A METHOD FOR SEGMENTATION OF TOUCHING HANDWRITTEN NUMERALS

Nicholas W. Strathy *AMDG*

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 1993

ISBN 0-315-87288-8

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Nicholas W. Strathy**

Entitled: **A Method for Segmentation of Touching Handwritten Numerals**

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:

[illegible] Chair

[illegible]

[illegible]

[illegible] Co supervisor

Co supervisor

Approved [illegible] 14/9/93
~~Chair of Department or~~ Graduate Program Director

Sept 13 19 93

Dean of Faculty

Abstract

A Method for Segmentation of Touching Handwritten Numerals

Nicholas W. Strathy

A method of separating the leftmost numeral from a string of touching unconstrained handwritten arabic numerals is proposed. A binary image containing a string of touching numerals is scanned to give contour chains. The chains are analysed and subdivided into four kinds of regions: valleys, mountains, holes, and open regions. Individual points of interest in the outer contour are then identified, e.g., points of high curvature, and other points which have been shown to be significant for performing perceptual grouping of objects in scenes. The separating path is assumed to pass between some pair of these significant contour points (SCPs). To find that pair, 9 features of the SCPs are measured and are used to sort the list of all possible pairings of SCPs. As with other segmentation methods, the output of this system is a short list of segmentation hypotheses sorted in order of confidence. Test results show that the correct cut is sorted within the first 3 choices in 89.5% of samples, and within the first 5 choices in 96.3% of samples. The total number of test images is 946 divided between 3 different sets. The system is trainable, something not seen in any other digit separation system in the literature. Training was done on a completely different set of 212 images.

Acknowledgements

First and foremost I would like to render thanks to Almighty God for providing such abundance of interesting challenges together with energy, inspiration, perseverance, and great joy in the working out of this thesis, whatever its merits may be.

Next, I would like to express my appreciation to my supervisors Drs. Ching Suen and Adam Krzyzak for having created such a stimulating and well-equipped lab environment, the fruit of years of hard work. The freedom and enjoyment I have had in working at CENPARMI will be a beautiful memory to carry with me down the years.

I am indebted to Dr. Mohamed Cheriet for many helpful hints and especially for the suggestion that I investigate the usefulness of corner detection for segmentation.

I also gratefully acknowledge the financial support of the National Networks of Centres of Excellence of Canada, the Natural Sciences and Engineering Research Council of Canada (NSERC), and *Fonds pour la Formation des Chercheurs et l'Aide à la Recherche* of Quebec (FCAR).

To all of my other friends at CENPARMI, may I never forget you, I say thank you for your friendship, help, presence: Zhisong Chen, Didier Guillevic, Yea-Shuan Huang, Louisa Lam, Raymond Legault, Bei Li, Steve Malowany, Irene Mazis, Christine Nadal, Patrice Scattolin, Khalid Siddiqui, Mary Smodis, Y.Y. Tang, William Wong, Daini Xie, Mike Yu, Jianxing Yuan, and Peiying Zhu.

How often do I get a chance to publicly thank my parents in print for providing so much undeserved love and support to one who too often takes advantage of their perennial doting attendance to his needs of body, mind and spirit? Thanks Mum and Dad.

A grateful thank you goes out to all of those family members and others who have supported my education over the years through prayer. And of course I am not forgetting four-legged supporters Jasper, Jigs, Jean-Guy, and Elizabeth, even though they may never read this.

A.M.D.G.

Seek ye first the kingdom of God and his righteousness, and all these things shall be added unto you.

–Matthew 6:33

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The challenge

This thesis proposes a method of performing one step in the machine recognition of images such as those appearing in Fig. 1. The proposal is not to recognise the individual digits—something most readers of this text will have little trouble doing it is merely to separate the digits one from another; no, that is even going too far: in fact, this thesis proposes a method of separating just the *leftmost* digit from a string of connected handwritten digits. Well, actually, it does not even propose to do that with much certainty! To come to the point, the best it has to offer is a handful of attempts at separating the leftmost digit, sorted from highest confidence to lowest.

This, however, is one of the approaches to the recognition process that is currently popular [13, 17, 25, 26], and not without some reason. In this strategy, segmentation is seen as an integral part of recognition, i.e., if an input cannot be recognised, then a segmentation hypothesis is applied, followed by another recognition attempt, and so on. In this manner, inputs such as those in Fig. 1 can conceivably be recognised by

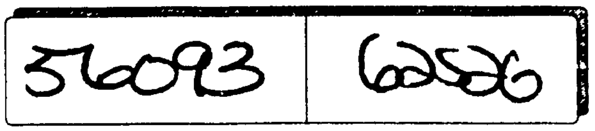

Figure 1: Touching digits.

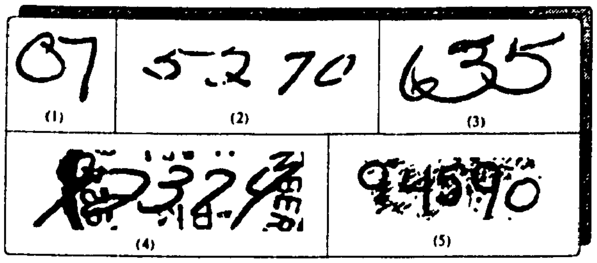

Figure 2: Other segmentation issues: (1) overlap, (2) breakage, (3) superimposition, (4) noncontextual strokes, (5) noise.

successive separations of the leftmost digit. Actually, one could work from the right hand side just as well, or even both sides, but this thesis is limited to showing the viability of a method for the left hand side.

Of course, connected digits are only one aspect of the digit-segmentation challenge. Fig. 2 gives examples of some of the other issues:

1. overlap: digits are not touching, but they overlap vertically, and may not be linearly separable;

2. breakage: a single digit is broken into two or more pieces;

3. superimposition: a severe touching case where the digits are effectively superimposed on one another.

4. noncontextual strokes: interference from non-random markings that are not part of the digits;

5. just plain noise.

The first issue does not pose a considerable problem, however, each of the others presents a major challenge which could itself serve as the subject of a thesis or major paper; therefore, these will not be explored here. Naturally, the majority of real world inputs will be free of such segmentation problems, however, the proportion of problematic cases is not insignificant: one study found that 13% of U.S. ZIP codes contain touching digits [24]. So, not only is it simply interesting to devise computer methods of solving the problem of touching digits, the phenomenon itself is one that

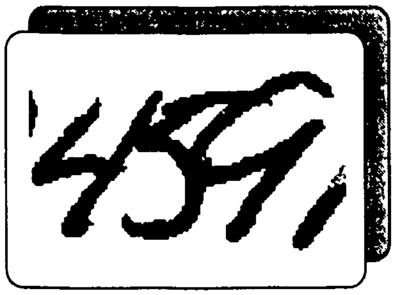
Figure 3: The 5 is simply connected on its left and complexly connected on its right.

is met with not infrequently in the real world, at least in certain realms of document processing such as mail sorting and cheque processing.

Within the class of touching digits we have some subclasses. Although many categories of touching configurations made themselves manifest over the course of the research, it is beyond the scope of this thesis to perform an extensive analysis of them. Such an analysis might lead to improvements in the method, however, that is a subject for further study. The topic is only mentioned here to draw attention to two of the most obvious types of connectivities between characters, namely, what is often termed *simply connected*, meaning connected at one place, and *complexly connected*, meaning connected at more than one place (see Fig. 3). In real world data, complex connections are rare compared to simple ones, nevertheless, they form one of the subclasses of connected digits to be handled.

1.2 Available methods

Other methods for separating touching digits have met with some success. A review of several of the most promising of them [8, 14, 13, 24, 31] appears in a report by the present author [33]. Another successful method has since appeared [17] which bears some similarity to the present method. Results of that method are compared with the present one in Chapter 9.

All of the above methods are structural in approach except that of Kimura *et al* [24] which is statistical. That method suffers from the restrictions that the cut it makes is always linear, with some postprocessing to fix up some cases that are not linearly separable; as well, it is only suitable for strings of 2 touching digits. All of the structural approaches except that of Fujisawa *et al* [17] suffer from vagueness in defining exactly where a path to cut two digits should enter and exit the digit string, and no published method has been seen that demonstrates robust handling of more than two connected digits ([31] includes some handling of 3 digits, but provides extremely limited test results for connected digits).

In this author's opinion, the method of Fujisawa *et al* [17] is the most coherent and exact seen to date, in that it analyses the upper and lower contours of the digit string in an effort to precisely identify a small set of significant candidate pairs of points between which to cut, something none of the other methods do. Its main drawbacks are that it is dependent on measurements of stroke thickness, and it relies too heavily on detection of certain configurations of strokes, a practice which is discouragingly ill-fated in handwriting analysis.

The present method is also dependent on measurements of structural features, but as the results show, the features selected appear to be more robust and/or meaningful than those used by Fujisawa. The critical weakness in all other methods seen is that they are all syntactic, i.e., there is no automatic learning capability. This is what really distinguishes the present method from the others. It is capable of learning the relative importance of each of the features it measures.

1.3 The proposal

As mentioned above, the present proposal is to provide a small number of segmentation hypotheses—five to be exact—sorted so that if a correct solution has been found it is likely to be closer to the front of the list than the back. The strategy of the method may be described informally as follows.

We observe that at each junction point of 2 touching characters one or both of the following is very likely to be true: (1) the junction point is at a discontinuity in the curve of the outer contour of the character string; and/or (2) the junction point is found in a vertical indentation of the outer contour. The latter observation was used to give some promising results for digit separation by Cheriet *et al* [8]. Incidentally, observation 1 is in keeping with the laws of perceptual grouping of Gestalt psychology which have also proven useful in methods to segment overlapping strokes [20, 22]. In the present method both of these observations are used to locate likely points on the outer contour where a separating path might enter and exit the character string. Discontinuities in the outer contour are detected with a corner-finding algorithm, and vertical indentations in the outer contour are found using a run length encoded version of the image.

Once a set of likely entry/exit points on the outer contour, called significant contour points (SCPs), has been found, a ranking based on features of the points is performed in an effort to find the most likely pair between which the separating path through the digit string should pass. A set of weights is trained in a separate offline phase in order to give meaningful relative importances to the SCP features measured. When the ranking is complete we have a list of cuts sorted in order of confidence.

The correct cutting path from an entry point in the outer contour to an exit point is very often a straight line, however, the path that separates a complexly connected digit from a string may be far from linear. This latter problem is not ignored here: the solution provided needs further development, but it has good potential.

We now proceed to a detailed description of each phase of the method.

Chapter 2

Preprocessing

2.1 Binarization

All images were first binarized by a simple thresholding operation. This is a very important step, but little effort was spent to achieve optimal performance since any images which could not be binarized satisfactorily by this method, and there were very few, were excluded manually from further processing. This step was done once offline in a completely different phase from the segmentation phase. All further preprocessing takes place as the initial part of the segmentation process.

2.2 Smoothing

Next, a smoothing operation is done to regularize the edges in the image and to remove small bits of noise. A white border one pixel thick is first added around the existing border of the image and a 3×3 mask (see Fig. 4) is passed over the entire image to smooth it. The mask begins in the lower right corner and processes each row from right to left moving upwards row by row. The pixel in the centre of the mask is the target. Pixels overlaid by squares marked 'X' are ignored. If the pixels overlaid by the squares marked '=' all have the same value, i.e., all zero, or all one, then the target pixel is forced to match them, otherwise it is not changed. This test is done 4 times for each target pixel, once for each possible rotation of the mask. The

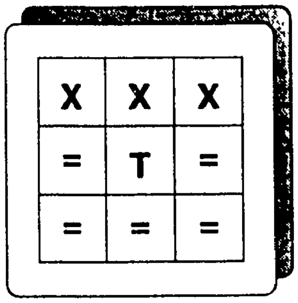

Figure 4: 3 × 3 mask for smoothing.

result is that single-pixel indentations in all edges are filled and single-pixel bumps are removed. Furthermore, the mask modifies the identical image that it scans, so that in certain cases, for reasons the reader will no doubt see, lines that are one pixel thick will be completely eroded (see Fig. 5). This is a small price to pay for a simple and efficient smoothing operation; indeed, it is extremely rare that anything other than noise is removed by this filter.

