It appears that binary morphology has found a new field of application in the area of cluster analysis. As a natural extension of this work, the authors intend to publish a follow-up paper that will discuss the application of multivalued set-theoretic operations to pattern classification.

ACKNOWLEDGMENT

The authors wish to thank O. M'hirit, Professor at the School of Forestry of Sale, Morocco, for providing the Cedrus data. They are also grateful to B. Ceurtemont for his programming assistance.

REFERENCES


Attributed String Matching by Split-and-Merge for On-Line Chinese Character Recognition

Yih-Tay Tsay and Wen-Hsiang Tsai

Abstract—Consecutive strokes of Chinese characters tend to be connected in fast writing, and this causes a problem for most stroke-based recognition approaches. In this correspondence, we propose a recognition scheme to recognize cursive Chinese characters under the constraint of correct stroke writing orders. The proposed recognition scheme consists of two phases: candidate character selection and detailed matching. In the former phase, an input script with $N$ strokes is used to split the strokes of each reference character into $N$ corresponding parts. In the latter phase, the connected input strokes are broken into multiple strokes under the guidance of candidate characters. In both phases, dynamic programming is employed for stroke or character matching. Good experimental results prove the feasibility of the proposed approach for cursive Chinese character recognition.

Index Terms—Character recognition, on-line Chinese character recognition, string matching.

I. INTRODUCTION

On-line Chinese character recognition has been studied for many years. Many techniques have been published to solve the problem. They can basically be classified into two major approaches: namely, the statistical method [1], [2] and the structural method [3]-[11]. In the statistical method, a feature vector is usually extracted from the strokes composing an input character. Character recognition is performed by selecting the reference character with the minimum distance from the input character.

In the structural method, a set of basic strokes are usually selected as the primitives [5]-[9], and stroke recognition is based on the use of certain geometrical features like line segment directions, stroke lengths, corner numbers, etc. Stroke numbers, stroke orders, stroke relations, etc., are also found to be useful for character recognition. The recognition schemes are based on the use of decision tree [3], string matching [7], syntax and/or semantics analysis [5], radical decomposition [4], etc. Most of the recognition schemes depend heavily on the recognition of strokes. Stroke-based recognition, however, is effective under the constraint of careful writing, which places a burden on the users. Furthermore, in many applications, there is a need for fast writing for data entry. When writing fast, a user tends to connect consecutive strokes. The stroke shape in a character also varies from time to time, even when it is written by the same user. This makes stroke-based recognition a very difficult problem that does not have very high recognition rates. Recently, several attempts have been made to remove unnatural writing constraints, and some algorithms [8]-[11] have been proposed to compensate for the variations of stroke connections, stroke orders, and stroke distortion.

In this correspondence, attributed string matching with split-and-merge based on dynamic programming (DP) techniques is proposed to recognize cursive Chinese characters under the constraint of correct

Manuscript received Aug. 29, 1992; revised January 7, 1993. This work was supported by National Science Council, Republic of China under Grant NSC 81-0404-E-009-017. Recommended for acceptance by Editor-in-Chief A. K. Jain.

Y. -T. Tsay is with the Department of Computer and Engineering Science, Yuan-Ze Institute of Technology, Taoyuan, Taiwan, R.O.C.
W.-H. Tsai is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
IEEE Log Number 9200754.
stroke orders. In order to reduce the computing time, the proposed recognition scheme is decomposed into two phases: candidate character selection and detailed matching. The detailed descriptions of the two phases are given in the following sections.

II. ATTRIBUTED STRING REPRESENTATION

An input character obtained from a digitization tablet can be represented as an attributed string by concatenating all the strokes according to the stroke writing order. Two kinds of primitives, namely, real primitive and pseudo primitive, are used in the attributed string. Each real primitive is a line segment that approximates part of a stroke, and each pseudo primitive is an implicit line segment that connects two consecutive strokes. Attributed string representations of on-line handwritten Chinese characters are easily obtained from the results of appropriate preprocessing and angle filtering processes [15].

