RECOGNITION FOR LARGE SETS OF HANDWRITTEN MATHEMATICAL SYMBOLS

Stephen M. Watt and Xiaofang Xie

Dept. of Computer Science
University of Western Ontario
London Ontario, Canada N6A 5B7
{watt,maggie}@csd.uwo.ca

Abstract

Natural and convenient mathematical handwriting recognition requires recognizers for large sets of handwritten symbols. This paper presents a recognition system for such handwritten mathematical symbols. We use a pre-classification strategy, in combination with elastic matching, to improve recognition speed. Elastic matching is a model-based method that involves computation proportional to the set of candidate models. To solve this problem, we prune prototypes by examining character features. To this end, we have defined and analyzed different features. By applying these features into an elastic recognition system, the recognition speed is improved while maintain high recognition accuracy.

1. Introduction

The development of Personal Digital Assistants (PDAs) and Tablet PCs provides an ideal potential environment to input mathematics by handwriting. Handwriting input would not only make writing mathematical documents much easier, it would also provide a friendly user interface for computer algebra systems such as Maple. Although the first applications on PDAs and Tablet PCs have no concept of mathematical input, the handwriting writing modality is so much more natural for mathematics than typing that we expect mathematical handwriting recognition to be important.

Mathematical handwriting differs significantly from the other forms of handwriting. First, the set of possible input symbols is very extensive, coming from several different alphabets and sources. A full recognizer for all mathematical handwriting would have to distinguish about 2000 symbols. Unlike Asian languages with large symbol sets, these symbols are typically composed of a few strokes, with no specific stroke order, and with many symbols being quite similar. Second, the spatial relation among symbols can use complex context-sensitive two dimensional rules. Third, mathematical handwriting can involve large operators such as matrix brackets, fraction bars or square roots. This layout and grouping makes mathematical handwriting akin to a blend of drawing and writing.

From these difficulties, we can see that in order to recognize handwritten mathematics well, we need to perform two major tasks: recognize a large set of mathematical symbols and analyze the two-dimensional mathematical formula structure. Recognition within large symbol sets is understood to be a hard problem [1]. This paper presents our first steps toward a system to address this problem, so we can make progress in recognizing handwritten mathematical symbols.

It is understood that feature extraction is important in handwriting recognition [2] and most existing recognizers use features to some degree. For the problem of recognition in large symbol sets, we expect that feature recognition will be more important than in other applications. In this paper, we define some features, give algorithms for extracting these features, and analyze the performance of a recognizer after applying some of these features.

2. Architecture of the Symbol Recognizer

2.1. Context

This work is part of a larger project, involving expression analysis, dictionary-based methods, and portability layers [3]. In this paper we concentrate on the problem of symbol recognition. Figure 1 represents the top-level architecture of our symbol recognition system. The overall organization is as follows: First we collected a representative set of stroke data as traces of (x, y) points, rather than as images. We applied various preprocessing methods to the data, such as smoothing, re-sampling, size normalization, *etc*. This data was used off-line to develop classification categories for the different mathematical symbols. When we wish to recognize a symbol on-line, given raw stroke data, we perform similar preprocessing, and then send the strokes to a feature extraction coordinator. This detects which features the symbol exhibits. These are used to select the class mathematical symbols known to exhibit these features. Then elastic matching is performed on the pruned set. We give details of each step in the following sections.