2.3 Size-doubling

After smoothing, the image is doubled in size if its height is lower then a certain number of pixels (see Appendix A for threshold values). The size is increased in order to magnify small features for further processing. Doubling is chosen because it is very cheap computationally, and a further smoothing operation can easily be incorporated. Smoothing is done by turning pixels black in the corners of any staircase encountered. Fig. 5 compares ordinary size-doubling with the present smooth-doubling method.

This concludes the preprocessing phase.

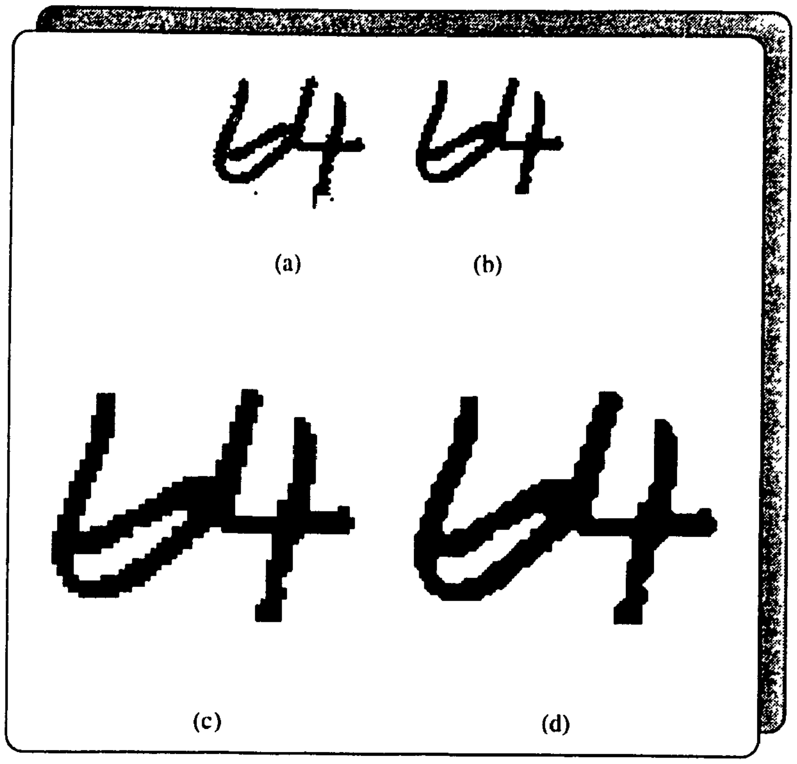

Figure 5: (a) input image, (b) smoothed (c) doubled, (d) smoothly doubled.

Chapter 3

Contouring

3.1 Introduction

The contour finding method used in this work is designed to quickly trace contours around regions of some given range of pixel intensities, i.e., the parameters of the algorithm are a rectangular image consisting of square pixels, and a single range of pixel values which are to be contoured. In the present application images are binary and foreground pixel regions are contoured, so we will simplify our discourse by referring only to foreground and background regions. Furthermore, foreground regions will often be referred to as *strokes*. The resulting data structure consists of a linked list of closed contours organized in such a way as to reflect the topological relationships of the contours. Each contour consists of a linked list of nodes called a *chain*, one node for each pixel in the contour. These chains follow outer contours in a counterclockwise direction, while inner contours are followed in a clockwise direction.

We define an outer contour as a cycle of 8-connected[1] stroke pixels which if followed in a *contourclockwise* direction always has a background pixel to the right of the direction of travel, except in the case where an outer contour touches one of the delimiting edges of the image. In this latter case, the region outside the image is to the right of the direction of travel instead of a background pixel. Similarly, we

[1]Two square pixels are said to be *8-connected* if they touch either on an edge or at a corner, thus, under this convention each pixel in an image, excepting pixels on the delimiting borders, has 8 neighbour pixels to which it is connected.

define an inner contour as a cycle of 8-connected stroke pixels which if followed in a *clockwise* direction always has a background pixel to the right of the direction of travel.

3.2 Contour data structure

As mentioned above, the contours of an image are extracted and stored in such a way as to reflect their topological relationships. The relationships that are captured are the order of occurrence of the leftmost pixel of the uppermost part of each contour as the image is scanned row by row, and the nesting of contours within other contours. Fig. 6 shows an example of this organization. The root contour is a dummy inner contour which does not appear in the contour graph of the figure. Any outer contours contained within an inner contour are linked horizontally to its right; any inner contours contained within an outer contour are linked vertically beneath it.

In the present implementation the contour data structure is defined so that a given node may be linked in both a *horizontal* and a *vertical* circular doubly linked list. If it is linked in both directions then it serves as a dummy head node in one of those directions, that direction depending on whether it is an inner or outer contour.

3.3 Chain data structure

The contour *chain* is a circular list of nodes, one node per pixel in the contour. It begins at the leftmost pixel of the uppermost part of the contour, but since any starting point in a cycle is arbitrary to some degree, and since some processing of contour chains requires uniform treatment of arbitrary subsequences of nodes in the chain, then the list structure is defined with no dummy head node. Many data relating to a given pixel of a contour may be important and can be stored in the chain node for that pixel. Two examples might be the coordinates of the pixel, and a measure of the degree of curvature of the contour in a region centred at that pixel.

In the present work contour chains are doubly linked for ease of processing and they contain numerous data items including the above examples. These data are

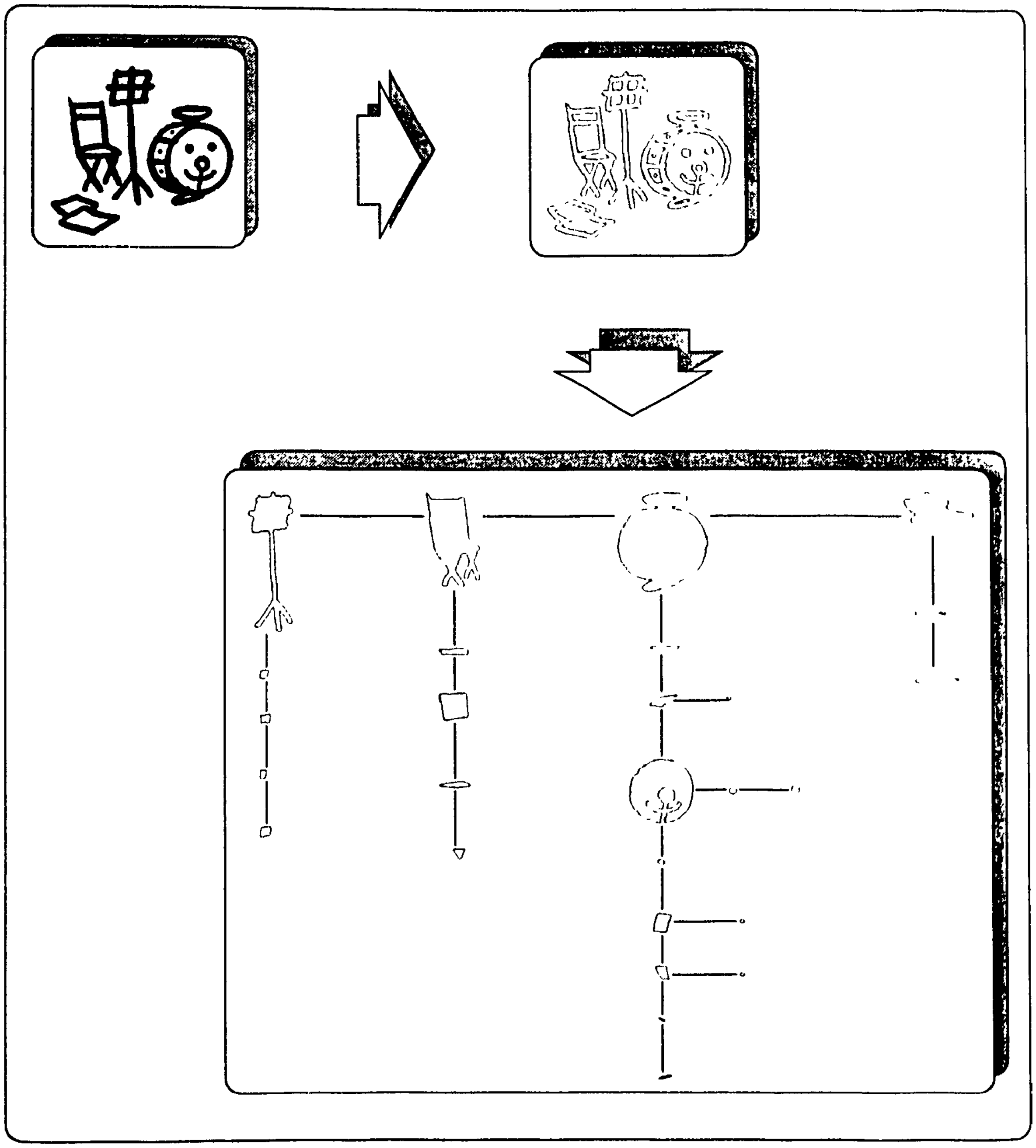

Figure 6: Extraction and list organization of the inner and outer contours of an image.

elaborated on at the appropriate places in this discourse. For example, data items pertaining to curvature are described in the chapter on corner detection (§ 4.2).

3.4 Contour extraction algorithm

There are two parts to the contour extraction process: the creation of the individual contour chains, and the insertion of each chain into the type of list structure depicted in Fig. 6. These two parts are done simultaneously, but we will deal with each separately.

3.4.1 Chain extraction

The algorithm for extracting contour chains is intuitively simple to describe. To state it informally, the image is scanned row by row from top to bottom, and from left to right within a row in order to find *runs of contiguous stroke pixels.* The runs in the current row are then matched up with 8-connected runs in the preceding row and operations are performed on chains in order to accommodate the new data (see Fig. 7):

- **Local Maximum**: when a run is encountered that is not 8-connected to a run in the preceding row a new contour chain is begun with two dangling ends to be completed during the remainder of the scan.
- **Continuation**: when a run in the preceding row is 8-connected to a single run in the current row the chain-ends from the preceding row are updated to accommodate the new run in the current row.
- **Stroke Split**: when two adjacent runs in the current row are 8-connected to a single run in the preceding row a local split in a stroke has occurred and a new contour chain is begun with two dangling ends.
- **Stroke Merge**: when two adjacent runs in the preceding row are 8-connected to a single run in the current row a local stroke merger has occurred and two chain-ends are linked together.