Define stroke string $A_s = S_1 S_2 \cdots S_N$ to represent character $A$ with $N$ strokes, where each substring $S_j = s_{r,j} s_{r,j+1} \cdots s_{r,j+L_j-1}$, $j = 1, 2, \cdots, N$ represents the $j$th stroke of $A$, where $L_j$ is the number of line segments contained in $S_j$. The $i$th primitive $s_{r,i}$ of $A$ is also the leading pseudo primitive of $S_i$, and the primitive $s_{r,i-1}$ is the $i$th real primitive of $S_i$. To describe the geometric properties of the primitives, an attribute vector $(q_{i}, l_{i})$ is associated with each real primitive $s_{r,i}$, where $q_{i}$ and $l_{i}$ are the direction and length attributes of $s_{r,i}$, respectively. Let $P_1$ and $P_2$ be the starting and the end feature points of primitive $r_i$. The direction attribute $q_{i}$ of $r_i$ is defined as the slope of the line segment $P_1 P_2$, which is quantized into one of 64 directions. The length attribute $l_{i}$ of $r_i$ is defined as the Euclidean distance between $P_1$ and $P_2$. In addition, an attribute vector $(q_{i}, l_{i})$ is associated with each pseudo primitive $p_{i}$, where $q_{i}$ and $l_{i}$ are the direction and length attributes of $S_i$, respectively, and the coordinates $(x_{i}, y_{i})$ and $(x_{i+1}, y_{i+1})$ specify the positions of the starting and the end feature points of $S_i$, respectively. Fig. 3(a) shows the stroke string of a character with seven strokes, whose pseudo primitives are drawn by dashed line segments.

Define another attributed string representation of a character (called the pseudo string) as the concatenation of all the pseudo primitives of the character in order. Let attributed string $A_p = p_1 p_2 \cdots p_N$ represent the pseudo string of character $A$ with $N$ strokes where $p_i$ represents the $i$th pseudo primitive. Instead of using the attribute vector defined for a pseudo primitive, another attribute vector $(q_{i}, l_{i}, a_{i}, s_{i}, x_{i}, y_{i})$ is used to specify the attributes of each pseudo primitive $p_i$ in the pseudo string, where $q_{i}$ is the direction of $p_i$, and $l_{i}, a_{i}, s_{i}, x_{i}, y_{i}$ are the directions of the preceding and the following real primitives in the stroke string, respectively. In addition, $(x_{i}, y_{i})$ is the coordinates of the starting and the end feature points of $p_i$, respectively. The pseudo string shown in Fig. 3(b) of an input script is $p_1 p_2 \cdots p_5$, where $p_1 = s_1, p_2 = s_6, p_3 = s_8, p_4 = s_{12}$, and $p_5 = s_{14}$.

III. CANDIDATE CHARACTER SELECTION

In this section, the candidate characters for an input script are determined by matching the pseudo string of the input (called the input pseudo string henceforth) with those of the reference characters.

The matching scheme may be regarded as a process of splitting the stroke sequence of each reference character by the pseudo primitives of the input script. The validity of this "reference character splitting" for candidate character selection is based on the following observation. When a user writes an input script in a cursive way, it rarely happens that a stroke in the input script is broken into two or more strokes. On the contrary, two consecutive strokes in the reference character are often connected together to form a single stroke, i.e., the pseudo primitive between the two consecutive strokes in the reference character often disappears in the input pseudo string. Therefore, under the assumption that the input script is written in a correct stroke order, a pseudo primitive in the input script always corresponds to a pseudo primitive in the reference character, and the stroke number of the input script is usually no greater than that of the reference character. In other words, the input pseudo string can be considered to be a partial string of that of the reference character.

Define the pseudo string distance between two characters as the minimum of the sums of the distances of the corresponding pseudo primitives in the two characters. Then, the reference characters with smaller pseudo string distance values can be selected as the candidates for detailed matching.

