Handwritten Chinese Character Recognition With Attributed Graph Matching

Siwei Lu and Ying Ren
Department of Computer Science
Memorial University of Newfoundland
St. John's, Newfoundland, Canada, A1C 5S7

Ching Y. Suen
Department of Computer Science
Concordia University
Montreal, Quebec, Canada, H3G 1M8

ABSTRACT

The paper presents a new method for the recognition of handwritten Chinese characters. A structural representation called hierarchical attributed graph representation(HAGR) is introduced to describe handwritten Chinese characters. With HAGR, the recognition process becomes a simple process of graph matching. A mapping cost function from a candidate to a model graph is introduced. This approach can tolerate the variations of HAGR which reflect the instabilities or variabilities of handwritten Chinese characters resulting from different writing styles. Several rules have been introduced to order the vertices of the graphs to avoid the combinatorial explosion in graph matching. In addition, the database of the character models is arranged in a search-tree structure. For a candidate character, the search process to find a corresponding model character can be divided into a number of simple and local decisions at different levels of the tree. Thus the efficiency and accuracy of the matching process can be greatly improved.

Keywords: Chinese character recognition, Search tree, Hierarchical attributed graph.

I. Introduction

The recognition of handwritten Chinese character(HCCR) is complicated and difficult due to the large number, complex structure of the characters and the infinite variety of character shape, style, position of the branches, and direction and length of the stroke. Typical approaches for HCCR are correlation matching[1], background feature distribution[2], background analysis[3], and stroke analysis[4].

For the correlation matching, some forms of normalization are required. The normalization has limited ability to compensate for the variety on the writing styles. The last three approaches depend on the features such as stroke length, stroke direction, stroke sequence, and connection relation of strokes [5]. But the features are unstable to some extent, as the writing may be different from person to person. To overcome the difficulties, one method is to shift the burden to the users by setting the constraints on character writing[6]. Not all users will accept this and some may have difficulties observing the specified rules. The other methods increase the number of models for each character in the database to allow for the variation caused by the instability of the features. However, this increases the character database of the Chinese language enormously. For example, the handwritten Kanji database ETL8 contains 152960 samples of 881 different Kanji characters. Each character category has 160 samples written by different people[7]. With such a large character set, the matching process may become very time-consuming.

In this paper, a new method is proposed which is not sensitive to the writing styles and offers users with high flexibility for effective recognition of Chinese characters. A recognition method which is insensitive to different writing styles is obtained by finding the stable features of the characters. However, in the recognition process, some of the unstable features can not be discarded, such as the orientation of the strokes. We develop a hierarchical attributed graph representation (HAGR) to represent and describe both invariant features and unstable features. A mapping cost function from a candidate graph to a model graph is also introduced. This approach provides some tolerations to the variation of characters handwritten with different styles. For graph matching, several rules are applied to order the vertices of the graph to avoid the combinatorial explosion in matching.

Furthermore, the database is organized into a search-tree structure according to the spatial relation between branches, the number of strokes and the geometric configuration of strokes in each branch. Using the HAGR representation of character models and a tree organization of the database, the efficiency of the matching processing can be greatly improved.

II. Preprocessing

Preprocessing includes seven steps: thresholding, normalizing, thinning, stroke segment extracting, stroke merging, stroke identifying and stroke grouping. Local thresholding method is used to obtained the binary image of an input character. The binary image is normalized to a size of 60 by 60 pixels. A thinning algorithm is adopted to obtain the skeleton of the character.

While extracting segments, the key-points such as end-points, corner-points and cross-points on the skeleton are determined. The segment between two key-points then can be extracted.

For each line segment, the orientation can be determined by $tan\alpha = (y_2 - y_1)/(x_2 - x_1)$, where (x_1, y_1)

and (x_2, y_2) are the two key-points of the segment. The orientation of a line segment is:

1) horizontal(H), if $-22.5° \leq \alpha < 22.5°$.
2) right diagonal(RD), if $22.5° \leq \alpha < 67.5°$,
3) vertical(V), if $67.5° \leq \alpha < 112.5°$,
4) left diagonal(LD), if $112.5° \leq \alpha < 157.5°$.

Two segments can be merged together to form one stroke if the segments share a common end-point and have the same orientation. The remaining segments which do not merge together to form strokes will remain as the strokes.

According to the relationships among them, the strokes are grouped into branches for HAGR construction.

III. Construction of HAGR

After preprocessing, the branch attributed graph can be constructed based on the strokes that make up the branch, and a hierarchical attributed graph(HAGR) for the character can be constructed based on the branches. The basic concepts about the HAGR are given first.

A. An attribute set and an attributed graph

Definition 1: An *attribute set* is an m-tuple $< p_1, p_2, \ldots, p_i, \ldots, p_m >$ where each element in the tuple is an attribute pair.

Definition 2: An attributed graph is a graph $G_a = (V_a, E_a)$ where $V_a = \{v_1, \ldots, v_p, \ldots, v_q, \ldots, v_m\}$ is a set of attributed vertices and $E_a = \{\ldots, e_{pq}, \ldots\}$ is a set of attributed edges. The edge e_{pq} connects vertices v_p and v_q with an attributed relation.