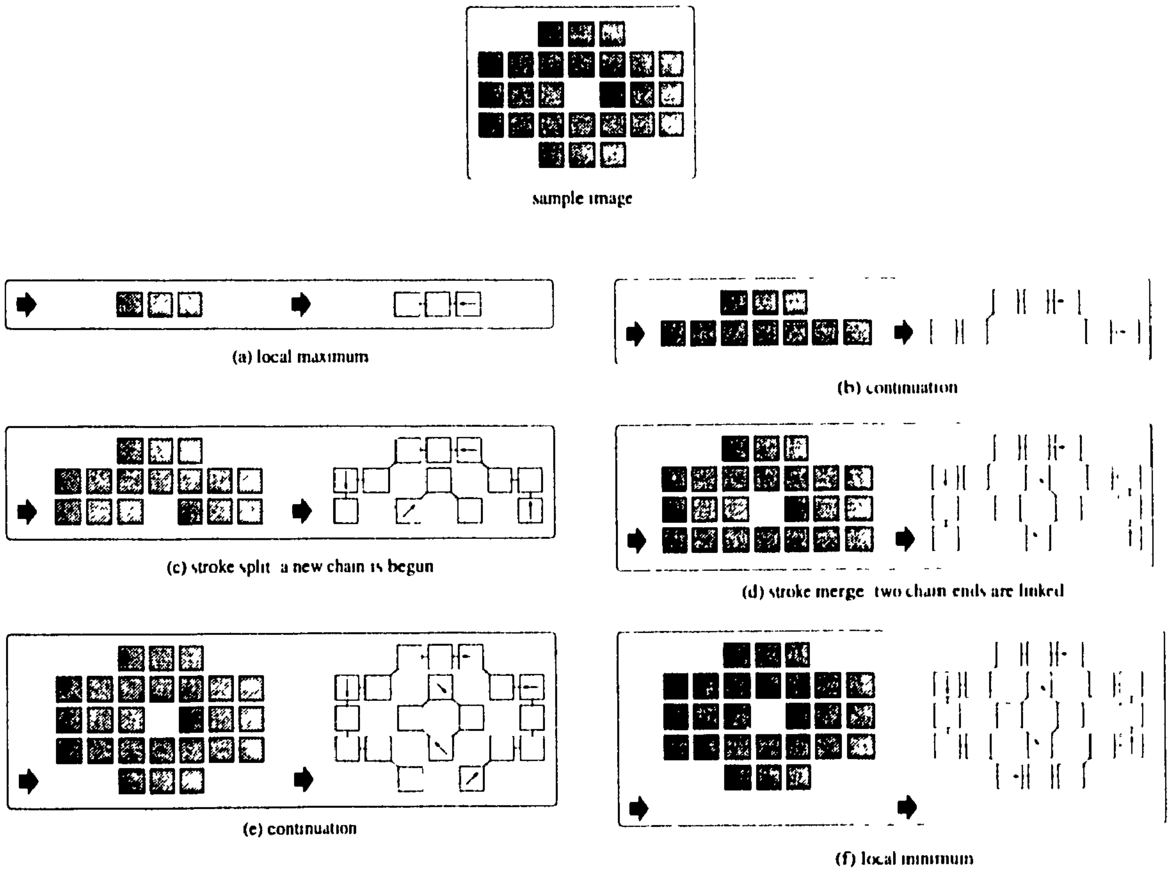
sample image
(a) local maximum
(b) continuation
(c) stroke split a new chain is begun
(d) stroke merge two chain ends are linked
(e) continuation
(f) local minimum

Figure 7: Steps in linkage of contour chains as an image is scanned. Each square represents one pixel, large arrows indicate the current scan row, small arrows indicate forward chain linkage direction.

- **Local Minimum**: when a run on the preceding row is not 8-connected to any run in the current row the two dangling chain-ends in the preceding row are linked together. This does not necessarily mean that the entire contour has become a cycle: it only means that a local minimum has been encountered.

3.4.2 Run length data structure

Each row is scanned and the run locations are recorded in an array of run length records. Each run has one dangling chain-end associated with its right hand side, and another with its left hand side. Each chain fragment, in turn, has 2 ends, and each end corresponds to some run. So the pertinent items that are stored in the run length record are:

- Pointers to the left and right chain-ends of a run are stored with each run length record so that when the next row is scanned the dangling threads can be updated.
- Pointers to the contour list nodes to which each of this run's two chain-ends belongs is stored (the chain-ends may belong to different contours or the same contour).
- Pointers to the runs at the other end of each of the two chains associated with the given run.

The meaning and purpose of the above should become clearer by reading on.

For the present algorithm it is only necessary to keep track of the runs in the current row and the previous row, however, in the implementation, the run length information for the whole image was stored in order to provide a richer information base for later processing, in particular, for further analysis of contour chains (see section 5.3).

3.4.3 Contour list creation

The first part of the contour extraction algorithm is the linking together of the chains and has been presented in the preceding sections. The second is the creation of the contour list to organise the set of contour chains in the image. In fact, this part is more complex in detail than the first part since it is done in parallel with the creation of the chains.

As soon as a new chain fragment is begun it is inserted into an appropriate position in the contour list. This may not be its final position since some chains may merge with other chains later in the scan. As well as merging of chains we must also handle splitting of strokes. When a stroke splits it produces multiple runs in subsequent rows which are connected globally, i.e., pixels from two distinct runs in the same scan row are linked in the same contour chain.

It is important in this algorithm to know which contour node corresponds to each dangling chain-end because when two chains merge we must determine whether or not they belong to previously distinct contour list nodes; if yes, one of the nodes must be deleted from the list. Therefore, as mentioned in a preceding section, we have pointers in the run length data structure to the contour nodes to which the left and right chain-ends belong, and one pointer each to the left and right *run* to which each chain-end is connected. The net result of this information capture is that after each row has been scanned and processed we have a record of which pairs of runs in that row are connected to each other by a chain.

In Fig. 8 we see an example of an intermediate stage in a scan where the contour list contains 9 chain fragments. Eventually, the list in this example will be reduced to a single inner contour linked below an outer contour. Since it can never be the case that two contour chains cross over each other[2] then, when scanning from left to right, if the left intersection of a chain with the current row is encountered, we may be sure that its right intersection will not be encountered before both the left and right intersections of any intervening chains have been encountered.

Given such a first-in-last-out organization it is logical to make use of a stack to determine the connectivity between runs within a row. Before proceeding to the algorithm we first completely enumerate all possible configurations of runs and contour

[2]This may not be immediately obvious. When a stroke is one pixel thick we have the case where more than one chain node is associated with the same pixel There is no question, however, of chains crossing. Incidentally, we note that a maximum of four chain fragments can pass through a single 8-connected pixel.

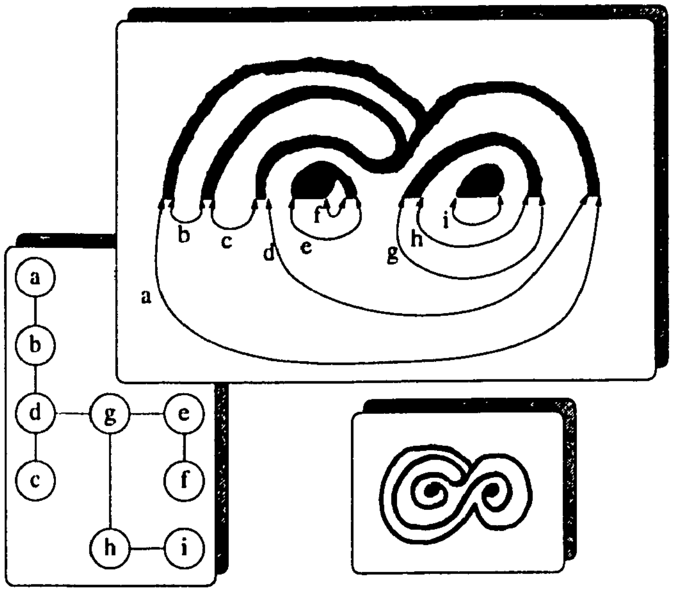

Figure 8: A doodle with of an outer contour and a single inner contour. At the intermediate scan stage shown enlarged there are 9 contour chains active with chain-ends indicated by arrows, and nesting shown in the graph.

fragments at any given point in the scan.

3.4.4 Configurations of runs and contour fragments

The following is a complete enumeration of the possible configurations of runs and contour fragments at any given point in the scan. Each configuration may be visualized with the help of Fig. 9.

1. If a run is not 8-connected to any run in the preceding row a new contour fragment begins at that run.

2. If a run is 8-connected to one or more runs in the preceding row one of the following is true about the *lefthand side* (LHS) of that run:

   (a) The run is the rightmost of a pair of runs initiating a stroke split and a newly-created contour fragment passes through its LHS.

   (b) The LHS of the run is the RHS of an inner contour fragment.

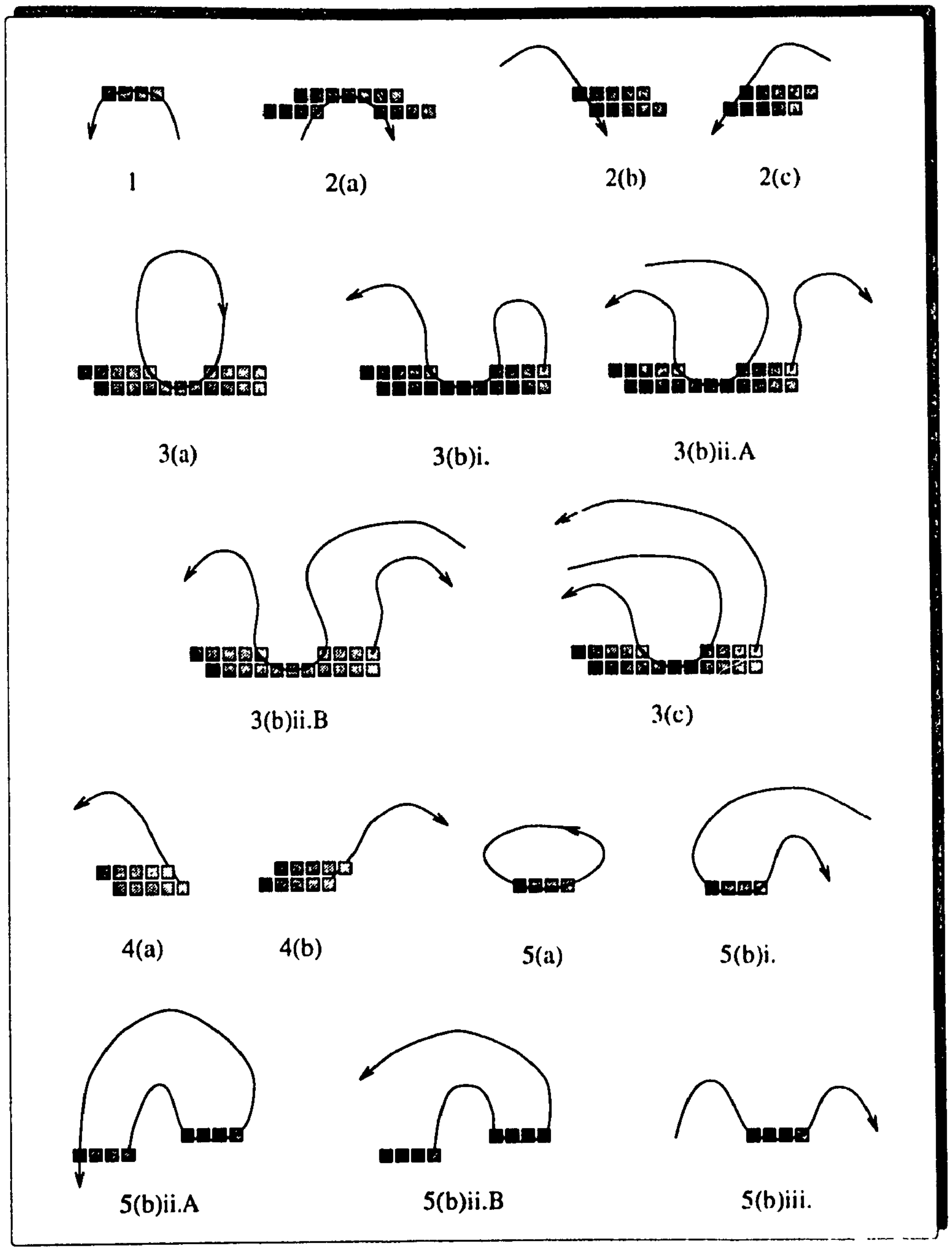

Figure 9: The possible configurations of runs and contour fragments at any given point in the scan (see text).

   (c) The LHS of the run is the LHS of an outer contour fragment.