More specifically, the phase of candidate character selection can be divided into three stages. The first stage is to determine an initial set $R$ of reference characters from which candidate characters can be selected. Under the assumption that each stroke in the reference character is seldom broken into two or more strokes, the number $N$ of strokes in the input script can be utilized as the lower limit of the number of strokes for each reference character in $R$. On the other hand, because each stroke in the input script may correspond to one or more strokes in each reference character, the numbers of line segments in the strokes of the input script can be used to estimate the upper limit of the possible number $U$ of strokes for each reference character in $R$. This can be done by the use of a heuristic function $F_h$ in the following way:

$$U = \sum_{i=1}^{N} F_h(n_i)$$

where

$$F_h(n_i) = n_i, \quad \text{if } n_i \leq 2;$$

$$n_i - 1, \quad \text{if } 3 \leq n_i \leq 4;$$

$$n_i - 2, \quad \text{if } n_i \geq 5$$

and $n_i$ is the number of line segments that approximate the $i$th stroke of the input script. The heuristic function defined above is based on the fact that if the line segments in the input stroke is longer, the strokes in the reference will be longer. Finally, the reference characters, each having its stroke numbers between $N$ and $U$, are selected to compose the set $R$. This completes the first stage of candidate character selection.

In the second stage of candidate character selection, the pseudo string of each reference character in $R$ is matched with that of the input script. Let attributed string $A_p = p_1 p_2 \cdots p_N$ represent the pseudo string of the input script $A$ with $N$ strokes and $B_p = p_1' p_2' \cdots p_M'$ represent the pseudo string of reference character $B$.
in R with M strokes. Assume that M > N. In addition, let 
\{(q_1, r_1, s_1, \tau_1), (q_2, r_2, s_2, \tau_2), \ldots, (q_N, r_N, s_N, \tau_N)\}
be the attribute vectors of the pseudo primitives of A and B, respectively. The
primitive distance \(d(p_i, p'_j)\) between primitives \(p_i\) and \(p'_j\) is defined as follows:

\[
d(p_i, p'_j) = H(q_{i1}, q'_{j1}) + H(q_{i2}, q'_{j2}) + H(q_{i3}, q'_{j3})
+ W_1(\tau_{i1} - \tau'_1) + [z_i - z'_j] + [x_i - z'_j].
\]

where \(H(p, q)\) is a function specifying difference measures, and \(W_1\) is a preselected constant used to normalize the weights of the
difference measures of the direction and position attributes. The
direction difference function \(H(q_{i4}, q'_{j4})\) is defined to be

\[
H(q_{i4}, q'_{j4}) = \min \{|q_i - q_j|, 64 - |q_i - q_j|\}.
\]

The pseudo string distance \(D_{s}(A, B)\) between A and B is defined as the minimum of the sums of the distances of the corresponding
pseudo primitives between A and B, i.e., \(D_{p}(A, B)\) can be defined as follows:

\[
D_{p}(A, B) = \min \sum_{i=1}^{N} d(p_i, p'_j(i))
\]

where the primitive-correspondence function \(f(i)\) must satisfy the following constraints:

\[
i \leq f(i) \leq i + M - N,
\]

\[
f(i - 1) < f(i), \quad \text{for } i = 1 \text{ through } N.
\]

The constraint \(f(i - 1) < f(i)\) comes from the assumption that the strokes in the input script are written in a correct order.

The pseudo string distance \(D_{p}(A, B)\) defined above can be computed as the minimum cost of a sequence of primitive edit operations
that transform \(B_p\) for a collection of primitive edit operations falls into one of the following two cases. 1) There exists one \(p_i\) of \(A_p\) corresponding to \(p'_j\). This case can be considered to be a change operation that transforms \(p'_j\) into \(p_i\), and the cost of the change operation is just the pseudo primitive distance \(d(p_i, p'_j)\); 2) there does not exist any \(p_i\) of \(A_p\) corresponding to \(p'_j\). This case can be considered to be a delete operation that deletes \(p'_j\) from \(B_p\), and the cost of the delete operation can be taken to be zero. Note that exactly \(M - N\) pseudo primitives in \(B_p\) should be deleted.

