32432	0.05312	0.00559	x	0.18650	0.61904	0.18650	0.00793
g	0.39736	0.50789	0.08157	0.01315	y	0.08144	0.78733	0.13122	0.00000
h	0.11460	0.68576	0.16081	0.03881	z	0.44444	0.44444	0.11111	0.00000

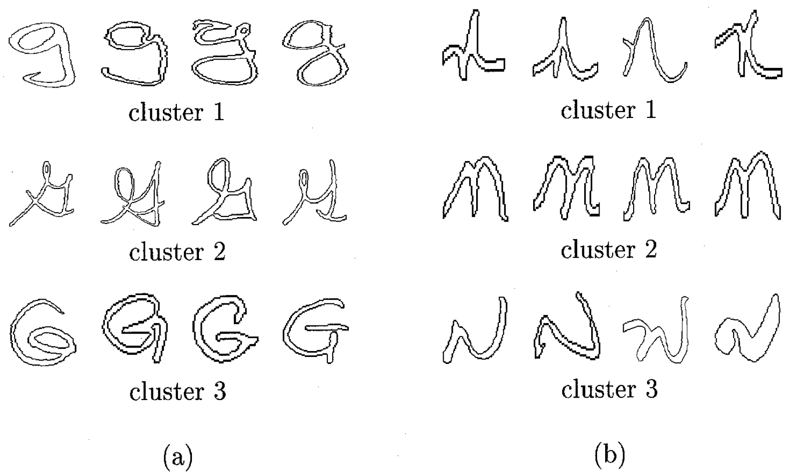


Fig. 9. Samples of clustering results. (a) Character **G/g**. (b) Character **N/n**.

- **Global Features**

$$F_{g_i} = \text{sigmoid}\left(\frac{H_i - V_i}{V_i}\right) \quad \text{for } i = 1, 2 \tag{4}$$

where
$H_1 = X_{max} - X_{min}$, $V_1 = Y_{max} - Y_{min}$ for aspect ratio
$H_2 = N_{horizontal\_stroke}$, $V_2 = N_{vertical\_stroke}$ for stroke ratio

- **Local Features**

$$F_{l_{ij}} = \frac{s_{ij}}{N_i S_j} \quad \text{for } i = 1, 2, \ldots, 9 \text{ and } j = 0, 1, \ldots, 7 \tag{5}$$

where s_{ij} = number of components with slope j from subimage i,

N_i = number of components from subimage i and

$$S_j = \max_i \left(\frac{s_{ij}}{N_i}\right)$$

5.2 Clustering Training Data

To provide reference feature vectors in the matching procedure, we use a clustering method based on character level data and build a code book. The code words are trained using 21,054 character images which are extracted from

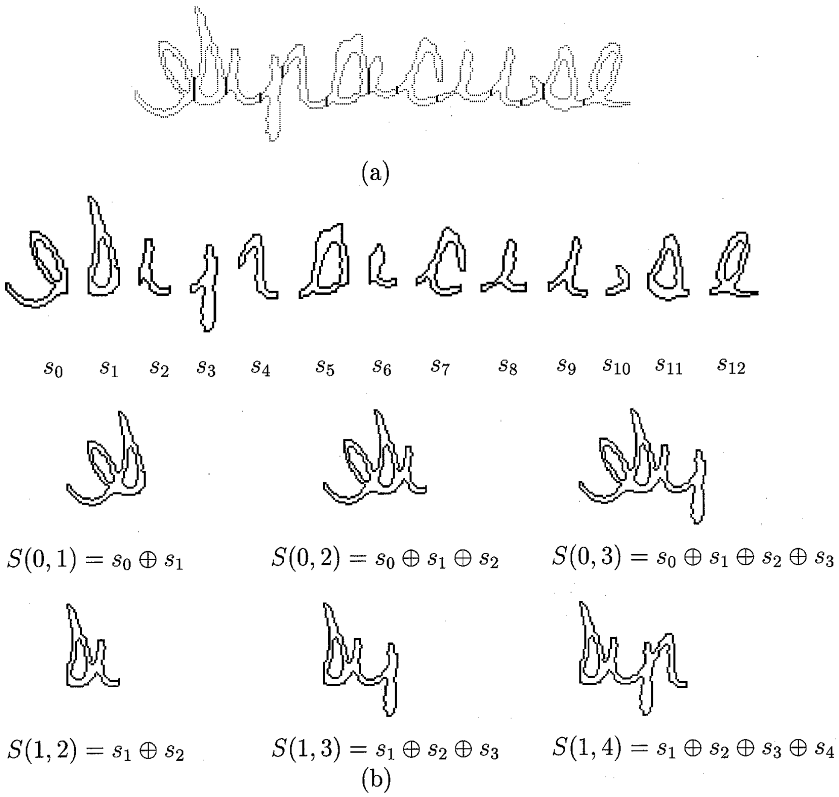


Fig. 10. Splitting and merging of segments. (a) Segmentation result. (b) Examples of splitting and merging and corresponding notations.

