

# Approximate Stroke Sequence String Matching Algorithm for Character Recognition and Analysis

Sung-Hyuk Cha, Yong-Chul Shin, Sargur N. Srihari

Center of Excellence for Document Analysis and Recognition  
State University of New York at Buffalo, Buffalo, NY 14260, U.S.A.  
{scha,ycshin,srihari}@cedar.buffalo.edu

**Abstract** – Given two character images, we would like to measure their similarity or difference. Such a similarity or difference measure facilitates the solution to character recognition and handwriting analysis problems. There is, however, no universal definition for similarity measure satisfying wide range of characteristics such as the slant, deformation or other invariant constraints. For this reason, we propose a new definition for the character similarity measure. First, the proposed method converts a two-dimensional image into a one-dimensional string. Next, it computes the edit distance by the modified approximate string matching algorithm. We describe how to extract the string information and compute the distance and then present the details of applications in handwriting analysis and both on-line and off-line character recognitions.

**Keywords** – Approximate String Matching, Chain-code, Character Recognition, Handwriting Analysis, Pattern Matching

## 1 Introduction

“Find all letters that look like this letter.” Such a query has received a great deal of attention in *handwriting identification* and *optical character recognition (OCR)*. Unfortunately, there is no universal definition for the term “Looks like”. Although various measures have been proposed based on mathematical transforms, contour analysis, structural decompositions, etc., similarity and difference between two letters are very subjective. The *Manhattan*, *Euclidean* and *Hamming* distances that are broadly and traditionally used in *template matching* problems, could be the ones for differencing method. However, they have limitations such that they are not suitable for character images with various deformations in shape, size and rotation.

Another common method involves finding essential features to the human visual system, putting them in feature vector space and measuring the feature vectors. There have been numerous significant features used successfully to recognize digits and characters [1]: *Gradient*, *Structural*, *Concavity*, *Histogram*, *Chain-code*, etc. Also, several similarity measures are encountered in various fields [2]: *Euclidean*, *Minkowski*, *cosine*, *dot product*, *Tanimoto distance*, etc. Carefully selected features and measures provides an excellent character recognizer, yet it is a moot point whether this approach is truly compatible with people’s decision. To induce an answer to this, one may attempt to reconstruct two dissimilar images from two similar vectors that have been classified as a same character. In all, it is a non-trivial problem to find adequate features and measures to answer the preceding query comprehensively.

Motivated from the above discussion, we propose a unique definition for the character similarity in terms of its shape. It is called *Approximate Stroke Sequence String Matching*. We convert two-dimensional image data into one-dimensional data consisting of *Stroke Sequence Strings*. Next, we perform a modified *approximate string matching* technique to

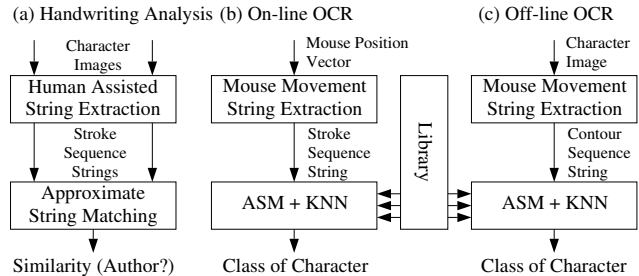


Figure 1: Applications of the similarity measure

measure the edit distance between them. Earlier definition and solution for the traditional *approximate string matching* problem are found in the literature [3] and extensive surveys on various techniques are shown in [4]. Landau and Vishkin showed an  $O(kn)$  algorithm using dynamic programming technique for the  $k$ -approximate string matching problem [5]. After building suffix trees in  $O(n)$ , the *Lowest Common Ancestor (LCA)*  $O(n)$  algorithm is applied  $k$  times.

In this paper using a newly defined edit distance based on the proposed measure, we show that the smaller the edit distance is, the more similar they look like to each other. There are three applications for which the measure is suitable as shown in Fig. 1. In Fig. 1 (a) *Handwriting Analysis* case, one would like to find out whether the given letters are similar or different or whether they were written by the same author. Among many features being currently used by the handwriting analysis practitioners or the document examiners, the *form* or *shape* is an important one for characterizing individual handwriting as it is quite consistent with most writers in normal undisguised handwriting [6]. The form can be described by a sequence of strokes and the similarity measure can be a key feature in handwriting analysis.