Let \(D\) be an array such that \(D(i, j)\) means the minimum edit cost of transforming \(p_1p_2\ldots p_i\) into \(p'_1p'_2\ldots p'_j\); or, equivalently, the minimum pseudo string distance between \(p_1p_2\ldots p_i\) and \(p'_1p'_2\ldots p'_j\). Based on the above concept of pseudo string edition, the array element \(D(i, j)\) can be obtained by the following DP technique:

\[
D(i, i) = d(p_i, p'_i),
\]

for \(i = 1, 2, \ldots, N\)

\[
D(i, j) = \min(D(i - 1, j - 1) + d(p_i, p'_j), D(i, j - 1)),
\]

for \(i = 1, 2, \ldots, N\) and \(j = 1, 2, \ldots, M\).

The final element \(D(N, M)\) is simply the minimum edit cost of transforming \(B_p\) into \(A_p\), which is also the desired minimum pseudo
string distance \(D_{p}(A, B)\) between A and B.

To obtain the index \(j\) of the corresponding \(p'_j\) of each \(p_i\), the above procedure can be easily modified by keeping the index flow
of \(D(i, j)\) into an array \(C\). After the final element \(D(N, M)\) is obtained, the corresponding pseudo primitive pairs can be obtained by tracing back through \(C\). By this correspondence information, it is easy to find the corresponding strokes in B for each input stroke in A. From these corresponding pairs, the stroke string of B can then be split into N substrings, where each contains one or more strokes. We call this process reference character splitting. Fig. 1 shows an example in which the pseudo string of an input script and a reference character are represented by \(A_p = p_1p_2\ldots p_i = s_1s_2s_3s_4s_5s_6s_7s_8\), and \(B_p = p'_1p'_2\ldots p'_j = s'_1s'_2s'_3s'_4s'_5s'_6s'_7s'_8\), respectively. Five corresponding primitives \((s_1, s'_1), (s_2, s'_2), (s_3, s'_3), (s_4, s'_4), (s_5, s'_5)\) can be obtained from the above procedure. Therefore, the stroke string of the reference character is split into five substrings \(s'_1s'_2s'_3s'_4s'_5, s'_6s'_7s'_8\), \(s'_1s'_2s'_3s'_4), s'_1s'_2s'_3s'_4s'_5s'_6s'_7s'_8\), and \(s'_1s'_2s'_3s'_4s'_5s'_6s'_7s'_8\), which correspond, respectively, to the input strokes represented by substrings \(s_1s_2s_3s_4s_5s_6s_7s_8, s_1s_2s_3s_4, s_1s_2s_3s_4s_5s_6s_7s_8,\) and \(s_1s_2s_3s_4s_5s_6s_7s_8\).

The final stage of the candidate character selection phase is to find out from the set \(B\) the \(N\) reference characters with the smallest pseudo string distances from the input script as the candidate characters for detailed matching described next.

IV. DETAILED MATCHING

In the detailed matching phase, each input stroke is matched with the corresponding strokes in each candidate character. Each stroke in the input or the reference is represented by a substring of pseudo and real primitives. The basic idea of matching here is to use the pseudo primitive in the substring of a candidate character to split the substring of an input stroke if the input stroke is matched with more than one stroke in the candidate character. This process is called input stroke splitting.

More specifically, when more than one stroke in the candidate character are considered to correspond to one input stroke, each pseudo primitive in the strokes of the candidate character may fall into one of the following two cases: 1) The pseudo primitive corresponds to a real primitive in the input script. For example, the pseudo primitive \(s'_1\) in Fig. 1(a) corresponds to the real primitive \(r_3\) in Fig. 1(b). It is reasonable here to match the two feature points of \(s'_1\) with the corresponding feature points of \(r_3\) and to split the first input stroke in Fig. 1(b) into \(r_2\) and \(r_3\). This match will be denoted as \(r_3 \rightarrow s'_1\); 2) the pseudo primitive disappears in the input script, i.e., the pseudo primitive reduces to one feature point. For example, the pseudo primitive \(s'_{11}\) in Fig. 1(a) becomes a feature point \(P\) that connects the real primitives \(r_{10}\) and \(r_{11}\) in Fig. 1(b). In this case, the two feature points of \(s'_{11}\) should be matched with the feature point \(P\), and the third input stroke in Fig. 1(b) should be split into \(r_{10}\) and \(r_{11}\). This match will be denoted as \(r_{11} \rightarrow s'_{11}\), where \(r_{11}\) is a null primitive connecting \(r_{10}\) and \(r_{11}\). The cost function of the above split operations will be described later, after the detailed matching process is described in the following.

