On-Line Recognition of Connected Handwriting

by

David Malcolm Ford BSc

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy, May 1991

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Colin Higgins, for all his suggestions and encouragement during the course of this research and during the preparation of this thesis. He has been a good friend and colleague throughout this PhD work.

I also wish to thank the National Physical Laboratory who funded the early part of this research, especially Hilary Symm, Paul Kenward and Ed Brocklehurst who provided the preprocessing and feature extraction routines for this work. The Central Computer and Telecommunications Agency of HM Treasury are also acknowledged for their input to this project. Many thanks also to the *Paper Interface* research team at Nottingham Polytechnic for the use of their dictionary with which this system has been tested.

To my friends and colleagues in the Department of Computer Science, I wish to extend my thanks for the countless suggestions, advice and assistance I have received over the years.

Finally, I wish to express my thanks to my parents for their continual encouragement, and to my wife, Fiona, without whose endless support, patience and understanding this research would not have been possible.

Table of Contents

Chapter 3 - ORCHiD System Overview

Chapter 4 - Segmentation

Chapter 5 - Template Matching

Chapter 6 - Candidate Word Verification

Chapter 7 - Training

Chapter 8 - Results

Chapter 9 - Conclusions and Further Work

List of Tables and Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 8

Appendix A

Appendix B

Abstract

Computer technology has rapidly improved over the last few years, with more powerful machines becoming ever smaller and cheaper. The latest growth area is in portable personal computers, providing powerful facilities to the mobile business person. Alongside this development has been the vast improvement to the human computer interface, allowing non-computer-literate users access to computing facilities. These two aspects are now being combined into a portable computer that can be operated with a stylus, without the need for a keyboard. Handwriting is the obvious method for entering data and cursive script recognition research aims to comprehend unconstrained, natural handwriting.

The ORCHiD system described in this thesis recognises connected handwriting collected on-line, in real time, via a digitising pad. After preprocessing, to remove any hardware-related errors, and normalising, the script is segmented and features of each segment measured. A new segmentation method has been developed which appears to be very consistent across a large number of handwriting styles.

A statistical template matching algorithm is used to identify the segments. The system allows ambiguous matching, since cursive script is an ambiguous communications medium when taken out of context, and a probability for each match is calculated. These probabilities can be combined across the word to produce a ranked list of possible interpretations of the script word.

A fast dictionary lookup routine has been developed enabling the sometimes very large list of possible words to be verified.

The ORCHiD system can be trained, if desired, to a particular user. The training routine, however, is automatic since the untrained recognition system is used as the basis for the trained system. There is therefore very little start-up time before the system can be used. A decision-directed training approach is used.

Recognition rates for the system vary depending on the consistency of the writing. On average, the untrained system achieved 75% recognition. After some training, average recognition rates of 91% were achieved, with up to 96% observed after further training.

To Fiona

Chapter 1

Introduction

1.1. Objectives

The aim of this research was to produce an on-line handwriting recognition system that could be used by any person with reasonably neat script. The constituent characters of the script could be connected and relatively unconstrained in formation and structure. The system would be automatically trained to an individual's own writing style, if desired, so that the recognition rate would increase with system usage.

To satisfy this aim, a system was developed called ORCHiD or On-line Recognition of Connected Handwriting Demonstrator. This research was part of a larger project, organised by the National Physical Laboratory (NPL), investigating a novel form of computer interface, termed *Electronic Paper* (EP) (see section 1.2.2). The Electronic Paper Project included research into suitable hardware, ergonomics and applications software, as well as more basic research into handwriting recognition.

Research into Cursive Script Recognition (CSR) was carried out at two sites, NPL and The University of Nottingham. NPL designed and developed the data collection and low-level processing routines. Nottingham designed the recognition routines, some of which were implemented at NPL, the rest at Nottingham. Nottingham designed and implemented the post-processing routines. The exact breakdown of the work is described in section 3.2.

The input script is collected *on-line* (in real-time, as it is being written), via a *digitising tablet*, and stored initially as a time-ordered list of coordinates. Since the low-level processing routines were not developed at Nottingham, the research described in this thesis was based on the assumption that the data would be

preprocessed and *segmented* into individual words and the *baseline* of the writing detected before being submitted to the recognition system. (See sections 1.3 and 1.4 for an explanation of italicised terms.)

ORCHiD recognises words, written with lower case letters, by comparing the input script to a database of ideal character formations. The method used for this comparison maps different writing styles to the perfect *copy-book* styles taught in schools. The ideal character formations, or *allographs*, can then be based on these copy-book styles.

Handwriting is an ambiguous communications medium since the interpretation of the ink marks on the page is often dependent on the context of the surrounding writing. For this reason, the ORCHiD system contains an ambiguous recogniser which provides a weighting that allows its output to be ranked. It also assumes that every sample word is contained in the system dictionary so that the output will always be a valid word, rather than a random sequence of letters.

In order to improve recognition accuracy, ORCHiD can be trained to a particular writer, if desired. Apart from a brief initialisation routine, this training can be carried out fully automatically, so that recognition improves as the system is used.

The rest of this chapter includes the background and motivation for research into CSR. A description of the technical terminology is then followed by a brief introduction to the principle stages that may be present in any CSR system. Specific details of the ORCHiD system are reserved for the later chapters of this thesis.

1.2. Background and Motivation

1.2.1. The Development of the Human-Computer Interface

As computer power and size has increased over the years, the interface between the computer operator and the machine has steadily improved[1]. Originally the computer program and data were either hard-wired into the machine at the time of manufacture or entered, literally, 'bit by bit' using binary switches or hand-keys. The computer programmer was typically a dedicated scientist who interacted with the computer at the very low level of its own machine code.

More practical computer installations, allowing the input and execution of more complex programs that could access large amounts of data, became

available with the development of punched card and paper tape readers. Computer programming was a very labour intensive process, involving preparation of the punched cards or paper tape, submission of the program to a computer operator who fed the cards or tape into the machine, and waiting for the program to run before collecting the output from a line-printer. As high-level language compilers were developed the computer programmer did not need such a detailed understanding of the inner workings of the computer, and so a new generation of non-technical computer users emerged. The computer itself, though, remained a mysterious entity, locked in its own air-conditioned environment, allowing communication only within a very restricted protocol via the operators.

The teletype terminal and the corresponding software and hardware developments, which provided for the interactive editing, compiling and running of programs stored on magnetic media, greatly improved the perceived usability of computers. The appearance of the Visual Display Unit computer terminal allowed for limited real-time graphical output on its character-based screen, and greatly increased programmer performance. Bit-mapped graphics displays became available with the option of pointing to objects on the screen with a light pen.

The computer was now accessible to anybody who could master a keyboard (provided they had sufficient funds). They could edit, compile and run their programs whenever they could gain access to a terminal and, as Personal Computers (PCs) were developed and became cheaper, they could have the computer sitting on their own desk. Traditional office practices were revolutionised, as data storage and retrieval and document preparation were increasingly carried out by the originator of the request rather than by specialist secretaries. Keyboard skills now became especially important, and not surprisingly secretaries found they could adapt to the new machine that was placed on their desks. However a large proportion of the population cannot type and many do not wish to learn, perhaps because of a fear of technology or because they think that typing is demeaning. It is not yet even standard practice in the UK to teach all schoolchildren how to type. Many people therefore remained outside of this technological revolution.

This remained the state of the art for a long time, until the appearance of powerful workstations with high-resolution bit-mapped screens and the Window, Icon, Mouse and Pointer (WIMP) interface, first successfully exploited in the Apple Macintosh[43]. This provided a revolution in access to computers for the

inexperienced keyboard user, as a whole host of applications could be run simply by manoeuvring the screen cursor with the mouse and by selecting items from menus or icons, with keyboard use restricted to entering text and numerical data.

1.2.2. The Electronic Paper Interface

A new computer interface is now appearing that will further increase computer accessibility for the lay person and improve access for the skilled user. Everyone learns to manipulate a pen and paper at an early age, whether it be to write with or just to sketch, and this new interface takes advantage of this skill. The computer display is replaced by a flat surfaced screen that can be written on with a special stylus (figure 1.1). The slightest movement of the stylus can be tracked by the computer, and when it is in contact with the screen, *electronic ink* is left on the display. This novel form of computer interface which will be referred to in this thesis as *Electronic Paper* has also been named *Active Book* or *Interactive Tablet*[8,11,27,40,64]. The concept of EP includes not only the hardware device itself, but also the controlling software. Intelligent software interprets any gestures that are made and acts accordingly. Often a book or personal organiser paradigm is used as the basis of the operating system, with chapters, table of contents, index, thumb guides, etc. This further enhances the intuitive nature of such a device.

This new device incorporates all the advantages of the mouse, light pen and touch screen, but reduces the hand/eye coordination needed to operate a mouse, increases the accuracy available for a light pen or touch screen, and also provides an intelligent, natural, software interface.

There are several obvious uses for EP, particularly in an office environment[40].

- The interface can be used as a jotting pad for notes and sketches. These can then be tidied up, sent electronically to another site for corrections or annotations, and returned for printing or inclusion in other documents. Faxes could be prepared, sent and received on-line, without the need for paper copies.
- It can be used for the interactive manipulation and annotation of digitally encoded documents and images.
- The executive working outside normal office hours can produce documents that would normally be sent to a secretary.

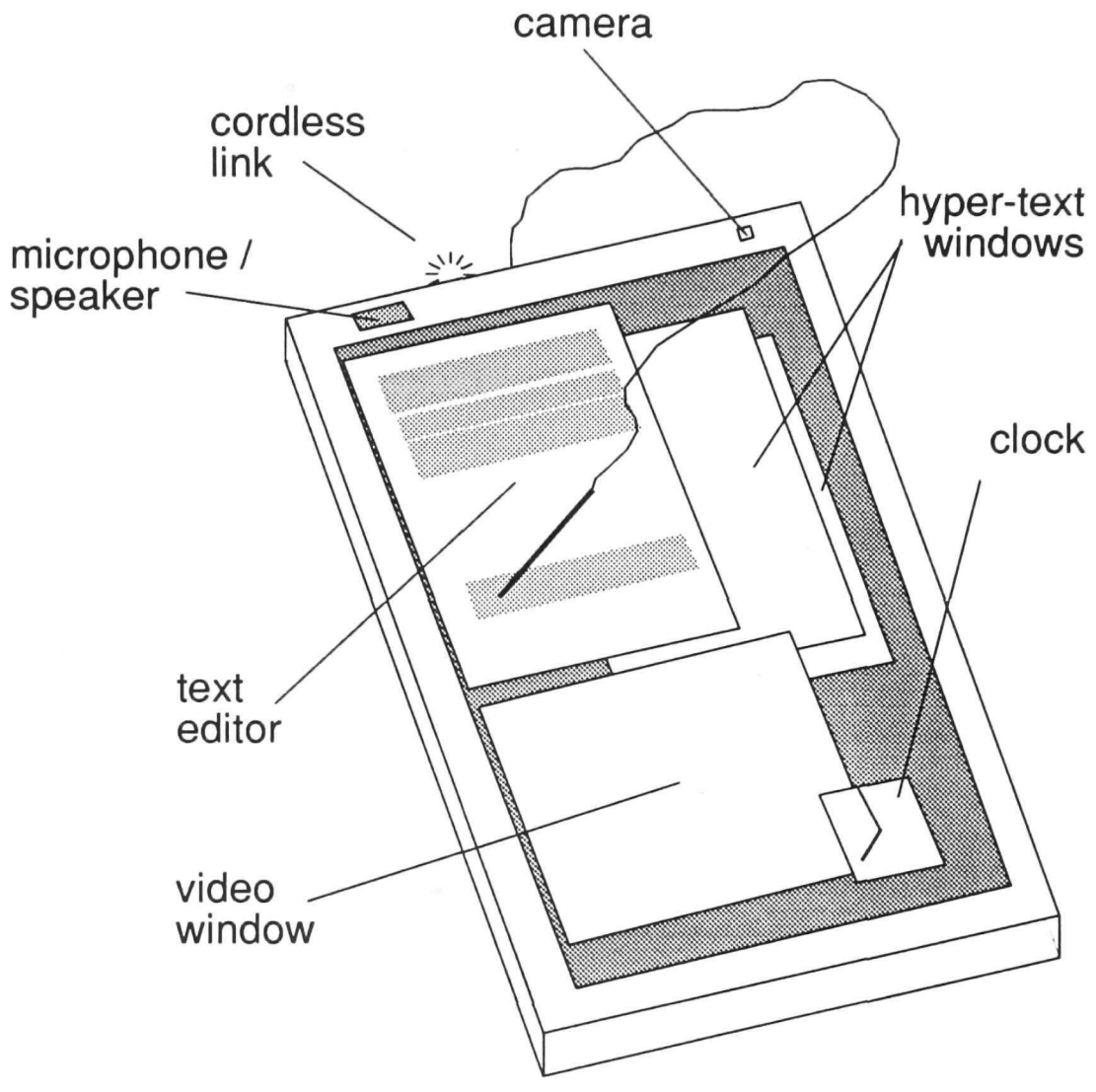


Figure 1.1 - Electronic Paper

- Two devices connected to each other via a communications channel could be used for remote, on-line, visual conferences, enabling distant parties to converse and simultaneously illustrate their ideas with free-hand sketches.
- This device would be ideal to take advantage of the imminent arrival of multimedia computing, allowing for video, text and sound to be combined in the same unit. All of these facilities could then be accessed with natural pen strokes.

One activity that may especially take advantage of this interface is document preparation. At present, a large amount of the initial preparation of documents is done with a pen and paper, drafting out text and sketches that will later be typed

up, possibly on some form of desk-top publishing system. An EP system could be used for the initial drafting[9], where the text would be automatically converted from handwriting to typescript and the rough sketches to neat diagrams, without the need for an intermediate conversion, either by the author or by a secretary. Subsequent editing of the document could then be carried out directly on the screen, with the changes immediately visible[18,87,99]. Figure 1.2 shows an original hand-drafted document and its typeset equivalent. The automatic conversion would be carried out step by step as each section of the document is drafted.

The editing stage would be even more efficient than using a standard word-processing package, since the editing commands could be the same as those used to mark up a paper manuscript[35]. For example, the handwritten instruction to move a paragraph merely requires the gesture of drawing an arrow from within the paragraph to its destination. With a conventional word processor it is necessary to specify the beginning of the paragraph, the end of the paragraph, select the move command from a range of options, and specify the destination - a large amount of data must be supplied for what should be an obvious action.

Electronic Paper offers the potential for people to interact with a machine without the need for training or constant recourse to voluminous instruction manuals. A few pages will suffice as the manual for EP, with pictures of the few symbols necessary to operate the device. Varying levels of help could be brought onto the screen to assist if necessary. Figure 1.3 shows the manual that would be necessary to operate a text editor[87].

1.2.3. The Need for Handwriting Recognition

In order to take advantage of this new type of interface a means of entering data into the computer is needed without using a conventional keyboard. Obviously it is possible to display a *soft* keyboard on the screen and pick off characters by pointing to them, but ideally some form of handwriting recognition is required. Speech recognition may be available in the future for automatic conversion of dictation to typescript, but typically we speak in a less structured manner than we write, and speech is not the best communications medium for, say, drawing a diagram.

In countries where the alphabet consists of a very large number of characters or ideograms, such as China and Japan, some form of handwriting recognition is especially desirable since a keyboard is, of necessity, very large and impractical.

