Contour regularity extraction based on string edit distance

José Ignacio Abreu Salas[1] and Juan Ramón Rico-Juan[2]

[1] Universidad de Matanzas,Cuba
jose.abreu@umcc.cu
[2] Dpto Lenguajes y Sistemas Informáticos, Universidad de Alicante, Spain
juanra@dlsi.ua.es

Abstract. In this paper, we present a new method for constructing prototypes representing a set of contours encoded by Freeman Chain Codes. Our method build new prototypes taking into account similar segments shared between contours instances. The similarity criterion was based on the Levenshtein Edit Distance definition. We also outline how to apply our method to reduce a data set without sensibly affect its representational power for classification purposes. Experimental results shows that our scheme can achieve compressions about 50% while classification error increases only by 0.75%.

1 Motivation

Finding a set of representative prototypes from a group of contour instances is often useful for improving a classifier response time and to simplify data to be analysed by a human interpreter [1]. Different approaches have been proposed in the literature, such as [1] [2] and [4], based on the computation of the mean shape or inferring prototypes by some *ad-hoc* procedure [9].

However, there are contexts where getting good prototypes are not sufficient because the lack of an understandable criterion about its constitution and performance. For example, with forensic purposes its important to construct a model characterising an individual handwriting style not in a black box sense but taking into account about the relations among different handwriting constitutive elements.

In this work, we present a new model for computing a set of contour prototypes based on the identification of contour segments satisfying some similarity criterion which can be controlled by the user. In section 2, we explain some related techniques which have been used by our method. Section 3 describes our approach, first an algorithm to construct a prototype from two contour instances and latter the application of this algorithm to construct prototypes from a set of contour instances. Finally, some experimental results are showed in section 4.

2 Background and Notation

2.1 Extended Freeman Chain Codes

The classical chain codes can be used to represent a contour by traversing it to produce a string of codes which describes the direction to the next point on the contour. When images are represented as a square grid, usually eight codes are defined to describe a square neighbourhood.

In our work, contours where encoded by an extension of classic Freeman chain codes [6]. To represent a contour, we have not defined a fixed number of codes, instead, we consider an infinite set of directions at range $0 \leq d < 8$ and the symbol "*?*" to denote an undefined direction.

2.2 Edit Distance

As a distance function for strings we take the Levenshtein string edit distance. Let Σ be an alphabet and $S_1 = \{S_{11}, S_{12}..S_{1m}\}$, $S_2 = \{S_{21}, S_{22}..S_{2n}\}$ two strings over Σ where $m, n \geq 0$, the edit distance between S_1 and S_2, $D(S_1, S_2)$, is defined in terms of elementary edit operations which are required to transform S_1 into S_2. Usually three edit operations are considered:

- *substitution* of a symbol $a \in S_1$ by a symbol $b \in S_2$, denoted as $w(a, b)$
- *insertion* of a symbol $b \in \Sigma$ in S_1, denoted as $w(\varepsilon, b)$
- *deletion* of a symbol $a \in S_1$, denoted as $w(a, \varepsilon)$.

where ε denotes an empty string. Let $E = \{e_1, e_2, ..., e_k\}$ be a sequence of edit operations transforming S_1 into S_2, if each operation have cost $c(e_i)$ the cost of E is $c(E) = \sum_{i=1}^{k} c(e_i)$ and the edit distance $D(S_1, S_2)$ is defined as:

$$D(S_1, S_2) = argmin\{c(E)| \text{ E an edit sequence to transform } S_1 \text{ into } S_2\} \ .(1)$$

In our case, cost are computed as follows:

$$c(w(a,b)) = \begin{cases} min\{|a-b|, 8-|a-b|\} & \text{if } a, b \neq \text{“?”}. \\ 2 & \text{in other case.} \end{cases} \quad (2)$$

The number 2, corresponding to insertion and deletion operations, is an half of the maximum substitution operation. The same fixed number is used in [10]. The dynamic programming algorithm exposed by Wagner and Fisher [5] lets to compute $D(S_1, S_2)$ in $O(n \times m)$ time.

