Attributed String Matching with Merging for Shape Recognition

WEN-HSIANG TSAI, MEMBER, IEEE, AND SHIAW-SHIAN YU

Abstract—A new structural approach to shape recognition using attributed string matching with merging is proposed. After illustrating the disadvantages of conventional symbolic string matching using changes, deletions, and insertions, attributed strings are suggested for matching. Each attributed string is an ordered sequence of shape boundary primitives, each representing a basic boundary structural unit, line segment, with two types of numerical attributes, length and direction. A new type of primitive edit operation, called merge, is then introduced, which can be used to combine and then match any number of consecutive boundary primitives in one shape with those in another. The resulting attributed string matching with merging approach is shown useful for recognizing distorted shapes. Experimental results prove the feasibility of the proposed approach for general shape recognition. Some possible extensions of the approach are also included.

Index Terms—Attributed strings, boundary primitives, combined primitives, edit distances, matching with merging, shape recognition, string edit operations, string matching.

I. INTRODUCTION

SHAPE recognition is one of the major problems encountered in pattern recognition applications. Shape recognition approaches [1], [2] can be classified roughly into two categories: the statistical (or decision-theoretic) and the structural (or syntactic) methods [3]-[5], [7], [8]. Recently, several attempts have been made to combine the methods from both categories, resulting in the so-called *combined approaches* [4], [9]-[11]. It is hoped that these approaches might include the advantages, and exclude the disadvantages, of both the statistical and the structural approaches. In this paper, we propose a new combined approach to shape recognition.

One way to recognize a shape is to analyze the structural information contained in the shape boundary. There are various methods for boundary representation [1], [2], [6], [8]. Most of them segment a shape boundary into a string of basic structural units (called *primitives*) which are either original boundary segments or just their approximations. Shape recognition is then transformed into string matching [12]-[16]. Conventional string matching only deals with strings of discrete symbols. No numerical data or *attributes* are included. For pattern recognition, it has been shown that injection of attributes into symbols for pattern representation makes it easier to handle noise or distortion and so increases recognition rates [17]. It also reduces the resulting number of symbols needed for the representation of each shape boundary and thus increases the speed of string matching [11]. Depending on how a shape boundary is segmented, various primitives and attributes have been proposed [9], [11], [18]. In this paper, we propose the use of line segments as primitives, and their lengths and directions as attributes. They are simple to extract but are found adequate for shape description and recognition by string matching in this study.

In conventional string matching applications [14], [16], [19], three types of *edit operations*, namely, *changes*, *insertions*, and *deletions* of symbols, are defined for transforming one string into another. Noise and distortion can be handled to some limit by the use of insertions and deletions of symbols. But they are found inadequate in this study for attributed string matching, as will be shown by examples later in this paper. Therefore, a new edit operation, called *merge*, is introduced to further reduce the influence of noise and distortion and improve matching accuracy.

Besides, it is well known that learning is necessary before recognition can proceed. In other approaches using string matching, the segmentation of reference shape boundaries into primitives in the learning stage should be carefully inspected by human operators to avoid erroneous segmentation results caused by noise or distortion. However, due to the use of simple primitives and attributes and the introduction of merging into matching, learning in the proposed approach is very simple and can be made automatic without human interruption. Actually, learning here is just to segment the shape boundary into primitives, as is performed in the recognition stage.

In the remainder of this paper, after reviewing conventional string matching and pointing out its weakness in Section II, we discuss how to extract useful primitives and attributes for shape boundary representation in Section III. The use of line segments as primitives and their lengths and directions as attributes is also proposed. In Section IV, we discuss attributed string matching without merging. Merge operations and their costs are defined in Section V, and matching with merging is presented in Section VI, followed by discussions on shape recognition using the proposed approach in Section VII and on a solution to the shape orientation problem in Section VIII.

Manuscript received March 23, 1984; revised November 21, 1984. Recommended for acceptance by Y. T. Chien.

W.-H. Tsai is with the Department of Information Science and the Microelectronics and Information Science and Technology Research Center, National Chiao Tung University, Hsinchu, Taiwan 300, Republic of China.

S.-S. Yu was with the Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, Republic of China. He is now with the Department of Control Systems, Electronics Research and Service Organization, Industrial Technology Research Institute, Hsinchu, Taiwan, Republic of China.

0162-8828/85/0700-0453$01.00

Experimental results and concluding remarks are included in Sections IX and X, respectively.

II. Conventional String Matching

Generally speaking, to match a finite string A of symbols with another B means to transform or *edit* the symbols in A into those in B with a minimum-cost sequence of allowable edit operations. Conventionally, as defined in [12], [13], the following three types of edit operations are available for symbol transformations:

1) Change—to replace a symbol a with another b, denoted as $a \to b$.

2) Insert—to insert a symbol a into a string, denoted as $\lambda \to a$, where λ is a symbol used to denote nothing (called *null symbol*).

3) Delete—to delete a symbol a from a string, denoted as $a \to \lambda$. An *edit sequence* is defined to be a sequence of ordered edit operations $s1, s2, \cdots, sm$, where si $(i = 1, 2, \cdots, m)$ is any of the above three types of edit operations. Next, let R be an arbitrary nonnegative real cost function which defines a cost $R(a \to b)$ for each edit operation $a \to b$. Also, define the cost of an edit sequence $S = s1, s2, \cdots, sm$ to be

$$R(S) = \sum_{i=1}^{m} R(si).$$

Finally, define the *edit distance* $d(A, B)$ from A to B to be the minimum of the costs of all the edit sequences taking A to B, i.e.,

$$d(A, B) = \min \{R(S) \mid S \text{ is an edit sequence taking } A \text{ to } B\}.$$

Before the matching algorithm can be described precisely, we need more definitions of notations. Given two strings A and B which include $\#A$ and $\#B$ symbols (i.e., with string lengths $\#A$ and $\#B$), respectively, define $A\langle i\rangle$ to be the ith symbol in A and $A\langle i{:}j\rangle$ to be the substring containing the ith to the jth symbols of A. Also, define $A(i)$, $B(j)$ to be $A(i) = A\langle 1{:}i\rangle$, $B(j) = B\langle 1{:}j\rangle$, respectively, and $D(i, j)$ to be $D(i, j) = d(A(i), B(j))$. Accordingly, $D(i, j)$ means the edit distance or the minimum cost taking the substring $A(i)$ to the substring $B(j)$. The following algorithm is proposed in [12] for computing all the edit distances $D(i, j)$ where $1 \leqslant i \leqslant \#A$ and $1 \leqslant j \leqslant \#B$.

Algorithm 1—String Matching:

```
1)  D(0,0) := 0;
2)  for i := 1 to #A do D(i,0) := D(i - 1,0) + R(A⟨i⟩ → λ);
3)  for j := 1 to #B do D(0,j) := D(0,j - 1) + R(λ → B⟨j⟩);
4)  for i := 1 to #A do
5)      for j := 1 to #B do
6)          begin
7)              m1 := D(i,j - 1) + R(λ → B⟨j⟩);
8)              m2 := D(i - 1,j) + R(A⟨i⟩ → λ);
9)              m3 := D(i - 1,j - 1) + R(A⟨i⟩ → B⟨j⟩);
10)             D(i,j) := min (m1, m2, m3)
11)         end.
```

The following example illustrates the use of the above algorithm for shape recognition.