Let attributed strings \(S_i = s_1s_2s_3\ldots s_i s'_{i+1}\ldots s'_m\) represent the substrings of the \(i\)th stroke of the input script \(A\) and the corresponding strokes in a candidate character \(B\), respectively. Note that the leading pseudo primitive \(s'_1\) is the only pseudo primitive in \(S_i\), and it should be matched with the leading pseudo primitive \(s_1\) of \(S_i'\). In addition, note that \(S_i\) may contain more than one pseudo primitive. Let the direction and the length attributes of \(r_i\) and \(r'_j\) be denoted as \((q_i, \tau_i)\) and \((q'_j, \tau'_j)\), respectively. In addition, let the attribute vectors of pseudo primitives \(s_i, s'_j\) be denoted as \((q_i, \tau_i, x_i, y_i, z_i, x_{i2}, y_{i2})\) and \((q'_j, \tau'_j, x'_j, y'_j, z'_j, x'_{j2}, y'_{j2})\), respectively. Generally speaking, the interstroke distance between \(S_i\) and \(S'_i\) can be defined in terms of the minimum-cost edit sequence from \(S_i\) to \(S'_i\). Allowable edit operations and their cost functions \(R\) dependent on the types of primitives are described as follows:

1) Change: To replace the first pseudo primitive \(s_1\) of \(S_i\) with \(s'_1\) of \(S'_i\), or to replace a real primitive \(r_i\) with another \(r'_j\), denoted...
where $s_i \rightarrow s_j'$ or $r_i \rightarrow r_j'$, respectively, with costs
\[
R(s_i \rightarrow s_j') = H(q_i, q_j') + W_2 \cdot [1 - l_j'] + E_d
\]
\[
\cdot (x_{j'1}, y_{j'1}), (x_{j1}, y_{j1})]
\]
\[
+ E_d(x_{j'2}, y_{j'2}), (x_{j2}, y_{j2})]
\]
\[
R(r_i \rightarrow r_j') = H(q_i, q_j') + W_2 \cdot [1 - l_j']
\]

2) Insert: To insert a real primitive $r_j'$ into $S_i$, denoted as $\lambda \rightarrow r_j'$, where $\lambda$ is a null symbol, with cost
\[
R(\lambda \rightarrow r_j') = K_i + W_2 \cdot l_j'
\]

3) Delete: To delete a real primitive $r_i$, from $S_i$, denoted as $r_i \rightarrow \lambda$, with cost
\[
R(r_i \rightarrow \lambda) = K_d + W_2 \cdot l_i
\]

4) Split: To split $S_i$ into two parts; denoted as $r_i \rightarrow s_j'$ or $\lambda \rightarrow s_j'$, i.e., for the case $r_i \rightarrow s_j'$, $S_i = s_1r_2 \cdots r_n$ is split into $s_1r_2 \cdots r_{i-1}$ and $r_{i+1} \cdots r_n$, whereas for the case $\lambda \rightarrow s_j'$, $S_i = s_1r_2 \cdots r_n$ is split into $s_1r_2 \cdots r_{i-1}$ and $r_{i} \cdots r_n$, with costs
\[
R(r_i \rightarrow s_j') = H(q_i, q_j') + W_2
\]
\[
\cdot [1 - l_j'] + E_d((x_{j',1}, y_{j',1}), (x_i, y_i))
\]
\[
+ E_d((x_{j',2}, y_{j',2}), (x_{i+1}, y_{i+1})),
\]
\[
R(\lambda \rightarrow s_j') = K_i + W_2
\]
\[
\cdot [l_j'] + E_d((x_{j',1}, y_{j',1}), (x_i, y_i))
\]
\[
+ E_d((x_{j',2}, y_{j',2}), (x_{i+1}, y_{i+1})),
\]

