Prototype Pruning by Feature Extraction for Handwritten Mathematical Symbol Recognition

Stephen M. Watt Xiaofang Xie

Department of Computer Science
University of Western Ontario
London Ontario, CANADA N6A 5B7

{watt,maggie}@csd.uwo.ca

Abstract

Successful mathematical handwriting recognition will require recognizers for large sets of handwritten symbols. This paper presents a recognition system for such handwritten mathematical symbols. The recognizer can provide a component of a handwritten interface for computer algebra systems such as Maple. Large sets of similar symbols present new challenges in the area of handwriting recognition, and we address these here. 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 maintaining high recognition accuracy.

1 Introduction

The development of Personal Digital Assistants (PDAs) and the Tablet PC provides an ideal potential environment to input handwritten mathematics. 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. Therefore handwritten mathematical symbol recognition has recently attracted increasing research interest [1]. 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.

A number of projects have made recent progress in mathematical handwriting recognition. Maple 10 has a built-in handwritten symbol selector. It allows users to input math symbols using mouse or stylus, then invoke the recognizer, and choose an answer from a list of candidates. Figure 1 is an interface for Maple handwriting input [2]. FFES is a handwriting-based equation editor [3]. Figure 2(a) is a screen shot of FFES [4], showing how the user can write math expressions in a window. The Infty editor is another mathematical expression

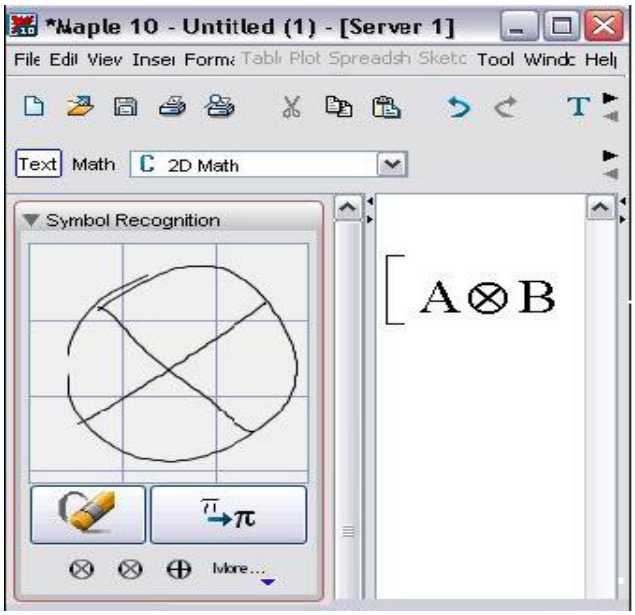


Figure 1: Maple 10 Handwriting Input

editor with on-line handwritten mathematical expression recognition. Figure 2(b) is the Infty handwriting input pad [5].

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 mathematical handwriting would have to distinguish between 1000 and 2000 symbols. Unlike Asian languages with large symbol sets, these symbols are typically composed of few strokes, with no specific stroke order. 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 make mathematical handwriting akin to a blend of drawing and writing.