3. If a run is 8-connected to more than one run in the preceding row, i.e., a stroke merge has just occurred, then one of the following is true about each of those 8-connected runs run_j in the preceding row and its connected successor run_{j+1} (except the last which has no successor):

   (a) The RHS of run_j is the LHS of an inner contour fragment, i.e., an inner contour becomes closed.

   (b) The RHS of run_j is the RHS of an outer contour fragment *and* one of

      i. The chain at the RHS of run_{j+1} is a chain whose other end falls *at* the LHS of run_{j+1}.

      ii. The chain at the RHS of run_{j+1} is a chain whose other end falls *to the right of* run_{j+1} *and* one of:

         A. The chain at the LHS of run_{j+1} is a chain whose other end falls to the left of run_j.

         B. The chain at the LHS of run_{j+1} is a chain whose other end falls to the right of run_{j+1}.

   (c) The RHS of run_j is the RHS of an outer contour fragment *and* the chain at the RHS of run_{j+1} is a chain whose other end falls to the left of run_j.

4. If a run is 8-connected to one or more runs in the preceding row one of the following is true about the *RHS* of that run:

   (a) The RHS of the run is the RHS of an outer contour fragment.

   (b) The RHS of the run is the LHS of an inner contour fragment.

5. If a run in the preceding row has no 8-connected run in the current row then one of the following cases is true:

   (a) The contour associated with the run becomes completely closed.

(b) The contour fragments associated with the run are linked, but the contour of which they form a part remains open in one of three ways:

   i. The chain at the LHS of the run is a chain whose other end falls to the right of the run.

   ii. The chain at the RHS of the run is a chain whose other end falls to the left of the run in one of 2 ways:

      A. The chain falls at the same run as the chain stemming from the LHS of the run in question.

      B. The chain falls to the left of the above run.

   iii. The two chains associated with the run fall on either side of the run.

3.4.5 Determination of intra-row run connectivities

The idea here is to cause the *lthread* and *rthread* pointer fields of each run length record to point to the run length record, in the same row, at the other end of the chains associated with the left and right end respectively of the run. In other words, at any given time we would like to be able to draw the arrows of Fig. 8. As previously mentioned, this information is needed if we are to decide which contour entry to delete from the contour list when two chains merge. No look-ahead is used in this method, so when these thread pointers cannot be immediately determined in the left-to-right processing of a row, the run length record in question is pushed onto the stack. The stack is empty both at the beginning of processing for each row and at the end of processing for each row.

We now proceed to the handling of each of the cases in the preceding section. For the following, i is the index to the current run being processed in the current row, and j is the index to a run in the preceding row which is 8-connected to run_i.

1. In this case no stack operations are necessary.
$lthread_i \leftarrow rthread_i \leftarrow run_i$.

2(a). In this case no stack operations are necessary.
$lthread_i \leftarrow run_{i-1}$. The RHS of run_i is processed later.

2(b). $lthread_i \leftarrow stack_{top}$;
$rthread_{stack_{top}} \leftarrow run_i$;
if $lthread_{stack_{top}} \neq null$ then pop.

2(c). push run_i;
$rthread_{lthread_j} \leftarrow run_i$;
$lthread_i \leftarrow rthread_i \leftarrow null$.

3(a). $lthread_{rthread_j} \leftarrow lthread_{j+1}$;
$rthread_{lthread_{j+1}} \leftarrow rthread_j$.

3(b)i. $rthread_{lthread_{j+1}} \leftarrow stack_{top}$.

3(b)ii.A. $lthread_{stack_{top}} \leftarrow stack_{top-1}$;
$rthread_{stack_{top-1}} \leftarrow stack_{top}$;
pop;
if $lthread_{stack_{top}} \neq null$ then pop.

3(b)ii.B. This case is handled in the same way as case 3(b)i.

3(c). This case is handled in the same way as case 3(b)ii.A.

4(a). $rthread_i \leftarrow stack_{top}$;
$lthread_{stack_{top}} \leftarrow run_i$;
pop.

4(b). $lthread_{rthread_j} \leftarrow run_i$;
$rthread_i \leftarrow rthread_j$;
if $stack_{top} \neq run_i$ then push run_i.

5(a). In this case no stack operations are necessary.
$rthread_{lthread_j} \leftarrow rthread_j$;
$lthread_{rthread_j} \leftarrow lthread_j$.

5(b)i. This case is handled in the same way as case 5(a).

```
5(b)ii.A. $lthread_{stack_{top}} \leftarrow rthread_{stack_{top}} \leftarrow stack_{top}$;
    pop.

5(b)ii.B. $rthread_{stack_{top}} \leftarrow rthread_j$;
    $lthread_{rthread_j} \leftarrow stack_{top}$;
    pop twice.

5(b)iii. $rthread_{stack_{top}} \leftarrow rthread_j$;
    $lthread_{rthread_j} \leftarrow stack_{top}$.
```

3.4.6 Closing comment

The configurations of runs and contour fragments proceed in an orderly fashion as a row is scanned, e.g., we cannot encounter an inner contour before we have passed its corresponding outer contour. The possible configurations have been enumerated (§ 3.4.4) roughly in the order in which they may be encountered for the intersections of a given connected component with a row. The method of detecting which of the enumerated cases applies in a given circumstance is too straightforward to be elaborated on in this work.

Chapter 4

Corner Detection

4.1 Introduction

It is beyond the scope of this thesis to do an extensive analysis of contour *corner point*, or *dominant point* detection methods. It appeared that it might be desirable to detect corners as a step in segmentation, therefore, a corner detection method was sought. The criteria used to select a method from the literature were: (1) in general, it should give "good" results, (2) it should execute quickly, and (3) it should be easy to implement. Unfortunately, while there are some comparative analyses of corner detection methods in the literature [38, 27], it is difficult to judge between methods since there are neither extensive standard data sets for testing, nor clearcut ways of automatically measuring the correctness of a result. The algorithm selected is that of Held [21, 3]. The method of Held was chosen because it performed well in tests with another popular method, that of Teh and Chin [38], and it had some success in achieving the following design goals:

- no need for any input parameters,
- good behaviour for widely differing shapes,
- rotation and scale invariance,
- speed.

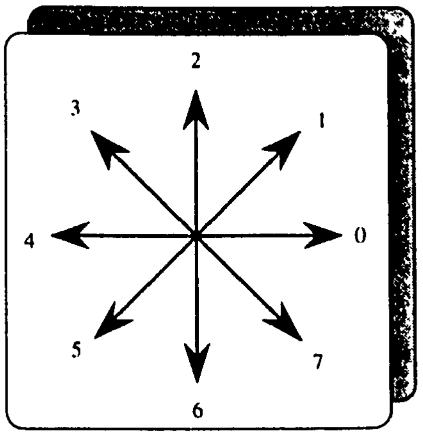

Figure 10: The Freeman codes for 8 directions.

4.2 The algorithm

The algorithm consists of two phases. In the first phase, points of high curvature are detected, however, the algorithm tends to find a cluster of points in the vicinity of a corner instead of just one point, hence, the need for a second phase which thins these clusters. The method of cluster-thinning proposed in [3] proved to be ineffective for the present type of data, thus, as will be explained, a different strategy was devised.

Although the corner detection algorithm used is well documented elsewhere it is perhaps appropriate to give some details here. It is one of a class of corner detection methods which operate on *chain coded* contours. Freeman code [16] is a popular way of encoding the direction to the next 8-connected pixel from a given pixel in a sequence. Fig. 10 shows the integer code value corresponding to each direction. The *differential chain code* value at a given pixel is computed by subtracting its chain-code value from that of the next pixel giving an encoded measure of the successive changes of direction, e.g., a difference of 1 indicates a 45-degree change of direction. This information is, however, too localized to allow decisions to be made as to corner points. These two code values as well as the N-code, to be described next, are stored in the chain node for each pixel in a contour chain.

N-code, or Gallus-Neurath code [18] is one way of using differential code to locate corners. It computes a weighted sum of differential chain codes in a sequence of pixels

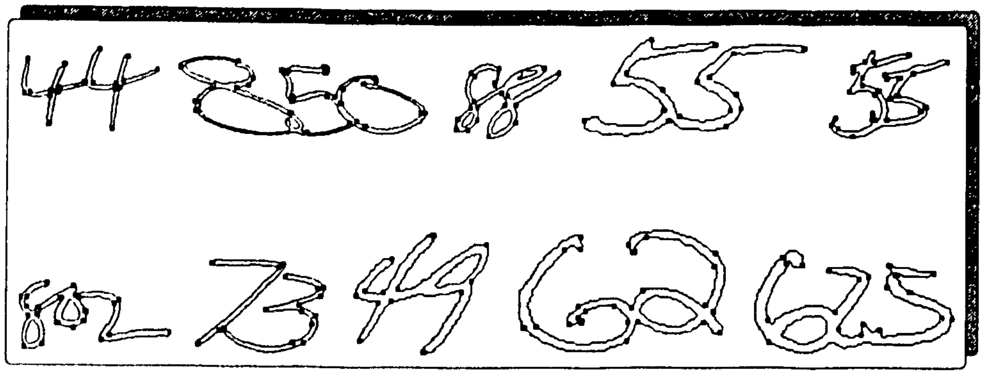

Figure 11: Examples of corner detection.

of length $2N - 1$ centred at the i^{th} pixel in the contour:

$$c_i^N = Nc_i + \sum_{k=1}^{N-1}(N-k)(c_{i-k} + c_{i+k}) \tag{1}$$

where c_i is the differential chain-code at the i^{th} pixel. The above formula gives higher weight to direction changes close to the point in question and lower weights to direction changes farther away from it. It can be used to find points of high curvature: the higher the absolute N-code value, the sharper the corner. The question is, what value should be chosen for N? In Held's method, global dominant points are located by choosing first a large value of N proportional to the length of the contour, then, successively smaller values of N are used to recursively find ever more localized points of direction change in the regions between known dominant points.

As was mentioned above, the so-called triangulation method used by Held for thinning clusters of dominant points proved to be ineffective for the present application, so another method was devised. In this other method, a pass is made through the contour, and corner points that are less than a small threshold distance, t_c, from each other are grouped in the same cluster. Each cluster is then reduced to those corners in the cluster for which the absolute N-code is greater than a threshold:

$$|c_i^N| = |c_i^{t_c}| \geq 2t_c \tag{2}$$

The idea is to keep calculations simple and to minimize the number of tunable parameters. In fact, there are no floating point operations anywhere in the implementation, but the number of parameters listed in Appendix A for this algorithm is perhaps greater than would be desired.

Inevitably, as with other thresholding methods, undesirable results can be produced: spurious corners may be detected, and actual corners may be overlooked; however, as is explained in Chapter 6, the segmentation strategy is not critically dependent on high accuracy from the corner detector. Fig. 11 shows some results of the algorithm.

Chapter 5

Contour Regions

5.1 Introduction

Another step in the segmentation strategy is the analysis of the outer contour of the digit string in order to locate vertical indentations from the top and bottom. The result is that sequences of pixels of each contour in the image are subdivided into 4 region types (see Fig. 12):

1. open region,
2. mountain region,
3. valley region,
4. hole region.