3 Problem Definition and Solution Outline

Let two contours encoded by the chain codes S_1 and S_2, we address to construct a prototype S_3 by finding two sets (C_S, C_T) of pairs $(S_1(k,l) \in S_1, S_2(g,h) \in S_2)$ where C_S contains pairs encoding contour segments that fulfill a similarity criterion while C_T holds dissimilar regions. Each element at C_S or C_T determines a S_3 contour region and by merging all together we build the prototype. Next subsections explain how we get S_3 segment from $S_1(k,l)$ and $S_2(g,h)$ (*fusion operation*) and the way to construct C_S and C_T.

3.1 String Fusion Operation

Let S_1 and S_2 strings encoding two contour sections, the *fusion operation* can be defined as $F(S_1, S_2) = S_3$, where S_3 satisfies:

$$D(S_1, S_2) \geq D(S_1, S_3) \ \wedge \ D(S_1, S_2) \geq D(S_2, S_3) \ . \tag{3}$$

Thus, if we consider $D(S_k, S_l)$ be a measure how well represented is a string S_k by S_l, this is, if $D(S_k, S_j) < D(S_k, S_l)$ holds, it means S_j represent better S_k than S_l, we can say that fusion operation ensures S_3 describe S_1 and S_2 as well they represent each other or even better.

Let examine how to define $F(S_1, S_2)$. Be S_1, S_2 two strings with respective lengths L_1 and L_2, S_3 length are fixed by:

$$L_3 = \left\lfloor \frac{(L_1 + L_2)}{2} \right\rfloor \ . \tag{4}$$

If $L_1 = L_2$, a simple way to construct S_3 is computing its first symbol $S_3[1]$ as the mean of $S_1[1]$ and $S_2[1]$, and so on, this procedure ensures S_3 satisfies constrains (3).

However, a more general scheme must be consider to compute S_3, since most times $L_1 \neq L_2$, so the simple criterion used above can not be applied. Without lost of generality, lets $L_1 > L_2$, so $L_2 \leq L_3 < L_1$. It means that S_3 symbols cant be determined by the mean of only two symbols at S_1 and S_2. Now, a symbol $S_3[i]$ is computed voting over all $S_1[h]$ and $S_2[k]$, where the weight W_j for the j-esime symbol of S_1 or S_2 are determined by:

$$W_j = \begin{cases} 1 & \text{if } \ j \geq i\frac{L_1}{L_3} \ \wedge \ (j+1) \leq (i+1)\frac{L_1}{L_3} \ . \\ (i+1)\frac{L_1}{L_3} - j & \text{if } \ i\frac{L_1}{L_3} \leq j \leq (i+1)\frac{L_1}{L_3} \ \wedge \ (j+1) > (i+1)\frac{L_1}{L_3} \ . \\ \frac{L_1}{L_3} & \text{if } \ j < i\frac{L_1}{L_3} \ \wedge \ (j+1) \geq (i+1)\frac{L_1}{L_3} \ . \\ j + 1 - i\frac{L_1}{L_3} & \text{if } \ j < i\frac{L_1}{L_3} \ \wedge \ (j+1) < (i+1)\frac{L_1}{L_3} \ . \end{cases} \tag{5}$$

Since a symbol from S_1 or S_2 represent a direction d, have no sense multiplying by its weight W_d and sum all of them, so we consider pairs $< d, W_d >$ like vectors with direction d and magnitude W_d, this way S_3 symbols can be computed as the direction of a vector obtained by the sum of each $< d, W_d >$.

Fig. 1 shows the result string S_3 when applied the fusion procedure over $S_1 = \{0, 0, 0, 0\}$ and $S_2 = \{2, 2\}$.

3.2 Finding Similar Regions

The algorithm to compute Edit Distance explained in section 2.2 from two strings S_1 and S_2 determines the minimum cost edit sequence (Q) to transform S_1 in S_2. Substitutions at Q maps a S_1 symbol to other at S_2, deletions denotes a

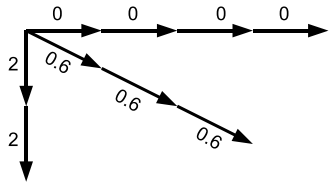

Fig. 1. Result of fusion operation over two strings.