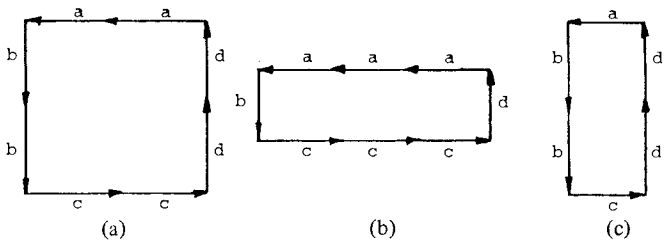


Fig. 1. Reference shapes and input shape used in Example 1. (a) Reference x. (b) Reference y. (c) Input z.

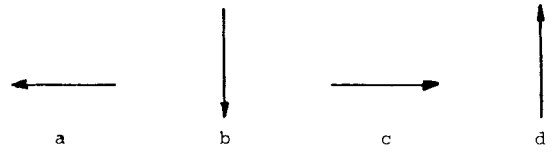


Fig. 2. Primitives and corresponding symbols used in Example 1.

Example 1—Shape Recognition by Conventional String Matching: Let the two shapes x and y shown in Fig. 1(a) and (b) be used as references, and the shape z shown in Fig. 1(c) be the input to be recognized. The primitives together with their symbols are shown in Fig. 2. Accordingly, the string representations for these three shapes are $x = aabbccdd$, $y = aaabcccd$, and $z = abbcdd$. According to the difference of the primitive directions, we define the following edit operation costs:

$$R(e \to \lambda) = R(\lambda \to e) = 1 \quad \text{where} \quad e = a, b, c, \text{or } d;$$

$$R(a \to a) = 0;\ R(b \to a) = 1;\ R(c \to a) = 2;\ R(d \to a) = 1;$$

$$R(a \to b) = 1;\ R(b \to b) = 0;\ R(c \to b) = 1;\ R(d \to b) = 2;$$

$$R(a \to c) = 2;\ R(b \to c) = 1;\ R(c \to c) = 0;\ R(d \to c) = 1;$$

$$R(a \to d) = 1;\ R(b \to d) = 2;\ R(c \to d) = 1;\ R(d \to d) = 0.$$

The results of matching z with x and matching z with y using Algorithm 1 are shown in Table I and Table II, respectively. Each number at the ith column and the jth row in the tables denotes the value $D(i, j)$. The circled numbers in either table indicate all the $D(i, j)$ values of one of the several minimum-cost matchings. Note that due to the costs defined above, the minimum-cost matchings are not unique here. According to Table I, the edit distance $d(x, z)$ is 2 and the corresponding matching is as follows:

$$\begin{array}{ll} x\text{:} & aabbccdd, \\ & \;|\;|\;|\;|\;\;|\;| \\ z\text{:} & a\ bbc\ dd. \end{array}$$

The second a and the second c in x are deleted. And according to Table II, the edit distance $d(y, z)$ is 4 and the matching result is as follows:

$$\begin{array}{ll} y\text{:} & aaabcccd, \\ & \;|\;\;|\;|\;|\;\;|\;| \\ z\text{:} & a\ bbc\ dd. \end{array}$$

Again, the second a and the second c in y are deleted, and the third a and the third c in y are changed to b and d, respectively. Since the edit distance $d(x, z)$ is smaller, the input shape z is recognized as x, which intuitively is correct as can be seen from Fig. 1. This completes Example 1.

TABLE I
STRING MATCHING OF z WITH x

i		0	1	2	3	3	4	5
j			a	b	b	c	d	d
0		(0)	1	2	3	4	5	6
1	a	1	(0)	1	2	3	4	5
2	a	2	(1)	1	2	3	4	5
3	b	3	2	(1)	1	2	3	4
4	b	4	3	2	(1)	2	3	4
5	c	5	4	3	2	(1)	2	3
6	c	6	5	4	3	(2)	2	3
7	d	7	6	5	4	3	(2)	2
8	d	8	7	6	5	4	3	(2)

TABLE II
STRING MATCHING OF z WITH y

i		0	1	2	3	4	5	6
j			a	b	b	c	d	d
0		(0)	1	2	3	4	5	6
1	a	1	(0)	1	2	3	4	5
2	a	2	(1)	1	2	3	4	5
3	a	3	2	(2)	2	3	4	5
4	b	4	3	2	(2)	3	4	5
5	c	5	4	3	2	(2)	3	4
6	c	6	5	4	4	(3)	3	4
7	c	7	6	5	5	4	(4)	4
8	d	8	7	6	6	5	4	(4)

From the above example, we see that the primitives used must not be too long in length so that shape boundaries can be divided into primitives without causing too much inaccuracy. However, this will increase the lengths of the resulting string representations, and so increase the resulting string matching time because the time complexity of Algorithm 1 above is to the order of $\#A$ by $\#B$. It is not difficult to see that this weakness lies in the use of symbols only. Symbols are discrete in nature while most problems of pattern recognition deal with attributes which are basically continuous in nature. To make string matching suitable for pattern recognition, is should be modified to handle attributes in addition to symbols [9]-[11], [22]-[24].

III. Structures and Attributes of Shape Boundary Primitives

Around the boundary of a given shape, the most critical points are those with locally maximum curvatures and so they may be extracted to characterize the shape structure [27]. They often appear on the turning points of corners. Therefore, it is appropriate to connect these points by line segments and accept the resulting polygon as the line approximation of the given shape. However, if the given shape contains at least one curve segment on its boundary with no locally maximum curvature (for example, a portion of a circle), then this curve segment will simply be approximated by a line segment connecting its two end points. Therefore, the curvature information existing on the curve segment will be ignored, resulting possibly in less recognition accuracy if this information is important for shape discrimination. In such cases, finer approximation should be made on the curve segments, using more line segments. We will call such refinement *curve splitting*. Based on the above ideas, a procedure for boundary segmentation and attribute extraction is proposed in Fig. 3.

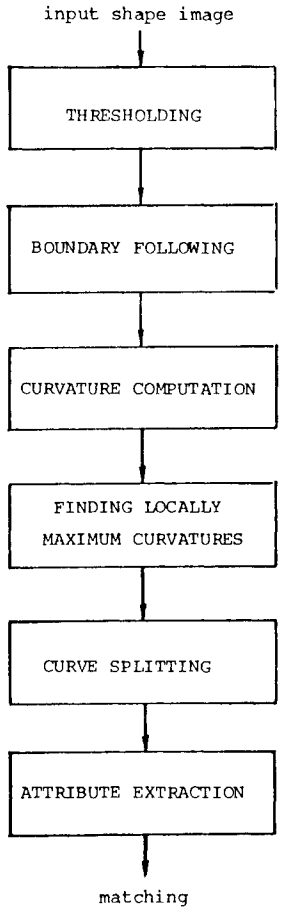


Fig. 3. Boundary segmentation and attribute extraction.

The first step is to threshold a shape image into a binary picture. The shape image is taken against a clean background, and so the result of thresholding is a clear black shape with a white background. Next, boundary following [6] is performed to thread the shape boundary points into a sequence of ordered points. The start point of the sequence is found by a raster scan of the picture from top to bottom. The direction of boundary following is counterclockwise. To reduce processing time, only the first of every three points found by boundary following is recorded.

The method we use to compute the curvature at each recorded boundary point is the one proposed in Rosenfeld and Johnston [29] which computes angle cosine values as the equivalents for curvatures. All the points with locally maximum curvatures on the boundary are then found out as *segmentation points*. When curve segments with no locally maximum curvatures exist on the boundary, no segmentation point

can be found on them and curve splitting should be applied, which is described in the following.

