RECOGNITION OF HANDWRITTEN NUMERALS USING ELASTIC MATCHING

Patrice Scattolin

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

November 1995

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Patrice Scattolin**

Entitled: **Recognition of Handwritten Numerals Using Elastic Matching**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

______________________________ Chair

____

____

____

____

______________________________ Supervisor

Approved ______________________________
Chair of Department or Graduate Program Director

__________ 19 ______ ______________________________
Dean of Faculty

Abstract

Recognition of Handwritten Numerals Using Elastic Matching

Patrice Scattolin

Elastic matching has been used for the recognition of handwritten characters for two decades. It is usually only used for writer-dependent systems with on-line data. We attempt to use this method in a multi-writer environment for both on-line and off-line recognition of handwritten numerals. By its nature, elastic matching is best suited to single writer on-line systems. Two challenges present themselves to attain reasonable results under these conditions. First, the algorithm must be modified to better generalize the models, to recognize a wider variety of patterns with a given number of models. Secondly the off-line data is not in a suitable format as the patterns are not represented by a sequence of ordered points. We will apply two modifications to the typical elastic matching system to adapt it to the multi-writer environment and for the off-line data. To process the off-line data, we use a stroke reconnection heuristic to create data nearly identical to typical on-line data. To adapt to a multi-writer environment, we modify the elastic matching algorithm to add weights to each point of the models. These weights allow portions of the characters that better discriminate between confusing classes to take on more importance. In so doing confusing classes are better separated, increasing the recognition rate.

Acknowledgements

I would like to thank my supervisor Dr. Adam Krzyzak for his patience during all of the delays throughout this work and for showing confidence in me by giving me the latitude I enjoyed in this work. I am also grateful to Dr. C.Y. Suen for his support, guidance, and help throughout my studies, and for handling my thesis defense. My thanks to Dr. Louisa Lam for providing updated thinned patterns as well as all her help in dealing with them. Thanks also go to Raymond Legault for his exemplary patience in answering all my questions and putting up with me in the same office for so long; Steve Malowany for the on-line data, UNIX system programming tips, and allowing me the use of his countless hours of LaTeXprogramming; Dr. A.D. Preece for his knowledge of expert systems and good drinking habits; Christine Nadal for providing the original off-line thinned data as well as the original singular point marking program; Dr. Charles Tappert for the introduction to the method, as well as Tetsu Fujisaki and Hamed Hellozy for making my stay possible, and all the other members of the on-line writing recognition team at the IBM T.J. Watson Research Canter at the time of my stay; Didier Guillevic and Nick Strathy for being such good Tim Horton and Peel Pub buddies; also thanks to Mrs. Grace Williams and Carol Gordon for the proofreading.

Finally Laurie, for the sacrifices, moral support and the patience to put up with my every whim during all this time. I am also grateful for the long hours she spent improving my writing abilities by a thorough reading of the work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Character recognition is a research problem that has been on-going since the sixties. Nonetheless, it still is an active area of research because the problem is complex in nature, and full of trade-offs. No solution has been offered that solves the problem both efficiently and completely.

The problem consists of recognizing which character is represented by a given set of points. It has been argued that, while this problem is complex, it is not fundamental. Solving character recognition is primarily a practical problem. This is echoed in the following quote from Jean Ward [111]:

> *An open question now, after over 30 years of published handwriting recognition research and development, is exactly what useful purpose our work will serve. There are very few, if any, successful applications using handwriting recognition.*

One is forced to admit that, apart from recognition systems, the work done in this field has found little application outside of the confines of the given problem. To be consistent with this observation, we will briefly look at the applications of this technology as well as at the constraints they place on the performance of the recognition system.

1.1 Character Recognition Problems

The first set of applications is the recognition of characters written on paper. This writing is then digitized to obtain a 2D image. This off-line processing can be done with any type of document or form. The two primary applications are the recognition of the address on an envelope and the processing of various financial documents, such as payment stubs and checks [41]. The important thing to notice is the cost associated with an error for either financial documents or with address recognition making an accurate recognition rate imperative.

The second class of applications using character recognition systems is found in interactive data acquisition. The user writes on a tablet with a specifically designed stylus, while the tablet digitizes its position at a given rate. Character recognition is then performed and the ASCII value is sent to the application. Such a system has been made commercially by Pencept, now defunct, and is described by Ward [112]. Other prototypes have also been described by Doster [28].

The Pencept system is based on an abstract representation of the strokes [113], such as straight lines and curves, and a set of model descriptions. The possible models are described as sequences of these abstract stroke representations. The abstract nature of the descriptions better lends itself to generalizing the representation of the patterns. As a result, this reduces the processing needed for recognition. However, some information is lost, making distinguishing between confusing patterns more difficult.

The main application for these early systems was the filling-in of preprinted forms. The user filled in a paper form with a pen, while the position was digitized, and recognition proceeded. It allowed limited interaction with the underlying application, making the integration of the pen with generic applications difficult, thus of limited usability.

Interest in this type of application was recently rekindled with the advent of new screen technology. The computer has a flat display upon which one can write with a special stylus provided with the machine. An electronic ink trace is echoed on the display, and recognition software is used to interpret the writing. Once understood, the words are passed to the appropriate application and the proper command is executed [8, 107], thereby replacing the keyboard and the mouse. The character recognition phase is now imbedded within the operating system, giving a machine where the pen is the sole mode of communication with the machine.

The interface that can be built with such a system is both powerful and intuitive [68, 88]. It also alleviates the duality often encountered with the typing in of data, and then its manipulation with a different object, such as a mouse.

In addition to the recognition of handwriting, the system can also recognize gestures or pictograms. Gestures are symbols that allow the user to specify various commands. Gestures, such as circling and crossing-out, and pictures, are both natural to the user and powerful in their expressivity [116, 71]. Because gestures are very dissimilar and can be rotated or scaled, a structural approach to recognition is preferred [56, 69].

The resulting tablet computer also makes the integration of both writing and drawings easy as the pen is the natural tool for drawing. Not surprisingly, the first applications targeted by the *Gridpad* are those where both graphic and textual data need to be integrated together [88].

In order for the interface to be truly natural, the recognition goal is 100%, as going back to correct previous mistakes removes the naturalness of the interface. It is thus important that recognition be reliable and the recognition rate high for a given user. Experimental work shows that users find recognition errors frustrating, as they don't necessarily understand them, and this hinders the acceptance of the system [89]. It is important that the recognition rate be high for the users the system trained on, but it can be low for other writing styles. Due to this requirement of reliability, methods that can accommodate a particular writing style are sometimes preferred [105].

We then have two main types of applications with different, sometimes diverging, requirements. Off-line applications are primarily the batch processing of financial documents from a wide array of writers. For these applications, because of the financial implications, it's imperative to be accurate. A lower recognition rate can be tolerated, but it must avoid making errors. Alternatively, on-line applications, almost exclusively pen-based computers, require high recognition rates with minimal errors.

Although an error does not have as much repercussion, it should really be avoided if the user interface is to be considered useful. This results in conflicting requirements: we either obtain a very reliable system with lower recognition rates but that may systematically reject certain writing styles; or we use an adaptable recognition algorithm that will allow us to handle rare writing styles, but which may lack in generality.

1.1.1 Recognition Methods

Pattern recognition, and character recognition in particular, has been attempted with many different systems and algorithms. We will briefly look at the most common ones.

The pattern recognition problem can be viewed as a signal processing problem where the closed contour of a pattern is considered as a periodic signal [80]. Such a signal can be expressed as a series of sums of complex numbers as given by the coefficient of the *Fourier Series*. The coefficients are then used as a feature vector from which distances will be calculated [40, 5], or used as a generic vector to be classified [59]. Fourier coefficients are invariant under affine transforms, such as scale, rotation and translation. This invariance proves to be problematic in the case of the digits “2” and “5”, as well as “6” and “9”, which are 180° rotations of each other and are resolved by other methods [6]. However Fourier coefficient do contain enough information to properly reconstruct the original shape suggesting that they do keep the bulk of the information [60].

Character recognition can be performed by matching unknowns against templates or sequences of templates [36]. In a method similar to templates matching in their organization, neural networks store information about the models implicitly in the weights attached to each connection. The first neural network, the Perceptron, can be considered a pattern recognizer. It accepted as input images of either **A** or **B** and then had to tell them apart [86]. Neural networks are given the binary image of the pattern to recognize, or sometimes a grey level representation of such a pattern. Successful networks are often multi-layered and use back-propagation, a more refined neural network, to perform recognition directly with the images [24, 23]. Networks can be cascaded to enhance the performance where the first level network has only to identify subfeatures, while higher layer networks assemble them to perform the recognition [35, 90, 94]. Alternatively, the networks can be fed feature vectors, such as a vector of Fourier descriptors, from which the network can be trained [59].

Hidden Markov Models represent another kind of fully trainable algorithm. Simply stated, a HMM is a finite state automaton where the transitions between states are probabilistic rather than deterministic. The transition probabilities are based on the probability of occurrence of the various features. The first terminal node reached is the recognized symbol. It is primarily used for cursive script, either off-line [17] or on-line, with low level metrics such as slope, curvature [10] or more syntactic features such as cusps and loops [11].

Syntactic pattern recognition is similar to HMM as they both make use, to a certain degree, of finite state automatons. With syntactic recognition systems we define a set of features, such as straight lines, curves, cavities, pointing up, etc. These features then form the tokens of a formal language from which we can generate grammar rules that describe the shapes to be classified. A crucial part of such a system is determining the features used, as their invariance represents the key to a successful system [21]. It is a popular technique that is applicable to a wide array of pattern recognition problems. Syntactic systems have been described for handwritten digits [20, 85], for printed alpha-numerals [26], on-line handwritten alpha-numerals [113], and for cursive script [30]. For high reliability it is sometimes favourable to have tight control on the feature extraction and the decision process. To this end, some syntactic systems are, instead, based on a decision tree. Depending on the first general features observed, various branches of the decision tree are taken and alternative features are evaluated on demand. This can be done manually (experts #1 and #4) [98], partly-automated [62] or completely automated [79].

But each of these methods has their respective weaknesses that create different errors with different patterns. To circumvent this problem, a combination of more than one recognizer may be used. Recognition gains in reliability since each type of error, specific to a particular recognizer, is counterbalanced by the results of the other recognizers. Such systems, of varying complexity, have been implemented starting with majority vote [98, 31] to more statistical approaches [51, 52]. The problem is still wide open, and there are no shortage of possible avenues to solve it [97].

We decided to use Elastic Matching which has been mainly used for on-line recognition systems, and attempt to improve on it to see if it can be made to serve as a viable off-line recognizer and a writer independent recognizer. To do so, we will describe an improved elastic matching algorithm where weights are added at each point of the model to emphasize the portions of the patterns that are more distinctive.

We will need an algorithm to find an appropriate set of weights. A training algorithm is also required to find an appropriate set of models to ensure that the models chosen are representative of the patterns that will be encountered.

As a first step, to be able to process the off-line data, we will describe the set of transformations that we need to apply to the image of a character to obtain data in a format identical to on-line data. Steps are also taken to reconnect strokes of the off-line patterns to obtain data that is nearly identical to on-line in its shape (number and orientation of strokes).

1.1.2 Elastic Matching, as a Recognition Method

Elastic matching has been successfully used with a variety of writing styles. Pioneered by Tappert as a character recognition method, many on-line handwriting recognition systems were developed with it. Tappert describes a system for run-on writing, where elastic matching is used as a recognition engine in a segment-recognize recognition system. Strokes of the patterns are taken and fed to the recognizer, where they are grouped together into characters. This system generates multiple possible word interpretations for a given set of strokes, but the final result is decided upon by complementary algorithms, such as a searches through a lexicon, or bigrams and trigrams [104]. It was later improved upon by using a *generate and test paradigm*, where the characters are recognized in smaller portions. Elastic matching is then used to determine which portion of which letter is present. Once hypotheses are established, the best ones are sent to a character recognizer for confirmation [34].

Another system is described where elastic matching is used as a recognition engine for cursive script recognition [100]. Elastic matching is applied to each letter of a word. Each letter is recognized until the entire word is processed. In later work, segmenting is explicitly applied within a word to isolate each letter [101, 102] thereby improving recognition.

Early work by Fujimoto attempted the use of elastic matching for recognition of scanned images. In this system a handwritten fortran program is digitized and converted to ASCII. After digitization, the patterns are thinned, and then they are "direction coded" to build sequences of points. A distance, based on a dynamic programming formula, similar to elastic matching, is calculated. If the distance is low enough, the pattern is recognized, it then improves the recognition rate by performing context verification, using the fortran language grammar rules. This system is quite complete, even by today's standards. The performance claimed, a 0.06% error rate and 0.20% substitution rate, is rather good [33], presumably because of the use of specially trained writers. His system is similar in function and design to the one we are presenting here.

Dynamic programming, similar to elastic matching, has also been used for the authentication of signatures [15]. Elastic matching is used as a distance measurement, based on a Euclidian distance, to determine whether a given signature sample is a forgery or an original. In similar work, Yoshimura uses an elastic matching method to compare signatures, adding pressure information [120]. It has also been used by Gorman to match partial generic contours described by Fourier series to process various shapes [40]. Dynamic programming is also used, by Ueda and Suzuki, to discriminate arbitrary shapes, matching shapes with various levels of refinement, or scale, to identify them. It uses dynamic programming to match inflection points between the unknown and the model. The model is then broken into subparts, delimited by the inflection points, that are refined and matched again until refinement is no longer possible, at which point the lowest distance measurement is considered matched [108, 109]. In a related application, a dynamic programming algorithm has been used to identify objects in grey scale 2D images. In this case, the cost of the match is incurred with a difference in grey level in the image [117, 118].

The term "elastic" has been associated with other methods for character recognition. Hinton creates models with a handful of points linked by splines. The elasticity comes from the deformation of the models used, at the given control points. The distance measurement from the control points to the unknown is expressed in terms of the probability of a control point explaining the presence of a given unknown point [49, 84, 115]. Burr also describes a similar system based on elastically stretching the models to fit the unknown, and in the process evaluating a cost [16]. Durbin uses the term elastic, in a similar fashion, to describe a method iteratively stretching a contour to fit a path, while the stretch is constrained by a cost function to keep the contour smooth [29]

One advantage of Elastic Matching is that, similar to a neural network, there is no need for complex feature extractions. We however believe that, because of this method's more direct measurement of the characters, it becomes easier to interpret the behaviour of the algorithm than with neural networks, where observation of the resulting weighting schemes yields little information about the characteristics considered important by the system. An elastic matching system where weights are applied, can provide a large degree of latitude in adaptive capacities of the algorithm; while retaining, for the developer, an ease of interpretation of the results of the intermediate steps, and helping in locating the areas creating most trouble to the recognition system. Doing so should prove valuable to the developer as it should make it easier to find modifications to apply in order to improve the performance.

1.2 Data

In character recognition, the data can take two forms, depending on the method of acquisition. The data is typically divided into either on-line data or off-line data. On-line data is captured as it is written by the user using a special purpose stylus to write on a digitizing tablet.

Off-line data is captured after the writing has occurred. The letters or numerals are written on paper with a regular pen that is then digitized with a scanner. This results in a two dimensional matrix of values, where each entry is a reading of the intensity of the reflected light.

1.2.1 On-line Data

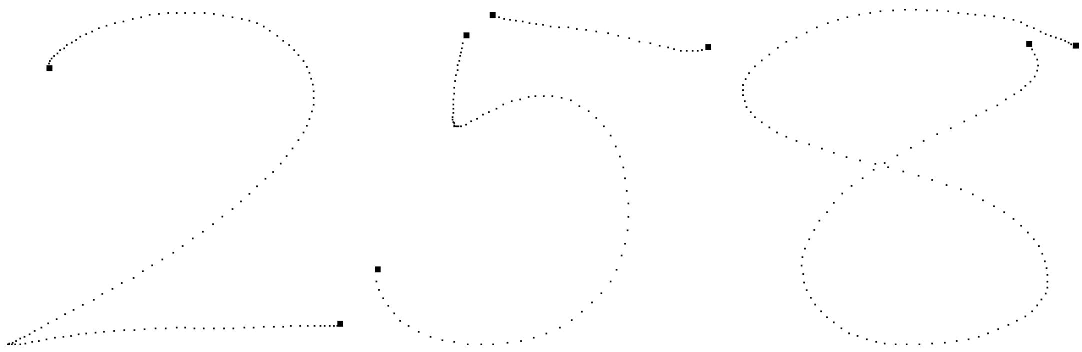

Figure 1: Sample On-line Data (before preprocessing)

The tablet captures the sequence of (x, y) coordinates of the pen tip in reference to the writing surface at a given scanning rate. Such devices are referenced in early work as methods to input handwritten cursive script [46, 32] or chinese pictograms [77], where information relating to a stroke, such as order and sequence, can be useful [22].

Data is captured from pen-down, when the pen starts touching the surface, to pen-up, when it leaves the surface. Although it is possible to know the position of the pen when it is not touching the surface, this is of little use for recognition as it is only when the stylus touches the writing surface that it can be interpreted as leaving an ink trace. It is useful to know the position of the stylus in the air, when the writing stylus also serves as a mouse, allowing the user to move the mouse without leaving an ink trace.

The on-line database we used was collected locally and used by Malowany [73] for his thesis work. The data was collected at Concordia in two different experiments. Both sets were written by subjects within the department of computer science, some even from CENPARMI. From this we obtained a training set containing one thousand samples, and a testing set containing twenty-three hundred samples.

From the results obtained by Malowany [73] we understood the data to have a few weaknesses. First, the instructions for collecting the data provided little direction to the subject. As a result, most subjects wrote characters that were much bigger (up to several inches in size) than they would normally write. Thus, this database, while containing sequences representing the ten digits, may not contain the difficulties and the subtleties encountered in regular handwriting styles.

This does avoid any potential shortcomings of the digitizing equipment used. As discussed by Ward [114], digitizing tablets of the period were more often designed as pointing devices rather than handwriting equipment. The precision of the location of the pen tip, large travel of the pen tip before registration of a pen down event, as well as general freeplay found in the pen tip itself, introduces undesirable noise that must be removed. The data used in our experiment is free of most of these problems because they occur at a scale much smaller than the writing we have. Another problem, noted by Tappert [106], is the parallax caused by the distance between the digitizing tablet and the flat display because influences writing style. This does not enter into play in our case, as we used an opaque tablet.

A second difficulty lies in the two databases' difference in level of difficulty. The training set is easier than the test set. Apparently, subjects who were members of CENPARMI, made an effort to make their data difficult, while others used consistent writing all through the experiment. Because the difference between the sets was marked [73], we repartitioned the database. Both sets were merged and then redivided, simply by alternatively placing each sample either in the training set or the testing set. This results in a training and testing set where the difficult cases are more evenly distributed. It does, however, have the drawback of having both sets being written by the same subjects, preventing testing as a true writer-independent system.

1.2.2 Off-line Data

With today's technology, scanners are available for various medias such as cheques, photos [25], and slides [19]. Grey level scanners associate one single value per pixel ranging from a binary value (0,1 or Black and White) to multiple grey levels.

Although a colour scanner [53] was used for some work done at CENPARMI with colour images of cheques, writing is usually treated as binary. We use binary data as it is readily available and relieves us of the need to convert grey level to binary images. To obtain this binary data, a thresholding operation needs to be performed. All of the pixels with a value above a certain threshold will become background (0), while the pixels below this threshold will become (1).

Figure 2: Sample off-line data (thresholded binary images, before preprocessing)

The database we used for off-line recognition is a well-known database of segmented handwritten digits described by Suen [98]. It was originally collected and first described by Ahmed [1]. Taken from dead letter envelopes from the U.S. Postal Services, it was digitized at about 166 dots per inch. After digitization, digits were segmented and thresholds were applied to obtain a binary image. The digits were segmented by looking for columns of white pixels separating them. Only the cases where the segmentation yielded five distinct blobs were kept. This ensures that the patterns present are composed of a single blob. This excludes cases like two-stroke fives, which are two blobs, and introduces a certain bias in the database.

Once thresholded and segmented, these images were then classified by a human to provide a reference against which to measure the results. Although the identity associated with some patterns can be questioned, the one provided is believed to be reliable [65]. As with most reported experiments, we use only a standard 6000 digit subset randomly extracted from this database, divided into two equal size training sets, **A** and **B** and a 2000 digit testing set.

1.2.3 Is On-line Data Easier to Process?

An easy answer to this question is yes. Looking at our results, we see that the recognition rate of the off-line data lags behind the on-line data. However evident this conclusion seems, one must question this assumption, because both representations of the patterns are recognized in the same fashion by humans.

So, if there is no reason for a human to interpret either type of data better, why would on-line data be easier to process? It is commonly believed that there is more information inherent in the on-line data. We can consider that there are two more sources of information available with on-line data: *acceleration* and *connectivity*.

On-line data is composed of pen tip positions at fixed interval in time. The larger the distance between two points the higher the speed: $v = \delta d/\delta t$. Similarly, the acceleration can be obtained by the difference of speed between two points: $a = \delta v/\delta t$. Because of the mechanics involved in writing, the pen goes slowly in corners and at the beginning of a stroke, while going much faster in straight lines. Acceleration becomes, then, highly correlated with curvature.

With signature verification we are looking for reliable signs to prove that, even though two signatures appear similar, they were produced by different persons [81]. Acceleration and pressure at the pen tip has been found to be reliable in this case because a forger will work only from the image of a signature, and will fail to reproduce the proper dynamics [76]. The dynamics of writing is more indicative of the writer himself than of the writing. As a proof, note that humans only need the static image of a pattern to recognize it with no acceleration information. The extra level of information provided by the dynamic information is often a source of error, and it is not uncommon to remove the dynamic aspect of the writing before recognition is attempted [73, 103].

It must be said that on-line data does give connectivity information. This information can be reconstituted, in part, from off-line data [27]. However, as we will see, the results, although good, are not perfect. In this respect, even with a stroke reconnection algorithm, on-line data has an advantage and this can, in part, explain why the algorithm performs better with on-line data.

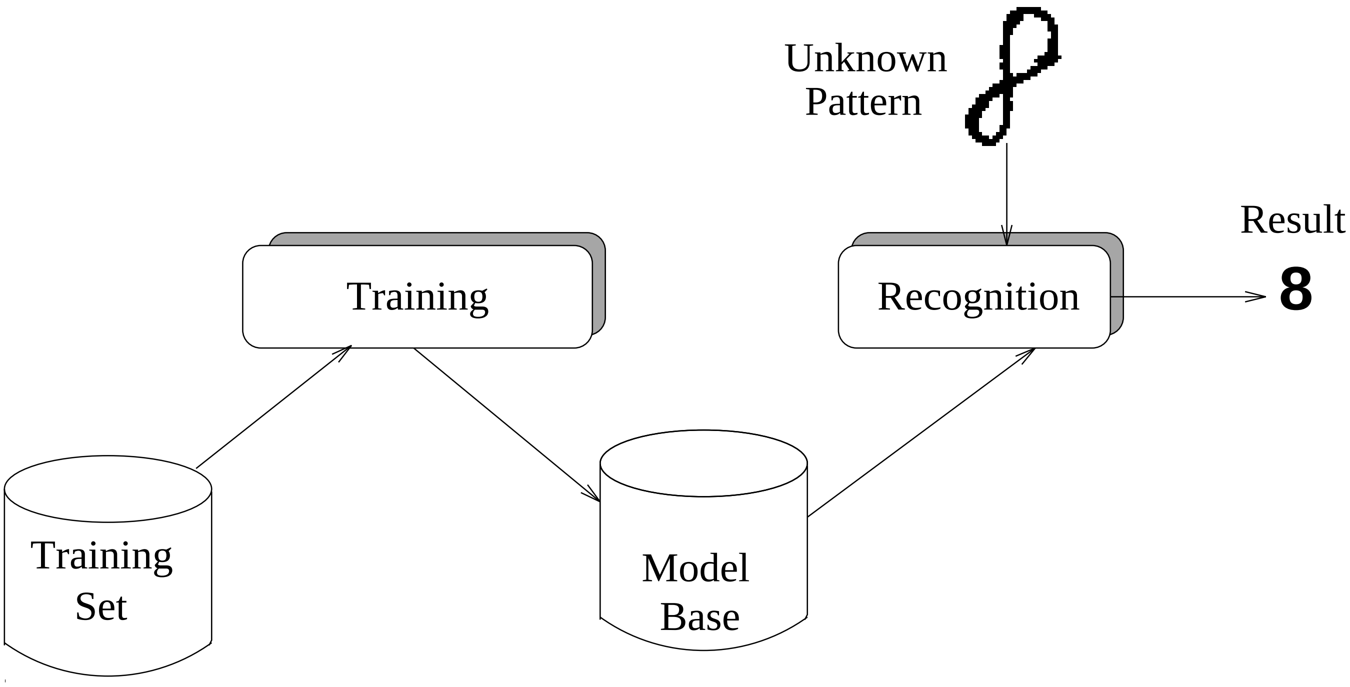

Figure 3: Overview of the system architecture.

1.2.4 System Overview

Elastic matching measures the distance between an unknown and a template. We therefore need to identify suitable templates. For this reason the system is broken into two major phases as illustrated by Figure 3. The first is a training phase, where we attempt to find the patterns that best represent the shapes of a given class. Training is typically done once at the creation of the system, and kept for the recognition of the unknowns.

The second phase of the system consists in a recognition phase where, once the set of models is established, unknowns are presented to the system. The distances between the unknowns and the models in the model base is computed. If the distance falls below a given threshold, the pattern will be identified as belonging to a certain class.

Chapter 2

Preprocessing Stages

Preprocessing is needed to remove insignificant scanning artifacts and noise. It is also used to give some low-level organization to the data, or to reduce the redundancy present in the data. Preprocessing is used to resolve areas that are problematic with the algorithm involved. Because the Euclidian distance is sensitive to positioning, we add a scaling step.

Although on-line and off-line data are digitized at different resolutions and with different devices, the noise problem is the same: data points are discrete and may deviate from optimal results. In the case of the tablet, data occurs as points that can be slightly off course, caused by some mechanical tolerances within the apparatus [114]. With off-line data, this error is caused by the discrete nature of the image itself and the thresholding operation.

Finally, there is a need to remove some of the data by normalizing the distance

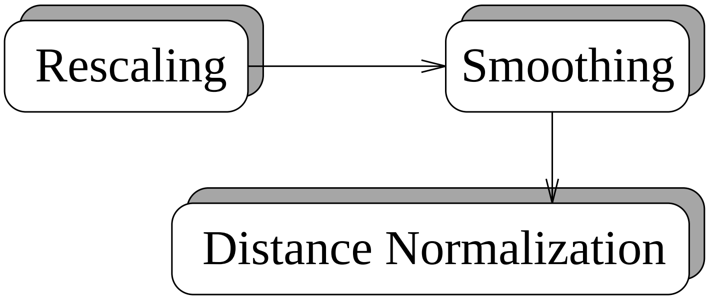

Figure 4: Sequence of generic preprocessing steps

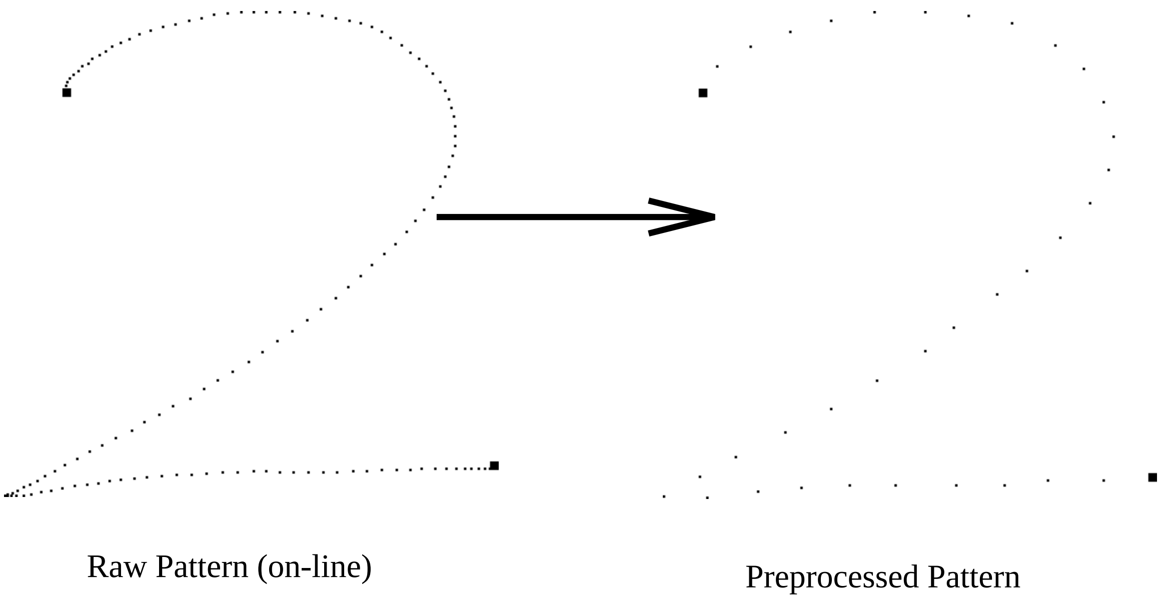

Figure 5: Results of generic preprocessing (on-line example)

between each neighbouring point. This ensures that the points are roughly equidistant.

2.1 Generic Preprocessing

The preprocessing stages we use, common to both types of data, are as follows:

- **Rescaling:** Normalize the character size.
- **Smoothing:** Eliminate digitizing quantization effects.
- **Equidistant Points:** Reduce data redundancy by reducing the number of points used.

2.1.1 Rescaling

When measuring the distance between points, it matters how the points relate to each other in space. If patterns are of a different size the distance measurement

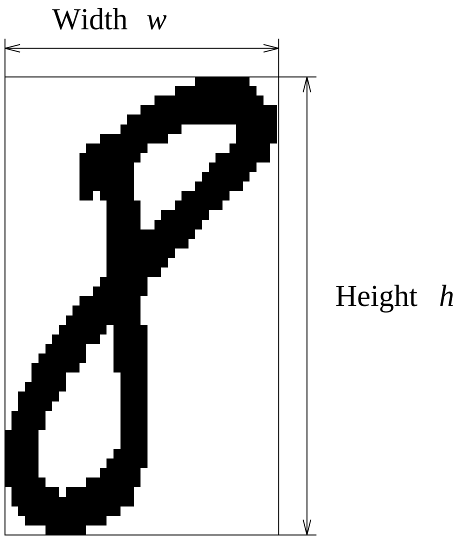

Figure 6: Character Bounding Box

would be large, even though they may have the same shape. It is therefore important to normalize the size of the patterns.

When rescaling, we need to preserve the shape of the pattern. In order to do so, both dimensions must be equally scaled. The algorithm we use is based on the *bounding box* of each already segmented digit. The bounding box is defined by the minimum and maximum (x, y) coordinates of the pattern. It's the smallest box in which the pattern will fit.

We first find the bounding box of the pattern, obtaining height h and width w. Rescaling is performed by applying the following transform to each pair of x, y coordinates: $c(x, y)$ where

$$c = max(100/h, 100/w) \tag{1}$$

Such normalization preserves the shape of the pattern while bringing the unknowns generally into a common size, with the largest of either axis 100 pixels in size, keeping the proportions intact.

Another rescaling is performed for purely computational reasons. As the measure of distances is computationally intensive, we looked to increase the speed by using integer operations instead of floating point operations. With current hardware floating point operations are typically two to ten times slower than the machine's natural integer even with hardware floating point depending on the architecture. In its original implementation, the speed difference between both types of data was quite appreciable.

The integer distance calculations are inaccurate at the scales contemplated of three or four pixels. The accumulation of errors would compromise the recognition rates. The pixel position resulting from this rescaling is multiplied by a given constant, in our case 16, thus creating a fixed decimal point representation, using regular integers. This technical rescaling is mentioned as it explains the high distance values between the patterns and the correspondingly high threshold values.

2.1.2 Intermediate Point Filling

Rescaling increases the pattern in size by a variable amount. This implies that the distance spawned between two points will no longer be equal for two different patterns and the points will no longer be well aligned with each other. This means that, although the point of the unknown may lie on the same line as the points of the model, the distance may still be relatively large. For this reason, we filled in the missing points after scaling had occurred, making points of unit distance once again.

In so doing we significantly increased the amount of data, hence increasing computational time as well. We had hoped that, because the points would be evenly spread throughout the model space, it would tend to reduce the cases where the points of the unknown fell between points of the model. This should have reduced the distance between patterns that are closely related and, as a result, improved recognition.

This step introduced a lot of new data resulting in much longer computation time without a corresponding increase in recognition. We stopped using this step as it provided few benefits.

2.1.3 Smoothing

Smoothing reduces the digitization errors present. Such errors are present in off-line, as well as on-line, data. The primary problem introduced by digitization noise is a local sharp change in angle. Because the distance measurement involves curvature and angle of elevation, we need to minimize these changes to obtain reliable data.

We perform the smoothing operation on the strokes. This form of smoothing treats the data as a simple 1D signal to which we apply filtering. This kind of smoothing has been described in similar character recognition systems [103, 107]. Their smoothing function for a point p_i is a weighted average of five points that is applied n times:

$$p'_i = \frac{-3p_{i-2} + 12p_{i-1} + 17p_i + 12p_{i+1} - 3p_{i+2}}{35} \tag{2}$$

This kind of smoothing is also used with off-line contour based systems [64]. In our case we use a simple average of three points applied n times.

$$p'_i = \frac{p_{i-1} + p_i + p_{i+1}}{3} \tag{3}$$

The number of iterations n is determined experimentally with the recognition system. We first create a recognition experiment with a model base, a given set of thresholds, and a test set. We then vary the number of iterations and observe the number of iterations giving the optimal recognition rate. In our case we observed the optimal number of iterations to be 1.

Recent work by Legault [64] surveys the field of smoothing 2D contours. Numerous approaches have been tried. Most of them can be stated in terms of weighted averaging. Weighted averaging for a point p_i is defined as

$$p'_i = \sum_{j=-k}^{k} \alpha_j p_{i+j} \tag{4}$$

where α_j is the weight for a given point and

$$1 = \sum_{j=-k}^{k} \alpha_j \tag{5}$$

The question is then to determine which weighted average smoothing is better for the application at hand. Legault [64] measures the amount of noise removed with various error measurement criteria. Analytical work was done for single pass smoothing filters with a smoothing window three pixels wide, applied to a digital horizontal line. The optimal smoothing parameter found is 1/3 for every point in the smoothing window, giving each point equal weight. Further experimental work done on digital circles confirms that giving an equal weight to each point is near optimal when the error is measured with mean square distance criteria.

We note that, although applying the equal weight smoothing twice is only near optimal, it may explain the difference in recognition with different numbers of iterations. The underlying question remaining to investigate is: which error criteria described [64] and, consequently, which set of optimal smoothing parameters, better models the needs of the recognition phase?

2.1.4 Point Distance Normalization

The final preprocessing step in generic preprocessing is to normalize the distance between points. We take the first point of a point sequence and then remove all the points that are at a distance smaller than a given threshold. Because this occurs after rescaling it ensures that, no matter the size of the original pattern, the points are evenly spread out. This step is used by Tappert and by Malowany [73, 103, 107] and, even though the recognition methods are different, they both benefit from the distance normalization.

This removes the acceleration information present within the data. Some may object and say that, in fact, this removes useful information. To find out, we performed recognition experiments. Using off-line data, we kept the recognition threshold and the model base fixed, varying only the minimum distance. We found that the recognition results improved when the minimum distance was increased to 6 (a distance of 6 pixels between points for patterns normalized to 100 pixels). A greater distance between pixels results in a progressive decrease in the recognition rate.