S_1 symbol without image at S_2 and insertions symbols at S_2 that is not the image of any S_1 symbol. Thus, given a subsequence of edit operations $Q(k,l)$ from Q , it maps a contour segment ${S_1}^{k,l}$ from S_1 to other ${S_2}^{k,l}$ on S_2 and we can compute the cost $E_{Q(k,l)}$ which represent those edit operations involved at $Q(k,l)$ over the edit distance.

To select similar regions from S_1 and S_2, we fix a parameter T called *tolerance factor*, $0 \leq T \leq 1$, which determines a set C_S of non-overlapping subsequences from Q satisfying the following constraint:

$$\sum E_{Q(k,l)} \leq D(S_1, S_2)T \mid Q(k,l) \in C_S \ . \tag{6}$$

We can see, there are multiples sets satisfying constraints (6). Since information from those $C_S \in Q(h,g)$ will be partially ignored, we are also interested finding a C_S subject to:

$$argmax\{\sum L(Q(k,l))|\ Q(k,l) \in C_S\} \ . \tag{7}$$

where $L(S)$ denotes the length of a string S. This constraint aims to find a set of similar regions covering as contour length as possible. Finally, we prefer solutions where C_S contains a few long length regions instead a large set of short regions.

Searching a set satisfying such conditions became a computational expensive problem, thus, defining:

$$P_{Q(k,h)} = \begin{cases} 2 \text{ if } Q(n,k-1) \ \wedge \ Q(h+1,m) \in C_S. \\ 1 \text{ if } Q(n,k-1) \in C_S \ \wedge \ Q(h+1,m) \ni C_S \text{ or viceversa} \ . \\ 0 \text{ if } Q(n,k-1) \ni C_S \ \wedge \ Q(h+1,m) \ni C_S. \end{cases} \tag{8}$$

we applied a heuristic approach to get C_S that is good enough, by the greedy procedure outlined below:

Let:

- C_T `a set of all subsequences` $Q(k,k)|k = 0..L(Q)$
- $C_S = \emptyset$

while $C_T \neq \emptyset$ **and** $\sum E_Q(k,l) \leq D(S_1,S_2)T \mid Q(k,l) \in C_S$

- **get** $Q(g,g) = argmin\{E_{Q(k,k)}|Q(k,k) \in C_T\}$`: if there is two or more select one with the smallest` $P_Q(g,g)$

- $C_S = C_S \cup Q(g,g)$
- $C_T = C_T \sim Q(g,g)$
- **case** $P_Q(g,g)$:
  - **2:** `replace` $Q(n,g-1), Q(g+1,m)$ `and` $Q(g,g)$ `at` C_S `by` $Q(n,m)$
  - **1:** `replace` $Q(n,g-1)$ `or` $Q(g+1,m)$ `and` $Q(g,g)$ `at` C_S `by` $Q(n,g)$ `or` $Q(g,m)$ `at case`

end while

3.3 Prototype Construction

To construct a prototype representing two contours encoded by the chain codes S_1 and S_2 we uses the algorithm to find similar regions and the fusion procedure defined before. First, we get sets C_S, containing those regions meeting the similarity criterion, and C_T which contains dissimilar contour segments. For each $Q(k,l) \in C_S$ get both regions ${S_1}^{k,l}$ and ${S_2}^{k,l}$ it determines and compute ${S_3}^{k,l} = F({S_1}^{k,l}, {S_2}^{k,l})$. To construct ${S_3}^{g,h}$ from regions at C_T we need its length, which is compute by equation (4), this time, each ${S_3}^{g,h}[i]$ contains the symbol "*?*". Finally, we get the prototype concatenating each ${S_3}^{g,h}$ which have been sorted by g. Now, we illustrate the behaviour of the algorithm by an example:

Be $S_1 = \{2, 2, 5, 1.5, 1.5, 1, 1, 1\}$ and $S_2 = \{3, 2, 1.6, 1.5, 7, 0.3, 0.3\}$; $T = 0.7$. Table 1 shows Q and cost for each edit operation.

Table 1. Minimum Cost Edit Sequence and costs.