Let $P1$ and $P2$ be any two neighboring segmentation points found so far with coordinates $(x1, y1)$ and $(x2, y2)$, respectively, and Pm be the middle point on the boundary between $P1$ and $P2$, with coordinates (xm, ym). First, compute the distance d from Pm to the cord connecting $P1$ and $P2$. When d is large enough (larger than a threshold), Pm is considered to be on a curve segment with fairly large curvature and is accepted as a segmentation point. This refinement is repeated between $P1$ and Pm and between Pm and $P2$ again, and so on, until no further refinement can be made between all point pairs. The final set of ordered segmentation points with an imaginary line segment connecting every two consecutive points is then accepted as the polygon approximation of the given shape. Recall that each imaginary line segment here is regarded as a boundary primitive, as is defined previously.

Finally, we come to the step of attribute extraction. Let $P1$ and $P2$ be the two end points of a primitive a; then the length attribute l of a is defined as the distance between $P1$ and $P2$, and the direction attribute θ as the angle formed between a and a reference line segment. The reference line segment is selected to be the first primitive in the primitive sequence, connecting the first and the second segmentation points. The angle θ is measured counterclockwise from the reference primitive.

As an illustration of the previous procedure, let the plier shown in Fig. 4(a) be used as the input shape. The results of all the intermediate steps of the procedure are shown in Fig. 4(b)-(e).

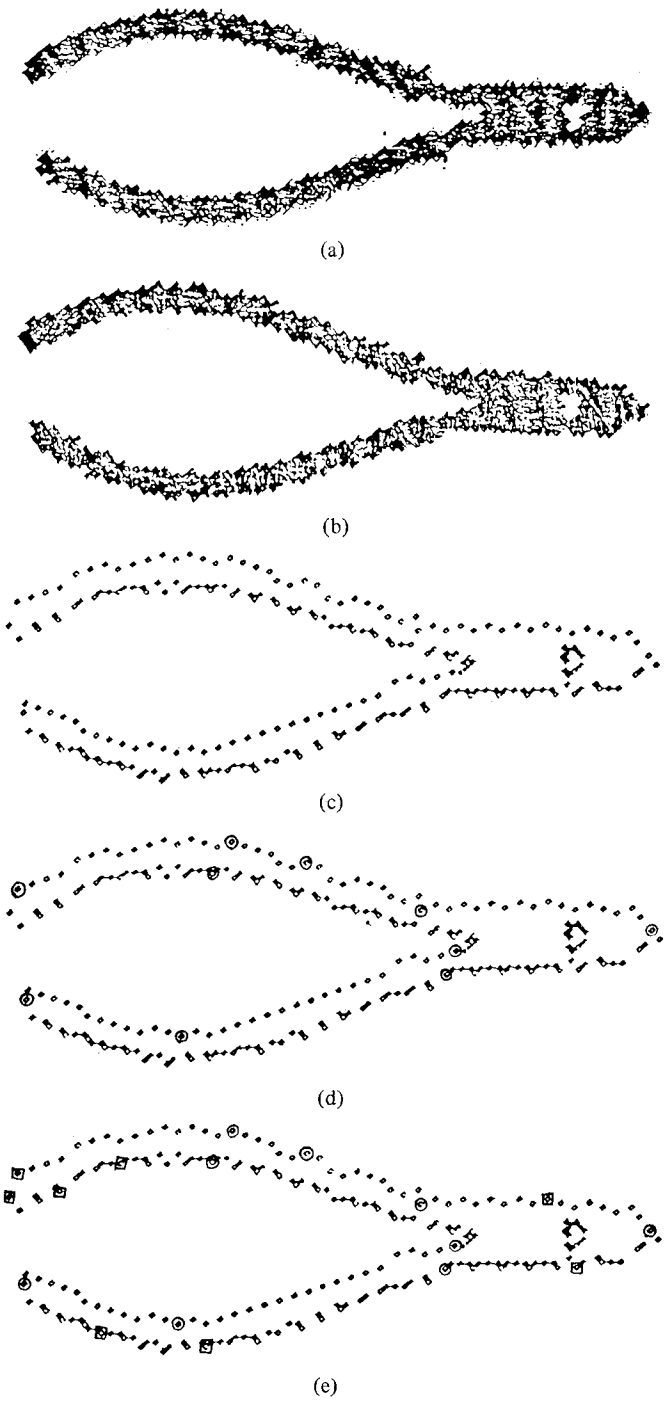


Fig. 4. Illustration of the procedure of boundary segmentation and attribute extraction. (a) Original shape of a plier. (b) Binary image after thresholding. (c) Result of boundary following. (d) Extracted segmentation points (circled ones) with locally maximum curvatures. (e) Result of curve splitting (squared points).

IV. Attributed String Matching

The first step before attributed string matching can be performed is to define the cost function for various edit operations. This should be done for attribute transformations instead of for symbol transformations, because all primitives as proposed are of only one type, line segment, and can be represented by a single symbol. Let A and B be two strings of extracted boundary primitives, $A\langle i\rangle$ and $B\langle j\rangle$ be two primitives in A and B with attributes $(li, \theta i)$ and $(lj, \theta j)$, respectively. Also let lA and lB be the total lengths of all the primitives (i.e., the sum of all primitive length attributes) in A and B, respectively. We define the cost function for a change operation $A\langle i\rangle \rightarrow B\langle j\rangle$ to be

$$R(A\langle i\rangle \rightarrow B\langle j\rangle) = H(\theta i, \theta j)/180^\circ + \left| li/lA - lj/lB \right| \cdot (\#A \times \#B)^{1/2}, \quad (1)$$

where the first term in the right-hand side defines the partial cost due to the angle difference $H(\theta i, \theta j)$ between the two primitives, with 180° in the denominator as a normalization factor which makes this partial cost to lie between 0 and 1. It is not difficult to figure out that $H(\theta i, \theta j)$ should be defined to be

$$H(\theta i, \theta j) = \left|\theta i - \theta j\right| \quad \text{when} \quad \left|\theta i - \theta j\right| \leqslant 180^\circ;$$
$$= 360^\circ - \left|\theta i - \theta j\right| \quad \text{when} \quad \left|\theta i - \theta j\right| > 180^\circ.$$

The second term in (1) above defines the other partial cost due to length difference. Again, lA and lB are used as normalization factors. The reason to include $(\#A \times \#B)^{1/2}$ in the second term is explained subsequently. Suppose $\#A = \#B = n$ now and only change operations are used in matching A with B. Then, each primitive $A\langle i\rangle$ in A will be transformed, using a change operation, to a primitive $B\langle j\rangle$ in B, and totally n change operations will be needed. Since the partial cost due to angle difference for each change operation is within 0 and 1, the total partial cost due to angle difference for all n changes will

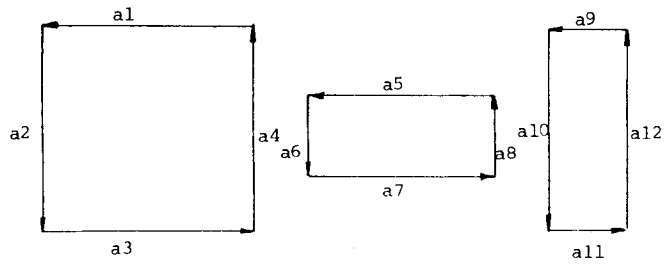


Fig. 5. Reference shapes and input shape used in Example 2. (a) Reference x. (b) Reference y. (c) Input z.

accumulate to be within 0 and n. However, if only $|li/lA - lj/lB|$ is employed for calculating the partial cost due to length difference, the total of such cost for n changes will be within 0 and 1, instead of within 0 and n, resulting in an undesirable cost bias for angle difference. Therefore, n should be included in the second term as a magnification factor to make both types of partial costs comparable in magnitude. For the more general case where $\#A \neq \#B$, we can simply replace n with $(\#A \times \#B)^{1/2}$. An advantage of the above cost normalization is the independency of shape scaling in the resulting matching.