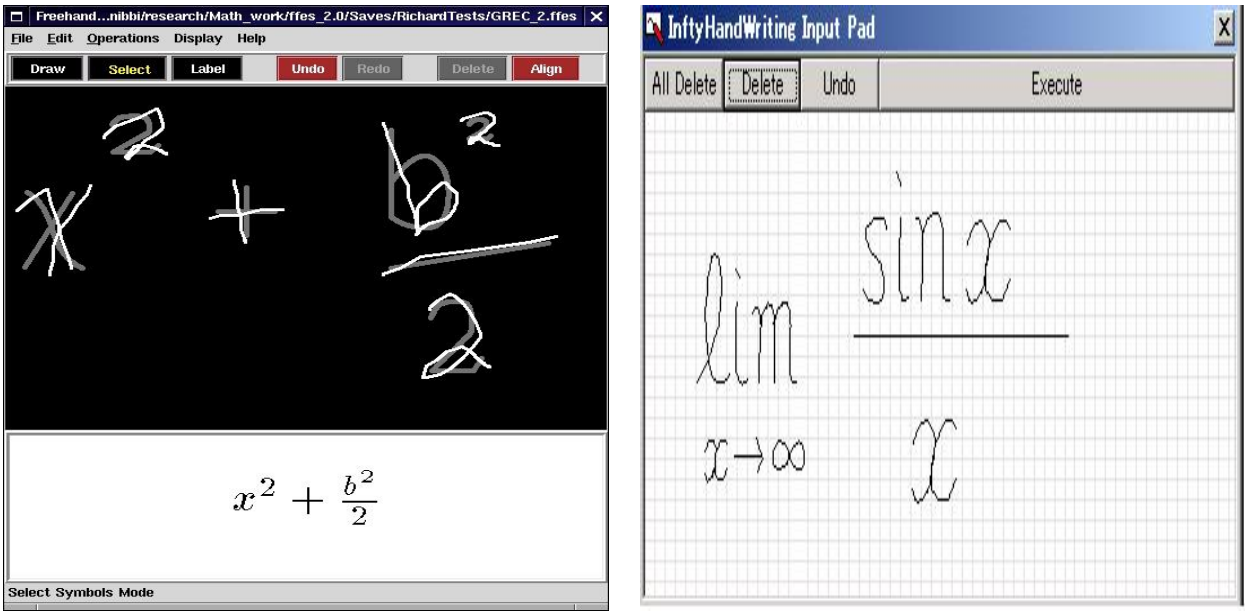
From these difficulties, we can see that in order to recognize handwritten mathematics precisely, we need to perform two major tasks: recognize a large set of mathematical symbols and analyze the two-dimensional mathematical formula structure. Recognition within large symbol sets is understood to be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

Pruning prototypes is an intuitive way to break a large vocabulary into several smaller groups. Given an unknown symbol, we find the small group to which it belongs, then try to find the symbol in the group. The computation for finding the small group should not be too expensive, otherwise we gain nothing on recognition performance. If we know something in advance about the possibilities for a symbol (e.g. that it is a Greek letter), we can narrow the search. Unfortunately, in most cases, we don't have this information. Pre-classifying large sets of symbols is another approach to prune prototypes. In this paper, we pre-classify our symbol set according to certain features.

It is well understood that feature extraction is important in handwriting recognition [6],

and most existing recognizers use features to some degree. There is, however, not a lot of ongoing work on feature analysis. There is as yet an agreed upon set of sufficient features that can be applied in different recognizers.

In this paper, we define some features (based on analysis of empirical data), give algorithms for extracting these features, use them for pruning prototypes, and also analyze the performance of the recognizer after applying them.