Valleys and mountains are extremely useful for segmentation purposes, since the digits are usually joined in these regions. The region types are derived from the work of Cheriet *et al* [8], where certain regions of the ***background*** of the image were isolated. In that method so-called 'face-up valleys' and 'face-down valleys' were located by passing the image through a number of filters. The drawback with that method, however, is that there are multiple passes through the image and the regions of background which are found are divorced from the stroke contours. The present method

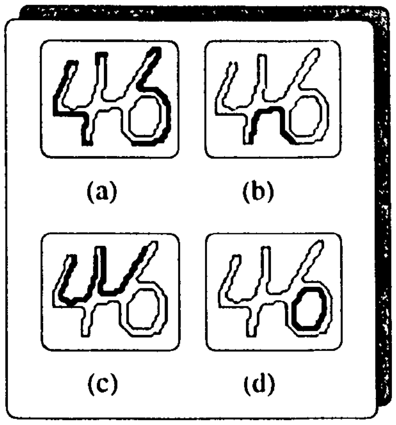

Figure 12: Contour regions: (a) open regions, (b) mountain region, (c) valley regions (d) hole region.

is more efficient in that it performs basically the same function as the other in locating vertical indentations between foreground pixels in a connected component, but it makes use of the contour chains and run-length information already extracted during the contouring phase, thus avoiding multiple passes through the image. Furthermore, it gives the contour boundaries where the background regions of the other method and the stroke regions meet, thus, we have more pertinent information at a lesser cost.

5.2 Algorithm for assigning region types

Given that nodes in an outer contour are 8-connected and linked in the counterclockwise direction, the basic algorithm used here to determine the region type of each pixel is:

```
$n_1 \leftarrow$ leftmost topmost node in chain;
while there are nodes without an
assigned region type do
  if compass direction from $n_1$ to
  next node $\in$ { W, SW, S, SE } then
    if $n_1$ is the leftmost pixel in its
```

```
      run of stroke pixels *and* is connected
      to some contour pixel $n_2$ on the same scan
      line, but on another run to the left
      of $n_1$ then
        set the region type of each pixel
        on the contour from $n_1$ to $n_2$ to
        *valley region*;
        $n_1 \leftarrow$ next node after $n_2$;
      else
        set the region type of $n_1$ to
        *open region*;
        $n_1 \leftarrow$ next node;
      end
    else
      if $n_1$ is the rightmost pixel in its
      run *and* is connected to some
      contour pixel $n_2$ on the same scan line,
      but on another run to the right
      of $n_1$ then
        set the region type of each pixel
        on the contour from $n_1$ to $n_2$ to
        *mountain region*;
        $n_1 \leftarrow$ next node after $n_2$;
      else
        set the region type of $n_1$ to
        *open region*;
        $n_1 \leftarrow$ next node;
      end
    end
  end.
```

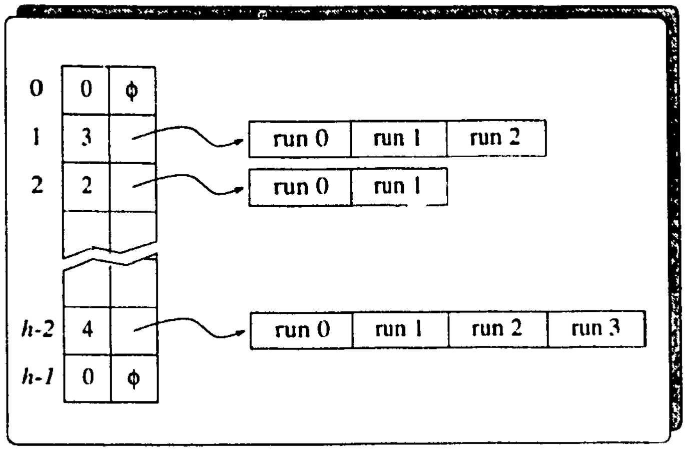

Figure 13: Run length encoded version of an image with h rows.

5.3 Run length image

When contours were extracted (see chapter 3) it was mentioned that we also captured a run length encoded version of the image. It is now appropriate to define the organization of this data structure: an array with one element for each row of the input image is created. Each entry in this array is a record containing the number of foreground runs in each row, and a pointer to an array of runlength records (see section 3.4.2), one record for each run (see Fig. 13).

Note that in the runlength data structure we have pointers to the contour to which each corresponding chain belongs, and in each chain node we have the row and the run number corresponding to that node. Thus, we have efficient cross linkage between any chain node and the run length image, and *vice versa*. This data enables the speedy execution of the above algorithm.

Chapter 6

Significant Contour Points

6.1 Introduction

The identification of *significant contour points* (SCPs) on the outer contour of the connected digit string is a key goal of the present segmentation strategy. The two most significant points are those between which a cutting path passes that separates the leftmost digit from the string (normally, there will be many pairs of points from a neighbourhood satisfying that requirement). In this chapter we define the qualities that render one point more significant than another, and the features that are measured in order to rank the significance of points on the contour.

6.2 Types of SCPs

We recognise three types of SCPs:

1. points found in the corner detection phase,

2. the maximum (minimum) of each mountain (valley) provided the condition stated below is satisfied,

3. the exit points of the imaginary straight lines formed by, wherever possible, extending the contour through the stroke at concave corners in mountains and

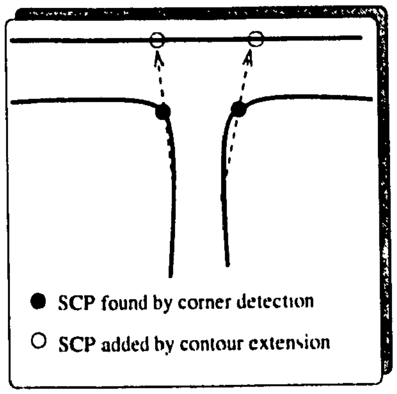
● SCP found by corner detection
○ SCP added by contour extension

Figure 14: SCPs added by extending contours through a stroke.

valleys (see Fig. 14), provided the conditions stated below are satisfied. The second reference point used to construct the line is found by backing up from the corner point along the contour chain a threshold number of pixels .

Conditions:

1. when identifying SCPs of types 2 and 3, if a candidate point falls closer than a small threshold distance from another SCP already identified it is not added.

2. If a contour extension passes through a hole region then the SCP candidate is not added.

SCPs identified by extending the contour from a mountain (valley) corner are given the region type *valley* (*mountain*), in order to enhance their subsequent ranking over that of corners in open regions. For the same reason, if a would-be candidate found by extending a contour at corner *a* is disqualified due to condition 1, the existing SCP *b* causing the rejection has its region type changed to *mountain* if the region of *a* is *valley*, otherwise, the region of *b* is set to *valley*.

The goal, of course, is to identify the correct pair of SCPs for the cut, while minimizing the number of candidates. Identification of the above types of SCPs, together with the limiting conditions will usually find the correct pair, although, there may be many spurious as well as reasonable candidates; however, in section 6.3 a method of ranking the significance of all of the candidates is described. In Fig. 15

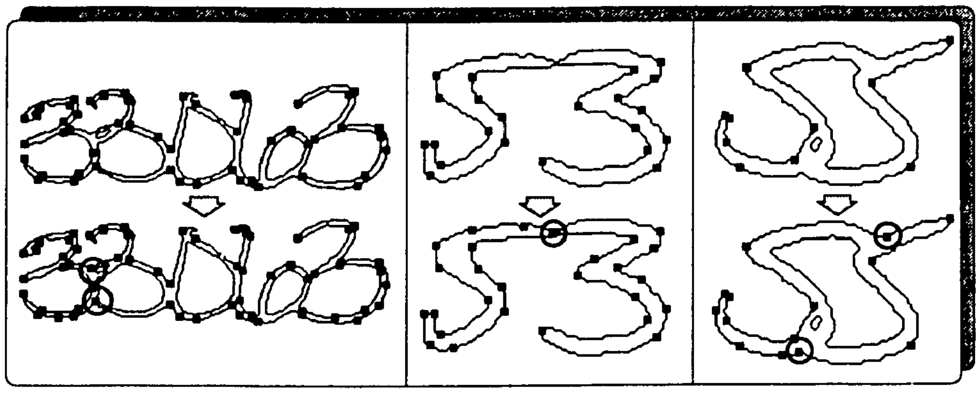
Figure 15: The correct SCP pair included due to addition of SCPs of type 2. In each case the top image shows detected corners and the bottom shows all 3 types of SCPs.

are some examples which justify the inclusion of SCPs of type 2. Similarly Fig. 16 shows examples of type 3, while Fig. 17(a) gives examples where the correct SCP pair is not found. In many such failures it is still possible to make a not unreasonable cut using the available SCPs (see Fig. 17(b)).

6.3 Ranking pairs of SCPs

A set P containing the SCPs is created. The preceding steps should result in the inclusion in P of at least one desirable pair of points between which to pass the cutting path. To locate this pair we measure 9 features of each SCP pair. First, a set C is created containing all combinations of p_1 and p_2 where $p_1, p_2 \in P$ such that each of p_1 and p_2 is chosen from a different region type (mountain, valley, or open). Next, each of the features is measured giving 9 *credit* values which are accumulated in an overall score s_{c_i} for each pair c_i, $c_i \in C$. The higher the score, the more likely it is that this pair will give a good cut. Before each credit is added to the score, however, it is multiplied by a weight described in the chapter on training (Chapter 8). Credits are assigned based on the following features:

1. a fixed number of credits for each mountain/valley pair;

2. a fixed number of credits for each SCP that was found by corner detection;

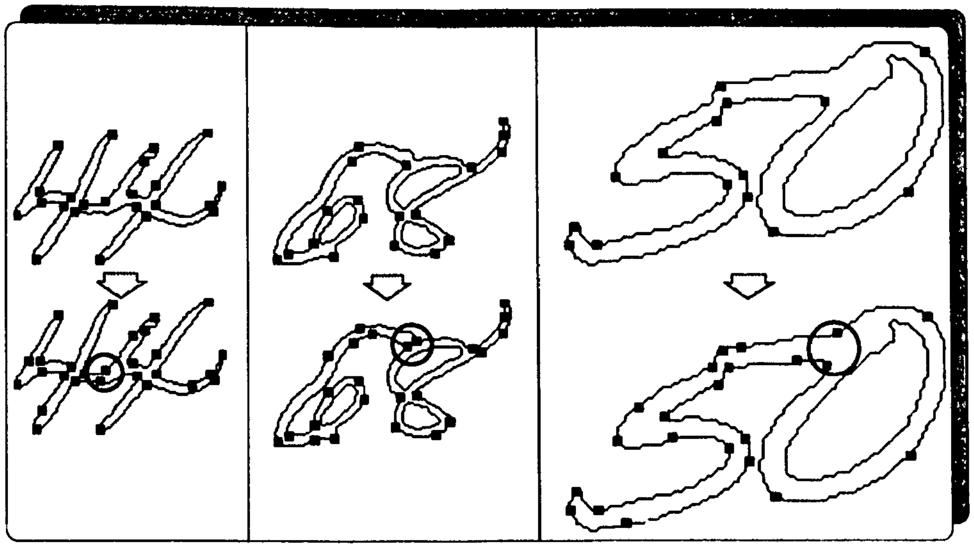

Figure 16: The correct SCP pair included due to addition of SCPs of type 3. In each case the top image shows detected corners and the bottom shows all 3 types of SCPs.

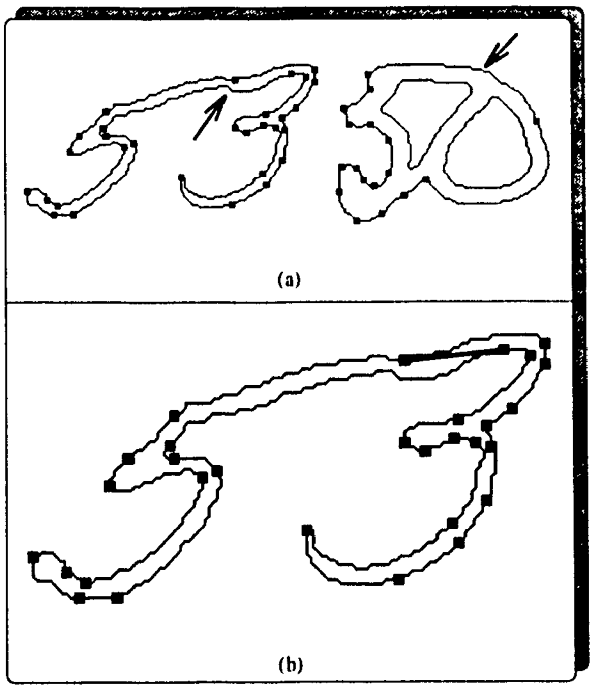

Figure 17: (a) failure to locate the correct SCP by identification of SCPs of all 3 types, (b) a feasible cut using the available SCPs.