In addition, the suggested matching technique is applicable to solve the character recognition problem. As a stroke sequence signifies the shape of the individual letters, a letter “a” is distinguished from a letter “b” by its different stroke sequences. After extracting the stroke sequence string, we perform the *nearest neighbor searching* to identify what letter the given sequence string is. For the on-line character recognition problem, the stroke sequence string is obtained from the movements of a mouse or a pen-based device. The system is capable of enumerating all retrieved letters by increasing similarity values. Both applications as shown in Fig. 1 (a) and (b) are discussed in section 2 as they involve the manual stroke sequence string extraction.

Unlike in the on-line OCR problem, it is difficult to extract the character stroke sequence string in the off-line character recognition problem. It involves the automatic extraction of approximate stroke sequence string, which is not trivial.

In this paper, we use the contour sequence derived from a chain-code as a string instead of the stroke sequence. We achieve high accuracy rate for handwritten digit recognition experimentally. Similar recognition techniques using the differently defined edit distance on chain-codes have been proposed [7]. A method for word recognition utilizing chain-codes appears in the literature [8]. They use a statistical-style contour-based feature extraction. Our approach is contrasting as it is a syntactic-style contour-based feature extraction and matching. The detailed description of the algorithm and its problems are discussed in section 3.

## 2 On-line Character Similarity Measure

In this section, we discuss the details of *handwriting analysis* and *on-line character recognition* problems. In both of applications, it is assumed that the stroke sequence is readily available as it is extracted manually or obtained from the input device. There are two stages in computing the distance between two letters: feature extraction and matching stages.

### 2.1 Extracting stroke sequence strings

The first step involves extracting stroke sequence strings for both letters to be compared. In the on-line character recognition case, stroke sequence strings are readily available in the on-line character acquisition process. In the handwriting analysis case, a trained human, known as a document examiner, is involved during the examination. Albeit impossible to extract the exact stroke sequence of the author of the letter, it is fairly easy for a human, especially for document examiners, to extract the pseudo on-line information. We utilize the pen based system rather than a mouse, to allow an examiner to extract the author's pseudo on-line information while he or she is tracing the strokes on screen. We define a stroke as a directional arrow whose length is 7 pixels long. There are 8 types of strokes:  $\{\rightarrow, \nearrow, \uparrow, \nwarrow, \leftarrow, \swarrow, \downarrow, \searrow\}$ , and contiguous strokes are represented in a pair of parentheses. Each stroke has its corresponding integer value:

$$\begin{pmatrix} \nwarrow & \uparrow & \nearrow \\ \leftarrow & & \rightarrow \\ \swarrow & \downarrow & \searrow \end{pmatrix} = \begin{pmatrix} 3 & 2 & 1 \\ 4 & & 0 \\ 5 & 6 & 7 \end{pmatrix}$$

Fig. 2 illustrates a sample character stroke sequence for a character "A".

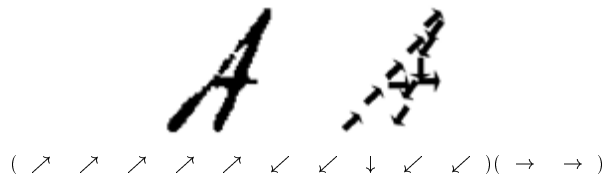


Figure 2: Manually obtained stroke sequence strings

### 2.2 Stroke Sequence String Matching

The next step is the comparison step that uses the modified approximate string matching algorithm. The stroke string matching problem defined below is quite different from the traditional string matching problem in terms of the type of elements in a string. The type of elements in the traditional string matching problem is either bounded or unbounded alphabets. No relation is defined among the alphabets, hence,

it is called nominal. In our case, the type of element is not nominal, rather it is directional. The individual distance  $d_{i,j}$  between the  $i$ 'th stroke in letter  $a_1$  and the  $j$ 'th stroke in letter  $a_2$  is defined as follows:

**Definition 1 distance between two individual strokes (turn):**

$$d_{i,j} = \begin{cases} |a_1(i) - a_2(j)|, & \text{if } |a_1(i) - a_2(j)| \leq 4 \\ 8 - |a_1(i) - a_2(j)|, & \text{otherwise} \end{cases}$$

One can edit a stroke to make the other stroke by turning it to the clockwise or counter-clockwise whichever is shorter. For example, the distance between the strokes  $\uparrow$  and  $\rightarrow$  is 2 as one can turn  $\uparrow$  two steps to the right to make  $\rightarrow$ . The term  $d_{i,j}$  denotes the minimum number of necessary steps in turning. If the number of stroke types increases or decreases, Def. 1 of edit distance changes accordingly. This operation, *turn* is another important difference between the traditional and our modified string matchings. While the former allows the substitution with the cost of 1, the later allows the turn with various costs.