Definition 3: A *branch attributed graph* $G_b=(V_b, E_b)$ is an attributed graph for representing a branch in a character. (1) V_b is a set of attributed vertices representing the strokes in the branch. (2) E_b is the set of attributed edges representing the relations between the strokes. The different types of relations between two strokes are tabulated as follows:

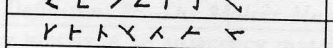
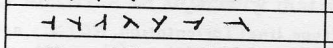
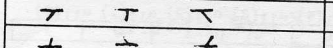
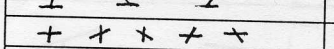
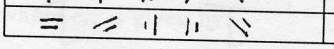
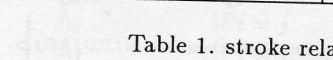
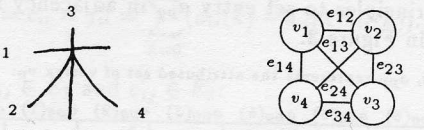
stroke relation	representation
	L
	⊢
	⊣
	⊤
	⊥
	X
	‖

Table 1. stroke relation

For example, the attributed graph of branch "木" is shown in Figure 1.

Definition 4: A *Hierarchical attributed graph* is an attributed graph $H=(V_H, A_H)$ which represents a Chinese character. The vertex represents a branch and each arc represents the spatial relation between two adjacent branches (left-right, top-bottom, and boundary-center).



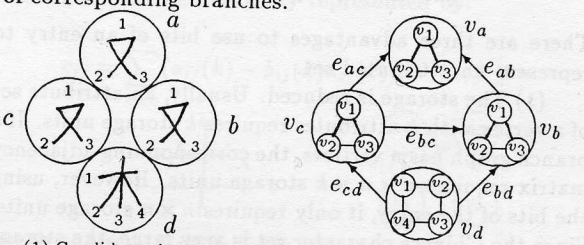
a) Branch b) Branch attributed graph

$G_b=(V_b,E_b)$ where $V_b=\{v_1,v_2,v_3,v_4\}$, $E_b=\{e_{12},e_{13},e_{14},e_{23},e_{24},e_{34}\}$
$v_1=<(type,line),(orientation,H)>, e_{12}=<(relation,\top)>,$
$v_2=<(type,line),(orientation,RD)>, e_{13}=<(relation,X)>,$
$v_3=<(type,line),(orientation,V)>, e_{14}=<(relation,\top)>,$
$v_4=<(type,line),(orientation,LD)>, e_{23}=<(relation,\dashv)>,$
$e_{24}=<(relation,L)>$, $e_{34}=<(relation,\vdash)>$.

Figure 1. Branch attributed graph representation.

A complete HAGR of the Chinese character "  " is given in Figure 2.

The character consists of four branches: three "又" at the top and one "木" at the bottom. The upper right of the figure shows a HAGR of the character. The four big circles represent four branch attributed graphs of corresponding branches.



(1) Candidate character (2) HAGR for the character

$H=(V_H,A_H), V_H=\{v_a,v_b,v_c,v_d\}$, $A_H=\{e_{ab},e_{ac},e_{bc},e_{cd},e_{bd}\}$
$v_a=<(number\ of\ strokes,3)>, e_{ab}=<(spatial\text{-}relation,top\text{-}bottom)>,$
$v_b=<(number\ of\ strokes,3)>, e_{bd}=<(spatial\text{-}relation,top\text{-}bottom)>,$
$v_c=<(number\ of\ strokes,3)>, e_{ac}=<(spatial\text{-}relation,top\text{-}bottom)>,$
$v_d=<(number\ of\ strokes,4)>, e_{cd}=<(spatial\text{-}relation,top\text{-}bottom)>,$
$e_{bc}=<(spatial\text{-}relation,left\text{-}right)>$.

Figure 2. A candidate character and its HAGR.

B. The data structure of HAGR

Definition 5: An *attributed adjacency matrix* is an incidence matrix of an attributed graph $G_a = (V_a, E_a)$ of order m with $\overline{v}_i$ as diagonal entries and $\overline{e}_{ij}$ as nondiagonal entries. The diagonal entry $\overline{v}_i$ represents the attribute set of vertex v_i and nondiagonal entry $\overline{e}_{ij}$ represents that of edge e_{ij}. If there is no edge between vertices v_i and v_j then $\overline{e}_{ij}$ can be set to 0. This is written as

$$I = [a_{pq}]_{m \times m}$$

$$a_{pq} = \begin{cases} 0 & \text{if no edge between } v_p \text{ and } v_q \\ \overline{v}_p & \text{if } p = q \\ \overline{e}_{pq} & \text{if } p \neq q, \quad e_{pq} \in A_a \end{cases}$$

where I is a symmetric matrix.

The bits of the entries in the matrix describe the attribute set associated with the vertex v_i or edge e_{ij}. Four orientations(H, LD, V, RD) and two stroke types(line and dot) are used to represent the attribute set of a

vertex. Seven types of the spatial relations between two strokes are used to represent the attribute set of an edge. The principles to set entry a_{pq} in adjacency matrix are given in Figure 3.

If $p = q$, a_{pp} represents the attributed set of vertex v_p:

$a_{pq}(6)$	$a_{pq}(5)$	$a_{pq}(4)$	$a_{pq}(3)$	$a_{pq}(2)$	$a_{pq}(1)$	$a_{pq}(0)$
	LD	V	RD	H	-	—

$a_{pq}(6) = 0$. If attribute value of orientation of v_p = left diagonal, then $a_{pp}(5) = 1$ else $a_{pp}(5) = 0$. Similarly other bits are set to the proper values.