3. credits for the nearness to each other of the points in the pair;
4. credits for the sharpness of the concavity at each of the SCPs;
5. credits for the nearness, where applicable, of an SCP's valley (mountain) to the top (bottom) of the image;
6. credits, where applicable, for the distance of an SCP from the bottom (top) of its mountain (valley);
7. credits, where applicable, for the degree to which a valley corner is above a mountain corner in the image;
8. credits for the degree to which stroke pixels outnumber background pixels in an imaginary straight line drawn between the pair;
9. credits for the nearness of the pair to the left hand side of the image.

Credit values, including fixed ones, are normalized according to the height of the bounding box of the given digit string. After the score for each pair has been computed the pairs are sorted from highest score to lowest, thus, the more favourable cuts should appear towards the front of the list. Notwithstanding a high ranking, if a cut would produce a component too small to be considered a digit, it is disqualified from the candidate list. Several thresholds are involved in this decision (see Appendix A).

Chapter 7

Making the Cut

7.1 Introduction

Given two points on the outer contour of a connected digit string, how should we guide a cutting path from the entry point to the exit point? This is no mean problem, especially when separating digits that are connected in two or more places.

7.2 Cutting strategy

In most real world cases, digits, if they touch, do so in one place and are separable by a simple straight line cut. The method used here, however, is moving towards being able to separate complexly connected digits where the cut is non-linear. The strategy is simply to attempt a straight line cut, and in regions where the straight line passes through background pixels to follow the contour until the straight line passes back into stroke pixels (see Fig. 18 for examples). The reason for following the contour when the straight line passes through background regions is that we have the option to cut or preserve stroke loops as will be shown shortly.

It may be interesting to mention a problem that arises with this approach due to the limitations of 8-connectivity. It sometimes happens that a digitized straight line will cross another such that they have no pixels in common as illustrated in Fig. 19. In this example the line formed from white pixels has passed from one hole region into

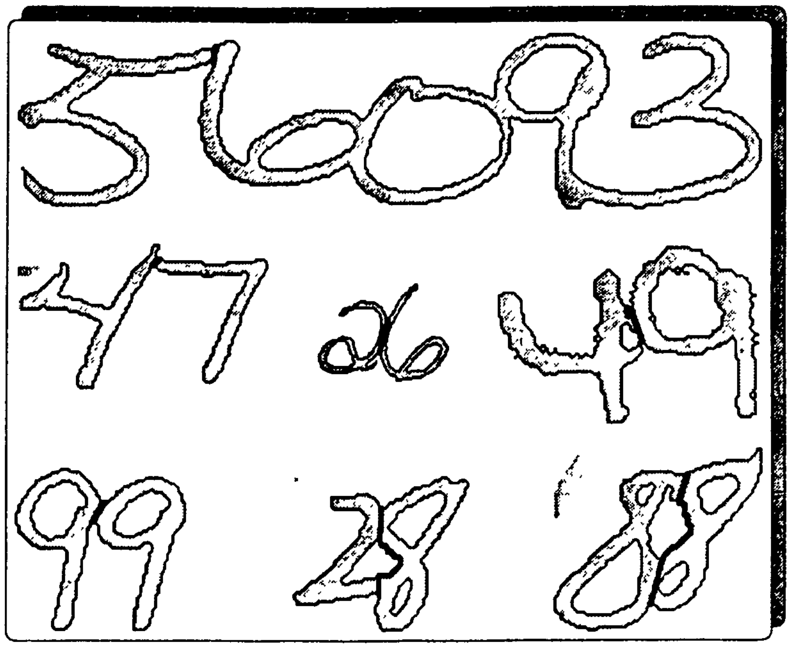

Figure 18: Examples of cutting paths between 2 points on the outer contour.

another one. Such problems are detected only when the cutting line finally strikes a dark pixel. With the present method the straight line always finishes on a dark pixel since we are cutting from one contour pixel to another. We can, therefore, detect the problem by ensuring that the pixel we have struck belongs to the contour we are expecting. When such problems are encountered in the present system, the cut is abandoned as too complex, and another is attempted using different end points.

7.3 Splitting the stroke

Once the cutting path has been determined we must split the single connected stroke component into two new components. The method used here is to trace the cutting path and the outer contour of the digitstring onto a blank image, im_c. Next, we initialize another blank image, im_l to contain the left component. We then follow the lefthand region of the contour chain counterclockwise and copy the contents of each corresponding row from the original image, pixel by pixel, walking inwards until another contour pixel is encountered in im_c. The right hand component is extracted

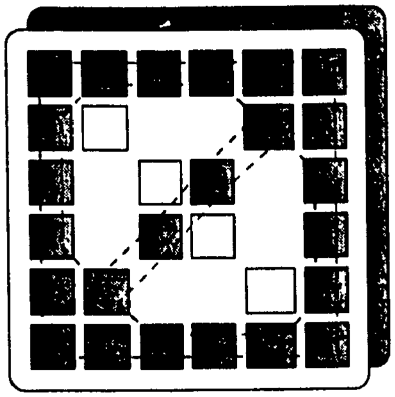
Figure 19: An 8-connectivity problem, dashed lines indicate contours.

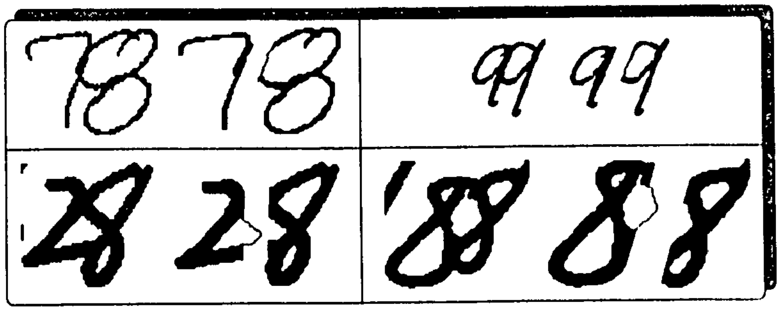
Figure 20: Handling of cuts that pass through holes.

in a similar fashion.

7.4 Preserving loops

When following the contour around holes we have the option to preserve or cut a loop depending on which side of the hole we follow, however, in the present method this feature was not exploited; instead, when passing through a hole we simply follow the shortest path along the inner contour. It was beyond the scope of this thesis to investigate other ways of deciding which path to take along an inner contour. Fig. 20 gives examples showing both desirable and undesirable results of the present method.

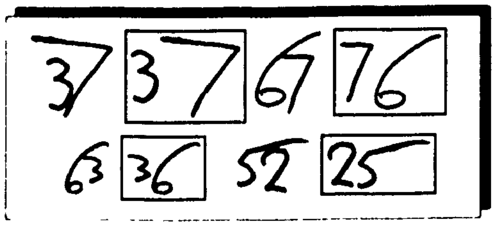

Figure 21: Some results of the method used to determine the leftmost component.

7.5 Knowing right from the left

Another problem which was not investigated deeply here is that of deciding which component of a split is the leftmost and which the rightmost. Given two points on the contour, a and b, the present method chooses the rightmost component by determining whether the directed contour arc ab contains a node on the right boundary of the bounding box of the digit string. If yes, then this arc is assumed to trace the righthand component, else the lefthand component.

Fig. 21 gives examples where this method does not give the desired result, i.e., having a pixel on the right border of the bounding box does not guarantee that a component is the rightmost one. The reasons this strategy was used, however, are that it is simple, and it is more reliable than taking the arc that touches the left border of the bounding box as the leftmost component, because, in the general case of separating the leftmost digit, the component remaining on the right extends further to the right then the left component does. Nevertheless, this criterion is not acceptable for use in a critical application due to the fact that in rare cases it can result in reversal of the order of digits when separating them.

Chapter 8

Training the System

8.1 Introduction

A simple summation of the 9 credit values determined in section 6.3 is alone not sufficient to perform a consistently good ranking of the SCP pairs, because determination of the relative importance of each feature measured is problematical. It was therefore necessary to find some coefficient with which to multiply each credit value before adding it to the final score for each SCP pair. This entailed the development of a method to search for coefficients, or weights, that would cause the best SCP pair to be favoured.

It would be interesting to attempt to formulate this problem for a neural network, but since such a formulation was not readily apparent, that idea was not pursued. For a given imag, there will be an unpredictable number of SCP pairs. Each SCP pair has 9 feature measurements associated with it and the goal is to locate the most likely pair by assigning it the highest score. Two other popular methods of performing random searches are simulated annealing and genetic algorithms. The latter method was selected due mainly to its computational simplicity.

8.2 The genetic search

A genetic search was performed to locate an optimal combination of the 9 coefficients. First, the credit values (feature measurements) were normalized according to the height of the digit string, then the typical range of credit values was ascertained for a set of digit strings, and modifications were made to parameters to ensure that the bandwidth of these integer feature measurements was roughly in the range 0 to 400. A bandwidth for the weights was then chosen as 0 to 511, or 9 bits. The idea was to minimize the search space size while allowing sufficient bandwidth to enable the weights to discriminate the relative importances of the features effectively.

Formulating the problem in terms of genetic algorithms, we have a population composed of strings of length 81 bits, i.e., 9 weights, each 9 bits long concatenated together; thus the size of the search space is 2^{81}, or 2.42×10^{24}.

Next we must have a function which evaluates the fitness of each string. We developed software tools that enabled the entry of the correct SPC pair for a given image using a mouse pointer. The coordinates of the two points were stored with each image in the training set for later processing. At training time the image and coordinates were read, the list of SCP pairs was determined as per the methods described in chapter 6, and the nearest pair to the entered coordinates was taken as the correct one. Then the SCPs were sorted using each of the weight strings in the genetic population, and the negated ordinal position of the correct pair in the sorted list was accumulated over all the images for each weight combination. The result was a score for each weight combination giving a measure of how close to the front of the list it sorted the correct cut over all the images in the training set. The strings in the population were then reproduced for the next generation with respect to this measure of fitness.

8.3 Crossover operator

A simple genetic crossover operator takes two strings s_1 and s_2 and interchanges a substring of random length l beginning at random position p in s_1 with the substring of the same length and same position in s_2. This method was tried and was found to converge very slowly (many days), therefore, a different operator was hypothesized and it gave much more rapid convergence (a couple of days). This operator is basically identical to the first except that l is constrained to be a multiple of 9, and p is constrained to fall on a 9-bit boundary. With this operator, only whole 9-bit weights are interchanged between strings so that in effect each string is really a string of 9 integer weights rather than a string of 81 bits.

8.4 Epidemics

Another method that was used to speed up a convergence was to reinitialize the population, from time to time, with copies of the currently best performing string. Thus, the search was constrained to the vicinity around the current best peak, and could more rapidly improve on it. Also, the mutation operator was modified at this stage of the search so that it could fine-tune the weights. The conventional mutation operator chooses a random number greater than zero in some predefined range and complements that number of bits at random in the population. The fine-tuning mutation operator used here, instead of flipping the bits at random, chooses at random a 9-bit weight in the population and *increments* it by some integer randomly chosen from a small range centred at zero. In effect, as the convergence proceeds the search become less random, more directed.

