Elastic Structural Matching for On-line Handwritten Alphanumeric Character Recognition

Kam-Fai Chan & Dit-Yan Yeung *

Department of Computer Science
Hong Kong University of Science & Technology
Clear Water Bay, Kowloon, Hong Kong
E-mail: {kchan,dyyeung}@cs.ust.hk

Abstract

In this paper, we will propose a simple yet robust structural approach for recognizing on-line handwriting. Our approach is designed to achieve reasonable speed, fairly high accuracy and sufficient tolerance to variations. Experimental results show that the recognition rates are 98.60% for digits, 98.49% for uppercase letters, 97.44% for lowercase letters, and 97.40% for the combined set. When the rejected cases are excluded from the calculation, the rates can be increased to 99.93%, 99.53%, 98.55% and 98.07%, respectively. On the average, the recognition speed is about 7.5 characters per second running in Prolog on a Sun SPARC 10 Unix workstation and the memory requirement is reasonably low.

1. Introduction

Character recognition has been an active research area for more than 30 years [12]. Different approaches, such as statistical, syntactic and structural, and neural network approaches, have been proposed. Characters consist of line segments and curves. Different spatial arrangements of these elements form different characters. In order to recognize a character, we should find out the structural relationships between the elements which make up the character. However, in the pattern recognition community, the syntactic and structural approach is always considered a nice idea, but is not particularly promising in most applications [8]. One of the concerns is the need for robust extraction of primitives. This issue will be addressed later in this section.

Using the structural approach, two-dimensional patterns, such as characters, can be represented in at least two different ways. The first one is to use a representation formalism which is by nature of high dimensionality, such as an array, a tree or a graph [9, 10]. The second one is to incorporate additional relationships between primitives into a one-dimensional representation form. Two well-known methods using the latter approach are the picture description language (PDL) [11] and the plex grammar [5].

Note that we need to consider the trade-off between expressive power and time complexity for processing when we choose any representation formalism. Graphs have the highest expressive power, but the detection of exact or approximate subgraph isomorphism is known to be intractable [2]. On the other hand, string matching is of polynomial time complexity, but its expressive power is much lower. When efficiency is one of our major concerns, then string representations are generally preferred.

The PDL and the plex grammar are mainly used to describe how primitives are connected. These schemes may become very tedious when there exist large variations in the character classes. Berthod and Maroy's primitives [1] attempt to address the problem of high variability. However, their method does not make use of directional information. On the other hand, Freeman's chain code [6] and some extended schemes (e.g., [13]) use directional information to form primitives, although the resulting representations are often not compact enough.

In this paper, we will propose a simple structure which is compact and at the same time contains useful directional information. Then we will discuss how to extract structural primitives from the input in a robust manner. Afterwards, we will explain how to perform classification through model matching. Finally, we will present and discuss some experimental results followed by concluding remarks.

2. Primitives of the structure

Characters are composed of line segments and curves. Every line segment or curve can be extended along a certain direction. A curve that joins itself at some point forms a loop. Hence, in our representation, we will use as primi-

*This research work is supported in part by the Hong Kong Research Grants Council (RGC) under Competitive Earmarked Research Grant HKUST 746/96E awarded to the second author.

tives different types of line segments and curves with some directional information. Note that there may be several different primitives in even just one stroke. Basically, there are five types of primitives:

- line,
- up (curve going counter-clockwise),
- down (curve going clockwise),
- loop (curve joining itself at some point), and
- dot (a very short segment which may sometimes be just noise; we, however, cannot simply ignore it since it may be part of a character, like in "i" and "j").

To represent the directional information, we will use Freeman's chain code [6] which uses eight values, i.e., 0 to 7 (corresponding to $0^\circ, 45^\circ, \ldots, 315^\circ$, respectively), to indicate how the current point is connected to the next one.

Note that no directional information is associated with dots. Also, for simplicity, we do not associate directions with loops at this stage. The direction of a line or a curve depends on the starting and ending points. Figure 1 shows some examples.

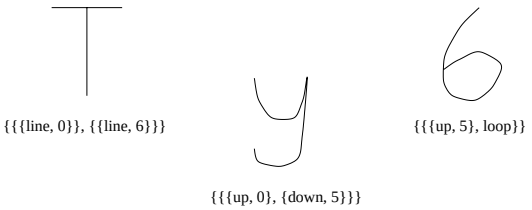


Figure 1. Examples of the representations for some characters

3. Structure extraction

In a character, there may be one or more strokes. Each stroke consists of a number of points that trace out a path on the writing surface from pen-down to pen-up in normal handwriting style. Every pair of consecutive points induces a direction. For points that follow the same direction or have only a slight turn, we will group them into one line segment. On the other hand, if there is a sharp turn along a stroke, we will represent it with multiple line segments. Figure 2 shows the steps taken to extract the structure of a digit "3".