If $p \neq q$ and $e_{pq} \in E$, a_{pq} represents the attributed set of edge e_{pq}

$a_{pq}(6)$	$a_{pq}(5)$	$a_{pq}(4)$	$a_{pq}(3)$	$a_{pq}(2)$	$a_{pq}(1)$	$a_{pq}(0)$
‖	X	⊥	⊤	⊣	⊢	L

If relation of e_{pq} is "‖", then $e_{pq}(6) = 1$ else $e_{pq}(6) = 0$. Similarly other bits are set to the proper values

Where $a_{pq}(i)$ represents the ith right most bit of the entry a_{pq} in the adjacency matrix.

Figure 3. Bits of an entry in the adjacency matrix and the corresponding attributed set.

There are three advantages to use bits of an entry to represent an attributed set.

(1) The storage is reduced. Usually, an attribute set of a vertex with k attributes requires k storage units. If a branch graph has n vertices, the corresponding adjacency matrix requires $n \times n \times k$ storage units. However, using the bits of the entry, it only requires $n \times n$ storage units. Since the Chinese character set is very large, the storage size can be significantly reduced.

(2) The efficiency of graph matching is improved. By using bits of the entry to represent the attributed set, the graph matching becomes a comparison of the corresponding bits in two entries.

(3) The variation related to the stroke type or orientation is allowed without having to increase the number of models for each character in the database. For example, the fourth stroke "木" of the branch in Figure 4 can be written as a dot or a line depending on the writing habits or styles of the individual. A simple way to represent the variations is to set both the 0th and 1st rightmost bits of a_{44} equal to 1.

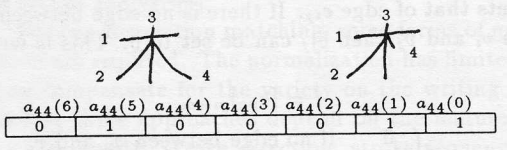


Figure 4. Using entry of adjacency matrix to represent the variations of stroke in model.

In the adjacency matrix of a hierarchical attributed graph, the diagonal entry h_{ii} stores the attributed value of the vertex v_i which represents a branch in the character. The attributed value is the number of strokes in the branch. The nondiagonal entry stores the attributed value of edge e_{ij} which represents the spatial relation of the branches.

IV. Recognition of handwritten Chinese characters

Once the HAGR of the input candidate handwritten characters is constructed, it can be compared with the model HAGR in the character database. Thus the recognition of handwritten Chinese characters becomes a graph matching problem. A mapping cost function between the adjacency matrices is introduced to the graph matching. With the cost function, the HAGR of a candidate character can be matched with its model HAGR which may describe several variations of the character. To make the match fast and efficient, two strategies are used: (1) apply vertex ordering in the HAGR description, and (2) arrange the database of the models as a search-tree structure.

A. Branch attributed graph matching

Let $G_c = (V_1, E_1)$ and $G_m = (V_2, E_2)$ be a candidate branch attributed graph and a model attributed graph respectively. A one-to-one correspondence Γ is assigned to $V_1 = \{v_1, \ldots, v_n\}$ and $V_2 = \{v'_1, \ldots, v'_n\}$. The mapping cost function is the difference between G_c and G_m under the one-to-one correspondence Γ. Let $A = [a_{ij}]_{n\times n}$ and $B = [b_{ij}]_{n\times n}$ be two adjacency matrices corresponding to G_c and G_m.

Definition 6: The Vertex Mapping Cost f_{ij} from the vertex $v_i \in V_1$ to the vertex $v'_j \in V_2$ is:

$$f_{ij} = \sum_{k=0}^{6} (a_{ii}(k) - b_{jj}(k) a_{ii}(k))$$

where $a_{ii}(k)$ and $b_{jj}(k)$ are the kth rightmost bits of their diagonal entries.

In order to allow reasonable variations in stroke orientation, stroke type, and spatial relation between strokes, all variations of the character model are represented by the entries in the adjacency matrix. For the candidate character, only one of the variations is represented in the corresponding entry of the branch attributed matrix as

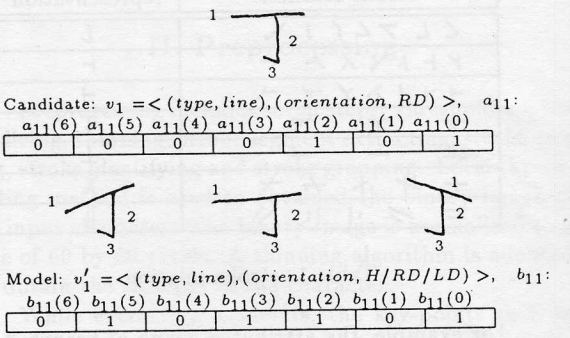


Figure 5. Vertex matching

described in previous section. If the stroke(vertex v_i) of a candidate character is one of the variations of the corresponding stroke(vertex v'_j), then the Vertex Mapping Cost f_{ij} is equal to zero.

Figure 5 shows an example of VMC of stroke v_1 in candidate branch " ⅂ " and stroke v_1' in its model. The top stroke v_1' in the model is a line stroke which has three variations: H orientation " — ", RD orientation " ⁄ ", and LD orientation " \ ". Hence, entry b_{11} associated with v_1' in B is set to "0101101". The top stroke v_1 in the candidate character has a horizontal orientation. Entry a_{11} associated with v_1 in A is set to " 0001001 ".