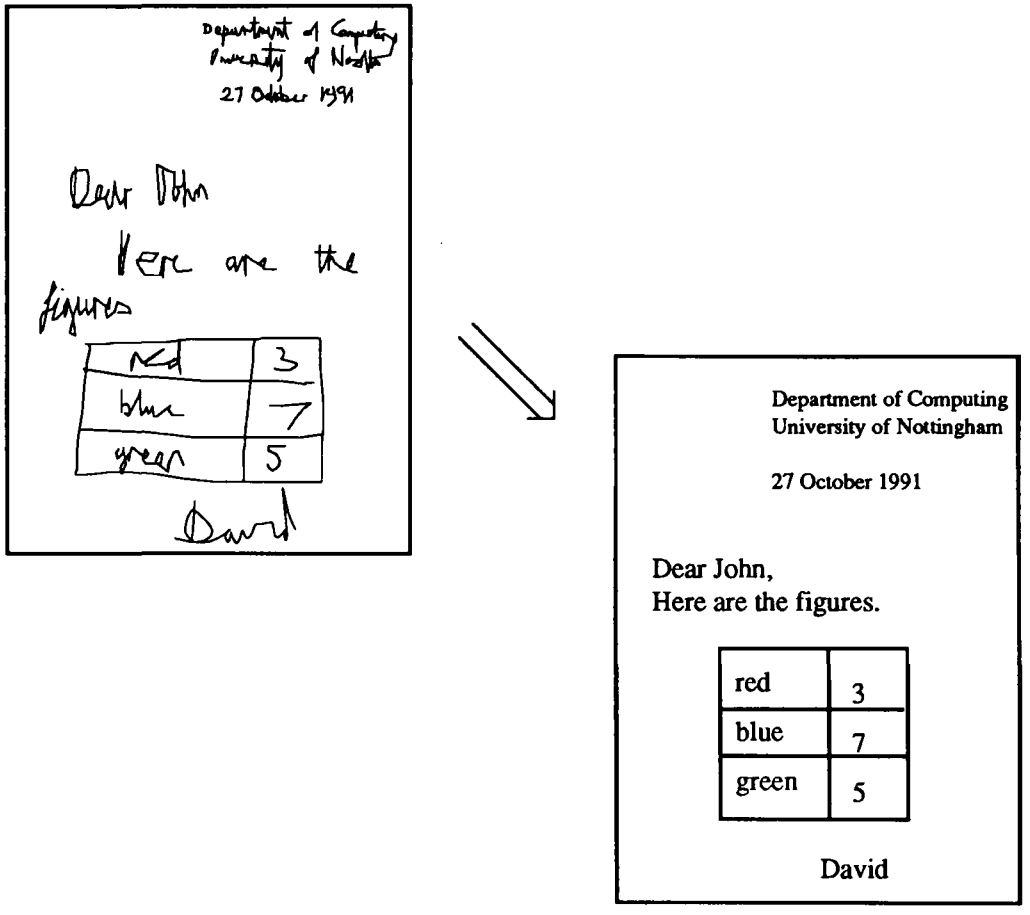


Figure 1.2 - A freehand document and its typeset equivalent

This problem is hindering such countries from fully exploiting the technical revolution since there is no easy method of entering data into a computer in their natural language. These languages are not within the scope of this thesis, and so we will restrict ourselves to languages based on the Roman alphabet.

1.3. Terminology

It is necessary at this stage to define some of the terms used in this thesis to distinguish this work from similarly titled work.

On-line Recognition

On-line handwriting recognition means that the data is captured as the user writes, usually on some form of digitising tablet or surface. This has also been referred to as *dynamic* or *real-time* recognition. The digitiser encodes the script into a time-ordered list of coordinates. Information is available on whether the

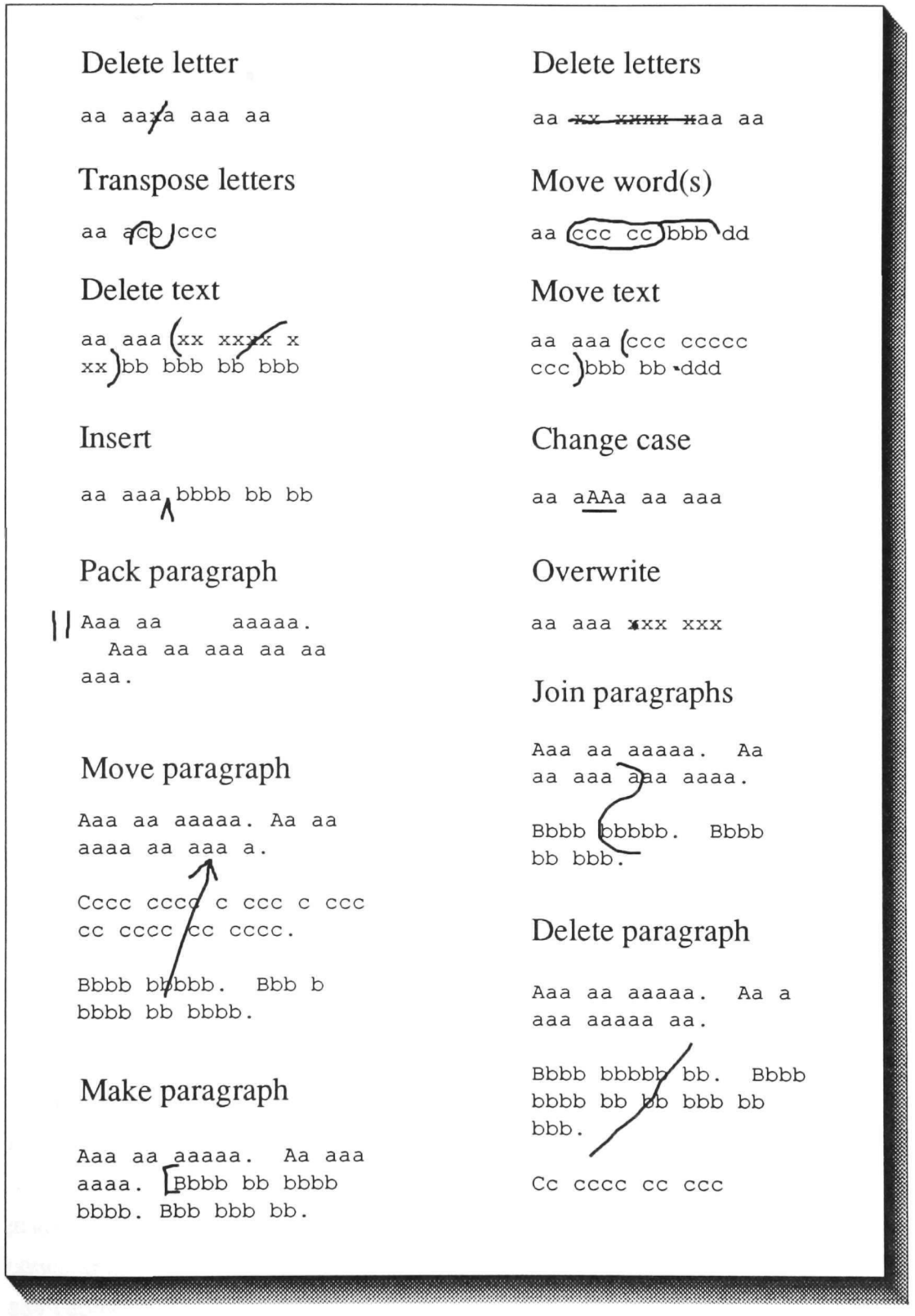


Figure 1.3 - A text editor manual

pen is in contact with the writing surface (*pen-down*) or not (*pen-up*) and some digitisers will provide the position of the pen when it is not in contact with the digitiser. *Off-line*, or *static*, recognition uses data supplied after the writing process is complete, usually in the form of an image or bit-map from a scanner or similar device. Static recognition is not within the scope of this thesis.

The segment of writing between a pen-down and a pen-up (or between *pen-lifts*) is defined to be a *stroke*. A written word is therefore made up of one or several strokes (see figure 1.4). Some research groups define a stroke differently to be the pen trace between segmentation points.

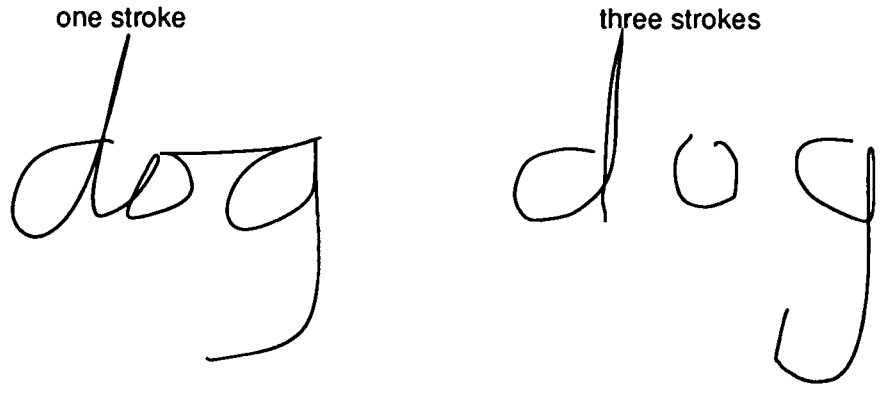


Figure 1.4 - One stroke and many stroke words

Connected Handwriting

There are two different types of handwriting which can be considered within Roman-based languages - *separated* characters (hand-printing) and *cursive* handwriting.

Connected handwriting, or *cursive script*, is writing with no constraints on the separation of the individual characters which make up a word, as opposed to *separated* characters, where there is some form of gap between each individual letter, whether temporal, in the form of a pause with the pen off the paper, or graphical, with a definite blank space on the page. The work described here has concentrated on words formed from lower case characters only, and assumes that any capital letters will be separated from the rest of the word before submission to the system. The term *cursive* has also been used with respect to single character recognition systems to indicate that a character is not necessarily constructed of separate strokes (as in block capitals) - see figure 1.5.

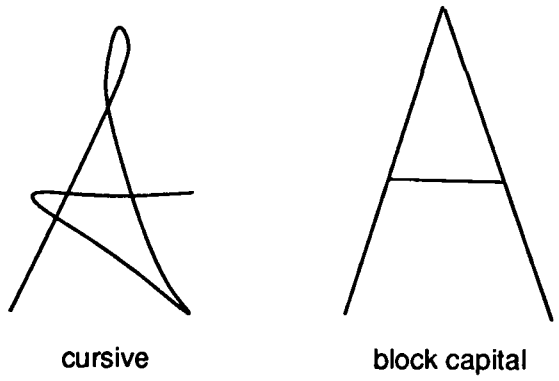


Figure 1.5 - A cursive and block capital A

Separated character recognition systems are now commercially available in several different forms. Although the constraint of separating each character is perhaps not unreasonable for a limited amount of textual entry, for example making brief notes or labelling a diagram, most people find this style of handwriting unnatural and will typically start to run letters together or overlap them. A separated character system will fail under these circumstances, so the user must adapt his/her own writing style to accommodate the machine.

Ideally, if computers are to be widely accepted and utilised by the general population, it should not be necessary to learn a new skill or to adapt a previously learnt skill in order to use the computer. Cursive script is an appropriate input medium as the writer can be relatively unconstrained in writing style, and can lift the pen or run the letters of a word together as desired.

Fixed Recognition or Trainable Recognition?

A perfect script recognition system should be able to recognise any handwriting presented to it with no intervention by the user. To develop such a system is not possible at present due to the complexity of the problem, so it is necessary for either the user to adapt to the system, or the system to the user, or both.

A *fixed* recognition system has the styles that it is able to recognise "hard-wired" into the recognition algorithm. Anybody wishing to use such a system must conform to the accepted style(s). This has the advantage that anyone who writes in the approved style can immediately use the system. However, the recognition accuracy can only be improved by the user adjusting to the machine.

For limited applications, such as recognition of separated block capitals, enough different styles can be programmed in to the system so that this is not a problem. (An example of such a system is the Pencept Penpad or GRiD Gridpad.)

A *trainable* recognition system is able to learn a particular user's own individual style. Anybody wishing to use such a system must first enrol by providing a sample of handwriting which is analysed. This is usually carried out by running an initialisation or training program. This type of system has the advantage that it is possible for virtually anybody to use the system with minimal adaptation on the part of the user. The success of such a system, though, is dependent on how well the training session has been carried out. It may be unacceptable, however, to insist that a new user must spend several hours training a system before it can be used. (An example of a trainable system is the Linus Writetop or Anotech.)

The research described in this thesis attempts to compromise these alternatives by providing a system which has fixed recognition built in, so that the user can immediately write, but which can be subsequently trained to that particular user during use.

1.4. The Cursive Script Recognition Process

Computer recognition of cursive script, in general terms, consists of several distinct stages, some of which are performed sequentially and some which may be performed in parallel (if the necessary processing environment is available). These stages can be loosely grouped together under the headings *preprocessing*, *recognition* and *postprocessing*. Tappert *et al*[93] provides a survey of on-line handwriting recognition with a good, brief introduction to the techniques involved.

Some of the major processes that might be used within an on-line CSR system are described, in general terms, below. More details and references are provided in Chapter 2. The ORCHiD system incorporates some of these processes, but not all. Further details specific to this system are detailed in subsequent chapters.

1.4.1. Preprocessing

Data Capture

For an on-line recognition system it is necessary to collect the data as it is being written so that the order of the strokes of the pen can be recorded as well as the position of the pen. The writing is collected using a special pen and writing surface called a *digitiser* or *digitising tablet*. This may provide a number of details about the motion of the pen. Typically, the information is stored as a time-ordered list of coordinates (or digitiser data points (DDPs)) with an indication of whether the pen is up or down. However, some digitisers also supply data about the angle at which the pen is being held, timing information that may be used to calculate dynamic information on the pen-tip travel, or vertical position of the pen above the digitising surface.

Cleaning

Deficiencies in the quality of data from digitising tablets can lead to peculiarities in the image data stored. Spurious points (or *noise*) can occur caused by hardware errors. The discrete quantisation due to the digitising process can cause near vertical or horizontal lines to appear as a succession of wobbles (see figure 1.6). Hooks can occur at pen-ups and pen-downs due to erratic upward movements of the pen, or they may be deliberately written as ornamentation or *serif*s. These, together with other unwanted data (eg noise) from the digitiser, can be removed in the cleaning process to produce a minimally distorted, digital image.

Word Segmentation

Modern western languages separate text into individual words delimited by white space. The word is therefore the natural unit of segmentation for handwriting recognition.

Straightening, Deskewing and Scaling

Across the population, there is a wide variety in the size and slope of everyday handwriting. When not forced to write on ruled paper many people will tend to deviate from the horizontal. For a recognition system to be as generalised as possible, and work successfully on a variety of handwriting styles, it is preferable

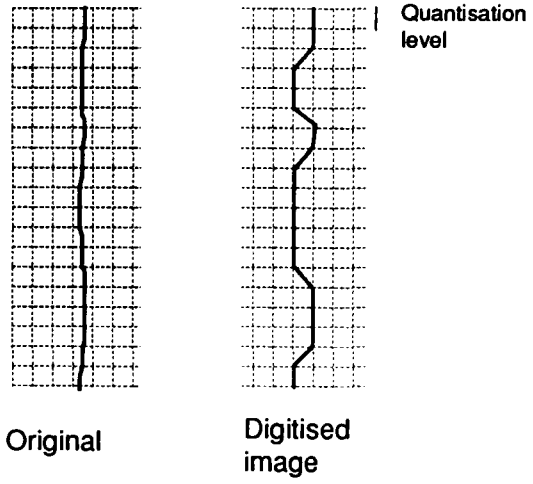


Figure 1.6 - Straight line wobble

for these consistent variations in the script to be standardised or *normalised.* The data can be automatically straightened, deskewed and scaled by the machine so that all samples are nominally similar in their gross features (see figure 1.7).