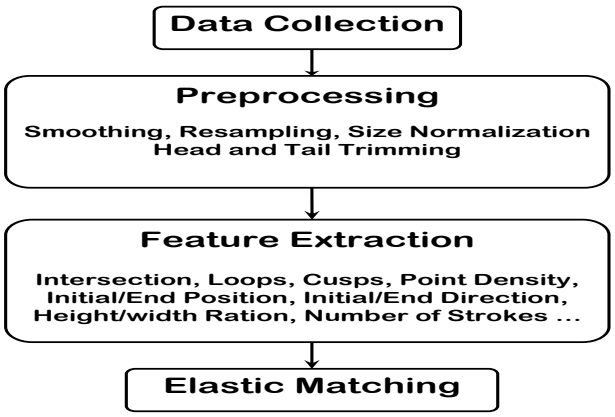

Fig. 1. Symbol recognizer global architecture

2.2. Data Collection

We collected data through using a questionnaire filled out a tablet PC. The questionnaire included 227 symbols and a number of formulas. Individual symbols included ten digits, lower case and upper case letters, Greek letters, mathematical operators and different arrows. For each symbol the data collected included x, y coordinates, pen up/down events, pen pressure and time stamps. When the user lifted the pen, as long as the pen could still be detected, the trace coordinates were also recorded. Each movement of the pen was recorded as a "Stroke", either with or without the pen touching the writing surface. Pressure is associated with each pair of x, y coordinates, and varies between 0 and 255, in device-dependent units. The time stamp includes the start time and the ending time of each stroke. The timing of the intermediate points can be determined by the sampling frequency of the device.

2.3. Preprocessing

We performed several pre-processing operations, including selectively chopping the head and the tail of strokes, re-sampling, smoothing and size normalization. The smoothing operation computes the average of each three consecutive points, and replaces the mid point by the average. The first and last few points usually exhibit errors, and usually they do affect the recognition results. We therefore chop the head and the tail of each stroke. In chopping the head, we calculate the angles among the first 5 points. If the angles change a lot (our experiments showed 90 degrees to be effective), then we chop the corresponding points. Same rules apply to angle chopping the tail. We used re-sampling to reduce the amount of data, and performed size normalization to remove the symbol size effect on recognition. Resampling also gives us a degree of device independence.

2.4. Feature Extraction

Feature extraction has been recognized as important in many recognizers, but selecting the relevant features is still quite an ad-hoc activity, with researchers still trying to find the best features. H. Winkler used angle and angle difference in HMM-based recognition [4]. He also integrated hidden stroke information with the actual strokes. Except for the angle feature, he used the information of whether the actual point belongs to a stroke or to an integrated hidden stroke. Recently, hidden/actual stroke information is called pen-down/pen-up feature in the literature. M. Okamoto and K. Yamamoto used clockwise/counterclockwise direction-change features, written-area features and circle features [5] in Japanese character recognition. K. Chan and D. Yeung [6] extracted lines, counter-clockwise/clockwise curves, loops and dots in their elastic matching recognizer.

J. Kurtzberg [7] did work similar to our own, but used different features for a smaller symbol set. He used the following seven features to improve the speed of his elastic matching recognizer: the number of strokes, the number of points per stroke, the number of points in the symbol, the height of the lowest point, the height of the highest point, the height of the lowest point per stroke and the height of the highest point per stroke. Some of these features, for example the number of points per stroke, are device dependent. Others features were special to certain data-collection conditions. For example, Kurtzberg used guidelines, so the height features are relative to the guidelines.

We considered the different features presented in the literature and concentrated on the device and context independent features. From these features, we selected a set for pre-classifying in an elastic matching based recognizer. We describe these in more detail below.

These features have elsewhere been tested on small set of symbols. Here we test our features on the larger set of mathematical symbols. We also give improved algorithms for some of these features. For example, K. Chan and D. Yeung [6] used loops, but as they pointed out, their loop finding algorithm can not find all loops. We note that our algorithm improves on this.

We organize these features into different categories. The definitions and algorithms are given in the following sections. The results of recognition using selected features are shown in the experimental results section.

3. Feature Families

When people recognize handwritten symbols, geometric and other features, such as loops, play an important role. Moreover, when the characters are too cursive to recognize individually, people rely on context, loops and intersections to distinguish characters. We outline some of these features here.

3.1. Geometric Features

Number of Loops: This includes complete loops and open loops. In order to find loops, we define the "minimum distance pair".

Minimum Distance Pair: A pair of points which has the minimum non-local distance in a given area. To ensure non-locality (e.g. sequential points in a trace), the time interval between the pair of points must exceed a certain threshold. There is only one minimum distance pair in a given neighborhood. Approximate loops may be found using minimum distance pairs.