5) Merge: To merge $k$ consecutive real primitives in $S_i$ into a longer real primitive, denoted as $r_i \cdots r_{i+k-1} \rightarrow r_i^*$
\[
R(r_i \cdots r_{i+k-1} \rightarrow r_i^*) = (l_j^* / L_i) \cdot \sum_{j=1}^{k-1} (H(q_i, q_j') \cdot l_j'^* / l_j')
\]

where $(q_i^*, l_i^*)$ is the attribute vector of the combined real primitive $r_i^*$, $E_d(x_1, y_1), (x_2, y_2)), L_i$, and $N_i$ are the total length and the number of all the real primitives in the input script, and $W_2, K_i, K_d, K_s$ are preselected constants. Detailed explanations of the cost functions and the attribute vector of the combined primitive defined above can be found in [13] and [14].

The interstroke distance $D_i(S_i, S_i')$ from $S_i$ to $S_i'$ is defined as the minimum of the cost of all the edit sequences taking $S_i$ to $S_i'$, and the character distance from the input script $A$ to the candidate character $B$ is computed as
\[
D_i(A, B) = \sum_{i=1}^{N} D_i(S_i, S_i')
\]

where $N$ is the number of strokes in $A$.

Because merge operations are only applied to the real primitives of the input stroke, the merge costs and the attributes of all possible combined real primitives can be computed independently and stored in an array before matching the input with the candidate characters. In addition, to reduce the computation time, it is assumed that the maximum number of real primitives allowed in a merge operation is three. This has been found to be reasonable according to our experimental results.

The interstroke distance $D_i(S_i, S_i')$ between the $t$th stroke $S_t = s_1r_2 \cdots r_n$ of $A$ and its corresponding strokes $S_i' = s_1'r_2' \cdots r_{n+1}'$ in candidate character $B$ is computed as
\[
D_i(A, B) = \sum_{i=1}^{N} D_i(S_i, S_i')
\]

where $N$ is the number of strokes in $A$. In our experiments, the handwriting area of each character is 15 by 15 mm on a digitization tablet whose spatial resolution is 20 lines/mm. After the preprocessing and angle filtering processes are performed to each stroke of a character, a size and position normalization operation is applied to reduce the character data into a size of 200 by 200 pixels, from which the attributed string representation is obtained. The attribute weighting factors $W_i$ described in Section III and $W_2$ described in Section IV were both set to be the 0.25, which equalizes the weights of the difference measures of the direction and length attributes. The constants $K_i$ and $K_d$ for insertions and deletions described in Section IV were both chosen to be 16, which is one half of the maximum value of the direction difference function. In addition, the constant $K_s$ for the split cost was chosen to be 8. The choice of the above preselected constants was based on our experimental results. In addition, the recognition rates changed only slightly when the preselected values were perturbed by small amounts.

A database composed of 3100 Chinese characters was constructed as the references. The stroke numbers of the characters range from 1 to 24. To create the input data, in total, 200 characters chosen from famous Chinese poems were written by ten persons. Each scriber was instructed to write characters in identical character style and with handwriting speed identical to writing daily notes. The scripter was also informed of the correct stroke writing order of each character. Table I shows the experimental results. Out of the 2000 input scripts, only 76 were misrecognized. The recognition rate, thus, was 96.2%. The fourth through the 11th columns of Table I show the statistics of the numbers of input scripts based on the number of connected strokes in the scripts. Out of the 2000 input scripts, only 1061 have
Fig. 2. Partial recognition results of 200 input scripts chosen from ten scripters. Each error is marked by the notation $A \rightarrow B$, which means that the script actually is $A$ but was recognized to be $B$.