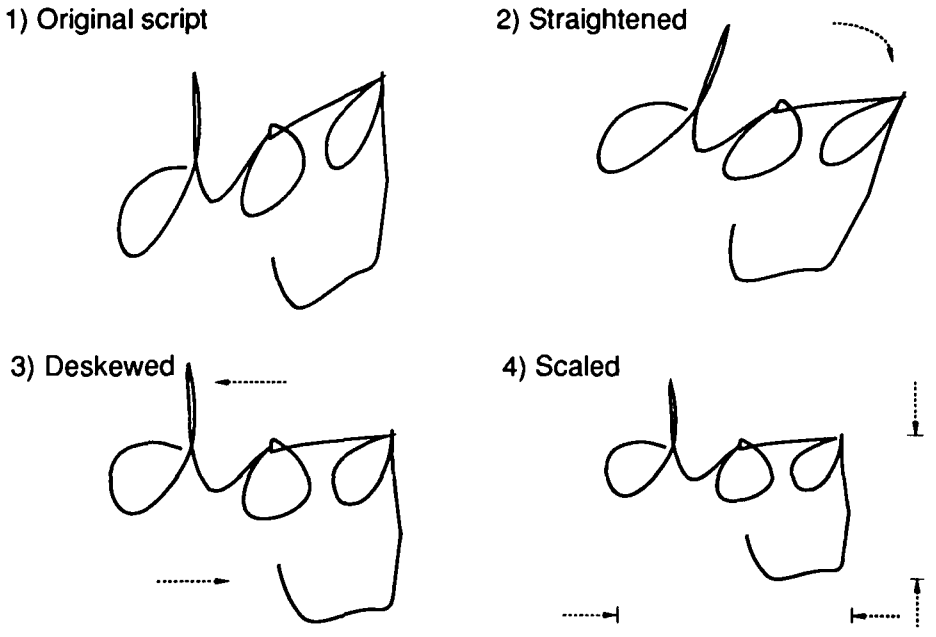


Figure 1.7 - Normalised script

Vertical Region Detection

Some recognition systems need to know the position of certain vertical regions within which the word was written. Typically of interest are the *baseline* on which the word was written; the position of the top of small characters (*a*, *e*, *c* etc), referred to in this thesis as the *halfline*; the line midway between these two lines is referred to as the *centre-line*; the position of the top of tall characters (*l*, *d*, *b* etc), referred to as the *full-height-line*; and the position of the bottom of descending letters (*g*, *y* etc), referred to as the *descender-line* (see figure 1.8).

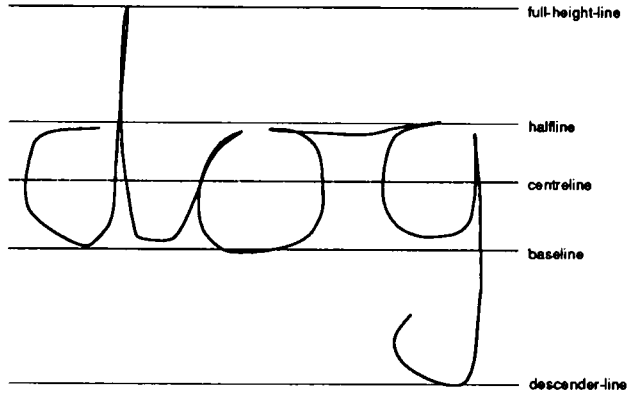


Figure 1.8 - Vertical regions of a script word

1.4.2. Basic Recognition

Prerequisite for Successful Recognition

At this stage it is possible to define a somewhat subjective prerequisite for successful automatic recognition of the handwriting.

The script sample when viewed after preprocessing must be easily recognisable to a human reader.

Some justification for this requirement is necessary. Handwriting has developed as a communication medium between humans. Consequently it relies on a human interpretation of the symbols drawn on the page to overcome the ambiguities inherent within it. It is therefore not possible for a machine to achieve comparable recognition rates without access to all of the information that a human reader has. By insisting that an isolated script sample is legible to a human, the same amount of information is then available to the recognition system as to the human reader.

Segmentation

There are two fundamentally different approaches to the text recognition problem, whole word recognition and segmentation based recognition.

Whole word recognition takes the preprocessed word and uses global information about the whole word, as the starting point, to deduce what was written. Some examples of such a system require the user to train the system with a sample of every word that is likely to be written. This obviously limits the vocabulary of such a system and makes training extremely tedious. Other examples extrapolate information from how one word is written to other similar words.

Segmentation based systems take advantage of the fact that words are made up of a small number of characters which are then combined to form a word. If it were simple to split a word accurately into its component characters, then the recognition problem would reduce to that of separated character recognition. Unfortunately this is not a trivial task. Characters when joined cursively can be segmented in different ways to form other characters. For example, the stroke in figure 1.9 could be interpreted as 'cw' or 'au'. Different algorithms can be used to segment the writing into smaller units. These may be possible characters, or perhaps sub-character components that may be combined at a later stage to form characters.

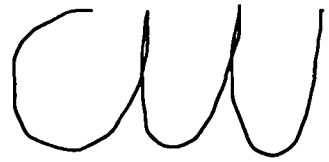


Figure 1.9 - Ambiguous letter segmentation - 'cw' or 'au'?

Most of this thesis is given over to describing a segmentation based recognition system. However, a simple whole word recognition system was implemented in the early phases of this research, and is described in Appendix A[7].

Feature Extraction

Due to the large amount of information available at this stage, the data is often reduced from the list of coordinates to a list of *features* which specify the writing, without loss of relevant information. Key characteristics of the script are

detected and measured to provide these features. A feature may be a binary description of a characteristic, for example *"This stroke is a descender"*, or it may be a list of numeric values describing a shape, for example *"A loop with height 10 units, width 5 units, drawn anticlockwise at position (27,12)"*, or it may be a description of the whole word, for example *"Estimated number of letters is 5."*

These features should describe the writing sample without any loss of important information. One possible way to test this might be to try to recreate the word from the feature description and see if it is recognisable to the eye[29].

Template Matching

The feature description of the sample script can now be compared with information already known to the system about how a word, character or sub-character element is formed. The topological structure of a character is referred to as its *allograph* and the information about this structure, stored by the system, is referred to as a *template*. Templates may be stored in a number of different ways, perhaps as a numerical representation of actual shapes in terms of the features, or perhaps as a set of rules that specify acceptable values of features. The template matching stage consists of somehow comparing the sample data with the templates to produce the most likely words or characters that were written. A number of methods are described in section 2.5.

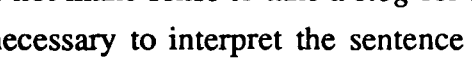
Output from the Recognition Stage

Depending on the approach used at the template matching stage, various forms of output can be produced by the recognition stage. A whole word recognition approach may yield the most likely word, or a list of possible words. A segmentation approach may produce a sequence of most likely characters, or a sequence of ordered lists of possible characters that may occur at various locations in a word.

1.4.3. Postprocessing

Contextual Removal of Ambiguity

Cursive handwriting is ambiguous in its interpretation. A character or combination of characters may be misrecognised as another character or combination (eg *a* may be mistaken for *o*, *au* may be mistaken for *cw* etc) or a word may

be misrecognised as another word (eg  is very similar to  and fairly similar to ). A human reader uses context at various levels to resolve the ambiguity. For example, at the letter level a *q* must be followed by a *u* in English; at the word level the letter combination *docj* does not exist; at the sentence level it does not make sense to take a *clog* for a walk. Even higher levels of context may be necessary to interpret the sentence  . One needs to know if the text in the rest of the passage is discussing a canine that has gone astray, a Morris Dancer in stockinged feet or an absent-minded potter.

Because of this inherent ambiguity, a purely computational CSR system can never be 100% accurate and must also apply context to resolve this ambiguity. Letter sequence and word level verification are now within the scope of modern computer power and theory. Context applied at the higher level of sentence syntax and semantics is being heavily researched. Further details of the use of context are given in section 2.7.

1.5. Scope and Organisation of this Thesis

Cursive script recognition contains problems that still require a large amount of work before a feasible commercial system becomes available. This thesis addresses some of the problems, in particular the requirement for a consistent segmentation method, a trainable, statistical template based recognition module and efficient contextual verification of the output.

The ORCHiD system takes data from a separate preprocessing routine and assumes that this will provide well preprocessed individual words, with the baseline and halfline correctly determined. The segmentation method, developed as part of this research is described in detail in this thesis, but the implementation of the segmentation and feature extraction was carried out elsewhere. The rest of the system, resulting in the output of lists of candidate words, is described in detail.

The remainder of this thesis is organised into eight further chapters followed by a bibliography and three appendices.

Chapter 2 contains a review of other relevant work described in the literature.

A brief overview of the complete recognition system is provided in Chapter 3. There is a discussion of the assumptions and scope of the work covered by this thesis, together with references to the technical descriptions of parts of the system

not covered in this text. The actual techniques used at each stage of the recognition process are briefly discussed.

Chapters 4, 5, 6 and 7 contain details of the major elements of the system. Each chapter has three components - a theoretical introduction to the reasoning behind the methods used, a discussion of the practical considerations to achieve an approximation to the theoretical ideal, and a resumé of the actual output from this section of the system. Chapter 4 provides technical details of the segmentation algorithm used. The features that are measured from the resulting segments are described. Chapter 5 discusses the template matching routines that take the list of features and produce a graph of possible characters. Chapter 6 considers the application of word level context to reduce the remaining ambiguity. Chapter 7 outlines the approach used to train the system to an individual user.

Chapter 8 contains experimental results from the system and recognition rates. These results are discussed.

Chapter 9 outlines what further work is needed to enhance the system.

All of the references in this work are detailed in the bibliography.

Appendix A contains details of a simple recognition demonstrator that was developed half-way through the project.

Appendix B shows an example of a word being processed by the recognition system.

Appendix C contains copies of papers published by the author relevant to this work.

Chapter 2

Literature Review

There follows a review of literature and research into the field of on-line cursive handwriting recognition and its associated subjects. The review is loosely divided into the separate processes described in section 1.4. Those areas which are directly addressed by this thesis (namely segmentation, template matching, dictionary lookup and training) are discussed in detail; for other areas references are provided which give a good grounding in that particular area.

2.1. Background

The history of computer recognition of cursive script (CSR) is long and varied. Lindgren[59] noted that the first research into the field in the 1960's was carried out by speech recognition research teams as a simplified introduction or stepping stone to solving their main problem. (Speech recognition has always been the computer scientist's "dream" machine interface - hence the talking computers of contemporary science fiction.) Speech and cursive script have a number of similarities as communications media, especially the wide variability of spoken accents and writing styles and the lack of clear segmentation points between recognitional units (words in speech, letters in script). The problem domain of recognition is therefore similar and this led speech recognition scientists to investigate cursive script. Few people at that time were interested in handwriting recognition in its own right.

In the 1970's, theoretical pattern recognition techniques, technology and computer power had improved sufficiently for the speech recognition research effort to return to its original goal. Consequently interest in handwriting recognition waned. Towards the end of this decade, however, Computer Aided Design (CAD) technology was becoming increasingly common and a number of companies (eg Numonics, Scriptel, Wacom, Summagraphics, Hitachi) began producing accurate digitising tablets for use with CAD and other applications. These were initially driven by a puck or mouse, but later had a crude stylus as a pointing

device (see section 2.2). During this period, research was continuing into Optical Character Recognition (OCR) and a number of researchers investigated the possibility of using the digitising tablet as a method of collecting data for separated handprinted character recognition. Suen *et al*[88] provided a survey of separated character recognition and Tappert *et al*[93] surveyed the on-line character recognition in particular.

In the 1980's, much research interest was focused on improving the human-computer interface (HCI), and a novel device was proposed based on a flat screen display overlaid with a stylus-driven digitiser, commonly referred to as Electronic Paper (EP)[4,27,89,92]. An ideal data input method for this type of interface is obviously some form of handwriting, and interest has subsequently increased.

2.2. Data Capture and Electronic Paper

The technology available for producing digitising tablets has dramatically improved since the first successful manufactured tablet, the RAND tablet[16]. Pressure sensitive tablets are available which require no special stylus but are not very accurate and are susceptible to errors due to objects resting on the digitising surface. Electromagnetic or electrostatic digitisers are now the most popular for handwriting research as they are very accurate. The major drawback with this type of tablet is a special stylus is required which is usually attached to the tablet by a cable. The stylus has often been designed for simple menu picking and drawing operations rather than sensitive handwriting data collection, and is consequently rather crude. Ward and Phillips[98] discussed the requirements for a good stylus, and Meeks and Kuklinski[62] compared the accuracy and stability aspects of a number of digitisers. Kim and Tappert[52] discussed the effects of poor digitiser accuracy on handwriting recognition rates.

The construction of a usable Electronic Paper style of device is discussed in Higgins and Duckworth[40] and Pobgee[74]. A number of research groups have discussed the underlying advantages of EP[11,27,64,89]. The Japanese interest in this field is especially evident since such a device overcomes the impracticality of Japanese character keyboards. The Far Eastern countries have so far been limited in potential access to the developments in computer technology since they are unable to interact with a machine in their own script.

Since EP devices are only recently becoming available, analysis of their suitability and practicality has been limited to simple development systems.

Rengger[76], Tappert *et al*[92] and informal observations by the author[41] have shown that EP will become a very powerful user interface in the future.

2.3. Preprocessing

The raw data collected from the digitiser is often preprocessed to aid the recognition stage. There are three main elements of the preprocessing stage - word segmentation, noise reduction, and normalisation. The word segmentation provides a sensible unit for the recognition software to act on. Noise reduction removes confusing data introduced by the hardware. Normalisation produces a more standardised piece of script for the recognition process. The different processes are detailed in section 1.4.1. A summary of work in these areas is given in Brown and Ganapathy[13] and Tappert *et al*[93].

Word segmentation is either carried out by identifying a spatial separation within a sample or a temporal separation (or time-out). Noise reduction eliminates some of the problems caused by inaccurate digitisers, though these problems are lessening with improved technology. Normalisation produces a uniform sample word for recognition.

2.4. Segmentation and Choice of Features

The segmentation approach (section 1.4.2) is very common in script recognition systems. The ultimate aim of segmentation must surely be to divide the script into its constituent letters so that the more advanced work into single character recognition can be applied. Unfortunately the nature of cursive script is that the letter segmentation points can only be correctly identified when the correct letter sequence is known, and the recognition of the characters can only be done successfully when the segmentation is correct! Segmentation based CSR systems must therefore attempt to estimate the letter segmentation and recognise characters with the knowledge that the segmentation may be incorrect, or take an approach which locates sub-letter segmentation points.

Three techniques are outlined below. The first uses easily locatable features of the script as possible segmentation points (PSPs) (section 2.4.1). The second attempts to locate theoretical strokes which have been identified by researchers investigating the processes used to produce handwriting (section 2.4.2). The third approach formalises the structure of characters within the script and then segments based on this information (section 2.4.3). Figure 2.1 shows examples of

some of the segmentations described in the literature, detailed in the following sections.

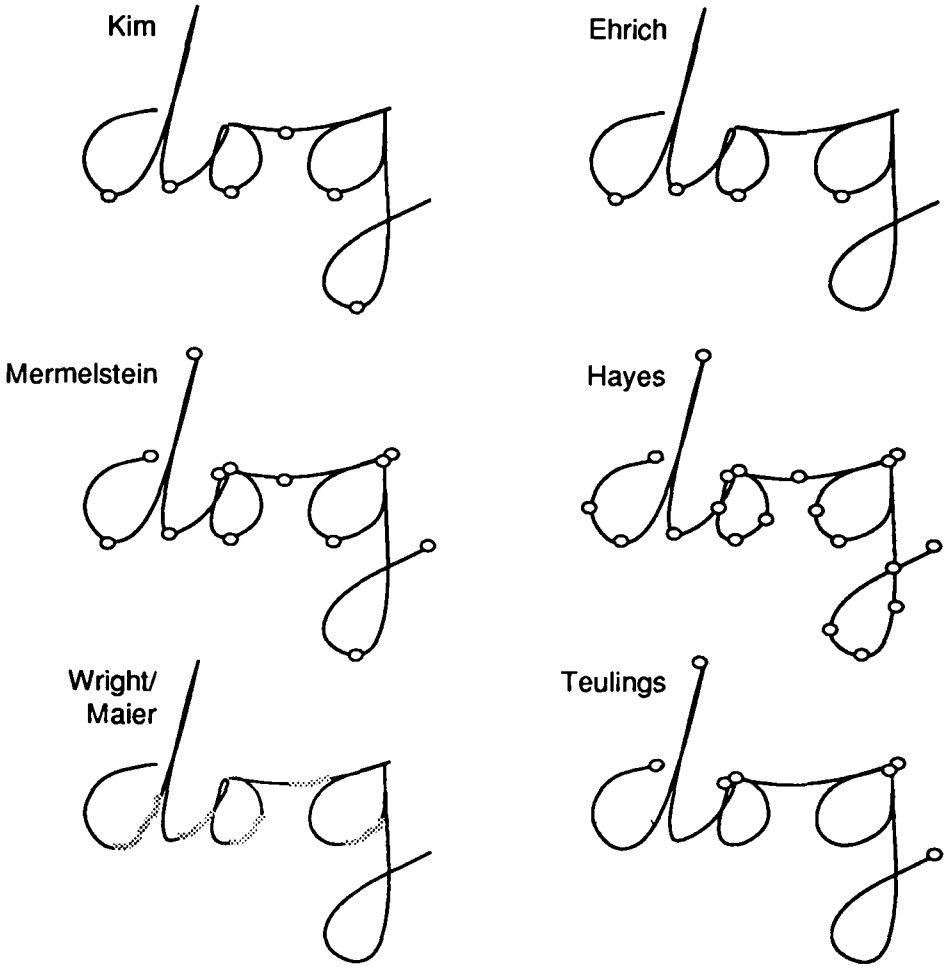


Figure 2.1 - Segmentation Points

2.4.1. A Maximum/Minimum Approach

Many systems base their choice of segmentation points on local maxima and minima in the x and y directions, often treating the y minima as most important. Kim[53] segmented at all local y minima. Ehrich and Koehler[23] used the same points, but excluded those relating to ornaments or small loops. This yields segmentation points that mostly lie near the baseline. Mermelstein and Eden[63] used y maxima and minima to segment words, noting that the down-strokes usually contain more information about the script than the up-strokes. Hayes[37] used all maxima and minima as PSPs as well as intersections within the script.