For the costs for insert and delete operations, we can regard a null primitive λ as a vector with zero length but with indefinite angle. Therefore, it is reasonable to define the partial cost due to angle difference as a constant which should be determined by specific application requirements. This leads to the following definitions

$$R(A\langle i\rangle \rightarrow \lambda) = Kd + (li/lA) \times (\#A \times \#B)^{1/2} \tag{2}$$

for the delete cost and

$$R(\lambda \rightarrow B\langle j\rangle) = Ki + (lj/lB) \times (\#A \times \#B)^{1/2} \tag{3}$$

for the insert cost, where Kd and Ki are two constants whose values should be assigned between 0 and 1. Using the above cost functions for the three types of edit operations, the conventional string matching algorithm (Algorithm 1) now can be used for shape recognition without any modification. The following is an illustrative example to be compared with Example 1.

Example 2—Shape Recognition by Attributed String Matching Without Merging: The reference and the input shapes, as shown in Fig. 5, are identical to those of Example 1 (Fig. 1), but with their boundaries segmented in a different way—in the way as proposed in the last section. We use a single symbol a to represent all primitives, but add numbers to a as postfixes to differentiate them. Accordingly, we have the following string representations for the shapes $x = a1\ a2\ a3\ a4, y = a5\ a6\ a7\ a8, z = a9\ a10\ a11\ a12$.

As an illustration of change cost computation, the value $R(a1 \rightarrow a11)$ is calculated as follows. First, we have attributes $l1 = 2$ and $\theta 1 = 0°$ for $a1$, and $l11 = 1$ and $\theta 11 = 180°$ for $a11$ where we use $a1$, $a5$, and $a9$ as the start primitives for angle reference. Next, the total lengths of the strings are $lx = 8, ly = 8$, and $lz = 6$. The number of primitives in each string is four. Also, we define both Kd and Ki in (2) and (3) to be $\frac{1}{2}$. Then, we have

$$\begin{aligned} R(a1 \rightarrow a11) &= H(\theta 1, \theta 11)/180° + |l1/lx - l11/lz| \\ &\quad \times (\#x \times \#z)^{1/2} \\ &= H(0, 180°)/180° + \left|\tfrac{2}{8} - \tfrac{1}{6}\right| \times (4 \times 4)^{1/2} \\ &= \tfrac{4}{3}. \end{aligned}$$

After Algorithm 1 is executed for matching z with x, we get the optimal matching as follows which includes neither insertions nor deletions.

$$\begin{array}{lcccc} x: & a1 & a2 & a3 & a4 \\ & | & | & | & | \\ z: & a9 & a10 & a11 & a12. \end{array}$$

The cost of matching or edit distance is $\frac{2}{3}$. Similarly, the optimal matching of z with y is

$$\begin{array}{lcccc} y: & a5 & a6 & a7 & a8 \\ & | & | & | & | \\ z: & a9 & a10 & a11 & a12. \end{array}$$

The edit distance is $\frac{10}{3}$, which is computed as follows.

$$\begin{aligned} d(y, z) &= R(a5 \rightarrow a9) + R(a6 \rightarrow a10) \\ &\quad + R(a7 \rightarrow a11) + R(a8 \rightarrow a12) \\ &= (0/180° + \left|\tfrac{3}{8} - \tfrac{1}{6}\right| \times 4) + (0/180° + \left|\tfrac{1}{8} - \tfrac{2}{6}\right| \times 4) \\ &\quad + (0/180° + \left|\tfrac{3}{8} - \tfrac{1}{6}\right| \times 4) \\ &\quad + (0/180° + \left|\tfrac{1}{8} - \tfrac{2}{6}\right| \times 4) \\ &= \tfrac{10}{3}. \end{aligned}$$

Since $\frac{10}{3}$ is larger than $\frac{2}{3}$, the input z is recognized as x, which is the same result as that of Example 1. This completes Example 2.

As can been seen above, the number of primitives for each shape is reduced to four only. This speeds up matching. Although this improvement is due to the use of longer primitives which result from the introduction of attributes, a certain problem caused by the effect of noise and distortion has yet to be solved before attributed string matching can be really useful for shape recognition. This problem arises when a primitive, supposed to be single, is broken into several shorter ones due to the interference of noise or distortion. The following example illustrates this problem.

Example 3—Problem Encountered in Attributed String Matching Without Merging: Shown in Fig. 6 are three shapes with the first two, x and y, as the references and the third, z, as the input. z actually is identical to x, but because of noise which causes some local curvature maxima to occur on the boundary, the left-hand side of z is segmented into three smaller primitives, $a12$, $a13$, and $a14$, instead of just one. Using the costs defined by (1), (2), and (3) with $Kd = Ki = \frac{1}{2}$ again and the following length information

$$l1 = l3 = l9 = l11 = l15 = 3;\quad l2 = l4 = l10 = l16 = l5 = 2;$$

$$l6 = l7 = l8 = 1;\quad l12 = l13 = l14 = \tfrac{2}{3},$$

we get the results as follows:

$$\begin{aligned} d(x,z) &= R(a1 \to a11) + R(a2 \to a12) + R(\lambda \to a13) \\ &\quad + R(\lambda \to a14) + R(a3 \to a15) \\ &\quad + R(a4 \to a16) \\ &= \tfrac{11}{3} \end{aligned}$$

and

$$\begin{aligned} d(y,z) &= R(a5 \to a11) + R(a6 \to a12) + R(a7 \to a13) \\ &\quad + R(a8 \to a14) + R(a9 \to a15) \\ &\quad + R(a10 \to a16) \\ &= \tfrac{5}{2}. \end{aligned}$$

Since $\frac{5}{2}$ is smaller than $\frac{11}{3}$, the input z is classified as from y instead of from x, contrary to the expected result! This completes Example 3.

The above example reveals that attributed string matching as proposed previously is not powerful enough to handle noise or distortion. This problem was also found in Tsai and Fu [10], [11]. It is caused partially by the inadequacy of available edit operations. It is expected that $a12$, $a13$, and $a14$ all together could be matched with $a2$, but this is not allowed so far. And so, $a13$ and $a14$ both are deleted, resulting in an increase of matching cost. If we could merge $a12$, $a13$, and $a14$ to form a single primitive which is then matched with the primitive $a2$, the final match cost $d(x, z)$ could be smaller and z would possibly be correctly classified as from x. These discussions lead to the idea of adding one additional edit operation, called *merge*, into string matching, as is discussed in the next section.

V. Merge Operation and Merge Cost

First, we have to define the new attributes of a primitive c which is the result of merging two other primitives a and b. We will call primitive c the *combined primitive* from a and b. Let the attributes of a, b, and c be $(la, \theta a)$, $(lb, \theta b)$, and $(lc, \theta c)$, respectively. We define lc as

$$lc = la + lb, \tag{4}$$

and θc as follows:

$$\begin{aligned} \theta c &= \theta a + |\theta a - \theta b| \times (lb/(la + lb)) \\ &\qquad \text{if } \theta a < \theta b \text{ and } |\theta a - \theta b| \leqslant 180^\circ; \text{ or} \\ &= \theta b + |\theta a - \theta b| \times (la/(la + lb)) \\ &\qquad \text{if } \theta a \geqslant \theta b \text{ and } |\theta a - \theta b| \leqslant 180^\circ; \text{ or} \\ &= \theta a + (360^\circ - |\theta a - \theta b|) \times (lb/(la + lb)) \\ &\qquad \text{if } \theta a \geqslant \theta b \text{ and } |\theta a - \theta b| > 180^\circ; \text{ or} \\ &= \theta b + (360^\circ - |\theta a - \theta b|) \times (la/(la + lb)) \\ &\qquad \text{if } \theta a < \theta b \text{ and } |\theta a - \theta b| > 180^\circ. \end{aligned} \tag{5'}$$