(a) **(b)**

Figure 2: (a) FFES Interface (b) Infty Handwriting Input

2 Architecture of the Symbol Recognizer

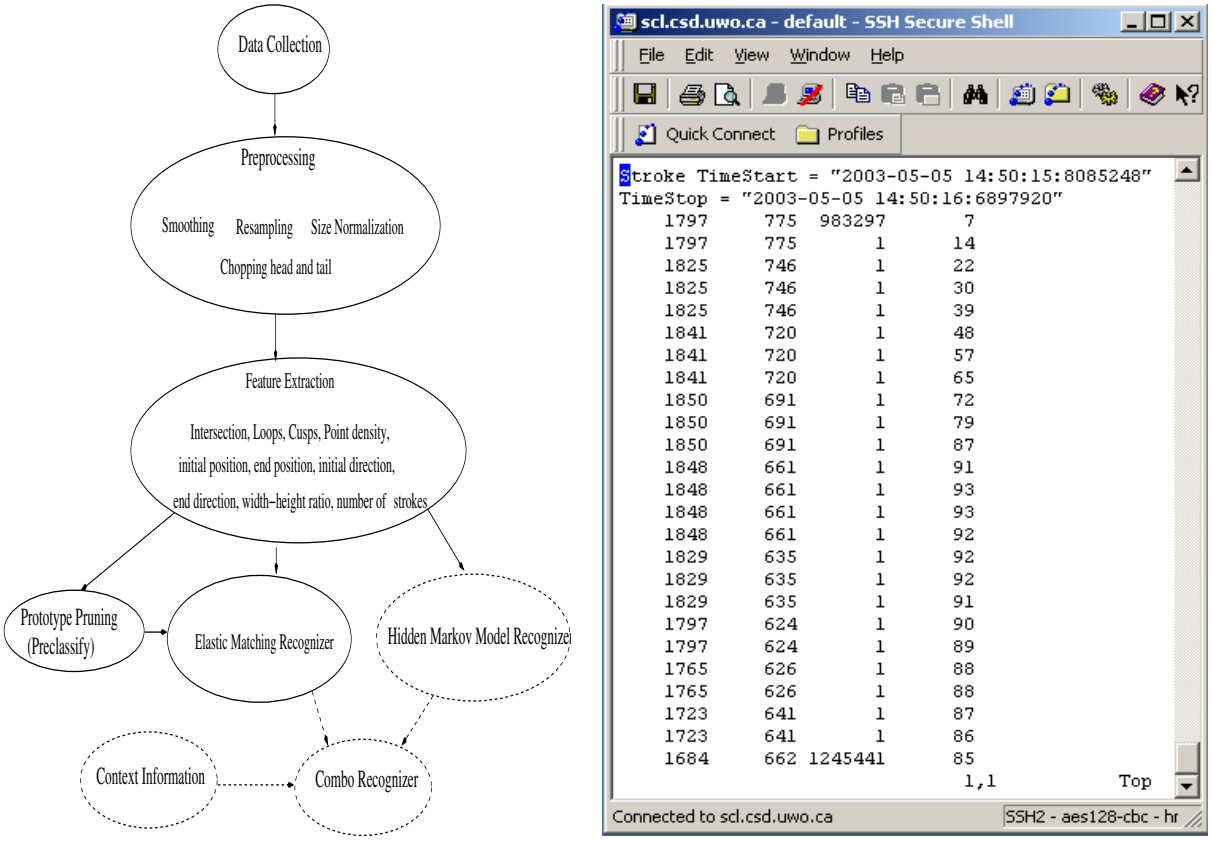
2.1 Context

This work is part of a larger project, involving expression analysis, dictionary-based methods, and portability layers [2]. In this paper we concentrate on the problem of symbol recognition. Figure 3(a) represents the top-level architecture of our symbol recognition system. The overall organization is as follows: First we collect a representative set of stroke data as traces of (x, y) points, rather than images. Then we apply various preprocessing operations to the data, such as smoothing, re-sampling, size normalization, *etc.* Next the processed data go to the feature extraction step. The extracted features are used either by recognizers or by prototype pruning. During prototype pruning, we group the mathematical symbols according to some features. The features for the prototypes are computed once and stored.

For an unknown symbol, we perform similar pre-processing, and then send the strokes to a feature extraction coordinator. The extracted features are compared with the features of the prototypes. This detects the to group which the unknown symbol belongs. Comparison during recognition is performed on groups instead of the whole set of symbols. Elastic

matching is performed on the pruned set. The computational cost is therefore reduced, improving overall recognizer speed.

We give details of each step in the following sections. We also combine elastic matching recognition with a hidden Markov model recognizer, applying context information to the combined recognizer. However in this paper we won't discuss the modules in the dash line ellipses of Figure 3(a).



(a) (b)

Figure 3: (a) Architecture of symbol recognizer (b) Data Sample

2.2 Data Collection

We collected data using a questionnaire filled out a tablet PC. The questionnaire includes 227 symbols and a number of formulas. Individual symbols include ten digits, lower case and upper case letters, Greek letters, mathematical operators and different arrows. For each symbol, the data collected included x, y coordinate traces, 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. This means we collected both visible and invisible stroke information. Each movement of the pen is a "Stroke", either with or without the pen touching the writing surface.

Pressure is associated with each pair of x, y coordinates, varies between 0 and 255, in device-dependent unit. 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.

The current database includes about 50 writers' handwriting. The writers are from difference academic backgrounds and different nationalities. For this paper, we created additional data sets for comparison purposes. In particular, we used a single user to form a dataset directly comparable to Kurtzberg's work. To compare with Henning's work [7], we use data from two different users.

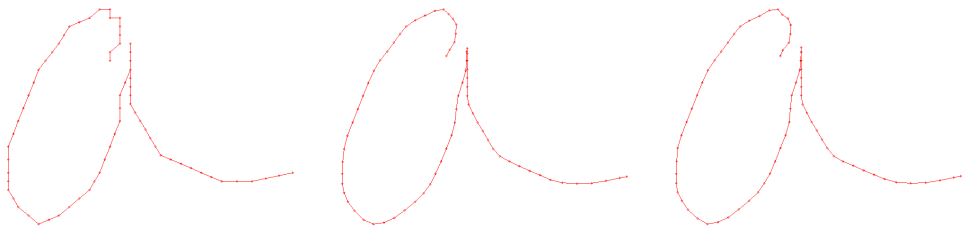
2.3 Preprocessing

We performed several pre-processing operations, including selectively chopping the head and the tail of strokes, re-sampling, smoothing and size normalization.