The approximate stroke sequence string matching problem is that given two stroke sequence strings, find the minimum edit distance between them by allowing turn, insertion and deletion. Consider two sample "A" letters,  $a_1$  and  $a_2$ :

$$a_1 : (\uparrow \nearrow \uparrow \nearrow \rightarrow \searrow \downarrow \searrow \downarrow) (\searrow \nearrow) \\ (2 \ 1 \ 2 \ 1 \ 0 \ 7 \ 6 \ 7 \ 6) \quad (7 \ 1) \\ a_2 : (\nearrow \nearrow \nearrow \nearrow \nearrow \swarrow \swarrow \downarrow \swarrow \swarrow) (\rightarrow \rightarrow) \\ (1 \ 1 \ 1 \ 1 \ 1 \ 5 \ 5 \ 6 \ 5 \ 5) \quad (0 \ 0)$$

We are allowed to insert, delete strokes or turn an individual stroke. The cost of turn is  $d_{i,j}$ . The costs of insertion and deletion are  $c = 2$ . Fig. 3 illustrates the distance computing table for the first parts of sample letters. First, put one string

		2	1	2	1	0	7	6	7	6
	0	2	4	6	8	10	12	14	16	18
1	2	1	2	4	6	8	10	12	14	16
1	4	3	1	3	4	6	8	10	12	14
1	6	5	3	2	3	5	7	9	11	13
1	8	7	5	4	2	4	6	8	10	12
1	10	9	7	6	4	3	5	7	9	11
5	12	11	9	8	6	5	5	6	8	10
5	14	13	11	10	8	7	7	6	8	9
6	16	15	13	12	10	9	8	7	7	8
5	18	17	15	14	12	11	10	9	9	8
5	20	19	17	16	14	13	12	11	11	10

Figure 3: Computing distance table

on the top of the table, and the other in the left side of the table. The individual strokes on the top and left side of the table are denoted as  $t_i$ 's and  $l_j$ 's, respectively. The initial values are assigned in the first row and column of the table with  $T[0, j] = c \times j$  and  $T[i, 0] = c \times i$ . Now  $T[i, j]$  is computed by taking the minimum value of these three:

$$T[i, j] = \min \begin{cases} T[i \Leftrightarrow 1, j \Leftrightarrow 1] + d_{i,j} & \Leftrightarrow \text{turn} \\ T[i \Leftrightarrow 1, j] + c & \Leftrightarrow t_i \text{ is missing} \\ T[i, j \Leftrightarrow 1] + c & \Leftrightarrow l_j \text{ is missing} \end{cases}$$

The distance between two stroke sequences is achieved from the table in Fig. 3 where the edit distance is 10.

Many English characters consist of more than one stroke sequences. This leads a problem of finding the closest pair of

strings. To solve this, we consider the length and the starting position of each string and then the final distance is computed by adding all distances of closest pairs of strings.

### 2.3 Example

Some sample letters from *bd\_testing set* [9] are shown in Fig. 4 as examples (see [10] for description of this character set in detail). Fig. 5 shows the stroke sequence strings for

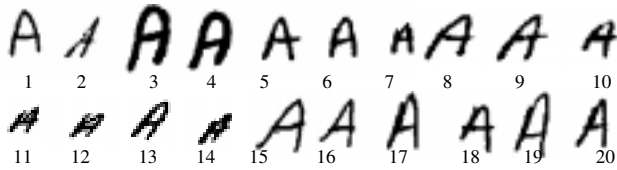


Figure 4: Sample A's

the examples. They are obtained after the size normalization

	1st part	2nd part
1	↑↑↑↑→↓↓↓↓	↘↘↘
2	↗↗↗↗↘↘↘↘	→→→
3	↗↗↗→↓↓↓↓	→↗
4	↗↗↗↘↓↓↓↓	→↗
5	↗↗↗↘↓↓↓↓	→→↗
6	↓↓↓↓↑↑↑↑↘↘↘↘	↗→
7	↓↓↓↓↑↑↑↑↘↘↘↘	→→→
8	↗↗↗↘↓↓↓↓	→→→
9	↗↗↗↘↓↓↓↓	→→→
10	↗↗↗↘↓↓↓↓	→→→
11	↘↘↘↘↗↗↗↗↘↘↘↘	→→→
12	↘↘↘↘↗↗↗↗↘↘↘↘	→↗
13	↘↘↘↘↗↗↗↗↘↘↘↘	→↗
14	↘↘↘↘↗↗↗↗↘↘↘↘	→↗
15	↗↗↗↘↓↓↓↓	→↗
16	↗↗↗↘↓↓↓↓	↗→
17	↓↓↓↓↑↑↑↑↘↘↘↘	↗↗
18	↓↓↓↓↑↑↑↑↘↘↘↘	→→→
19	↓↓↓↓↑↑↑↑↘↘↘↘	→↗
20	↘↘↘↘↗↗↗↗↘↘↘↘	→↗