Wright[104] developed a handwriting system based on Freeman coding (see section 2.5.2) and rejected the use of *y* minima as possible segmentation points in favour of identifying upward right-pointing strokes which are commonly used as letter joins. Maier[61] segmented by attempting to identify commonly occurring letter join strokes or *ligatures*. These techniques are a form of *y* minima segmentation, however, with the actual location of the segmentation point moved along the stroke.

There are several problems associated with the use of maxima and minima as PSPs. Firstly, many *y* minima, for example, occur in the middle of an allograph and so are not letter segmentation points. Ehrich and Koehler's approach removed some of these PSPs but not all. Secondly, subtle changes in the formation of a letter can add or delete maxima or minima. It is then necessary to use more than one template to represent the same allograph (see figure 2.2). Thirdly, if just *y* minima are used, for example, there can often be no segmentation point between letters (see figure 2.3). It might therefore seem preferable to include more PSPs, *y* maxima for example, but this would cause great deterioration in the performance of the system. There must be a trade-off with this sort of approach of consistency of segmentation against performance.

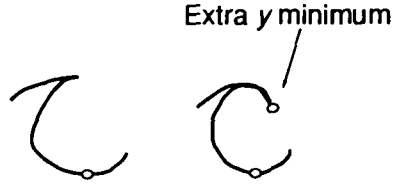


Figure 2.2 - Extra y minima

2.4.2. Absolute Pen Velocity

Teulings *et al*[94] segmented the script into strokes at points of zero absolute velocity. This produces a stroke similar to the hypothetically "correct" stroke defined by psychophysicists[66]. Kadirkamanathan and Rayner[47], however, pointed out that the theoretical segmentation points attributed to zero pen velocity are often obscured due to the discrete quantisation of the digitising process. They used an algorithm which compares the curvature and pen velocity graphs of the script after it has been smoothed by varying amounts. This produces a more accurate hypothetical stroke, but the segmentation points are again located at *x* or

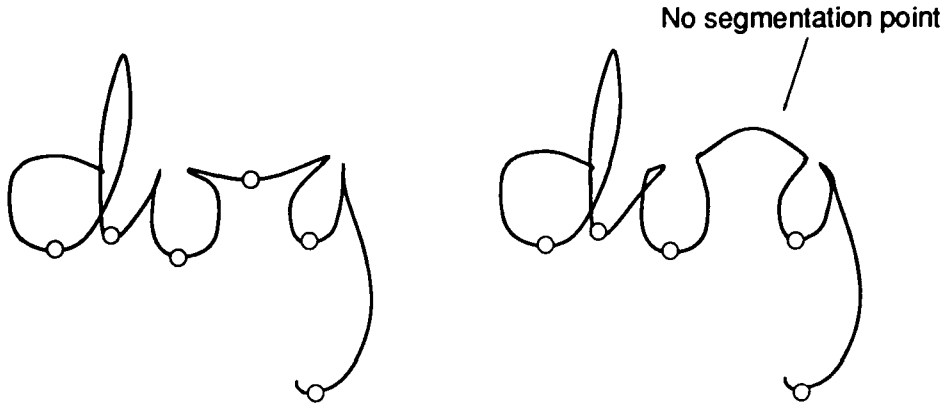


Figure 2.3 - No *y* minimum segmentation point

y maxima or minima depending on the local circumstances.

2.4.3. Characterisation of Script

A more intuitive approach might be to try and characterise or formalise the characters and constituent elements of cursive script in order to try and locate the correct segmentation points. There have been a number of different approaches to characterising characters within a cursive word. Eden and Halle[22] defined a rather complex set of primitives from which words can be built up (figure 2.4). James[45] produced a simplified but less precise set of primitives (figure 2.5) which Higgins[38] simplified still further. Hayes[37] used i-dots, t-crosses, x-slashes, ascenders, e-loops, circles, c-shapes, i-spikes, humps and descenders (figure 2.6). These primitives are all devised with the underlying hypothesis that the initial segmentation uses minima and maxima. In fact, Higgins and Hayes use these characterisations as a second level representation within a hierarchical recognition method.

Berthod[2] took a more theoretical approach and identified key features of the script that may be recognised relatively easily (figure 2.7). These included:-

- Eden and Halle primitives;
- cusps in four directions;
- humps, clockwise and anticlockwise;
- loops delimited by an intersection;
- closures, as occur in the letters *a*, *d*, etc;

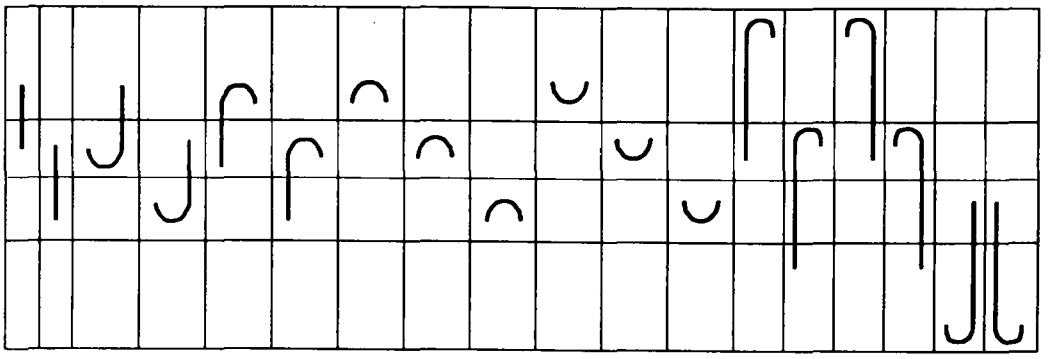


Figure 2.4 - Eden and Halle primitives

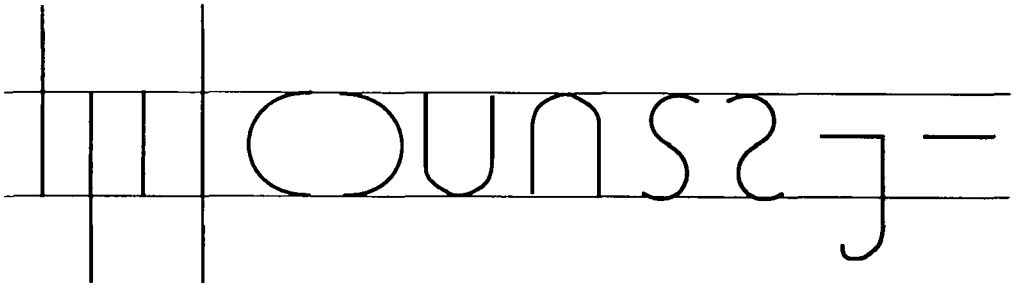


Figure 2.5 - James primitives

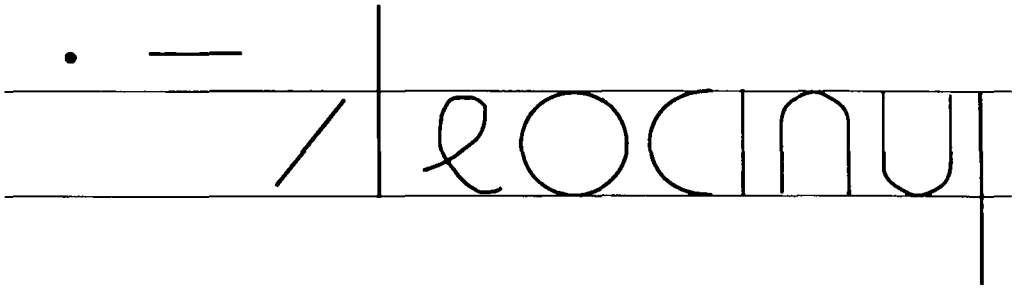


Figure 2.6 - Hayes primitives

- diacritical marks, such as i-dots, t-crosses, x-slashes.

Berthod and Ahyan[3] implemented a system based on x and y extrema, cusps, intersections and inflections.

The segmentation points defined by these script characterisations are often more difficult to detect but appear to be more consistent between slight variations of the same letter. The segmentation described in Chapter 4 of this thesis is based

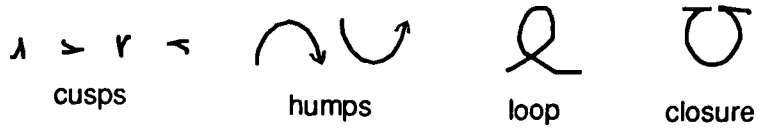


Figure 2.7 - Berthod primitives

on this approach.

2.5. Template Matching

After segmentation of the script, the usual approach is to compare the segments against templates or prototypes of actual characters. There are four main techniques that predominate for template matching.

2.5.1. Elastic Matching

The elastic matching technique directly compares the segment or stroke with a prototype or template and calculates a distance metric from the minimum distance that can be calculated between the digitiser data points (DDPs) on the sample stroke and the template stroke. Figure 2.8 indicates the measurements used to calculate the elastic matching distance. A dynamic programming method, common in speech recognition systems[80], is used to calculate the minimum possible distance.

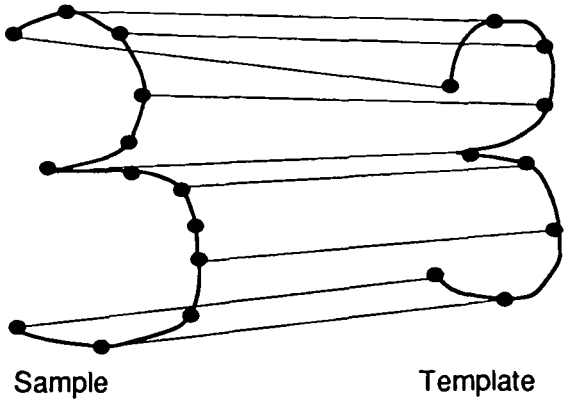


Figure 2.8 - Elastic matching

Burr[14], Tappert[90], Wong and Fallside[103] and Kadirkamanathan and Rayner[47] all used this approach.

Elastic matching provides a graphical approach to template matching, comparing sample shapes to ideal templates. Efficient algorithms can be used to calculate the distance measurement relatively quickly. The main problem, in a cursive script recognition context, is that two letters, similarly structured as far as the writer is concerned, can look very different when drawn on the page. For example, a letter *l* can be written either with a very large loop or with a carefully overdrawn line. These would require separate templates for successful recognition using an elastic matching technique.

2.5.2. Freeman Coding

Freeman[32] proposed a method for representing a geometric configuration, such as a stroke within script, by a simple numerical code. A small number of directions are specified and labelled, as in figure 2.9. A stroke is then encoded by dividing it into equal length parts, and coding each part by its nearest directional label. A complex curve can in this way be represented by a short list of digits. Templates are then stored as Freeman codes and the sample is compared with the templates and a distance measure calculated.

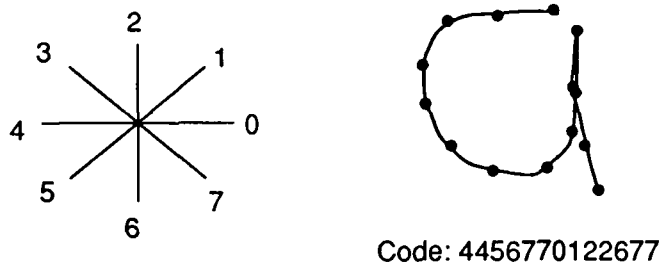


Figure 2.9 - Freeman coding

Miller[65] segmented a stroke into six parts and encodes these with an 8 direction code so that each stroke is represented by six digits. A simple sum of the differences between the coded sample and template is used as the distance measure.

Wright[104] used an 8 direction code, but compresses any sequential vectors in the same direction into one code, and stores a length for each vector. Each template is stored as a five vector code, by compressing any short vectors into a neighbouring vector (figure 2.10). A sample segment is similarly compressed, and for those templates that match the coding precisely, a Cramer Von-Mises

goodness of fit measure[56] is calculated based on the lengths of the vectors.

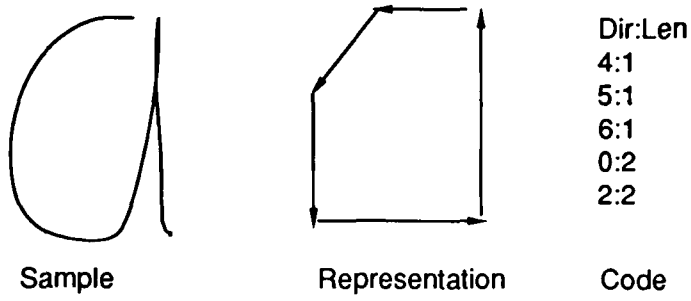


Figure 2.10 - Wright's coding

Ouladj *et al*[71] used a 16 direction coding of the script, and then modified this to a 4 direction coding for the recognition stage. This coding is then modified to allow for degradation of the writing from the ideal template model (figure 2.11), and a distance calculated based on the amount of degradation needed for a match.

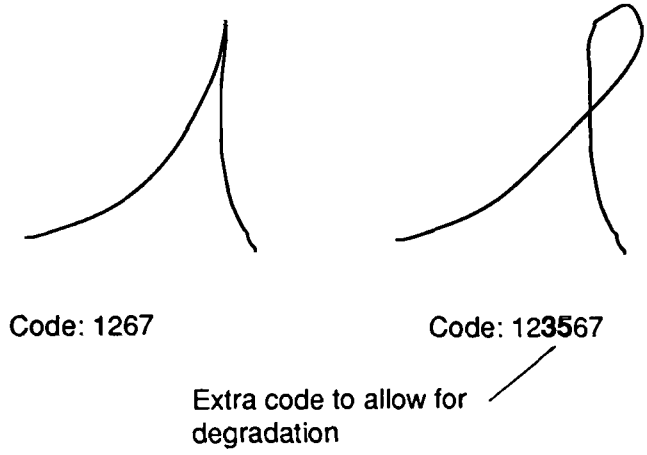


Figure 2.11 - Ouladj's modification due to writing degradation

The problem with systems based on Freeman coding is one which is common with any system which measures features on a discrete scale. Errors will often be introduced due to the discrete quantisation as a feature nears the boundary from one discrete value to another. Figure 2.12 shows some examples where small changes in the shape of a character cause large changes in the Freeman coding.