In practice, some writers may produce characters which are hard to recognize. Such examples include zig-zag line segments, broken strokes, and writing strokes in reverse directions. Such problems can be fixed during the preprocessing stage or some later stages. Due to the paper size limit,

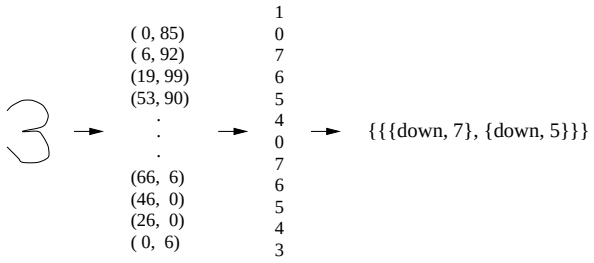


Figure 2. Steps taken to extract the structure

we will not explain here details of the algorithms used to tackle these problems. The details can be found in [4].

4. Elastic structural matching

After extracting the structure of a character, we can then match it against a set of models. However, due to different writing styles and habits, variations within the same character class are not uncommon. In order to increase the recognition rate, those characters that do not have an exact match will be slightly varied in shape and direction in an attempt to find approximate matches.

Most structural matching methods deal with graph representations directly [9]. Our method, instead, works on string representations. The following is our matching algorithm:

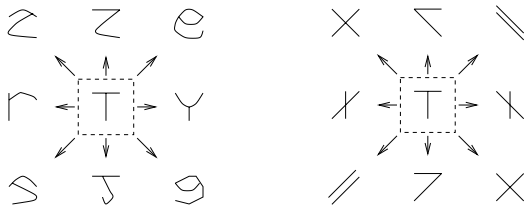
Algorithm *Elastic Structural Matching*

1. Load the set of models in Z.
2. Extract the structure of the test character C.
3. Initialize the deformation level L to be 1.
4. Let the candidate set $S = deform(L, C)$.
5. Let the match set $M = match(Z, S)$.
6. If M is not empty, return M. Otherwise, $L = L + 1$.
7. If L is less than or equal to the maximum deformation level, go to step (4). Otherwise, exit and report failure of finding an exact match.

Basically, there are four levels of structural deformation. Here we have to emphasize again that the search will stop once a match (or matches after deformation is performed) has been found. As an example, a regularly written "T" is likely to get correct classification during the first level of matching. However, in order to illustrate the generality of our scheme, we will use this simple character as an illustrative example anyway in our following discussions:

1. No deformation:
   The test pattern has to be exactly the same as one of the models.

2. Primitive type deformations:
   When there is no exact match, we will vary the primitive type in an attempt to find an approximate match. In so doing, line may become either one of its two neighboring types, i.e., up and down (since line is midway between up and down). However, up can only become line, but not down (since line is the only neighbor of up). Similar restrictions also apply to down. As a result, a "T" will have eight relaxed versions as shown in Figure 3 (a).



(a) Applying type deformation on "T"

(b) Applying directional deformation on "T"

Figure 3. Examples of structural deformations

3. Directional deformations:
   Similarly, we may also vary the direction. To do so, we find a neighboring code of the current one. For example, {line, 5} may become {line, 4} or {line, 6}. As a result, a "T" will have eight relaxed versions, though two of them are the same as shown in Figure 3 (b).

4. Simultaneous type and directional deformations:
   When no exact pattern can be found during the previous relaxation steps, we may consider finding the nearest match by deforming both the primitive type and direction together. As a result, a much larger number of patterns will be covered. For example, a "T" will have 80 possible deformed versions.

With elastic structural matching, some ambiguities may occur. Here we have a choice either to have all the ambiguous cases reported as answer, or to add some additional post-processing steps to discriminate among those possibilities and find the most probable one. For example, "D" and "P" often come together as the best matches found. In order to distinguish between them, we may consider the relative position of the vertical bar and the curve. Another example is the pair "n" and "r". Exactly where the curve ends now becomes crucial.

Note that many false-positive cases may be resulted if too much flexibility is allowed. Hence, we may need some additional steps to verify the answer if other domain-specific information is available.

5. Experimental results

In our experiment, we used an on-line handwriting data set collected by the MIT Spoken Language Systems Group [7]. It is a subset of the full set for isolated alphanumeric characters only. There are 62 character classes (10 digits, 26 uppercase and 26 lowercase letters) in our set. Each character class has 150 different entries written by 150 different people. Totally, there are 9300 characters. More than half of them are regularly written. The remaining ones are those either with noise in the data, poorly written, deliberately written in some strange and unusual way, or with zig-zag line segments. Figure 4 shows some examples of the characters in the data set.