We would expect that, if the acceleration information were valuable, the best recognition results would be obtained with a minimum distance of 1 or 2 pixels and then steadily get worse. But we observe an increase in recognition up to a certain point where it levels off and slowly degrades. The fact that recognition does, for a while, increase while the distance between points increases, indicates that an even point distribution is preferred and suggests that acceleration does not contribute to

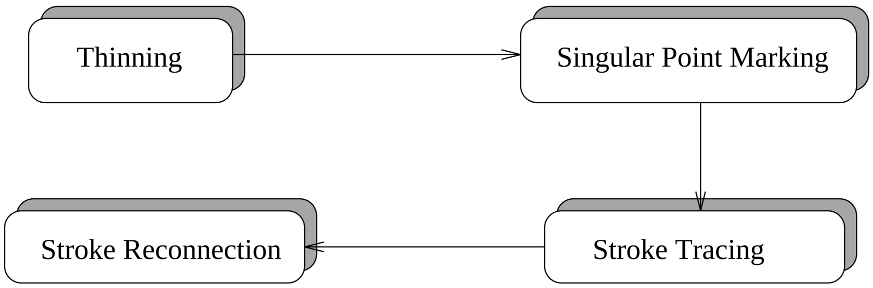

Figure 7: Off-line preprocessing steps

increased recognition. A similar conclusion, made by Lopresti and Tomkins, was that velocity is one of the least useful discriminants, while position related discriminants are highly ranked [70].

Because this algorithm ignores angular change, it may remove points at a cusp or a corner that would have been useful to handle "4" "9" confusion. This problem is shared by the smoothing algorithm and should be addressed by more discriminating algorithms.

We should mention that both the point distance normalization, and the smoothing steps, act together. Because of the possible interaction between these two steps, additional experiments were performed where the parameters for both steps were modified in concert. We obtained optimal recognition with one (1) smoothing pass and a six (6) pixel distance separating the points. We will keep these parameters constant throughout the rest of the recognition experiments.

2.2 Off-line Data Preprocessing

Other preprocessing stages are needed for off-line data. In order to use the off-line data we must first have sequences of points. The points are next to each other but the points have no predecessor or successor. We must then build, in a repeatable and reliable fashion, sequences of points from skeletons. The steps involved, illustrated in Figure 7, can be described as follows:

- **Thinning:** Thinning of pattern performed to obtain 1 pixel wide representation.

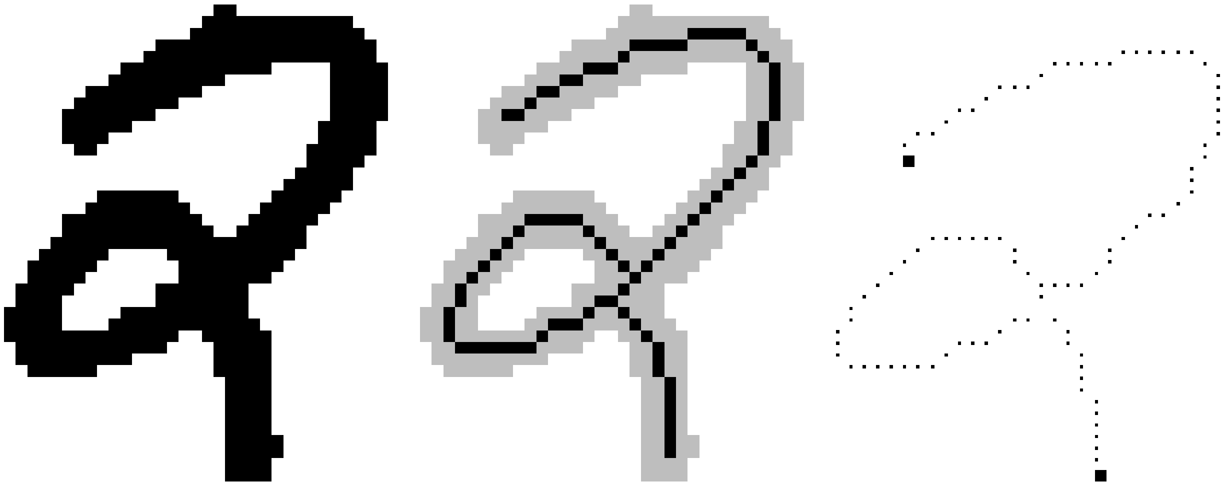

Figure 8: Transformation of off-line data into on-line data

- **Singular Point Marking:** Singular points are identified as they are required by the point tracing algorithm.
- **Stroke tracing:** An ordered sequence of points is created.
- **Stroke Merging:** Individual strokes are merged into larger strokes.

The skeletons will be traced to form sequences of points very similar to the sequences we obtain from the digitizing tablet. These preprocessing steps are performed before, and in addition to, the preprocessing steps used with the on-line data. The results of preprocessing are shown in Figure 8.

2.2.1 Image Processing

A binary image is composed of a collection of pixels that are either 0 for the background or 1 for the foreground, arranged in a two dimensional matrix. Each pixel has eight neighbours. Four of these pixels are immediately adjacent, as they share an edge, and are known as *4-connect* neighbours. They are, from Figure 9, pixels x_1, x_3, x_5, x_7. A pixel also has diagonal neighbours, known as *8-connect* neighbours, pixels x_2,x_4,x_6,x_8.

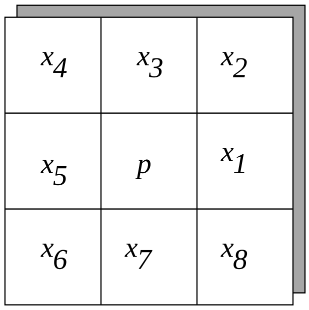

Figure 9: A pixel's immediate neighbours

Each image contains only one pattern. The pattern is typically composed of only one contiguous set of foreground pixels called a blob. In our case, blobs are 8-connected, a constraint imposed by the data. Digital geometry then dictates that the background is 4-connected [87].

2.2.2 Building Sequences

The contour of a pattern can be followed, pixel by pixel, to form a sequence of points. But a pattern may have more than one contour. An external contour is a contour bordering between a blob and the external background and is more indicative of the general shape, but internal contours are needed as well.

Contours require less processing, and tracing algorithms are well known. However, the contour does not really represent the position of the pen tip accurately, as the contour is composed of both sides of a stroke and it introduces sharp turns at a stroke end.

A skeleton is defined as a thin-line representation of a pattern, where the line is composed of the minimum number of pixels. Skeletons typically give points in the middle of a stroke similar to the sequence of points of on-line data.

Because the elastic matching algorithm was first used with on-line data, that is essentially pen tip position, it was felt that using skeletons would represent less of a departure from the original data used. Since the computational time involved with elastic matching is directly proportional to the number of points matched, it was felt that the use of skeletons would also improve the performance in terms of recognition speed.

2.2.3 Thinning

Thinning basically consists of reducing a pattern to a thin-line representation. Thinning algorithms can be divided into two categories: *contour-stripping* and *non-iterative* [61]. A non iterative algorithm attempts to find the skeleton by finding which pixels are in the "middle" of a stroke, by tracing both sides of the contour. This presents difficulties that are overcome with heuristics. The two types of contour stripping algorithms are known as *parallel* and *sequential* thinning algorithms. In sequential thinning algorithms, pixels are examined in a predetermined order, and changing the order implies changing the resulting skeleton. In parallel thinning algorithms, the pixels can be examined simultaneously, since only the results of the previous iteration influence the outcome of the current iteration, which proves advantageous for parallel implementations [50, 48].

Our experiments started with a version of the parallel thinning algorithm described by Zhang [121] later modified as proposed by Lu [72]. At a certain point a run length encoded version of the algorithm by Zhang [121] was also used to process the digits. Instead of processing the bit map representation of the patterns, the algorithm directly treats the run length encoded image, which speeds the process as it easily skips white runs in the patterns.

Later, more data was added and an improved sequential algorithm became available [63]. On each iteration it traces the contour [87, 119], removing points marked for deletion. Under certain conditions, some points are marked to avoid deletion, primarily at sharp corners.

We changed the algorithm to improve the quality of the skeletons produced, mainly to be relatively free from noise in the form of small spurs of a few pixels. This newer algorithm also preserved the geometric properties of the pattern better. Comparisons between different algorithms show that preserving the geometric properties of a pattern is not always an easy task [99], especially with patterns of varying widths. One controlled experiment was conducted with human subjects [82, 83]. Their opinions were asked on skeletons to rank them for the quality of the results. The algorithm on

End point	only one black neighbour
Skeleton point	point with exactly two black neighbours
Junction point	point with more than 2 black neighbours

Table 1: Point types on skeletons

which the one we use is based [18] finished among the first and the current one should fare better. Since elastic matching makes no assumptions about the behaviour of the thinning algorithm, it can be an objective measure of the quality of thinning algorithms, and the recognition results would determine how the quality of the skeleton influences the overall goal of recognition.

Thinning is prone to failure in cases where an image is made up of blobs that are of irregular shape [4]. Although it is not always critical, stroke width can be useful to distinguish confusing cases [65]. Still, skeletons are adequate to perform recognition but are disadvantaged in these cases.

2.2.4 Singular Point Marking

The points resulting from the thinning algorithm are still not in sequence. The sequence is created by a slightly modified version of the contour tracing algorithm that is described by Rosenfeld and by Yokoi [87, 119].

Suffice it to say that a sequence starts at a singular point and ends at a singular point, and is composed, in order of appearance, of all the skeleton points between both singular points. Singular points can be either junction points or end points. We first give a definition of the three types of points present in a skeleton in Table 1.

An example of the results produced by the above definition, given in Figure 10 is taken from pattern "*88 42 3 35 29*" of training set B.

Figure 10 contains a pattern that has three different junctions. However, we see that one of the junctions is composed of three junction points. These points are correct according to the definition as it is given in Table 1, but two of these points are not needed because each can belong to just one stroke.

Because it simplifies the detection of connected strokes, extraneous junction points will be removed. We use the *generation and test* paradigm, whereby junction points

```
                  .....
             ....MMMMMM...
          ...MMMM......MM..
        ....M....      .M.
       .EMMM..         .M.
       ......          .M.
                       .M.
                      .M..
                    ..M..
                  ..MM.
                ..MM..
             ...MM..
     ..........M..
  ....MMMMMMM.M.
...MMM.......J..
..E.....   ...M..
....         ..M..
              ..M.
               .M.
                .M.
                ..M.
                 .M.
                 .M.
                 ..M.
 ......           .M.
..MMMMMM.  ....   .M.
.M......MMMMMMM.....J....
.M        .....M...M.MMME
.M             .JJM..
..M..          .J...
 ..MMM.........M...
   ...MMMMMMMMM...
      .........
```

Figure 10: End and Junction points before removal of extraneous junction points

$8jctcnt$	$=$	0	Lonely point must be kept
$ptcnt$	$=$	5	Four branches. Removing would break continuity
$4jctcnt$	$=$	$8jctcnt$	All connected junctions are non consecutive.
$4jctcnt = 0$	$\wedge$	$8jctcnt \neq 0$	Keep only if has diagonal junctions.
$4jctcnt$	$=$	$max4cnt$	Keep the centre of a "+" cluster of junctions

Table 2: Junction point keeping conditions.

```
.M.M.
..J..
..J..
..J..
.M.M.
```

Figure 11: Short straight segments of junction points

are generated to be tested and, if necessary, removed in three consecutive steps. Each of the conditions used by the three steps are confined to the neighbourhood of the pixel evaluated, making the algorithm parallel.

In the first stage, we evaluate the count variables and eliminate all of the points that do not conform to one of the conditions in Table 2. We also remove a point if the following condition is true: $8jctcnt = 3 \wedge 4jctcnt = 1$ thereby removing the points at each corner of a "+" shape junction.

We define $8jctcnt$ as a count of 8-connect neighbor points that are marked as junctions. Similarly, $4jctcnt$ is the count of 4-connected neighbours marked as junctions. Finally, $ptsum$ is defined as the number of skeletal points neighbouring the pixel, and $max4cnt$ is the count of the total number of 4-connect points.

The second stage removes the middle point of horizontal and vertical sequences of junction points as in Figure 11. The count variables are re-evaluated as needed, and points are removed if they meet the following condition:

$$4jctcnt > 1 \wedge \ 8jctcnt - 4jctcnt = 0 \tag{6}$$

The last step removes micro-spurs. A *micro-spur* is a junction that creates two pixel strokes. This is caused primarily because the skeleton is not strictly unit thin.

```
  ..M..
 ...M.
...J.
..jJ...
 ...MMMM.
```

Figure 12: Micro-spur branch to be removed

Such a case can be seen in pattern "*71 9 2 19 20*" from training set B in Figure 12.

In Figure 12 the point "j" is referred to as a micro-spur. This point will be removed and the two junction points will be relabelled as skeleton points. A point is considered a micro-spur if $8jctcnt = 2 \wedge ptsum = 0$, and if the two junction points are consecutive. This step modifies the skeleton as it removes one point from it. The skeletons provided by Louisa Lam [63] only have 5 such occurrences over a total of 6000 patterns.

2.2.5 Branch Tracing Algorithm

We create a point sequence by using a modified contour tracing algorithm [87, 119] that assumes an 8-connected pattern. It looks at the immediate neighbours of the pixel counterclockwise (see Figure 13). Each of the neighbours are visited counterclockwise, starting at the last pixel we visited, until we have visited all of the neighbours, or until we have found a pixel that is not part of the background. Suppose the point preceding the current point p, in the trace is its neighbor at position x_4, we will then start looking at position x_5 to find our next neighbor.

We start tracing from one singular point and stop at the next singular point marking skeletal points to prevent their reuse. We will start at least two different searches from a junction point. Because the marked pixels will be skipped, the algorithm is forced to use each of the different neighbours of the junction point. In order to trace all of the strokes, we apply the algorithm to each of the singular points, until all of their neighbours are marked as used.

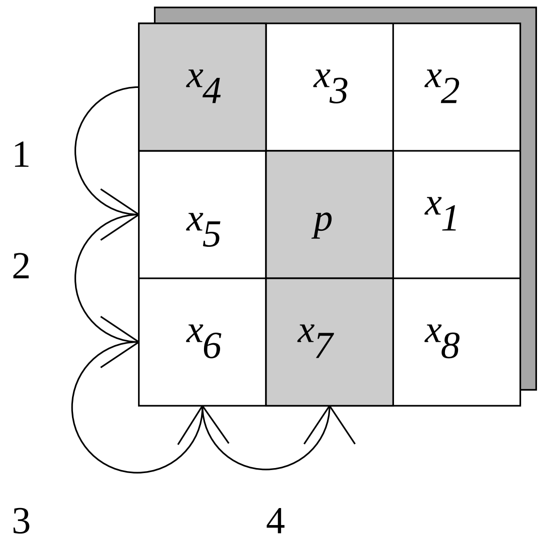

Figure 13: Counterclockwise walk of a pixel's neighbours

```
        ........M...
...............M...
.......MMbaJxMM...
...........z......
...........M......
...........M......
```

Figure 14: Potential tracing infinite loop (with 8-connect)

Unfortunately, most skeletonizing algorithms do not guarantee the type of connectivity achieved. Hence, the skeleton is, almost always, a mix of 8-connected and 4-connected segments. A strictly 4-connected skeleton is often viewed as being too thick, with diagonal lines expressed as staircases. On the other hand, some junction points must be 4-connected, otherwise it would break the connectivity between segments. As a result, we modify the tracing algorithm as follows: *When an 8-connected point is encountered, verify if its following 4-connected neighbor is also a skeletal point. If so, use it instead.*

Using strictly 8-connect may result in infinite loops. Consider the example shown in Figure 14. Say tracing starts at the junction "`J`" down to point "`a`" , "`b`" and so

```
           .........
           bcMMMMM..
     ...MMMa
```

Figure 15: Omitting points while tracing (with 8-connect)

```
  .EMMMMMMMd    ......
    .....  .b...MMMM
           .JacM...
           .J..
MMMMMMMMMMMM M.
........    .MMM..
```

Figure 16: Neighbouring points near a junction point

on. Arriving back from point "`z`" the tracing algorithm then has the choice of using point "`x` or the junction "`J`". Because it is 8-connected, the algorithm will choose point "`x`" before seeing point "`J`". In doing so, it will start tracing another branch and will not complete the loop properly.

Using strictly 8-connected rule also results in omitting some points with the typical staircase (Figure 15). In this case, as we are tracing from the point "`a`" to the point "`c`", the 8-connect algorithm can omit point "`b`" completely.

Another problem now appears at junction points and is illustrated by Figure 16. Point "`a`" and point "`b`" are legitimate 8-connected points. Depending on the direction of the search, either point "`b`" or point "`a`" could be found by the tracing algorithm. Once point "`b`" is chosen, the choice of the next point to continue the trace is important. If it started the search at the regular starting point, it would choose "`a`" and would trace the wrong branch, the branch with point "`c`". However, to trace the second stroke, the one containing point "`d`", a problem would arise since "`b`" is already in use and can't be reused, omitting a branch. As a direct consequence of the modification introduced above, a special rule applies to initiate the search for the third point of a sequence, to avoid the problem just described: *Start the search sequence on the next 8-connected point that follows the normal start point.* This makes the search avoid neighbours that are common to both the first and the second point.

2.2.6 From Off-line to On-line Data

It was observed by Govindaraju, Wang and Srihari, that on-line character recognition systems are more successful than off-line systems, and it is believed that recovering the stroke order information may allow the use of the techniques used with on-line data when they can be of benefit [43]:

> *Research in this paper is motivated by the need for an integrated system where the off-line and on-line techniques can be interchangeably applied, independent of the data capture process. Such a unified approach is possible only if the data representation in both on-line and off-line writing is the same. Since the on-line representation has inherent advantages, it will be useful to transform the off-line data into "on-line type" representation.*

In our case, the data is stored in exactly the same format for both on-line and off-line data. But with the converted off-line data we have more strokes. In fact, each time there is a junction a stroke is created because the tracing algorithm cannot reliably determine which of the possible branches must be taken. The basic problem to be solved is to decide which strokes belonging to a given junction should be reconnected.

There are essentially two views we can take of this problem. We reconnect two strokes, which must have been written using a continuous pen movement, due to restriction on direction changes imposed by the pen-hand-arm writing system [43]. The second view states that we reconnect the strokes as the brain would. This draws on knowledge from Gestalt psychology, which essentially states that the brain prefers the organization that changes the least. This has been formalized by the *Good Continuity* rule as implemented in a computer version [55]. We are in fact lucky. Both of these views, approaching the problem from different sides, the perception side and the writing generation side, happen to coincide to the same rule of interpretation of the data. This happens to be a convenient convergence of interest.

Perceptual Basis for Reconnecting

Gestalt dates back to Kurt Koffka, Wolfgang Kohler and Max Wertheimer, known as the Gestalt psychologist, in the 1920's. Their primary contribution was to study how stimuli integrate during perception [92]. While they applied their theory to wide areas of human behaviour [57], of primary interest to us is their contribution to visual perception, which specifies how each element combines into a larger organized whole. In doing so, they stated the following laws of perceptual organization taken from Spoehr [92]:

- **Principle of proximity:** Grouping of individual elements occurs with nearness or small distances.
- **Principle of similarity:** Elements physically similar are grouped together.
- **Principle of common fate:** Elements moving simultaneously are grouped together.
- **Principle of good continuation:** Elements are grouped together if they yield few changes in a continuous line.
- **Principle of closure:** Elements that form closed figures are grouped together.

These rules, as general as they are, fall under the **Law of Pränganz**, first stated by Wertheimer, and formulated by Koffka [57] as:

> *Psychological organization will always be as "good" as the prevailing conditions allow. In this definition the term "good" is left undefined.*

Attempts were made to develop a process, implemented on a computer, which would model these principles as well as the early vision processes that are known to occur in the brain [110]. Of main interest to us is the *Principle of Good Continuation.* This implies that, at a junction point, the best interpretation is the one where there is the least amount of change in a continuous sequence, either a curve or a line [27]. It has been implemented as minimizing the angle difference between the two strokes meeting at a junction point [42, 55, 78]. We use the same general definition

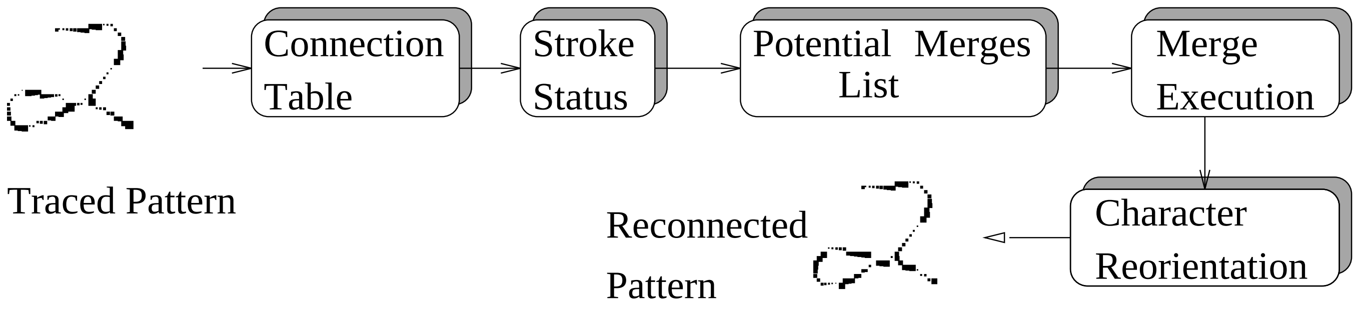

Figure 17: Stroke reconnection steps

of continuity used above. Two strokes are considered to have *good continuity* if the difference in the angle of elevation is less than 45^o:

$$\|\phi_i - \phi_j\| < 45^o \tag{7}$$

where ϕ_i and ϕ_j are calculated as in Equation 13, and the points used in calculating ϕ are

$$\begin{array}{cl} p_n \wedge p_{n-c} & \text{at stroke end} \\ p_0 \wedge p_c & \text{at stroke beginning} \end{array}$$

where c is an arbitrary constant established empirically and is currently set at $c = 5$. If the condition stated in Equation 7 is true, we can allow the reconnection of the two strokes at their extremities i, j. The algorithm described in section 2.3 explains how this is done. The 45^o limit is an upper bound to avoid reconnect widely different strokes.

2.3 Stroke Reconnection Algorithm

The reconnecting algorithm is illustrated in Figure 17. We will describe each of the stages separately. At this point we assume that the strokes have been traced and we have a sequence of points. The steps can be briefly described as follows:

- **Connection Table:** Creation of a table listing which stroke is connected to the other.

- **Stroke Status:** Give a stroke type to each stroke.
- **Merge List:** Create a list of all possible merges.
- **Merge Execution:** Perform the merge operation following a given priority.
- **Character Reorientation:** Reverse the order of points within the resulting character as needed.

2.3.1 Connection Table

The connection table is a list of all the singular points, and for each singular point: which extremity of the stroke, either the start or the end, contains this singular point. As a result, we obtain a list of the all the strokes that touch each other because they share a singular point. We call such strokes *Adjacent strokes.*

There are cases where two junction points may be neighbours. They can't be removed, as doing so would break connectivity. Figure 18 illustrates a case where two junction points are immediate neighbours of each other, the other point being 8-connected. We call such junction points twins. If one of the junction points was removed, it would then be a member of both strokes. The tracing algorithm, in order to complete the trace of a stroke from singular point to singular point, would have to pass over a skeletal point twice but, by definition, this is disallowed. As a consequence, the twin junction points must be kept to keep the tracing algorithm working.

To simplify the detection of adjacent strokes, we added an intermediate step to merge twin junction points after tracing has been performed. If two junction points are closer than a certain threshold, we consider them candidates for a merge. We then pick one of the twin points and change all of the strokes involved to make this one point common to all of the strokes, making the strokes adjacent. The threshold used is $\sqrt{2} + 0.1$. This distance is small enough to make only twin junction points, as the ones illustrated in Figure 18, candidates for this merge.

Increasing this threshold value would allow reconnection of strokes that are not touching. It would then become a broken stroke reconnecting algorithm which is

```
 57    20     2    38    31
          ......
          EMMMMM..
               .M..
                .MM.
                 ..M.
                   .M.
                   .M.
                    M.
                    M.
                    .M
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                    .M.
                   .M.
                   .M.
                   .M.
   ...             .M.
 ..MMMM.......    .M.
..M....MMMMMM.....M..
.M       ....MMMMJ...
M            .....J..      .
M              ..M.MM..    .
M.             .M....MM.....
.M.           .M.......MMME.
 .M.         .M..     .......
  .MM.    ..MM..        .....
   ..MMMMMMM..
      ......
```

Figure 18: Immediate neighbour junction points that can't be simplified

useful in cases where noise breaks a stroke. It was successfully attempted on newer data. Further criteria, such as the good continuity rule, may be needed to restrict the unwanted merging of unrelated strokes.

2.3.2 Stroke Status

We then create a stroke status table which contains, for each stroke, the type of stroke and whether or not either end of the stroke has been merged. The strokes can be divided into five categories:

- **Simple Stroke:** A point sequence with end points on both ends. Such a sequence will not be merged.
- **Stroke:** A point sequence with one end point and a junction point. Such a sequence can be a candidate for reconnecting at the junction point.
- **Bridge:** A point sequence that has a junction point at each end. Can be a candidate for reconnecting on both ends.
- **Small bridge:** A bridge that is shorter than a given amount. Can be merged on both ends or jumped over.
- **Loop:** A point sequence that has a junction points at each end but both junction points are equal. Can be merged on both ends or with itself.

2.3.3 Small Bridges

We allow strokes to be merged only once. This means that, if two straight lines cross each other, each would need to go through the bridge that inevitably forms at the junction as an artifact of all the thinning algorithms. A bridge, and how it connects two strokes, is illustrated in Figure 19. A bridge is typically short in length, and can form a sharp angle between the two branches of the two strokes it connects. The bridge will generally merge well, according to the good continuity rule, with the point sequence that form one of the strokes, but will form a relatively sharp angle with the point sequence of the other stroke.

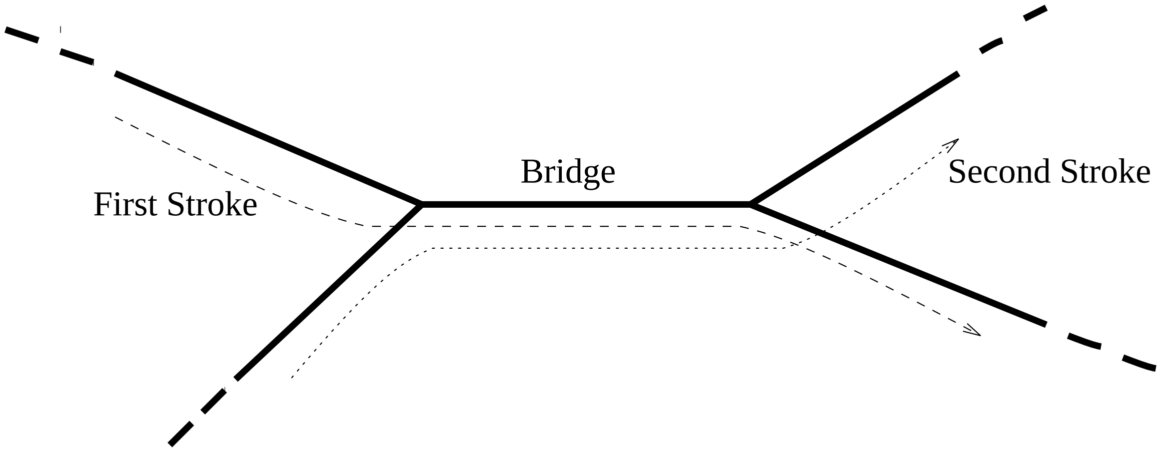

Figure 19: Bridge linking two strokes

All strokes present at the bridge are competing for its use. One stroke would, arbitrarily, win and prevent the other strokes from merging. By observation of the data, we find that length is what best characterizes the bridges that are formed at the junction of two line segments. Consequently, we distinguish bridges that are smaller than a certain threshold to be *small bridges*. Currently, the threshold is set at 10% of the total pixel count for the entire pattern. Small bridges are the only type of stroke that can be used more than once in merge operations. We say that two strokes are attached by a small bridge if they both exist on each side of the small bridge.

2.3.4 Potential Merges

We define potential merges as all the possible combinations of strokes that respect the good continuity criterion. For strokes attached by a small bridge, the good continuation criterion is calculated with the points of both strokes involved, the bridge not entering into play. This is needed, because thinning will often make a bridge relatively parallel to one of the pair of the candidate strokes. The two other competing strokes may form a large angle with the small bridge, but the angle difference between the two strokes will be small. In these cases small bridges show a preference for one of the stroke pairs, but merging of both stroke pairs should be preferred. Ignoring the bridge in these calculations gives a more reliable system.

We added another condition to the use of small bridges for stroke merging. In order to be able to use a small bridge to merge two strokes, we require the small bridge to respect the good continuity rule with at least one of its adjacent strokes. This is the condition required for a *good bridge*. This is to ensure that, not only is it small enough, but that it is also generally in the right direction of at least one of the strokes.

This is needed because the character "4" can often have a small bridge present in it. The bridge will show up as a short horizontal stroke linking two vertical strokes. Because of the algorithm's preference for merges that involve jumping over bridges, this makes it possible for the top left stroke to merge with the bottom right stroke. Since the small bridge will be at a 90^o angle from the other strokes, it will not meet the good continuity criterion for either of the two strokes involved. This prevents it from being a good bridge and consequently prevents this erroneous merge from taking place. Two strokes, on either side of a bridge, can be merged if:

- Both strokes conform to the *good continuity* rule with each other.
- Both strokes are attached by a *good bridge*.

All potential merges, those of adjacent strokes, or those attached by a good bridge, are added to a merge list.

2.3.5 Merge List

The merge list is a linked list that contains an entry for each of the potential merges that can be performed. The entries contain enough information to perform, in a later stage, the merge operation. The contents of the merge list entries are as follows:

- **Merge type:** The type of merging operation that is required. Can be **bridge-merge** or **merge**.
- **First item:** Stroke id to be merged.
- **Second item:** Id of the second stroke needed.

- **Bridge id:** Id of the bridge stroke, if necessary.

In a merge list element we keep a pointer to the two strokes that are to be merged. We also note whether the merge operation is to take place with the head of the stroke or with the tail of the stroke.

The use of a list is required, because more merges are proposed than will be done. First, it is possible that the good continuity criterion applies to a given stroke more than once. Since only one merge can be performed, the decision as to which merge to perform is delayed. Merges over good bridges have priority over regular merges, so all possible merges must be computed first before any are performed to maintain this priority.

Performing the Merge

The merge is performed by creating a new stroke from the concatenation of the points from the two strokes following the rules outlined below.

- A junction point cannot be merged more than once.
- Merges over Good Bridges are performed first.
- Good Bridges can be used more than once in bridge merges.
- Good Bridges used in bridge merges can't be used in regular merges.

Note that, if we are attempting to merge the heads of two strokes, one must be reversed so that the same point is at the end of the first stroke and at the head of the second stroke. Because of this, it is no longer certain that the stroke ordering implied by the tracing algorithm will be maintained.

Our method of identifying small bridges is similar to the one presented by Ogawa [78]. We have two primary conditions in common. Bridges must be relatively short and, secondly, the angle between the strokes on both parts of the small bridge must be small enough. This ensures, in part, that the strokes involved are related and that they obey the same general good continuity criteria as used to join strokes. Small bridges are also used by Govindaraju [43] and are identified as *Cross Stroke Structures*. The rules he uses result in merging that is similar to that of Ogawa.

```
................
EMMMMMMMMMMMM...
.............E..

................
.MMMMMMMMMMMME..
E...............
```

Figure 20: Two nearly identical strokes with different starting points

2.3.6 Character Reorientation

We must remember that stroke orientation plays an important role in the elastic matching algorithm. A stroke and a reversed copy of itself are not a good match. In Figure 20 we see two similar strokes that are nearly identical, except for the position of their endpoints. The tracing algorithm will give two strokes with different orientations. The top stroke is going from left to right, and the other one is going in the opposite direction. This is caused by the change in starting point for the tracing algorithm, and it introduces unnecessary variations in the strokes.

To reduce the number of variations introduced by a simple change of order, we use a stroke reorientation procedure. We want to keep the same kind of orientation as provided by the stroke tracing algorithm. This orientation is, first, from top to bottom and, within a line, from left to right. Using this criterion to determine which end of the stroke should go first can work well, but it is very sensitive to a change in the height of both endpoints as illustrated in Figure 20.

We use a simple criterion to determine which end of the stroke should be considered the beginning. We take the end that minimizes the distance to the upper left corner of the character. This distance is given by:

$$dist = \sqrt{(x_{Orig} - x)^2 + C(y_{Orig} - y)^2} \tag{8}$$

where x_{Orig}, y_{Orig} is the origin and C is an empirically determined constant. The role of C is important; it elongates the pattern in the y-axis. In doing so, it creates a bias for points located higher. With this bias, it will still prefer points that are located near the top of the character but it does not enforce a strict order. With the example given in Figure 20, both strokes would now have the same orientation. This simplistic criterion is not intended to perfectly solve the problem of spatial ordering; it does, however, give a solution that works well with the data at hand.

Loop Reopening

Because the starting and end points are equal, the minimum distance criterion will not work for loops. We want loops to have a consistent orientation, which is counterclockwise as is ensured by the tracing algorithm. At the same time, we want loops to always start at the same location, at the top leftmost pixel. We call this operation *loop reopening.*

The good continuity criterion will merge the two sides of the loop together, leaving the noise or stroke isolated. The merging algorithm then creates loops in the normal course of merging.

Loop reopening consists of two steps. The first step is to find the leftmost point in the topmost line of the loop. This becomes the new beginning of the loop. After the new beginning of the stroke is found, the second step is to ensure that the direction of the loop is counterclockwise. A counterclockwise orientation is intuitively understood for a circle, but not so obvious for the character eight. So, for our purposes, we require only that the sequence rotates counterclockwise at the beginning of the stroke. This gives consistent results with the digits, minimizing variations in all types of models.

We determine the direction of a loop with the following test. Since we have the leftmost point of the topmost line, any point that is found will be below it. So, if the point y_1 is to the left of y_0, the stroke is counterclockwise; otherwise, it's clockwise and will be reversed.

2.3.7 Reconnection Results

The success of the reconnection process is measured by a count of the number of strokes for a given pattern. Essentially, we are looking for a reduction in the stroke count. We hope to arrive at a stroke count that is as low as on-line data, on-line

Char id	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	200	151	45	2	1	1	0	0	0	0	0	0	0	0	0
1	200	197	0	3	0	0	0	0	0	0	0	0	0	0	0
2	200	22	6	78	70	19	2	2	1	0	0	0	0	0	0
3	200	46	2	122	9	18	1	1	1	0	0	0	0	0	0
4	200	0	0	62	11	86	10	31	0	0	0	0	0	0	0
5	200	111	5	66	5	7	4	1	1	0	0	0	0	0	0
6	200	7	147	18	27	0	1	0	0	0	0	0	0	0	0
7	200	143	1	51	0	5	0	0	0	0	0	0	0	0	0
8	200	0	21	69	85	17	4	1	3	0	0	0	0	0	0
9	200	22	110	50	13	3	2	0	0	0	0	0	0	0	0

Table 3: Number of patterns with a given stroke count before stroke reconnection

Char id	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	200	181	18	0	1	0	0	0	0	0	0	0	0	0	0
1	200	197	3	0	0	0	0	0	0	0	0	0	0	0	0
2	200	95	91	13	0	0	1	0	0	0	0	0	0	0	0
3	200	49	106	38	6	1	0	0	0	0	0	0	0	0	0
4	200	0	123	69	8	0	0	0	0	0	0	0	0	0	0
5	200	117	68	13	2	0	0	0	0	0	0	0	0	0	0
6	200	163	35	2	0	0	0	0	0	0	0	0	0	0	0
7	200	144	50	6	0	0	0	0	0	0	0	0	0	0	0
8	200	174	19	6	1	0	0	0	0	0	0	0	0	0	0
9	200	121	73	6	0	0	0	0	0	0	0	0	0	0	0

Table 4: Number of patterns with a given stroke count after stroke reconnection

being the lower boundary. Intuitively, we would believe that the stroke count should be one. However the digits “4” and “5” are often written with two pen strokes, as well as the crossed zero “0”. Since the on-line database and the off-line database are different, we can't really hope for identical numbers. We would also like to point out that the off-line database did not allow for multi-blob patterns. As such, the two stroke digit “5” is not present.

We have the stroke counts for the skeletons, as they are provided by the thinning algorithm. Table 3 gives the stroke count before the stroke reconnecting algorithm, and Table 4 gives the stroke count after stroke reconnection.

