COMPUTER RECOGNITION OF HANDPRINTED CHARACTERS BASED ON SYNTACTIC AND SEMANTIC ANALYSIS

Y. Mong and C. Y. Suen

Department of Computer Science,Concordia University,1455 de Maisonneuve West, Montreal,Quebec H3G 1M8

ABSTRACT

A system which recognizes handwritten numerals has been designed and implemented. An algorithm has been developed to extract topological features such as loops, concavities and edges in different orientations from the graph representation of a character. These primitives are used to recognize or reconstruct a character image. By combining both syntactic and semantic information such as the position of the primitives in the character frame, the depth and degree of opening of a cavity, this algorithm has become a very powerful classifier of handwritten characters. This paper addresses mainly to the part related to the extraction of topological features and graph encoding.

KEYWORDS: character recognition, syntactic analysis, semantic analysis, feature extraction topological features.

INTRODUCTION - TOPOLOGICAL FEATURES

One of the approaches used in character recognition is the syntactic approach. The techniques of this approach consist of the decomposition of a character into subpatterns or more basic components called primitives according to the structure of the character. In this approach, the extraction of primitives is one of the major component contributing to the performance of the recognition system. The extraction of primitives may be taken as a two step process: (i) the selection of primitives and (ii) the identification of the primitives. Basically, for a given set of characters, in order to evaluate the effectiveness of a selected set of primitives, there are three main criteria. First, the primitives should be the basic structural units which can contribute the maximum information of the character pattern. Second, the primitives should be able to characterize the character. Third, they can be extracted easily.

In this paper, we propose an algorithm to extract a set of topological features shown in Fig. 1 as pattern primitives. There are several advantages of using these topological features as character primitives. First, topological features are basic structures of most character patterns. The use of these structures is also part of the process of human perception of character. Second, with these high-level primitives, extremely long and complicated string-representations are no longer required. A more compact form can be used to describe the character shape. Third,with above advantage, the complexity of syntactic parsing can be reduced in the classification process. Moreover, using the algorithm to be discussed below, the extraction of topological features becomes simple.

GRAPH REPRESENTATION

Reducing a digitized image to a thin line is recommended for line-like structures such as characters. The necessity of obtaining the skeleton of a pattern is usually determined by the method of feature extraction. In this system, a graph form is chosen to represent a character and thus the thinning operation is advantageous in locating graph vertices and end points. This representation is desirable for syntactic analysis since this representation can reduce the complexity of the grammers for the structural description of the entire character.

Preliminary Coding

Before the transformation of the skeleton of the pattern into a graph form, each pixel in the skeleton is given a condition code in order to locate the end points of strokes, branch points (junctions) and detect any significant corner points. A coding scheme is adapted from [Ogawa and Taniguchi]. Consider each point in the skeleton, $P_{i,j}$ with its 3x3 window, with the condition code of $P_{i,j}$ denoted by $C_{i,j}$ which is assigned a value in the set $\{-10, -8, -7, \ldots, -1, 0, 1, \ldots 9\}$, the rules for assigning the code are as follows:

Rule 1: With any of the patterns shown below in Fig. 2 with the central pixel $P_{i,j}$

$$0 \leq C_{i,j} \leq 9 \quad \text{or} \quad -8 \leq C_{i,j} \leq -1$$

outer and inner boundaries.

In the encoding, there are two basic problems to be solved. First, the starting point of the encoding has to be chosen so that all the geometrical structures can be retained in the code. Second, the algorithm should be able to traverse the whole set of vertices of the connected graph. To solve the first problem, we adopt the convention that a vertex is chosen as the starting point according to the following priorities: 1) a terminal node 2) a corner node, and finally 3) a junction node. If more than one potential starting point exists, the upper left-most one is chosen.

To solve the second problem, the following convention is used: starting from the first node, the lowest numbered non-empty direction link (0 to 7) is chosen. For every move, starting from the direction opposite to which it came from, search anti-clockwise in the directions of the eight-direction links, the first non-empty link encountered is chosen as the path to the next node. In this way, the path which returns to the node from where it started is taken as the last resort. To avoid the repetition of the same direction path between two nodes a direction link is disconnected once it has been processed. Given a graph, assume its set of nodes consisting of m distinct element be

$$N_m = \{n_1,\ldots,n_m\}$$

where n_1 is the starting node;

$L_i[k] = j$ denotes that node n_i is linked to n_j in the k-direction.