Our algorithm starts with a pair of points within a certain distance threshold on a stroke. We call these points the "begin" and "end" point and compute the straight line between them. We then sweep a line parallel to this in the direction that makes shorter the distance between the two neighboring intersections with the stroke. See figure 2. We continue until the pair of intersections with the parallel line are seen to be locally connected, or until we find a minimum distance pair.

As K.Chan and D.Yeung pointed out, the detection of loops is not always trivial [6]. They used chain code sequences to detect loops, but where the stroke starts may affect the results. Our algorithm does not have this limitation. We detect the loop in figure 3 which is their failure case.

Number of Intersections: For this we count self intersections and inter-stroke intersections. We implemented the modified Bently-Ottman sweepline algorithm. Every time we find an intersection, we delete the two line segments associated with the intersection.

Number of Cusps: A cusp is a sharp turning point in a stroke. A cusp is formed locally by three points, *e.g.* p1, p2, p3 in figure 4. If the angle of p1, p2 and p3 is small, it could be a cusp. In order to determine well-defined cusps, we check two more neighboring points: p0 and p4. The points p0, p1 and p2 should be on a relatively straight line. Likewise for p2, p3, p4. We take 170 degrees as our straightness threshold.

3.2. Ink Related Features

Number of Strokes: This information is manifest in the data structures returned by the ink collection.

Point Density: We determine whether the ink density is most similar to the letter "o," "p" or "b." For o density, ink is evenly distributed in the whole stroke. With p density, ink is distributed in the upper part is more than that in the lower part, while for b density, ink is distributed in the lower part more than that in the upper part. To compute these, we divide the ink bounding box into three parts vertically, the upper 40%, the middle 20% and the lower box is 40%. We divide the ink bounding box into three boxes instead of two due to the variance in handwriting. For example, the lower part of letter "b" may occupy more than 50% of the symbol height.

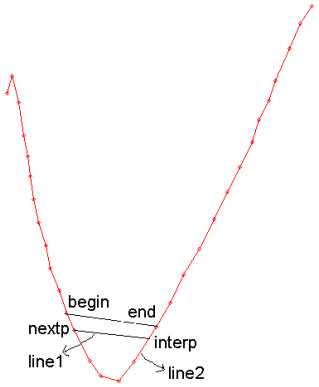

Fig. 2. Example 1 for Loop Detection

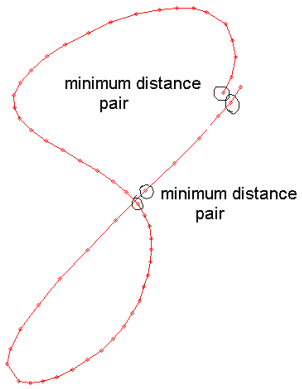

Fig. 3. Example 2 for Loop Detection

3.3. Directional Features

Initial, End and Initial-End Directions: The initial and end directions are calculated to the third neighboring point. The initial-end direction is the direction from the initial point to the end point. We discretized all directions to the nearest of N, NE, E, SE, S, SW, W, NW.

3.4. Global Features

Initial and End Position: We divide the ink bounding box into four quadrants, NE, NW, SE, SW. We take as a feature into which of these boxes the initial and end points fall.

Width to Height Ratio: We discretized this ratio to three values, 0 for a slim symbol, 1 for a wide symbol, and 2 for a normal symbol.

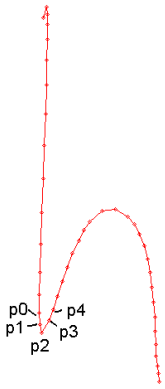

Fig. 4. Cusps

4. Recognition

We use elastic matching against models for the final recognition step. This is achieved by calculating the minimum distance between the unknown symbol and a set of models. The mapping between the two sequences of points allows for both one-to-one and many-to-one correspondences between points using dynamic programming. The total distance between the ith point of the unknown character and the jth point of the model, $D(i,j)$, is given calculated as:

$$D(i,j) = \delta(i,j) + \begin{cases} \sum_{k=0}^{j-1} \delta(0,k) & \text{if } i = 0 \\ \sum_{k=0}^{i-1} \delta(k,0) & \text{if } j = 0 \\ \min \begin{cases} D(i-1,j) \\ D(i-1,j-1) \end{cases} & \text{if } i > 0, j = 1 \\ \min \begin{cases} D(i-1,j-1) \\ D(i-1,j-2) \end{cases} & \text{if } i > 0, j > 1 \end{cases}$$

$$\delta(i,j) = (x_i - x_j)^2 + (y_i - y_j)^2 + C\,|\phi_i - \phi_j|$$

where ϕ represents the orientation and the curvature.

5. Experimental Results

Our initial objective was to use features to reduce computation in our symbol recognition system. We have therefore based our selection of features, not just on their effectiveness, but also on their computational cost. The features we use in our experiments were: number of strokes, initial position, width to height ratio, end direction and initial to end direction. We required the number of strokes and the (discretized) width to height ratio to match exactly. The initial position and initial to end direction could differ by one. We allowed end direction to differ by two.

Our test data set had 227 symbols, including digits, Latin letters, some Greek letters and mathematical operators. We asked our users to write eight sets of the symbols. The first four sets were used as prototype sets, and the rest sets were used as testing sets. The results are shown in Table 1, Table 2 and Table 3. We use P as notation for prototype sets. P1 indicates the data from the first prototype set. We use T as notation for test sets. T1 indicates the data from the first testing set.

Experiment	# Prototypes	Recog.
P1:T1,2,3,4	227	81.8 %
P1,2:T1,2,3,4	454	90.1
P1,2,3:T1,2,3,4	681	93.9
P1,2,3,4:T1,2,3,4	908	94.8

Table 1. Results Without Features

Experiment	Prototypes			Recog.
	#	Candidate	Pruned	
P1:T1,2,3,4	227	26	88.5 %	76.0 %
P1,2:T1,2,3,4	366	38	89.6	85.5
P1,2,3:T1,2,3,4	495	52	89.5	90.0
P1,2,3,4:T1,2,3,4	575	60	89.6	91.9

Table 2. Results With Features

Table 1 shows the experimental results without using features. The first column indicates the prototype sets and testing sets. The second column is the number of prototypes used for recognizing a symbol. For example, if the prototype set is P1, for each symbol in T1, we need to do 227 symbol comparisons. When another prototype set is added, the number of prototypes is 227×2 for each symbol in the test sets.

Table 2 shows the experimental results using features. The first column is same as that in table 1. The second column is the number of prototypes in the prototype sets. The prototypes are updated by training on the second set, and so on. The third column is the number of prototypes used for recognition. For example, if the prototype set is P1, for a symbol in T1, we need to do 26 comparisons. The fourth column is the percentage of prototypes pruned.

From Table 1 and Table 2, we see a final recognition rate is 94.8% without using any features. With features, the recognition rate is 91.9%, but the prototypes are pruned by 89.6%, reducing computation proportionately. Note that the percentage of prototypes pruned was relatively constant for each prototype set.

Table 3 is the test results on 72 symbols, including 10 digits, 52 upper and lower case Latin letters and 10 punctuation symbols. This is the same set as in J.Kurtzberg's paper [7]. We acknowledge that it is difficult to compare different recognition systems when the test data sets are not exactly same. if there are large differences in the results, however, the comparison can still be useful.

Experiment	# Prototypes		Candidate Prototypes		Pruned (%)		Recog. Rate (%)	
	J.K's	ours	J.K's	ours	J.K's	ours	J.K's	ours
P1,2,3,4: T1,2,3,4	121	169	47	24	61.5	85.8	99.0	97.6
P1,2,3,4: T1,2,3,4	122	288	92	288	N/A	N/A	99.0	99.7

Table 3. Our vs. J.Kurtzberg's Results

In Table 3, we can see the fraction of prototypes pruned has improved significantly over Kurtzberg's result, from 61.5% to 85.8%. The recognition rate has reduced by about 1%, but still remains high. As the features we used to prune prototypes were easy to compute, the computation was insignificant compared to the elastic matching. Compared with J.Kurtzberg's features, ours are more device independent and somewhat more general. The last row in Table 3 shows the experimental results without using features. Since J.Kurtzberg used some parameters, the number of prototypes is reduced. We note our recognition rate is better, but that the data sets were not the same.