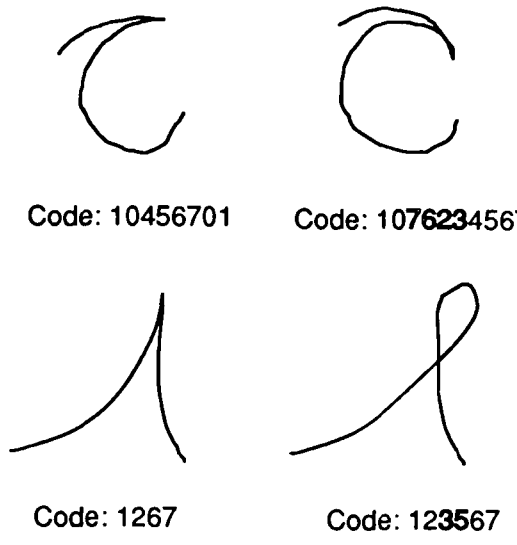


Figure 2.12 - Possible errors due to Freeman coding

2.5.3. Feature Matching

Another technique involves measuring various attributes or *features* of a segment and using these to calculate the distance measure during template matching. This technique is similar to elastic matching since the segment is not simplified (as is the case with Freeman coding). It has an advantage over elastic matching, however, since the shape of the segment is treated as a whole, rather than as a sequence of points that may change dramatically between samples. It will also take a fixed amount of time for the comparison whereas elastic matching is proportional to the number of points in the segment. The main drawback with this method is that it is essential to measure the correct features, otherwise vital information about the segment may be lost.

Teulings *et al*[94] extract four types of feature from a stroke - the vertical positions of the endpoints of the stroke, five angles measured between certain points along the stroke, the size of the area enclosed by the stroke, and information about pen ups within the stroke. (A stroke as defined in Teulings' work includes pen motion above the paper.) A Euclidean distance measure is used to describe the success of the template match.

The work described in Chapter 5 of this thesis uses a feature matching approach.

2.5.4. Rule-Based Matching

It is possible to describe characters in terms of a set of rules, and to use these rules to distinguish between the characters. For example, to distinguish between an *a* and a *d* would require a rule that defined the height of the last part of the character. By defining a set of rules that differentiates between the complete alphabet or stroke set, a recognition system can be constructed.

Frishkopf and Harmon[33] develop a set of rules based on the vertical extent of characters below and above the baseline and halfline, the presence of retrograde (right-to-left) strokes, the position of cusps, the presence of closures, and diacritical marks.

The problem with any rule-based recognition system is that it is frequently possible to find a counter-example which does not satisfy the rules. It is therefore necessary to either restrict the styles which can be recognised or add a large number of special case rules which seriously degrade the performance. Also such a system cannot easily be trained to individual users to improve performance.

2.6. Whole Word Recognition

Rather than attempting to solve the problem of segmenting script into letters, some researchers have investigated the possibility of identifying a word as a single unit. The problem with this approach is that it is often necessary to provide a training sample of every single word that the system must recognise. This is obviously not possible for a large number of words.

Earnest[21] developed a system which identified a small number of easily recognisable features of the script. These were ascenders, descenders, closures, t-crosses and centre-line crossings. These features yielded a category code for the word which identified matching words from a dictionary of ideal template words. The x coordinates of the features were then used to calculate a match weighting.

Brown and Ganapathy[12] used a larger set of features, including y maxima and minima, centre-line crossings, ascenders, descenders, t-crosses, i-dots, cusps and closures. The length of the word was estimated by counting the total number of centre-line crossings, and the position of each feature was registered with reference to its approximate letter position. A nearest neighbour approach was then used to identify likely words.

Frishkopf and Harmon[33] examined each x and y maximum and minimum for vertical position, positive or negative slope and concavity. They generated a

feature vector of this information which was used for a nearest neighbour comparison with a dictionary of ideal template words.

Farag[26] developed a simple system based on a Freeman style coding of the script and a Markov chain model to calculate a weighting when comparing the sample with a template word. This technique only used a very small template set.

2.7. Postprocessing

As explained in Chapter 1, handwriting is inherently ambiguous at all levels, and it is impossible to uniquely classify a script sample. This ambiguity can often be resolved if contextual information at a higher level can constrain the set of valid output. A dictionary (lexicon) of this information can then be constructed, and the list produced by the recognition process reduced and validated. Often very large substitution sets are produced which need to be checked quickly and efficiently against a large dictionary. This is especially the case with cursive script recognition systems.

Several different techniques have been proposed to make use of contextual information.

Statistical information about the transition probabilities between letters can be used to remove or reduce unlikely letter sequences and produce the required output (eg Neuhoff[69], Riseman and Hanson[78], Hull and Srihari[42]). This is often referred to as a *bottom-up* technique. These techniques are usually very efficient, but do not guarantee valid output words.

Another approach assumes that the written word comes from a fixed dictionary and the nearest matching word provides the required output (eg Duda and Hart[19]). This is often referred to as a *top-down* technique. These techniques always produce a valid output word but can be inefficient to implement.

Several hybrid methods have been suggested to balance out the advantages and disadvantages of the two approaches (eg Srihari *et al*[86], Shinghal and Toussaint[82]).

Most of the approaches to using contextual information have been based on constructing a post-processor for systems that produce a single best-match letter sequence as output. The contextual disambiguation process takes this sequence, consisting of the most likely letters that span the script, as its input and returns the most likely written word as its output. There are several techniques which have been widely reported, the most common are listed below.

2.7.1. Spelling Correctors

One simple method to verify the output is to use the wealth of research into spelling error detection and correction that has been accumulated over the years. Peterson[73] and Pollock[75] provide good reference lists to the literature. The main problems that these techniques address are character substitution, omission and insertion. Kashyap and Oommen[48,49], Wagner and Fischer[97] and Lowrance and Wagner[60] discussed different solutions to this problem. The Levenshtein metric is frequently used to define the distance between two strings based on the number of corrections that need to be made to convert one string to the other[70,81].

These techniques work reasonably well but do not take into account the *likelihood* of a letter being misrecognised, or incorrectly substituted.

2.7.2. N-Gram Techniques

The probability of any individual n-letter sequence occurring can be calculated by examining large pieces of text. These probabilities can then be used to calculate the most likely written word given the output word from the recognition system, (see Riseman and Ehrich[77]). An alternative approach using n-grams to reduce a graph of possible letters is discussed in Chapter 6.

2.7.3. Viterbi Algorithm

The Viterbi Algorithm (VA) takes the output word from the recognition system and calculates the most likely input word, using statistical information on the sequence of letters in English and likely errors from the recognition system. The algorithm is first described in Viterbi[95]. Forney[31] provided a thorough tutorial introduction to the theory behind the algorithm. Neuhoff[69] described how it could be applied to the problem of text recognition. Various authors have discussed its application, including Riseman and Hanson[78] and Hull and Srihari[42] who compared its performance against a binary n-gram approach.

The VA makes use of a *confusion matrix* of *a priori* probabilities observed from the recognition system, together with the transition probabilities between characters. In other words, the probability that a given letter may be misrecognised as another letter is calculated and stored, together with the probability that it can be preceded or followed by any other character.

A $26 \times l$ node trellis is constructed, where l is the length of the word, linking every letter with every other letter (see figure 2.13). On the nodes of the trellis

are the confusion probabilities, and on the arcs are the transition probabilities. (Notation: The confusion probability that a written letter X will be misrecognised as a Y is written P(X | Y). The transition probability that a letter X may be followed by a letter Y is written P(X - Y).) By tracing a path through this trellis, and combining the probabilities on the arcs and nodes included in the path, the probability that the traced word could have been the original input word can be calculated. The algorithm finds the most likely path through this trellis, so yielding the most likely interpretation of the input word.

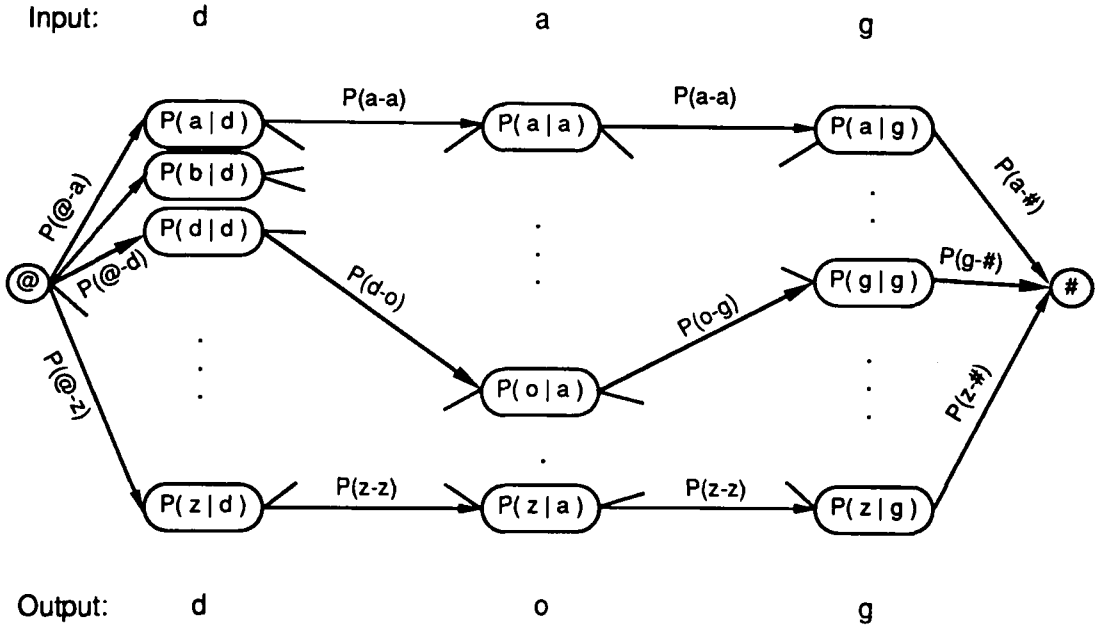


Figure 2.13 - Viterbi algorithm trellis

2.7.4. Dictionary Viterbi Algorithm

The VA produces the most likely interpretation of the written word, but does not guarantee that the word exists. The Dictionary Viterbi Algorithm (DVA), developed by Srihari *et al*[86], is an improvement on the VA making use of a simultaneous full dictionary search, in place of the transition probabilities, to ensure that only a valid word is produced. The dictionary lookup uses a *trie* structure to store the dictionary. This is described in section 2.7.7.

2.7.5. Modified Viterbi Algorithm

Shinghal and Toussaint[83] described another variant of the VA, called the Modified Viterbi Algorithm (MVA). Here a heuristic depth of search d is set by the user so that only the d most likely probabilities in each letter position are checked. The computational overheads of the VA are thus reduced, as only a $d \times l$ trellis needs to be traced, but the performance degrades as d is reduced.

2.7.6. Predictor-Corrector Algorithm

Shinghal and Toussaint[82] further improved on the MVA by combining it with an efficient dictionary lookup algorithm (DA). This is called the Predictor-Corrector Algorithm (PCA).

The dictionary is partitioned into sub-dictionaries of same-length words. Each sub-dictionary is then sorted by *value*, where the *value* is calculated by combining the transition probabilities of the letters of the word. This *value* was found to be nearly always unique for any sub-dictionary. A binary search is used to see if the output word from the MVA exists in the dictionary. If so, then that is taken as the required output. If not, then the DA is employed. A *score* is calculated for the nearest f words to where the output word was expected. f is a heuristic set by the user. The *score* is calculated by combining the transition probabilities between the letters with the confusion probabilities. The word with the largest *score* is the required output.

Shinghal[84] described a further enhancement to the PCA, where the n most likely words from the MVA are checked in decreasing order to see if they exist in the dictionary. If none of them exist, the DA is employed. The value of n is determined by experiment.

2.7.7. Dictionary Tree Structure

A dictionary or word-list can be restructured in the form of a tree, based on the *trie* structure suggested by Knuth[54]. This is shown pictorially in figure 2.14, where the tree represents the word list {a, an, and, at, be, bet, but, by}. Each of these words can be found by tracing a path from left to right. The '@' symbol represents the start of a word and the '#' symbol represents the end of a word.

The *trie* can be used as an efficient structure to store a dictionary for a variety of applications, but is especially applicable for contextual post-processing of script recognition systems. The DVA proposed by Srihari *et al*[86] used a

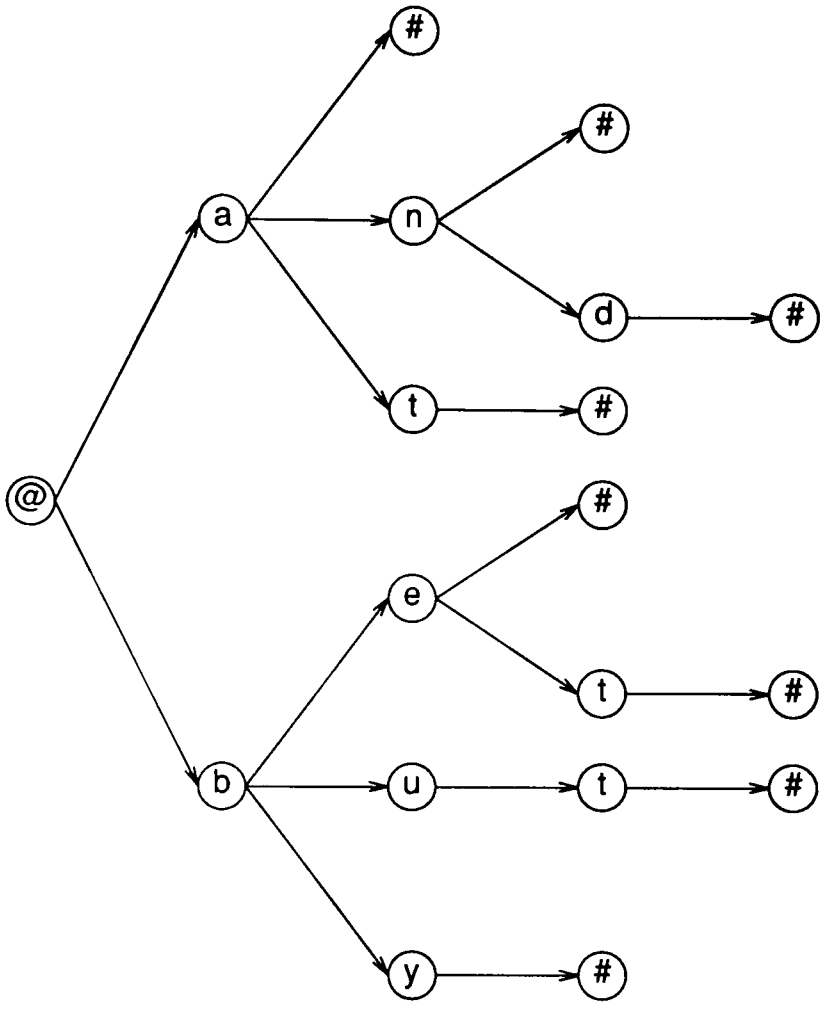


Figure 2.14 - Dictionary tree

dictionary *trie* at the same time as the Viterbi lattice is traced to guarantee that a valid word is found. Bozinovic and Srihari[5] combined a stack-decoding search algorithm with a *trie*-structured dictionary with a small dictionary of 1027 words. Bozinovic and Srihari[6] used a similar approach but adds a depth of search heuristic to limit the computation needed.

Until recently, the availability of computer memory and CPU power has prevented the use of such a data structure with very large dictionaries, however these can now be seriously considered. Ford[28] discussed the development of a simultaneous graph traversal and dictionary tree lookup technique (described in detail in Chapter 6). A number of other research teams have also subsequently shown that this method is a very efficient and effective way of improving the

accuracy of text recognition systems[24, 100, 101].

2.8. Higher Level Context

Since some ambiguity can remain even after a dictionary verification of the output from a CSR system, some investigative research is being carried out into the use of higher level context to reduce the ambiguity still further. This includes analysis of the syntax and semantics of English sentences[50, 51, 79]. A machine readable dictionary is appended with information about the word which can be used by the higher level contextual system.

2.9. Training

A number of systems (identified in table 2.1 below) can be trained to an individual writer's style. This training usually improves the recognition rates obtainable by the system. The exact method of training is obviously dependent on the recognition method but usually involves calculating certain parameters of writing that vary between different writing styles and different characters. The problem with training a segmented CSR system is that either the training must be carried out manually, identifying each character of a word in turn, or automatically, in which case the system must decide where each character lies within a word. The automatic approach is preferable, since this requires least effort on behalf of the user, but is most complex. For whole word recognition systems there are other problems, notably that it may be necessary to provide samples of each word that the system is to recognise.

A manually trained system may be devised that requests the user to write each character separately to train the templates. Unfortunately this is not ideal since most people write characters differently when in the middle of a word to when written separately (even when consciously trying not to) and often write letters differently in different locations within the word, depending on the preceding and following letters. It is also necessary to take into account the joining strokes between letters.