Let s be the numeric code string to be evaluated;

$D[n_i]$ be the degree of n_i; and
'//' denotes the concatenation of two code strings.

The algorithm can be described in the following pseudo-code.

```
STEP 1  :(* To initialize the starting node and
           direction. *)

        n_i <--- n_1;

        k <--- min{k'/L_i[k'] = 0};

STEP 2  :(* Start a new code string. *)

        s<--- '*' ;

STEP 3  :(* Keep the direction code *)

          IF (D[n_i] > 2) THEN

              set FLAG on;

          j <--- L_i[k];

          L_i[k]<--- 0;

          s <--- s // 'k' ;

STEP 4  :(* Look for the next path. *)

          i <--- j;

          k' <--- (k+4) Modulus 8;

          k <--- (k'+1) Modulus 8;

          WHILE (k ≠ k') AND (L_i[k] = 0)  DO

          k <--- (k+1) Modulus 8;

          IF (k ≠ k') GOTO STEP 3;

STEP 5  :(* Process until whole graph is
           encoded. *)

          s <--- s // '*' ;

          IF (FLAG is on) AND (L_i[k] ≠ 0) GOTO

          STEP 2.
```

There are three cases to be considered in the traversal.
Case 1: For a graph with no junction node, it is sufficent to traverse all the nodes just once in order to describe the character shape. For example, in Fig. 3(a), traversing nodes (1), (2), (3),...,(7) yields the codes 4,6,0,7,5 and 4.
Case 2: For a graph with junction nodes but no inner loops, each junction node must be traversed the number of times equal to its degree. For example, the junction node (3) is traversed four times in Fig. 3(b).
Case 3: For a graph with inner loops as illustrated in Fig. 3(c), when the encoding halts at the starting node (1) with the inner boundary not encoded, then the upper left-most junction node (3) which is not yet traversed completely is taken as the new starting node. With cases 2 and 3, there is more than one code string generated from the graph.

Rule 2: If $P_{i,j}$ is not in any of the above patterns

$$C_{i,j} = -10.$$

With the codes defined above, the meaning of each code can be summarized as follows:

(1) $C_{i,j} = -10$ implies $P_{i,j}$ is a branch point;

(2) $-8 \leq C_{i,j} \leq -1$ implies $P_{i,j}$ is a terminal (end) point;

(3) $C_{i,j}$ in {0,5,6,7,8,9} implies $P_{i,j}$ is likely the turning point of a sharp intrusion;

(4) $C_{i,j}$ in {1,2,3,4} implies $P_{i,j}$ is a point on a horizontal, diagonally right, vertical, diagonally left line respectively

Locating Graph nodes

A graph can be denoted as an ordered pair G (V, E) where V is the set of vertices (or nodes) and E is the set of edges. Each edge is assigned an integer to denote the direction of the edge in the plane. Graph nodes can be classified into three kinds according to their degrees. The terminology is explained as follows:

(1) a terminal node is a vertex of degree 1;
(2) a corner node is a vertex of degree 2;
(3) a junction node is a vertex of degree greater than 2.

An edge from vertex i to vertex j is denoted by E_i^j.

From the coding convention defined above, the terminal node and the junction node can be located from the end points and branch points which either range from -1 to -8, or are -10 respectively. To locate all the graph nodes, the skeleton is traced. Tracing starts from an end point if any, otherwise from a branch point. Tracing halts once an end point or a branch point is encountered which implies that a potential node has been found. For each branch point encountered, in order to find all the branches originated from it, its eight neighboring points are searched through. If another branch point is encountered, the searching process is repeated to ensure that all the branches are located. A branch is said to be found if a point which is neither an end point nor a branch point is encountered during recursive searching.

When each segment is traced, all the points are linked to form a list for further detection of corner ndoes. This process again makes use of the condition codes defined above. The principles of corner detection is to compare the angular difference of successive subsegments at those turning points with a reference threshold so that a curve segment is divided into an appropriate number of straight line subsegment connected by corner nodes.