Vertex mapping cost of v_1 and v_1' :

$$f_{11} = \sum_{k=0}^{6}(a_{11}(k) - b_{11}(k)a_{11}(k)) = 0.$$

So vertex v_1 matches with vertex v_1'. For the VMC from v_2 to v_1', entry a_{22} associated with v_2 is "0010001", the cost is:

$$f_{21} = \sum_{k=0}^{6}(a_{22}(k) - b_{11}(k)a_{22}(k)) = 1.$$

This means that vertex v_2 does not match with v_1', because the vertical orientation of v_2 is not one of the variations of v_1'.

Definition 7: The Edge Mapping Cost d_{ij} $(i \neq j)$ is defined as:

$$d_{ij} = \sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)), if\ e_{ij} \in E_1\ and\ e'_{ij} \in E_2$$

where $a_{ij}(k)$ and b_{ij} are the kth rightmost bits of their nondiagonal entries.

Similar to the VMC, the Edge Mapping Cost d_{ij} is equal to zero, if the candidate character belongs to one of the variations in the model.

Proposition 1: $f_{ii} \geq 0$, $d_{ij} \geq 0$.

Proof: For $0 \leq k \leq 6$, both $a_{ii}(k)$ and b_{ii} can only have values 1 or 0.

If $a_{ii}(k) = 0$, $a_{ii}(k) - b_{ii}(k)a_{ii}(k) = 0 - b_{ii}(k) \times 0 = 0$.

If $a_{ii}(k) = 1$, $a_{ii}(k) - b_{ii}(k)a_{ii}(k) = 1 - b_{ii}(k) \times 1 \geq 0$.

So for $0 \leq k \leq 6$, $a_{ii}(k) - b_{ii}(k)a_{ii}(k) \geq 0$.

Hence, $f_{ii} = \sum_{k=0}^{6} a_{ii}(k) - b_{ii}(k)a_{ii}(k) \geq 0$.

In the same way, $d_{ij} \geq 0$ can be proved.

Definition 8: The Matrix Mapping Cost from adjacency matrix A to adjacency matrix B is defined as:

$$MMC(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}$$

where c_{ij} is given by:

$$c_{ij} = \begin{cases} f_{ij} & \text{if } i = j \\ d_{ij} & \text{if } i \neq j,\ e_{ij} \in E_1 \text{ and } e'_{ij} \in E_2 \\ \sum_{k=0}^{6} a_{ij}(k) & \text{if } i \neq j,\ e_{ij} \in E_1 \text{ and } e'_{ij} \notin E_2 \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 2: c_{ij} can be uniformly represented by:

$$c_{ij} = \sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)), 1 \neq i, j \neq n.$$

Proof: if i = j :

$$c_{ij} = c_{ii} = f_{ii} = \sum_{k=0}^{6}(a_{ii}(k) - b_{ii}(k)a_{ii}(k)).$$

If $i \neq j$, $e_{ij} \in E_1$ and $e'_{ij} \in E_2$:

$$c_{ij} = d_{ij} = \sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)).$$

if $i \neq j$, $e_{ij} \in E_1$ and $e'_{ij} \notin E_2$, then $b_{ij}(k) = 0$ for $0 \leq k \leq 6$.

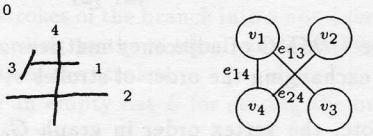
$$\sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)) = \sum_{k=0}^{6} a_{ij}(k) = c_{ij}.$$

If $i \neq j$, $e_{ij} \notin E_1$, $a_{ij}(k) = 0$ for $0 \leq k \leq 6$.

$$\sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)) = \sum_{k=0}^{6}(0 - b_{ij}(k) \times 0) = 0 = c_{ij}.$$

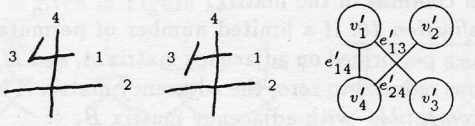
Hence, c_{ij} can be uniformly represented by:

$$c_{ij} = \sum_{k=0}^{6}(a_{ij}(k) - b_{ij}(k)a_{ij}(k)), 1 \leq i, j \leq n.$$



candidate branch: $G_c=(V_1,E_1)$, $V_1=\{v_1,v_2,v_3,v_4\}$, $E_1=\{e_{13},e_{14},e_{24}\}$, $v_1=<(type,line),(orientation,H)>$, $v_2=<(type,line),(orientation,H)>$, $e_{13}=<(relation,\vdash)>$, $v_3=<(type,line),(orientation,RD)>$, $e_{14}=<(relation,X)>$, $v_4=<(type,line),(orientation,V)>$, $e_{24}=<(relation,X)>$,

$$A = [a_{ij}] = \begin{pmatrix} 0000101 & 0000000 & 0000010 & 0100000 \\ 0000000 & 0000101 & 0000000 & 0100000 \\ 0000010 & 0000000 & 0001001 & 0000000 \\ 0100000 & 0100000 & 0000000 & 0010001 \end{pmatrix}$$



Model: $G_m=(V_2,E_2)$, $V_2=\{v'_1,v'_2,v'_3,v'_4\}$, $E_2=\{e'_{13},e'_{14},e'_{24}\}$, $v'_1=<(type,line),(orientation,H)>$, $v'_2=<(type,line),(orientation,H)>$, $e'_{13}=<(relation,\vdash\ /L)>$, $v'_3=<(type,line),(orientation,RD)>$, $e'_{14}=<(relation,X)>$, $v'_4=<(type,line),(orientation,V)>$, $e'_{24}=<(relation,X)>$,

$$B = [b_{ij}] = \begin{pmatrix} 0000101 & 0000000 & 0000011 & 0100000 \\ 0000000 & 0000101 & 0000000 & 0100000 \\ 0000011 & 0000000 & 0001001 & 0000000 \\ 0100000 & 0100000 & 0000000 & 0010001 \end{pmatrix}$$