The first and last few points usually exhibit errors, and usually these affect 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 degree to be effective), then we chop the corresponding points. Same rules apply to chopping the tail.

The sampling rate of most devices used for handwriting is over 100 points/sec. The high sampling rate gives us precise information about the shape of the symbol. On the other hand, it also increases computation time. The re-sampling operation makes the points evenly distributed. In re-sampling we remove points which are too close, and insert some points if the distance between two points is too great. In the end, the number of points is reduced. We also performed size normalization to remove the effect of symbol size on recognition.

The smoothing operation is used to remove noise caused by shaking of hand or uneven surface of the device. We implemented two types of smoothing operations: average smoothing and Gaussian smoothing. The average smoothing is used for this paper. Average smoothing computes the average of every three consecutive points, then replaces the mid point by the average. Gaussian smoothing uses a discrete approximation of a Gaussian distribution as in the following formula. A point is replaced by its Gaussian distribution. The three coefficients we used are $\alpha_{-1} = 1/4, \alpha_0 = 1/2, \alpha_1 = 1/4$. Figure 4 shows examples of the smoothing operations.



(a) **(b)** **(c)**

Figure 4: (a) before smoothing, (b) after average smoothing, (c) after Gaussian smoothing

2.4 Feature Extraction

In handwriting recognition, especially for large sets of symbols, extracting the right features is not easy due to the variance of handwriting. To find the proper features, we need to study the variance of handwriting. Figure 5 gives some examples of handwriting from our database. We can see that the number of strokes, the connection of strokes and other factors have a high degree of variance.

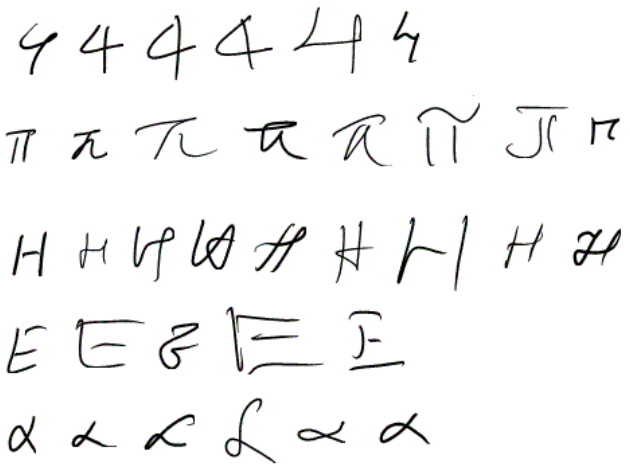


Figure 5: Variance in Handwriting for $4, \pi, H, E, \alpha$

Images are the data format of off-line handwriting. In on-line handwriting, data is represented by a sequence of points. We need to extract features from images [8] or a sequence of points [9] for off-line and on-line handwriting recognition respectively. The extracted features can be used for recognition and pre-classification. In fact, features are often used to represent handwritten characters and may thus be though of as the abstract characteristics of a character.

Although 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. We give some examples: G.Rigoll and A.Kosmala used the angle between two consecutive points and a coarse character bitmap as major features [10]. H.Winkler used angle and angle difference in HMM-based recognition [11]. He also integrated hidden stroke with the actual stroke. Except for the angle feature, he used the information of whether the actual point belongs to a stroke or not to an integrated hidden stroke. Recently, hidden/actual stroke information is called a 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 [12]. They used these features for Japanese character recognition. K.Chan and D.Yeung [13]extracted lines, counter-clockwise/clockwise curves, loops and dots in their elastic matching recognizer.

J.Kurtzberg [14] 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. Other features were special to certain data-collection conditions. For example, J.Kurtzberg used guidelines, the height features were relative to the guidelines.

We studied different features in the literature and gave preference to device and context independent features. From these features, we select 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 large set of mathematical symbols. We also give improved algorithms for some of these features. For example, K.Chan and D.Yeung [13] used loops, but as they pointed out, their finding loop algorithm could not find all loops. We note that our algorithm improves on this.