Quantization of Node Linkage

A straight line segment connecting two nodes is now represented by an edge. The orientation of an angle is quanitized to one of the eight directions shown in Fig. 2 to encode the graph.

Consdier the set of nodes of the graph consisting of m nodes as $N_m = \{n_i / 1 < i < m\}$. Given an edge E_i^j from node n_i to n_j with coordinates (x_i, y_i), (x_j, y_j) respectively, the inclination of E_i^j is given by

$$\theta_i^j = \tan^{-1} \frac{y_j - y_i}{x_j - x_i} \quad \text{if } x_i \leq x_j$$

$$= \tan^{-1} \frac{y_j - y_i}{x_j - x_i} + \Pi \text{ if } x_i > x_j$$

The graph of a character can then be described by the interconnections of graph nodes in terms of link tuples, $\{L_i[k], K = 0..7\}$. The characteristics of the link tuple are listed below:

(1) $L_i[k] = j$ if node n_i is linked to noding in the k-direction, k = 0...7,

(2) $L_i[k] = 0$ if node n_i is linked to no node in the k-direction

(3) $L_i[k] = j \Leftrightarrow L_j[(k=4) \text{ modulus } 8] = i$;

(4) $L_i[k] = j$ and $L_i[k] = n \Leftrightarrow j = n$

Graph Encoding

The encoding of the graph formed generates a numeric code for a character which has the function of guiding the extraction of primitives as well as expressing the relations between primitives. The encoding is accomplished by the traversal of all the vertices in the graph to give a code describing the shape of the character. In the case of the character with no embedding loops, the code describes the boundaries of its graph. On the other hand, for a character which consists of holes, the code describes both the

EXTRACTION OF TOPOLOGICAL FEATURES AS PRIMITIVES

Detection of Primitives

The extraction of primitives is guided by the encoding of the shape of the character. While traversing the graph of a character, the direction change from one edge to another edge is noted. The direction change is defined as the minimal change in a circular sense. For example, a change in the direction code from 0 to 5 in Fig. 4 implies a clockwise change in direction since it gives shorter path than the anti-clockwise change. Then, according to the direction change in an anti-clockwise and clockwise sense, a sign is assigned to the primitive described by these edges as positive or negative respectively.

During the traversal of edges, a primitive is built up by these edges until a change in sign results. Any difference in the sign of direction change indicates that the extraction of the current primitive is completed and a new primitive is expected. With the sign of each primitive and the end nodes (vertices) of this topological feature, its type can be determined. Fig. 5 shows some examples of the detection of primitives.

The algorithm for the detection of primitives is described in Y. Mong's thesis.

CONCLUSION

The graph encoding convention has been developed to extract primitives and define their structural relationship. Meanwhile, with graph representation of a character, semantic information of various types of primitives such as loop location, depth of cavity, shape of cavity can also be evaluated easily.

An experiment was run using the above algorithm for the extraction of primitives of 6000 handwritten numerals. An automatic character recognition system has been implemented and tested on these numerals with 600 samples per class written by 30 different subjects. An overall recognition rate of 99.15% was obtained with a very low misrecognition rate of 0.05%. Experimental details can be found in [Mong].

ACKNOWLEDGEMENT

This work was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Ogawa, H. and Taniguchi, K., "Preprocessing for Chinese Character Recognition and Global Classification of Handwritten Chinese Characters", Pattern Recogntion, Vol. 11, pp. 1-7, 1979.

Mong, Y., "An Experimental Study of the Syntactic Analysis and Recognition of Hand-Written Numerals," M. Comp. Sc. Thesis, Department of Computer Science, Concordia University, August 1982.