Figure 6. Adjacency matrices A and B

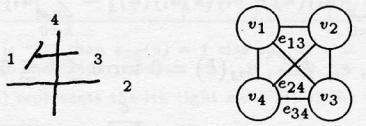
In Figure 6, the attributed graph of a candidate and that of a model are represented in a matrix form. The Matrix Mapping Cost $MMC(A,B)$ can be used to measure the difference between the matrices.

$$c_{11} = \sum_{k=0}^{6}(a_{11}(k) - b_{11}(k)a_{11}(k)) = 0$$

In the same way, $c_{ij} = 0$ for $0 \le i, j \le 6$. Hence:

$$MMC(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} = 0$$

In the above example, the candidate branch "丰" is one of the variations of the model "丰" and "丰". Hence, $MMC(A,B)$ is equal to zero. However, if the vertices in the graph of the candidate branch are ordered differently, $MMC(A,B)$ cannot be equal to zero. For instance, exchanging the order of stroke v_1 with stroke v_3 will cause the 3rd row and column to switch with the 1st row and 1st column in A(see Figure 7).



$$A = [a_{ij}] = \begin{pmatrix} 0001001 & 0000000 & 0000010 & 0000000 \\ 0000000 & 0000101 & 0000000 & 0100000 \\ 0000010 & 0000000 & 0000101 & 0100000 \\ 0000000 & 0100000 & 0100000 & 0010001 \end{pmatrix}$$

$$MMC(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} = 4$$

Figure 7. MMC of adjacency matrices A and B after exchanging the order of strokes v_1 and v_3.

Therefore, the vertex order in graph G_c should be arranged properly. To order the vertices, an operation called permutation of the adjacency matrix A is introduced. The MMC and permutation of adjacency matrix can be used to find a match from G_c to G_m.

Definition 9: A Permutation of adjacency matrix A exchanges the ith and jth rows after exchanging the ith and jth columns in the matrix.

Definition 10: If a limited number of permutations have been performed on adjacency matrix A, and $MMC(A,B)$ has reduced to zero, the adjacency matrix A is said to be *compatible* with adjacency matrix B.

Definition 11: Branch attributed graph G_c is said to *match* branch attributed graph G_m if and only if A is compatible with B.

As the bitwise technique is used to represent the attribute set of the vertices and edges in the adjacency matrices, the concept of unit matrix and operations on the adjacency matrices is different from those of traditional matrices which contain numbers as their entries.

Definition 12: The *unit matrix* is a adjacency matrix with 1111111 as its diagonal entries and 0000000 as its nondiagonal entries.

Definition 13: The matrices obtained by performing primitive row or column operation once on the unit matrix is called a primitive matrix P_{ij}.

For example, performing the row operation on the 2nd row and 3rd row of unit matrix, primitive matrix P_{23} will be as following:

$$P_{23} = \begin{pmatrix} 1111111 & 0000000 & 0000000 & 0000000 \\ 0000000 & 0000000 & 1111111 & 0000000 \\ 0000000 & 1111111 & 0000000 & 0000000 \\ 0000000 & 0000000 & 0000000 & 1111111 \end{pmatrix}$$

Definition 14: The *multiplication* $*$ of two adjacency matrices $F = [f_{ij}]_{n\times n}$ and $L = [l_{ij}]_{n\times n}$ is defined as:

$$F * L = [\sum_{t=1}^{n} (f_{ik} \otimes l_{kj})]_{n\times n}$$

where $\sum_{t=1}^{n} x_t = x_1 \oplus x_2 \oplus \ldots \oplus x_n$, $\otimes$ is "bitwise and" operator and $\oplus$ is " bitwise or" operator.

Proposition 3: A permutation on A is to multiply A on both sides with a primitive matrix.

Proposition 4: The adjacency matrix A is compatible with the adjacency matrix B if and only if there are primitive matrices $P_1, \ldots, P_l$ such that

$$MMC(P_l * P_{l-1} * \ldots * P_1 * A * P_1 * P_2 * \ldots * P_l, B) = 0$$

Furthermore from Definition 11, G_c matches with G_m.

B. Example for branch graph match

Figure 8 shows the attributed branch graph G_c of branch " 木 " and the attributed graph G_m of its model. Their adjacency matrices are A and B:

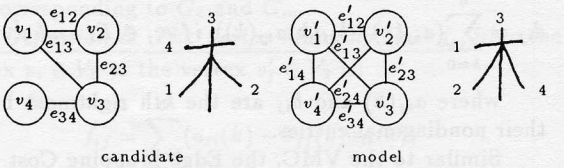


Figure 8. Branch " 木 " and its model

$$A = \begin{pmatrix} 0001001 & 0000001 & 0000100 & 0000000 \\ 0000001 & 0100001 & 0000010 & 0000000 \\ 0000100 & 0000010 & 0010001 & 0100000 \\ 0000000 & 0000000 & 0100000 & 0000101 \end{pmatrix},$$

$$B = \begin{pmatrix} 0000101 & 0001000 & 0100000 & 0001000 \\ 0001000 & 0001001 & 0000100 & 0000001 \\ 0100000 & 0000100 & 0010001 & 0000010 \\ 0001000 & 0000001 & 0000010 & 0100011 \end{pmatrix}.$$

$$MMC(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} = 11 \ge 0.$$

In the first iteration, primitive matrix P_{14} is selected.