handwritten images of U.S. city names using the same segmentation method used in recognition. K-means clustering algorithm with a fixed signal-to-noise ratio and fixed maximum number of clusters is used. Fig. 9 shows samples of the clustering results for character **G**/**g** and **N**/**n**. The clustering algorithm gathers characters which have similar shapes in terms of features and feature vectors of the clustered characters form code words to represent the cluster. A table containing all code words of characters in the code book is used in the recognition phase.

6 Recognition

The objective is to find the best match between a word in the lexicon and the image. Depending on the metric used for classification and the method of lexicon handling, various matching schemes are possible. A dynamic matching approach which has been extensively studied by the speech recognition community [23], [24], [25] is used. Instead of passing on combination of segments to a generic OCR, the lexicon is brought into play early in the process. A combination of adjacent segments (up to a maximum of four based on our segmentation criteria) are compared to only those character choices which are possible at the position in the lexicon entry being considered.

In the matching procedure, comparisons between feature vectors of several possible combinations of segments and reference feature vectors of code words are made to find the best match. Since code words are trained at the character level and a character can be composed of up to four segments, a segment or a combination of segments is compared to the code words of reference characters within a permissible window in the first phase of the matching. For each match, the minimum distance value of each comparison is retained. In the second phase of the match, a global optimum path is obtained by using dynamic programming based on the saved minimum distances of the first matching phase.

Let an input word image be denoted as

$$T = \left\{ s_0, s_1, \ldots, s_{S_N - 1} \right\} \tag{6}$$

in which S_N is the number of segments in the input image and s_k represents the kth segment. Let the lexicon entries be referred to as R_i.

$$R_i = \{c_i(0), c_i(1), \ldots, c_i(C_N(i) - 1)\} \quad 1 \le i \le L_N \tag{7}$$

where $C_N(i)$ is the number of characters in the ith lexicon entry, L_N is the number of lexicon entries, and $c_i(j)$ represents the jth character of the ith lexicon entry. Let us define the notion of merged segments as follows:

$$\begin{aligned} S(b,e) &= \left\{ s_b \oplus s_{b+1} \oplus \ldots \oplus s_e \right\} \\ &= \left\{ s_v \right\}_{v=b}^{e} \qquad \text{for } 0 \le b < S_N,\ 0 \le e < S_N, \text{ and } b \le e \end{aligned} \tag{8}$$

where b is a beginning segment and e is an ending segment, and $\oplus$ is the merging operator (Fig. 10).

From the training data of clusters, code words of all characters, $\mathbf{C}_m$, can be obtained:

$$\mathbf{C}_m = \left\{ \mathbf{x}_m(i) \right\}_{i=1}^{N_m} \tag{9}$$

where m = `0, ⋯, 9, a, ⋯,z`, N_m is number of code words of character m, and $\mathbf{x}_m(i)$ represents the ith code word of character *m*.

Therefore, the basic problem of matching can be described as follows: For a given input word image, *T*, find a lexicon entry, R_i, which has the minimum distance from *T*:

$$D^* = \min_{1 \le i \le L_N} D(R_i, T)$$
$$R^* = \arg\min_{R} D(R_i, T) \qquad (10)$$

First, matching between each character of a given lexicon entry ($c_i(j)$, $0 \le j < C_N(i)$) and an arbitrary portion of the segment(s), specified by *b* and *e*, of the test input *T* is performed.

$$\hat{D}(v, b, e) = d\left(\mathbf{F}(S(b, e)), \mathbf{x}_{c_i(j)}(v)\right) \quad \text{for } 0 \le v < N_{c_i(j)} \qquad (11)$$

where d(·, ·) is distance between two feature vector and **F**(·) represents feature extraction operation for segment(s). Equation (11) gives the distance between each code word of character $c_i(j)$ and all possible pairs of test segments starting at *b* and ending at *e*. The best match for all *v* is obtained by (12).

$$\overline{D}(b, e) = \min_{1 \le v \le N_{c_i(j)}} \left[\hat{D}(v, b, e)\right] \qquad (12)$$

For meaningful alignment of segment(s) against a character of a lexicon entry and minimization of computational complexity, we apply constraints on the segmentation criteria, number of characters in a lexicon entry, and number of segments in a test image. These factors together determine the size of the permissible matching window. Accordingly, the range of ending segments for $c_i(j)$ is decided as follows.

$$e^j_{max} = \min\left(S_N - (C_N(i) - j), (j+1) \cdot M_d - 1\right)$$
$$e^j_{min} = \max\left(S_N - (C_N(i) - (j+1)) \cdot M_d - 1, j\right) \quad \text{for } j = 1, \ldots, C_N(i) - 1 \qquad (13)$$

where M_d is the maximum number of segments per character, *i*.e., four.