Figure 5: Stroke Sequence Strings for "A"s

process. Fig. 6 shows the distance matrix obtained from the stroke sequence string matching algorithm for the example. Elements of the resulting matrix  $M$  have properties of iden-

1	0	$M[x, y]$	$= 0 \Leftrightarrow x = y$
2	12		$\geq 0 \Leftrightarrow x \neq y$
3	5		$= M[y, x]$
4	5		$\leq M[x, z] + M[z, y]$
5	7		
6	13		
7	14		
8	12		
9	14		
10	11		
11	23		
12	25		
13	19		
14	17		
15	7		
16	10		
17	15		
18	15		
19	13		
20	15		
1	2	3	4
2	5	9	0
3	5	9	0
4	7	9	7
5	13	16	13
6	14	13	13
7	12	8	9
8	14	8	11
9	11	5	10
10	23	13	20
11	25	16	22
12	19	11	16
13	17	12	14
14	7	5	5
15	10	5	9
16	15	17	14
17	15	14	14
18	15	14	14
19	13	13	12
20	15	14	14

Figure 6: Distance matrix between sample A's

tity, non-negativity, commutative law and triangle inequality.

### 2.4 On-line Character Recognition

A simple *On-line Character Recognizer* with the supervised training ability is also implemented. One can draw a character and provides a truth information about the character to train the system. Training means adding the stroke sequence strings in the library with its truth. Later one can draw a character and ask the system to recognize it. The system will perform the nearest neighbor search in the reference set and returns the class of the string whose distance is the minimum.

There are two additional operations for the on-line character recognition. They are merging two sequence strings and reversing a string. One or several breaks may occur in drawing a single line, curve or circle. This must be considered as a single string rather than multiple strings. To obviate this inadequacy, the *merge* operation is necessary. If the starting position of a certain string and the ending position of another string are matched or close, they are merged. The *reverse* operation is also needed because people may write a letter in the reverse order.

### 3 Off-line Character Matching

In this section, we discuss the implementation of the new similarity measure for the off-line character matching problem. An off-line character/digit recognizer takes a character or digit image as an input and classifies its class as shown in Fig. 1 (c).

#### 3.1 Contour Sequence String Extraction

To obtain pseudo stroke sequence strings from off-line character images, a contour sequence is defined using the idea of the chain-code. Like a chain-code, a contour sequence is a representation of boundaries of objects in the image. We distinguish the outer and inner contour sequence; the former is the outer boundary due to a blob of black pixels and the later is due to a lake of white pixels inside of a blob of black pixels. Fig. 7 shows the contour sequence representation of a character "A". There are one outer contour sequence and



Figure 7: A contour sequence representation for "A"

one inner contour sequence. We define the abstract data type of a contour sequence string as follows: inner or outer type, its length, a list of strokes and the centroid. A centroid is the center of a contour sequence:

$$(\bar{x}, \bar{y}) = \left( \frac{\sum_x \sum_y xB(x, y)}{\sum_x \sum_y B(x, y)}, \frac{\sum_x \sum_y yB(x, y)}{\sum_x \sum_y B(x, y)} \right)$$

$B(x, y) = 1$  if the pixel is labeled as an element of the segment and  $B(x, y) = 0$  otherwise. Note that the centroid is computed after the character image is resized by adjusting the height to a fixed size. Both type and *centroid* are important because a character image has multiple contour sequence strings and the decision for selecting a corresponding contour sequence string must be made. There are 8 directional types of stroke in a chain-code. The size of each stroke is 7 pixels long.

To extract contour sequence strings, several procedures are preceded such as noise removal, connected component analysis and chain-code generation. Although we expect the input character or digit image to be a single clean image, there are many undesired noises. Some broken or smeared character images occur due to the degradation of document images. Moreover, poor segmentation and improper background removal create certain noises. While the image restoration is beyond the scope of this paper, we use simple unary and binary filters that handle salt and pepper noises. The *i-dots* in *i* and *j* are preserved.

Second, perform a connected component analysis and find the centroid of every connected component. Next, generate chain-codes by following component outer most boundaries. Finally, starting from the top of each chain-code, generate the contour sequence by fitting strokes to the chain-code. A stroke is 7 pixel long. Geometrically best fitting strokes are selected to replace pixel-based strokes in the chain-code.