We organize these features into different categories. The definition and algorithm 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 closed 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 7(a). 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 that detection of loop is not always trivial [13]. They used chain code sequence to detect loops, but where the stroke starts may affect the results. Our algorithm does not have this limitation. We can detect the loop in figure 7(b) which is their failure case.

Number of Intersections: : including self intersections and intra-stroke intersections. We implemented the modified Bently-Ottman[15] sweepline algorithm [16] [17]. Every time we find an intersection, we delete the two line segments associated with the intersection. Then we insert two lines into the set of line segments. These two lines begin from the intersection point, and end with their old ending points.

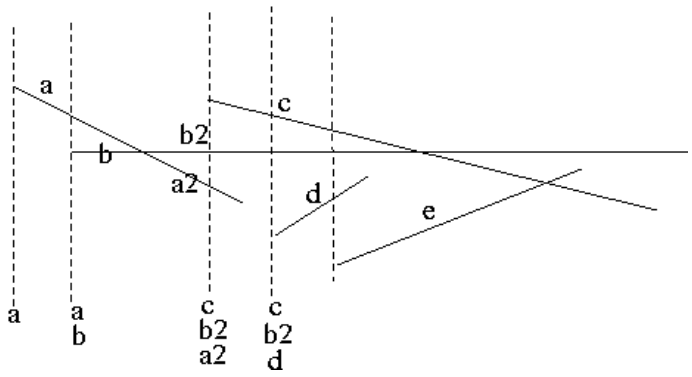


Figure 6: Modified Sweepline algorithm

The line segments are ordered. In figure 6, we say $a > b$ at the beginning point of line segment a, if the intersection of a with the sweep line is higher than the intersection of b with the sweep line.

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 8(a). 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.

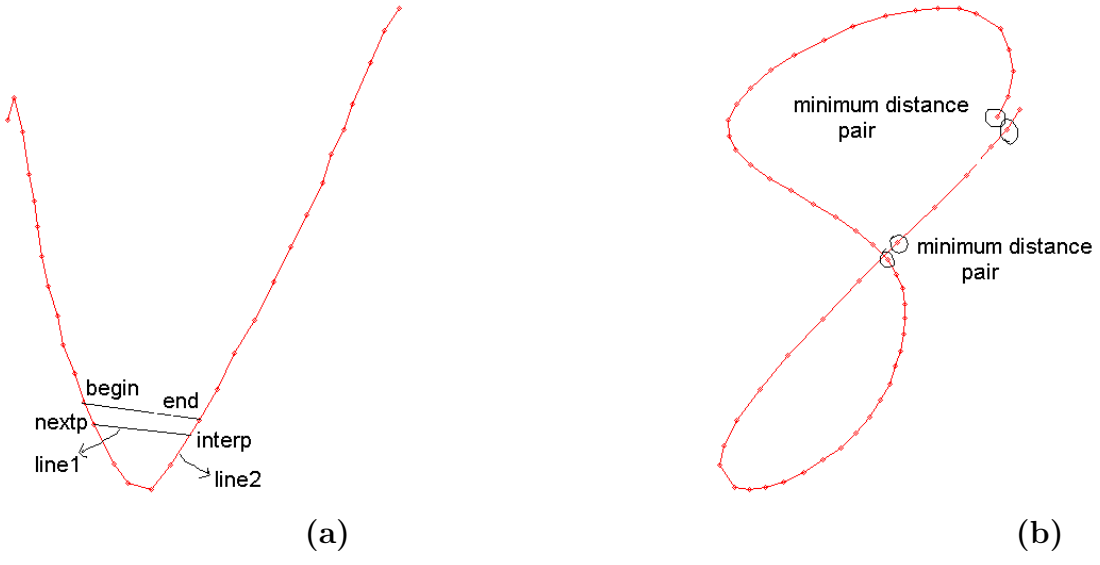
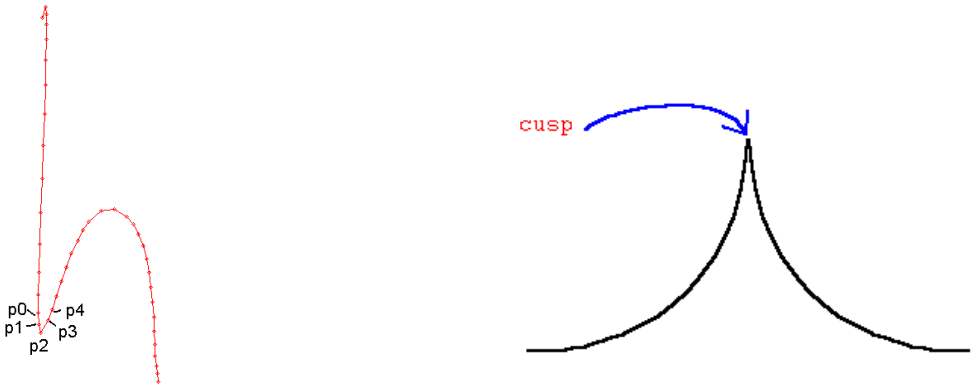