Chapter 9

Experimental Results

9.1 Introduction

The system was implemented in C code. Details of the implementation are not of great interest here. The idea was simply to produce a system that would segment touching digits. The overall goal is to develop a somewhat general purpose OCR image processing system of which one component is a segmenter. Therefore, there is much functionality included in the system that is not directly related to this thesis, e.g., the input/output of images can be done using 8 different image file formats. The result is that there are about 18,000 lines of new code.

Testing of the system was done on 3 different test sets after training had first been performed using a separate training set.

9.2 Training

As described in chapter 8, the 9 feature values determined for each SCP pair are each multiplied by a weight to give each a relative importance before adding them to the final score. These weights were determined in a separate training phase by marking the correct SCP pair in each image of a training set, and then using a genetic algorithm to search for an optimal combination of the 9 weights (see Chapter 8. Weight combinations were ranked by scoring each on how close to the front of the list it sorted the correct SCP pair over the whole training set.

The system was trained on 212 images of touching handwritten digit strings extracted from the bu0100 and bu0200 subdirectories of the USPS Office of Advanced Technology Database of Handwritten Cities, States, ZIP Codes, Digits, and Alphabetic Characters. Each image consists of from 2 to 5 touching digits extracted from U.S. mail pieces. The genetic population contained 200 randomly chosen weight combinations for the initial phase of the search, and typically, it took about 20 minutes to process a generation.

9.3 Testing

The system was tested on a total of 946 images of touching digits divided into three sets:

1. **CENPARMI test set**: 295 samples of from 2 to 4 touching digits written by 12 authors scanned at 300 dots per inch (DPI).

2. **Fujisawa test set**: 460 samples of touching pairs of digits written by one author scanned at about 200 DPI.

3. **USPS test set**: 191 samples of from 2 to 5 touching handwritten digits from U.S. mail pieces scanned at 300 DPI. While this set is the most realistic of the three it is also the smallest, but, that is almost all of the touching digit samples designated for testing in the USPS data base mentioned in the preceding section. Some dozen or so samples were not included in this test because they contained broken digits, or severe noncontextual noise.

The first set was gathered here at Concordia University by Mohamed Cheriet and others. Results for the second set have been published by Hiromichi Fujisawa *et al* [17] who kindly supplied CENPARMI with this data. The third set was extracted from images of U.S. ZIP codes in the testing subdirectories of the above mentioned USPS data base (Appendix B gives exact information).

	1	2	3	4	5	Fail
C	55.9	78.3	87.5	92.2	93.6	6.4
F	47.6	73.5	90.7	97.0	98.3	1.7
F_w	42.6	76.8	92.0	97.7	99.1	0.9
U	59.7	81.2	89.5	93.2	95.8	4.2

Table 1: Percentage of samples whose leftmost digit was successfully separated within 1 attempt, 2 attempts, etc., up to 5 attempts. C = CENPARMI test set, F = Fujisawa test set, F_w = Fujisawa *weighted*, U = USPS test set.

The results are summarized in Table 1 in terms of how many attempts were needed to correctly separate the leftmost digit, and the complete results for the USPS test set are shown in Appendix B. Performance was judged by visually inspecting images such as those in the appendix. An attempt was made to integrate a recogniser into the system to automate the performance measurement, but it was postponed because the available recognisers required modifications to handle the kinds of images output by the segmenter, and doing such modifications was outside the scope of this thesis.

9.4 Comparison with other methods

It was possible to compare the present method with two other published methods since common data was obtained. The two methods are that of Cheriet *et al* (CHS) [8], and that of Fujisawa *et al* (FNK) [17].

The CHS method was tested on 120 samples taken from the CENPARMI test set, and yielded a successful separation in 80.8% of cases.

The success rate of 'about 95%' for the FNK method reported in the *IEEE* paper was obtained after 're-calibration' of the 'raw' results according to the frequency counts appearing in Table 2. This data was provided in recent months by Mr. Fujisawa through correspondence. As well, the 'raw' performance was reported as follows.

> For the first sample sheet, we obtained 191 correct separations out of 230 touching pairs (83.0%). For the second, we obtained 187 correct separations (81.3%).

This gives an overall correct separation rate for all 460 samples of 82.3%. The *IEEE* paper gives no upper limit on the number of cuts attempted, however, some examples in the paper show 3 candidate cuts.

The row labelled F_w in Table 1 gives the re-calibrated, or weighted results determined according to the information in Table 2. The Fujisawa test set is composed of 10 samples of each of 46 digit strings. A count of 289 real touching digit samples was conducted by Fujisawa *et al* to give some idea of the likelihood of encountering a given pair.

9.5 Time & space

The SPARC station 10[1] CPU time to read each image in the USPS test set and produce internally (i.e., with no physical output) all of the 191 split images appearing in Appendix B was clocked at 0.8 sec. This gives roughly a rate of 240 digit strings per second.[2] As for space, the above processing was done in 2.0 megabytes of memory. A comment is in order about this large value: 1.5 megabytes is dynamically allocated to store information about the given input image, and the other 500 kbytes are for code. A significant portion of the code consists of functions which are loaded as part of a module, but are not used in this particular application. There are doubtless many avoidable inefficiencies in the space allocated for data, e.g., no attempt was made to economize where possible by storing data at the sub-byte level. The primary goal in this implementation was to produce a fast working system.

[1]SPARC station 10 is a registered trade mark of Sun Microsystems Inc

[2]The elapsed time is greater, of course, due to reading the 191 image files over a network, and sharing of the CPU and memory with other processes. It was measured at 1.5 minutes.

digit string	frequency /289	1	2	3	4	5	digit string	frequency /289	1	2	3	4	5
00	5	10	10	10	10	10	48	14	1	4	6	9	9
03	2	8	10	10	10	10	50	24	3	8	9	10	10
04	1	2	2	6	9	9	53	3	0	3	6	9	9
05	1	10	10	10	10	10	54	30	3	5	8	9	10
07	3	1	2	9	10	10	55	4	1	9	10	10	10
09	5	8	10	10	10	10	56	25	3	10	10	10	10
20	24	5	9	10	10	10	57	24	2	7	9	10	10
21	2	6	10	10	10	10	58	4	4	9	10	10	10
22	4	3	4	8	9	10	59	6	2	8	10	10	10
23	6	5	6	10	10	10	61	1	9	9	10	10	10
24	2	1	5	6	8	8	64	2	1	2	5	9	10
25	8	3	10	10	10	10	76	1	7	8	9	10	10
26	6	4	10	10	10	10	78	1	10	10	10	10	10
27	1	4	6	6	10	10	80	6	10	10	10	10	10
28	3	6	9	10	10	10	82	2	10	10	10	10	10
33	1	3	8	9	9	9	83	2	10	10	10	10	10
34	1	3	4	10	10	10	84	14	10	10	10	10	10
40	10	2	6	10	10	10	85	1	10	10	10	10	10
42	1	1	1	8	9	10	86	3	10	10	10	10	10
44	2	0	4	7	7	8	87	4	10	10	10	10	10
45	14	3	7	10	10	10	88	5	10	10	10	10	10
46	2	2	5	9	10	10	89	4	10	10	10	10	10
47	3	1	4	8	10	10	93	2	10	10	10	10	10

Table 2: The number of occurrences of each digit string in 289 samples and the number of strings correctly separated after 1, 2, ..., 5 cut attempts on 10 samples of each string.

Chapter 10

Conclusion

10.1 Summary of contributions

A method for separating the leftmost digit from a string of connected handwritten digits has been described and experimental results presented. So far as is known, this is the first trainable digit segmenter. The performance of the method has been shown to be better than two other published methods. As well, an efficient analytical method to extract contours has been presented.

10.2 Future research

10.2.1 Strengths & weaknesses of the method

This thesis has focussed on just one of the many component parts of a generalised digit segmentation system, which in turn would be one component part of a complete recognition system. The magnitude of the analysis involved in construction of an intelligent reliable document processing system for an application such as cheque processing is somewhat daunting when one considers the amount of activity involved in just the component studied here.

Some of the strengths of the method are:

1. As pointed out by Suen *et al* in [37], automatic learning capability has proven to

be critically important as complexity builds. The automatic learning capability of the present method is perhaps its strongest feature.

2. The features chosen are very robust, in particular, corner points, mountains, and valleys are very good indicators for segmentation of connected numerals.

3. The number of tunable parameters (see Appendix A) has been kept relatively small for a fairly complex system.

4. The features and methods used are of such a robust nature that many of the same techniques can readily be applied to segmentation and recognition of cursive script in general.

The principal weakness of the method is that during the extraction of SCPs it can sometimes happen that no viable pair of SCPs in the outer contour is detected. If no such pair is included in the set of candidates, it does not matter how well the set is sorted, a correct cut will never be found. There are a number of cases where this can happen, a few of which are listed below.

1. The most common one is that the corner detector can fail to find a corner it should be able to detect.

2. Thresholds do not work in all situations.

3. The point of connection of two characters may be so situated that even if the present feature detectors perform perfectly they will not detect it.

It is feasible to improve corner detection. It is not feasible to improve thresholds other than by finding ways to eliminate the need for them: one cannot endlessly tune them. The thresholds in this system are used mainly to limit the number of SCPs detected. The scheme proposed below might allow the elimination of some thresholds. As for the third item above, there is little that the present method can offer. Fortunately, such cases are rare in real world data, at least in the test data used here. One avenue left unexplored by the present method is the detection and exploitation of features of inner contours. If this were done it could prove useful for handling an intractable outer contour. It will also be necessary to analyse inner contours in order to do robust handling of complexly connected digits.

10.2.2 Schemes to improve performance

One avenue to explore is the effect of negative weight values on the performance. It was assumed that all features measured were non-negative indicators for segmentation, but this assumption may be false.

The present version of the segmenter was trained on general, unclassified types of touching digits. It is likely that performance can be improved by training a number of copies of the present segmenter on selective kinds of touching data such as the 46 classes of touching characters shown in Table 2. This would result in a number of digit splitters. Each splitter will be expert at handling its own class of touching digits. When all segmenters are run in parallel it will then be highly likely that one or more of them will give a correct split at its first or second attempt. If each segmenter were integrated with its own dedicated recogniser then a correct cut could quickly be recognised.

References

[1] P. Ahmed and C.Y. Suen. Segmentation of unconstrained handwritten postal zip codes. In *Proc. Int. Conf. on Pattern Recognition*, pages 545–547, 1984.

[2] Pervez Ahmed. *Computer Recognition of Totally Unconstrained Handwritten ZIP Codes*. PhD thesis, Concordia University, Montreal, Canada, 1986.

[3] C. Arcelli, A. Held, and K. Abe. A coarse to fine corner-finding method. In *Proc. IAPR Workshop on Machine Vision Applications*, pages 427–430, 1990.

[4] A. Badreldin. *Structural Recognition of Handwritten Numeral Strings*. PhD thesis, University of Windsor, Windsor, Canada, 1985.

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

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

[7] L.T. Chen, L.S. Davis, and C.P. Kruskal. Efficient parallel processing of image contours. *IEEE Trans. Patt. An. and Mach. Intel.*, 15(1):69–81, Jan 1993.

[8] M. Cheriet, Y.S. Huang, and C.Y. Suen. Background region-based algorithm for the segmentation of connected digits In *Proc. 11th International Conference on Pattern Recognition*, pages v2 619–622, 1992.

[9] Christopher E. Dunn and P.S.P. Wang. Character segmentation techniques for handwritten text – a survey. In *Proc. 11th International Conference on Pattern Recognition*, pages 577–580, The Hague, The Netherlands, August 1992.

[10] M.J. Eccles, M.P.C. McQueen, and D. Rosen. Analysis of the digitized boundaries of planar objects. *Pattern Recognition*, 9:31–41, 1977.

[11] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of curves in the plane—topology, combinatorics, and algorithms. *Theoretical Computer Science 92*, pages 319–336, 1992.

[12] Willis D. Ellis. *A Source Book of Gestalt Psychology.* Humanities Press Co., New York, 1967.