Char Id	Average Removed	Percentage Removed
0	0.175	13.67
1	0.015	1.456
2	0.19	8.225
3	0.38	8.225
4	2.26	48.23
5	0.565	27.36
6	1.15	49.04
7	0.305	18.89
8	2.475	67.90
9	0.93	39.49
Average	0.8445	28.25

Table 5: Strokes removed by reconnecting for each class of pattern

To summarize the two tables, the average number of strokes per character, before stroke merging, is 2.60. After the reconnection this average goes down to 1.75. So 0.85 strokes, on average, are removed by the process, or about 35%.

Table 5 gives the breakdown for each of the pattern classes. It should be noted that the algorithm has almost no effect on some classes. The "0" and the "1" were often optimal from the start. As we can see, the algorithm successfully removes strokes in more complex patterns where there are stroke crossings. It is less successful with patterns "2", "3", "5" and "7".

2.3.8 Reconnecting, Qualitative Experiment

We note that new optimal characters are easily obtained with patterns "4", "6", "8" and "9". Unfortunately for the other patterns, although there is significant stroke reduction, we do not achieve what we consider an optimal pattern.

To observe this phenomenon, we created a second experiment. In this experiment we decided to observe the stroke reconnection process. We created a program that will allow us to visualize each of the stroke reconnections as they were performed. The experiment consisted of a program where six windows were opened on a graphical workstation, using basic libraries from X11 [39]. At the time of writing, the X11 functions used permitted running the program on either Unix based machines (Suns or DEC Ultrix) or on VAX/VMS machines without any loss of the graphical interface, and without the need for recoding the application.

Figure 21 shows an overview of the entire tool. Each of the windows represents the pattern at a different stage of merging. The bottom right window represents the final result; the left top window represents the original pattern after the first stage of preprocessing. The "`Raw Pattern`" window shows the unprocessed original pattern, with each stroke separate. Singular points are marked with large squares. As well, to denote the stroke order, the first point of a stroke following the singular point is printed with a small rectangle. Each subsequent point is drawn with a progressively larger rectangle. Due to size limitations, after a while, the cycle restarts with a small rectangle. Varying rectangle size allows us to determine the location of the beginning and the end of the strokes. You will note that, in this example, the pattern is completely reconnected.

With this tool, we can determine which strokes were merged together, as well those that were left unmerged. Each of the two thousand patterns of training set A that were skeletonized with the algorithm provided by Louisa Lam, were inspected. At inspection we determined to which category the pattern belonged. Below is a list all the categories used, as well as a brief description.

- **WasOk:** The stroke count was already optimal for the given pattern.
- **Optimal:** After the stroke reconnecting, the stroke count became optimal for the given pattern class.
- **Retrace:** Reconnecting reduced the number of strokes, but retraced strokes prevented us from achieving an optimal stroke count.
- **Ok Noise:** Stroke count is not optimal, primarily due to spurious noise strokes, but merges were performed correctly.
- **ErrNoise:** Noise caused an error in reconnection.
- **Orientation:** Strokes were not reconnected because they exceeded good continuity criterion.

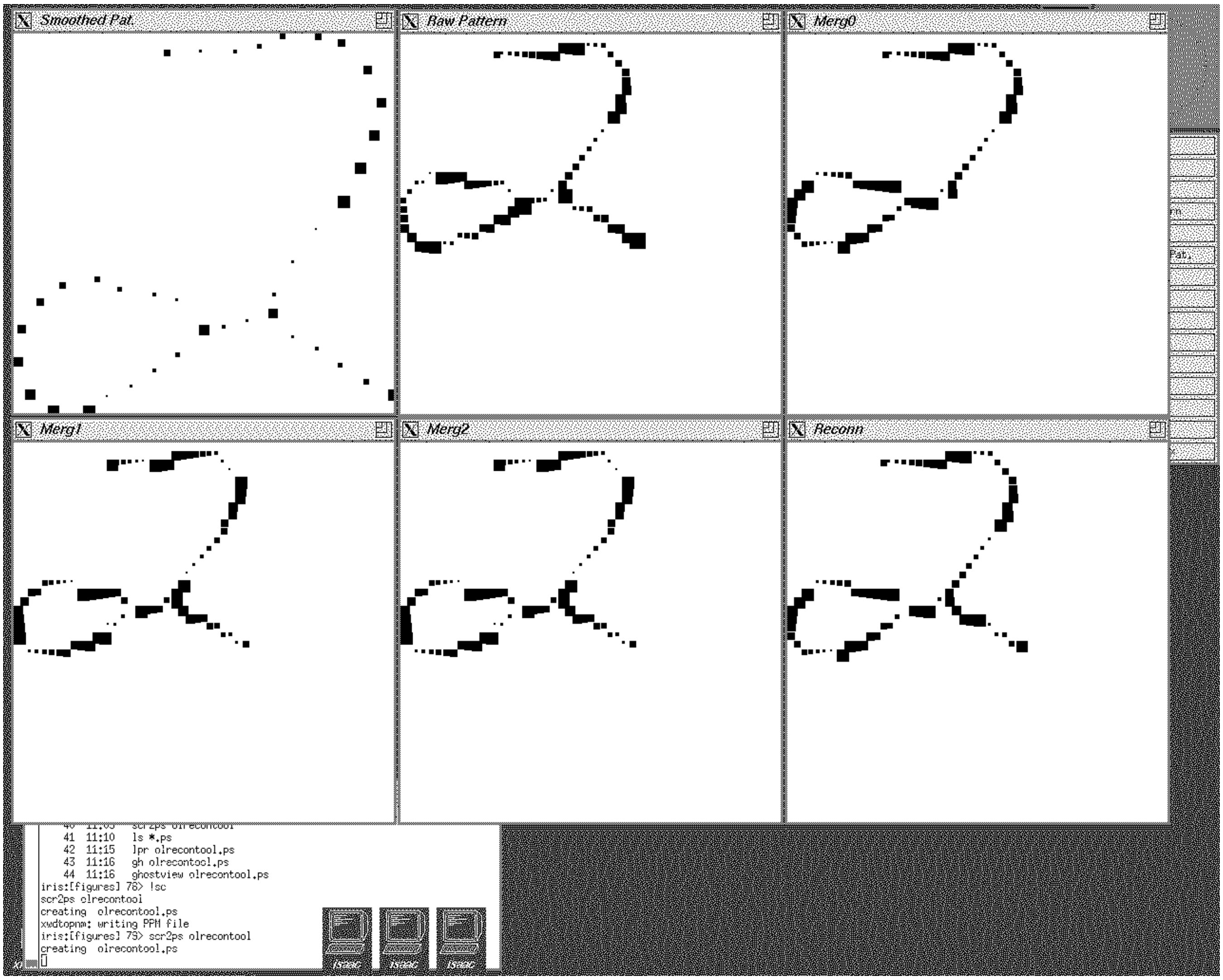

Figure 21: Reconnection process experiment tool

Char id	WasOk	Optimal	Retrace	OkNoise	ErrNoise	Orientation
0	162	19	0	18	1	0
1	197	0	0	3	0	0
2	22	73	74	30	0	1
3	46	3	111	0	11	29
4	0	169	24	0	7	0
5	111	6	67	3	6	7
6	7	156	23	1	11	2
7	143	1	46	5	5	0
8	0	174	10	11	2	3
9	22	99	67	8	3	1

Table 6: Qualitative reconnecting results. Per pattern class

A pattern is deemed "optimal" if the number of strokes is identical to the number of pen strokes used, usually one. The number four is written with at least two pen strokes, one long vertical bar as well as a second stroke with a ninety degree corner attached to the first one. We must also allow, in the case of the number four, for an optimal pattern with three strokes.

It turns out that some of the fours bear a close resemblance to the letter "H". They are both composed of two long vertical bars linked together by a short horizontal bar. The resemblance of these samples to the letter "H" leads us to believe that trying to find a retrace in these cases was probably counterproductive. Often, both branches are of relatively equal length, and their slope difference is small. Since the letter "H" is frequently written with three strokes, it was felt that forcing a retrace interpretation on these strokes would probably hinder the generality of the algorithm, with little benefit.

Table 6 shows the results of the retrace experiment. The first observation we can make is that patterns that have a tendency to form optimal merges have crossing strokes. These features are prominent with the patterns "8", "6" and "9"; these strokes will, most of the time, meet the merging criterion.

In the case of the "3", "2", "5" and "7", there is an abrupt directional change that is, most often, not marked by a loop. This sharp change in direction is represented by a single straight line that is often of varying thickness. This is better conceptualized

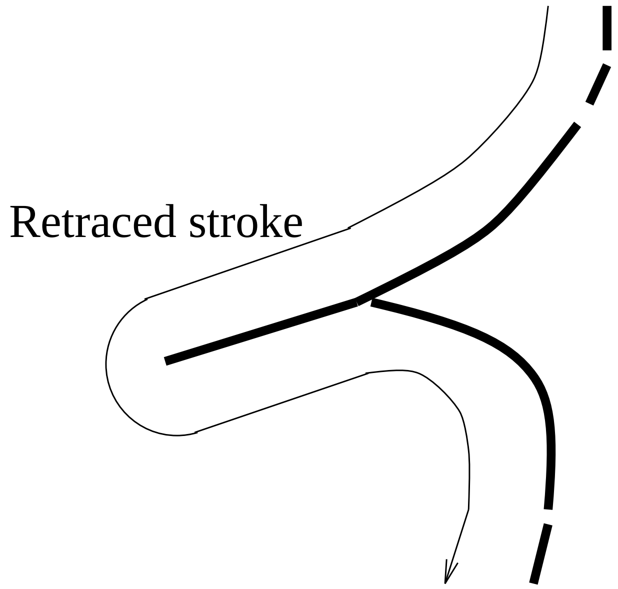

Figure 22: Pen retrace as often found (e.g. character 3)

by a retrace. A retrace is a case where there must have been two different pen movements. The pen moved from one end of the stroke to the other and back. Inspection of the original images reveals that often, but not necessarily always, these retraced strokes are represented by wider pen marks.

The retrace is quite often present in the case of the middle bar of the three (Figure 22), the bottom line of the "2", the "5" (the tail end at the top of the loop) and "9" (the upper portion of the straight bar). It should be noted that the main characteristics of the retrace are that it is short in length, attached to two other strokes by the same junction points, and is unattached at the other end.

Retraces represent the majority of the cases where the result is not what we consider optimal. Basically, each of these characters has a portion that is wedged in between two larger parts. Reversing this retrace stroke and using it as a link between the other two strokes will yield an optimal pattern. Retracing is a problem also expressed by Govindaraju [43]. In their case they provide an alternate interpretation:

> *The branch that is retraced is the one which makes the maximum angles with the other two branches.*

Unfortunately, this does not concur with our results. The retrace stroke will often merge with one of its siblings. This implies that the angle made by the retraced stroke and its neighbours is not maximal but low enough to meet the merge criteria. Because Govindaraju was dealing with cursive writing, the needed heuristic rules are probably different.

We are currently not processing the retrace and consider this acceptable as it will typically merge consistently. Although this impairs generalisation, it represents an improvement over the non-merged patterns. We outline here the solution that we believe would be correct, derived after observation of the experimental data, to process retraces, but that we have not implemented. From our observation of the data, a retrace stroke, is the shortest of the three strokes attached at a given junction and terminated by an end point. Once identified, a retraced stroke is created by the concatenation of the stroke with a copy of itself reversed. Using this retraced stroke, we concatenate it with the two other strokes involved, as the middle stroke.

We were surprised to see how many optimal merges are obtainable from such simple rules. Even in cases where optimal merges are not obtained, the algorithm merges the strokes in a consistent manner. Finally, the stroke orientation rule we used is simple enough to be accurate most of the time. This is important, as reversed strokes create a new variant of the patterns, when there is actually little change in the represented image.

2.3.9 Reconnection Experiment Observations

The definition of the good continuity criterion we use is very simple. It does not take into account the curvature of the strokes involved. As a result, some of the merges that look like they should have been made were ignored. This is most often the case where the strokes involved have sharp curves that make the angle difference between the two strokes too high to meet the good continuity criterion. A better good continuity criterion would probably increase the reliability of the merges.

Good Continuity is Sufficient

The Gestalt good continuity rule has been shown to extend to more complex pictures, such as groupings of objects. Objects placed in a straight line tend to merge together [7]. This suggests that, at lower levels of perception, even for complex shapes, the tendency to observe the Prängnanz law is strong. We take this to mean that using the good continuity rule is a solid base for stroke reconnection. It is also to be noted that, when object groupings are performed, stroke crossing has less impact than the orientation or the shape [9]. As such, crossing is not strictly the only grouping mechanism involved for strokes, as demonstrated by psychological studies. However, the alphabet has managed to avoid the potential confusion introduced by shape orientation by a good selection of the shapes for the letters of the alphabet. As a result, stroke reconnection is always a preferred interpretation and makes the use of the good continuity rule the best rule of interpretation in our case.

Recognizing early on that the good continuity rule was not a foolproof algorithm for stroke merging, the original algorithm for merging the strokes was, instead, based on the recognition algorithm itself. We would attempt elastic matching for each candidate stroke from the unknown with the beginning of the model, keeping the best match, then merging the remaining strokes, to yield the lowest elastic matching distance with the remainder of the model. This would represent a form of interaction between stroke merging and recognition, similar to the interaction that occurs between digit segmenter and recognition algorithms [93]. It was not implemented to simplify preprocessing.

2.4 Preprocessing Performed

Preprocessing is divided into two subsystems, a generic preprocessing step and an image processing step which is applied before generic preprocessing to off-line data. The generic preprocessing is relatively simple, but may not be optimal, especially in preserving sharp angles. It is, however, in line with steps performed by similar systems.

The images of the off-line data need to be preprocessed to generate sequences.

These sequences, already in a format identical to on-line data, need further processing to reconnect the strokes broken by the junction points. This stroke reconnection attempts to rebuild the writing sequence from a static image. This allows us to perform recognition using an on-line recognition algorithm on both types of data.

Chapter 3

Elastic Matching

Elastic matching is a process that is related to string matching. In both cases the basic problem is to measure the difference between two sequences of items and decide whether these differences should be considered large or small. In an elastic matching pattern recognition system, the objective is to calculate the distance between unknowns and models. An unknown is *recognized* when the distance is deemed low enough. There are two primary functions involved in the elastic matching process:

- **String Matching:** A process by which two sequences are compared together.
- **Distance function:** The measure of distance between two different items of given sequences. Used by the above string matching process.

We will first look at the original elastic matching function. We will then look at an iterative version of elastic matching. Although this iterative elastic matching is not identical to the original function, we will show that it exhibits a similar behaviour. We will then look at removing the constraint in the depth of the look-ahead and observe the resulting behaviour. In the next chapter we will look at the training process before looking at a weighted elastic matching function. Weighted elastic matching, based on iterative elastic matching, will then be used for recognition experiments.

3.1 String Matching

String matching is a process by which one attempts to match one sequence of elements against another. The matching algorithm will attempt to evaluate the differences between two similar sequences. In order to evaluate the degree of similarity between two sequences one must qualify the nature of the differences between items of a sequence by associating a cost function to the differences. The nature of the cost function is dependent on the application domain. The various algorithms used for string matching are described by Krushkal [58]. We will briefly describe here the salient points involved. Three operations are allowed in string matching, and with each type of operation there is an associated cost.

- **Substitution:** Replacement of one component with another.
- **Insertion:** Adding a new component into the middle of the string to reduce the distance.
- **Deletion:** Removing an extraneous component.

Matching a string involves applying each of the possible operations, with its associated cost component, to obtain the smallest distance as measured by the sum of the costs incurred by the application of each operation.

3.2 Elastic Matching

The application of the string matching theory in character recognition is known as *elastic matching*. It is an application of the Dynamic Programming algorithms used in string matching, where the sequences of points are considered as strings, and the purpose is to measure the distance between an unknown string and a set of reference strings.

3.2.1 String Matching Operations

The notion of substitution is an important one in string matching as it allows inexact matches. Of course substitutions can be expressed in terms of insertions and deletions. Conversely, insertions and deletions can be expressed in terms of substitutions where the substitution occurs with a NULL element.

The distinction between the insertion, deletion and substitution operations is important in string matching because there may be a different cost associated with each of the different operations. This will influence the sequence of operations used by the algorithm.

However with elastic matching, both insertion and deletion are considered equivalent to substitution. This is possible because the only cost involved with the operation comes from the point distance function. Still, insertion deletion and substitution occur in this system. In elastic matching, every time a look-ahead greater than one is used, a deletion operation occurs because one or more points will not be compared. They are ignored for the rest of the process. This is, essentially, the definition of the deletion operation.

When a look-ahead of zero is used, an insertion is done. Because the same point is used twice, it is equivalent to inserting a copy of that point in the string. This is the form that insertion takes in this context. Because we almost never obtain an exact match between points, we practically always perform a substitution. The cost associated with a substitution is simply the distance between the two points, as evaluated by the distance function.

3.2.2 Elastic Matching Recognition

In elastic matching, the algorithm matches an unknown against a known pattern stored in a database that we call a model base. As in string matching, elastic matching attempts to minimize the distance between the unknown and the given model. Here the associated cost function is a Euclidian distance between two points, one from the model and one from the unknown. The definition given by Formula 9 also ensures that all of the points of the unknown are compared against a point of the model. This

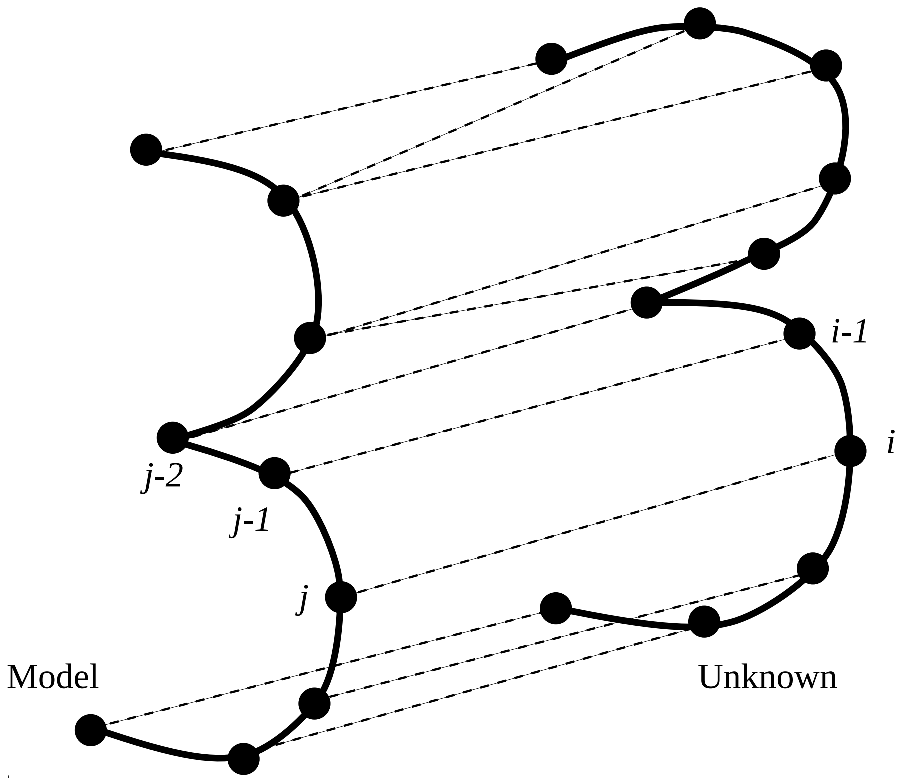

Figure 23: Point to point distance measurement as performed by elastic matching

is illustrated in Figure 23.

Elastic matching is expressed mathematically as a minimization problem. The total distance between two patterns is expressed as $D(n, m; k)$ for a unknown of n points against the model k of m points. The distance between individual point i of the unknown against point j of the model $k$is expressed as $d(i, j; k)$ and sometimes abreviated $d(i, j)$. The goal is to minimize $D(n, m; k)$ where the distance D is defined by Formula 9.

$$D(i,j;k) = d(i,j;k) + \left\{ \begin{array}{ll} min \left\{ \begin{array}{l} D(i-1,j;k) \\ D(i-1,j-1;k) \\ D(i-1,j-2;k) \end{array} \right\} & j > 2 \\ min \left\{ \begin{array}{l} D(i-1,j;k) \\ D(i-1,j-1;k) \end{array} \right\} & j = 2 \\ min \left\{ \begin{array}{l} D(i-1,j;k) \end{array} \right\} & j = 1 \end{array} \right\} \tag{9}$$

The actual distance used in recognition is an average distance per point. Normalizing of the number of points is necessary to meaningfully compare patterns of different lengths.

$$D(i,j;k) = \frac{D(n,m,k)}{n} \tag{10}$$

For a given unknown of n points matched against a model k of m points.

Point Distance Function

The algorithm is centred around the distance between two points $d(i,j;k)$. The first point distance function used consisted strictly of a geometric distance, as given by Equation 11.

$$d(i,j;k) = (x_i - x_{j,k})^2 + (y_i - y_{j,k})^2 \tag{11}$$

The distance given by Formula 11 is not, strictly speaking, a Euclidian distance but the square of the Euclidian distance. It is, however, a one-to-one mapping but Formula 11 will increase more rapidly than the normal Euclidian distance.

We are not using the Euclidian distance primarily to save computations. Evaluating the square root would increase computation time, but would not improve the distance measurement function significantly. Empirical measurements provided by *prof*, the UNIX system profiling utilities, suggest that the entire recognition program can spend anywhere between 60% and 80% of its time evaluating the distance function. We believe that losing the linearity of the Euclidian distance function is acceptable, considering the serious impact in the reduction of computation time. It is also in line with previous work by Tappert [103].

Function 11 only uses the distance between two points; it fails to consider the orientation and curvature of the stroke. This information is very often used by other recognition systems. Therefore, the point distance function used with elastic matching adds a third term, measuring the slope-angle difference. Equation 12 adds this term and is the one reported in the literature [103].

$$d(i,j;k) = (x_i - x_{j,k})^2 + (y_i - y_{j,k})^2 + c\,|\phi_i - \phi_{j,k}| \tag{12}$$

Here, c is an empirically determined constant and ϕ represents the angle of elevation of the tangent to the point.

$$\begin{array}{ll} \phi_i = \arccos(\frac{x_{i+1}-x_i}{hyp}) & y_{i+1} < y_i \\ \phi_i = \arccos(\frac{x_{i+1}-x_i}{hyp}) + \pi & y_{i+1} >= y_i \\ \phi_i = \phi_{i-1} & i = n \end{array} \tag{13}$$

where

$$hyp = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{14}$$

In our case we modified this formula to add a fourth term representing the curvature of the sequence of points.

$$d(i,j;k) = (x_i - x_{j,k})^2 + (y_i - y_{j,k})^2 + c\,|\phi_i - \phi_{j,k}| + b\,|\chi_i - \chi_{j,k}| \tag{15}$$

In Equation 15, both b and c are empirically determined constants. They are used because the angles described by ϕ and χ are expressed in radians in a well defined range of values, between 0 and π. This is a small range compared to the squared Euclidian distances that can be anywhere from 0 to the tens of thousands. These constants are therefore needed to increase the importance of ϕ and χ, if they are to make any contribution.

These constants also play a role of weights. Changing the value of either constant changes the contribution of the corresponding feature to the total distance. The values we used were empirically determined. We are using values that, for a given set of data, maximized the recognition rate.

The measurement of the difference in the slope, denoted by ϕ, is given as the difference in orientation between the points from the unknown and the model. The measure of difference of curvature, denoted by χ, determines if the segments are curved by the same amount. This measure is rotation independent and can help in denoting features that are rotation independent.

$$\begin{array}{ll} \chi_i = \tau & i < n \wedge \tau < \pi \\ \chi_i = -\tau & i < n \wedge \tau >= \pi \\ \chi_i = \chi_{i-1} & i = n \end{array} \tag{16}$$

where

$$\tau_i = \phi_i - \phi_{i+1} \tag{17}$$

3.2.3 Iterative Elastic Matching

Elastic matching, as described by Equation 9 is computationally intensive, especially if the recursion is carried out directly. We devised an alternate iterative formula $DI(i,j;k)$ that is not recursive and that can be shown to be nearly identical to the recursive description. The main advantage, for our purposes, is the reduction in computational complexity.

$$DI(i,j;k) = DI(i-1,j;k) + \left\{ \begin{array}{ll} min \left\{ \begin{array}{l} d(i,j;k) \\ d(i,j+1;k) \\ d(i,j+2;k) \end{array} \right\} & j < m-1 \\ min \left\{ \begin{array}{l} d(i,j;k) \\ d(I,j+1;k) \end{array} \right\} & j = m-1 \\ min \left\{ \; d(i,j;k) \; \right\} & j = m \end{array} \right\} \tag{18}$$

where $DI(1,1;k) = d(1,1)$

For a model k with m points Equation 18 is evaluated, iteratively for $D(1,1;k)$ up to $D(n,m;k)$ where we then obtain the distance of the unknown with model k. Despite the similarities between Formula 18 and Equation 9, they are not identical. For brevity we will omit the constant term k in the following equations.

$$\begin{aligned} DI(i,j) &= DI(i-1,j') + \min\{d(i,j'), d(i,j'+1), d(i,j'+2)\} \\ & DI(i-1,j') + d(i,j) \\ & DI(i-2,j'') + \min\left\{\begin{array}{l} d(i-1,j'') \\ d(i-1,j''+1) \\ d(i-1,j''+2) \end{array}\right\} + d(i,j) \\ &= d(i,j) + d(i-1,j') + DI(i-2,j'') \end{aligned} \tag{19}$$

where

$$d(i,j) = \min\{d(i,j'), d(i,j'-1), d(i,j'-2)\} \tag{20}$$

In effect, j' is the value at which the minimum value of the function $d()$ can be found. The same remark applies for j'' with respect to j'. We can express j' to be the value that the second parameter that the function $d(i,j)$ needs to have at the previous iteration of the algorithm to obtain the current value of j. Because of the substitution performed previously, Equation 18 can alternatively be expressed as follows:

$$DI(i+1,j) = DI(i,j) + d(i+1,j') \tag{21}$$

where $d(i+1,j') = \min\{d(i+1,j), d(i+1,j+1), d(i+1,j+2)\}$

We can perform the same manipulation to Equation 9, which gives the following:

$$\begin{aligned} D(i,j) &= d(i,j) + \min\{D(i-1,j), D(i-1,j-1), D(i-1,j-1)\} \qquad (22) \\ & d(i,j) + D(i-1,j') \\ & d(i,j) + d(i-1,j') + \min\left\{\begin{array}{l} D(i-2,j') \\ D(i-2,j'-1) \\ D(i-2,j'-2) \end{array}\right\} \\ & d(i,j) + d(i-1,j') + D(i-2,j'') \qquad (23) \end{aligned}$$

where the j' substitution applied to function D takes on the same meaning as used in Equation 19, and is described as:

$$D(i,j) = \min\{D(i,j'), D(i,j'-1), D(i,j'-2)\} \tag{24}$$

Although this substitution expresses the same concept, it does not guarantee that it is valid for both functions $D(i,j)$ and $d(i,j)$. However we need it to be valid if we want to have Equations 9 and 19 to be equal. It can easily be seen that for the special cases of $D(1,1), D(2,2), D(3,3)$ the equations are equal. But for the general case of $D(n,m)$ such a demonstration is not trivial.

We are attempting to demonstrate this relation with patterns similar to each other. If the patterns are similar, their distances should be low. In such cases, we expect each point to be close to its corresponding point in the model. A pattern being compared with itself, a copy of itself shifted or modified by noise, or with a very similar pattern are examples of this. For instance, when comparing a pattern with itself, the result of $\min\{d(i,j), d(i,j-1), d(i,j-2)\}$ will always result in $i=j$ since $d(i,i)=0$ by definition.

We will show that the following also holds:

$$D(i,i) \leq \{D(i,i+2), D(i,i+1), D(i,i-1), D(i,i-2)\} \tag{25}$$

when we are assuming that Equation 26 holds true for the patterns at hand:

$$d(i,i) < \{d(i,i+2), d(i,i+1), d(i,i-1), d(i,i-2)\} \tag{26}$$

Then Equation 27 also holds true. In the case of $D(1,1)$ by definition since $D(1,1) = d(1,1)$ and:

$$\begin{aligned} D(1,2) &= d(1,2) + \min\{d(1,1)\} \\ & \quad d(1,2) + d(1,1) \end{aligned} \tag{27}$$

from 26 and 27 we can infer that $DI(1,1) < DI(1,2)$. A similar argument can be made to show $D(1,1) < D(1,3)$. Hence 25 holds true for the case $D(1,1)$.

Assuming that 25 holds true for an arbitrary $D(i-1,i-1)$ and $D(i-2,i-2)$, show that it remains true for $D(i,i)$.

$$\begin{aligned}
D(i,i) &= d(i,i) + \min\left\{\begin{array}{l} D(i-1,i) \\ D(i-1,i-1) \\ D(i-1,i-2) \end{array}\right\} \\
&= d(i,i) + D(i-1,i-1) && (28) \\
D(i,i+1) &= d(i,i+1) + \min\left\{\begin{array}{l} D(i-1,i+1) \\ D(i-1,i) \\ D(i-1,i-1) \end{array}\right\} \\
&= d(i,i+1) + D(i-1,i-1) && (29) \\
D(i,i+2) &= d(i,i+2) + \min\left\{\begin{array}{l} D(i-1,i+2) \\ D(i-1,i+1) \\ D(i-1,i) \end{array}\right\} && (30) \\
D(i,i-1) &= d(i,i-1) + \min\left\{\begin{array}{l} D(i-1,i-1) \\ D(i-1,i-2) \\ D(i-1,i-3) \end{array}\right\} \\
&= d(i,i-1) + D(i-1,i-1) && (31) \\
D(i,i-2) &= d(i,i-2) + \min\left\{\begin{array}{l} D(i-1,i-2) \\ D(i-1,i-3) \\ D(i-1,i-4) \end{array}\right\} && (32) \\
& && (33)
\end{aligned}$$

From Equations 27 and 26 we see that the minimum sum is $d(i,i)+D(i-1,i-1)$.

We will show that, in such a case, the substitution of Equation 20 by Equation 24 will yield the same j for a given j'. We will show that 24 can be reduced to minimize the same function. The terms within the *min* function of Equation 24 can be expanded as follows:

$$D(i-1,i) = d(i-1,i) + \min\left\{\begin{array}{l} D(i-2,i) \\ D(i-2,i-1) \\ D(i-2,i-2) \end{array}\right\} \qquad (34)$$

$$D(i-1,i-1) = d(i-1,i-1) + \min\left\{\begin{array}{l} D(i-2,i-1) \\ D(i-2,i-2) \\ D(i-2,i-3) \end{array}\right\} \quad (35)$$

$$D(i-1,i-2) = d(i-1,i-2) + \min\left\{\begin{array}{l} D(i-2,i-2) \\ D(i-2,i-3) \\ D(i-2,i-4) \end{array}\right\} \quad (36)$$

From our assumption we can simplify the *min* functions in 34, 35, 36 and rewrite them as follows:

$$\begin{aligned} D(i-1,i) &= d(i-1,i) + D(i-2,i-2) && (37) \\ D(i-1,i-1) &= d(i-1,i-1) + D(i-2,i-2) && (38) \\ D(i-1,i-2) &= d(i-1,i-2) + D(i-2,i-2) && (39) \\ & && (40) \end{aligned}$$

We can then rewrite the minimization part of 24 as follows:

$$\begin{aligned} D(i,i) &= d(i,i) + D(i-2,i-2) + \min\left\{\begin{array}{l} d(i-1,i) \\ d(i-1,i-1) \\ d(i-1,i-2) \end{array}\right\} && (41) \\ &= d(i-1,i-1) + D(i-2,i-2) && (42) \\ & && (43) \end{aligned}$$

This equality establishes that both minimization functions yield the same results, in terms of which j is selected for a given j'. The substitution of j for j' will yield the same result with both functions $d(i,j)$ and $D(i,j)$.

We now want to show that $D(i,j) = DI(i,j)$ holds true. It trivially holds true for $D(1,1) = DI(1,1)$. Assume that it is true for an arbitrary $D(i-1,i-1) = DI(i-1,i-1)$ and $D(i-2,i-2) = DI(i-2,i-2)$, then substituting $DI()$ with $D()$ in equation 19 gives:

$$DI(i,j) = d(i,j) + d(i-1,j') + DI(i-2,j'') \quad (44)$$

$$
\begin{aligned}
& d(i,j) + d(i-1,j') + D(i-2,j'') && (45) \\
= \quad & D(i,j) && (46)
\end{aligned}
$$

We have shown that, when comparing similar patterns, both equations will yield the same results. But patterns are not always that similar. The two formulae may yield different results, when the patterns are far apart. When this is the case, the resulting distance will be high enough, with either formula, to exceed the recognition threshold. It is then important to notice that in the cases where both equations are likely to differ, the results do not influence recognition. Therefore, the difference, relatively small as it will be, will not play a role, since the recognizer will discard these values as too high. Although the constraints we placed on the patterns being compared are restrictive in this demonstration, experimentation shows that for recognition purposes $D(i,j)$ and $DI(i,j)$ can be considered equivalent for our purpose.

3.3 Unconstrained Elastic Matching

As the reader may have noticed, a characteristic of elastic matching is the limit on the number of points one can look ahead. The look-ahead is the number of points we can look at to choose which point has the minimum distance $d(i,j;k)$. At first glance, this fixed limit appears to be arbitrary, and the question then arises if recognition would improve with a deeper look-ahead.

To this end, we wrote a variant of our algorithm that allows an arbitrary depth look-ahead. This allows us to observe the behaviour of the recognizer and see if there is any chance of improving the results by changing the look-ahead. Furthermore, we will see where the natural tendencies of the algorithm are.

The role of the depth search is to allow us to better adapt to the variability we encounter. If a point is not an exact match with the corresponding point in the model, it may make a better match with its neighbour. This allows for slight variations in the pattern. Because our main goal is to improve the generalizing characteristics of the algorithm, there is reason to believe that allowing for deeper searches may well improve the distance measurement. By allowing for deeper searches, slight errors will be corrected sooner, because areas causing trouble will get skipped over more rapidly. Deeper look-aheads will mean that fewer points of the unknown will be matched with far points.

3.3.1 Unconstrained Depth

In this version of the algorithm, the equation is altered to allow for arbitrary depth look-ahead. In Equation 47, n is the depth of search we will allow to occur. Note that the boundary condition is slightly more complex. We will define the look-ahead n as the number of points, behind the current point on the model, we are allowed to look at. In elastic matching the look-ahead is set at two, while in linear matching the look ahead is set at zero.

$$\begin{aligned} DN(i,j;k,n) &= d(i,j;k) + min \left\{ \begin{array}{l} DN(i-1,j;k,n) \\ DN(i-1,j-1;k,n) \\ \vdots \\ DN(i-1,j-n;k,n) \end{array} \right\} \quad i > n \\ &= d(i,j;k) + min \left\{ \begin{array}{l} DN(i-1,j;k,n) \\ DN(i-1,j-1;k,n) \\ \vdots \\ DN(i-1,0;k,n) \end{array} \right\} \quad i <= n \end{aligned} \tag{47}$$

For this set of experiments we will set a given look-ahead and attempt recognition. The system parameters, including recognition thresholds, remain constant. This, in part, puts a bias toward the look-ahead value with which the recognition thresholds were found. Alternatively, we can adapt the recognition thresholds for each of the specific look-ahead values used. This removes the bias present in the previous experiment. We tried both experiments, with the same results. We are trying to answer the following questions:

- What is the role of the limited look-ahead?
- Can a different look-ahead improve the results?

3.3.2 Function of the Look-ahead

The role of the look-ahead is twofold:

- Replaces the cost component in the substitution operation.
- Allows variability in the models.

The original string matching algorithm, on which elastic matching is based, has a cost component for each operation. In our case, the cost component is strictly based on the distance measurement provided. Each point within the look-ahead window incurs no cost for being further away from the current point in the model. For this reason, the insertion and deletion operators, as we have described them above, are degenerate cases of substitution.