$P_{14} * A * P_{14} =$

$$\begin{pmatrix} 0000101 & 0000000 & 0100000 & 0000000 \\ 0000000 & 0100001 & 0000010 & 0000001 \\ 0100000 & 0000010 & 0010001 & 0000100 \\ 0000000 & 0000001 & 0000100 & 0001001 \end{pmatrix}$$

$$MMC(P_{14} * A * P_{14}, B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} = 6.$$

In the second iteration, primitive matrix P_{24} is selected.

$P_{24} * P_{14} * A * P_{14} * P_{24} =$

$$\begin{pmatrix} 0000101 & 0000000 & 0100000 & 0000000 \\ 0000000 & 0001001 & 0000100 & 0000001 \\ 0100000 & 0000100 & 0010001 & 0000010 \\ 0000000 & 0000001 & 0000010 & 0100001 \end{pmatrix}$$

$$MMC(P_{24} * P_{14} * A * P_{14} * P_{24}, B) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} = 0.$$

So matrix A is compatible with matrix B and G_c matches with G_m.

C. Ordering the vertices in branch graph

In order to find a match from G_c to G_m, in the worst case all possible sequences of permutations on A have to be examined to determine if a sequence of permutations with corresponding primitive matrices $P_1,\ldots,$ P_l makes $MMC(P_l * \ldots * P_1 * A * P_1 \ldots P_l, B) = 0$. Unfortunately this is a combinatorial problem. In our approach, the geometrical information of the strokes of a branch is used to order the vertices in the branch attributed graph, and the time to find the match by performing the permutations on A will be reduced significantly.

For a stroke, its equation $Ax + By + C = 0$ can be determined by the coordinates of its two end-points of the stroke. For determining the relative order between two strokes, the sign of B is chosen to be positive. In case $B = 0$, the sign of C is chosen to be positive.

Suppose that the equation of stroke a with two end-points (x_{a1}, y_{a1}) and (x_{a2}, y_{a2}) is $A_a x + B_a y + C_a = 0$, and the equation of stroke b with two end-points (x_{b1}, y_{b1}) and (x_{b2}, y_{b2}) is $A_b x + B_b y + C_b = 0$. There are four rules to determine the relative order of the two strokes:

Rule 1: Projections of strokes a and b on X or Y axis do not overlap. If $min\{y_{a1}, y_{a2}\} \geq max\{y_{b1}, y_{b2}\}$, then stroke a is ordered before stroke b else if $max\{x_{a1}, x_{a2}\} \leq min\{x_{b1}, x_{b2}\}$, then stroke a is ordered before b.

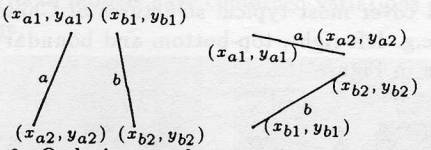


Figure 9. Ordering strokes a and b using rule 1.

Rule 2: Strokes a and b do not cross each other and their projections on the X axis overlap. Two end-points (x_{b1}, y_{b1}) and (x_{b2}, y_{b2}) of the stroke b lie on the opposite sides of stroke a (see Figure 10). This can be mathematically represented as: $(A_a x_{b1} + B_a y_{b1} + C_a)(A_a x_{b2} + B_a y_{b2} + C_a) < 0$.

If $A_b x_{a1} + B_b y_{a1} + C_b > 0$ and $A_b x_{a2} + B_b y_{a2} + C_b > 0$, then a is ordered before b as illustrated on the left hand side of Figure 10. If $A_b x_{a1} + B_b y_{a1} + C_b < 0$ and $A_b x_{a2} + B_b y_{a2} + C_b < 0$, then a is ordered after b as shown on the right hand side of Figure 10.

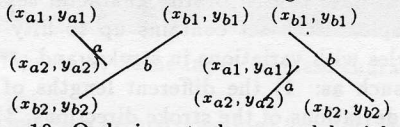


Figure 10. Ordering strokes a and b with rule 2.

Rule 3: Strokes a and b intersect each other. The strokes are ordered according to their orientations: H, RD, V and LD. For example, the stroke a is horizontal(H) and the stroke b is right diagonal(RD), then a is ordered before b (See Figure 11). If the code of a is the same as that of b and the inclined angle of b is greater than that of stroke a, then a is ordered before b.

Figure 11. Ordering strokes a and b with rule 3.

Rule 4: Stroke a and stroke b can not be ordered by rule 1-3. If $A_b x_{a1} + B_b y_{a1} + C_b > 0$ and $A_b x_{a2} + B_b y_{a2} + C_b > 0$, then a is ordered before b (see Figure 12).

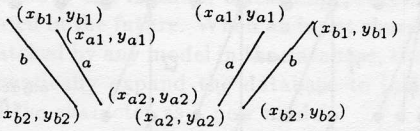


Figure 12. Ordering strokes a and b with rule 4.

Applying the ordering rules, a set of strokes of a branch can be sorted to create an order list by the following procedure:

1) Sort the strokes of the branch into a non-increasing sequence T according to the vertical coordinates of the upper end-points of the strokes.

2) Initialize an empty list L for storing the ordered list.

3) While ($T \neq empty$) do

a) remove the first stroke f from T and put it at the end of the list L.

b) use ordering rules to insert stroke f into an appropriate place in L.