Q	$w(2,3)$	$w(2,2)$	$w(5,\varepsilon)$	$w(1.5,1.6)$	$w(1.5,1.5)$	$w(1,7)$	$w(1,0.3)$	$w(1,0.3)$
Cost	1	0	2	0.1	0	2	0.7	0.7

It drives to $C_S = \{Q(0,4), Q(6,7)\}$ and $C_T = \{Q(5,5)\}$. From $Q(0,4)$ we get $S_1^{0,4} = \{2, 2, 5, 1.5, 1.5\}$; $S_2^{0,4} = \{3, 2, 1.6, 1.5\}$ and by the fusion $S_3^{0,4} = \{2.5, 2, 1.5, 1.5\}$. $Q(6,7)$ turns $S_1^{6,7} = \{1, 1\}$; $S_2^{6,7} = \{0.3, 0.3\}$ and $S_3^{6,7} = \{0.65, 0.65\}$. Putting all $S_3^{0,4}$, $S_3^{6,7}$ and $S_3^{5,5} = \{$"?"$\}$ together we get a prototype: $S_3 = \{3, 2, 1.6, 1.5, ?, 0.65, 0.65\}$

3.4 Building prototypes from a contour set

Actually our algorithm can handle only two instances at time, thus we need to define an additional procedure if want to get prototypes from a set C_c greater than two objects. To address this problem we iteratively find instances $S_i, S_j \in C_c$ which meet $argmin\{D(S_k, S_l) | S_k, S_l \in C_c\}$, from those instances we build a prototype S_p, which will replace S_i and S_j from C_c. This procedure is repeated while holds:

$$\frac{L_i - C_?(S_i) + L_j - C_?(S_j) - 2C_?(S_p)}{L_i - L_j} \leq P \ . \tag{9}$$

where $C_?(S_x)$ counts the amount of symbols from S_x equals to "?", and P $(0 \leq P \leq 1)$, called *persistence factor*, is a measure of how many information haven been lost by the fusion operation.

4 Experimental Results

The proposed algorithm was applied to construct a set of prototypes from a group of contour instances represented by chain codes. Separated experiments where carried with two independent contour subsets, containing digits and letters respectively, from the NIST SPECIAL DATABASE 3 of the National Institute of Standards and Technology. With this subsets we perform a 4-fold crossvalidation where each train and test sets have 60 and 20 instances per class respectively.

First, we classified every instance at test set by the *nearest neighbour rule* and compute the absolute error commited. Later, we use our scheme separately over instances from the same class at the train set to get a new set which will replace the former. We compute a *compression index* C that is a measure (as percent) of how much we reduce de original train set, and by classifying again the test set, we get the *error rate variation*. These values characterised the algorithm behaviour since we hope improve the compression index without sensibly increases the classification error.

We try different values for *tolerance* and *persistence* factors to analyse how they affect both compression and error rate. As Table 2 shows, our algorithm performs well for some T and P values. Fig 2 illustrates how this values determines the similar segments from contours. We also are interested into compare how

Table 2. Values for C and ΔE when $T = 0.5$ and $P = 0.1$ (digits experiment)

Fold	C	ΔE
0	52.5	0
1	53.0	0.5
2	54.66	0
3	52.33	2.5
Mean	53.12	0.75

values for T and P determines both compression and error rate variation across different data sets. Graphics at Fig 3 shows our algorithm performs similarly through both databases for the compression index, similar results was obtained for the error rate variation.

5 Conclusions and future work

A new method to construct prototypes from a set of contours instances was presented. Our approach lets to identify similar segments through contours by