This is the first conclusion that we can draw about a variable look-ahead. Because there is no cost involved, deletion and insertion operations tend to be overused. They do result in cheaper matches, and the total distance is lower, but we have also ignored significant components of the models.

3.3.3 Depth Histogram

The depth histogram is an attempt at visualizing the behaviour of the algorithm in terms of the points it will match. More precisely, it displays the number of times a given look-ahead is used under the same conditions for a given match.

For each graph we pick two patterns. We then apply the elastic matching algorithm, using the unconstrained version of the algorithm. We set a fixed look-ahead that can be used by the algorithm. We then record the number of times each look-ahead is used. That is to say, we record the number of points skipped each time we made a distance measurement. We perform the match described above, with all the possible look-aheads within a fixed range, starting with a maximum allowed look-ahead of one, up to the number of points in the model. Note that we can't use a look-ahead bigger than the number of points in the model, since that is the maximum number of points we can match or can skip.

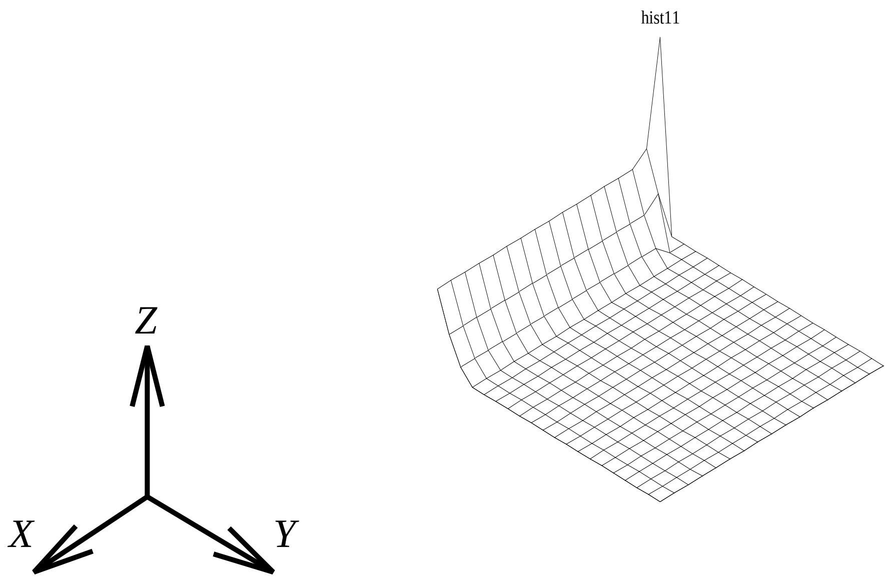

Figure 24: Depth Histogram for two closely matched patterns.

We do not display look-aheads of zero so we do not count the number of times a point, in a model, has been matched more than once with points of a given unknown. If some sequences of points of an unknown are all matched against just one point in the model, just one point will be recorded. It then appears as if fewer points were matched, but this is not so. Fewer points of the model are matched, but some of these points are matched more than once. Still, all the points of the unknown are matched against a point of the model.

This depth histogram represents how likely the algorithm is to use a large look-ahead, when the opportunity is provided. This, at first, may not sound too meaningful. However, remember that we are supposed to compare similar strings. The position of a point inside the string is as meaningful as its position in Euclidian space.

We expect that a point in the middle of an unknown should be close to a point in the middle of the model. This is, in fact, what happens when a pattern is compared with itself. Each point will match itself exactly. We would then expect a similar behaviour, when comparing an unknown to a model fairly similar to it. This graph attempts to illustrate how close the algorithm comes to this ideal model.

Figure 24 shows, in the Z-axis, the number of times a given depth has been used.

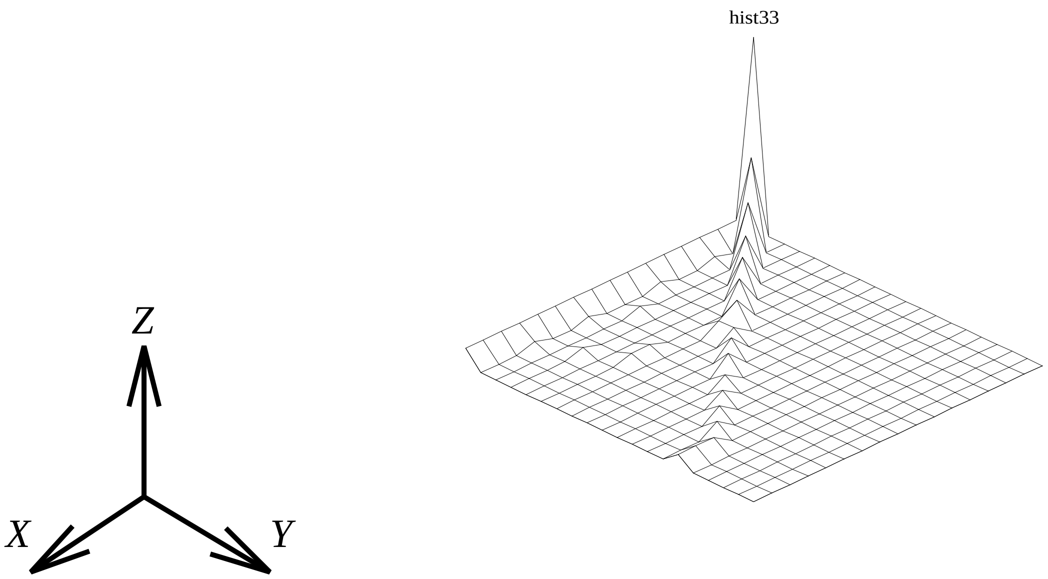

Figure 25: Depth Histogram for far patterns from the same class(mistracking).

The greater the height of the point, the more times this particular look-ahead has been used. The X-axis shows the number of times a look-ahead is used when up to a given look-ahead is allowed. Since there are the same number of points in the unknown, the sum of all the points at a given value on the X-axis for a given Y should be a constant. However, remember that we do not count zero look-aheads. They increase when bigger look-aheads are used to compensate for the large jumps. Hence, the observed apparent loss of points. The Y-axis represents all of the look-aheads possible during the experiment. Note that it is constrained by the number of look-aheads allowed on the X-axis. This means that only the upper triangle part of the matrix can have values different from zero.

We can observe from these graphs that the algorithm will tend to use as large a look-ahead as it can. This occurs primarily when the pattern is from another class. Consequently, it matches fewer and fewer points in the model with each matched model point getting matched more often. Still, the total distance between two models is reducing and, as such, the minimization aspect of the algorithm behaves as expected.

But the elastic matching algorithm does not behave as we would have hoped.

Instead of closely tracking each point one by one, it races to the end of the model. This racing to the end of the model is what we can call *mistracking*. This notion is hard to define, as the difference between mistracking and error correction is hard to express. Mistracking is the tendency to match two points that differ much in distance, in terms the number of points, to the origin of the stroke.

Experimentation shows that mistracking occurs primarily among distances measured between patterns of different classes. Thus, by allowing for a deeper look-ahead, the algorithm will minimize the distances by skipping many of the points that are distant. This does lower the total distance, but this lowering does not indicate an improvement in the generalizing of the matching algorithm; it rather signifies that the algorithm finds itself trying to minimize the distance by avoiding using penalizing points. But it can be these exact points that differentiate between two classes of patterns.

An important observation that can be made is that the racing tendency observed previously is not universal. In fact, when we match models that are relatively close to each other, say from the same class, this behaviour will not occur. This is well illustrated by Figure 24, while racing can be observed in Figure 25.

We can see, in Figure 24, that after a depth of three the look-ahead stabilizes. Even if we allow the algorithm to use a larger look-ahead, it will keep the same solution. It deems this solution to give the minimum distance, i.e. the best match. We observe the behaviour we had hoped to see at the beginning of this experiment: the points are matching their neighbours from the model.

One would expect this behaviour to be displayed for most cases of patterns of the same class being matched with each other, simply because the structure of the patterns are fairly similar. However, some cases are not this clear. In certain cases, the behaviour of the patterns is as bad as if they were from unrelated classes such as in Figure 25.

3.3.4 Distance Surface

We have seen in the previous section that increasing the depth at which the algorithm looks ahead leads to a decrease in performance instead of an improvement. To explain

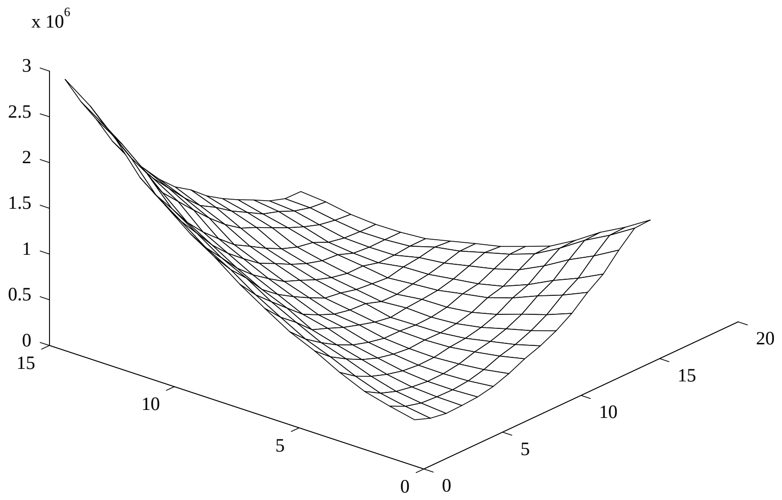

Figure 26: Distance surface for the case of a 1 measured against a 1

the phenomenon we will to look at the distance surface.

The distance surface graph shows for given i, j, the distance $d(i, j)$ between point i of pattern A and the point j of model B. This mesh represents all the possible points the elastic matching algorithm can take to solve the problem.

Figure 26 is the simplest example of such a distance surface graph. In this figure, two patterns representing the digit "1" are used. Both patterns were taken at random. Each point on the graph represents a distance between two points of both patterns.

What is the best shape to encounter in a distance surface graph? We need to answer this question to interpret the graphs. The algorithm attempts to cross between the plane from $(0, 0)$ to (n, m) by the cheapest path. Because the only cost involved in taking a path is the distance between the points, we are essentially looking for the deepest valley on the surface between $(0, 0)$ and (n, m). We can consider the algorithm as a ball trying to roll between these two points that, because of the law of gravity, always seeks the lowest elevation.

When matching patterns of the same class, we expect to find the valley stretching

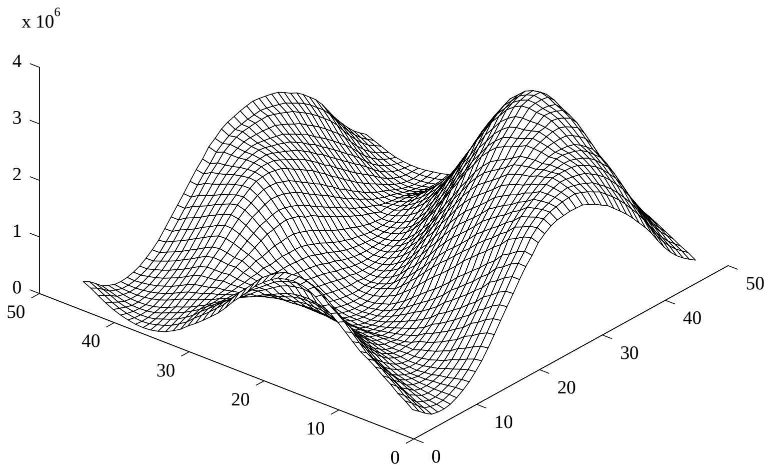

Figure 27: Distance surface for a 0 measured against a 0

from $(0,0)$ to (n,m). Figure 27 shows a case where the valley is not as smooth as in Figure 26. However, we can see that there is a valley extending on the diagonal between $(0,0)$ and (n,m) and that is the lowest region on the surface.

In Figure 28 we have a more complex surface. This is to be expected with an "8", since the strokes cross each other in both patterns, and there is at least a chance that the junction point has a low distance, with at least two points in the stroke. The first point would be the corresponding point in the other pattern and the second point would be the point in the model where the stroke comes back to cross itself. Because the patterns matched were taken at random there is no way of knowing if this is the best surface that can be made. We can still see a valley between $(0,0)$ and (n,m), which is the path we expect the elastic matching algorithm to take.

In Figure 29, where we compare a "5" against an "8", we don't have a nice valley from one end of the surface to the other. The important thing to notice is that the valleys formed do not run in a straight line from the start and end points of the model. Furthermore, distance maps were performed, and they confirmed that a nice

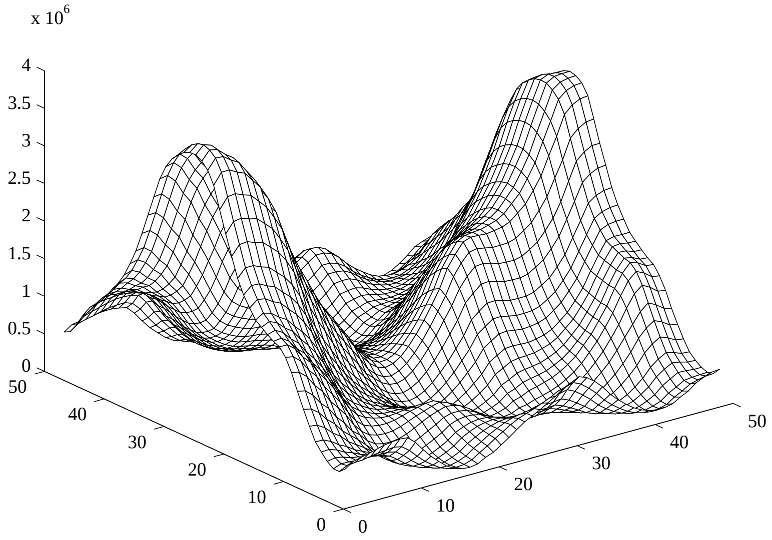

Figure 28: Distance surface between $8-8$. A more complex surface

valley does not form between the two ends of the model, although some valleys will form.

3.3.5 Actual Path Taken

Now that we have a relatively good idea of the environment the algorithm evolves in, we can answer the following questions:

- What path will the algorithm choose?
- Why does performance decreases when the look-ahead is allowed to increase?

In order to determine the reason why performance decreases, we will look at the path the algorithm takes when the look-ahead is allowed to be large. We have already seen that the algorithm will, at a few chosen locations, make large jumps.

To illustrate this, we will use distance surfaces we have shown previously and use the unconstrained depth version of the elastic matching algorithm by implementing

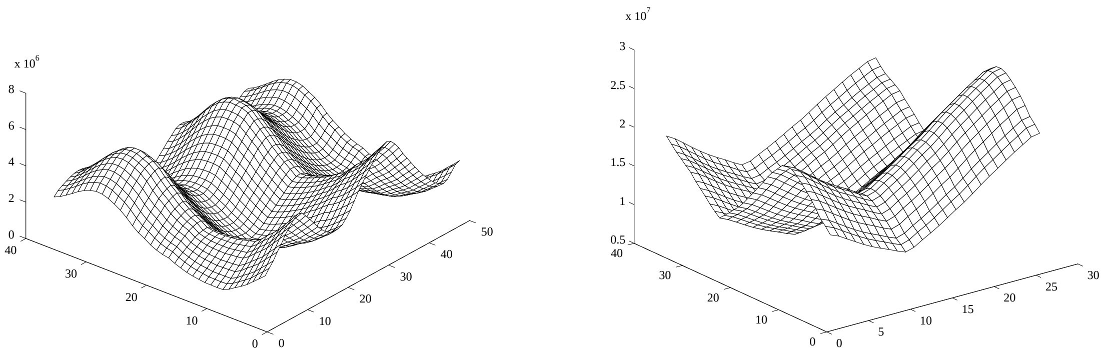

Figure 29: Distance surface for $5 - 8$ (left) and $2 - 7$ (right)

Formula 47. Furthermore, the algorithm was modified to set to zero the value of each point used in the final path of the formula. As a result, we obtain a distance surface where a deep gorge marks the algorithm's choice of points.

We will now compare the two different zero patterns with the look-ahead set at three (3). We want to observe how the path changes with a change in the look-ahead. In Figure 30, where we allowed a look-ahead of three, the algorithm closely follows the diagonal. In Figure 31, where we allowed a look-ahead of eight pixels, the path has changed little, remaining close to the diagonal. This tells us that the path along the diagonal is close to being optimal, following the lowest cost path. This is what we expected as we earlier observed a valley stretching across the diagonal between $(0, 0)$ and (n, m) with two patterns of the same class.

Figures 32 and 33 illustrate the opposite. In Figure 32 we allow a look-ahead of three. Despite the fact that the valley is not located exactly on the diagonal, the algorithm still follows a path relatively close to the diagonal. The result, however, is the relatively high distance which is needed, since this is a comparison between the characters "2" and "8". In Figure 33 we allow the look-ahead to go as deep as eight pixels. As a result, the algorithm quickly takes the opportunity to find a lower cost path. Unfortunately, we stray from the diagonal and, therefore, the points are no longer compared with their corresponding point, but with points that are geographically close. It will also tend to use these points more often, in effect

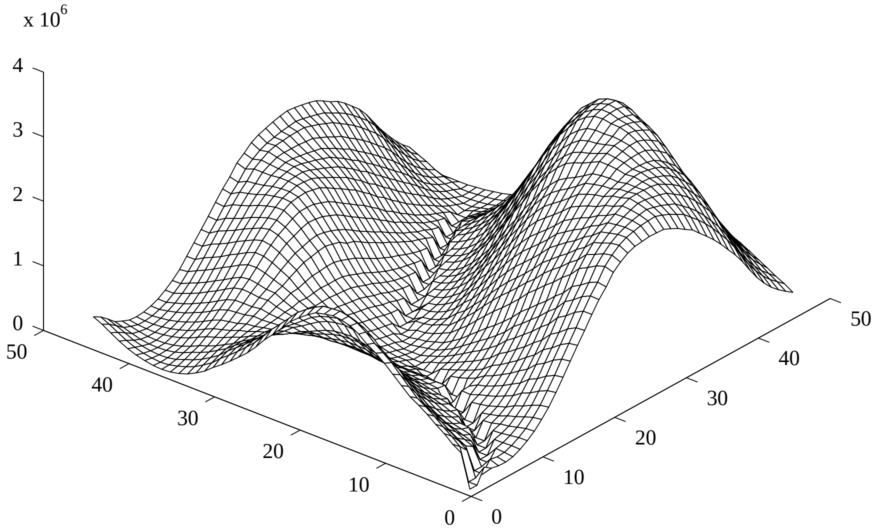

Figure 30: Path taken distance between 0-0, look-ahead 3

reducing the model to only a few points.

As observed in section 3.3.4, a match with a pattern from the correct class gives a valley that stretches, more or less in a straight line, from $(0, 0)$ to (n, m). However, when we have a comparison between two patterns of different class, the lowest path may be in a different location. As a result, allowing the algorithm to use a lower cost path far off the diagonal, will lead to a lower distance. Since the diagonal represents a match between similar points, lower distances obtained, at the expense of going away from the diagonal become less meaningful, the diagonal representing correct tracking.

As we can see, allowing the algorithm a large look-ahead does produce lower distances. However, these distances are not a meaningful measure of the similarity between two patterns. Allowing deeper look-aheads did not prove to be advantageous, as recognition experiments with deeper look-ahead resulted in significantly lower recognition.

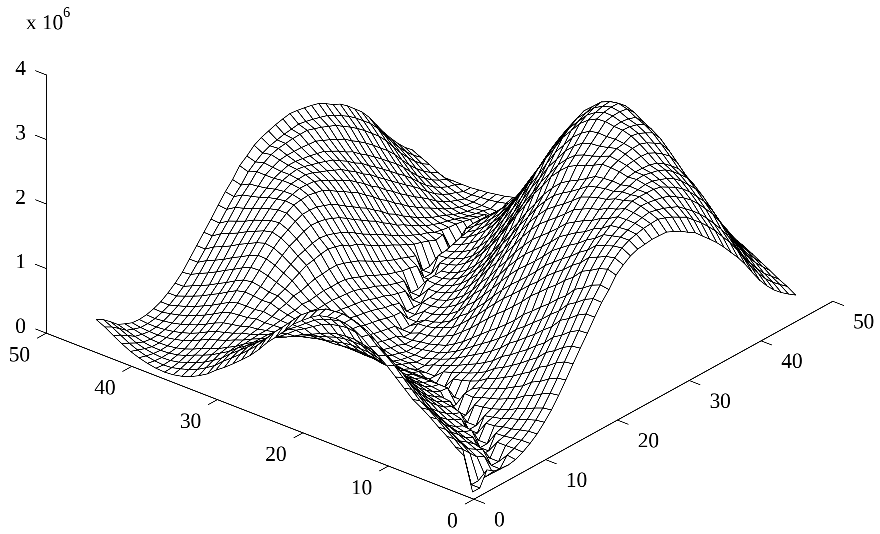

Figure 31: Path taken distance between pattern $0-0$, look-ahead 8

3.3.6 Cost of Operations

The question arises: should we associate a cost with the look-ahead? We clearly saw in section 3.3.3, relating to look-aheads, that allowing the look-ahead to go to an arbitrary depth can result in mistracking. This mistracking does result in lower distances which come closer to the threshold values. Eventually, this increases the chances of confusion, reducing the recognition rate, or increasing the error rate.

Should we remedy this problem by associating a cost with the depth of the look-ahead? We answer by a no. Mistracking is in good part taken care of by the limited look-ahead we allow the algorithm. Replacing this hard limit by some cost associated with a look-ahead is a possibility but does have two drawbacks; first it increases the complexity of the algorithm and, as a result, the cost of computations. Because of the cost of the extensive matching done in the process of recognition, the computations performed need not be increased.

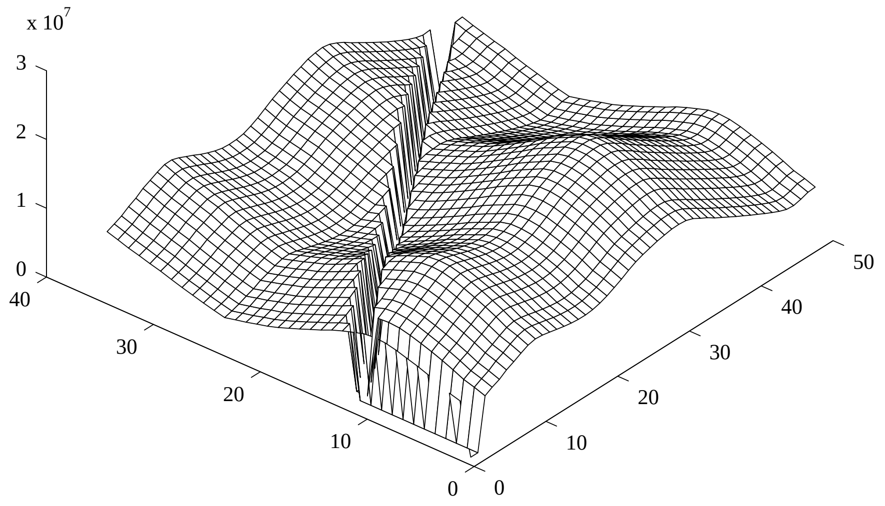

Figure 32: Path taken distance between 2-8, look-ahead 3

Secondly, it will be very difficult to determine the increase in cost each increase in the look-ahead should incur. Intuitively, we can see that the further ahead we look the higher the cost should be, but by how much? This increase in cost should compensate for the reductions incurred by the stepping over of points. In fact, this cost should be representative of how much the resulting distance decreases when we look-ahead deeper. Unfortunately this is highly dependent on the pair of pattern classes that are matched and, in fact, of the patterns themselves.

Finally, giving a cost component to the depth of the look-ahead is completely context-free. Although this appears desirable at first glance, it does not take advantage of the patterns. Pattern recognition algorithms have often recognized that some portions of patterns are more important than others. This is best described by the differences between similar patterns, such as the handwritten letters “U” and “V”. But these cases are precisely the ones giving the most problems and not addressed

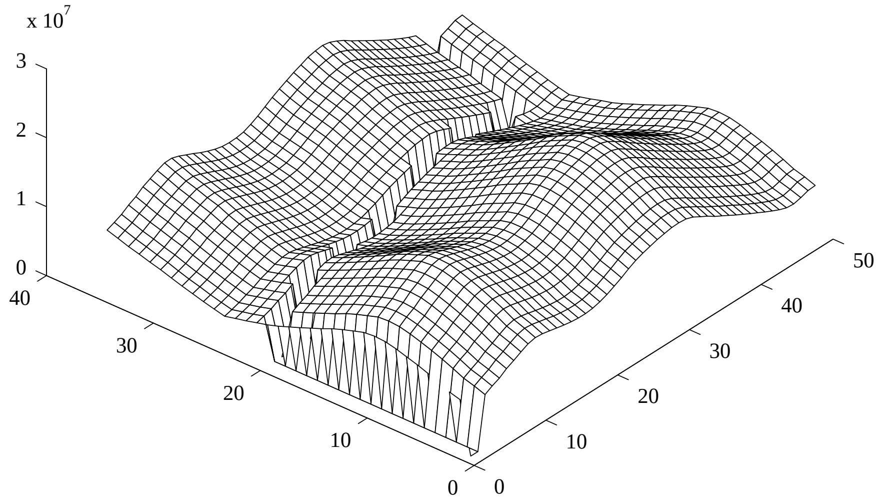

Figure 33: Path taken for match between 2-8, look-ahead 8

by a cost to the depth of the look-ahead.

We did attempt several recognition experiments, associating a linear cost component, increasing with a deeper look-ahead. Various values were attempted. This did not result in a more usable system. It becomes difficult to determine which cost value to associate with the look-ahead. With a low cost we observe the same problems as when we have no cost associated with the look-ahead. Too high a cost and the algorithm quickly degenerates into a zero look-ahead algorithm.

Because of the difficulty in associating a meaningful cost and because, as mentioned above, it does not take advantage of specific model characteristics, we decided not to associate a cost function with the look-ahead. Instead we will keep using elastic matching, as originally described, enhancing it with weights. But before we can fully describe the weights, and the process used to derive them, we first take a look at the training process.

Chapter 4

Model Base Selection

We need to find representative models for each class of digits and each subclass. We call this set the model base. Given an arbitrary set of models in the model base, it is possible to achieve a high degree of recognition, simply by using brute force, and this was our first attempt at creating a model base. But bad models hindered recognition so we needed them removed. A random model base also fails to cover the entire model space.

4.1 Cluster Based Training

We will consider, in turn, each step involved in the training sequence used to create the model base as illustrated in Figure 34:

- **Bad Models:** Removes models causing more errors than recognition.
- **Thresholding:** Selection of the thresholds appropriate for each class.
- **Clustering:** Creation of clusters to extract the most representative patterns.

The training process can be viewed as a multi-step process. First a set of recognition thresholds needs to be selected. They are used for the clustering algorithm and for the bad model removal stage. Once recognition thresholds have been established, we will perform a clustering step to create clusters from which we will extract representative patterns to form an intermediary model base.

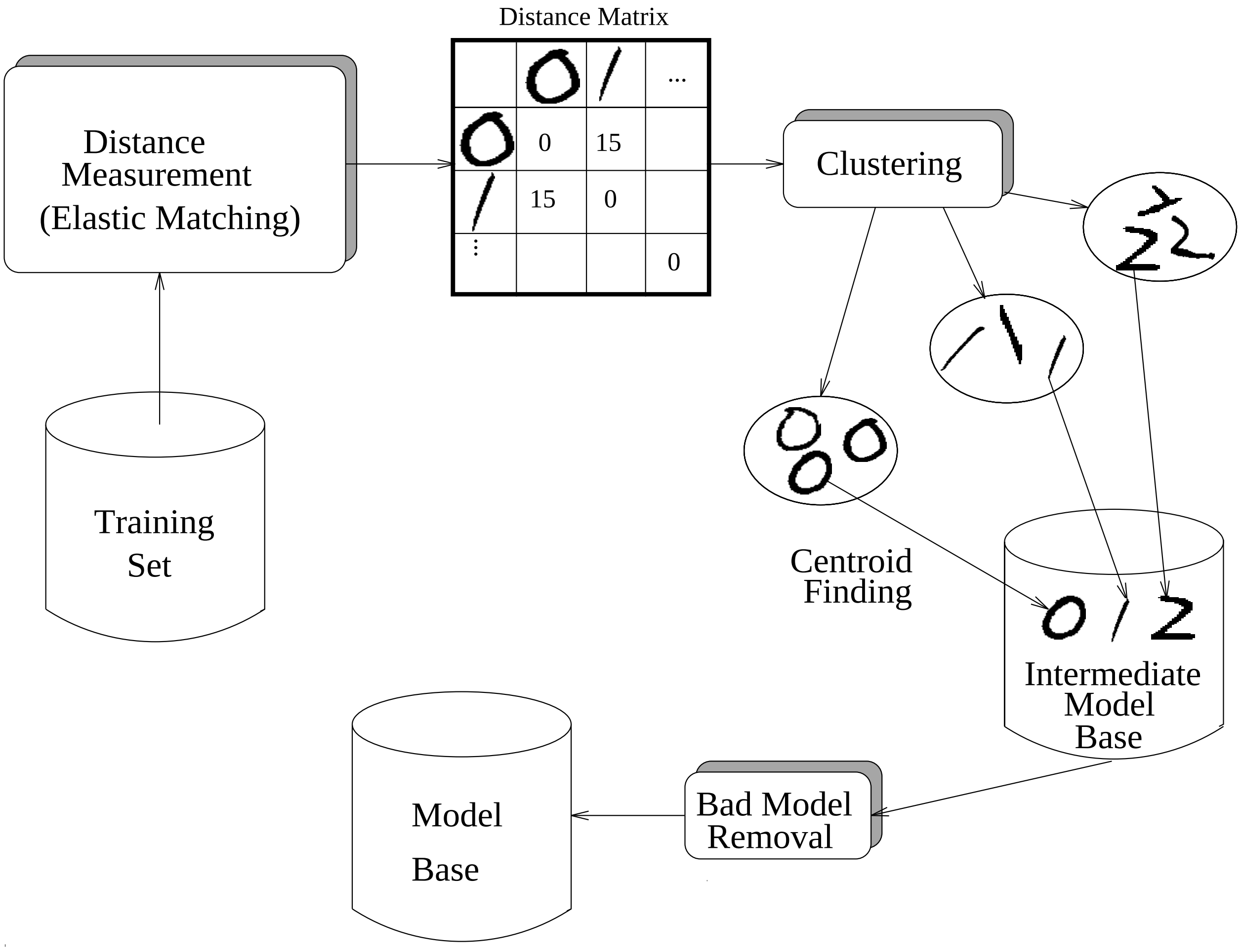

Figure 34: Basic training algorithm, building a model base

Within this intermediate model base we can find patterns that can cause more recognition errors than actual recognition. The final step is a bad model removal, cleaning up the intermediate model base of patterns laying too close to the boundary separating two classes of patterns.

4.2 Threshold Selection

The selection of the thresholds is a simple, but exhaustive, search through all possible thresholds. We run a regular recognition experiment with a given model base and a test set. For each unknown encountered, we keep a record of the lowest distance found for each of the possible classes, telling us how close a given unknown is to each class. We then have a tally of minimum distances for each unknown. We perform an exhaustive search of the possible thresholds, trying all values between 5000 and 60000 in steps of 1000.

For each class, at each threshold, we evaluate the number of correctly classified patterns, substitutions, rejections and confusions. A correctly classified pattern occurs when the distance is below the threshold for the class currently under consideration. A substitution occurs when the distance is below the threshold, but for a class different than the one under consideration. It's a confusion if the pattern is below the threshold on more than one class. We keep the thresholds that maximizes recognition over all the thresholds tried for each class. As a result, the threshold for some classes are higher than for others.

4.3 Bad Models

To determine bad models, we compute, for each model, the number of patterns that were correctly classified and the number of errors caused. A model is useful if it contributes toward the recognition of unknowns in the database. This contribution must not be negated by recognition errors. A model is judged to be a bad model if the following is true:

$$R - (S + C) < 0 \tag{48}$$

where:
- R is the number of patterns recognised
- S is the number of substitutions
- C is the number of confusions

4.3.1 Bad Model Removal

Bad models, as defined by equation 48, need to be removed from the model base. In a fashion similar to the way we calculate thresholds, we perform a recognition experiment. For each unknown, we keep the lowest distance for each class as well as which model gave this distance. With the thresholds supplied, we then classify each of the unknown patterns. In so doing we count how often a given model contributes to a recognition, substitution or a confusion, keeping a running tally of Equation 48. We remove each model that has a negative contribution to the recognition process. The result is a smaller model base with a higher recognition rate.

4.3.2 Models' Influence on Thresholds

Once the bad models have been removed, we can reevaluate the recognition thresholds. Typically, they will increase slightly and result in an improved recognition rate. Bad models influence the setting of the thresholds by inducing a large amount of substitution, bringing down the recognition thresholds.

It becomes important to attempt to avoid bad models, since the thresholds are selected with a recognition experiment. If the model base happens to contain bad models, it will negatively influence the thresholds, lowering them, resulting in overall lower recognition and providing less generic models.

4.4 Clustering

The original model selection process was simple. We picked a given number of patterns, at random, and called them models. This easily provided us with models but the results highlighted the need to be careful in model selection. We reached better recognition with a more careful selection of models, by simply removing models introducing errors. However, two problems remain. The first is one of coverage. How did we know that we had models representing all of the possible styles of writing? We couldn't guarantee coverage, but we needed a procedure that increased the chances that our model base covered all the possible writing styles. We could easily have accomplished this by simply using more models, increasing the chance that we had representative models. However, this brought us to our second problem. The size of the model base is restricted. In the original implementation it was restricted by hardware limitations. Adding models also implied more models to match against, each time we tried to recognize a pattern. So not only storage but recognition speed, a far more limiting factor, needed to be considered.

To avoid the problems described above we decided to turn to clustering to find the best models. From each cluster we select a pattern to be representative of the cluster. By doing so over the entire training set, we obtain, for each class, a set of clusters from which we can extract patterns that we will call models. A recent overview of the field can be found by Jain [54].

Clustering is a problem that has been solved to a level sufficient to satisfy our needs. This field is rather old, and early research focused on developing algorithms that clustered, using the least amount of resources [44]. Later work focused on a different approach to clustering and the properties that the clusters produced exhibit [2]. Clustering is often used to find appropriate groupings of elements for a set of data where it is unknown if clusters exist and, if so, what are their properties [45]. Since clustering is a rich field of research, we restrict ourselves to implementing a class of algorithms known as hierarchical clustering.

4.4.1 Hierarchical Clustering Algorithms

We chose hierarchical clustering for its simplicity and because it gives us control over the clustering process. There are two varieties of hierarchical clustering algorithms described by Späth: *Divisive* and *Agglomerative*. In a divisive algorithm, we start with the assumption that all the data is part of one cluster. We then use a distance criterion to divide the cluster in two, and then subdivide the clusters until a stopping criterion is achieved. An agglomerative clustering starts with each element being in a different cluster. Clusters are then merged according to various strategies, based on the distance matrix between the elements of the training set [91]. We compute the entire distance matrix, and refer to it as often as needed. The computation of the matrix being the limiting factor in terms of CPU requirements, the simple implementation of the merging strategy makes it easy to code and ensures the correctness of the result.

4.4.2 Complete Clustering Strategy Used

We use an agglomerative clustering technique. Clusters are merged if the distance between all the members of both clusters is below a predefined threshold. We call this condition *complete connection.* So to merge two clusters they must be completely connected, i.e. all the elements must be completely connected to the other cluster.

The algorithm starts with each element forming a cluster. We search in the distance matrix for the lowest distance between two elements. We then attempt to merge two clusters if they are completely connected. We keep doing this until there are no two elements left whose distance is below the threshold. Since we cluster only one class of pattern at a time, we use the recognition threshold we have for this given class as a cluster threshold.

This is an inherently conservative strategy and leads to a large number of small clusters. However, we are sure of the coverage of our models. It ensures that we can take any of the patterns in the cluster and the classifier will recognize the entire cluster, but does not preclude models from matching unknowns of another cluster.