The range is determined according to our segmentation assumptions: A character in R_i consumes at least one segment and a character can be segmented as many as M_d segments. The interval between e^j_{min} and e^j_{max} is used to determine the matching window size for the reference character $c_i(j)$. For a given range of ending segments, a starting segment, *b*, is determined based on the notion that *b* should be bigger than e_{min} of the previous character and at most M_d segments can be combined for the match. Table 3 shows the computation results of $\overline{D}(b, e)$ in (12) for the example word image in Fig. 10. The horizontal axis represents the ending segment number and the vertical axis represents the *j*th character of a lexicon entry. Four rows for each character are shown representing the duration of segments, one to four, from bottom to top. For example, a component in third column (ending segment number of two) and second row (duration of two) of second character (*j* = 1, **y** in this particular example) represents $\overline{D}(1, 2) = 12.41$. Similarly, the component with ending segment number of 5, *j* = 2(**r**) and duration of four represents $\overline{D}(2, 5) = 11.59$. Equations (11) and (12) are computed within the matching window, thus significantly reducing the computational complexity.

In addition to the limiting matching ranges, early rejection of some lexicon entries is a big advantage of this lexicon driven scheme. If a lexicon entry R_i satisfies one of the following conditions in (14), R_i is rejected before the first stage matching is performed.

$$S_N > M_d \cdot C_N(i)$$
$$S_N < C_N(i) \qquad (14)$$

In contrast to other lexicon reduction schemes, the rejection does not affect recognition accuracy. Furthermore, if a part of a word segment has been compared with a particular character in a specific position in a lexicon entry, future comparison with the same character in other lexicon entries is avoided by reusing all the matching results.

In the second stage, the individual matching scores of the first stage are combined to compute the accumulated cost (distance) over the entire lexicon entry. This is accomplished by using dynamic programming. Distance of the best path ending at segment *e* for each *j*th character is computed by (15).

$$\overline{D}_j(e) = \min_{1 \le b \le e}\left[\overline{D}(b, e) + \overline{D}_{j-1}(b-1)\right] \qquad (15)$$

Based on the recursion of (15), we can formulate the second stage of this dynamic matching procedure for determining the overall best path as follows.

- Step 1: Initialization

$$\overline{D}_0(e) = \overline{D}(0, e) \quad 0 \le e \le M_d - 1 \qquad (16)$$

- Step 2: Recursion

$$\begin{aligned}
\overline{D}_1(e) &= \min_b\left[\overline{D}(b, e) + \overline{D}_0(b-1)\right] & e^1_{min} \le e \le e^1_{max} \\
\overline{D}_2(e) &= \min_b\left[\overline{D}(b, e) + \overline{D}_1(b-1)\right] & e^2_{min} \le e \le e^2_{max} \\
&\ldots \\
\overline{D}_j(e) &= \min_b\left[\overline{D}(b, e) + \overline{D}_{j-1}(b-1)\right] & e^j_{min} \le e \le e^j_{max} \\
&\ldots \\
\overline{D}_{C_{N-1}}(e) &= \min_b\left[\overline{D}(b, e) + \overline{D}_{C_N-2}(b-1)\right] & e^{C_N-1}_{min} \le e \le e^{C_N-1}_{max}
\end{aligned} \qquad (17)$$

- Step 3: Final Minimum Distance

$$D^* - \overline{D}_{C_N-1}(S_N - 1) \qquad (18)$$

Table 4 shows the computation results of dynamic programming based on the first stage matching shown in Table 3.

Further minimization of computational complexity is achieved by using the statistics of Table 2. We have assumed that a character can be split into at most four segments. However, according to the segmentation statistics shown in Table 2, most characters are composed of less than four segments. Furthermore, the frequently used characters, such as "*i*" and "*l*," are split into less than three segments 98 percent of the time. Table 5 shows the maximum number of segments per character used in the system (Table 2). M_d used in (13) is now replaced by the variable duration shown in Table 5. Because the variable duration of each character is less than or equal (for only "*m*" and "*w*") to M_d, the matching window size is further limited, hence increasing speed. Therefore, more lexicon entries are rejected by the modified conditions of (14). It should be noted