The reason for defining lc by (4) above can be observed from a case where a and b both are on a line segment, like $a12$ and $a13$ in z of Fig. 6. The resulting length lc of course should be the sum of la and lb. The reason for defining θc as in (5') above is, for the case where $\theta a < \theta b$ and $|\theta a - \theta b| \leqslant 180^\circ$, that θc resulting from merging should be larger than the smaller of θa and θb (i.e., θa), and be as close to the larger one (i.e., θb) as is weighted by the term $lb/(la + lb)$. The reasons for the other cases are similar. Actually, if we define a new function as follows:

$$\begin{aligned} M(\theta a, \theta b) &= \min(\theta a, \theta b) \quad \text{if } |\theta a - \theta b| \leqslant 180^\circ; \\ &= \max(\theta a, \theta b) \quad \text{otherwise,} \end{aligned}$$

and use the function $H(\theta a, \theta b)$ we defined before, (5') can be expressed simply as

$$\theta c = \theta d + H(\theta a, \theta b) \times (1 - ld/(la + lb))$$
$$\text{where } d \text{ is such that } \theta d = M(\theta a, \theta b). \tag{5}$$

For the case where $\theta a = \theta b$, θc can be, using (5) above, easily computed to be θa or θb. This means that if an originally noise-free line segment c, supposed to be a single primitive, is somehow divided into two segments a and b which are still aligned on the direction of the original primitive c, then the combined primitive from a and b, according to (4) and (5) above, will be exactly c itself which is the desired result.

The next step is to define the cost for a merge operation. Let $A\langle i-k+1:i\rangle = a(i-k+1)\,a(i-k+2)\cdots ai$ be a sequence of primitives on a boundary to be merged, with primitive a as the combined primitive from all the primitives in $A\langle i-k+1:i\rangle$. Let $(lj, \theta j)$ and (l, θ) be the attributes associated with any aj $(j = i-k+1, i-k+2, \cdots, i)$ and a, respectively. For the merge cost to be defined reasonably, it seems necessary to obey the following criteria.

1) The larger the angle difference between θj and θ is, the larger the partial cost due to aj should be.

2) The larger the length lj is, the larger the partial cost due to aj should be.

3) The larger the number of all aj is, the larger the total merge cost should be.

Additionally, to simplify the notations to be used in the merge cost definition and the subsequent matching with merging algorithm, we use $A^k\langle i\rangle$ to denote the combined primitive a, and further denote the merge operation as $A\langle i-k+1:i\rangle \to A^k\langle i\rangle$. Note that the k primitives merged are indexed from $i-k+1$ to i, instead of from i to $i+k-1$. For $k=1$, $A^k\langle i\rangle = A^1\langle i\rangle = A\langle i:i\rangle = A\langle i\rangle$, which is just a single primitive, $A\langle i\rangle$, itself. Now, by considering $A^k\langle i\rangle$ as a single primitive, we can accept $A^k\langle i\rangle \to B^m\langle j\rangle$ as a single change operation which is performed after the k primitives in $A\langle i-k+1:i\rangle$ are merged as $A^k\langle i\rangle$ and the m primitives in $B\langle j-m+1:j\rangle$ merged as $B^m\langle j\rangle$. For $k=1$ and $m=1$, no merge is really performed and the above change operation reduces to the conventional one-to-one change operation $A\langle i\rangle \to B\langle j\rangle$.

Based on the above-mentioned three criteria, we define the merge cost as follows:

$$R(A\langle i-k+1:i\rangle \to A^k\langle i\rangle) = k \times \sum_{j=i-k+1}^{i} (H(\theta, \theta j)/180^\circ) \times (lj/l). \tag{6}$$

Recall that (l, θ) is the attributes of $A^k\langle i\rangle$. The term $H(\theta, \theta j)/$

180° in (6) above is included to satisfy criterion 1) above, the term lj/l to satisfy criterion 2), and the number k at the beginning of the right-hand side of (6) to satisfy criterion 3) above. For $k = 1$, (6) reduces to $R(A\langle i\rangle \rightarrow A\langle i\rangle) = 0$ as it should be because $H(\theta, \theta j) = H(\theta i, \theta i) = 0$ now.

VI. Attributed String Matching with Merging

In this section, we discuss how to incorporate merging into conventional string matching, and then propose an algorithm for matching with merging.

Consider a primitive a on a shape boundary in which one picture, due to noise or distortion, is divided into a sequence of shorter primitives $A\langle i-k+1:i\rangle = a(i-k+1)a(i-k+2)\cdots ai$ after segmentation. The same primitive a in another picture is similarly divided into another sequence of shorter primitives $B\langle j-m+1:j\rangle = a(j-m+1)\ a(j-m+2)\cdots aj$. It is desired to allow all the k primitives in $A\langle i-k+1:i\rangle$ to be matched with all the m primitives in $B\langle j-m+1:j\rangle$ in the matching process. Or more generally, it is desired to allow the combined primitive from any set of consecutive primitives in one string A to be matched, using a change operation, with the combined primitive from any set of consecutive primitives in another string B. The conventional one-to-one change operation $A\langle i\rangle \rightarrow B\langle j\rangle$ is already seen to be only a special case here. Therefore, to include merge cost computation into the conventional string matching algorithm (Algorithm 1), we have to replace line 9 of Algorithm 1, which is used to compute and add to $D(i-1, j-1)$ to one-to-one change cost $R(A\langle i\rangle \rightarrow B\langle j\rangle)$, with the following four lines of operations:

9.1 for $k := 1$ to i

9.2 for $m := 1$ to j

9.3 $m3(k,m) := D(i-k, j-m) + R(A\langle i-k+1:i\rangle \rightarrow A^k\langle i\rangle) + R(B\langle j-m+1:j\rangle \rightarrow B^m\langle j\rangle) + R(A^k\langle i\rangle \rightarrow B^m\langle j\rangle)$;

9.4 $m3 := \min_{\substack{1\le k\le i \\ 1\le m\le j}} m3(k,m)$;

Line 9.3 above is used to compute the partial cost of matching $A\langle i-k+1:i\rangle$ with $B\langle j-m+1:j\rangle$, which includes three parts: two for merging the primitives in $A\langle i-k+1:i\rangle$ and $B\langle j-m+1:j\rangle$, respectively, and the third for matching $A^k\langle i\rangle$ with $B^m\langle j\rangle$, using a change operation. We will denote this partial cost as $R(A\langle i-k+1:i\rangle \rightarrow B\langle j-m+1:j\rangle)$ subsequently, and regard the operation $A\langle i-k+1:i\rangle \rightarrow B\langle j-m+1:j\rangle$ as a general type of many-to-many change operation which includes the conventional one-to-one change operation as a special case with $k = 1$ and $m = 1$. The value $R(A\langle i-k+1:i\rangle \rightarrow B\langle j-m+1:j\rangle)$ is just the cost of one of the i by j different possibilities of matching the combined primitive from $A\langle i\rangle$ and any consecutive $k-1$ primitives in $A\langle i-k+1:i-1\rangle$ with the combined primitives from $B\langle j\rangle$ and any consecutive $m-1$ primitives in $B\langle j-m+1:j-1\rangle$. This cost is added in line 9.3 to $D(i-k, j-m)$ which is the previously computed minimum cost of matching $A\langle 1:i-k\rangle$ with $B\langle 1:j-m\rangle$. Line 9.4 is used to select the minimum of the values, $m3(k, m)$, which replaces the old $m3$ value of line 9 of Algorithm 1. In short, the difference between the conventional matching and matching with merging lies in the use of different types of change operations—the conventional one-to-one type or the many-to-many type. The resulting algorithm will be called Algorithm 2. The following is an example continued from Example 3 to illustrate the improvement of recognition after merging is included in attributed string matching.