We cannot guarantee that we obtain optimal clusters. Since clusters are merged only when all the members of both clusters are fully connected, it may be possible that some patterns of one cluster could be fully connected to the members of another cluster, but the two clusters are not fully connected. Here one pattern, a member of one of the clusters, would recognize all of the other patterns and make a more optimal cluster.

In the end, the possible overlap of models found here might be more of a welcome feature than a hinderance. Because this clustering does not consider the models of error classes, some models may introduce more cases of confusion, or substitution, than they contribute to recognition. Once potential models are found using clustering, we may need to remove some of them, using a bad model removal strategy.

4.4.3 Centroid of a Cluster

The centroid is defined by Späth [91] as follows:

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{49}$$

It represents the average element of a cluster. A clustering algorithm (presented by Späth as HMEANS) is based on this concept. Basically, an initial set of clusters is given from which the centroid is calculated. Each element is moved to the cluster whose centroid is nearest in the Euclidian distance sense. Centroids are recomputed at each iteration, and the process continues until no more changes occur. But this implies that we can compute the distance between cluster members and the centroid. This would require synthesizing a model representing the cluster centroid, upon which we could apply elastic matching. No simple mechanisms to do this are available, making the use of such an algorithm difficult, nor are there any formulas to estimate the distance between an unavailable synthetic model and an unknown.

In our case, we define the centroid as the pattern that has the smallest distance, on average, to all the members of its cluster. We are interested in using the centroid of the clusters as models. Improvement should come from the fact that patterns should provide the most uniform coverage of the clusters, being near the centre of the cluster. Once the clustering algorithm has stopped, we have a set of clusters from which to take models. Two strategies were used. The first strategy is to take the first element that formed the cluster. Since the cluster is fully connected, any element can represent it completely.

$$D_{centroid} = \min \textstyle\sum_{i=1}^{n} D_{k,i} \quad k = 1..n \tag{50}$$

However, to improve generality, we wanted the element that was in the "centre" of the cluster. We then settled on taking the centroid of the cluster, as we define by Equation 50, as the most representative element of the cluster. This strategy does slightly improve the recognition rate over a random selection from each cluster, when we select the models from the first n biggest clusters provided by the algorithm for each class of patterns.

4.4.4 Model Selection

From the above we realize that there are two main steps involved in model selection. First, we create clusters containing patterns that give us potential models. But these models must be weeded out with the bad model removal procedure. Essentially a model can come from a small cluster, or from patterns that are near the border between two clusters. For this reason they can produce more errors than correct recognition and should be removed. We tried three different variations of this procedure using clustering and bad model removal.

- **Simple Clustering:** Simply cluster candidate models and take one of the models for each cluster.
- **Cluster/Removal:** Cluster the candidates and take models as in the case of Simple clustering. Then apply bad model removal on this model set.
- **Removal/Cluster:** First do a bad model removal for each of the classes. Once the bad models are removed, apply clustering to each class.

Simple Clustering

In this case, simply apply the clustering algorithm and select a given amount of models. This does provide for a reasonable set of models. From information provided by the clustering algorithm we know that, with our data, all of the patterns within the training set are usually represented by a limited set of clusters we take as models.

Removal/Cluster

In this strategy, we find which patterns can be bad models by performing a recognition experiment with all of the candidates of a given class. The patterns that are considered bad models are then immediately removed. Once this step is performed, clustering is applied to the remaining patterns. From the clusters obtained we keep a set of models.

Cluster/Removal

With this strategy we first perform the clustering operation. From this we keep a given set of models. We then apply the bad model removal step to the given set of models. We then obtain the model set.

Strategy Used

To determine which method to keep we performed recognition with the sets, as provided by all the methods, keeping the Cluster/Removal procedure. This is the method that gave the highest recognition rate by 1%. It was originally thought the other methods would perform better. It appears that removing the bad models, first changes the shape of the clusters and draws them away from the border between potentially confusing classes. As a result, the clusters would not be spread as wide and would not represent as large a variety of unknowns.

4.4.5 Clustering as a Measure of Generalization

The quality of a recognition algorithm is often measured by recognition results and the reliability of the system. The generalization power of a method is more difficult to characterize. It is loosely defined as the capacity of an algorithm to be able to correctly handle a wide set of input shapes, and is a measure of the robustness of the algorithm.

The best measure of generalization, in this system, is related to the model base. The larger the number of unknowns it can classify, the more general the model base is. Alternatively, we can use the clustering results we obtain to attempt to measure the generalization power of an algorithm. When we select the model from a cluster, we can consider the size of the cluster as an equivalent measure. This measure may tell more about the distribution of the shapes of patterns than generalization. So we are more interested in how this measure changes with modifications to the algorithm. Do the models, on average, represent more elements than before the modification to the recognition algorithm? Observations were made during the clustering phase. Only a few models represent quite a large portion of the patterns, while most of the other models represent only a few unknowns.

When the size of the training set shrinks, we then believe that, at least, some models gain in generality. It is unclear if this is concentrated on some already relatively general models, or if it is spread across a wider base of relatively specific models.

4.5 Weighted Elastic Matching

One striking feature of elastic matching, as we have presented it so far, is its lack of specificity. Each of the points of the model is considered to be of equal importance as is reflected in the original formula (Formula 9). The total distance is the sum of the distance at each point. Each point being matched, in the unknown, contributes an equivalent amount to the total distance. It is frequently desirable in character recognition algorithms to give some portions of a pattern more importance than others, as stated by Suen [95]:

> *On the other hand, it may also lead us to concentrate our efforts on the most significant (or crucial) distinctive parts of the characters for more effective and efficient recognition*

In the system described by Suen and Li [67, 95], the bounding box of the pattern is divided into subsections. Recognition is then based on determining which part of the image distinguishes most between templates. A hierarchical model is then defined to take advantage of the most distinguishing portions [96]. Essentially, this is a template based system, enhanced by placing more importance on some portions of the patterns.

In early work in pattern recognition, the importance of some features over others was noted. The presence of features or their shape can vary a lot, while the impact on the meaning may stay unchanged [12]. Blesser [13], paraphrasing Neisser, states it as follows:

> *To do so, however, probably requires the use of algorithms which have a direct correspondence to the rules which are used by human observers. Such techniques as template matching must therefore be rejected since they are not good descriptions of human perceptual behaviour. The fact that such techniques can be used successfully with machine-printed characters does not argue for their applicability to recognizing handprinted characters.*

This leads to a system where the distinctions between patterns of similar classes are made by *functional attributes*. They give importance to differences that are not very large in terms of Euclidian distance, for example in the case of the letter "U" and the letter "V". With such a system, one pays special attention to some localized features of a given pattern [13].

Consequently, we believe that it is important to add a way of embedding this notion within elastic matching. Elastic matching should not treat each point equally. It should be given the possibility to place more importance on certain points than on others. Therefore, we are adding weights to the points of the models, hoping this will increase the distance between models of a given class and unknowns from a different class, reducing the distance between unknowns and models of the correct class.

The introduction of weights changes the pattern that is matched. In doing so, it changes the shape of the cluster associated with a given model. It should, if the weights are calculated appropriately, increase the number of patterns that are members of the clusters. Hopefully, it will enlarge the clusters, or change their shape to better delineate the border between patterns of the correct set and the error set.

4.5.1 Weighted Elastic Matching Process

The questions are: how to add weights, and how they will behave. To understand the implications, we must take a closer look at Equation 9. Expanding the formula we obtain Equation 51.

$$D(i,j;k) = d(0,a;k) + d(1,b;k) + d(2,c;k) ... + d(n,z;k) \tag{51}$$

Recognition is a linear combination of distances. We may then be tempted to add the weights to each of the distance terms.

$$D(i,j;k) = \omega_j d(i,j;k) + min \left\{ \begin{array}{l} D(i-1,j;k) \\ D(i-1,j-1;k) \\ D(i-1,j-2;k) \end{array} \right\} \tag{52}$$

Equation 52 now includes weights. We may be inclined to believe that it does what we want. At first glance it does. Depending on the weight, the distance for a given point will take more importance in the sum. However, this will also affect how the recursion will proceed.

If we expand at the min part of the equation, we realize that there is a possibility of a different value for j being found, which means that the lower distance is now achieved by looking at a different point. This may appear, at first, inconsequential, as we are attempting to minimize the distance.

4.5.2 Weight Induced Tracking Error

With Equation 52 we are still applying the same dynamic programming technique. We are now minimizing the distance, not by finding the best pattern, but by avoiding, in part, matching with the significant parts. Even though the distance d will be lower because of the higher weights, a point, presumably significant, will be penalized by the higher weight ω. Therefore, the more significant portion of the pattern, with a correspondingly higher weight, will not match as often.

This is the opposite of the desired effect. Instead of more significant parts taking more importance, they are avoided and, by that token, take less importance. This is very similar to the tracking errors we noted with the unconstrained elastic matching. This conflicts with the role of the weights. As is the case with search depth induced tracking errors, it is likely that recognition will suffer more errors as a result.

4.5.3 Improved Weighted Elastic Matching

Because the tracking error is introduced by the addition of the weights within the recursion, we will remove the weights from the recursion. We have seen that the result of the recursion can be expressed by a linear combination of coefficients in Equation 51. The action of expanding, performed by the application of the recursion, creates a relation between the model and the unknown:

$$track(i) = j \mid d(i,j) \in D(n,m;k) \tag{53}$$

This relation gives, for every point of the unknown, a corresponding point in the model. Each point of the unknown can now be linked to a point of the model. This tracking function can now be used to generate a new formula for weighted elastic matching that expands into:

$$DW(i,j;k) = \begin{array}{l} \omega_{track(0)}d(0,track(0);k)+ \\ \omega_{track(1)}d(1,track(1);k)+ \\ \omega_{track(2)}d(2,track(2);k)+ \\ \vdots \\ \omega_{track(n)}d(n,track(n),k) \end{array} \tag{54}$$

which we write as:

$$DW(n,m;k) = \sum_{i=1}^{i=n} \omega_{track(i)}d(i,track(i);k) \tag{55}$$

Let us now define a function $dtot(j;k)$ that gives the sum of all the distances that were used on model point j while matching with an unknown pattern.

$$dtot(j;k,l) = \sum_{i=0}^{i=n} \left\{ \begin{array}{ll} 0 & track(i) \neq j \\ d(i,j;k,l) & track(i) = j \end{array} \right\} \tag{56}$$

With function $dtot(j;k,l)$ as defined in Equation 56 we can now rewrite a simpler, and more familiar, version of Equation 55. Equation 57 is the simplest expression we can use to describe the elastic matching process.

$$DW(n,m;k,l) = \sum_{j=1}^{j=m} \omega_j dtot(j;k,l) \tag{57}$$

Equation 57 is implemented, using the iterative elastic matching. This implicitly gives us the $track(j)$ function. Once $track(j)$ is identified we keep a tally of $\omega(track(j))d(track(j),j;k)$ which is equivalent to Equation 56, where only the non zero terms are evaluated. We prefer Equation 55, as it mirrors well how the procedure is implemented. This formula now behaves as we would like, the weights not interfering with the tracking properties of the algorithm. The higher the weight at a given point, the more critical a close match to the model is at that point.

4.5.4 Determining Weights

The question now is how to determine the value of the weights. We could consider doing it manually by assigning weights to portions we deem more significant. Experiments in determining which portions of characters humans think are important,

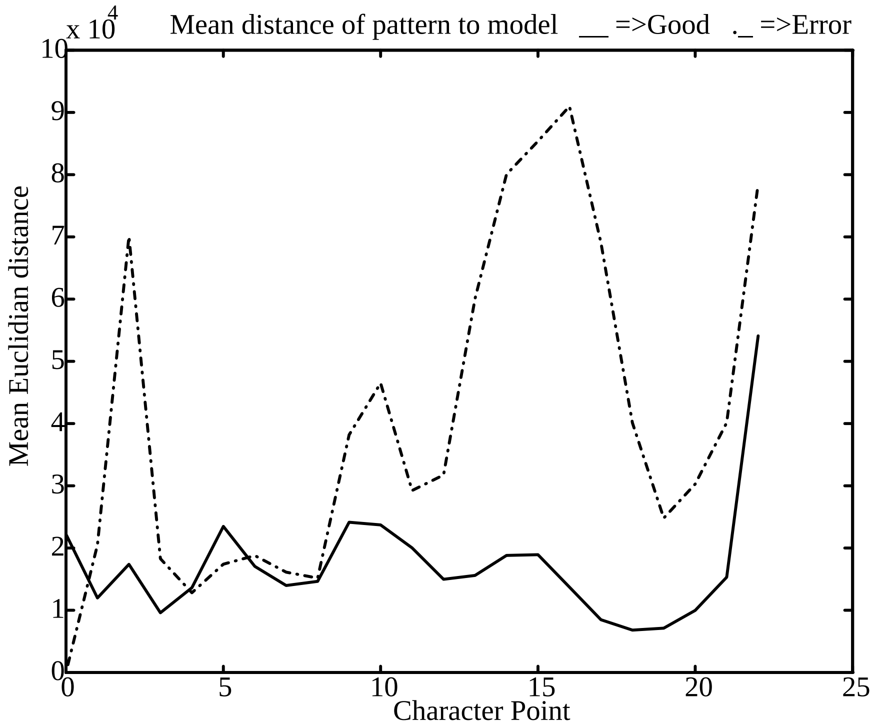

Figure 35: Average distance for each point between correct and error set to a model

have been done in the past [95]. But this is impractical in our case as we need to process several hundred patterns. Furthermore, it is not clear, at first glance, that what is significant to a human experimenter will be significant to the elastic matching algorithm. Since we have a lot of models, we need an automated manner of evaluating these weights.

What are the characteristics one would like weights to have? Since the formula is a linear sum of terms, the higher the weight a point is given the more important a role it will play. We then need high weights for points that will be good discriminators. To do so, we associated with each model two sets: a set of unknowns, which belong to the same class as the model, and a set of unknowns which belong to any other class but that are at a close distance to the model. We will refer to these two sets as *correct set* and *error set*.

We are looking for a characteristic that will be a good indicator, at any given point, of whether or not the point is a good discriminator. We are interested in points where the distance function $d(i, j)$ will be low for the correct set and high for the error set. This is illustrated in Figure 35. This graph shows, for each point of the model (on the X-axis), what the average distance is with respect to the corresponding point of the unknown from either the correct set or the error set. In order to calculate this, we perform a recognition experiment where, for all the models in the model base, we keep the $d(i, j)$ from unknowns for both sets with respect to each point in the model.

To measure how good a discriminator a point is, we use the ratio of the average distance of the error points over the average distance of the correct points. Formula 58 behaves as needed, as it grows when the distance between the two sets increases.

$$\omega_j = \frac{d_{err}(track_{err}(j), j; k)}{d_{good}(track_{good}(j), j; k)} \tag{58}$$

This measure is quite simple to evaluate and does not require a large amount of data. As a result, we are using it to evaluate the weights in the training algorithm. Applying this formula to the same model used in Figure 35 gives the result shown in Figure 36.

We are not using any statistical information. Of interest would be the distance of the mean and the variance of the distributions of distance for the error set and the correct set. This would require more computations and more data, so we decided to leave it out, even if it could provide us with more reliable weights.

4.6 Calculating Weights

We now have Formula 58 to add weights to the models. The question is how to go about creating a model base with weights inserted. We need to create, for each potential model, a set of unknowns of the correct set and of the error set. Once we have obtained the needed sets, we evaluate the average distance, from points of the unknowns to the points of the models by using Formula 59.

$$\overline{dtot}(j; k, set) = \sum_{\forall l \in set} \omega_j dtot(j; k, l) \tag{59}$$

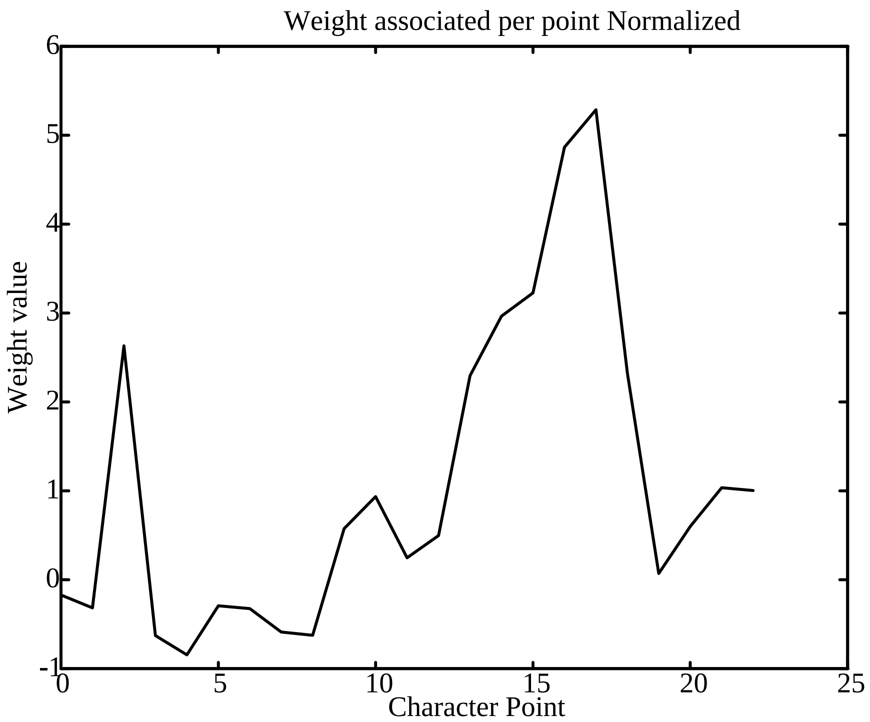

Figure 36: Weights given to each point as estimated by *error/correct* set

With the average obtained over both sets for all the points of the model k, we can then apply Formula 58 to each point of the model. This will then give us weights for the entire model as given in Formula 60.

$$\omega_j = \overline{dtot}(j; k, err)/\overline{dtot}(j; k, good) \tag{60}$$

We should mention that in the definition of $dtot(j; k, l)$, the distance formula used contains a weight ω. At this point we assume that all the weights are equal to the unit weight 1. This is true when we calculate the weights for the first time.

In order to prevent the weights from taking values that are too high or too low we normalize them so that the highest weight has a maximum value of β, or the lowest weight a minimum value of $1/\beta$. This normalization process is used to prevent the weights from reaching extreme values during the iteration process. All of the recognition experiments were performed with normalized weights. In this case $\beta = 4$. We later performed experiments with no normalization step. Recognition results were slightly lower but not significantly. Although we can play with the scaling constant to increase the recognition rate, it does not have a large impact. A recent experiment with $\beta = 10$ yielded only a 0.06% increase in recognition with on-line data with the test set against itself, while increasing by 0.37% the recognition with the training set against the test set.

4.6.1 Iterative Weight Calculation

Once we have calculated the weight, we hope that the distance measurement will change for the better. This implies that some patterns will move in closer, while others will move farther away. This means that both the correct set and the error set used to calculate the weights will change if we attempt to calculate the weights again. Hence the weights are no longer optimal. This iterative weight calculation process is illustrated in Figure 37.

We want to mention a difference with the process used to create the model base. There is no bad model removal phase. This is because we don't know, until we have established which iteration is the best, which models will be retained. As a result,

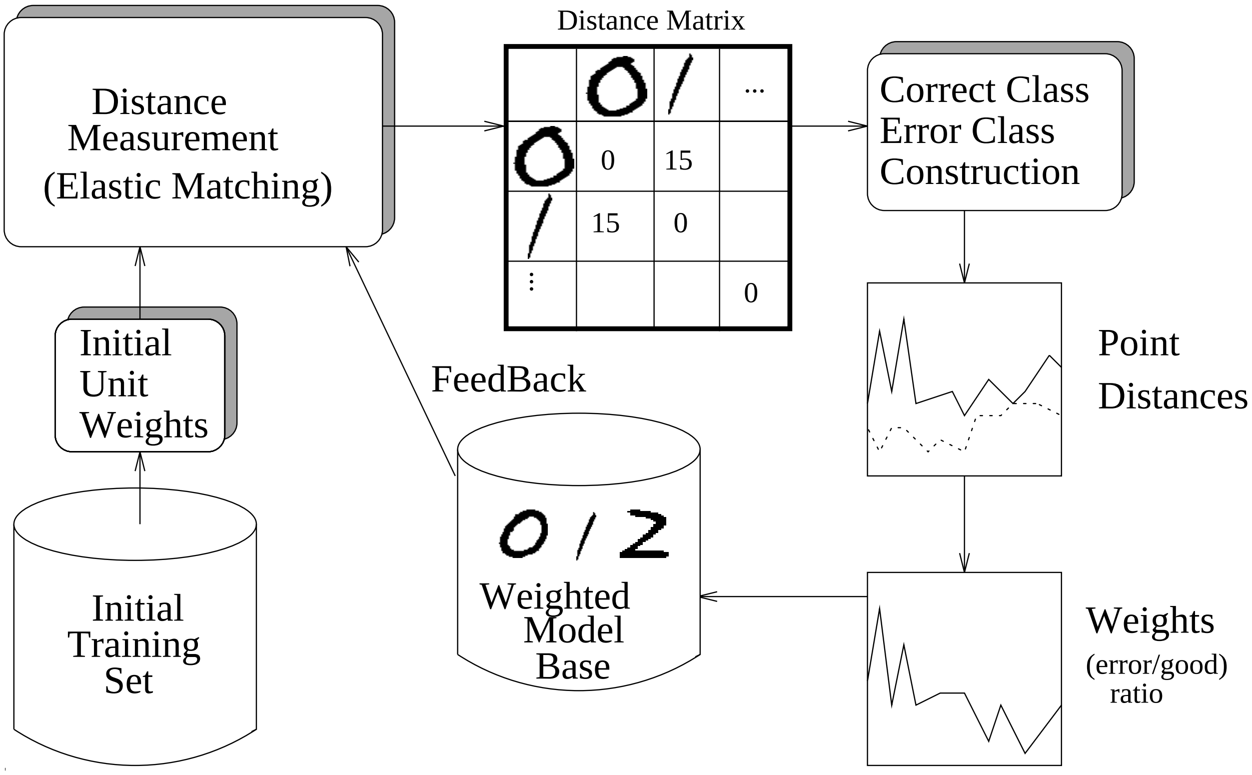

Figure 37: Iterative process used to refine weights

we calculate weights for each of the potential candidates available in the training set. Each pattern in the training set will get correct and error sets calculated. From here, new weights will be calculated and the current weights will be updated. Because there is no model base needed for the creation of weights, there is no clustering involved.

One problem remains: the thresholds needed by the recognition, as well as the process used to establish the correct set, need to be reevaluated at every iteration. For the weights to serve their purpose, the distance between the error set and the models should increase. As a result, to take advantage of the increase in distance, we need to modify the recognition thresholds.

Consequently, we update the training process by adding another feedback loop to update the recognition thresholds. At each step, an optimal model base is created. With this model base, a recognition experiment is performed. From that recognition experiment we can calculate the optimal thresholds. This updated weight calculation process is illustrated in Figure 38. These thresholds are then kept for the next iteration. Note that the initial thresholds need only to be a good approximation of the optimal thresholds needed under these circumstances.

The two feedback loops makes this training process harder to analyze. However, we believe that there are no other ways of ensuring that the recognition results stay good. One could be tempted to fix the recognition thresholds at a given value. But, because the thresholds essentially determine the recognition results, it is important to keep them as near optimal as possible. What was optimal for a given iteration may not be at the next one.

Experimental results show that recognition thresholds change at each iteration. However, they stay within a given band of values throughout the iteration process (from the first few iterations to the last one). Furthermore, the order of each recognition threshold remains relatively constant. Although there are good reasons to doubt the stability of this iteration process, it does not appear to behave erratically.

We need a formula to update the weights. Equation 61 is the first we used for the weights. We simply average the past weights with the new one.

$$\omega_i' = \frac{\omega_i + \overline{dtot}(j;k,err)/\overline{dtot}(j;k,good)}{2} \tag{61}$$

C is a constant, in our case equal to 1. Note that this is not the only possible way of adding the weights together. We also use an alternative formula, based on the product of the value shown by Equation 62:

$$\omega_i' = \sqrt{\omega_i * \overline{dtot}(j;k,err)/\overline{dtot}(j;k,good)} \tag{62}$$

Both of the above formulae now allow us to update the weights of a model that already has weights. The question is: which of the above formulae should we use in the iterative training process? At first glance these functions appear interchangeable for our purpose, and we consider them functionally equivalent. However, we did compare the recognition results between the two to determine if one is preferable. We performed an experiment with an iterative training process, using the training data from the on-line testing database.

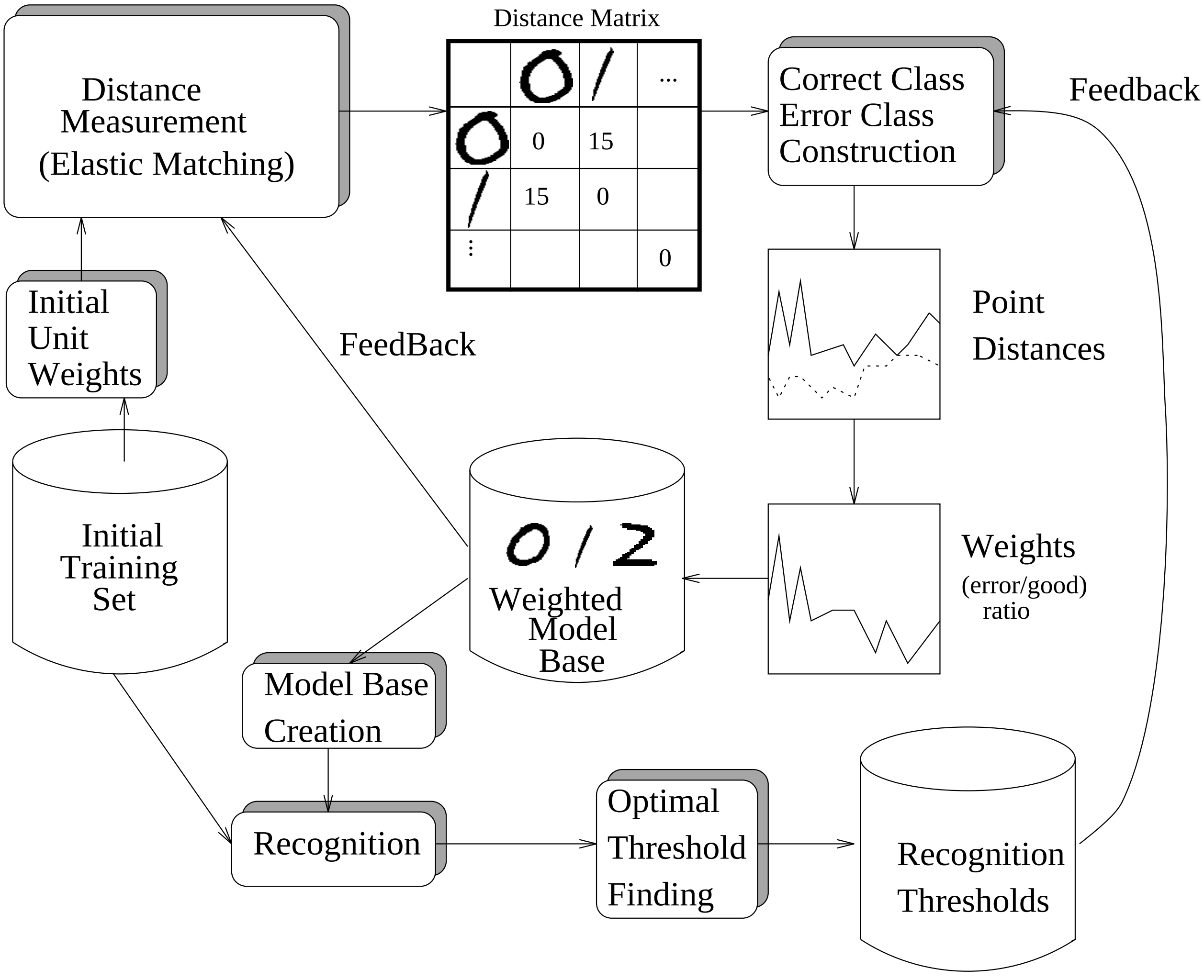

Figure 38: Updated weight refinement process

The results are illustrated graphically in Figure 39. We notice that neither formula consistently outperforms the other, for either recognition values or reliability. Because the product-based Formula 62 improves recognition faster than the sum-based formula, we decided to use it for the remaining experiments.

Iterative Training Process

We have mentioned that including weights would probably change the distances and that this warrants more training. When weights are added to the models, they change the contents of the error and correct sets. Weights will then need to be recalculated because the distance function 59 will change. The training then simply becomes a matter of reapplying the training algorithm to the weighted models, using either of the formulae discussed before.

Figure 39 gives the results of the iterative weight training procedure. We benefit from training as we can obtain better results after several iterations. However, the algorithm does not converge to a solution. As we can see, recognition results oscillate over iterations. This is consistent with other recognition systems of the same type, such as the perceptron. The only stopping criterion we defined is a maximum number of iterations. In this context, we can keep the best result obtained after a fixed amount of iterations. In most cases that we tried, we obtained a very good solution within the first ten iterations. We can keep the results of the best iteration as the best weight.

Although we do not converge to a solution, in all the cases that we tried, the iterative training improved the results. Thus we conclude that even if weights improve the recognition over simple elastic matching, we can still improve recognition further with the iterative process.

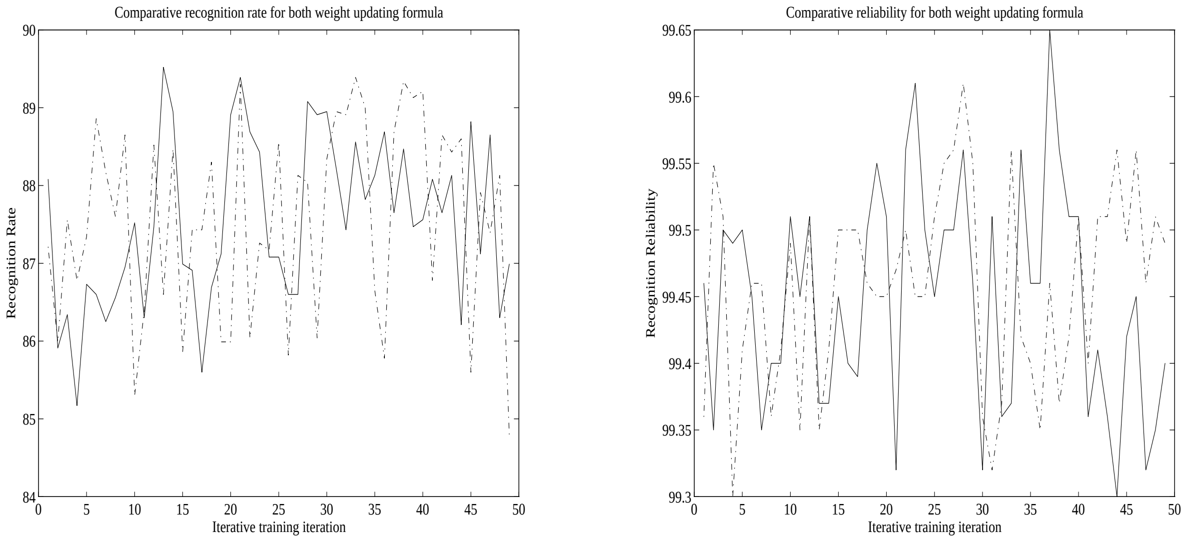

Figure 39: Comparison of weight updating formula. Solid=Sum Dashed=Square

4.7 Error Class Pattern Selection

It is important that, in both the correct and error sets we find patterns that are closely related to the one in which we are interested. In doing so, we hope that the particular feature that needs to take importance will then be emphasized. If we allow too many models in either set, it is possible that some distinguishing feature will be drowned out by patterns that are less related but which appear in larger numbers.

Patterns for the correct set are easy to find. We keep the patterns that match below a certain distance, probably related to the recognition threshold for that given class. Similar reasoning can be applied with the pattern for error set by allowing for greater distances. Because the recognizer achieves reasonable results, it implies that few models from the erroneous class match, constraining us to use unknowns that are further than the recognition threshold.

To determine under what conditions we can select patterns to form an erroneous class, we look at the distribution of the distance function. We are interested in a few things. First, the aspect of the distribution itself. How do error classes measure against the models? Is there a cluster of error class patterns close enough to the threshold value?

In Figures 40 to 43, we show the distribution function for the models of each class against the potential models of the training set. With the histograms in Figure 44, we observe the overall tendency. The histograms in Figures 40 and 41 represent the distances of the models of a given class to all the unknowns. The X-axis shows the distance between the model and the unknown. The Y-axis shows the number of times a given distance between an unknown and a model was obtained.

The majority of the distance measurements are large. In most cases, the distance is so large that it is arbitrarily set to a high value. Because of their large number, these results were omitted, emphasizing the beginning of the graph.

As we expected, the matching of the patterns with the right models gives a large hump in the region near the recognition threshold values for each of the classes. With much fewer patterns, the same phenomenon is observable with error class matches. However in this case the hump is, on average, at twice the value found for the correct

Thresh.	Nb	Prcnt	Thresh.	Nb	Prcnt	Thresh.	Nb	Prcnt	ID
18000	1867	23.44 %	36000	4011	50.35 %	54000	4719	59.24 %	0
10000	1358	9.06 %	20000	4889	32.61 %	30000	6715	44.79 %	1
24000	1872	21.55 %	48000	3976	45.77 %	72000	4987	57.41 %	2
16000	2167	13.31 %	32000	5410	33.22 %	48000	7258	44.56 %	3
15000	848	8.77 %	30000	4374	45.22 %	45000	6920	71.55 %	4
24000	2382	32.88 %	48000	4688	64.72 %	72000	5510	76.06 %	5
17000	1776	18.36 %	34000	4642	47.98 %	51000	6492	67.11 %	6
11000	777	6.39 %	22000	2726	22.42 %	33000	4138	34.04 %	7
35000	1771	35.81 %	70000	2019	40.83 %	105000	2193	44.35 %	8
19000	1824	15.77 %	38000	3685	31.87 %	57000	4279	37.00 %	9

Table 7: Distance histogram statistics Good Match

Thresh.	Nb	Prcnt	Thresh.	Nb	Prcnt	Thresh.	Nb	Prcnt	ID
18000	10	0.31 %	36000	169	5.18 %	54000	407	12.47 %	0
10000	3	0.04 %	20000	188	2.26 %	30000	899	10.82 %	1
24000	13	0.42 %	48000	640	20.48 %	72000	1232	39.42 %	2
16000	8	0.09 %	32000	727	8.08 %	48000	2039	22.65 %	3
15000	14	1.40 %	30000	137	13.70 %	45000	276	27.60 %	4
24000	4	0.47 %	48000	113	13.23 %	72000	251	29.39 %	5
17000	105	3.17 %	34000	586	17.67 %	51000	1348	40.64 %	6
11000	100	1.39 %	22000	558	7.75 %	33000	1011	14.04 %	7
35000	1	0.05 %	70000	48	2.31 %	105000	148	7.13 %	8
19000	37	0.63 %	38000	300	5.14 %	57000	575	9.85 %	9

Table 8: Distance histogram statistics. Match all but correct class

class match.

The statistics in Table 7 are obtained using the histogram data of Figure 44 . It indicates the number and percentage of patterns that matched the models with a distance that is less than specified. The thresholds were selected to be, first, the recognition threshold, and then, multiples of two or three times the threshold. Table 7 gives the distance where the patterns from the given class were matched against all the unknowns. This gives us a good idea of the distribution function. Table 8 gives the distance where the patterns from the given class were matched against all of the unknowns, except those of the same class as the models.