TABLE 3
FIRST STAGE OF MATCHING PROCEDURE

		Ending Segment												
j	d	0	1	2	3	4	5	6	7	8	9	10	11	12
0 (s)	4	0.00	0.00	0.00	7.72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	0.00	0.00	8.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	2	0.00	7.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	1	5.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1 (y)	4	0.00	0.00	0.00	0.00	9.73	11.59	9.67	0.00	0.00	0.00	0.00	0.00	0.00
	3	0.00	0.00	0.00	10.99	10.07	9.64	9.30	0.00	0.00	0.00	0.00	0.00	0.00
	2	0.00	0.00	12.41	4.75	10.69	10.22	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	1	0.00	10.33	10.05	7.51	10.73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2 (r)	4	0.00	0.00	0.00	0.00	0.00	10.48	8.52	10.26	0.00	0.00	0.00	0.00	0.00
	3	0.00	0.00	0.00	0.00	11.41	9.39	8.08	6.94	0.00	0.00	0.00	0.00	0.00
	2	0.00	0.00	0.00	9.29	9.28	7.56	7.40	7.08	0.00	0.00	0.00	0.00	0.00
	1	0.00	0.00	4.07	6.01	4.96	6.72	3.82	5.62	0.00	0.00	0.00	0.00	0.00
3 (a)	4	0.00	0.00	0.00	0.00	0.00	0.00	7.55	7.54	6.43	0.00	0.00	0.00	0.00
	3	0.00	0.00	0.00	0.00	0.00	8.41	5.41	5.78	6.74	0.00	0.00	0.00	0.00
	2	0.00	0.00	0.00	0.00	8.22	7.28	4.65	7.10	5.33	0.00	0.00	0.00	0.00
	1	0.00	0.00	0.00	6.60	7.17	4.55	4.49	5.03	3.72	0.00	0.00	0.00	0.00
4 (c)	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.07	6.43	6.77	0.00	0.00	0.00
	3	0.00	0.00	0.00	0.00	0.00	0.00	7.11	9.20	9.45	11.12	0.00	0.00	0.00
	2	0.00	0.00	0.00	0.00	0.00	6.44	8.26	8.82	6.89	5.83	0.00	0.00	0.00
	1	0.00	0.00	0.00	0.00	7.63	4.91	3.88	2.75	2.71	3.34	0.00	0.00	0.00
5 (u)	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.68	5.51	6.77	0.00	0.00
	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.90	7.32	7.24	5.62	0.00	0.00
	2	0.00	0.00	0.00	0.00	0.00	0.00	4.82	5.56	4.91	2.91	5.85	0.00	0.00
	1	0.00	0.00	0.00	0.00	0.00	6.86	6.26	5.85	5.89	6.39	7.36	0.00	0.00
6 (s)	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.57	8.39	11.37	0.00
	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.55	8.68	9.70	8.17	0.00
	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.18	5.68	5.39	3.63	0.00
	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.89	5.24	6.98	4.27	0.00
7 (e)	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.30
	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.97
	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.95
	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.31

that introducing the variable duration results improves recognition accuracy as well as speed as evidenced by the experimental results shown in the next section.

To incorporate the notion of duration probability as part of finding the best match [26], the local distance shown in (11) is modified as follows:

$$\hat{D}(v, b, e) = d\left(\mathbf{F}(S(b, e)), \mathbf{x}_{c_i(j)}(v)\right) - \lambda \cdot dur\left(e - b + 1, c_i(j)\right) \quad \text{for } 0 \le v < N_{c_i(j)} \qquad (19)$$

where $dur(\cdot, \cdot)$ is the duration probability defined in (3) and λ is estimated based on characteristics of the feature set. Experiments show that the incorporation of the duration probability improves recognition accuracy as well.

As a byproduct of our approach, we can also obtain character boundaries in a word image from Table 4. Table 6 shows the boundary of each character. While Table 3 shows the character "*s*" best matches with the first segment (using local information) Table 6 shows "*s*" best matches two segments (using global information). Locating character boundaries is useful in automatic generation of character databases for training and testing.

7 EXPERIMENTS AND RESULTS

7.1 Performance Evaluation

To evaluate the speed and recognition accuracy of the system, 3,000 postal words (digitized at 212 dpi), including city names, firm names, personal names, street names, and state names are used. Given a test word image, corresponding dictionaries are randomly generated with size of 10, 100, and 1,000 words. The true word is always present in the lexicon.