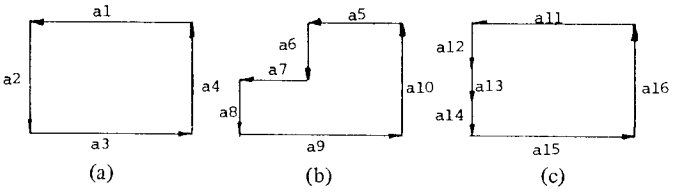


Fig. 6. Reference shapes and input shape used in Example 3. (a) Reference x. (b) Reference y. (c) Input z.

Example 4—Shape Recognition by Attributed String Matching with Merging: With merge operations allowed, the cost of matching $a2$ of x in Fig. 6(a) with the combined primitive from $a12$, $a13$, and $a14$ of z in Fig. 6(c) is

$$R(a2 \rightarrow a12a13a14) = R(a2 \rightarrow a2) + R(a12a13a14 \rightarrow a14^3) + R(a2 \rightarrow a14^3) \quad (7)$$

where $a14^3$ with attributes $(l14^3, \theta 14^3)$ is used to represent the combined primitives from $a12$, $a13$, and $a14$. Now, since $\theta 12 = \theta 13 = \theta 14 = 270^\circ$, repetitive applications of (5) results in $\theta 14^3 = 270^\circ$ as can be easily predicted. So, all $H(\theta, \theta j)$ terms in (6) are zero, and so is $R(a12a13a14 \rightarrow a14^3)$. This means that $a12$, $a13$, and $a14$ are merged with no cost. This result is reasonable because they are originally consecutively aligned on a single line segment. Also, $l14^3 = l12 + l13 + l14 = 2$. Thus, (7) reduces to $R(a2 \rightarrow a12a13a14) = R(a2 \rightarrow a14^3) = 0$. Also, $R(a1 \rightarrow a11) = R(a3 \rightarrow a15) = R(a4 \rightarrow a16) = 0$. Therefore, it is easy to see that

$$d(x,z) = R(a1 \rightarrow a11) + R(a2 \rightarrow a12a13a14) + R(a3 \rightarrow a15) + R(a4 \rightarrow a16) = 0,$$

which is the minimum error. Consequently, the decision now can be reversed, i.e., we decide that z is just a noisy version of x but not y. This completes Example 4.

Example 4 illustrates the merge of several primitives to match with a single primitive. Actually, using points with locally maximum curvatures in boundary segmentation, followed by merging and matching any numbers of consecutive primitives, can be expected to be quite powerful for handling the problems caused by noise and distortion. For example, the shapes shown in Fig. 7(a) and (b) are two distorted versions of a square shape. They still look quite similar on the whole but their detailed boundary primitives are all different both in numbers and in positions. Matching without merging will result in at least two primitives being deleted from the shape of Fig. 7(b) [or equivalently, being inserted into that of Fig. 7(a)] and nine pairs of primitives being matched (by change operations). The total matching cost will be high because of the high length dissimilarity between the primitives in each pair.

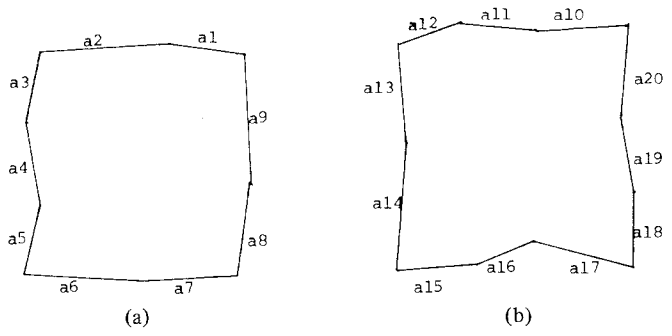


Fig. 7. Two distorted shapes to be matched. (a) Distorted shape 1. (b) Distorted shape 2.

However, with merging incorporated into matching, the final matching will include just four many-to-many change operations: $a1a2 \rightarrow a10a11a12$, $a3a4a5 \rightarrow a13a14$, $a6a7 \rightarrow a15a16a17$, $a8a9 \rightarrow a18a19a20$. Since the primitives on either side of each many-to-many change operation above are quite similar in their directions, merging them into a longer primitive will cost very little. Furthermore, the two longer combined primitives, resulting from merging the primitives on both sides of each change operation above, also look quite similar in their lengths and directions. Therefore, the change costs will also be low. This means that the overall cost for matching Fig. 7(a) with 7(b) will be much lower when merging is allowed, as is desired.

VII. Shape Recognition Through Matching with Merging

Given a set of N reference shapes represented by strings $A1, A2, \cdots, AN$, and an input unknown shape represented by string B, we want to classify B as coming from one of these reference shapes. This shape recognition problem can be solved by assigning B to the reference shape An if the similarity between B and An is larger than that between B and any other Ai. It seems that a similarity measure appropriate for this purpose can be defined in terms of the edit distance computed by Algorithm 2. For Ai and B with $\#Ai$ and $\#B$ as their numbers of symbols, this distance is $D(\#Ai, \#B)$ which includes the total cost of four types of edit operations, namely, changes, inserts, deletes, and merges. But it is not difficult to see that the cost of each type of these operations depends on $\#Ai$ and $\#B$, because in (1)-(3), the term $(\#Ai \times \#B)^{1/2}$ is included, and because in (6), the term k is included, which is the number of merged primitives in a merge operation, also related to the magnitude $\#Ai$ or $\#B$. To eliminate this undesired effect, we normalize the value $D(\#Ai, \#B)$ by the quantity $(\#Ai \times \#B)^{1/2}$ and define a new between-string similarity measure as follows:

$$S(Ai, B) = 1 - D(\#Ai, \#B)/(\#Ai \times \#B)^{1/2}. \quad (8)$$

The value of S is between 0 and 1. The larger S is, the more similar the two strings Ai and B are. The reference shape An, to which B should be assigned, now can be determined by the following decision rule:

$$\text{assign } B \text{ to } An \text{ if } S(An, B) = \max_{1 \le i \le N} S(Ai, B). \quad (9)$$

VIII. Shape Orientation Problem and a Solution

Often encountered in the applications of syntactic or structural pattern recognition is the problem of shape orientation which causes a single shape in different orientations to be represented by different strings. These strings actually are all identical in the noise-free case if each of them is viewed as a loop by tying the last symbol of the string to the first. String matching proposed so far in this paper only compares two strings right from their respective start symbols or primitives. Therefore, the strings of a single shape in two different orientations will not match perfectly even in the noise-free case. The way we solve this problem is to make use of sharp corners on the shape boundary. Sharp corners, with their large curvatures, usually are less affected by noise or distortion than dull ones, i.e., they are more likely to be kept in the given shape. Therefore, in forming a string representation, it is reasonable for us to select as the start primitive the one right at one side of the sharpest corner (with the maximum curvature). In this way, the chance for us to get two identical strings of a single shape in two distinct orientations will be largely increased. Of course, the shape might be severely distorted right at the sharpest corner, resulting in a lower curvature at that corner. In such a case, we may still try the next sharpest corner as the start point, and the chance to get two identical strings may still be high. To be safer, the third sharpest corner of course may still be tried and so on. If two strings are found to match quite well with a very small match cost, then we can simply stop there and use the match cost to compute the final measure of similarity.