To illustrate the stroke ordering algorithm, an example is given in Figure 13.

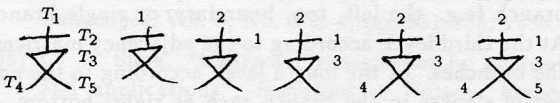


Figure 13. Ordering strokes in a branch.

D. Structure of the Database

The models for Chinese characters recognition are stored in a database at two levels. The first level records the vertices and arcs of a HAGR. At this level the attributes and their values for each vertex as well as for

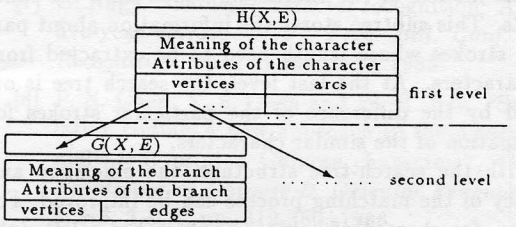


Figure 14. Data-structure of HAGR

each arc are stored. At the second level, corresponding to each vertex of the HAGR, the branch attributed graph is stored(see Figure 14).

Since the Chinese character set is very large, in order to speed up the matching process, the database is organized as a search-tree structure depicted in Figure 15.

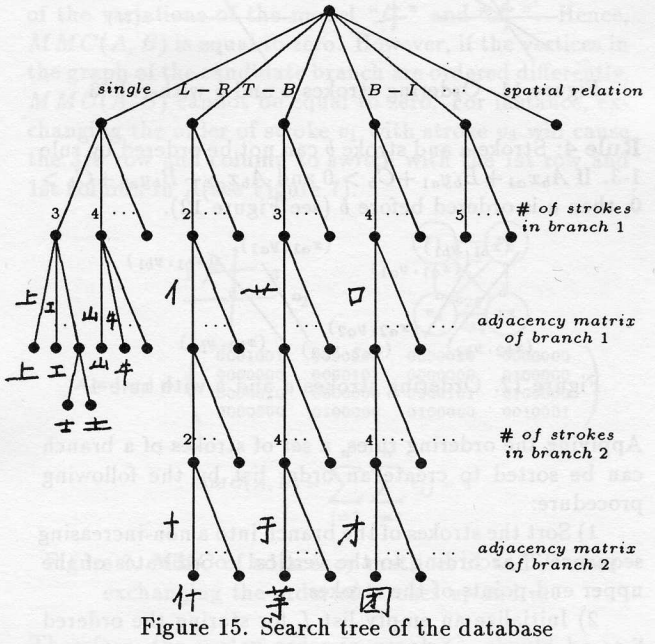


Figure 15. Search tree of the database

At the first level of the tree, the models of the character are classified into several classes according to the spatial distribution of the branches in the character. Namely, left branch with right branch(L-R), top branch with bottom branch(T-B), boundary branch with center branch(B-I), or single branch. At the second level, the models are organized according to the number of strokes in the branch (e.g. the left, top, boundary, or single branch). At the third level, according to the adjacency matrices of the branches. At the fourth level, according to the number of strokes in the branch such as right, bottom , or center-branch. At the fifth level, the models of character are arranged by the adjacency matrices of the branches.

There are few pairs of Chinese characters which have same HAGRs. They can only be distinguished by the difference between length of particular strokes. For example, the characters "土" and "士". In order to identify such characters, a subtree is attached on of the nodes with the models of the characters which have the same HAGRs. This subtree stores the information about particular strokes whose length need to be extracted from the characters. At the last level, the search tree is organized by the difference of the particalar strokes for identification of the similar characters.

With the search-tree structure, the efficiency and accuracy of the matching process can be improved. For instance, for character" 相 " which has a left-right structure, models without left-right structure can be excluded during the search in the database.

E. Character recognition

Definition 14: For a hierarchical attributed graph H_c of a candidate handwritten character, and a hierarchical attributed graph H_m of a model, H_c is said to match with H_m if the following necessary conditions are satisfied:

1) Each vertex v_i in H_c one to one corresponds to a vertex v_j in H_m such that the branch attributed graph B_i represented by $v_{i'}$ is matched by the branch attributed graph $B_{j'}$ represented by v_j.

2) Each arc e_{kl} in H_c one to one corresponds to an arc $e_{k'l'}$ in H_m in such way that spatial relation between v_k and v_l is the same as that between $v_{k'}$ and $v_{l'}$.

The character recognition procedure is briefly summarized as follows:

(1) Preprocessing.

(2) Construct all the branch attributed graphs for all the branches of the input character.

(3) Construct the hierarchical attributed graph H_i for the input character.

(4) Search the database to find a match between H_i and H_m. If a match is found, the character has been recognized.

V. Experimental result and analysis

The recognition experiment has been conducted on a VAX 11/780 using UNIX. During the experiment, our model database consists of 460 different character classes which cover most typical structures of Chinese characters, e.g. left-right, top-bottom and boundary-center as shown in Figure 16.

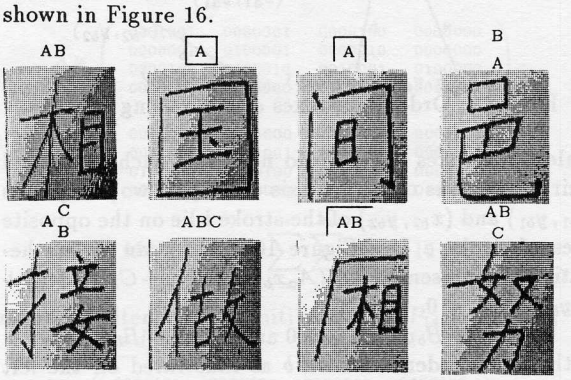


Figure 16. Different structures of Chinese characters.