In addition to the 74 feature based system described in Section 5, a 38 feature based system was designed as a faster system for comparison purpose. The 38 feature set is a subset of the 74 feature set. It consists of two global features (4) and 36 local features (20).

$$F_{l_{ij}} = \frac{s_{ij}}{S_j} \qquad (20)$$

where

i = subimage number $(1, 2, \ldots, 9)$,
j = slope mod 4,
s_{ij} = number of components with j, from subimage i,
S_j = $\max_i s_{ij}$

All components are classified into four directional categories, horizontal, vertical and two diagonal. Table 7 and Table 8 show the improvements obtained by using the concept

TABLE 4
SECOND STAGE OF MATCHING PROCEDURE

					Ending		Segment						
j	0	1	2	3	4	5	6	7	8	9	10	11	12
0(s)	5.10	7.33	8.15	7.72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1(y)	0.00	15.43	17.38	12.07	14.83	17.79	17.02	0.00	0.00	0.00	0.00	0.00	0.00
2(r)	0.00	0.00	19.50	23.39	17.03	19.63	20.16	21.77	0.00	0.00	0.00	0.00	0.00
3(a)	0.00	0.00	0.00	26.10	27.72	21.58	21.68	22.81	23.46	0.00	0.00	0.00	0.00
4(c)	0.00	0.00	0.00	0.00	33.73	32.54	25.45	24.43	25.52	26.80	0.00	0.00	0.00
5(u)	0.00	0.00	0.00	0.00	0.00	40.59	38.56	31.30	30.32	27.34	30.05	0.00	0.00
6(s)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	35.19	35.56	34.33	30.97	0.00
7(e)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	34.28

TABLE 5
VARIABLE DURATION

char	0	1	2	3	4	5	6	7	8	9	a	b
no. of segments	2	1	3	3	3	3	3	3	2	2	3	3
char	c	d	e	f	g	h	*i*	*j*	k	l	m	*n*
no. of segments	3	3	3	3	3	3	2	3	3	2	4	3
char	o	p	q	r	s	t	u	v	w	x	y	z
no. of segments	3	3	3	3	3	3	3	3	4	3	3	3

TABLE 6
BOUNDARY OF EACH CHARACTER

j	0	1	2	3	4	5	6	7	8	9	10	11	12
0(s)	0	1	2	3	-	-	-	-	-	-	-	-	-
1(y)	-	0	1	1	0	2	3	-	-	-	-	-	-
2(r)	-	-	1	2	3	3	3	4	-	-	-	-	-
3(a)	-	-	-	2	2	4	4	4	4	-	-	-	-
4(c)	-	-	-	-	3	3	5	6	7	8	-	-	-
5(u)	-	-	-	-	-	4	4	6	7	7	7	-	-
6(s)	-	-	-	-	-	-	-	-	7	8	9	9	-
7(e)	-	-	-	-	-	-	-	-	-	-	-	-	11
	^^^^^^^^^^		^^^^^^^^^^		^	^^^^^^^^^^		^	^^^^^^^^^^		^^^^^^^^^^		^
	s		y		r	a		c	u		s		e

TABLE 7
TIMING PROFILE IN *MSEC* ON SPARC 10

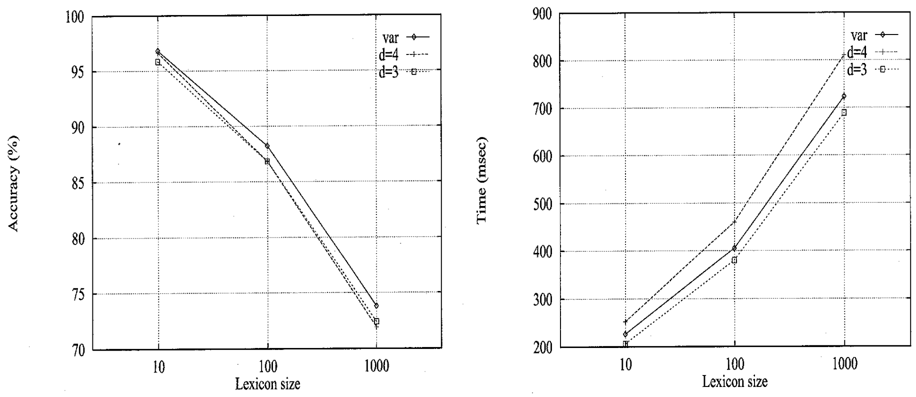
		10			100			1,000		
ftr	Module	var	d = 4	d = 3	var	d = 4	d = 3	var	d = 4	d = 3
74	Chain Gen.	22	21	21	22	22	22	27	26	26
	Slant Norm.	10	10	9	10	10	9	11	10	10
	Segmentation	15	15	15	15	15	15	15	16	16
	Feature Ext.	34	34	24	34	34	25	37	37	26
	Recognition	145	172	136	324	379	308	633	720	610
	Total	226	252	205	405	460	379	723	809	688
38	Chain Gen.	22	22	22	22	22	22	27	26	26
	Slant Norm.	10	9	9	10	10	10	11	11	10
	Segmentation	15	15	15	15	15	15	15	15	15
	Feature Ext.	32	31	22	32	32	22	35	34	24
	Recognition	67	80	63	147	175	226	387	439	383
	Total	146	157	131	226	254	295	475	525	458