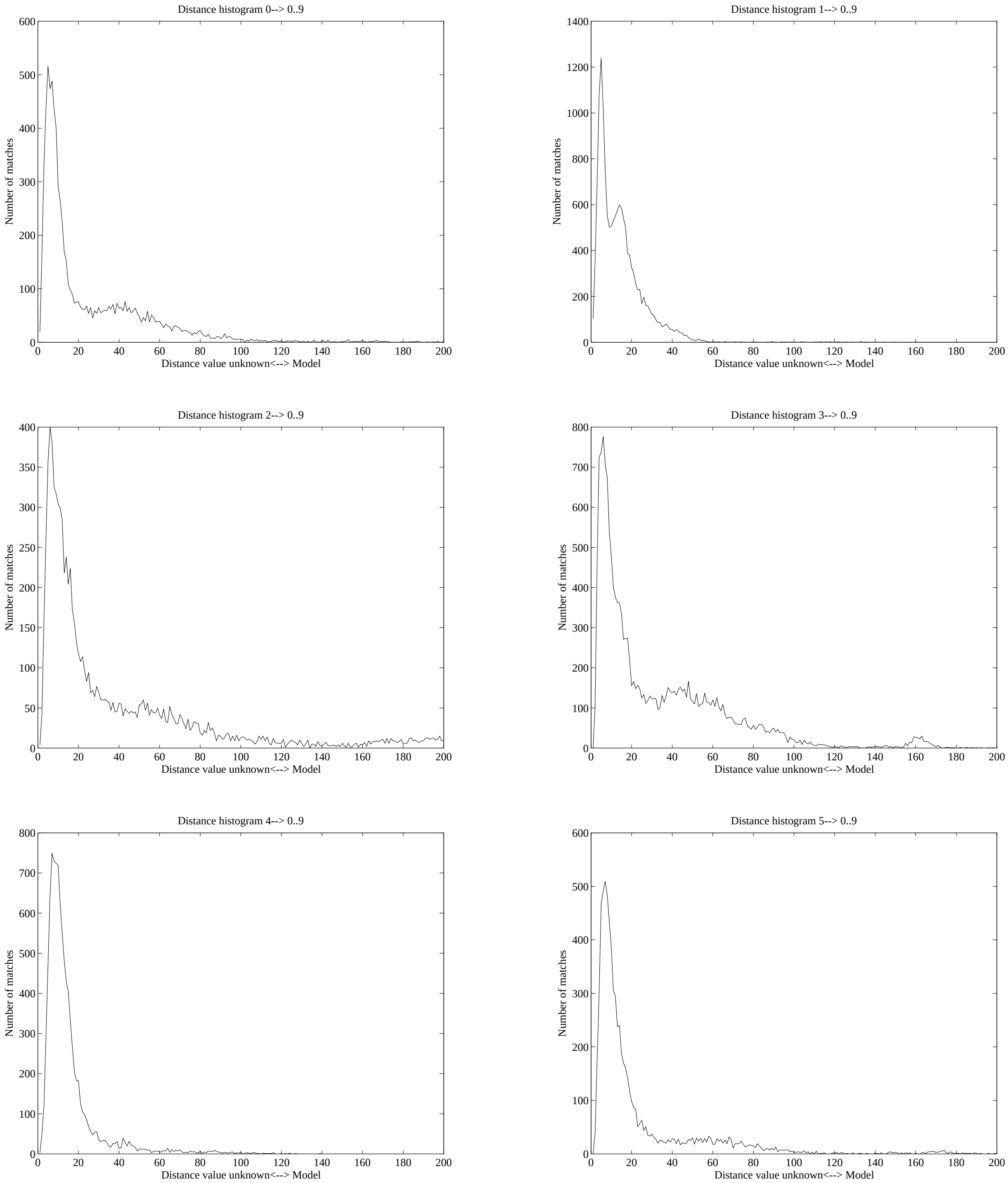

Figure 40: Distance of models versus all of the other patterns 0 to 5

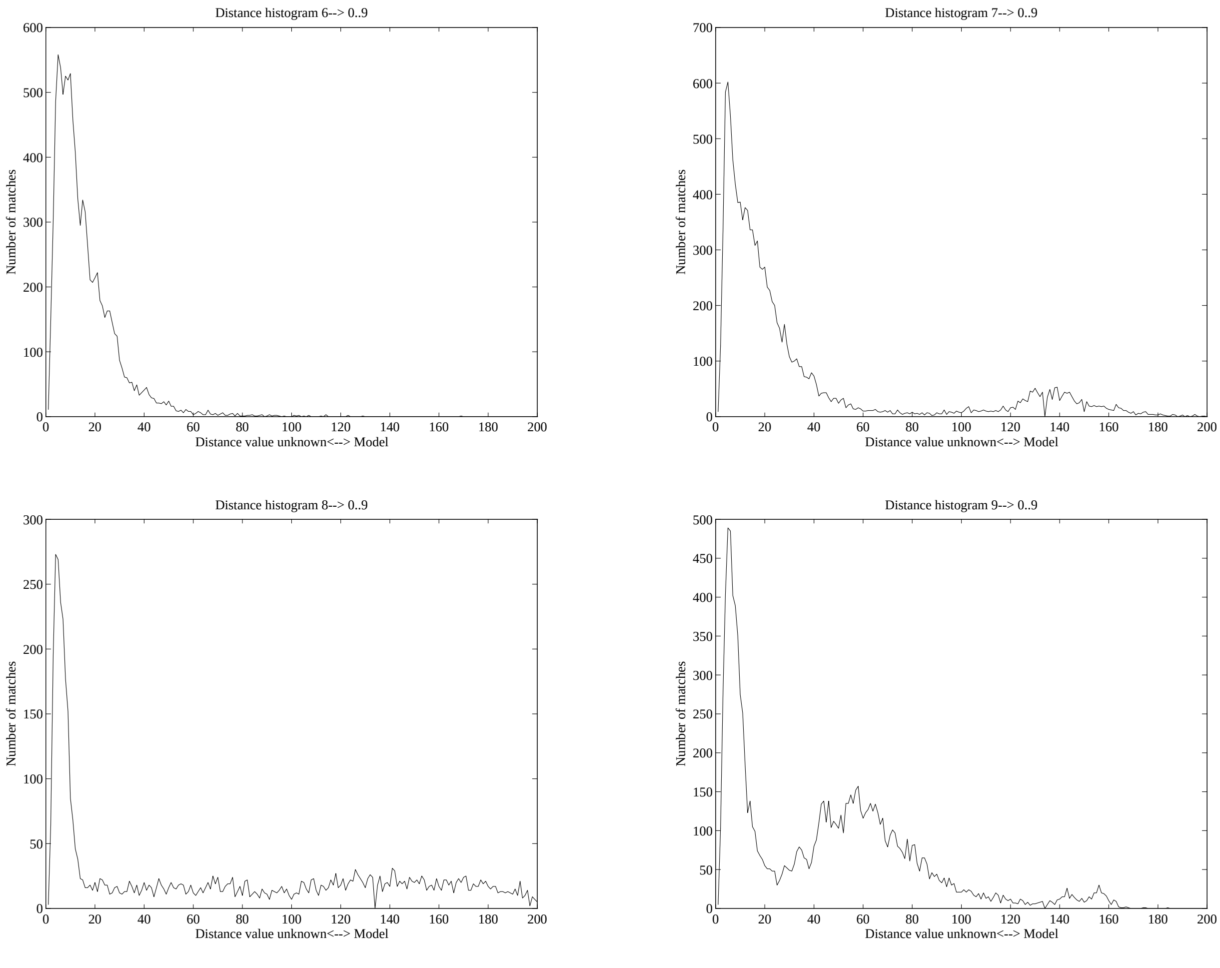

Figure 41: Distance of models versus all of the other patterns 6 to 9

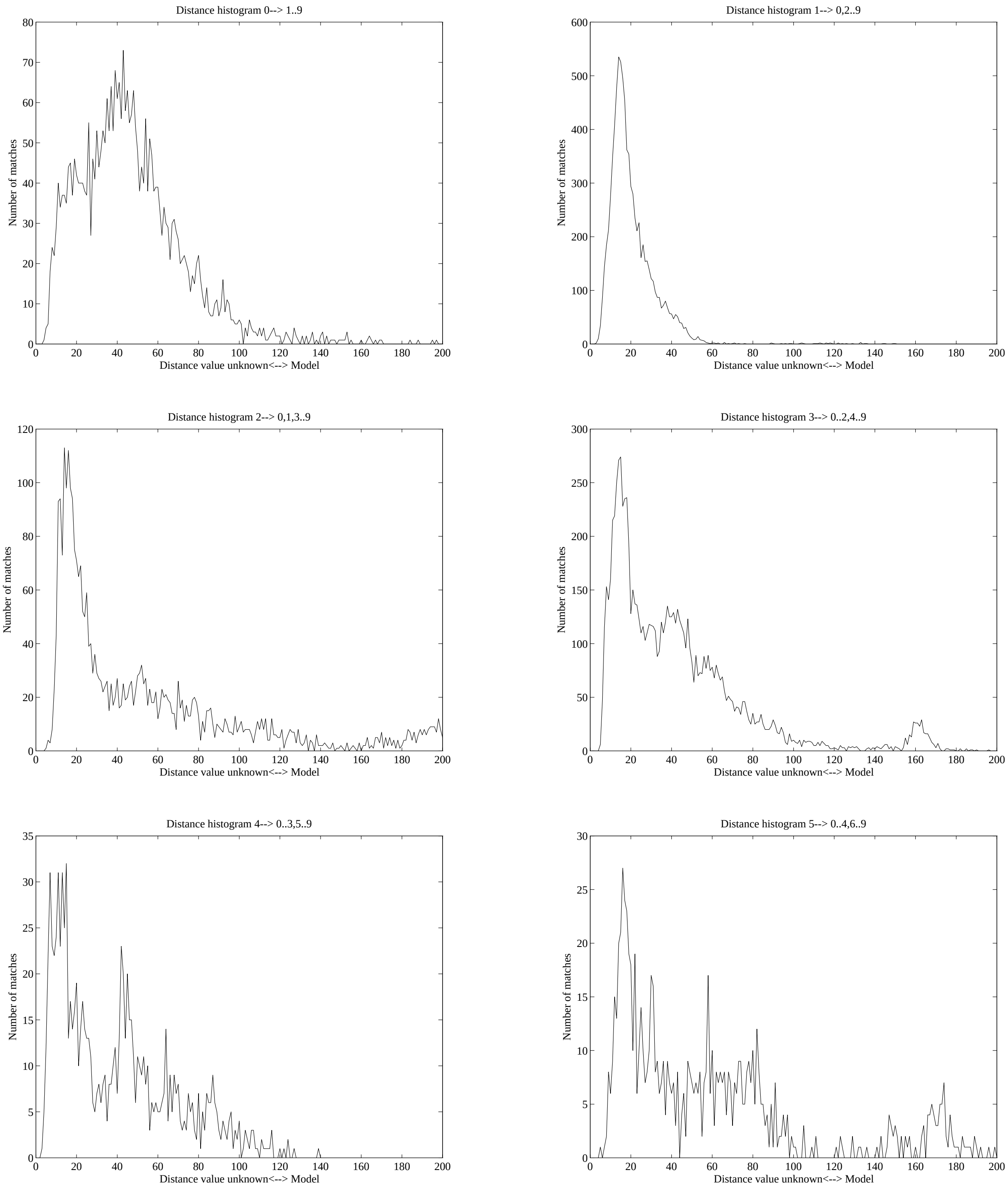

Figure 42: Distance of models versus all of the other patterns 0 to 5

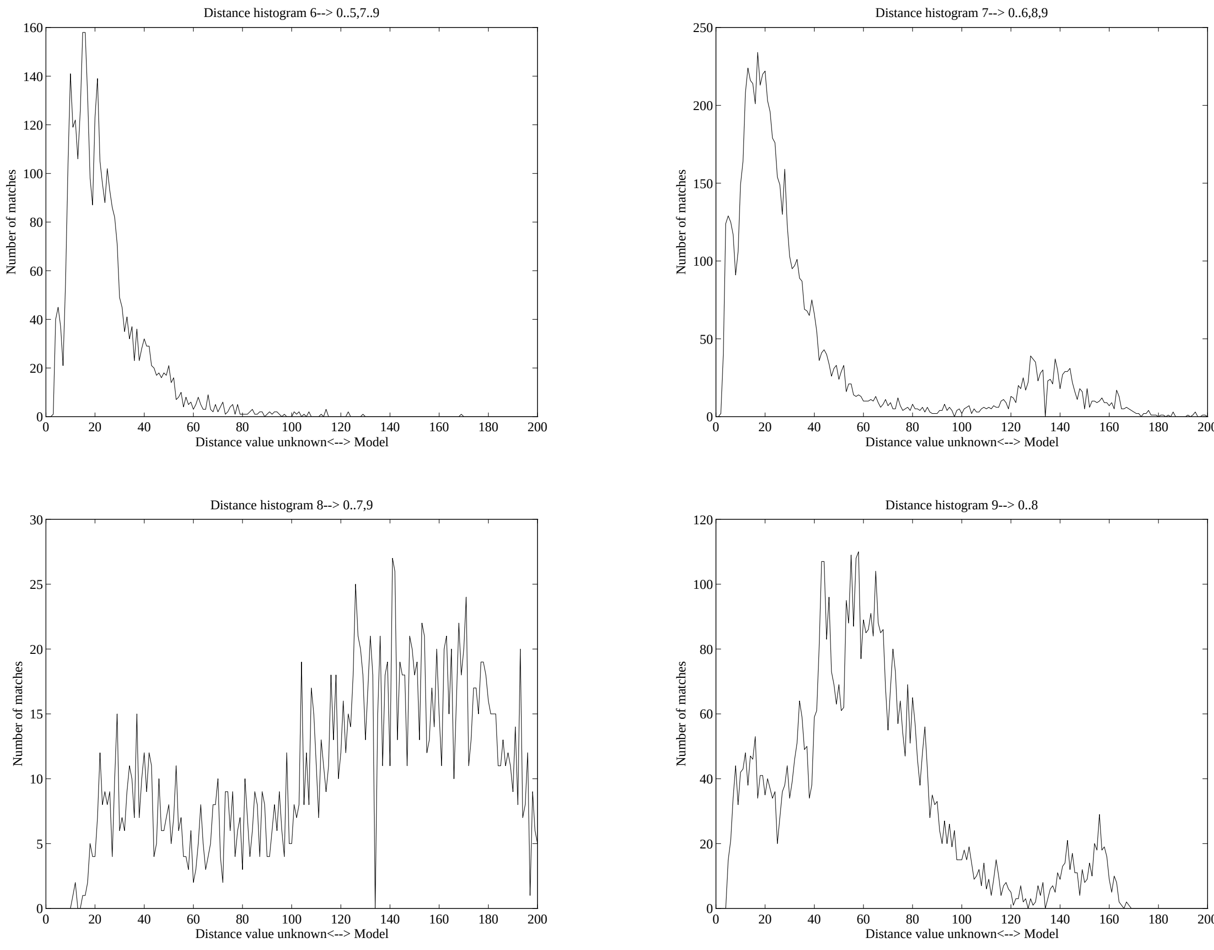

Figure 43: Distance of models versus all of the other patterns 6 to 9

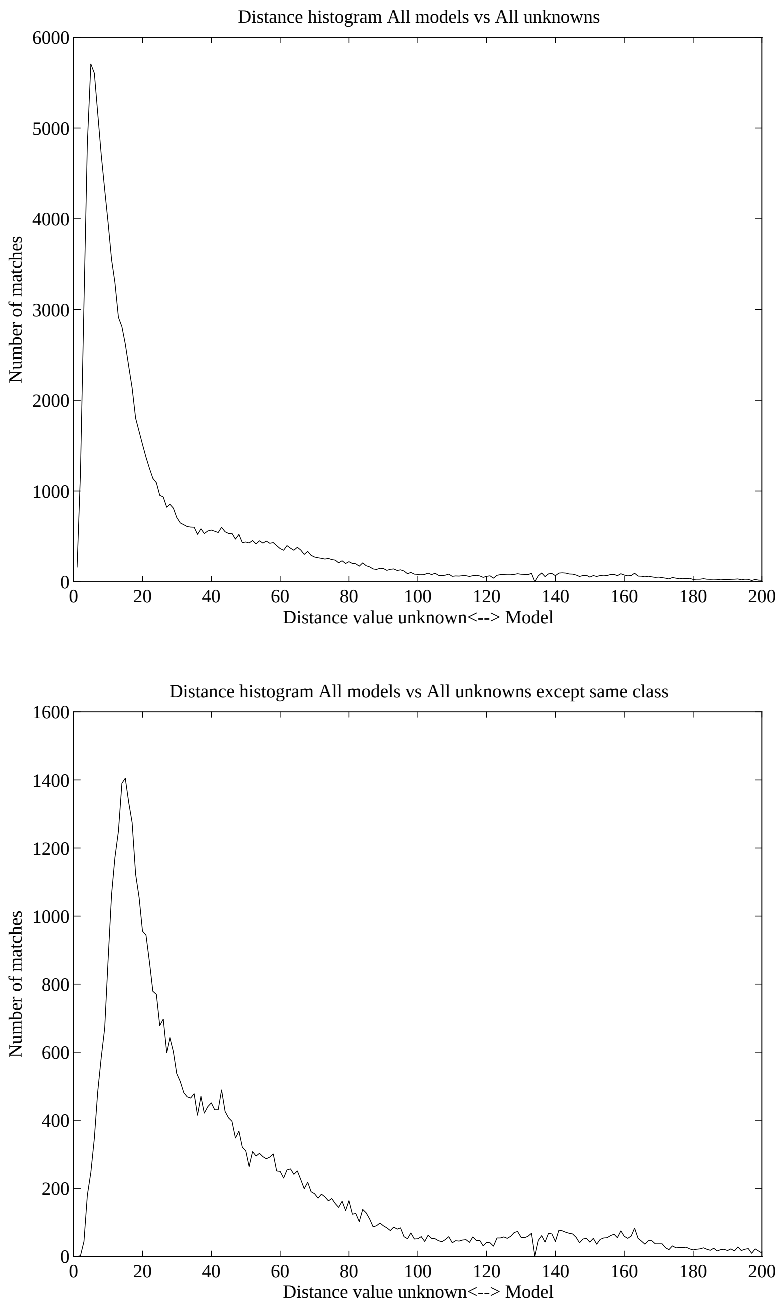

Figure 44: Distance of all models vs all unknowns.

We observe that there is a good percentage of patterns that fall below the threshold value when the models are matched against all the unknowns. The important thing to note is the number of patterns that are matched at two or three times the threshold value.

These numbers, and the shape of the distribution, indicate that there are a large number of unknowns that match at a distance close to the threshold value. We can then confidently use a maximum distance as a criterion for selecting the error class unknowns. We will obtain unknowns that are relatively close to the models. Also, the graphs indicate that there are relatively compact clusters of patterns to match with distances close to the threshold for most classes of models. Because of the compactness of these clusters, we believe that they represents a pool of potentially confusing characters. Because of their distribution we believe that the probability of getting an unrelated pattern is small, and this makes using a maximum threshold as a criterion a good choice. The data indicates that after a distance that is about three times the recognition threshold, the number of patterns that match is low, suggesting a natural cut-off point that we will use. We come to the conclusion that, in most cases, a multiple of the recognition threshold can be used as an upper limit for error pattern selection.

Chapter 5

Recognition Phase

5.1 Nearest Neighbour Classifier

We have referred to the recognizer several times without explaining how it works. Essentially, recognition is based on a nearest neighbour classifier. We keep a model base against which we measure the distance to the unknown. If the distance between an unknown and a given model is below a predetermined threshold, the pattern is recognized, as illustrated in Figure 45.

We should note that the original elastic matching algorithm described performed a breadth-first search instead of the depth-first which we implemented. The breadth-first approach evaluates the distance to each of the potential models, always evaluating the most promising model [100]. A minimal number of distance measurements are performed, since a match to completion will only be performed with the best match.

This is different from what we implemented. Since our distance measurement function only measures the distance to one pattern at a time, to find the best match we must completely match all the other possible patterns. This is a design decision made to simplify the distance measuring process. Doing so would allow us to have just one distance function for several purposes. We use elastic matching distance in several other processes for prototype establishment, such as threshold establishment. Having just one distance function makes code maintenance simpler, at the cost of needing more CPU for recognition.

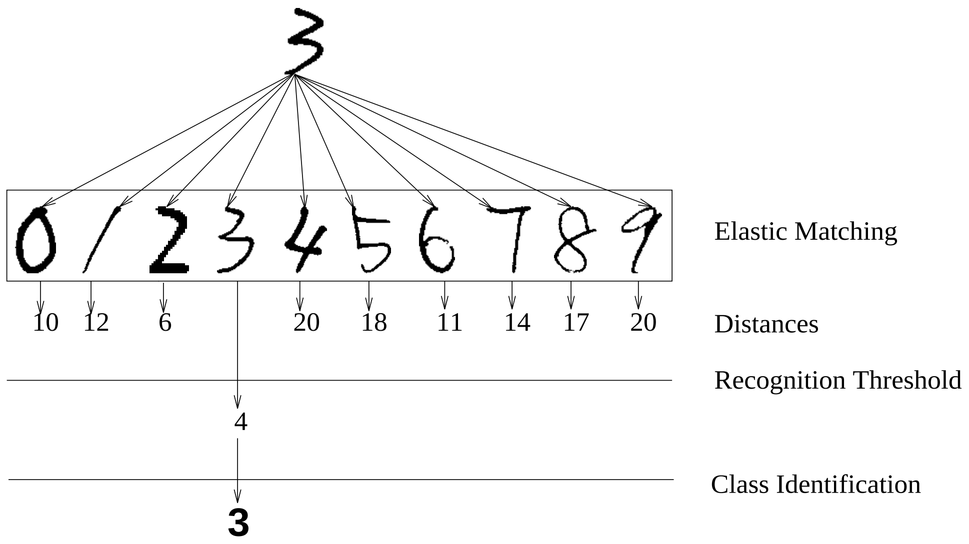

Figure 45: Overview of the nearest neighbour recognition process

5.2 Distance Measurement

As described above, each unknown is matched against each model. We will now describe steps in a typical match (Figure 46).

- **Model overlapping:** To minimize position dependence introduced by the Euclidian distance.
- **Pruning:** Eliminates models that are very far from the unknown in order to reduce computation.
- **Matching each stroke:** Match each compatible stroke of the unknown against the model.

5.2.1 Model Overlapping

The first step of the matching between an unknown and a model is to overlap them as well as possible. The measure of the distance function, mostly Euclidian, is position dependent. We must position the patterns in such a fashion so as to avoid the

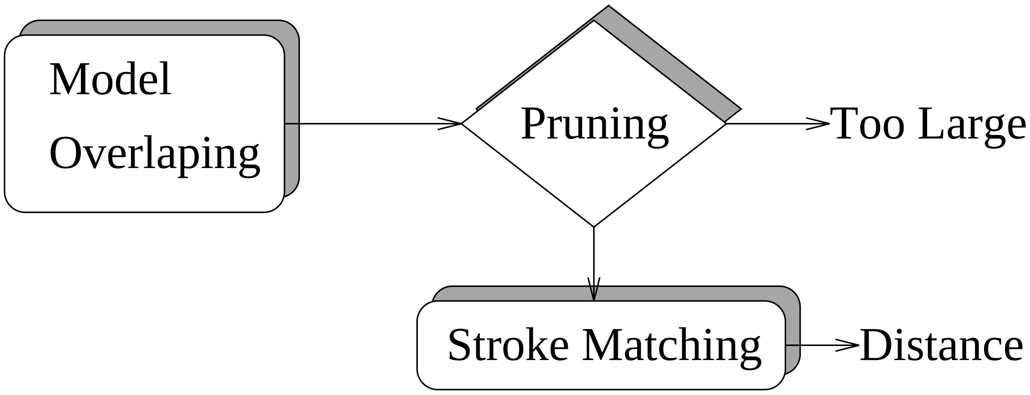

Figure 46: Distance measurement for an unknown

introduction of a bias, simply by the way the coordinate systems of the unknown and the model are aligned.

We overlap the patterns by overlapping their centre of gravity. The first step is to calculate the centre of gravity for the entire pattern. All of the points in the pattern, irrespective of the number of strokes, are treated equally.

$$cg_x = \frac{\sum_{i=1}^{n} x_i}{n} \tag{63}$$

$$cg_y = \frac{\sum_{i=1}^{n} y_i}{n} \tag{64}$$

The difference between the centre of gravity of the unknown and the model is then found, and the unknown is shifted to make the centre of gravity of both patterns coincide. In previous experiments we found that, when attempting to overlap similar, or almost identical patterns, the above procedure often produces an overlap that minimizes the Euclidian distance function. When the optimal solution was not found, it was often within one pixel, and almost always within two pixels, from the position found.

From these results we judge that this method is appropriate, since it is simple to implement, relatively inexpensive in terms of computation time, and the distance results are more than sufficient, considering all the other processing steps involved. Other methods, such as overlapping based on the centre of gravity of the bounding box, could also be considered. If the patterns are dissimilar, the distance between them ought to be large and optimal overlapping is not a concern.

5.3 Pruning

Although this view of recognition is accurate, it is computationally expensive. The first recognizer was implemented as we just described. Speed improvements are needed for two reasons. Recognition is never fast enough in an actual application, and overall faster recognition means cheaper recognition. What is more important, we often use recognition as a final measure of merit, to determine thresholds of various kinds, or to compare two different methods of performing an operation. Using the program profiler *prof* available from Unix, we found that the program spends upwards of 80% of total CPU time in the elastic matching function. This shows that whatever improvements are made should be concentrated around eliminating, as much as possible, comparisons between strokes with elastic matching.

Most of the speed improvements described by Tappert [103] revolved around pruning the search tree by finding cheap mechanisms to determine whether elastic matching should be performed between an unknown and a given candidate. Two main forms of pruning were described. At the stroke level, strokes that were too long or too short are not matched. This is reasonable since these matches have little chance of yielding a good distance. As well, a crude linear match is applied to determine which pairings of strokes are good candidates. At the character level, unknowns are matched against models having the same number of strokes.

5.3.1 Implemented Pruning

We have implemented pruning mechanisms similar to Tappert's. At the stroke level we have kept the size constraints. To match, the length of the strokes in points, must be within the following constraints:

$$\frac{LEN(seq_{mdl})}{2} <= LEN(seq_{unkown}) <= LEN(seq_{mdl}) * 2 \qquad (65)$$

where LEN denotes the length of the stroke in points. The introduction of this rule did not significantly influence the recognition rate.

The second pruning technique used consisted in matching the first, last and mid points of strokes of the unknown against strokes of the model. If the resulting distance is below a given threshold, the patterns are considered a good enough match to be kept for further processing. This filter could use some improvements. It does match a large number of strokes against the other. Even though only three points are matched, a large number of distances $d(i, j)$ are computed. An enhancement to this method could be to match only the first one or two longest strokes of a given pattern, thereby eliminating a possible explosion of combinations.

The threshold against which the resulting distance is compared is derived experimentally. Successive recognition experiments were conducted with different thresholds, but with a fixed model base, recognition threshold and test set. A threshold was selected so as to only slightly impact on recognition, but with a value low enough to eliminate most patterns. The current threshold eliminates around 85% to 90% of the potential candidates over an entire recognition experiment. The recognition algorithm doubled in speed with the introduction of this preprocessing. As an alternative, we also attempted to match four points instead of just three. The results were similar in terms of the number of models retained for an equivalent recognition rate. Performance did not justify the added computations performed.

Finally, at the model level, we match a model against an unknown only if the number of strokes from the unknown is greater or equal to the number of strokes in a model. This is different from Tappert [103] and was done to allow for noisy unknowns. It is likely that, with scanned data, the unknowns will contain some small noise spurs. They can be caused either by the digitization process, or by the thinning algorithm. They are, however, frequent enough to warrant attention.

By allowing the match of an unknown against a model with fewer strokes, it gives us the possibility of ignoring these spurious branches by simply not matching them at all. Because of the stroke length constraint imposed, a stroke may not be matched. To make sure that we do not ignore significant portions of the unknown, we require that more than 80% of the points of an unknown be matched with a stroke of a model.

Moreover, we also impose that more than 80% of the model strokes be matched. A model stroke counts toward this total only once, therefore preventing that one stroke, by being matched twice, to fool the system into thinking that the entire model has been matched. Imposing such a restriction ensures that most, if not all, of the model is matched. It also gets rid of possible confusion between off-line two stroke 7's and one stroke 1's for instance.

We should note that this noise problem is not present with the on-line data but, for simplicity, we make no distinction between the data in terms of matching rules. With on-line data, forcing a match of more than 80% is roughly equivalent to the constraint imposed by Tappert in terms of matching an unknown against same stroke count models, but does impose a runtime penalty.

5.3.2 Stroke Matching

We attempt to match each stroke of the unknown against the strokes of the model, regardless of whether they were previously matched or not. This is because we allow unknown strokes to match a model stroke more than once to handle digitization noise. This is most useful for off-line data recognition that has not undergone stroke reconnection. With stroke reconnection, the probability of such an occurrence is remote. There are better chances that the noise stroke won't be matched at all because it is too small.

5.4 Perceptron

When we modified Elastic Matching to add weights, we derived Formula 55 which describes a linear classifier. These kinds of classifiers have been known for a while and the most popular example is the *Perceptron*. Such similarity warranted some investigation into the perceptron. The perceptron is the first of a class of adaptive algorithms known as Neural Networks. We draw most of this historical perspective presented here from the collected reprints and the accompanying introductory notes published by Anderson and Rosenfeld [3].

Some attribute the first work with simulated neurons to McCulloch and Pitts [74]. They mathematically demonstrate that, with networks of Boolean neurons, one can build a machine that has the same computational power as a Turing machine. Later, Rosenblatt presents a model of a trainable system built of several layers of simply connected cells. There is an input layer, called the retina, composed of cells connected to an association layer, which are connected to a response layer. The connections, which are made of weights, are adapted, as training progresses, to make the response layer activate the one cell that represents the proper result for the pattern presented to the retina [86]. The weights are then modified to improve the results. The system is trainable and capable of generalization, or what can be called learning. A description of the learning mechanism used by the perceptron actually built is made by Block [14]. It was later proved, in the book by Minsky and Papert, that the perceptron could not correctly discriminate clusters that are not linearly separable, which proves to be a major drawback.

We will use the description of the simple two-layer perceptron found in Hertz and al. [47]. This perceptron has one input layer and one output layer, but no hidden layers. The recognition phase of this perceptron can be mathematically described as follows:

$$O_i = g(\sum_{\kappa} w_{i,\kappa} S_{\kappa}) \tag{66}$$

Where $g(h)$ is an activation function, either a sigmoid or a simple threshold function, S is the input at position κ and w is a weight associated with the position

κ.

The training mechanism needed to establish weights can be described as follows:

$$w'_{i\kappa} = w_{i\kappa} + \Delta w_{i\kappa} \tag{67}$$

where the definition of $\Delta w_{i\kappa}$ can vary and corresponds to various learning rules. In most cases $\Delta w_{i\kappa}$ is made of two components, a learning rate η and an error function, based on the difference between the result produced by the algorithm and the expected value. A simple case is:

$$\Delta w_{i\kappa} = \eta(\zeta_i^\mu - O_i^\mu)\xi_\kappa^\mu \tag{68}$$

where ζ represents the target output, O output obtained, ξ representing the input vector for a given pattern μ. By convention, the outputs are often binary: $1, -1$. This formula simply adds a fraction of the current input vector to the weights when the output is in error. The perceptron learning rule used by Rosenblatt, Minsky and Papert is expressed as follows:

$$\Delta w_{i\kappa} = \eta\Theta(N_\kappa - \zeta_i^\mu h_i^\mu)\zeta_i^\mu \xi_\kappa^\mu \tag{69}$$

where Θ is a step function, N_κ is the threshold for the size of the total vector and where h_i^u is defined as follows:

$$\zeta_i^\mu h_i^u \equiv \zeta_i^\mu \sum_\kappa w_{i\kappa}\xi_\kappa^\mu > N_\kappa \tag{70}$$

Note that in the case where there is only one output element, Equation 69 can be more simply expressed as:

$$\Delta w = \eta\Theta(N_\kappa - w \cdot x^\mu)x^\mu \tag{71}$$

given the following change to Equation 70 to

$$w \cdot x^\mu > N_\kappa \tag{72}$$

We see with Equation 71 that we are still essentially adding some portion of the input vector as a weight increment. The perceptron learning algorithm modifies these weights by adding a fraction of the current node contribution to the recognition result. This contribution is measured by $w \cdot x^{\mu}$, the same coefficient used in the recognition.

5.4.1 Elastic Matching and Perceptron

The simple perceptron described above can be compared to the definition of elastic matching given by Equation 57. Both use two functions, one for training and one for recognition. Furthermore, these functions are similar in both cases. A point can be made that the perceptron algorithm is composed of two distinct functions and that any departure from them yields a different algorithm. That is the case, but both algorithms share enough characteristics to warrant a closer look.

The recognition function is essentially the same in both cases. Both are linear sums of coefficients which, in the case of the neural net, is traditionally the pattern itself, without any preprocessing or feature extractions. However, in our case, the input pattern can be considered as a feature vector. The feature is composed of the distance between points of the unknown and a model. This is the main difference between both algorithms, as far as recognition is concerned. Both use the same algorithm, but elastic matching performs recognition with a different feature vector.

The training phase does present true differences, but it also has some similarities. The algorithms are similar, as both use a function evaluating the error caused by each node to calculate their new weights, and do so iteratively. The feedback functions are dissimilar in several ways. We perform some operations on the input vectors before calculating the weights, instead of modifying the weights directly with each pattern. The patterns involved in these computations are known to be closely related to the model, which is not necessarily the case with neural networks. So, as much as the training phase is different, it does share some characteristics in common with the perceptron. It does, for instance, exhibit the characteristic of the perceptron training algorithm of going from a good solution to a worse one at any given iteration [38], a shortcoming that various algorithms try to overcome. Our algorithm, though, provides a good solution at the first iteration, requiring far fewer iterations than the perceptron training algorithm, to attain a near optimal solution.

5.4.2 Weights Interpretation

There is a direct interpretation that we can make of the weights generated by the weighted elastic matching algorithm. The higher the weight, the more important this portion of the character is to distinguish between two confusing classes. Because we can attach a weight to a specific portion of the character, say the top cavity or the right corner, we can also get an intuitive feel for which part of the pattern is important.

Knowing which part to emphasize over another is what most algorithms try to accomplish, and what is often performed by humans [66]. Successful handwriting recognition expert systems have been built, based on this principle [98]. Our weights now provide some crude way of emphasizing portions of patterns.

The question, then, is: can we derive some sort of interpretation from these automatically generated weights? For example, if the system places an emphasis on the top cavity, we would like it to be able to explicitly state this preference in some way. It would help the designer of a character recognition expert system, to concentrate on the features that are useful and most likely to improve recognition.

This could be derived directly from the weights provided for each model. Such a system has been proposed by Gallant [37]. Using data from an Acute Sarcophagal Disease database, composed of several hundred cases, with different variables, such as symptoms, diseases and treatments, it attempts to build a system, using neural networks, that behaves like an expert system. The neural network is composed of a layer of symptoms, linked to a layer of diseases, that is linked to the last layer composed of treatments. Network weights are found using the appropriate neural network training algorithm. The system exhibits most characteristics associated with expert systems; it has inferencing and a knowledge base that separates the knowledge from the inference algorithm. It also has the capacity of justifying its conclusions, generating explanations based on the neural network nodes participating in the conclusion and their associated weights.

The advantage of such a system is in its automated knowledge-acquisition properties. By training the neural network with traditional neural network algorithms, knowledge is built into the system. We think it is quite appropriate with this field, as it fills most of the conditions stated by Gallant [37]:

> *We believe connectionist expert systems represent a promising approach to the knowledge-acquisition problem for expert systems. These methods are most appropriate for classification problems in environments where the data is abundant and noisy, and where, conversely, humans tend to generate brittle and perhaps contradictory* **if-then** *rules.*

The field of character recognition relates well to this description. Expert systems for pattern recognition are complex, sometimes brittle and tedious to build. We believe this presents an interesting opportunity. We have a recognition system that is similar in numerous ways to neural networks. On the other hand, a system is described where neural networks are used to create an expert system. We have a field where expert systems have been used, but where knowledge elicitation is very difficult.