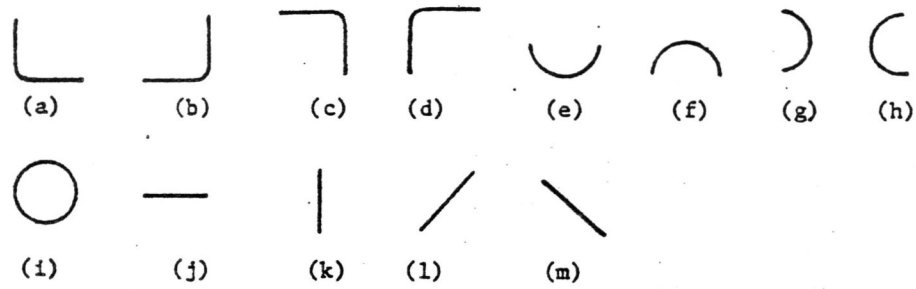


Figure 1. The Set of Primitives Defined by Topological Features.

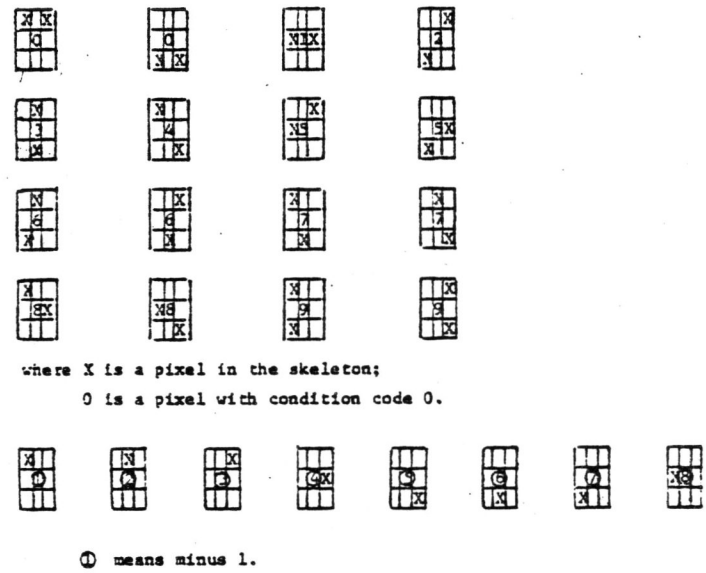


Figure 2. Preliminary Coding of a Pixel in the Skeleton

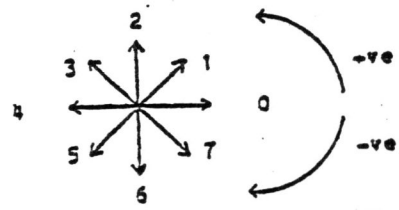
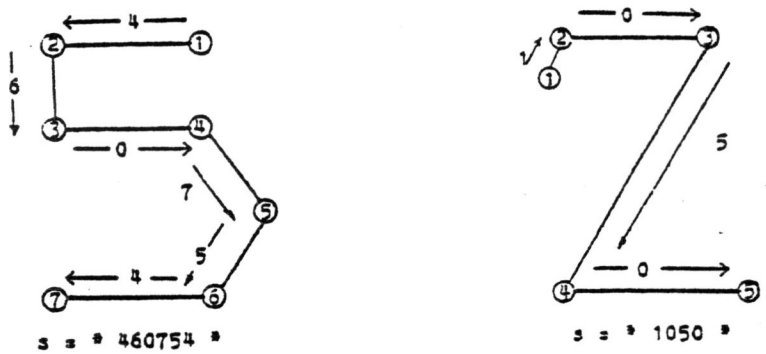
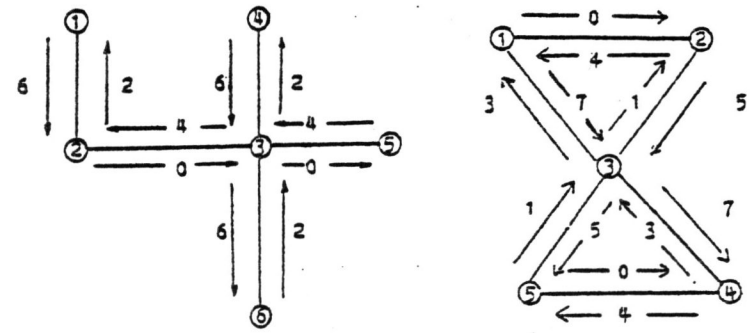


Figure 4. 8-Directions with Assigned Signs



(a) Graph Examples with Terminal and Turning Nodes Only.