IX. Experimental Results

To study the feasibility of the proposed approach to shape recognition, sample shapes from various objects are collected and tested. The procedures for training and for recognition are almost identical except that at the end of recognition, one more step, matching with merging, is performed. Actually, we can say that no particular operation is needed for the learning stage.

For those simple shapes such as squares, triangles, or more generally, polygons, the proposed approach works very well with a nearly 100 percent recognition rate. So, we selected some more complicated shapes with curve boundaries for testing which are three different pliers as shown in Fig. 8. They are quite similar in shape. The images taken by a video camera for testing are of the size 128×128. To improve recognition speed so that more experimental data can be obtained, the number of segmentation points (i.e., the number of primitives) for each shape is limited to be less than or equal to ten. That is, points with smaller curvatures are ignored if segmentation points obtained during the procedure of boundary segmentation are more than ten in number. Three reference shapes of the pliers are first taken and processed to obtain their string representations. Unknown shapes are then input one by one to match with all the three reference shapes. Three similarity measures are computed for each input and a decision is made. According to experimental experience, if a similarity measure is computed to be higher than 95 percent then it is not neces-

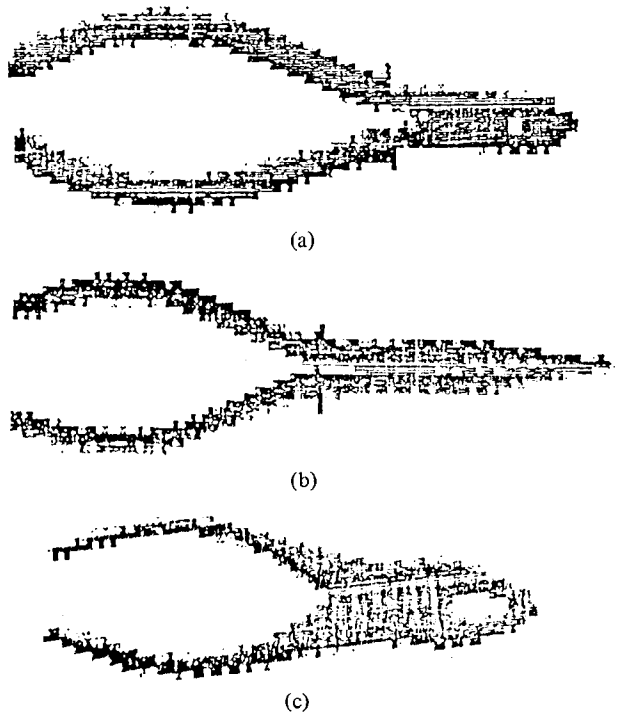


Fig. 8. Three distinct pliers used in experiments of shape recognition. (a) Plier of model 1. (b) Plier of model 2. (c) Plier of model 3.

TABLE III
EXPERIMENTAL RESULTS OF MATCHING INPUT SHAPES WITH THREE REFERENCE PLIERS

	MODEL 1	MODEL 2	MODEL 3
MODEL 1	* 93.0108 (1,1)	84.8502 (3,2)	78.3915 (3,3)
	* 96.5935 (1,1)	87.4023 (3,2)	69.4336 (3,3)
	* 89.0625 (2,1)	84.9764 (3,2)	81.6573 (3,1)
	* 92.2263 (3,2)	89.3154 (3,1)	87.4815 (3,2)
	57.1775 (3,1)	X 60.5015 (3,2)	57.5552 (3,3)
	* 97.5781 (1,1)	----------	----------
	* 93.4375 (2,2)	78.8437 (2,1)	92.2656 (2,2)
	* 96.3281 (1,1)	----------	----------
	* 97.5781 (1,1)	89.1406 (2,2)	82.5781 (3,3)
	* 96.6406 (1,1)	----------	----------
	* 95.3125 (1,1)	----------	----------
	* 96.3095 (2,2)	----------	----------
	* 96.2755 (2,2)	----------	----------
	* 93.7429 (2,2)	85.1562 (3,3)	90.0929 (1,1)
	* 69.2992 (2,2)	61.7068 (3,3)	63.4889 (3,1)
	* 96.0521 (2,2)	----------	----------
	* 93.4449 (1,1)	74.2969 (3,1)	91.3592 (2,2)
MODEL 2	58.9892 (2,1)	* 63.6639 (3,2)	61.0937 (3,2)
	88.7197 (2,1)	* 92.4882 (1,1)	83.3594 (3,2)
	83.0357 (2,1)	* 91.6147 (1,1)	81.4062 (3,2)
	77.1888 (2,2)	* 83.5789 (3,2)	52.5000 (3,2)
	85.7533 (2,2)	* 95.6327 (2,2)	----------
	84.5312 (2,1)	* 99.1406 (1,1)	----------
	86.7969 (2,2)	* 94.4531 (1,1)	----------
	57.4200 (2,1)	56.4862 (2,2)	X 58.6339 (3,3)
	83.7524 (2,1)	* 91.0358 (2,2)	79.4570 (3,3)
	87.1094 (2,1)	* 95.5469 (2,2)	----------
	76.4844 (2,1)	* 92.3437 (1,1)	68.6719 (3,2)
	89.1991 (2,2)	* 91.0156 (3,3)	85.6981 (3,1)
	85.9215 (2,1)	* 91.1719 (1,1)	85.6981 (3,2)
	X 85.7726 (2,1)	78.7500 (3,1)	83.0909 (3,2)
	86.1450 (2,2)	* 92.2656 (3,3)	84.7297 (3,1)
	X 90.0929 (1,1)	83.3594 (3,1)	87.8583 (3,2)
	88.0817 (2,2)	* 95.3906 (3,3)	----------
MODEL 3	90.7413 (1,1)	81.2175 (3,2)	* 92.0515 (3,2)
	71.4062 (2,1)	X 74.2241 (3,1)	68.6719 (2,1)
	87.5781 (2,1)	86.6591 (3,2)	* 94.8437 (3,2)
	88.5576 (2,1)	77.5346 (3,2)	* 92.8376 (3,2)
	81.1719 (2,2)	76.2006 (3,1)	* 92.0312 (3,3)
	92.3437 (2,2)	76.0937 (3,1)	* 96.5450 (2,2)
	74.2328 (2,1)	75.8924 (3,3)	* 76.1419 (3,2)
	72.1094 (2,1)	X 73.3594 (3,3)	68.9998 (3,1)
	93.8857 (1,1)	78.8622 (3,1)	* 94.1537 (1,1)
	X 96.6236 (2,2)	----------	----------
	73.7853 (2,2)	X 88.1250 (3,2)	73.3328 (3,1)
	83.9215 (2,1)	75.6520 (3,1)	* 90.9123 (2,2)
	X 76.9310 (2,1)	76.7659 (3,3)	74.8490 (3,3)
	89.5715 (2,2)	83.3594 (3,3)	* 95.0092 (1,1)
	91.3630 (2,1)	74.8829 (3,1)	* 93.7970 (2,2)
	93.6987 (2,2)	79.2703 (3,1)	* 94.8362 (1,1)

sary to compute the remaining similarity measures, and decisions can be made right away according to the measures already computed because the remaining measures are found to be always smaller. Some experimental data are shown in Table III. Totally, 50 unknown shapes taken from the three pliers are tested. Each row in Table III shows the three similarity measures between an input shape and the three reference shapes. The largest measure is marked with an "*" if the final decision is correct; otherwise, it is marked with an "X". Whenever a similarity measure larger than 95 percent is computed, hyphens are placed in the positions of the remaining similarity measures, which means that computations of these values are omitted.