Although our system does not have as many layers as described in the system by Gallant [37], we believe that some sort of merging of both systems could prove promising. An intermediate layer may need to be added to create a multilayer neural network. Other features, apart from the Euclidian distance slope and curvature, may need to be added to the feature vector. But the benefits would be a system capable of explaining its classification decisions. The explanations provided may be difficult to interpret and to convert into features such as the ones used by character recognition expert systems. The system could, however, provide good guidelines to the design of a character recognition expert system on the features needed. If anything, it could prove helpful to extend the character recognition expert systems to larger character sets. Inversely, a character recognition expert system designer could see the weaknesses in the rules generated and point out where this system needs improvement or better feature measurements. Hopefully by improving the neural network expert system to behave more like a real expert, on a limited character set like digits, we would give new skills to the neural network that could make it useful for larger problems such as the entire alphabet. It may also be possible to combine the benefits of a trainable system, mainly the ease to adapt to new character shapes, and the benefits of the character recognition expert systems, primarily the capacity of fine tuning, to achieve the best results and take care of extreme cases.

Chapter 6

Results

A large number of recognition experiments were performed with the various databases that we described earlier. The purpose of these experiments is to determine if the improvements we made behave as expected. Specifically, which weights improve the elastic matching algorithm and secondly, whether stroke reconnection would improve the results with off-line data. We attempt to compare our results with other published results when possible.

6.1 Experiments

The experiments are conducted using the iterative training technique described in section 4.6.1. The recognition thresholds used for the clustering stage are the optimal recognition threshold found at the previous iteration, while the recognition results are made with the optimal thresholds found for that particular iteration.

The experimental setup provides us with two recognition rates, one against the training set and one against a test set. The test set is disjoint from the training set and there should be no bias favouring the test set. The results against the training set have some inherent bias. With weighted elastic matching the weights are calculated with the unknowns from the same set, the thresholds are optimized for that specific set, and depending on the set, the same limited number of writers.

Since we do not synthesize models, the models we have still have the same exact shape in the training and testing set. Because they both have the exact same shape, the model will match exactly to its corresponding unknown, resulting in a distance of zero. We detect the zero distance and reject it, knowing that we obtained a match of the model against itself. Such recognition against the training set follows the *take one out* paradigm. Doing so removes the obvious advantage in recognizing the training set while making the best of a limited database. Recognition experiments bear this out as we usually don't show a large difference between the recognition rate of the test and training sets.

6.2 Recognition Tables

There are, in our case, five possible outcomes for the recognition process. A pattern is considered:

- **Classified:** When the identification found by the classifier matches the one given in the database.
- **Substitution:** When the identification returned from the classifier is different from the one given in the database.
- **Confusion:** When more than one identification is possible, the classifier does not attempt to make a decision.
- **Rejection:** When no identification is found by the classifier.
- **SecondBest:** A confused pattern where one of the possible identifications is the valid one.
- **Reliability:** $Classified/(Classified + Substitution)$.

We add a confusion class because of the nature of the algorithm; it is possible for an unknown to have a low enough distance to models of more than one class. This distinction is useful because it hints at the nature of the recognition problem. It can be that the pattern is at the border between two classes. Alternatively, it could be a pattern that is a significant departure from the known set of patterns, in which case it will be a rejection. Confusion is just a different variety of rejection.

These results depend on the patterns being identified properly. There is evidence that the database we used contained some confusing patterns. Not that the data is flawed, but simply that humans can't seem to agree on the proper label for some patterns, the patterns themselves are confusing [65, 66]. For our purposes, we ignore pattern mislabelling and consider that the identity of the pattern is correct.

We would like to point out that we always get Second Best results and never any that are strictly Confusion. This tells us that when the recognizer is confused, with the choice between at least two classes, one of them will be the correct class. Unfortunately we have not kept any statistics on the percentage of times where the match with the lowest distance is indeed of the correct class. We therefore cannot say, if we were to have the algorithm give both results with an associated probability, how often this would result in a correct guess.

Reliability

We use a standard definition of reliability that is defined as follows:

$$Reliability = Classified/(Classified + Substitution) \tag{73}$$

The reliability figure can be misleading. A high reliability can be easily achieved by reducing the recognition rate. The reverse is also true, a high recognition rate does not imply a perfect system, especially if it confuses everything it does not recognize properly. For this reason, in evaluating a recognition process, both reliability and the recognition rate are needed.

Recognition Speed

The speed of the recognition system is of interest, simply because it is a good indicator of the resources needed for recognition. We performed the recognition experiments with a **DECstation 5000/200** running Ultrix. The speed benchmarks we performed show this machine to be comparable to a **SUN Sparc 10**. The system can recognize between 40 to 70 characters per second, depending on the size of the model base used. Another factor influencing the speed is the number of strokes found in the patterns. This influences the speed because the pruning algorithm has to look at every stroke, regardless of its size, increasing the computations performed. For this reason the processing of reconnected off-line is slower, around 40 cps., showing that suboptimal reconnection creates a database with more strokes per chars. On-line, on the other hand, is favoured because most characters have the minimum number of strokes, most often just one. This speed measurement includes I/O to read the unknown and the generic preprocessing step, but it does not include the preprocessing needed to convert the off-line data into on-line. For the off-line data, the skeletonization stroke tracing and stroke reconnection are performed ahead of time.

6.3 On-line Database

We now present the recognition results for the on-line database with weights added, in a separate section we will compare the results against regular elastic matching. In this section we will also compare the results with those obtained by Malowany with the same database. We will also give the results for the re-split version of the database.

6.3.1 Comparison with Malowany

We performed a recognition experiment to compare our method against the one used by Malowany [73]. We used the original on-line database we trained with the 1000 digit training set and tested against the 2300 digit test set. We actually tested against 2299 digits since an untraced bug prevents the recognition program, although it completes the recognition phase, to terminate correctly. The results are compared in Table 11. Table 9 gives our results for the on-line training set where we observe that substitution occurs only between "9" and "4". Table 10 gives our result for the testing set.

Both algorithms show a large differential of the recognition rate between the test

Result	0	1	2	3	4	5	6	7	8	9
0	95	0	0	0	0	0	0	0	0	0
1	0	89	0	0	0	0	0	0	0	0
2	0	0	97	0	0	0	0	0	0	0
3	0	0	0	98	0	0	0	0	0	0
4	0	0	0	0	97	0	0	0	0	1
5	0	0	0	0	0	94	0	0	0	0
6	0	0	0	0	0	0	98	0	0	0
7	0	0	0	0	0	0	0	85	0	0
8	0	0	0	0	0	0	0	0	89	0
9	0	0	0	0	1	0	0	0	0	95
Reject	5	8	3	2	2	5	1	11	11	2
Confused	0	3	0	0	0	1	0	5	0	2

Table 9: On-line data, original training set against itself

Properly classified :	937	93.70
Confused :	11	1.10
Rejected :	50	5.00
Substitution:	2	0.20
SecondBest:	11	1.10
Reliability:	937	99.79

set and the training set caused by the differences between them. Because of this problem, the recognition rate for the test set is much lower than the recognition rate for the training set. This explains, in part, why the best recognition rate for both sets are found at different iterations. Our recognition results are comparable to ones obtained by Malowany, but it is unclear as to which will be easier to extend. However since our algorithm is fully trainable we expect it to be easier to adapt to new data.

6.3.2 Results with Re-split Database

Because of the problem of representativity with both the training and testing sets of on-line data, we merged the sets and then divided them into two equal sets. Unfortunately this gives two sets with the same writers. It does, however, provide two sets that are more representative of the various writing styles. The results are given in Tables 12 and 13.

Result	0	1	2	3	4	5	6	7	8	9
0	183	0	0	0	0	0	1	0	0	0
1	0	200	0	0	0	0	0	3	0	0
2	0	2	196	0	0	0	0	2	0	0
3	0	1	0	208	0	0	0	0	0	2
4	1	0	0	0	213	0	0	3	0	2
5	0	0	0	0	0	183	0	0	0	1
6	1	0	0	0	1	0	192	0	0	0
7	0	1	1	0	0	0	0	143	0	0
8	0	0	0	0	0	0	0	0	195	0
9	0	0	0	0	0	0	0	0	0	158
Reject	45	22	33	22	12	47	37	73	35	66
Confused	0	4	0	0	0	0	0	4	0	1

Table 10: On-line data, original training set against original test set

Properly classified :	1871	81.35
Confused :	9	0.39
Rejected :	392	17.04
Substitution:	28	1.22
SecondBest:	9	0.39
Reliability:	1899	98.53

Both best results, train against itself and train against test, were obtained at the same iteration with 293 models. Because both sets contain the same writers we can't qualify these results as writer independent. It is, however, the best we can do to have the widest coverage of the possible shapes given the database available to us. These are about the best results we can obtain, the database being fairly representative of various writing styles.

Method	Recognition	Substitution	Reliability
Malowany train	96.60%	0%	100%
Scattolin train	93.70%	0.20%	99.79%
Malowany test	81.30%	1.17%	98.42%
Scattolin test	81.35%	1.22%	98.53%

Table 11: Recognition rate comparison between Malowany and our algorithm

Result	0	1	2	3	4	5	6	7	8	9
0	148	0	0	0	0	0	0	0	0	0
1	0	146	0	0	0	0	0	0	0	0
2	0	2	146	0	0	0	0	0	0	0
3	0	0	0	152	1	1	0	0	0	0
4	0	0	0	0	150	0	0	0	0	0
5	0	0	0	0	0	153	0	0	0	0
6	1	0	0	0	0	0	154	0	0	0
7	0	1	0	0	0	0	0	150	0	0
8	0	0	0	0	0	0	0	0	149	0
9	0	0	0	0	0	0	0	0	0	137
Reject	14	12	16	12	13	10	9	13	15	16
Confused	2	4	3	1	1	1	1	2	1	12

Table 12: Confusion Matrix for On-line re-split, train against itself

Properly classified :	1485	90.05
Rejected :	130	7.88
Substitution:	6	0.36
SecondBest:	28	1.70
Reliability:	1491	99.60

6.3.3 Tappert's Results

We can't directly compare our results to those obtained by Tappert for several reasons. First, we don't have the same database, precluding any direct comparisons. Secondly, for most of Tappert's experiments the recognizer was trained for a particular writer, giving a writer-dependent system. Our system, in comparison, is an attempt at a writer independent system but over a more limited alphabet, where we encounter a wider variation of shapes for any given character, as well as possible inter-writer confusions, for example in the case of the European "1" and American "7".

We are confident that both recognition systems are fairly equivalent and should exhibit the same behaviour if placed in the same context. For this reason we believe that the improvements made with weighted recognition should also be applicable to a wider alphabet in a writer-dependent system.

Result	0	1	2	3	4	5	6	7	8	9
0	137	0	0	0	0	0	0	0	0	0
1	0	143	0	0	0	0	0	0	0	0
2	0	1	143	0	0	0	0	1	0	0
3	0	0	0	153	0	0	0	0	0	0
4	0	0	0	0	155	0	0	0	0	0
5	0	0	0	0	0	151	0	0	0	0
6	1	0	0	0	0	0	155	0	0	0
7	0	1	1	1	0	0	0	144	0	0
8	0	0	0	1	0	0	0	0	149	0
9	0	0	0	0	1	0	0	0	0	134
Reject	25	17	17	9	10	12	8	16	15	20
Confused	2	3	4	1	0	2	2	4	1	11

Table 13: Confusion Matrix for On-line re-split, train against test

Properly classified :	1464	88.67
Rejected :	149	9.02
Substitution:	8	0.48
SecondBest:	30	1.82
Reliability:	1472	99.46

6.4 Off-line Database

As we mentioned, we also performed recognition with off-line data. Two types of off-line data were attempted, before stroke reconnection and after stroke reconnection. Tables 14 and 15 give the recognition results for training set **A** and the testing set without reconnection. Tables 16 and 17 give the results with the same database but with reconnected strokes. Both of these experiments were performed with the weighted elastic matching algorithm using iterative training.

Similar results are also obtained with training set **B**. As we can see there is a big difference between the training set and the test set. We attribute this to the fact that both sets are built with different writers. To improve the results we attempted to train with both sets at the same time, but we encountered some programming problems that could not be solved quickly and we did not pursue the experiment.

We are interested in knowing if stroke reconnection did improve the performance

Results	0	1	2	3	4	5	6	7	8	9
0	191	0	0	0	0	0	0	0	0	0
1	0	195	0	0	0	0	0	0	0	0
2	0	1	152	3	0	0	0	0	0	0
3	0	0	1	144	0	1	0	0	0	0
4	0	0	1	0	135	0	0	1	0	0
5	0	0	0	2	0	156	0	0	0	5
6	0	0	0	0	0	1	183	0	0	0
7	0	0	0	0	0	0	0	171	0	0
8	0	0	0	2	0	0	0	0	164	0
9	0	0	0	0	1	0	0	1	1	180
Reject	9	3	36	36	54	38	14	17	30	9
Confused	0	1	10	13	10	4	3	10	5	6

Table 14: Confusion Matrix, Off-line unreconnected training A against itself

Properly classified :	1671	83.55
Confused :	62	3.10
Rejected :	246	12.30
Substitution:	21	1.05
SecondBest:	62	3.10
Reliability:	1671	98.76

of the recognition algorithm. By comparing the tables from both experiments we see that recognition has increased slightly with stroke reconnection, but reliability has suffered slightly. But another good measure of the performance of the algorithm is the size of the model base. In the experiment without stroke reconnection, 423 models were used to achieve 83.55% recognition, while the experiment with the stroke reconnection uses 376 models to achieve a recognition rate of 84.30%. If the increase in recognition rate is slight, the simultaneous reduction in the size of the model base is more significant.

From this we can say that stroke reconnection does contribute to the generality of the algorithm, not necessarily by increasing the recognition rate significantly but by using fewer models. Stroke reconnection also serves to speed up the recognition process. Without stroke reconnection, the recognition experiment takes 1'58" for 2000 unknowns. With the stroke reconnection algorithm the recognition experiment takes 40" for 2000 unknowns using the same data. We observe almost a five fold

Results	0	1	2	3	4	5	6	7	8	9
0	177	0	0	0	0	0	3	0	0	0
1	0	193	0	0	0	0	0	0	0	0
2	0	0	143	2	0	0	0	0	0	0
3	0	0	0	139	0	0	0	0	1	0
4	0	0	0	0	129	0	0	2	0	0
5	0	0	0	0	0	166	0	0	0	5
6	0	0	0	0	0	0	171	0	0	0
7	0	0	0	1	1	0	0	158	0	1
8	0	0	2	0	0	0	0	0	156	0
9	0	0	0	0	3	0	0	1	0	163
Reject	23	5	48	51	50	27	14	23	37	27
Confused	0	2	7	7	17	7	12	16	6	4

Table 15: Confusion Matrix, Off-line unreconnected training A against test set

Properly classified :	1595	79.75
Confused :	78	3.90
Rejected :	305	15.25
Substitution:	22	1.10
SecondBest:	78	3.90
Reliability:	1595	98.64

reduction in the time needed to perform the experiment. This, in itself, is a significant improvement, making the algorithm more practical.

The stroke reconnection process improves the performance of the algorithm by using fewer models, making each of them more general, and by reducing the time needed to process the patterns. Our algorithm does not perform as well as the other published algorithms that used the same database. The algorithm most closely related is by Nadal as both make use of information gathered strictly from the skeletons. Nadal achieves 86.05% with a substitution rate of 2.25% [96]. Nadal also happens to be the one with the lowest published results for this given database. It then seems that the use of just skeleton information does penalize the algorithm, by getting rid of the width information available from the contour.

We also want to know if the weights improve the results with the off-line data. Comparing Tables 16, 17 against 18, 19 we see that the weights do improve the recognition results.

Results	0	1	2	3	4	5	6	7	8	9
0	195	0	0	0	0	0	0	0	0	0
1	0	192	0	0	0	0	0	0	0	0
2	0	0	167	0	0	1	0	0	0	0
3	0	0	0	146	0	0	0	1	0	0
4	0	0	0	0	165	0	0	1	0	2
5	0	0	0	0	0	149	0	0	0	5
6	0	0	0	0	0	0	188	0	0	0
7	0	0	0	0	0	0	0	162	0	0
8	0	0	0	0	0	0	0	0	159	0
9	0	0	0	0	1	0	0	0	0	163
Reject	4	7	27	37	27	36	12	19	38	26
Confused	1	1	6	17	7	14	0	17	3	4

Table 16: Confusion Matrix for Reconnected training set A against itself

Properly classified :	1686	84.30
Confused :	70	3.50
Rejected :	233	11.65
Substitution:	11	0.55
SecondBest:	70	3.50
Reliability:	1686	99.35

6.4.1 Off-line Recognition with On-line Model Base

By accident, we ran a recognition experiment with a model base built from on-line patterns while the test set was made of unreconnected off-line patterns. The results were terrifying, about 40% of the “0” were recognized while nothing else was. Once the error was discovered the results became a pleasant surprise. We could actually recognize off-line patterns with an on-line database.

We later pursued this experiment to see what kind of recognition rate we can have with reconnected off-line data against an on-line model base. The experiment is set as follows. The training proceeds with the a weighted on-line re-split training data base, while the testing is performed with the off-line. Later, two different testing runs are performed. For the first test phase we use the recognition threshold found to be optimal for the previous iteration with the on-line data. This gives recognition results of around 40%. We then improve recognition by finding the optimal recognition

Results	0	1	2	3	4	5	6	7	8	9
0	179	0	0	0	0	0	3	0	1	0
1	0	191	0	0	0	0	0	0	0	0
2	0	0	156	2	0	0	0	0	0	0
3	0	0	0	150	0	0	0	0	0	0
4	0	0	0	0	160	0	0	1	0	1
5	0	0	0	2	0	152	0	2	0	2
6	0	0	0	0	0	0	182	0	0	0
7	0	0	0	0	0	0	0	137	0	0
8	0	0	0	0	0	1	0	0	149	0
9	0	0	0	0	8	2	0	1	0	147
Reject	20	8	35	33	27	32	12	21	49	41
Confused	1	1	9	13	5	13	3	38	1	9

Table 17: Confusion Matrix, Off-Line Reconnected set A against reconnected Test set

Properly classified :	1603	80.15
Confused :	93	4.65
Rejected :	278	13.90
Substitution:	26	1.30
SecondBest:	93	4.65
Reliability:	1603	98.40

threshold to use for that model base with the off-line unknowns; this is done employing the method normally used to find thresholds, improving the recognition results. These new thresholds are then used for the next training iteration, the best iteration over a set of 25 gives the results below in Table 20. The best recognition results obtained are at 48.30% while the average result oscillates around 47.5%.

At first glance these results are disappointing, but we must remember that the two databases are completely disjoint. Summary inspection reveals that the type of writing present in both sets is different, the on-line model base containing more European characters. Secondly, we note that the recognition rate is not uniform across the classes. Some classes, such as "3" perform abysmally. To explain this we must come back to the stroke reconnection explained before. As we saw, because we are not processing the stroke retrace, we have, for certain classes, a much lower number of optimal characters after the merge, for example the middle bar of the "3"

Classification Results

Results	0	1	2	3	4	5	6	7	8	9
0	193	0	0	0	0	0	0	0	0	0
1	0	190	0	0	0	0	0	0	0	0
2	0	1	156	2	0	1	1	0	0	0
3	0	0	0	123	0	0	0	0	0	0
4	0	0	0	0	133	0	0	2	0	1
5	0	0	0	3	0	155	0	0	0	7
6	0	0	0	0	0	0	187	0	0	0
7	0	0	0	0	0	0	0	138	0	1
8	0	0	0	0	0	0	0	0	156	0
9	0	0	0	0	1	0	0	1	0	146
Reject	6	6	27	44	26	28	10	24	41	24
Confused	1	3	17	28	40	16	2	35	3	21

Table 18: Confusion Matrix, Off-line reconnected no weights, train A against itself

Properly classified :	1577	78.85
Confused :	166	8.30
Rejected :	236	11.80
Substitution:	21	1.05
ReallyConfused :	166	8.30
Reliability:	1577	98.69

is almost always reconnected with either of the two strokes, yielding a two stroke character. Unfortunately most of our on-line "3" are one stroke. So even if the first part of the unknown "3" matches well, the second part will almost constantly match poorly. This, we believe, explains the wide discrepancies between the various classes. We should also note that stroke reconnection performs other small tasks such as loop reopening and stroke reordering that can negatively impact on these results. Further experiments where both the model base and the unknowns were uniformly subjected to these two steps could be performed.

6.5 Model Contribution to Recognition

A side benefit from the on-line off-line recognition experiment was to give us a model base, for the same problem, that was suboptimal. As a result, recognition was lower, Classification Results

Results	0	1	2	3	4	5	6	7	8	9
0	179	0	0	0	0	1	3	0	1	0
1	0	187	0	0	0	0	0	0	0	0
2	0	0	148	2	0	0	0	1	0	0
3	0	0	1	133	0	0	0	0	1	0
4	0	0	0	0	146	0	0	1	0	1
5	1	0	0	0	0	151	0	0	0	0
6	0	0	0	0	0	0	181	0	0	0
7	0	0	0	0	0	0	0	128	0	0
8	0	0	0	0	0	0	0	0	144	0
9	0	0	0	0	7	1	0	2	1	139
Reject	20	11	37	43	20	24	12	29	51	41
Confused	0	2	14	22	27	23	4	39	2	19

Table 19: Confusion Matrix, Off-line reconnected, no weight, train against test

Properly classified :	1536	76.80
Confused :	152	7.60
Rejected :	288	14.40
Substitution:	24	1.20
ReallyConfused :	152	7.60
Reliability:	1536	98.46

around 85%, with a smaller model base. The interesting portion of the experiment is the contribution towards recognition of the extra models found in the optimal version as given in Table 21 where we repost the recognition results for the on-line re-split training set against itself under both conditions (optimal and suboptimal).

As we see, the optimal model base recognizes 39 more unknowns but adds 42 models. The added models don't even contribute towards the recognition of one character each. We can draw the conclusion that the last models added have little influence on recognition, even with weighted elastic matching. This confirms what we know of the cluster sizes. When the clusters are created, there are a few that will contain a relatively large number of patterns; while there will be many small clusters of one or two patterns, these models did not generalize well.

Identity	0	1	2	3	4	5	6	7	8	9
0	151	0	0	0	0	0	0	0	2	0
1	0	95	0	0	0	0	0	1	0	0
2	0	0	73	0	0	1	0	0	0	0
3	0	2	0	20	0	0	0	0	0	0
4	0	0	0	0	95	0	0	2	1	22
5	0	0	0	1	0	76	0	0	0	2
6	0	0	0	0	0	0	152	0	0	0
7	0	0	0	1	0	0	0	112	0	1
8	6	0	0	0	0	0	0	0	125	0
9	0	0	0	0	0	0	0	0	0	67
10	41	101	122	177	89	100	44	70	69	98
11	0	0	0	0	0	0	0	0	0	0
12	2	2	5	1	16	23	4	15	3	10

Table 20: Recognition of off-line with on-line model base (optimal thresholds)

Properly classified :	966	48.30
Confused :	81	4.05
Rejected :	911	45.55
Substitution:	42	2.10
ReallyConfused :	81	4.05
SecondBest:	0	0.00
Reliability:	966	95.83

6.6 Recognition Improvement with Weights

We are attempting to improve the recognition system by adding weights to the model. Do the recognition results support this hypothesis? Tables 24 and 22 give the recognition results for the re-split on-line database without weights. We also note that weighted recognition system uses 293 models when the unweighted version uses 337 models.

Experiment	Reco Rate	No Recog	No Models
Sub opt.	85.51%	1410	253
Optimal	87.76%	1449	295

Table 21: No of models vs. recognition rate

Result	0	1	2	3	4	5	6	7	8	9
0	141	0	0	0	0	0	1	0	0	0
1	0	141	0	0	0	0	0	0	0	0
2	0	2	142	0	0	0	0	0	0	0
3	0	0	0	145	0	1	0	0	0	0
4	0	0	0	0	151	0	0	0	0	0
5	0	0	0	0	0	155	0	0	0	0
6	0	0	0	0	0	0	145	0	0	0
7	0	1	0	0	0	0	0	108	0	0
8	0	0	0	0	0	0	0	0	149	0
9	0	0	0	0	4	0	0	0	0	148
Reject	21	14	17	12	9	7	15	28	16	7
Confused	3	7	6	8	1	2	3	29	0	10

Table 22: Confusion Matrix On-line re-split no weights, train against itself

Properly classified :	1425	86.42
Confused :	69	4.18
Rejected :	146	8.85
Substitution:	9	0.55
SecondBest:	69	4.18
Reliability:	1425	99.37

Comparing these results we see that weights do contribute to an increased recognition level, which is what we hoped. It is also important to notice that the substitutions have not increased and the reliability has increased as seen in Table 23. An increase in the recognition rate with an increase in substitution would not be as significant as it would be difficult to determine if the increased recognition rate is achieved by simply allowing for more substitutions. Since we observe the opposite, increasing recognition and decreasing substitution, expressed by an increase in the reliability, we conclude that we do improve on elastic matching.

Another improvement we observe is a reduction in the size of the model base. One could say that an improvement is achieved by using a larger model base, but here this is not so, on the contrary, the database decreases in size. So not only is an increase in recognition achieved, but it is done while reducing the size of the model base. The conclusion that we draw is that we are observing an increase in the generalization power of the models, each model representing a wider range of unknowns. So, on all accounts, the weights added to the models do improve the recognition system.

Method	Recognition	Substitution	Reliability
On-line weights	88.67%	0.48%	99.46%
On-line	85.22%	0.61%	99.29%
Off-line recon. weight.	80.15%	1.30%	98.40%
Off-line recon.	76.80%	1.20%	98.46%

Table 23: Recognition rate with and without the weights, train against test set

Result	0	1	2	3	4	5	6	7	8	9
0	132	0	0	0	0	0	0	0	0	0
1	0	141	0	0	0	0	0	0	0	0
2	0	0	139	0	0	0	0	0	0	0
3	0	1	0	150	0	1	0	0	0	0
4	0	0	0	0	152	0	0	0	0	1
5	0	0	0	0	0	155	0	0	0	0
6	0	0	0	0	0	0	152	0	0	0
7	0	1	0	0	0	0	0	100	0	0
8	0	0	0	1	0	0	0	0	145	0
9	1	0	0	0	3	0	0	1	0	141
Reject	31	19	17	8	10	9	11	40	17	14
Confused	1	3	9	6	1	0	2	24	3	9

Table 24: Confusion Matrix On-line re-split, no weights, train against test

Properly classified :	1407	85.22
Confused :	58	3.51
Rejected :	176	10.66
Substitution:	10	0.61
SecondBest:	58	3.51
Reliability:	1407	99.29

Chapter 7

Conclusion

In conclusion, character recognition remains a complex research area that has defied simple solutions in the past, and still does. We originally set out to improve the shape discrimination capabilities of the elastic matching algorithm. To do so, we added weights to the models in the hopes that the discriminating points would take on more importance and result in increased recognition and reliability. This was done in an attempt to make elastic matching a more general-purpose character recognizer. It was also done to see if the recognizer would work well for both writer-dependent systems as well as writer-independent systems.

Furthermore, we successfully processed off-line data with the use of preprocessing to convert it to on-line data. With the off-line data in this form, we were able to use it with an on-line character recognition method. In so doing we have a system that is also relatively independent of input means.

7.1 Contributions

We believe that we have made original contributions to the elastic matching recognition system. First, we added weights to the models. The added weights provided us with a more discriminating algorithm to perform better in the following respects:

- Recognition is improved. We obtain higher recognition rates with weighted models.

- The algorithm generalizes better, as it can recognize more patterns with fewer models. This, then, means that the models get to represent a wider variety of unknowns, without hindering the recognition results.
- Weights are a direct representation of the importance of a given feature. This helps the designer in identifying which part of the pattern is a better discriminator. This is a direct consequence of the way we generate weights.
- We have also found that elastic matching is related, to a certain degree, to neural networks, with the help of a tracking function. This is true of weighted elastic matching, but is also true with regular elastic matching.

Adding weights to the elastic matching formula, and the method used to create them, is the modification that best contributes to improving the recognition results of this algorithm. It is also what best differentiates this algorithm from the various other dynamic programming algorithms described in the literature. It allows dynamic programming algorithms to place more emphasis on specific portions of the pattern being matched. This permits the algorithm to better emulate behaviour that is encountered in humans, when recognizing digits or other tasks, by placing more importance on some more critical sections than others. Placing emphasis on discriminating parts must be expected from a computer if it is to try to achieve recognition levels as high as human subjects.

We have also found that there are certain resemblances to Neural Networks in general. Elastic Matching is certainly primitive as neural networks go, but this opens what could be an interesting area of research. Elastic matching could certainly benefit from the numerous advances done in the field of neural networks, especially when it comes to training algorithms, thus opening new possibilities for improving elastic matching as a pattern recognition system.

We also described a method of selecting models for our recognition system. Although we make no claims in terms of its performance, we think it is well adapted to the problems we face. The training algorithm allows us to create a reasonable set of models from a large set of patterns. It also provides us with an intuitive way to evaluate the weights we need.

7.2 Future Work

There are two main areas where this system needs to be improved. First, the recognition results need to be improved. The results obtained are fairly good in view of the fact that the system is not handcrafted to the particular data set we used. The recognition system should work faster than it currently does, if it is to be of practical use.

7.2.1 Better Training

The training mechanism we use does not guarantee convergence, or even to improve results. Admittedly, iterative training does provide improved results after iteration, but this cannot be guaranteed. Similar in concept to the Perceptron training algorithm, it would be worth exploring whether advances in neural network training algorithms would prove beneficial in this case.

7.2.2 Better Preprocessing

There are two main improvements that need to be applied to the preprocessing. The first modification would be to use more discriminating algorithms at the generic preprocessing stage. We noted that sharp corners are not well kept by the smoothing and distance normalization steps. This is a problem because the sharpness of corners has been found to be a significant feature for the discrimination of several similar shapes like a “4” and an open “9”.

Secondly, in the off-line preprocessing stage we should process retraces. Longer strokes lead to more distinctive unknowns and make it easier to differentiate one class from the other. So we expect that processing retraces should bring the recognition level of the classes where suboptimal reconnection occurs more in line with the results observed with on-line data.

7.2.3 Enhanced Pruning

Pruning was used as a means of increasing the speed of the recognition process, while keeping the same recognition rate. The primary area where pruning can be enhanced is at character based pruning. Better use of the distance found by the three point match currently used could help eliminate more matching. This filter only eliminates one pattern at a time for elastic matching. It may well be possible that using this filter with one sample of each class could help eliminate, immediately, certain classes from consideration, saving us from evaluating the filter on whole subsets hence saving computations.

Placing first the models that match the most frequently, within each class, would help as it would reduce the average number of times the elastic matching algorithm is performed. Both modifications are relatively simple and should provide non-negligible speed improvements.

7.2.4 Better Weights

Of all the ways this system could be improved, the one we believe to be most significant is with the weights. We have seen that the weights prove to be valuable in increasing recognition. These weights are, however, limited in what they express. They only show how important a section is relative to another one. Other character recognition systems have shown it to be useful to view some features as more important on some occasions than on others. For this reason, we believe that the best way to increase the recognition rate is to modify the basic distance function $d(i,j)$, by adding weights to each of its components. We could modify Equation 15 in the following way:

$$d(i,j;k) = \rho_i((x_i - x_{j,k})^2 + (y_i - y_{j,k})^2) + \varrho_i\,|\phi_i - \phi_{j,k}| + \tau_i\,|\chi_i - \chi_{j,k}| \qquad (74)$$

These new weights, ρ, ϱ, τ, would then also provide us with a way of putting emphasis on feature measurements that prove to be more reliable in any given case. For instance, angle of elevation may prove to be more reliable than strict distance, to distinguish between "1" and "7", while curvature may be more reliable than angle of elevation to discriminate between an open "0" and a "6". Doing so will not only provide us with weights that will tell us which part of the pattern is more reliable, but also which feature in what part is most likely to be a good discriminator. In identifying which of the measurements is the most reliable discriminator at a given section between the patterns of the correct and error set, it may well be the case that for some sections a feature is relatively reliable but the other measurements prove to introduce noise that masks this characteristic and hinders recognition. We have mentioned it before, but some statistical information may prove to be helpful in this case as it could determine the distribution function of the various measurements and determine which has the least amount of overlap.

It has also been said that elastic matching suffers from taking only local measurements. It is true that the measurements are local to each portion of the patterns. It may be worthwhile to see if more global features can prove to be more indicative of the properties of the pattern. We are thinking in terms of the distance from the point to a singular point (such as end points), or the distance from the point to the top, bottom or one of the sides, or the quadrant to which it belongs. Although this does generate a wealth of features, they can be evaluated on demand, only when needed to distinguish between confusing patterns, thus saving computational costs.

7.2.5 Stroke Reconnection Algorithm

There is agreement in the literature that the *Good Continuity* rule makes sense from a perceptual point of view, as well as from a writing generation point of view. However, reconnection has yet to find a robust heuristic to handle the problem of retraced strokes [43], even when using the full image [27]. While our system creates on-line data from off-line from a data representation level, it fails to treat retraced strokes. While this is not a hindrance to the usage of on-line recognition techniques, it still penalizes their use by providing suboptimal data.

In our case, a large proportion of the stroke merges are done through the use of a small bridge. This properly handles the loops found in the "2", "6", "8", "9" and is responsible for our success in achieving a high percentage of optimal merges with these characters. More work is needed to bridge the gap for these digits and achieve a reliable treatment of the retraced stroke. From observation of the data, we believe that we have described an algorithm that properly treats retraces in the majority of cases.

7.2.6 Use of Contour Information

We have noted that the performance for the off-line database is lower than that of other published results. Also the results of thinning based algorithms are lower than algorithms using more information such as the contours or the stroke width. For this reason we believe that we need to use more information to recognize the off-line data.

Two main options are open to us. The first one is to keep some information about the width of the stroke at each pixel of the skeleton. This should keep information as to how sharp a turn is or how strokes merge, and help disambiguate stroke reconnection.

Alternatively, contours have also been found to be good descriptors of the shape of a character, as with fourier coefficients. We could use, instead of skeletons, the contour of the pattern as it keeps more information about the stroke width. Such a study should be easily performed once the contour is extracted from the data since it would essentially still be a sequence of points.

7.2.7 True Off-line and On-line Comparison

We have used the system with both off-line and on-line data. Any comparisons made between both types of data are inconclusive because we use two different databases. For this reason we believe that the best way to compare the results between off-line and on-line would be to use the same database. To do so we suggest creating a off-line database from our on-line data. In so doing, any recognition difference between the off-line and the on-line could then be attributed to the loss of information incurred because of the loss of connectivity information. It could prove useful to determine how much the connectivity information present in on-line data contributes to recognition, and by the same token, how good our off-line specific preprocessing is in reconstructing the time related information. As a side benefit it would also provide an easy method to measure the quality of the various heuristics that can be used for the reconnection of strokes since we could pair the off-line and on-line versions of the pattern, their distance measurements being a criterion, to determine the quality of the reconstruction of the connectivity information.

7.2.8 Machine Weights *vs.* Human Expertise

As we noted before, there have been experiments that try to determine the importance of some character features over others. Because the purpose of the weights is to emphasize discriminating portions of characters, it would be interesting to see if they are in accord with these experiments [65, 66].

7.2.9 Making Better Use of the Weights

The weights were chosen to be representative of the importance of a feature. In this respect, weighted elastic matching resembles many other recognition systems. We noted the resemblance with neural networks and the fact that neural networks have been used to create quality expert systems. The character recognition problem is one suitable to that approach and could be used as another source of knowledge for the creation of rule-based expert system character recognizers. Conversely, the rules used by expert systems may help in refining this system, by defining other measurements that would generate weights emphasizing the same features expert systems use for recognition.

7.3 Conclusion

Due to the complexities of the problems studied, and their requirements involved, researchers tend to divide the problem into smaller subproblems. However, this may well be the opposite of what is needed. We come to realize that every stage of processing is dependent upon the preceding one. Not only the nature of the problems makes it difficult to correctly specify subproblems, but it makes it equally difficult to verify the quality of the intermediate results. As a result, the quality of a character recognizer does not depend on one single idea but the good integration of many simple processes, a phenomenon also noted in the discussion on the human mind by Minsky [75]. The system integration, often seen as a purely development process, then takes as much importance as the various algorithms, as only when we are testing the integrated system can we find its true measure of quality.