TABLE I
RECOGNITION RESULTS OF 200 INPUT SCRIPTS WRITTEN BY TEN SCRIPTERS

<table>
<thead>
<tr>
<th>scripter</th>
<th>error count</th>
<th>recognition rate</th>
<th>distribution of input scripts based on number of connected strokes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$k=0$</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>94.0%</td>
<td>53</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>98.9%</td>
<td>124</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>97.0%</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>92.5%</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>97.5%</td>
<td>142</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>92.0%</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>96.3%</td>
<td>112</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>95.0%</td>
<td>138</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>99.5%</td>
<td>146</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>99.0%</td>
<td>177</td>
</tr>
<tr>
<td>total</td>
<td>78</td>
<td>96.2%</td>
<td>1061</td>
</tr>
</tbody>
</table>

TABLE II
CUMULATIVE RECOGNITION RATES

<table>
<thead>
<tr>
<th>order</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>15th</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>96.2%</td>
<td>99.6%</td>
<td>99.35%</td>
<td>99.55%</td>
<td>99.75%</td>
<td></td>
</tr>
</tbody>
</table>

It has been found that three factors lead to the above 76 misrecognitions of the input scripts. The first is that the candidate selection phase fails to select correct references. Five errors come from this factor. The second factor is that similar character structures exist in Chinese characters. In total, 66 errors were caused by this factor. The third factor is the distortion existing in the input scripts. Five input scripts were misclassified due to this factor. Define the cumulative recognition rate in the $L$th order as the rate at which the correct character for an input script appears in the list of the first $L$ candidates selected according to the character distance values. Table II shows the results of the cumulative recognition rates. Only the first three candidates need to be selected for the rate to reach 99%.

A prototype recognition system has been constructed on an IBM PC/AT. The C language was used for programming. The average time to recognize an input script is about 2.5 s. Due to the large number of Chinese character categories, most of the processing time was spent in the candidate selection phase. The number of references selected in the candidate character selection phase tends to be large when the strokes of the input script are heavily connected, causing the processing time to increase.

VI. CONCLUDING REMARKS

A new method for on-line handwritten Chinese character recognition has been proposed in this paper. Two kinds of attributed string representations are proposed to represent handwritten Chinese characters. By using the split concept, an input script can be used to split the strokes of each of the references to obtain the desired stroke correspondence in the candidate selection phase. In addition, in the detailed matching phase, the proposed split operation can be employed to solve the input stroke connection problem caused by fast writing. Good experimental results show the effectiveness of the proposed recognition approach in handling the stroke connection problem.
Most errors in the experimental results were caused by the problem that there are too many Chinese characters with similar structures. At least two methods can be used to solve this problem. One is to perform more detailed recognition using more subtle features after the detailed matching phase. The other is to modify the attributed string representations of similar reference characters to make the substrings of their identical parts (such as their common radical) the same. This will enhance the discrimination power of the distances between the input script and similar reference characters.

In addition, the constraint of correct stroke orders, which increases burden to the user and limits writing speed, imposed by the proposed database will increase the recognition time. It also seems difficult to handle all possible stroke orders of different users. Because the stroke order of each character seems stable for each particular user, it is feasible to use the proposed approach to design a writer-dependent recognition scheme more flexible.

**REFERENCES**


---

**Abstract**—Let $\Lambda$ be a finite plaintext alphabet and $\Gamma$ be a cypher alphabet with the same cardinality as $\Lambda$. In all one-to-one substitution cyphers, there exists the property that each element in $\Gamma$ maps onto exactly one element in $\Lambda$ and vice versa. This mapping of $\Gamma$ onto $\Lambda$ is represented by a function $T^\gamma$, which maps any $\gamma \in \Gamma$ onto some $\lambda \in \Lambda$ (i.e., $T^\gamma(\lambda) = \lambda$). In this correspondence, we consider the problem of learning the mapping of $T^\gamma$ (or its inverse $(T^\gamma)^{-1}$) by processing a sequence of cypher text. The fastest reported method to achieve this is an elegant relaxation scheme due to Peleg et al. [8, 9] that utilizes the statistical information contained in the unigrams and trigrams of the plaintext language. In this correspondence, we present a new learning automation solution to the problem called the cypher learning automaton (CLA). The proposed scheme is fast, and the advantages of the scheme in terms of time and space requirements over the relaxation method have been listed. The correspondence contains simulation results comparing both cypher-breaking techniques.

**Index Terms**—Cryptography, learning automata, relaxation methods, substitution cyphers.