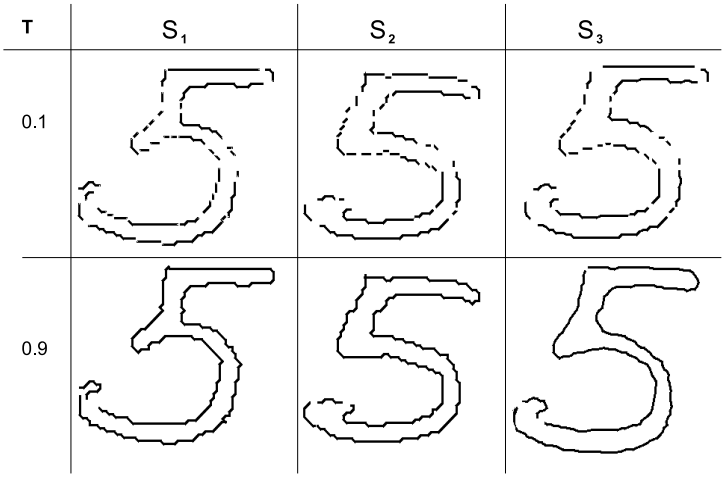

Fig. 2. Each row shows similar segments from two contours S_1 and S_2 and the correspondent ones at the build prototype from different T values.

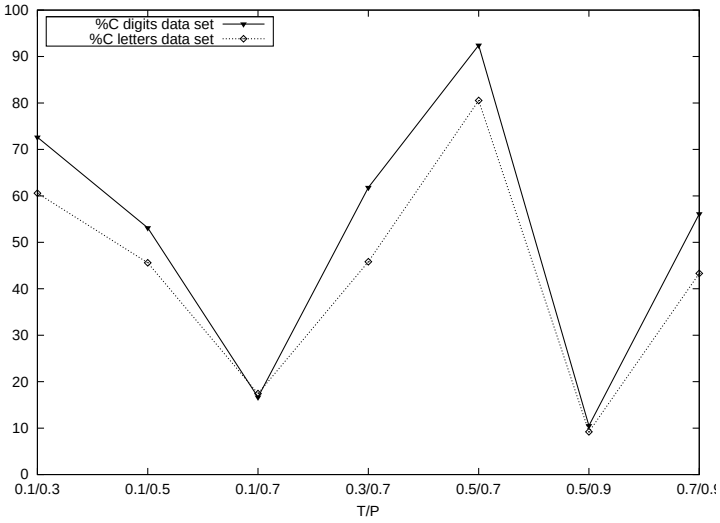

Fig. 3. Values for C through different T and P for digits and letters (*dashed line*) data sets.

a criterion that can be adjusted by the user, and, from those segments we get the prototypes. Experiments shows our model is suitable for compress contours data sets without affect its representative power. We think our approach can be used for characterise a class of contours and for data compression. Further investigations can be addressed to extend the algorithm to other representations for contours, like cyclic strings.

Acknowledgements

This work is partially supported by the Spanish CICYT under project DPI2006-15542-C04-01 and by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018).

References

1. Duta, N., Sonka, M., Jain, A.:Learning Shape Models from Examples Using Automatic Shape Clustering and Procrustes Analysis.Information Processing in Medical Imaging. LNCS, vol. 1613, pp. 370-375.(2008)
2. Jiang, X., Abegglen, K., Bunke, H., Csirik, J.:Dynamic computation of generalised median strings. Journal Pattern Analysis and Applications, vol 6, pp. 185-193. (2003)
3. Jiang, X.,Schiffmann, L., Bunke, H.:Computation of median shapes. 4th Asian Conference on Computer Vision.(2000)
4. Cárdenas, R.:A Learning Model for Multiple-Prototype Classification of Strings.In:17th International Conference on Pattern Recognition, vol. 4, pp. 420-42.(2004)
5. Wagner, R., Fischer, M.:The String-to-String Correction Problem. Journal of the ACM, vol. 21, pp. 168-173.(1974)
6. Freeman, H.:Computer Processing of Line-Drawing Data. Computer Surveys, vol. 6, pp. 57-96.(1974)
7. Chikkerur, S., Wu, C., Govindaraju, V.: A Systematic Approach for Feature Extraction in Fingerprint Images.First International Conference on Biometric Authentication. LNCS, vol 3072, pp. 344-350.(2004)
8. Govindaraju, V., Shi, Z., Schneider, J.: Feature Extraction Using a Chaincoded Contour Representation of Fingerprint Images. AVBPA, pp. 268-275.(2003)
9. Duta, N., Jain, A., Dubuisson-Jolly, M.: Automatic Construction of 2D Shape Models. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, pp. 433-446.(2001)
10. Rico-Juan, J. R, Mico L.: Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. Pattern Recognition Letters, vol. 24, pp. 1417-1426.(2003)