Out of the 50 shapes tested, nine are misclassified. The recognition rate thus is 82 percent. The algorithms are programmed in Fortran and run on an 8-bit Zilog microcomputer developmental system connected to a multimicroprocessor image processing system [28] for image input/output and storage. Depending on how complicated a shape to be recognized is, each recognition takes from 20-40 s. Using 16-bit microcomputers or larger computers, or programming in assembly languages, will improve the recognition speed. Detailed analyses of the erroneous classifications reveals the following several possible factors for the misclassifications.

1) The limitation of the number of boundary primitives for each shape (ten only) which we impose for the purpose of increasing processing speed.

2) The low image resolution (128 X 128 only) for rather high shape complexity.

3) The rather high shape similarity between the three pliers selected for testing.

4) The use of integers only for numerical computation (to avoid low-speed real-number computation on the microcomputer) which results in poor computation accuracy.

5) Nonoptimal selection of the thresholds and constants used in the algorithms.

If the above disadvantageous factors are removed, we can expect the recognition rate to be improved. In particular, for the common shapes encountered in practical applications, which are not so mutually similar as the pliers we tested here, the recognition rate can be expected to be much higher than 82 percent.

X. CONCLUDING REMARKS

Attributed string matching with merging is proposed in this paper as a new approach to shape recognition. It is illustrated by examples to be more effective for recognizing noisy or distorted shapes than conventional string matching without merging. Only shapes without occlusion are tested. But the proposed approach is also applicable to recognition of occluded shapes. Experimental results show the feasibility of the proposed approach for general shape recognition. Some extensions of this approach are possible, such as the improvement of the operation cost functions, the inclusion of primitive splitting into the matching algorithm, more intelligent solutions to the shape orientation problem, applications of the proposed approach to recognizing occluded shapes, etc. Including merging into grammatical parsing is also interesting for further investigation.

REFERENCES

[1] T. Pavlidis, "A review of algorithms for shape analysis," *Comput. Graphics Image Processing*, vol. 7, 1978.

[2] —, "Algorithms for shape analysis of contours and waveforms," presented at 4th Int. Joint. Conf. Pattern Recognition, Kyoto, Japan, Dec. 1978.

[3] K. S. Fu and A. Rosenfeld, "Pattern recognition and image processing," *IEEE Trans. Comput.*, vol. C-25, Dec. 1976.

[4] K. S. Fu, "A step towards unification of syntactic and statistical pattern recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-5, Mar. 1983.

[5] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[6] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, vol. II. New York: Academic, 1982.

[7] K. S. Fu, *Syntactic Methods in Pattern Recognition.* New York: Academic, 1974.

[8] T. Pavlidis, *Structural Pattern Recognition.* New York: Springer, 1977.

[9] K. C. You and K. S. Fu, "A syntactic approach to shape recognition using attributed grammars," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, June 1979.

[10] W. H. Tsai and K. S. Fu, "Attributed grammar–A tool for combining syntactic and statistical pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, Dec. 1980.

[11] —, "A syntactic-statistical approach to recognition of industrial parts," presented at 5th Int. Conf. Pattern Recognition, Miami Beach, FL, Dec. 1980.

[12] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. Ass. Comput. Mach.*, Jan. 1974.

[13] R. Lowrance and R. A. Wagner, "An extension of the string-to-string correction problem," *J. Ass. Comput. Mach.*, Apr. 1975.

[14] S. Y. Lu and K. S. Fu, "A sentence-to-sentence clustering procedure for pattern analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-8, May 1978.

[15] A. V. Aho and T. G. Peterson, "A minimum distance error-correcting parser for context-free languages," *SIAM J. Comput.*, vol. 4, Dec. 1972.

[16] L. R. Bahl and G. F. Jelinek, "Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition," *IEEE Trans. Inform. Theory*, vol. IT-21, July 1975.

[17] W. H. Tsai and K. S. Fu, "A pattern deformational model and Bayes error-correcting recognition system," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, Dec. 1979.

[18] W. A. Perkins, "A model-based vision system for industrial parts," *IEEE Trans. Comput.*, vol. C-27, Feb. 1978.

[19] L. W. Fung and K. S. Fu, "Stochastic syntactic decoding for pattern classification," *IEEE Trans. Comput.*, vol. C-24, July 1975.

[20] K. S. Fu and T. L. Booth, "Grammatical inference–Introduction and survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-5, Jan. and July 1975.

[21] S. Y. Lu and K. S. Fu, "Stochastic error-correcting syntax analysis for recognition of noisy patterns," *IEEE Trans. Comput.*, vol. C-26, Dec. 1977.

[22] K. C. You and K. S. Fu, "Distorted shape recognition using attributed grammars and error-correcting techniques," *Comput. Graphics Image Processing*, vol. 13, 1983.

[23] W. H. Tsai and K. S. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, Dec. 1979.

[24] —, "Subgraph error-correcting isomorphisms for syntactic pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, Jan./Feb. 1983.

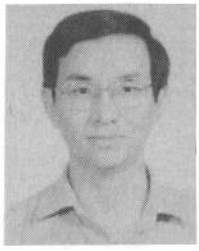
[25] H. Freeman, "Computer processing of line-drawing images," *Comput. Surveys*, vol. 6, 1974.

[26] L. S. Davis, "Understanding shapes: Angles and sides," *IEEE Trans. Comput.*, vol. C-26, Mar. 1977.

[27] H. Freeman, "Shape description via the use of critical points," presented at IEEE Comput. Soc. Conf. Pattern Recognition Image Processing, 1977.

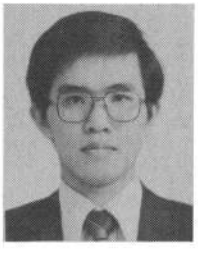
[28] W. H. Tsai, C. C. Chou, P. C. Cheng, and Z. Chen, "Architecture of a multi-microprocessor system for parallel processing of image sequences," presented at IEEE Comput. Soc. Workshop Comput. Arch. Pattern Anal. Image Database Management, 1981.

[29] A. Rosenfeld and E. Johnston, "Angle detection on digital curves," *IEEE Trans. Comput.*, vol. C-22, Sept. 1973.



Wen-Hsiang Tsai (S'79–M'80) was born in Tainan, Taiwan, Republic of China, on May 10, 1951. He received the B.S. degree from National Taiwan University, Taipei, in 1973, the M.S. degree from Brown University, Providence, RI, in 1977, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1979, all in electrical engineering.

From 1973 to 1975 he served in the Chinese Navy as an Electronics Officer. From 1977 to 1979 he worked as a Research Assistant in the Advanced Automation Research Laboratory in the School of Electrical Engineering at Purdue University. From November 1979 to August 1984 he was with the Institute of Computer Engineering at National Chiao Tung University, Hsinchu, Taiwan, Republic of China. He is now a Professor of Computer Engineering, Head of the Department of Information Science, and Assistant Director of Microelectronics and Information Science and Technology Research Center, all at National Chiao Tung University. His current research interests include image processing, pattern recognition, parallel processing, multiprocessor systems, and computer vision applications in robotics and automation.



Shiaw-Shian Yu was born in Taiwan, Republic of China, on June 20, 1959. He received the B.S. and M.S. degrees in computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1981 and 1983, respectively.

During his study at National Chiao Tung University, he worked in the Departments of Computer Engineering and Information Science as a Teaching Assistant. In the meantime, he was also a Consultant to the University Computer Center. Since November 1983, he has been with the Department of Control Systems, Electronics Research and Service Organization, Industrial Technology Research Institute, Hsinchu. His current research interests are image understanding, pattern recognition, artificial intelligence, and expert systems.