Although we have not used all the features we have studied (such as loops, intersections, and so on) for pruning prototypes, they are still useful for other purposes. For example, we can improve the elastic matching procedure by applying these features. They can also be used in other recognition methods.

To compare our results with more recent work, we implemented some of the features in Henning's paper on word recognition [8]. The features we implemented were ascender, descender, length of character, and number of center horizontal line crossings. Section 3 of Henning's paper discussed dictionary reduction, which is similar to our prototype reduction. To the best of our knowledge, there is no similar recent work for on-line handwritten symbol recognition. We have applied these features of Henning at the character level to see the results. We have compared these features in terms of coverage. We use the coverage notion of Koerich [9], where coverage refers to the capacity of the reduced (pruned) lexicon to include the correct match. We tested our features and Henning's features (applied to symbols instead of words) on 227 symbols of 2 different writers. All the features, except for length, require exact matches, and length must match within a certain tolerance. Our experiments showed that both our features and Henning's features give 100% coverage.

6. Conclusions

Most recognizers focus on limited range of symbols such as postal codes or Latin letters. Even recognizers for Asian languages deal with a limited vocabulary of strokes which must be given in a particular order. Recognition for large set of symbols is still a challenging research problem.

We have made progress in this area by using feature sets to prune the number of candidates considered in a character match. This is a central feature of our symbol recognition framework, and influences our overall mathematical handwriting project.

Our recognizer can recognize digits, English letters, Greek letters, most of the common mathematical operators and notations. We have selected our features based on their effectiveness and computational cost. We have attempted to identify these features empirically, having analyzed a database of 10,000 mathematical handwriting samples. We have found the use of features in pruning to be effective, greatly reducing computation time at only a modest decrease in recognition rate.

7. References

[1] D.Blostein and A.Grabvec, "Recognition of mathematical notation," in *Handbook of Character Recognition and Document Image Analysis*, H.Bunke and P.S.P Wang, Eds. 1996, pp. 557–582, World Scientific, Singapore.

[2] O.D.Trier, A.K.Jain, and T.Taxt, "Feature extraction methods for character recognition–a survey," *Pattern Recogniton*, vol. 29, no. 4, pp. 641–662, 1996.

[3] Elena Smirnova and Stephen Watt, "A context for pen-based mathematical computing," in *Proceedings of the 2005 Maple Summer Conference*, July 2005.

[4] H-J. Winkler, "Hmm-based handwritten symbol recognition using on-line and off-line features," in *International Conference on Acoustics Speech and Signal Processing*, May 1996, pp. 3438–3441.

[5] M.Okamoto and K.Yamamoto, "On-line handwritten character recognition method using directional features and clockwise/counterclockwise direction-change features," in *International Conference on Document Analysis and Recognition*, September 1999, pp. 491–494.

[6] K.F.Chan and D.Y.Yeung, "Elastic structural matching for online handwritten alphanumeric character recognition," Tech. Rep., Department of Computer Science, Hong Kong University of Science and Technology, March 1998.

[7] Jerome M. Kurtzberg, "Feature analysis for symbol recognition by elastic matching," Tech. Rep., IBM Research Center, January 1987.

[8] A. Henning and N. Sherkat, "Cursive script recognition using wildcards and multiple experts," *Pattern Analysis and Applications*, vol. 4, no. 1, pp. 51–60, 2001.

[9] A.L. Koerich, R. Sabourin, and C.Y. Suen, "Large vocabulary off-line handwriting recognition: A survey," *Pattern Analysis and Applications*, vol. 6, no. 2, pp. 97–121, 2003.