(b) Graph Example with Junction Node.

(c) Graph Example with Inner Loops.

Figure 3. Examples of All Cases of Encoding

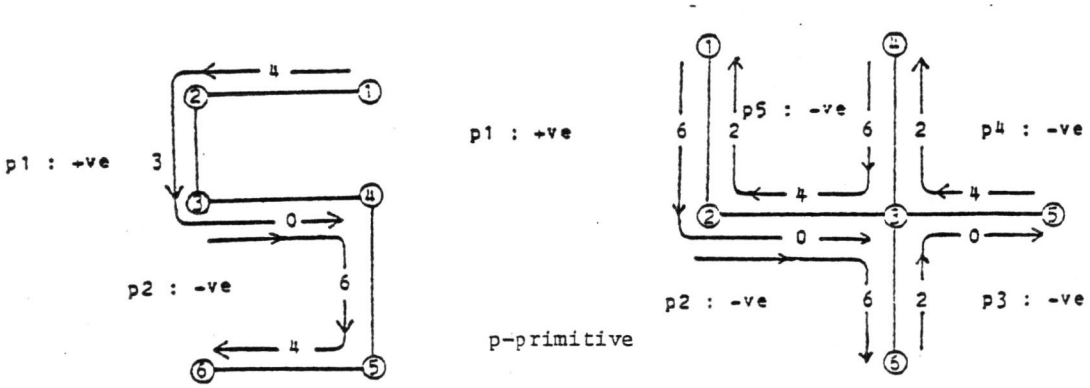
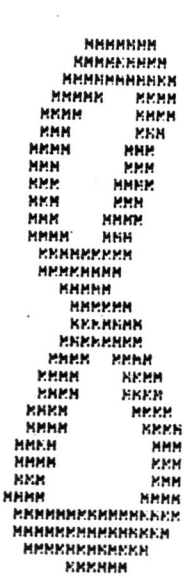
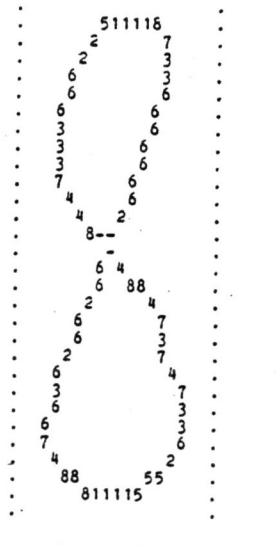


Figure 5. Examples for the Detection of Primitives



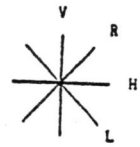
(a) Digitized Image



(b) Thinned-line codes

Vertices :

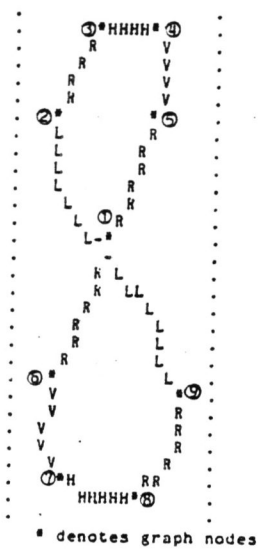
1 : (14,9)
2 : (7,4)
3 : (2,8)
4 : (2,13)
5 : (7,13)
6 : (22,4)
7 : (28,5)
8 : (29,12)
9 : (23,16)



THE ADJACENCY MATRIX :-

From\To	1	2	3	4	5	6	7	8	9
1	-	3	-	-	1	5	-	-	7
2	7	-	1	-	-	-	-	-	-
3	-	5	-	0	-	-	-	-	-
4	-	-	4	-	6	-	-	-	-
5	5	-	-	2	-	-	-	-	-
6	1	-	-	-	-	-	6	-	-
7	-	-	-	-	-	2	-	0	-
8	-	-	-	-	-	-	4	-	1
9	3	-	-	-	-	-	-	5	-

(c) Adjacency Matrix



(d) Graph Representation

Figure 6. Reduction of a Digitized Image to its Graph Representation