(a) Regularly written characters

(b) Poorly written characters

(c) Characters which may be ambiguous or have noise in the data

Figure 4. Some examples of the characters in the data set

For all character classes, we have developed procedures to resolve the ambiguities. As a result, there are three possible outcomes in the recognition: *correct*, *incorrect*, and *rejected*. The following are the preliminary results:

	Correct	Incorrect	Rejected
Digits	$\frac{1479}{1500}$(98.60%)	$\frac{1}{1500}$(0.07%)	$\frac{20}{1500}$(1.33%)
Upper-case letters	$\frac{3841}{3900}$(98.49%)	$\frac{18}{3900}$(0.46%)	$\frac{41}{3900}$(1.05%)
Lower-case letters	$\frac{3800}{3900}$(97.44%)	$\frac{56}{3900}$(1.43%)	$\frac{44}{3900}$(1.13%)
All	$\frac{9058}{9300}$(97.40%)	$\frac{178}{9300}$(1.91%)	$\frac{64}{9300}$(0.69%)

Incorrect recognition is sometimes due to the ambiguous nature of the characters. Figure 5 shows some examples. Rejection, on the other hand, is often the result of unusual writing style, e.g., "4", "8", "A", and "j" in Figure 4 (c).

"1", recognized as "2" | "S", recognized as "J" | "U", recognized as "V" | "j", recognized as "i" | "n", recognized as "m"

Figure 5. Examples of some incorrectly recognized characters

If we do not include the rejected cases, the result is as follows:

	Reliability rate [1]
Digits	$\frac{1479}{1480}$ (99.93%)
Uppercase letters	$\frac{3841}{3859}$ (99.53%)
Lowercase letters	$\frac{3800}{3856}$ (98.55%)
All	$\frac{9058}{9236}$ (98.07%)

Note that rejection can be avoided by adding that example as a new model of its class. Caution should be taken though, as the number of models will increase and some models may be so specific that they are only responsible for very few (mostly just one) examples. Here, we also show the numbers of models used in the different character sets.

	Number of character models		
	Minimum	Maximum	Average
Digits	2	10	5.40
Uppercase letters	1	14	4.96
Lowercase letters	1	10	4.27
All	1	14	3.90

6. Conclusion

Nowadays, relatively few researchers use the structural approach for character recognition. Our experiment shows that by making use of structural information contained in a character together with a simple, flexible matching mechanism and some additional procedures for fine-grain classification, we can indeed achieve fairly good recognition results. On the average, the speed of recognition is about 7.5 characters per second running in Prolog on a Sun SPARC 10 Unix workstation and the memory requirement is reasonably low.

There are some more advantages with our approach:

1. Since our approach is a model-based one, all the patterns have semantically clear representations that can be used for subsequent manual verification.
2. New models may be added any time, though some effort has to be put on resolving conflicts between the new models and some existing ones.

However, at this stage, model creation is not automatic yet. In other words, we still have to manually design the set of models in advance. Fortunately, automatic extraction of models from the data is feasible in our scheme and will be one of our future directions to pursue.

In summary, with this simple and robust structural approach, we already have an effective and efficient on-line character recognition module. Such module can be used as part of a larger system, such as a pen-based mathematical equation editor which is one project currently being investigated by the authors.

[1] The *reliability rate* refers to the percentage of *correct* classification when rejected cases are excluded in the calculation.

References

[1] M. Berthod and J. P. Maroy. Learning in syntactic recognition of symbols drawn on a graphic tablet. *Computer Graphics and Image Processing*, 9:166–182, 1979.

[2] H. Bunke. Hybrid pattern recognition methods. In Bunke and Sanfeliu [3], chapter 11, pages 307–347.

[3] H. Bunke and A. Sanfeliu, editors. *Syntactic and Structural Pattern Recognition - Theory and Applications*. World Scientific, Singapore, 1990.

[4] K. F. Chan and D. Y. Yeung. Elastic structural matching for recognizing on-line handwritten alphanumeric characters. Technical Report CS98-07, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, Mar. 1998.

[5] J. Feder. Plex languages. *Info. Sciences*, 3:225–241, 1971.

[6] H. Freeman. Computer processing of line drawing images. *ACM Computing Surveys*, 6(1):57–98, 1974.

[7] R. Kassel. *A Comparison of Approaches to On-line Handwritten Character Recognition*. PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, June 1995.

[8] S. Lucas, et al. A comparison of syntactic and statistical techniques for off-line OCR. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications (ICGI-94)*, pages 168–179. Springer-Verlag, Sept. 1994.

[9] T. Pavlidis. Structural descriptions and graph grammars. In *Pictorial Information Systems*. Springer-Verlag, 1980.

[10] A. Rosenfeld. Array, tree and graph grammars. In Bunke and Sanfeliu [3], chapter 4, pages 85–115.

[11] A. C. Shaw. A formal picture description scheme as a basis for picture processing systems. *Information and Control*, 14:9–52, 1969.

[12] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[13] P. S. P. Wang and A. Gupta. An improved structural approach for automated recognition of handprinted characters. In P. S. P. Wang, editor, *Character and Handwriting Recognition: Expanding Frontiers*, pages 97–121, Singapore, 1991. World Scientific.