Teulings *et al*[94] automatically trained their system by comparing the features of similar words and deducing which features represented which letters. For example, the training phase may require the user to write the words *as* and *an*. Those features which were the same between these two words would then be noted as the 'a' template. If the user then writes *vase* and *van* the 'v'

template can be identified, and so on. With this approach, care must be taken to use a sensible training sequence of words. For example, the words *ash* and *ask* would not correctly identify the 's' since the ascender of the next letter would also be the same between the two words.

There is an inherent problem with the completely automatic training of templates. If the automatic system is not 100% accurate at identifying the templates to be trained, errors will occur and templates will be trained incorrectly. It is possible that these errors may be compounded and the user will have no knowledge of the situation, other than falling recognition rates. This problem is discussed in detail in section 7.3.

2.10. Recognition Rates

It is very difficult to compare recognition rates of CSR systems. The extent of the recognition process covered by the system - up to letter recognition, or word recognition, or dictionary verified word recognition - obviously affects the recognition rates quoted dramatically. The experimental details of the test routines used to generate the statistics also affect the results. A number of researchers, however, quote results of recognition where the test set is the same as the training set. This clearly biases the results. A trained system generally produces better accuracy than a similar untrained system. Some works make use of a dictionary verification of results which improves the recognition rates still further, especially when only a small dictionary is used.

A table of main results published to date is given below. These results are for on-line cursive script recognition and quote dictionary-verified word recognition rates where applicable. A number of works only quote letter recognition rates. In these cases an estimate of a six letter word recognition rate is given in brackets. Some papers only quote word recognition without dictionary verification. Some of the recognition rates are for the correct word appearing in the top few (<5) words in the output list of candidate words. It seems reasonable to quote these figures as it is often not possible to distinguish between some words without sentence or higher level context.

Name	Year	Trained	Dictionary Size	Word Rate
Frishkopf[33] (1)	61	No	Letter rate	59 [4]
Frishkopf[33] (2)	61	Yes	100	63
Harmon[36]	62	No	Letter rate	93 [65]
Earnest[21]	62	No	10,000	60†
Mermelstein[63]	64	Yes	None	80
Ehrich[23]	75	No	None	71
Brown[12]	80	Yes	None	~70
Berthod[3]	80	Yes	None	87
Tappert[91]	84	Yes	Letter rate	97 [83]
Higgins[38]	85	Yes	25,000	85-93
Wright[104]	89	No	60,000	82-94
Ouladj[71]	90	Yes	110	94

† This rate is for the correct word appearing in the output list. This list may consist of a large number of words, 20 on average.

Table 2.1 - Recognition rates

2.11. Conclusions

The recent increase in research into the field of cursive script recognition does not appear to be reflected in vastly improved recognition rates. Careful examination of the experiments, however, show that the more recent work is generally based on more realistic test criteria, for example more sample words from larger vocabularies, written in less restricted styles. It should, of course, be noted that even humans cannot achieve 100% recognition of handwriting without the additional use of context. Neisser[68] conducted an experiment to determine the human recognition rates for a very limited set of single hand-printed characters, consisting of the upper case capitals, and the digits 2-9. The best results that were achieved were 96% recognition. Leedham and Chan[11,58] showed that for unconstrained, upper and lower case separated characters and digits, the human

recognition rate is as low as 74%. From informal observations it seems that human recognition of isolated handwritten words is around 95%, though no formal study has been identified in the literature. It would therefore seem that this must be a realistic goal to aim for with an automatic system, without the use of higher context.

There are still a number of problems that need to be addressed in CSR and these are discussed in this thesis. These include the need for a consistent segmentation method, the use of context to improve recognition, the ability to quickly and easily train the system to a particular user or writing style, and the further investigation of whole word recognition approaches.

Chapter 3

ORCHiD System Overview

This chapter provides an overview of the complete recognition system, from on-line data input, to ascii-coded data output. The system is called ORCHiD or On-line Recognition of Connected Handwriting Demonstrator. A brief description is given of the techniques and algorithms used at each stage of the process. Further details on each of the stages that are relevant to this thesis, including technical and experimental results which accounted for the selection of the algorithms, are given in the subsequent chapters.

3.1. Outline of the System

There follows a brief summary of the processes and the information available at the various stages of the ORCHiD system. Figure 3.1 shows a diagram of these processes.

i) The *digitiser* produces a time ordered list of coordinates, referred to as **Digitiser Data Points** (DDPs).

ii) The *preprocessing* routine takes the raw DDPs and adjusts them to provide a truer image of the script. Global information is calculated about the word, including the **baseline**, **halfline** and **diacritical marks**.

iii) The *segmentation* routine processes the adjusted DDPs and produces a list of **Possible Segmentation Points** (PSPs) and refines this to a list of **Definite Segmentation Points** (DSPs). Each segment of the script is then measured and a list of **features** is produced.

iv) The *template matching* routine takes the list of features and compares it with the **template database** to produces a weighted **letter graph** of possible letters and letter-joins (the **candidate allographs**) that may occur within the script word.

v) The *dictionary lookup* routine traces the letter graph and a tree-structured **lexicon of valid output** simultaneously to produce a ranked list of **candidate words**. Information regarding **permissible letter joins** is also used at

this stage to reduce the output produced.

vi) The *output* routine reduces the list of candidate words by comparing the ranks, applying the diacritical mark information and using *a priori* information of the **word frequency** of English usage. The final candidate word list is displayed to the user for confirmation.

- The *training* routine trains the template database using the knowledge of the confirmed output from the user.

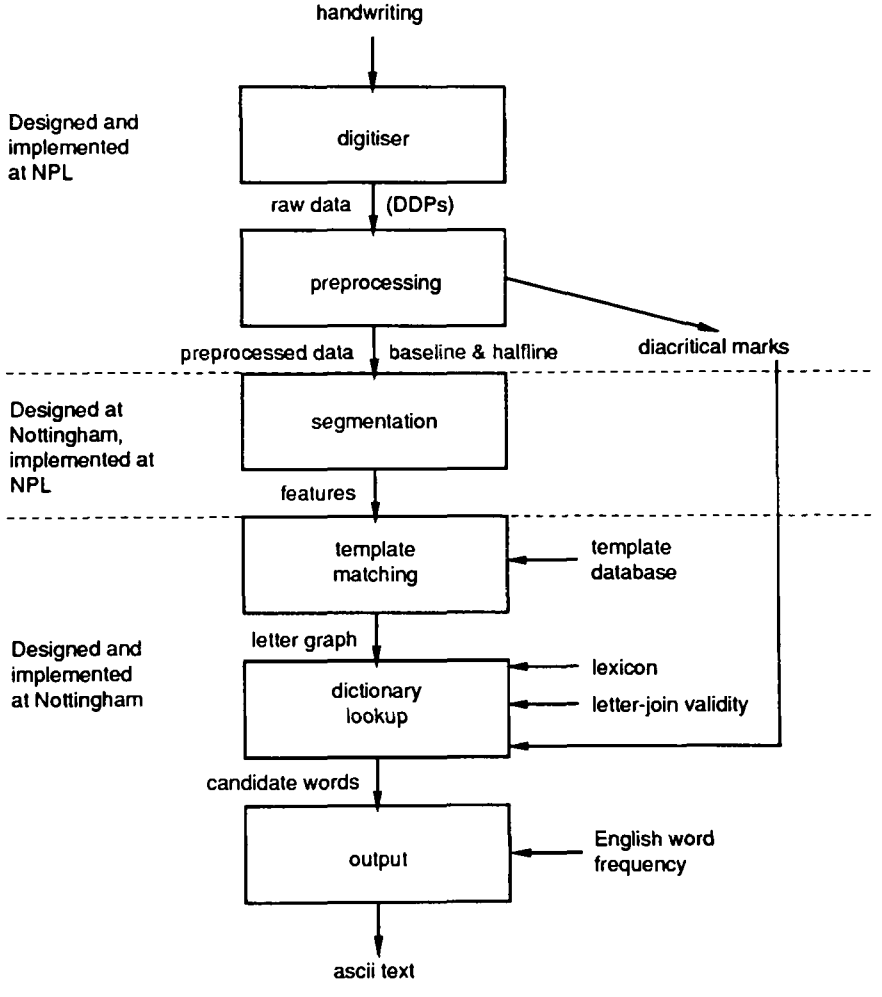


Figure 3.1 - The recognition process

The software has been written in separate modules which can be run independently of the other modules. These modules consist of

- preprocessing, verification, segmentation and feature extraction;

- template matching;
- dictionary verification;
- output sorting and reduction;
- output verification or correction;
- template training.

For a research system this modular approach gives the greatest flexibility for development. Each module can be separately developed and tested. Possible replacement modules can be tested by inserting them into the complete system at the appropriate place and comparing them with the original modules. The output from each stage can be stored in files for later examination.

When the methods and techniques are proved, these separate modules could be combined within a single process to improve the speed of execution. Under a UNIX† environment the modules can be piped together to give the impression of a single process.

3.2. Project Organisation

This research was initiated as part of the Electronic Paper Project (EPP)[8,11,27], undertaken by the National Physical Laboratory (NPL). The EPP included investigations into all aspects of the *Electronic Paper* concept including the supporting hardware and the intelligent controlling software. Research into cursive handwriting recognition was split between two sites, NPL and The University of Nottingham. The work was divided into logical sections as follows:- the data capture and preprocessing was carried out entirely by the NPL; the segmentation and feature extraction was specified by Nottingham and implemented by the NPL; the template matching, dictionary lookup and automated training were carried out entirely at Nottingham.

The preprocessing is described in Brocklehurst and Kenward[10]. Implementation details of the segmentation and feature extraction is described in Ford and Symm[30] and Cox and Harris[15].

The Electronic Paper Project was terminated at NPL before the completion of the handwriting recognition research. The preprocessing software has therefore remained static while the development of the remainder of the system was

† UNIX is a trademark of Bell Laboratories.

completed. The research described in this thesis has been developed assuming that a well preprocessed individual word can be supplied to the system. Since a large proportion of the preprocessing routines are hardware specific (notably to the digitiser), these need to be rewritten for any new technology that is used in conjunction with the system. It therefore seems reasonable to base this work on the assumption that well preprocessed data will be available to the rest of the recognition system.

3.3. The Underlying Principles of the ORCHiD System

3.3.1. A Representation of the Script Recognition Process

One way of representing the script recognition process is to consider the input word as a set of data (X) which is acted on by the recognition system (r) to produce a set of possible output *word candidates* (Y) [$Y = r(X)$].

The philosophy used by the ORCHiD system described in this thesis employs an extension to this model, that is that the set of input data (X) has itself been produced by another process, the writing process (w). In the writing process, the brain of the writer decides that it wishes to write a word (W) and produces some form of a mental picture of that word. The motor control part of the brain then instructs the muscles of the hand, wrist and arm to move in such a way that the pen draws shapes on to the page that resemble the mental picture, to a greater or lesser extent. This is our input data (X) described above [$X = w(W)$]

The image produced on the page deviates from the mental picture in such a way that the image is still identifiable as the correct word to a human reader. The deviations must therefore remain within certain tolerances which will differ depending on circumstance. For example, an isolated word must be written more carefully to be successfully recognised than a word within the context of a paragraph.

It can now be seen that the ideal recognition system (r_i) would be the inverse of the writing process [$r_i = w^{-1}$]

The ORCHiD system attempts to make use of this extra information in the development of a cursive script recognition (CSR) system.

3.3.2. A Statistical Approach

In order to allow the recognition system to be as generalised as possible it was decided to use a statistical approach throughout. This has a number of benefits.

- By using sound statistical principles at each stage of the recognition, all decisions and probabilistic calculations can use standard techniques and theories.
- The recognition routines can be data-driven, rather than rule-based, so it is not necessary to specify in the code particular features of characters. The definition of a template for a character was defined by presenting the system with a number of samples of that character, and recording the average values and spread of the samples. This allowed total flexibility of character definition, so that the system could easily be trained to recognise new characters.
- The template matching algorithm calculates a probability of match for each segment of the sample word, assuming a multivariate normal distribution for each segment. (Section 4.7 discusses the normality of features for each segment.) This is a true probability, not an *ad hoc* weighting, conditional on the fact that any other template may match the segment. These probabilities are then combined across the whole word for each candidate word to give a probability that the whole word is correct.
- The various stages of recognition developed within this system are not limited to cursive script, but might be readily applied to other fields of pattern recognition. In particular, the template matching and template training routines might be applicable for any trainable recognition system that provides feature information from an unambiguously segmented data sample, and the dictionary lookup may be applied to any system that requires lexicon-based context verification as a postprocessor of the recognition phase.

3.3.3. Retention of Ambiguity

Handwriting is an inherently ambiguous communication medium. It is frequently not possible to classify a character or word without reference to the surrounding context. There is a conflict here with any automatic recognition system that needs to make definite *binary* decisions about whether some information is relevant or can be discarded so that the data stored within the program does not

become unwieldy and unmanageable. A successful recognition system will need to retain ambiguity for as long as possible, so that contextual postprocessors can apply their additional information to produce more accurate (or more probable) output.

The ORCHiD system described here attempts to retain as much ambiguity as possible throughout the whole recognition process. Binary cut-offs are necessary at certain stages to reduce the processing required, but this is kept to a minimum. A weighted list of candidate words is produced. At present, the most likely of these words is displayed as output, but the whole list is available if further contextual postprocessors are supplied, for example syntactic or semantic verification.

3.4. Hardware Configuration

Several different hardware configurations have been used at various stages in the development of this system, taking advantage of the constantly evolving technology. Initial work was carried out on an ICL Perq 2 graphics workstation, with input from a Summagraphics digitising tablet with specially adapted stylus. With the development of the PAD, the Electronic Paper Demonstrator[40], handwriting data was captured by its digitising surface and transferred to the Perq via serial line. The original demonstration system was developed using this technology - see Appendix A.

The PAD was integrated with a Sun 3 workstation, and work was transferred to this processor. The stylus on the PAD was not of sufficient quality for collection of large quantities of handwriting sample data, and so the Pencept Penpad was used as an input device, connected to the Sun workstation via the serial line. This opaque digitiser has a very acceptable stylus for data collection[98].

An ideal hardware configuration for this system might be a commercial electronic paper hardware interface connected to a very powerful reduced instruction set (RISC) processor with a large amount of memory sufficient to hold the very large data structures used.

3.5. Preprocessing and Stroke Reconnection

The data is preprocessed by the NPL routines described in the documents listed above. *Diacritical marks*, which we will define to mean *dots*, as occur on the letters *i* and *j*, and *crosses*, as might occur on the letters *t* and *f*, are detected by these routines. Descriptive and positional information about them is

extracted and stored to be used later, at the dictionary lookup stage, to eliminate unlikely words. The strokes which form these diacritical marks are then removed from the sample and ignored in all further processing. The existence of diacritical marks is not ambiguous within this system, but their precise identification is ambiguous.

After diacritical mark removal all non-connected strokes within a word are connected together to produce a sample word with a minimum number of pen-lifts within it. This stroke reconnection or gap removal is done for several reasons. Primarily, reconnection eliminates unintentional gaps from the script caused by light pen pressure or an insufficiently sensitive digitising tablet, but it has the advantage that many letters that can be written with numerous strokes appear similar to completely cursive variations of the same letter after reconnection (see figure 3.2 for examples).