This character recognition system is no different. The recognition results are dependent on many steps, most of which we have not attempted to improve, but their influence is quite measurable in overall system performance. As we have seen, adding weights to the elastic matching algorithm does improve recognition results, and we believe that weights, by extending the usability of the elastic matching algorithm, improve performance enough to warrant the extra work involved in finding the weights as well as the extra CPU needed at the recognition phase.

References

[1] P. Ahmed. *Computer Recognition of Totally Unconstrained Handwritten Zip Code*. PhD thesis, Concordia University, July 1986.

[2] M.R. Anderberg. *Cluster analysis for applications*. Academic Press, 111 Fifth Avenue, New York, New York 10003, 1973. Originally presented as the author's Phd thesis, University of Texas at Ausin, 1971.

[3] J.A. Anderson and E. Rosenfeld. *Neurocomputing: Fondations of Research*. The MIT Press, Cambridge Massachusetts, 1988.

[4] C. Arcelli. Pattern thinning by contour tracing. *Computer Graphics and Image Processing*, 17:130–144, 1981.

[5] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, and J.S.B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(6):209–216, March 1991.

[6] A.M.K. Badreldin. *Structural Recognition of Handwritten Numeral Strings*. PhD thesis, University of Windsor, 1985.

[7] W.P. Banks and W. Prinzmetal. Configurational effects in visual information processing. *Perception & Psychophysics*, 19(4):361–367, 1976.

[8] W.M. Bartlett and al. Tablet: Personal computer in the year 2000. *Communications of the ACM*, 31(6):639–646, June 1988.

[9] J. Beck. Effect of orientation and of shape similarity on perceptual grouping. *Perception & Psychophysics*, 1:300–302, 1966.

[10] E.J. Bellegarda, J.R. Bellegarda, D. Nahamoo, and K.S. Nathan. A probabilistic framework for on-line handwriting recognition. In *Frontiers in Handwriting Recognition III*, pages 225–234, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[11] S. Bercu and G. Lorette. On-line handwritten word recognition: An approach based on hidden markov models. In *Frontiers in Handwriting Recognition III*, pages 385–390, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[12] B. Blesser, R. Shillerman, T. Kuklinsky, C. H. Cox III, M. Eden, and J. Ventura. A theoretical approach for character recognition based on phenomenological attributes. In *Proceedings of the First Joint Conference on Pattern Recognition*, pages 33–40, Washington D.C., Oct. 30–Nov 1 1973.

[13] B. A. Blesser, T. Kuklinski, and R. J. Shillerman. Empirical tests for feature selection based on a psychological theory of character recognition. *Pattern Recognition*, 8:77–85, 1976.

[14] H.D. Block. The perceptron: a model for brain functionning. *Reviews of Modern Physics*, 34:123–135, 1962. Reprinted in Anderson and Rosenberg 1988.

[15] J-J. Brault. *Proposition et vérification d'un coefficient de difficulté d'imitation des signatures manuscrites.* PhD thesis, Ecole Polytechnique, Université de Montréal, Octobre 1988.

[16] D.J. Burr. Elastic matching of line drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 3(6):708–713, November 1981.

[17] M.Y. Chen and A. Kundu. An alternative to variable duration hmm in handwritten word recognition. In *Frontiers in Handwriting Recognition III*, pages 82–91, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[18] Y.K. Chu and C.Y. Suen. An alternate smoothing and stripping algorithm for thinning digital binary patterns. *Signal Processing*, 11(3):207–222, October 1980.

[19] G-M. Cogne. Coolscan nikon. *Chasseur D'Image*, (152):118–121, Avril 1993.

[20] A.Y. Commike and J. Hull. Syntactic pattern classification by branch and bound search. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 432–437, Lahaina,Maui, Hawaii, June 3–6 1991.

[21] C.H. Cox, P. Coueignoux, B. Blesser, and M. Eden. Skeletons: A link between the theoretical and physical letter description. *Pattern Recognition*, 15(1):11–22, 1982.

[22] H.D. Crane and J.S. Ostrem. Confusion grouping of strokes in pattern recognition and method and system. United States Patent, 4,573,196, February 1986.

[23] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann, San Mateo CA, 1990.

[24] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, and H.S. Baird. Constrained neural network for unconstrained handwritten digit recognition. In *Frontiers in Handwriting Recognition*, pages 145–154, CENPARMI Concordia University, Montreal, Quebec Canada, April 2-3 1990.

[25] Stanford Diehl and David L. Edwards. Scanning the spectrum, color flatbed scanners. *BYTE*, 17(7):230–242, July 1992.

[26] S. DiZenzo, M. DelBuono, M. Meucci, and A. Spirito. Optical recognition of hand-printed characters of any size, position and orientation. *IBM Journal of Research and Development*, 36(3):487–501, May 1992.

[27] D.S. Doermann and A. Rosenfeld. The interpretation and reconstruction of interfering strokes. In *Frontiers in Handwriting Recognition III*, pages 41–50, CEDAR, SUNY Buffalo, New York, USA, May 25–27 1993.

[28] W. Doster and R. Oed. Word processing with on-line script recognition. *IEEE Micro*, pages 36–43, October 1984.

[29] R. Durbin and D. Willshaw. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326(16):689–691, April 1987.

[30] J.T. Favata and S.N. Srihari. Recognition of general handwritten words using a hypothesis generation and reduction methodology. In *Proc. USPS Adv. Tech. Conf.*, pages 237–252, Washington DC., Nov. 30–Dec. 2 1992.

[31] J. Franke, L. Lam, R. Legault, C. Nadal, and C.Y. Suen. Experiments with the cenparmi data base combining different classification approaches. In *Frontiers in Handwriting Recognition III*, pages 305–311, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[32] L.S. Frishkopf. Automatic recognition of handwriting. United States Patent, 3,133,266, May 1964.

[33] Y. Fujimoto, S. Kadota, S. Hayashi, and M. Yamamoto. Recognition of hand-printed characters by nonlinear elastic matching. In *Proceedings of the third Joint Conference on Pattern Recognition*, pages 113–118, Coronado California, November 8–11 1976.

[34] T. Fujisaki, C.C. Tappert, M. Ukelson, and C.G. Wolf. On-line recognition of run-on handwriting: Stroke based recognition and evaluation. In *From Pixels to Features III, International Workshop on Frontiers in Handwritting Recognition*, pages 205–216, Chateau de Bonas, France, September 8–11 1991.

[35] K. Fukushima and N. Wake. Handwritten alphanumeric character recognition by the neocognitron. *IEEE Transactions on Neural Networks*, 2(3):355–365, May 1991.

[36] P. Gader, B. Forester, M. Ganzberget, A. Gillies, B. Mitchell, M. Whalen, and T. Yocum. Recognition of handwritten digits using template and model matching. *Pattern Recognition*, 24(5):421–431, 1991.

[37] S.I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[38] S.I. Gallant. Perceptron-based learning algorithm. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[39] J. Gettys, R.W. Scheifler, and R. Newman. *Xlib – C Language X Interface*. MIT X Consortium Standard, May 1989. X Version 11, Release 4.

[40] J. W. Gorman, O. Robert Mitchell, and F.P. Kuhl. Partial shape recognition using dynamic programming. *IEEE Transactions on Pattern Recognition Machine Intelligence*, 10(2):257–266, March 1988.

[41] V.K. Govidan and A.P. Shivaprasad. Character recognition – a survey. *Pattern Recognition*, 23(7):671–683, 1990.

[42] V. Govindaraju and S.N. Srihari. Separating handwritten text from overlapping non-textual contours. In *From Pixels to Features III, International Workshop on Frontiers in Handwritting Recognition*, pages 111–119, Chateau de Bonas, France, September 23–27 1991.

[43] V. Govindaraju, D. Wand, and S.N. Srihari. Using temporal information in off-line word recognition. In *United States Postal Service Advanced Technology Conference*, pages 529–544, November 30 – December 2 1992.

[44] J.C. Gower and G.J.S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

[45] P.E. Green, F.J. Camone, and J. Kim. A preliminary study of optimal variable weighting in k-means clustering. *Journal of Classification*, 7:271–285, 1990.

[46] L.D. Harmond. Method and apparatus for reading cursive script. United States Patent, 3,111,646, November 1963.

[47] J.A. Hertz, R.G. Palmer, and A.S. Krogh. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[48] C.J. Hilditch. Comparison of thinning algorithms on a parallel processor. *Image Vision Computing*, 1(3):115–132, 1983.

[49] G.E. Hinton, C.K.I. Williams, and M.D. Revow. Adaptive elastic models for hand-printed character recognition. In S.J. Hanson J.E. Moody and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1–8. Morgan Kauffmann, San Mateo CA, 1992. Preprint.

[50] C. Holt and A. Stewart. A parallel thinning algorithm with fine grain subtasking. *Parallel Computing*, 1:329–334, 1989.

[51] Y.S. Huang and C.Y. Suen. The behavior-knowledge space method for combination of multiple classifiers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition '93*, pages 347–352, New York City, NY, June 15-17 1993.

[52] Y.S. Huang and C.Y. Suen. An optimal method of combining multiple classifiers for unconstrained handwritten numeral recognition. In *Frontiers in Handwriting Recognition III*, pages 11–20, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[53] Image-In Inc., 406 EAST 79th street, Minnieapolis, MN 55420. *IMAGE-IN-COLOR reference manual*, 1992.

[54] A.K. Jain and R.C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Englewood Cliffs, New Jersay 07632, 1988.

[55] T. Kasvand and N. Otsu. Segmentation of thinned binary scenes with good connectivity algorithm. In *Proc. 7th Int. Conf. Pattern Recognition*, pages 297–300, Montreal, July 30–August 2 1984.

[56] J. Kim. On-line gesture recognition by feature analysis. In *Vision Interface 88*, pages 51–55, Edmonton, Alberta, June 1988.

[57] K. Koffka. *Principles of Gestalt Psychology.* Hartcourt Brace & World Inc, New York, 1935.

[58] J.B. Krushkal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983.

[59] A. Krzyzak, W. Dai, and C.Y. Suen. Uncostrained handwritten character classificaiton using modified backpropagation model. In *Frontiers in Handwriting Recognition*, pages 155–165, CENPARMI Concordia University, Montreal, Quebec Canada, April 2-3 1990.

[60] A. Krzyżak, S.Y. Leung, and C.Y. Suen. Reconstruction of two-dimensional patterns from fourrier descriptors. *Machine Vision and Applications*, 2(2):123–140, 1989.

[61] L. Lam, S-W. Lee, and C.Y. Suen. Thinning methodologies - a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[62] L. Lam and C.Y. Suen. Structural classifiction and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition*, 21(1):19–31, 1988.

[63] L. Lam and C.Y. Suen. A dynamic shape preserving thinning algorithm. *Signal Processing*, 22(2):199–207, 1991.

[64] R. Legault and C.Y. Suen. Smoothing of 2-d binary contours. Technical report, June 1993.

[65] R. Legault, C.Y. Suen, and C. Nadal. Classification of confusing handwritten numerals by human subject. In *Frontiers in Handwriting Recognition*, pages 181–194, CENPARMI Concordia University, Montreal, Quebec Canada, April 2-3 1990.

[66] R. Legault, C.Y. Suen, and C. Nadal. Difficult cases in handwritten numeral recognition. In K. Yamamoto H.S. Baird, H. Bunke, editor, *Structured Document Image Analysis*, pages 235–249. Springer-Verlag, Berlin, 1992.

[67] Z.C. Li, C.Y. Suen, and J. Guo. Computer algorithm for recognizing the distinct part of handprinted characters. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*, pages 197–200, Charlottesville Virginia, Oct 13–16 1991.

[68] Owen Linderholm, Steve Apiki, and Michael Nadeau. The pc gets more personal. *BYTE*, 17(7):128–138, July 1992. Cover Story.

[69] J.S. Lipscomb. A trainable gesture recognizer. *Pattern Recognition*, 24(9):895–907, 1991.

[70] D.P. Lopresti and A. Tomkins. Pictographic naming. Technical Report MITL-TR-21-92, Matsushita Information Technology Laboratory, 1992.

[71] D.P. Lopresti and A. Tomkins. Approximate matching of hand-drawn pictograms. In *Frontiers in Handwriting Recognition III*, pages 102–111, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[72] H. E. Lu and P. S. P. Wang. A comment on a "a fast parallel algorithm for thinning digital patterns". *Communications of the ACM*, 29(3):239–242, 1986.

[73] Steven Malowany. A graphical rule-based system for recognizing on-line digits. Master's thesis, Concordia University, Montreal, Canada, April 1993.

[74] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. Reprinted in Anderson and Rosenberg 1988.

[75] Marvin Minsky. *The society of mind*. Simon and Schuster, New York, 1986.

[76] W. Nelson and E. Kishon. Use of dynamic features for signature verification. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*, pages 201–205, Charlottesville, Virginia, Oct 13–16 1991.

[77] F. Nouboud and R. Plamondon. On-line recognition of handprinted characters: Survey and beta tests. *Pattern Recognition*, 23(9):1031–1044, 1990.

[78] H. Ogawa and K. Taniguchi. Thinning and stroke segmentation for handwritten chinese character recognition. *Pattern Recognition*, 15(4):299–308, 1982.

[79] Y. Park and J. Skylansky. Automated design of linear tree classifiers. *Pattern Recognition*, 23(12):1393–1412, 1990.

[80] E. Persoon and K-S. Fu. Shape discrimination using fourier descriptors. *IEEE Transaction on Systems, Man and Cybernetics*, 7(3):170–179, March 1977.

[81] R. Plamondon and G. Lorette. Automatic signature verification and writer identification– the state of the art. *Pattern Recognition*, 22(2):107–131, 1989.

[82] R. Plamondon and C.Y. Suen. On the definition of reference skeletons for comparing thinning algorithms. In *Vision Interface 88*, pages 70–75, Edmonton, Alberta, June 1988.

[83] R. Plamondon and C.Y. Suen. Thinning of digitized characters from subjective experiments: A proposal for a systematic evaluation protocol of algorithms. In T. Kasvand A. Krzyzak and C. Y. Suen, editors, *Computer Vision and Shape Recognition*. World Scientific Publishing Co., Singapore, 1989.

[84] M. Revow, C.K.I. Williams, and G.E. Hinton. Using mixture of deformable models to capture variations in hand printed digits. In *Frontiers in Handwriting Recognition III*, pages 142–152, CEDAR, SUNY Buffalo, New York, USA, May 25–27 1993.

[85] J. Rocha and T. Pavlidis. A shape analysis model with applications to a character recognition system. In *Proc. IEEE Workshop on Applications of Computer Vision*, pages 182–189, Palm Springs, Nov. 30–Dec. 2 1992.

[86] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. Reprinted in Anderson and Rosenberg 1988.

[87] A. Rosenfeld. Connectivity in digital pictures. *Journal of the ACM*, 17(1):146–160, January 1970.

[88] I. Roux. L'ordinateur gridpad, les nouveaux horizons. *Science & Vie Micro*, (73):90–94, Juin 1990.

[89] J.W. Schoonard, J.D. Gould, M. Bieber, and A. Fusca. A behavioral study of a computer hand print recognition system. Research Report RC 12494, IBM, June 1987.

[90] A.W. Senior and F. Fallside. An off-line cursive script recognition system using recurrent error propagation network. In *Frontiers in Handwriting Recognition III*, pages 132–141, CEDAR, SUNY Buffalo, New York, USA, May 25-27 1993.

[91] H. Späth. *Cluster Analysis Algorithms for data reduction and classification of objects*. Ellis Horwood Publishers, 605 Third Avenue, New York, N.Y. 10016 USA, 1980.

[92] K.T. Spoehr and S.W. Lehmkuhle. *Visual information processing*. W.H. Freeman and Company, San Francisco, 1982.

[93] N.W. Strathy, C.Y. Suen, and A. Krzyzak. Segmentation of handwritten digits using contour features. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, October 20– 22 1993.

[94] L. Stringa. Efficient classification of totally unconstrained handwritten numerals with a trainable multilayer network. *Pattern Recogition Letters*, 10(4):273–280, October 1989.

[95] C.Y. Suen, J. Guo, and Z.C. Li. Computer and human recognition of handprinted characters by parts. In *From Pixels to Features III, International Workshop on Frontiers in Handwriting Recognition*, pages 161–174, Chateau de Bonas, France, September 23–27 1991.

[96] C.Y. Suen, J. Guo, and Z.C. Li. Analysis and recognition of alphanumeric handprints by parts. In *Proc. 11th Int. Conf. Pattern Recognition*, pages 338–341, The Hague, The Netherlands, August 30– Sept 3 1992.

[97] C.Y. Suen, R. Legault, C. Nadal, M. Cheriet, and L Lam. Building a new generation of hadnwriting recognition systems. *Pattern Recognition Letters*, 14(4):303–315, April 1993.

[98] C.Y. Suen, C. Nadal, T.A. Mai, R. Legault, and L. Lam. Recognition of totally unconstrained handwritten numeral based on the concept of multiple experts. In *Frontiers in Handwriting Recognition*, pages 131–144, Concordia University, Montreal, Quebec, Canada, April 2-3 1990.

[99] H. Tamura. A comparison of line thinning algorithms from a digital geometry viewpoint. In *Proceedings of the 4th International Joint Conference in Pattern Recognition*, pages 715–719, Kyoto, Japan, November 7-10 1978.

[100] C.C. Tappert. Cursive script recognition by elastic matching. *IBM Journal of Research and Development*, 26(6):765–771, November 1982.

[101] C.C. Tappert. Segmentation function enhancement to cursive script recognition. *IBM Technical Disclosure Bulletin*, 27(4), September 1984.

[102] C.C. Tappert. Multi-segment system for recognizing cursive writing. *IBM Technical Disclosure Bulletin*, 27(11), April 1985.

[103] C.C. Tappert. Speed, accuracy, flexibility trade-offs in on-line character recognition. Research Report RC13228, IBM, October 1987.

[104] C.C. Tappert. Recognition system for run-on handwritten characters. United States Patent, 4,731,857, March 1988.

[105] C.C. Tappert. Rationale for adaptive online handwriting recognition. In *Frontiers in Handwriting Recognition*, pages 181–194, CENPARMI Concordia University, Montreal, Quebec Canada, April 2-3 1990.

[106] C.C. Tappert, A.S. Fox, J. Kim, S.E. Levy, and L.L. Zimmerman. Handwriting recognition on transparent tablet over flat display. *Proceedings of the SID*, 28(1):67–73, 1987.

[107] C.C. Tappert, C.Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 12(8):787–807, 1990.

[108] N. Ueda and S. Suzuki. Automatic shape model acquisition using multiscale segment matching. In *Proceedings of ICPR*, pages 897–902, Atlantic City, New Jersey, June 16–21 1990.

[109] N. Ueda and S. Suzuki. Learbubg visual models from shape contours using multiscale convex/concace structure matching. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 15(4):337–352, April 1993.

[110] D. Walters. Selection of images primitives for general-purpose visual processing. *Computer Vision, Graphics, and Image Processing*, 37(3):261–298, 1987.

[111] J.R. Ward. One view of on-going problems in handwriting character recognition. In *Frontiers in Handwriting Recognition*, pages 181–194, CENPARMI Concordia University, Montreal, Canada, April 2-3 1990.

[112] J.R. Ward and B. Blesser. Interactive recognition of handprinter characters for computer input. *IEEE Computer Graphics and Applications*, 5(9):24–37, September 1985.

[113] J.R. Ward and T. Kuklinsky. A model for variability effects in handprinting with implications for the desing of handwriting character recognition systems. *IEEE Transaction on System, Man, and Cybernetics*, 18(3):438–451, November 1988.

[114] J.R. Ward and M. Philips. Digitizer technology: performance characteristics and the effects on the user interface. *IEEE Computer Graphics and Applications*, 7(4):31–44, April 1987.

[115] C.K.I. Williams, M.D. Revow, and G.E. Hinton. Hand-printed digit recognition using deformable models. In L. Harris and M. Jenkin, editors, *Spatial Vision in Humans and Robots*, pages 1–23. Cambridge University Press, Cambridge, 1992.

[116] C.G. Wolf. Can people use gesture commands? *SIGCHI*, 18(2):73–74, 1986.

[117] H. Yamada, C. Merritt, and T. Kasvand. Recognition of kidney glomerulus by dynamic programming matching method. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 10(5):731–737, 1988.

[118] H. Yamada and K. Yamamoto. Recognition of echocardiograms by a dynamic programming matching method. *Pattern Recognition*, 24(2):147–155, 1991.

[119] S. Yokoi, J-I. Toriwaki, and T. Fukumara. An analysis of topological properties of digitized binary pictures using local features. *Syst. Comput. Contr.*, 4(6):32–39, 1973.

[120] I. Yoshimura and M. Yoshimura. On-line signature verification incorporating the direction of pen movement. In *From Pixels to Features III, International Workshop on Frontiers in Handwritting Recognition*, pages 435–440, Chateau de Bonas, France, September 23–27 1991.

[121] T. Y. Zhang and C. Y. Suen. A fast parellel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

Appendix A

Models Used

We show the models selected by the training algorithm. This data is taken from the resplit on-line training set during the experiment recognizing off-line digits with on-line models. All the models used by the recognizer are shown.

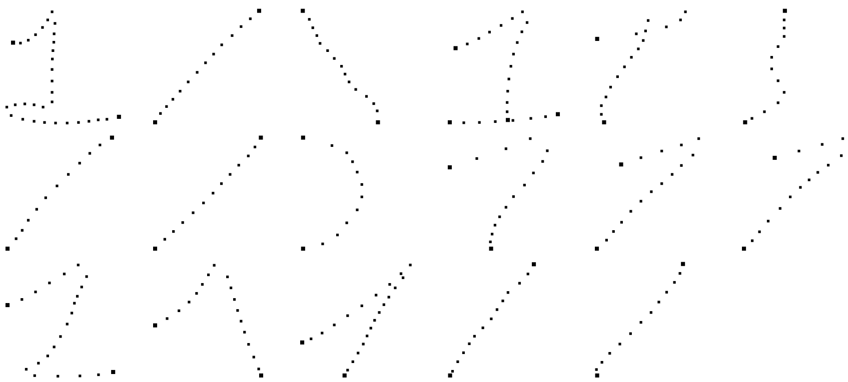

Figure 47: Samples of models for 1

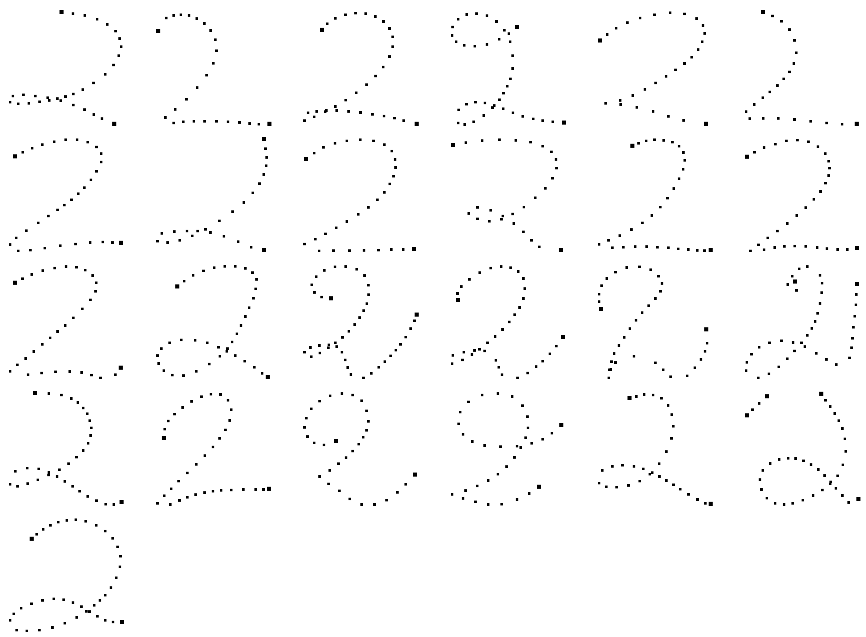

Figure 48: Samples of models for 2

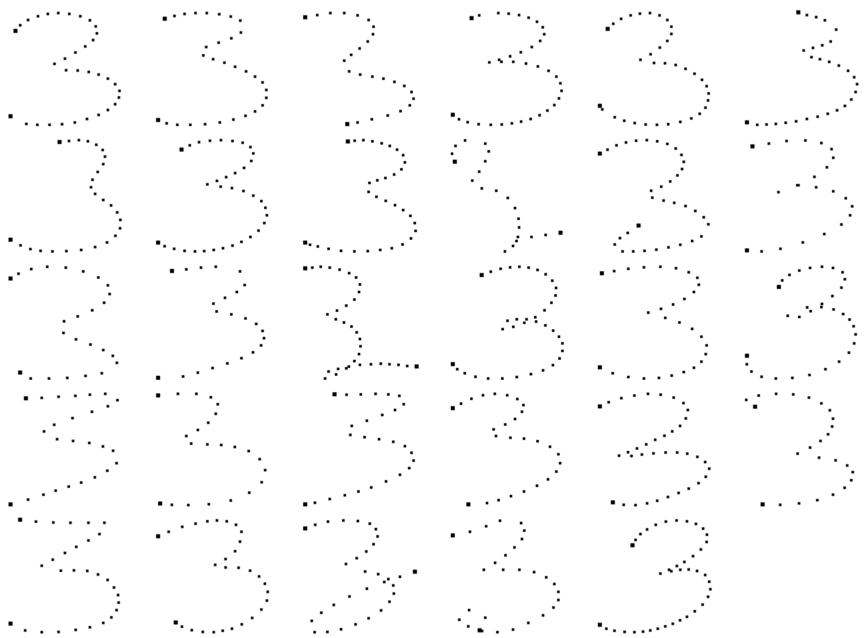

Figure 49: Samples of models for 3

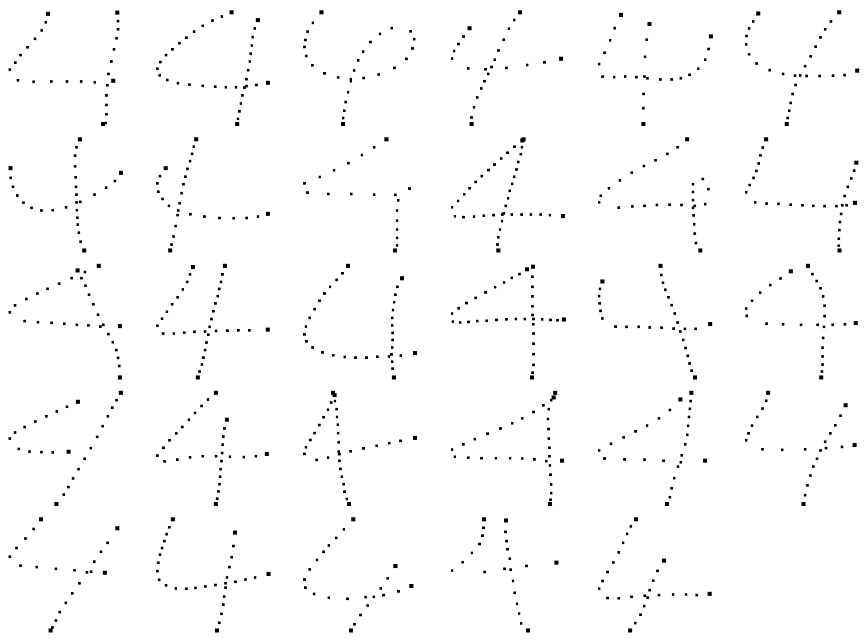

Figure 50: Samples of models for 4

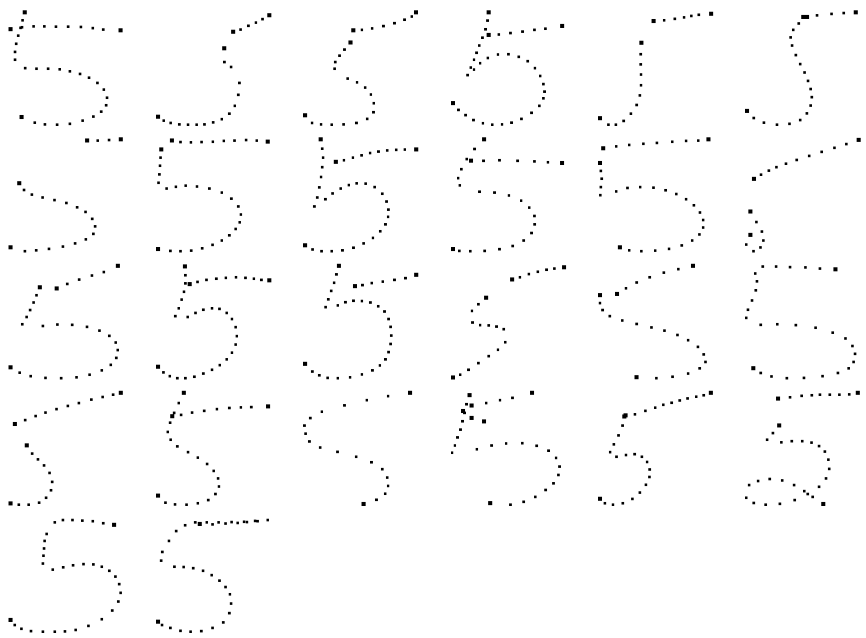

Figure 51: Samples of models for 5

Figure 52: Samples of models for 6

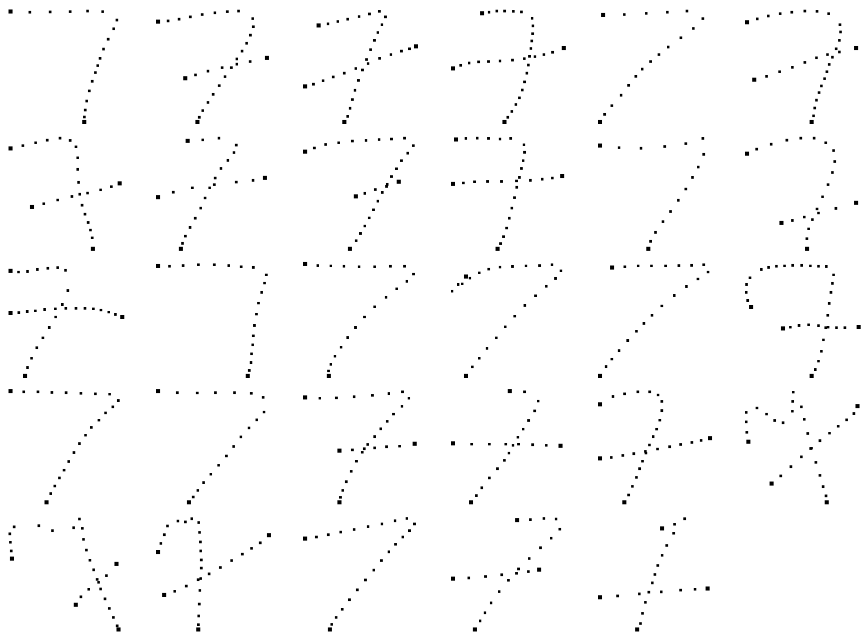

Figure 53: Samples of models for 7

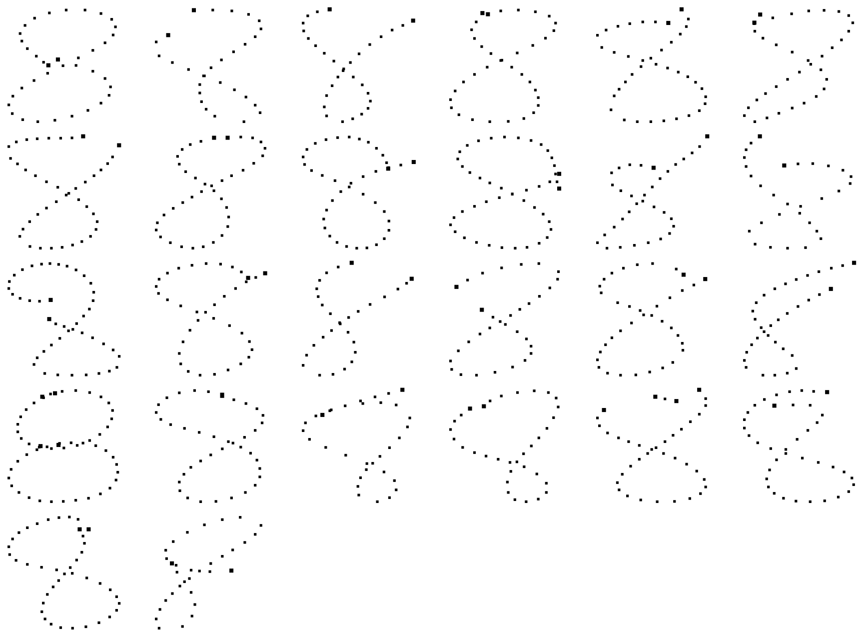

Figure 54: Samples of models for 8

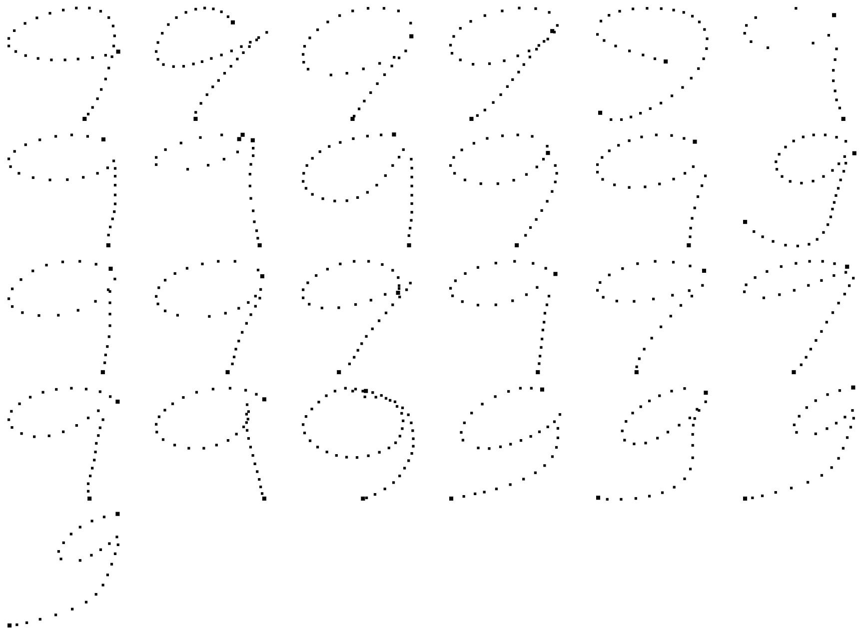

Figure 55: Samples of models for 9

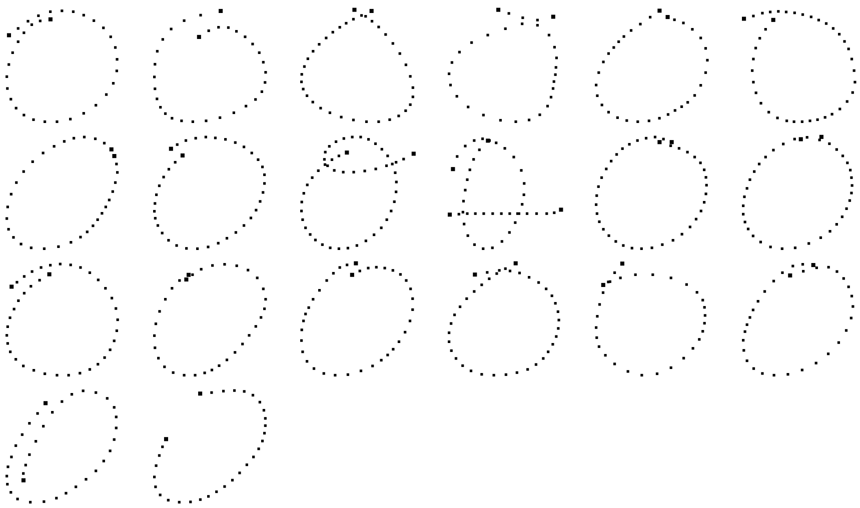

Figure 56: Samples of models for 0