TABLE 8
RECOGNITION ACCURACY (IN PERCENT)

		10			100			1,000		
ftr		var	d = 4	d = 3	var	d = 4	d = 3	var	d = 4	d = 3
74	Top 1	96.80	96.63	95.83	88.23	86.83	86.83	73.80	71.90	72.40
	Top 2	98.63	98.76	98.30	93.36	92.86	92.36	83.20	81.40	82.20
	Top 20				98.93	99.00	98.40			
	Top 50							98.70	98.50	97.60
38	Top 1	95.80	95.40	95.10	85.86	84.63	84.96	68.70	65.20	67.90
	Top 2	98.26	98.23	97.80	91.93	91.20	90.80	80.20	78.20	78.90
	Top 20				98.96	99.03	98.76			
	Top 50							97.90	97.50	97.10

TABLE 9
FAILURE ANALYSIS

Type	Recognition	Segmentation	Image quality
No. of images	51 (53.1 percent)	35 (36.5 percent)	10 (10.4 percent)



(a) (b)

Fig. 11. Performance and speed comparison. (a) Performance—the system with variable duration is represented by the solid line and gives the best recognition accuracy for all lexicon sizes. (b) Speed of the system with variable duration is between the two other systems.

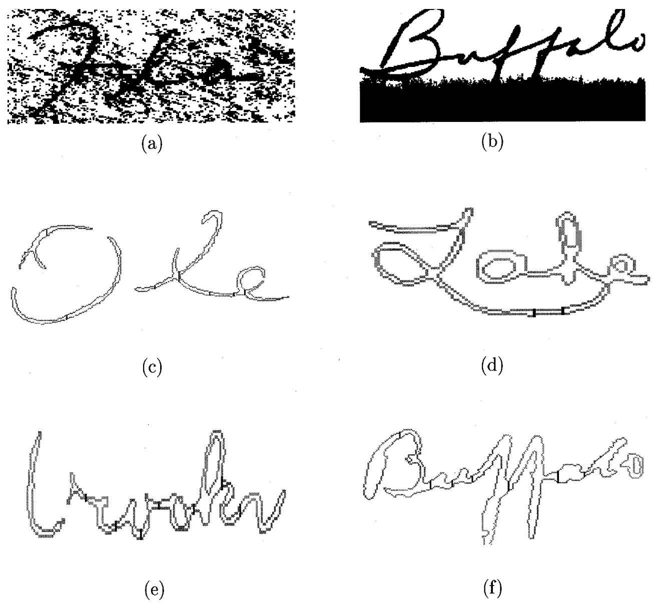


Fig. 12. Error analysis. (a) Background noise. (b) Noise introduced by scanning device. (c) Oversegmentation. (d) Unable to segment. (e) Recognition error (confused with a similar entry in lexicon). (f) Recognition error (due to tough edge).

of variable duration described in the previous section. In Table 7 and Table 8, the first column (var) of each lexicon size represents the results with the variable matching window size. The second column (d = 4) and the third column (d = 3) represent results with fixed matching window sizes.

Table 7 shows time taken by each module (Fig. 2) using the two feature sets. As matching window size decreases, feature extraction time also decreases. Time taken for recognition is determined by both the lexicon size and matching window size. Recognition time increases at the rate of $log(L_N)$ as lexicon size L_N increases because of the variable duration and the early rejection during matching.

Table 8 shows the recognition performance of both configurations. When the matching window size is fixed at four, performance is not as good as when the window size is kept adaptive. Fig. 11 provides a graphical illustration of the results for the 74 feature based system. More than 20 percent of characters "*m*" and more than 10 percent of characters "*w*" split into four segments (Table 2). Both "*m*" and "*w*" characters are frequent in English. Table 8 and Fig. 11a reflect the drop in performance mainly on images with "*m*"s and "*w*"s when the window size is three. The fast system with simplified 38 feature set produces compatible accuracy against the other system for small size lexicon entries.