Figure 7: (a) Example 1 for Loop Detection (b) Example 2 for Loop Detection



(a) **(b)**

Figure 8: (a) Cusps in "h" (b) Cusps

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.

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.

4 Recognition Method

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. Figure 9

shows an example of elastic matching. The mapping between the two sequences of points allows for both one-to-one and many-to-one correspondences between points using dynamic programming [18]. 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.

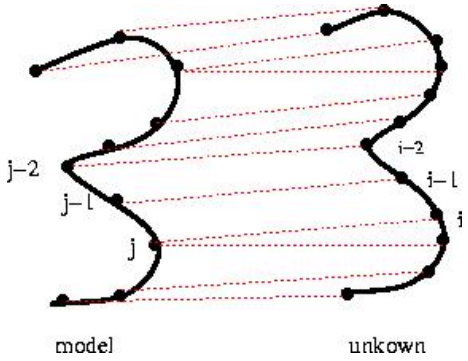


Figure 9: Elastic Matching

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.

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 [14]. 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.

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 [7]. 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 [19], 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.

Experiment	# Prototypes	Recog. Rate(%)
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	Candidate Prototypes	Percentage Pruned	Recog. Rate(%)
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

Experiment	# Prototypes		Candidate Prototypes		Percentage 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

6 Conclusions and Future Work

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.

Ambiguity is one of the factors which contribute to the error rate. Sometimes we reply on context information to get the right result. Figure 10 is the examples of ambiguities which need context information to solve. In future, we will combine the elastic matching recognizer with a HMM recognizer, and apply context information to improve recognition performance.

Figure 10: Context Information

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

[3] Lucy Zhang and Richard Fateman, "Survey of user input models for mathematical recognition: Keyboards, mice, tablets,voice," Tech. Rep., University of California, 2003.

[4] R. Zanibbi, K. Novins, J. Arvo, and K.Zanibbi, "Aiding maniplulation of handwritten mathematical expressions through style-preserving morphs," in *Proceedings of Graphics Interface*, 2001, pp. 127–134.

[5] Mitsushi Fujimoto, Toshihiro Kanahori, and Masakazu Suzuki, "Infty editor - a mathematics typesetting tool with a handwriting interface and a graphical front-end to openxm servers," in *3rd International Conference on Mathematical Knowledge Management*, 2004.

[6] Ø.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.

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

[8] X.Y.Liu and M.Blumenstein, "Experimental analysis of the modified direction feature for cursive character recognition," in *International Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 353–358.

[9] J.Hu, S.Lim, and M.Brown, "Hmm based writer independent on-line handwritten character and word recognition," in *International Workshop on Frontiers in Handwriting Recognition*, May 1998, pp. 143–155.

[10] G.Rigoll and A.Kosmala, "A systematic comparison between on-line and off-line methods for signature verification with hidden markov models," in *International Conference on Pattern Recognition*, August 1998, pp. 1755–1757.

[11] 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.

[12] 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*, Sep 1999, pp. 491–494.

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

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

[15] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press,England, 1989.

[16] Wolfgang Freiseisen and Petru Pau, "A generic plane-sweep for intersecting line segments," *RISC-Linz Report*, pp. Series No. 98–18, Nov 1998.

[17] Bernard Chazelle and Herbert Edelsbrunner, "An optimal algorithm for intersecting line segments in the plane," *ACM*, vol. 39, no. 1, pp. 1–54, Jan 1992.

[18] Bo Wan, "An interactive mathematical handwriting recognizer for the Pocket PC," M.S. thesis, Department of Computer Science, the University of Western Ontario, 2002.

[19] 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.