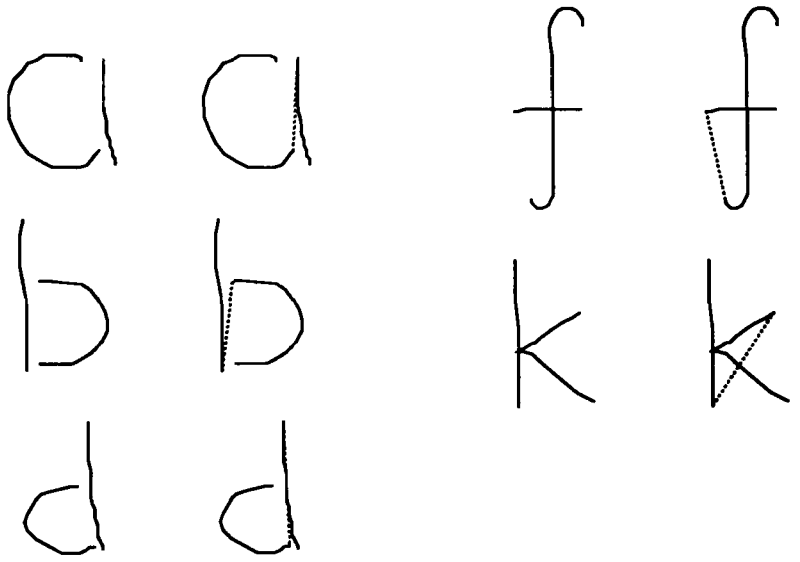


Figure 3.2 - Letters before and after stroke reconnection

The reconnection is at present carried out by joining the two end-points with a straight line. Some digitisers can provide positional information about the pen motion when it is not in contact with the digitising surface. This might provide a more accurate image of the pen motion.

3.6. Visual Verification and Correction of Preprocessing

As mentioned in section 3.2, the preprocessing software has not been fully developed along with the rest of the recognition system. The preprocessing software occasionally fails causing incorrect data to be passed onto the rest of the system. A routine has been implemented that allows the manual correction of word-split errors and baseline and halfline errors which have occurred within the preprocessing. In this way, only correctly preprocessed words are passed on for recognition.

The prerequisite discussed in section 1.4.2 is thus satisfied.

3.7. Segmentation Algorithm

The segmentation algorithm consists of two parts. Firstly, specific features of the pen-strokes of the script define *Possible Segmentation Points* (PSPs). Secondly, each segment between two PSPs is examined and PSPs may be deleted to yield a list of *Definite Segmentation Points* (DSPs). This process is detailed in Chapter 4.

3.7.1. Possible Segmentation Points

A PSP is recorded at the following points within the sample word (see figure 3.3) :-

Intersections - Wherever a stroke crosses its own path to produce a loop, two PSPs are recorded with a segment in between.

Cusps - Wherever there is a discontinuity in the slope of the stroke, two superimposed PSPs are recorded with a zero length segment in between. A cusp is thus treated as an infinitessimally small loop.

Points of Inflection - Wherever there is a sign change in the angle of curvature, a PSP is recorded.

End Points The position of a pen-up or pen-down is treated as if it was a cusp.

3.7.2. PSP Deletion

This segmentation alone is not perfect, since it is possible to have multiple intersections which overlap, and short, irrelevant segments caused by wobble of the pen or pen flicks at pen-ups and pen-downs. We wish to use only those PSPs that were intentional, and not those that are irrelevant. In figure 3.4, the important

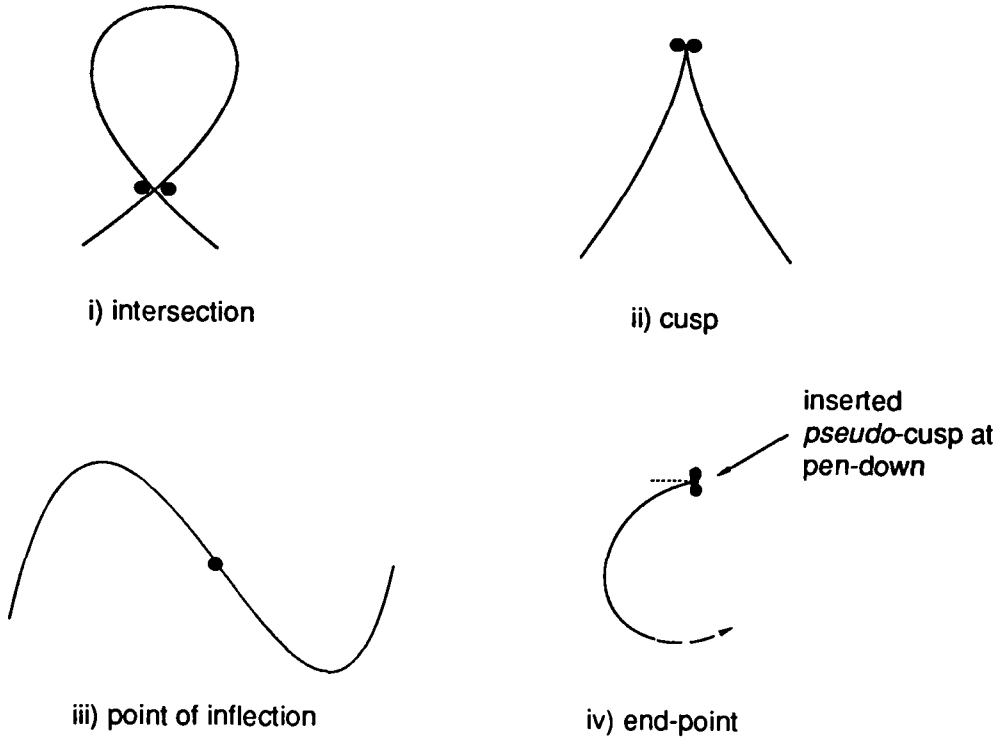


Figure 3.3 - Possible segmentation points

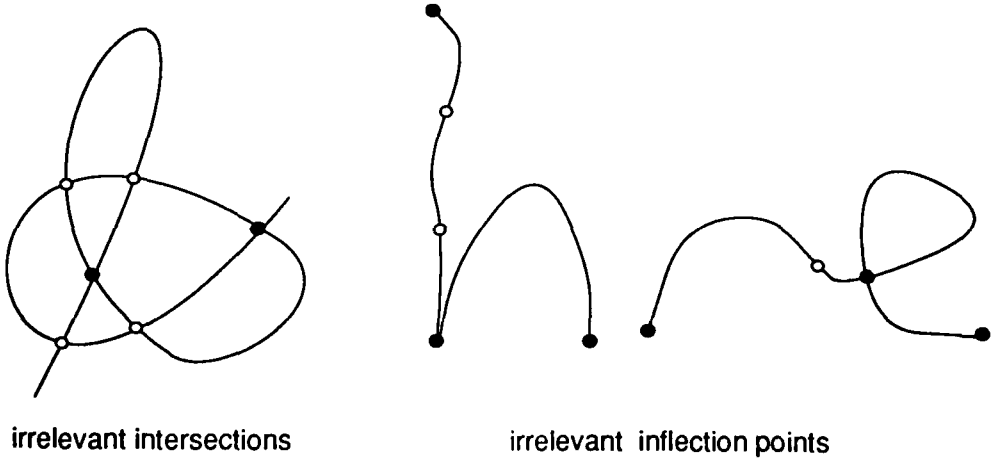


Figure 3.4 - Irrelevant PSPs

PSPs are indicated in black and the irrelevant PSPs in white.

To identify which intersections are intentional and which are accidental, every intersection is examined to see if its connecting segment contains any other intersections. If it does then the segment is repeatedly stretched horizontally with

respect to time until only uninterrupted segments are produced between the intersections (see figure 3.5). These PSPs are then added to the list of DSPs and the other accidental PSPs removed.

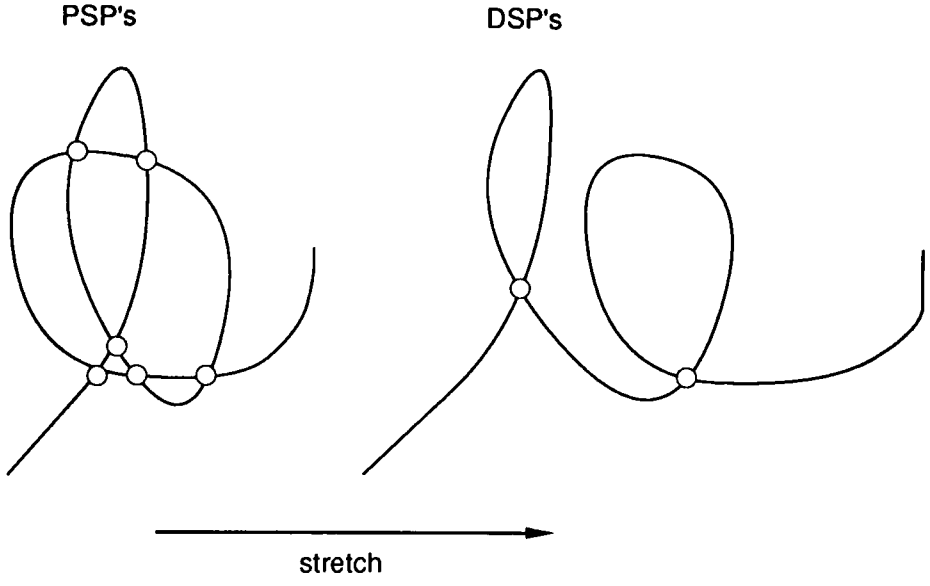


Figure 3.5 - PSP deletion

The PSPs bounding short segments are deleted from the list of DSPs if either the segment preceding or the segment following is not a cusp. Short segments immediately preceding a pen-up or following a pen-down are regarded as *serifs* or flicks of the pen and are deleted.

3.8. Feature Extraction

The segments between DSPs fall into one of three general categories, illustrated in figure 3.6, either a *loop*, a *cusp* or a *hump*. A number of descriptive features are measured for each segment. Section 4.6 discusses the selection of these features.

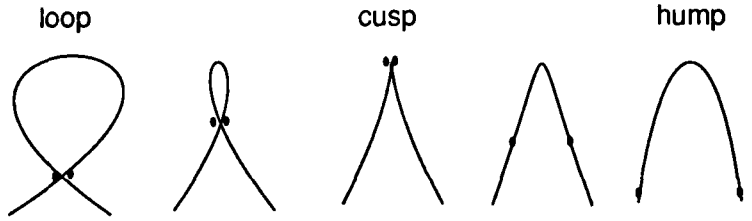


Figure 3.6 - Segment shapes

3.9. Template Matching

A database of templates is stored containing the allographs for every variety of letter and letter join recognised by the system. A further personal database for each user records only the styles which they use, and can be constantly updated to match their individual handwriting. Details of the template matching routines are found in Chapter 5.

The templates for letters and letter-joins are segmented using the same algorithm as that used to segment the sample script, in order that the sample and template segments should match in size and shape. A template can then be several segments long. For each segment of the template, the mean and standard deviation of each feature is stored. The mean and standard deviations are calculated by presenting a number of samples of the letter to the system and their feature values analysed. In this way the consistency of each of the features is represented by the standard deviation.

The template matching process consists of several stages - *template comparison, segment normalisation* and *letter graph formation.*

3.9.1. Template Comparison

Each segment of each template in the template database is compared against every segment of the sample word. For each feature of each segment of the template a *height value* is calculated by evaluating an approximation to the height of the template's statistical distribution at the sample word's feature value, see figure 3.7.

The height values for a segment are then combined together to give a *comparison score* for that segment of that template. These scores are recorded in a *template comparison array*, and this array is located at the appropriate segment position within the *sample word comparison array*. Figure 3.8 shows a sample word comparison array for a five segment sample D compared with template database containing three different length templates R, S and T. $\phi_{X_i}(D_j)$ represents the height value when segment j of the sample is compared against segment i of template X. A further description can be found in section 5.7.1.

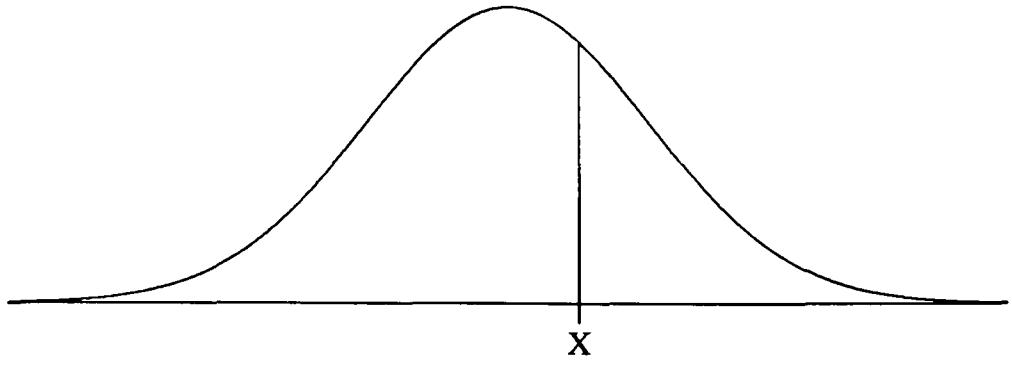


Figure 3.7 - Height value of the template distribution

Seg 1	Seg 2	Seg 3	Seg 4	Seg 5
$\phi_{R_1}(D_1)$	-	-	-	-
-	$\phi_{R_1}(D_2)$	-	-	-
-	-	$\phi_{R_1}(D_3)$	-	-
-	-	-	$\phi_{R_1}(D_4)$	-
-	-	-	-	$\phi_{R_1}(D_5)$
$\phi_{S_1}(D_1)$	$\phi_{S_2}(D_2)$	-	-	-
-	$\phi_{S_1}(D_2)$	$\phi_{S_2}(D_3)$	-	-
-	-	$\phi_{S_1}(D_3)$	$\phi_{S_2}(D_4)$	-
-	-	-	$\phi_{S_1}(D_4)$	$\phi_{S_2}(D_5)$
$\phi_{T_1}(D_1)$	$\phi_{T_2}(D_2)$	$\phi_{T_3}(D_3)$	-	-
-	$\phi_{T_1}(D_2)$	$\phi_{T_2}(D_3)$	$\phi_{T_3}(D_4)$	-
-	-	$\phi_{T_1}(D_3)$	$\phi_{T_2}(D_4)$	$\phi_{T_3}(D_5)$

Figure 3.8 - A sample word comparison array

A number of templates are rejected at this stage if the comparison score for any segment falls below a threshold.

3.9.2. Segment Normalisation

When comparison scores have been calculated for every template, the scores for each segment of the sample word comparison array are normalised across the segment to give *segment-normalised comparison scores.*

3.9.3. Letter Graph Formation

The segment-normalised comparison scores of a matching template's constituent segments are combined to give a *weight* for that template. If any template weight exceeds an experimentally derived threshold it is accepted as a *candidate allograph.* It is placed in the appropriate position on a *directed letter graph*, connected to any other candidate allographs which may precede or follow it in physical location (see figure 3.9). By tracing the letter graph, candidate words can be generated and associated *candidate word weightings* calculated by combining the weights of each template used. ('@' represents start-of-word, '#' end-of-word.)

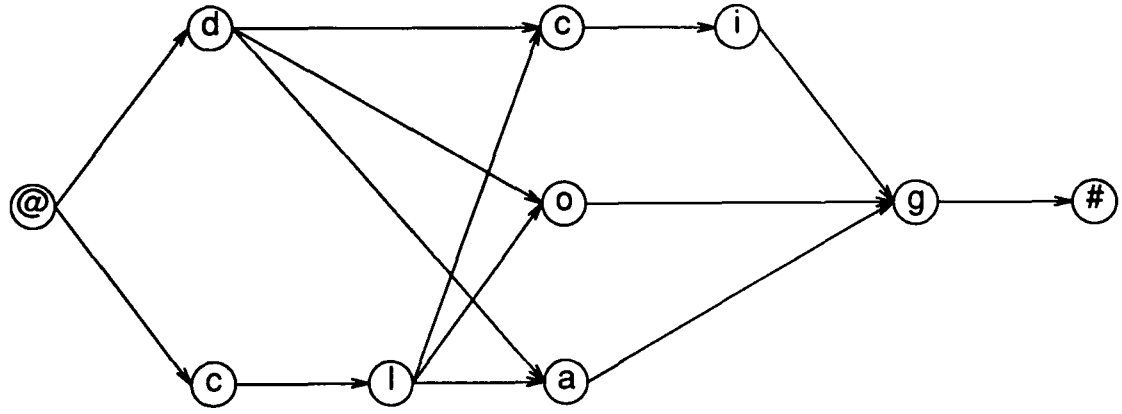


Figure 3.9 - A directed letter graph

3.10. Dictionary Lookup

A lexicon of valid words is constructed into a *trie*-based data structure (see section 2.7.7). The letter graph, produced by the template matching routine, and the dictionary tree are traced simultaneously with a highly efficient recursive algorithm that yields every possible valid word candidate that can be formed from the letter graph. Each word has a weighting associated with it allowing the output word candidate list to be ranked. Full details are given in Chapter 6.