[13] R. Fenrich. Segmentation of automatically located handwritten words. In *Int. Workshop on Frontiers in Handwriting Recognition (IWFHR-2)*, pages 33–44, 1991.

[14] R. Fenrich and S. Krishnamoorthy. Segmenting diverse quality handwritten digit strings in near real-time. In *Fourth USPS Advanced Technology Conference*, pages 523–537, 1990.

[15] James D. Foley and Andries Van Dam. *Fundamentals of Interactive Computer Graphics.* Systems Programming Series. Addison-Wesley, Reading Mass., U.S.A., 1982.

[16] H. Freeman. *Boundary Encoding and Processing. In Picture Processing and Psychopictorics, B.S. Lipkin and A. Rosenfeld eds.*, pages 241–266. Academic, New York, 1970.

[17] H. Fujisawa, N. Yasuaki, and K. Kiyomichi. Segmentation methods for character recognition: From segmentation to document structure analysis. *Proc. IEEE*, 80(7):1079–1092, 1992.

[18] G. Gallus and P.W. Neurath. Improved computer chromosome analysis incorporating preprocessing and boundary analysis. *Phys. Med. Biol.*, 15(3):435–445, 1970.

[19] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, Mass., U.S.A., 1989.

[20] Venu Govindaraju and Sargur N. Srihari. Separating handwritten text from overlapping non-textual contours. In *Int. Workshop on Frontiers in Handwriting Rec.*, pages 111–119, Chateau de Bonas, France, Sep 1991.

[21] Andreas Held. Description of binary shapes in two dimensions. Master's thesis, Shizuoka University, Japan, 1992.

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

[23] David Katz. *Gestalt Psychology.* Ronald Press Co., New York, 1950.

[24] F. Kimura and M. Shridhar. Recognition of connected numerals. In *Proc. of 1st International Conference on Document Analysis and Recognition*, pages 731–739, 1991.

[25] F. Kimura, M. Shridhar, and N. Narasimhamurthi. Lexicon directed segmentation-recognition procedure for unconstrained handwritten words. In *Proc. of 3rd International Workshop on Frontiers in Handwriting Recognition*, pages 122–131, Buffalo, U.S.A., May 1993.

[26] E. Lecolinet. *Segmentation d'Images de Mots Manuscrits.* PhD thesis, L'université Pierre et Marie Curie, Paris VI, 1990.

[27] Hong-Chih Liu and M.D. Srinath. Corner detection from chain-code. *Pattern Recognition*, 23(1/2):51–68, 1990.

[28] H. Nishida and S. Mori. A model-based split-and-merge method for recognition and segmentation of character strings. In *Advances in Structural and Syntactic Pattern Recognition, Proc. Int. Workshop on Struc. and Synt. Patt. Rec., H. Bunke, Ed.*, pages 300–309, New Jersey, U.S.A., August 1992. World Scientific.

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

[30] A. Rosenfeld. Connectivity in digital pictures. *Journal of the ACM*, 17(1):146-160, Jan 1970.

[31] M. Shridhar and A. Badreldin. Recognition of isolated and simply connected handwritten numerals. *Pattern Recognition*, 10(1):1-12, 1986.

[32] Peter L. Sparks, M.V. Nagendraprasad, and Amar Gupta. An algorithm for segmenting handwritten numeral strings. Technical report, International Financial Services Center, Sloan School of Management, MIT, Massachusetts, U.S.A., 1992.

[33] N.W. Strathy. Segmentation of handwritten digits: A review of selected methods and proposal of a new method. Final report for seminar course in man-machine communication, April 1992.

[34] N.W. Strathy. Segmentation of touching numerals using contour features. In *Student Papers of the 3rd Annual IRIS-PRECARN Conference*, page 58, Ottawa, Canada, June 1993.

[35] N.W. Strathy, C.Y. Suen, and A. Krzyzak. Segmentation of connected digits using contour regions and corner points. In *Proc. of the Second International Conference on Signal Processing*, Beijing, China, October 1993. In press.

[36] N.W. Strathy, C.Y. Suen, and A. Krzyzak. Segmentation of handwritten digits using contour features. In *Proc. of the Second International Conference on Document Analysis and Recognition*, Ibaraki, Japan, October 1993. In press.

[37] C.Y. Suen, R. Legault, C. Nadal, and L. Lam. Building a new generation of handwriting recognition systems. *Patt. Rec. Letters*, 14:303-315, Apr 1993.

[38] C. Teh and R.T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Machine Intell.*, II(8):859-872, August 1989.

[39] G.A. Vignaux and S. Michalewicz. A genetic algorithm for the linear transportation problem. *IEEE Trans. on Sys., Man, and Cybernetics*, 21(2):445-452, Mar/Apr 1991.

[40] Deborah Walters. Selection of image primitives for general-purpose visual processing. *Comp. Vis., Gr., and Image Processing*, 37:262–298, 1987.

[41] S. Yokoi, J. Toriwaki, and T. Fukumura. Topological properties in digitized binary pictures. *Systems Computers Controls*, 4(6):32–39, 1973.

Appendix A

Parameters

This appendix consists of a complete list of the tunable parameters used in the method. The intention is to give an explicit idea of the number and nature of all arbitrary constants affecting the performance.

	§	PHASE	DESCRIPTION	VALUE
1	2.3	size doubling	images below this height in pixels are doubled	70
2	4.2	corner detection	maximum number of chain nodes between corners in the same cluster (t_c)	7
3	4.2	corner detection	corner cluster thinning threshold ($2t_c$)	14
4	4.2	"	initial value of N	$\lceil nodes\_in\_chain/64 \rceil$
5	4.2	"	value of N for detection between known level 0 corners	$\max(\lceil nodes\_in\_seg/4 \rceil, 3)$
6	4.2	"	value of N for detection between known corners at level > 0	$\max(\lceil nodes\_in\_seg/8 \rceil, 3)$
7	4.2	"	maximum level	2
8	4.2	"	min |N-code| for corner at level 0	$\max(N, 2)$
9	4.2	"	min |N-code| for corner at level > 0	$\lfloor N(6 - level)/6 \rfloor$
10	6.2	identification of SCPs	min pixels between existing SCP and new candidate	10
11	6.2	"	number of pixels from corner point to second reference point for SCPs of type 3	6
12	6.3	ranking SCP pairs	value of N used to measure curvature at an SCP	9
13	6.3	"	min abs. concavity N-code value to permit nonzero credit for sharpness of corner	32

	§	PHASE	DESCRIPTION	VALUE
14	6.3	ranking SCP pairs	max ratio of pixels on cutting path to pixels on outer contour of component resulting from split	0.25
15	6.3	"	max ratio of pixels on outer contour to pixels on cutting path of component resulting from split (only for 'long' cuts defined by next threshold)	0.25
16	6.3	"	min pixels in a 'long' cut	15
17	6.3	"	min pixels in outer contour of component resulting from a split	100
18	6.3	"	weight 1	7
19	6.3	"	weight 2	197
20	6.3	"	weight 3	321
21	6.3	"	weight 4	6
22	6.3	"	weight 5	2
23	6.3	"	weight 6	382
24	6.3	"	weight 7	12
25	6.3	"	weight 8	253
26	6.3	"	weight 9	12
27	8.2	feature extraction	normalized range of feature values (approx.)	(0, 400)
28	8.2	"	range of weight values	(0, 511)

Appendix B

Sample Results

This appendix contains the results for the USPS test set (see section 9.3). It is included to give some idea of the degree of complexity of data to be encountered in a real application, to show explicitly the performance of the system, and to give examples of the standards of judgment used to identify successful splits. Each touching digit string was extracted from the image of a ZIP code using a semi-automatic process. The names of the files in the test set are given below for reference. The first 6 characters of each name comprise the name of the original USPS file from which the digit string was extracted, and the 2 remaining digits identify a connected component within the original image.

The files in the USPS test set

bb000801.tif	bd096901.tif	bd230800.tif	bl085801.tif	bs016500.tif	bs035300.tif
bb003000.tif	bd098100.tif	bd231300.tif	bl087401.tif	bs017701.tif	bs035301.tif
bb003001.tif	bd100902.tif	bd233403.tif	bl091601.tif	bs017702.tif	bs035600.tif
bb006201.tif	bd101202.tif	bd244400.tif	bl092307.tif	bs017703.tif	bs036400.tif
bb011101.tif	bd103102.tif	bd246801.tif	bs000500.tif	bs017801.tif	bs037802.tif
bb011102.tif	bd105603.tif	bd249700.tif	bs000900.tif	bs017901.tif	bs038001.tif
bb012101.tif	bd105604.tif	bd256101.tif	bs001301.tif	bs018300.tif	bs038200.tif
bb014802.tif	bd111100.tif	bd256900.tif	bs003002 tif	bs019701.tif	bs038202.tif
bb017102.tif	bd112300.tif	bd258701.tif	bs003102.tif	bs020202.tif	bs038601.tif
bb026300.tif	bd113900.tif	bd258900.tif	bs003900 tif	bs020401.tif	bs039000.tif
bb026301.tif	bd114600.tif	bd258901.tif	bs004101.tif	bs020500.tif	bs039300.tif
bb028700.tif	bd117701.tif	bl000402.tif	bs005501.tif	bs020501.tif	bs039301.tif
bb029500.tif	bd130800.tif	bl003402.tif	bs005702.tif	bs020601.tif	bs040100.tif
bc016700.tif	bd130801.tif	bl013001.tif	bs005802.tif	bs020603.tif	bs040901 tif
bd000103.tif	bd137903.tif	bl020200.tif	bs006003.tif	bs021501.tif	bs04[illegible]003 tif
bd002001.tif	bd141000.tif	bl028601 tif	bs007002.tif	bs021602.tif	bs042502.tif
bd002002.tif	bd141002.tif	bl032603.tif	bs007702.tif	bs021802.tif	bs042600.tif
bd015100.tif	bd144303.tif	bl035900.tif	bs008001.tif	bs022302.tif	bs043603.tif
bd015203.tif	bd146902.tif	bl042002.tif	bs008704.tif	bs022602.tif	bs044902 tif
bd029503.tif	bd152601.tif	bl046201.tif	bs009202.tif	bs023101.tif	bs045000.tif
bd036400.tif	bd154001.tif	bl054000.tif	bs009601.tif	bs024500.tif	bs046000.tif
bd045200.tif	bd166707.tif	bl056001.tif	bs010102.tif	bs026103.tif	bs046704 tif
bd046101.tif	bd168000.tif	bl059100.tif	bs011003.tif	bs027002.tif	bs046801.tif
bd049402.tif	bd168002.tif	bl059102.tif	bs011401.tif	bs027801.tif	bs047402.tif
bd055800.tif	bd173600.tif	bl059502.tif	bs011803.tif	bs028700.tif	bs048001.tif
bd058300.tif	bd175501.tif	bl063702 tif	bs011902.tif	bs028703 tif	bs048300.tif
bd070002.tif	bd176601.tif	bl064200.tif	bs012002.tif	bs028705.tif	bs048401.tif
bd077201.tif	bd178900.tif	bl065701.tif	bs013300.tif	bs029602.tif	bs048702 tif
bd077801.tif	bd199100.tif	bl076302.tif	bs014801.tif	bs030205.tif	bs049006.tif
bd082700.tif	bd202902.tif	bl078500.tif	bs015301.tif	bs032802.tif	bs049401.tif
bd084101.tif	bd223701.tif	bl084003.tif	bs015503 tif	bs033805.tif	bs049503.tif
bd088001.tif	bd227505.tif	bl085400.tif	bs015802.tif	bs035001.tif	

In each row of what follows, the input string appears at the left and is followed by the 5 top-scoring attempts to separate the leftmost digit. A mark appears below the first attempt which was judged to be successful (if any).