7.2 Failure Analysis

We examined 96 images classified as failures (74 feature based system and lexicon size of 10). Four main sources of error have been identified:

1) *Image quality*: Fig. 12a shows an example of acute background noise and Fig. 12b shows typical scanning noise not handled by the preprocessing routine.
2) *Segmentation*: Fig. 12c shows character "*O*" split into four segments. The matching window size of character "*O*" is three (Table 5) hence the matching procedure ignores the last segment of the character reducing the matching confidence. This type of error is closely related to fragmentation of strokes caused by binarization. Fig. 12d shows the stroke of "*L*" touching "*e*" leading to a segmentation problem.
3) If the lexicon has very "similar" entries (Fig. 12e), *Chuck* in the lexicon is selected as the first choice and *crooks* (truth) is selected as the second choice.
4) Jagged chain code contour also reduces recognition confidence (Fig. 12f).

Table 9 shows the different causes of failure.

7.3 Comparison

Chen and Kundu [27] report results on the same data set and with the same lexicons as we have used. They report top choice recognition performances of 93.2 percent, 85.2 percent, and 64.6 percent for lexicons of sizes 10, 100, and 1,000, respectively. This compares with the performance numbers reported in this paper: 96.8 percent, 88.23 percent, and 73.8 percent, respectively. Also, the top choice recognition accuracy of 88.29 percent obtained by us compares with 72.3 percent reported in [21], where 271-word lexicon and 94 test images were used. Chen, Kundu, and Srihari [28] report the same performance on the small size lexicons (96.8 percent). But, 78.7 percent and 59.6 percent were reported for lexicon sizes of 100 and 1,000.

7.4 Applications

The recognition algorithm described in this paper has been integrated into several real-time systems by CEDAR. One of the systems is for handwritten address interpretation on mail pieces [2]. The system employs fast processors to meet the USPS speed requirement of processing over 10,000 mail pieces per hour. Same extent of accuracy has been obtained for images having digit strings, such as street names (e.g., *7th Ave* in Fig. 1a), by providing the corresponding lexicons.

8 Conclusion

A fast chain code based handwritten word recognition system has been implemented. The speed is 100 ~ 200 msec on a single Sparc-10 platform with a 10 word dictionary. The corresponding top choice performance is 96.8 percent. Development of efficient methods for the preprocessing, segmentation, and feature extraction resulted in speed improvements. Use of variable duration in the recognition procedure improved performance as well as speed. The size of the feature set, type of features, and the concept of restricted matching window size based on the variable duration all contribute to improvements in speed.

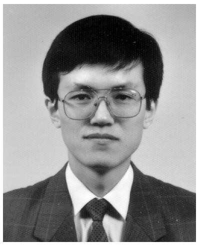
Acknowledgment

The authors would like to thank the reviewers for many valuable comments and suggestions.

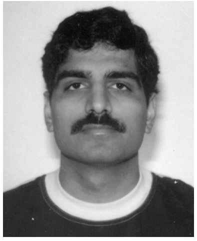
References

[1] S.N. Srihari, "High-Performance Reading Machines," *Proc. IEEE*, vol. 80, no. 7, pp. 1,121–1,132, 1992.
[2] V. Govindaraju, A. Shekhawat, and S.N. Srihari, "Interpretation of Handwritten Addresses in US Mail Stream," *Third Int'l Workshop on Frontiers in Handwriting Recognition*, Buffalo, N.Y., pp. 197–206, 1993.
[3] E. Cohen, J. Hull, and S.N. Srihari, "Control Structure for Interpreting Handwritten Address," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, pp. 1,049–1,055, Oct. 1994.
[4] D. Guillevic and C.Y. Suen, "Cursive Script Recognition: A Sentence Level Recognition Scheme," *Fourth Int'l Workshop on Frontiers in Handwriting Recognition*, Taipei, Taiwan, pp. 216–223, 1994.
[5] J.C. Simon, O. Baret, and N. Gorski, "A System for the Recognition of Handwritten Literal Amounts of Checks," *Proc. Conf. Document Analysis System*, Kaiserslautern, Germany, pp. 135–155, 1994.
[6] T.M. Breuel, "Design and Implementation of a System for the Recognition of Handwritten Responses on US Census Forms," *Proc. Conf. Document Analysis System*, Kaiserslautern, Germany, pp. 109–134, 1994.
[7] S. Madhvanath, V. Govindaraju, V. Ramanaprasad, D.-S. Lee, and S.N. Srihari, "Reading Handwritten US Census Forms," *Third Int'l Conf. Document Analysis and Recognition (ICDAR-95),* Montreal, Canada, pp. 82–85, 1995.
[8] S.N. Srihari, Y.-C. Shin, V. Ramanaprasad, and D.-S. Lee, "Name and Address Block Reader System for Tax Form Processing," *Third Int'l Conf. Document Analysis and Recognition (ICDAR-95),* Montreal, Canada, pp. 5–10, 1995.
[9] M.K. Brown and S. Ganapathy, "Preprocessing Techniques for Cursive Word Recognition," *Pattern Recognition*, vol. 16, no. 5, pp. 447–458, 1983.
[10] R.M. Bozinovic and S.N. Srihari, "Off-Line Cursive Script Word Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 68–83, Jan. 1989.
[11] J.C. Simon, "Off-Line Cursive Word Recognition," *Proc. IEEE*, vol. 80, no. 7, pp. 1,150–1,161, 1992.