### 3.2 Contour Sequence String Matching

The algorithm is very similar to the one stated earlier in subsection 2. The major difference is the source of strings. We provide a succinct description of the algorithm. First, we find the closest pair of contour sequence strings. We consider the type, centroid and length of a string as criteria. The distance between the centroids and difference in lengths must be small. Furthermore, we consider the number of contour sequence strings as well. If the difference in the number of strings is too big, we eliminate it from consideration. For example, a digit “1” typically has one string whereas a digit “8” has 3 strings. When “1” is a query, we limit our search templates whose number of strings is less than 3. This enables to expedite searching.

After finding all corresponding contour sequence strings, compute each edit distance and accumulate all distances. If there is a string without a corresponding one, add the length of the string times the penalty value 2 to the total edit distance. Finally, select top 5 similar templates whose edit distances are smallest. The class of a query image is determined from a vote of these 5 templates.

## 4 Experiments

**Handwriting Analysis:** To show the compatibility of the new similarity measure with the human decision, we asked 25 professional scientists and graduate students at CEDAR to select similar pairs from the list in Fig. 4. It is noted that similar letters in human eyes tend to have small edit distance values between them. The pairs  $\{(3, 4), (11, 12), (8, 9)\}$  are chosen to be similar by more than 15 people and their edit distances in Fig. 6 are 0, 4, 2, respectively. Interestingly, 12 people responded (1, 6) as a match while the edit distance is 13. This is due to the leading ligature in the letter number 6. Considering the leading ligature, the other 13 people’s decision is more appealing.

**Off-line Digit Recognition:** We consider 18,465 digit images from *br\_testing set* [9]. 1,000 out of these images are chosen as query images and the rest of them are used as the reference or prototype images. Several number of selecting queries were performed and the average accuracy is about 96.08%. From the error analysis, we find an interesting consistency that the majority of errors are due to the broken characters resulting unexpected discontinuity in

the contour sequence string. It is expected that provision on the broken characters will improve the performance significantly. Note that the performance of GSC is 98.87% [11] and many other methods with the above 97% performance are also known [1].

We tested several different penalty values, *c* for insertion and deletion and we observe that the best performance occurs when  $c = 2$ . We also performed the experiments with and without the pepper and salt filters. The former performs 2% better than the later. There are two parameters that affect the performance. They are the number of types of stroke(directions) and the size of each stroke. We used 8 directional strokes with a size of 7. In a later study, further testing with more than 8 directions with various size will be carried out and a better performance is expected as an analysis on errors has indicated that a new strategy in handling noisy images (broken characters) can correct the errors properly.

## 5 Conclusion

We proposed the approximate stroke or contour sequence string matching algorithm. It is approximate matching because the retrieval is based on similarity rather than the exact matching. Thus, it is useful to retrieve the similar style of handwritings with a certain degree of variations since the results are quite comparable with human subjective decision. The new similarity measure has a plethora of potential and important roles in character recognition and analysis.

## References

- [1] CEDAR, “Intelligent character recognition.” <http://www.cedar.buffalo.edu/ICR>, 1997.
- [2] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. New York, Wiley, 1st ed., 1973.
- [3] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” *Journal of the ACM*, vol. 6, pp. 168–178, Jan 1974.
- [4] P. A. V. Hall and G. R. Dowling, “Approximate string matching,” *ACM Computing Surveys*, vol. 12, pp. 381–402, Dec 1980.
- [5] G. M. Landau and U. Vishkin, “Introducing efficient parallelism into approximate string matching and a new serial algorithm,” in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pp. 220–230, 1986.
- [6] R. R. Bradford and R. B. Bradford, *Introduction to Handwriting Examination and Identification*. Nelson-Hall Publishers: Chicago, 1992.
- [7] A. Marzal and E. Vidal, “Computation of normalized edit distance and applications,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 926–932, Sep 1993.
- [8] G. Kim and V. Govindaraju, “Efficient chain code based image manipulation for handwritten word recognition,” in *the SPIE sym. on electronic imaging science and technology(Document Recognition III)*, vol. 2660, pp. 262–272, Feb 1996.
- [9] CEDAR, “<http://www.cedar.buffalo.edu/databases/cdrom1/>”
- [10] J. J. Hull, “A database for handwritten text recognition research,” *Transactions on Pattern Analysis and Machine Intelligence*, pp. 550–554, May 1994.
- [11] J. T. Favata, G. Srikantan, and S. N. Srihari, “Handprinted character/digit recognition using a multiple feature/resolution philosophy,” in *IWFHR-IV*, pp. 57–66, Dec 1994.