The test data consist of fifty character sets written by ten people. Each set contains up to fifty different writing styles with variations in strokes and stroke connections; such as: 1) the different lengths of strokes, 2) certain deviations of the stroke directions, 3) various stroke connections, 4) the different stroke types(e.g. dot or line) according to the different writing habits. The recognition rate can still be maintained at over 90% in

the difficult situations. The rejection rate is 8% and the misclassification rate is much less than 1%.

From the experiment, very interesting results are achieved by the proposed method.

(1) Some Chinese characters such as (甲,由), (已,已), (叮,可), (土,士), (未,未) and (日,曰) are considered difficult to be identified by existing methods [5], [8]. But, they are correctly recognized by our algorithm with no difficulty.

(2) Characters with different writing styles such as (木,木), (本,本), (攵,攵) and (古,古) can be correctly recognized as same characters.

The reasons are anlyzed and highlighted below:

Class-a: In groups (甲,由) and (已,己), the different connections for characters of each group with similar configurations are differently described in their HAGRs and can be used to distinguish them. For instance in group "甲,由", the connection between top horizontal stroke and middle vertical stroke in character "甲" represented as "丁". is different from the connection in character "由" which has no relation "".

Class-b: In group (叮,可), the spatial relation between branch "口" and "丁" in character "叮" is L-R, while that in character "可" is B-I.

Class-c: The characters with similar configurations in the groups (土,士), (未,末) and (日,曰) can be correctly recognized, by the special arrangements in the search-tree data-base(see section IV.D.).

Class-d: In the groups (木,木), (本,本), (攵,攵), (古,古), the characters written differently with different connections between strokes and different stroke types can still be identified with our method because the HAGRs of models have contained variations of these characters,

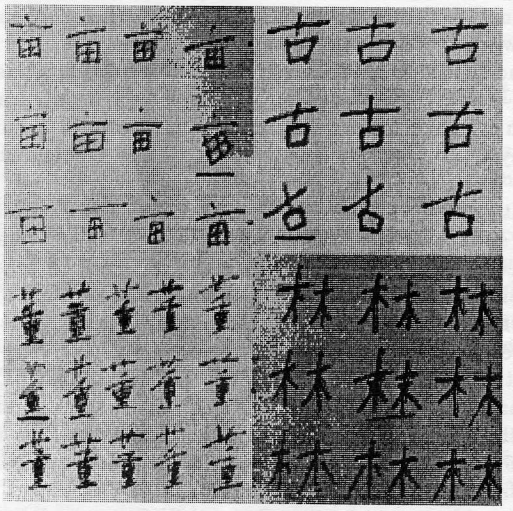


Figure 17. Some input Chinese characters.

The recognition errors(rejection) for some characters are marked by underlines in Figure 17. The errors are due to too much rotations and perfunctory nature of the strokes and by careless writing.

VI. Conclusion

In this paper, we introduce the structural representation(HAGR) to describe the structural information in handwritten Chinese characters. With HAGR, a mapping cost function is used to measure the matching of graphs and a database of character models is constructed as a search-tree. This method overcomes the problem of the stroke variability or stroke connection sensitivity in the traditional recognition methods. Some difficult characters that can not be recognized by most of the tradition methods can still recognized by this method. The search-tree structure database reduces the time spent for the matching process. And it is easy to enlarge the database.

To improve the method, a learning process will be implemented in the future. When an input character can not be matched by any model in the database, the system will automatically expand the database to include the HAGR of the character as a new model.

References

[1] Y. T. Yamashita, K. Higuchi, Y. Yamada, and Y. Haga, "Classification of handprinted Kanji characters by structured segment matching method", *Trans. IECE Japan,* Vol. PRL81-93, Feb. 1982.

[2] S. Naito, K. Komori, and E. Yodogawa "Stroke density feature for handprinted Chinese recognition", *J. IECE Japan,* Vol. J64-D, pp. 757-764, Aug. 1981.

[3] S. Akamatus and K. Komori, "Concentrated structural feature for handprinted characters recognition", *Trans. IECE Japan,* Vol. PRL80-83, Feb. 1981

[4] Tetsuji Morishita. Masahiko Ooura and Yasuo Ishii, "A Kanji recognition method which detects writing errors", *Computer Processing of Chinese & Oriental Languages,* Vol. 3, No. 3 & 4, pp. 351-365, 1988.

[5] C. Y. Suen, "Character recognition by computer and applications", *Handbook of Pattern Recognition and Image Processing,* Academic Press, Inc. pp. 569-585, 1986.

[6] Keh-jiann Chen, Kuo-chun Li and Yeong-long Chang, "A system for on-line recognition of Chinese characters", *Computer Processing of Chinese & Oriental Languages,* Vol. 3, No. 3 & 4, pp. 309-318, 1988.

[7] G. Nagy, "Chinese character recognition: A twenty-five-year retrospective", *Proc. Intl. Conf. Pattern Recognition,* pp. 163-167, Nov. 1988.

[8] P. N. Chen, Y. S. Chen and W. H. Hsu, "Stroke relation coding-A new approach to the recognition of multi-font printed Chinese characters", *Computer Processing of Chinese & Oriental Languages,* Vol. 3, No. 3 & 4, pp. 319-350, 1988.