[12] A.M. Gillies, "Cursive Word Recognition Using Hidden Markov Models," *Fifth Advanced Technology Conf.*, Washington, D.C., pp. 557–562, 1992.
[13] A. Kaltenmeier, T. Caesar, J.M. Gloger, and E. Mandler, "Sophisticated Topology of Hidden Markov Models for Cursive Script," *Int'l Conf. Document Analysis and Recognition*, Tsukuba Science City, Japan, pp. 139–142, Oct. 1993.
[14] L.R. Rabiner and B. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, N.J.: Prentice Hall, 1993.
[15] J.T. Favata, *Recognition of Cursive, Discrete and Mixed Handwritten Words Using Character, Lexical and Spatial Constraints*. PhD thesis, Dept. of Computer Science, State Univ. of New York at Buffalo, 1993.
[16] H. Freeman, "Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves," *Proc. Nat. Electronics Conf.*, vol. 17, pp. 412–432, 1961.
[17] J. Schürmann, N. Bartneck, T. Bayer, J. Franke, E. Mandler, and M. Oberlander, "Document Analysis—From Pixels to Contents," *Proc. IEEE*, vol. 80, no. 7, pp. 1,101–1,119, 1992.
[18] H. Freeman, "Computer Processing of Line-Drawing Images," *Computing Surveys*, vol. 6, no. 1, pp. 57–97, 1974.
[19] C.-C. Lu and J.G. Dunham, "Highly Efficient Coding Schemes for Contour Lines Based on Chain Code Representations," *IEEE Trans. Communications*, vol. 39, no. 10, pp. 1,511–1,514, 1991.
[20] G. Kim and V. Govindaraju, "Efficient Chain Code Based Image Manipulation for Handwritten Word Recognition," *Proc. SPIE Symp. Electronic Imaging Science and Technology (Document Recognition III)*, San Jose, Calif., vol. 2,660, pp. 262–272, Feb. 1996.
[21] M.-Y. Chen, A. Kundu, and J. Zhou, "Off-Line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, pp. 481–496, May 1994.
[22] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, Inc., 1973.
[23] H. Sakoe, "Two-Level DP-Matching—A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 27, pp. 588–595, Dec. 1979.
[24] C.S. Myers and L.R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 284–297, 1981.
[25] H. Ney, "A Comparative Study of Two Search Strategies for Connected Word Recognition: Dynamic Programming and Heuristic Search," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 5, pp. 586–595, 1992.
[26] C. Lee and L.R. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, pp. 1,649–1,658, Nov. 1989.
[27] M.-Y. Chen and A. Kundu, "An Alternative to Variable Duration HMM in Handwritten Word Recognition," *Pre-Proceedings, Third Int'l Workshop on Frontiers in Handwriting Recognition*, Buffalo, N.Y., pp. 82–91, 1993.
[28] M.Y. Chen, A. Kundu, and S.N. Srihari, "Variable Duration Hidden Markov Model and Morphological Segmentation for Handwritten Word Recognition," *IEEE Trans. Image Processing*, vol. 4, pp. 1,675–1,688, Dec. 1995.



Gyeonghwan Kim received the BS (cum laude) and the MS in electronics engineering from the Sogang University, Korea, in 1984 and 1986, respectively. He received the PhD degree in electrical and computer engineering from the State University of New York at Buffalo in 1996. From 1986 to 1991 he was a researcher in the Goldstar (currently LG) Precision Technology Institute (Korea). He is currently with the Center of Excellence for Document Analysis and Recognition (CEDAR) at SUNY Buffalo as a research scientist. His research interests include handwriting recognition, document analysis, neural networks, and human-computer interface.



Venu Govindaraju is associate director of the Center of Excellence for Document Analysis and Recognition at SUNY Buffalo. He is the project manager and senior research scientist for the CEDAR Handwritten Address Interpretation Project. He holds the Research Assistant Professorship in the Department of Computer Science at SUNY Buffalo. Dr. Govindaraju received his BTech (Hons.) in computer science from the Indian Institute of Technology at Kharagpur in 1986 and his MS and PhD in computer science from SUNY Buffalo in 1988 and 1992, respectively. Dr. Govindaraju has coauthored more than 50 research papers in various conferences and journals in the fields of pattern recognition and computer vision.