The diacritical mark information saved from earlier is now used to reduce the output list by removing words which do not feature the correct number of marks. Even the identification of these marks is ambiguous, so it is necessary to allow the counts of dots and crosses to vary, ie all dots are allowed to be short crosses and vice versa. Allowance is also made for a double 't' to be crossed with just one stroke.

3.11. Visual Display of Input and Verification of Output

The final word candidate list can now be presented to the user for verification. A final system must be able to make an automatic decision as to which of the candidate words to propose as the correct one, or it must decide that there is nothing to choose between a small number of contenders and present all of the possibilities for the user to select from.

The word weighting produced by this system cannot be used as a guide to the quality of the output since it is a relative measure with respect to the other possible candidate words. If a number of words appear at the top of the candidate word list with equal likelihood, then their absolute weightings will be lower than a single word. This does not imply that the quality of the match is any lower. This can be judged by examining the spread of weightings of the words. If the top word, or small group of words, is separated from its nearest neighbour by a large amount, then the match is probably better than if there is a very large group of words at the top of the list. Within this system at present, all words that have probabilities greater than a percentage of the top word are displayed.

The output list can be further enhanced by including the *a priori* probability of a word being written, obtained from a frequency count of English usage.

In the experimental system the output is displayed one word at a time with a graphical display of the input data. The correct word can be selected from the output list, or the correct word entered if it is omitted from the list.

3.12. Training

Training takes place in two stages, *initialisation* and *continuous training*. Firstly, a new user is initialised into the system. The system prompts the user to write a small number of words which contain every letter. From this data a personal database of allographs is selected and created. Later, the user writes any words from the dictionary. The system attempts recognition and requests

verification or correction of each word. The personal database is then updated with the new data. This continues until sufficient recognition accuracy is attained. This is discussed in Chapter 7.

3.13. Conclusion

An experimental system has been produced which enables a user to write any number of words from a large vocabulary. Each word is recognised and presented to the user for confirmation. After confirmation, the system automatically trains the personal database of the user. Further details of the individual components of the system now follow.

Chapter 4

Segmentation

This chapter describes the segmentation method used by the ORCHiD system. Section 2.4 described the common segmentation methods and problems that are associated with them. Although any segmentation method will include similar problems, it is the goal of this work to minimise these. It is desirable for a segmentation method to produce a consistent set of segments, especially for input samples with the same underlying structure. By studying the way that minor changes in letter formation affect the image written on the paper, a segmentation method has been conceived which appears to achieve this consistency and minimises the associated problems.

4.1. Background to the Segmentation Algorithm

To develop an automatic handwriting recognition system which can read most styles of handwriting with no training requires a large collection of data containing the many differing styles and variations of each character within the alphabet. (The way in which a particular character is formed is referred to as its *allograph* and its computer representation is referred to as a *template* for that character). A brief examination of a small number of manuscripts is sufficient to indicate the wide variety of styles in everyday use. It is clearly impractical, in terms of storage and computation, to record each individual style separately. It is therefore necessary to find a way to identify a particular character given a number of variations in its formation.

The most common method used in schools to teach handwriting is the *copy-book* method. A page of a typical copy-book might have a carefully drawn example of how to write a character, perhaps with numbers and arrows to indicate the order and direction in which the strokes are made, together with guidelines on which the pupil can copy the letter. Figure 4.1 shows a typical page as might be found in a copy-book[85]. Children then diligently copy each letter repeatedly until they have mastered how to form the shapes correctly. The position, direction and

order of the pen strokes is well specified.

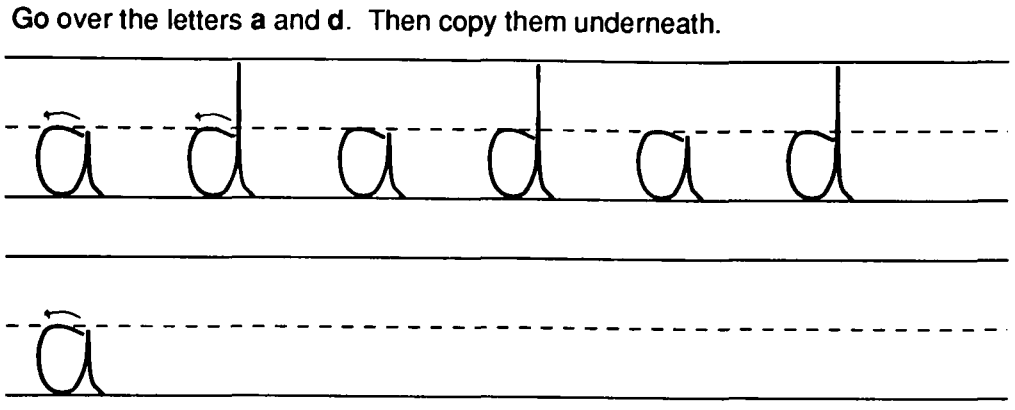


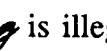
Figure 4.1 - A typical page from a copy-book

As we grow older and need to write increasing amounts of text more quickly (and, indeed, as we get more lazy), the near-perfect formation of schoolbook handwriting degenerates. The precise manner in which each individual letter and letter combination is formed during this period of deterioration is unique to each individual and this gives rise to a personal style. Often the quality of the handwriting continues to deteriorate until a stage is reached when the script is illegible to anyone with whom the writer needs to communicate. At this stage the writer is forced to check the decline in standards, if the writing is to be understood by anybody else, or learn to type!

There are only a limited number of copy-book styles from which everyone has learnt to write, and most of these styles overlap considerably across different copy-books. If we can somehow map the *graphical representation* of the writing which appears on the page to the *theoretical representation* of the copy-book style that the writer learnt, we can greatly reduce the number of templates which need to be stored to just the number of different copy-book styles.

This is perhaps not a completely impossible task, since everyone has a sub-conscious image of the letters they are trying to form as they write. An informal indication that this is true is to ask anyone to write a word as they were originally taught, or as they might teach a child to write. Most people will have little difficulty with this task, and it is, in fact, a very common occurrence with parents helping children to learn to write.

In order to verify this a number of people were asked to write a few words in their ordinary handwriting, and then in a very neat style, as they were taught at school or as they might teach a child to write. With very few exceptions the "standard" copy-book style templates were used, even by those people who had very unusual writing styles.

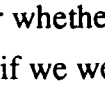
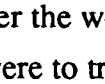
In general, the accuracy of letter formation compared to the copy-book standard decreases as writing speed increases. The major feature of the script which is affected by speed of writing is the *cusp* or sharp corner. This feature either becomes smoother until it is just a wiggle or *hump*, or it becomes more pronounced and turns into a loop. These processes also occur in reverse whereby a loop or hump becomes a cusp. Figure 4.2a shows how this occurs and figure 4.2b shows this deterioration within the context of letters. As the writing deteriorates further, whole letters can disappear and become just squiggles or lines which, taken out of the context of the word, are completely meaningless. In figure 4.2c, the end of the word  is illegible when the letters  are taken out of context. There is a limit to the quality of handwriting that is legible to even a human reader, so it is not unreasonable to insist that an automated recognition system must be presented with reasonably neat handwriting that is relatively easy for a human to recognise. Hence the prerequisite of section 1.4.2.

We can make use of the above information to develop a segmentation method that is consistent across a number of writing styles where there are minor variations in the graphical formation of allographs.

4.2. Aims for a Good Segmentation Method

We can now define some requirements for an ideal segmentation method for such a cursive script recognition system.

a) *The method should segment the script at least at letter boundaries, ie there should be no more than one actual letter between two adjacent segmentation points.*

It is clearly not possible to insist that segmentation should occur *only* at letter boundaries, which would make the subsequent recognition far simpler, since the letter boundaries may be ambiguous even after the character recognition stage. For example it may not be clear whether the word  or  was written. (Note: This may not be the case if we were to treat letter combinations as a single unit for recognition, for example it may be desirable to

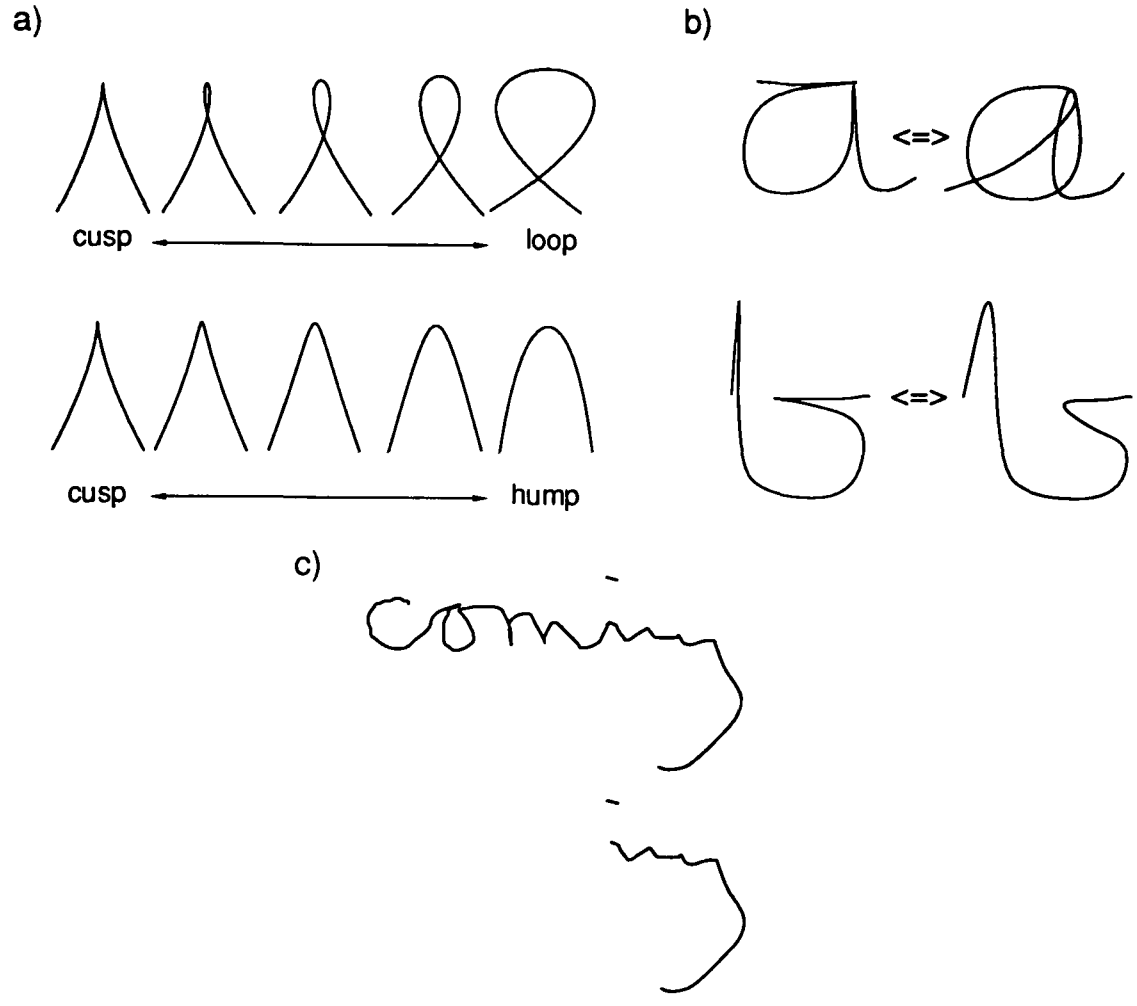


Figure 4.2 - Cusp deformation

recognise *ing* or *th* as single entities. A better description of this requirement might be to say that the algorithm should segment at least at "recognitional unit" boundaries. The ORCHiD system does not in fact use multi-character templates, but it would be simple to define and match such templates if so desired.)

This requirement ensures that the template matching stage only involves comparison of the segment with each template in turn in isolation. It is never necessary to consider two templates in combination to match to a single segment.

b) *The segmentation algorithm should base its decision on whether to segment at a particular Digitiser Data Point (DDP) independently of the other Segmentation Points (SP).*

In this way the algorithm will define if a DDP may be a SP or may not. There is no allowance for the position of a SP to be ambiguous. (For example, in figure 4.3 the segmentation algorithm should be of the form "If there exists a segmentation point between X and Y then it shall be at Z" rather than of the form "If there exists a segmentation point between X and Y then it is *either* at A *or* at B, but not both".)

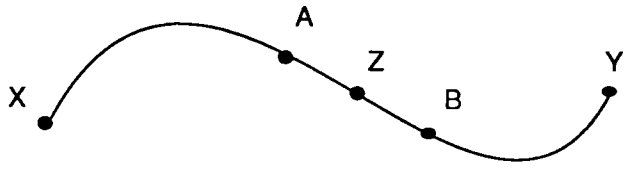


Figure 4.3 - Segmentation points with variable position

This requirement allows the ambiguity in segmentation to be carried forward for resolution by the later stages of the recognition system. (For example, it may be desirable to correctly identify the letter segmentation and this could be more successfully done after the template matching stage.) The ambiguity of the precise location of a SP cannot be carried forward using a statistical system since there is no valid theory known to the author for comparing segments of differing lengths. For example, in the diagram, if there is doubt as to whether the segmentation of the stroke XY should be at points A or B there is no statistically valid way of comparing the segmented pieces XA to XB, or AY to BY. (See section 5.2 for a discussion on the statistical validity.)

c) *The segmentation should emphasise the important characteristics of the handwriting which help to differentiate between the letters, and suppress the unimportant characteristics.*

d) *The segmentation should aim to consistently map variations of the same character onto a single stored letter template.*

In this way the number of letter templates that need to be stored will be reduced. In the terms discussed in section 4.1, we wish to map each written character back to the writer's mental image of the copy-book character.

4.3. The ORCHiD Segmentation Algorithm

The algorithm attempts to highlight the deformation of cusps described in section 4.1 above, since the graphical appearance of the cusp on the page is one of the main features of individuality between writers. For example, the copy-book letter *d* may be as in the right of figure 4.4. Each of the other *d*s are based on it in some way. If the copy-book *d* is described in words as

> "a cusp, pointing right, on the halfline, joined to an upward pointing cusp at the full-height-line, directly above the first cusp, by a smooth curve which touches the baseline,"

and the word *cusp* is freely interchanged with *loop*, then the similarity emerges.

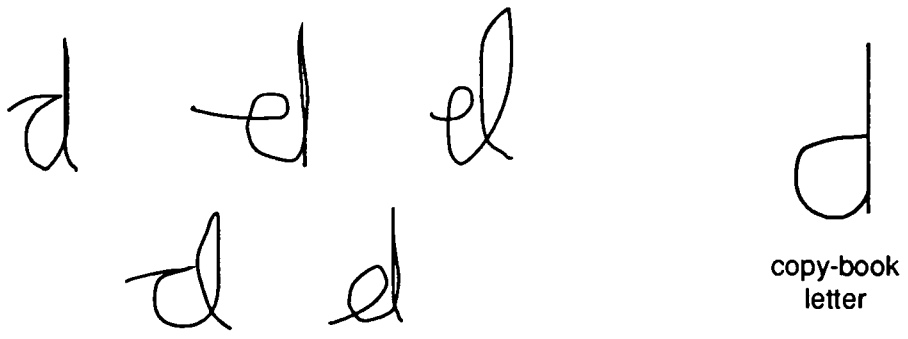


Figure 4.4 - Letter d's based on the same copy-book letter

Ouladj *et al*[71] have taken a similar theoretical basis for their recognition system. Their description of the script, however, is based on Freeman coding, so it is necessary to apply an additional information source that describes the changes to the Freeman code as cusps, loops, etc deform (see figure 2.11). The approach described below provides a measure of deformation of a cusp, loop, etc and so allows direct comparison between the different types of segment.

The ORCHiD system is primarily a statistically based recognition system. It is therefore important at each stage to consider the statistical validity of that stage and its repercussions on subsequent stages. To compare the goodness of fit of a sample with a number of templates it is essential that the templates have the same number of features as the sample, and that the feature arrays are of the same length otherwise we are comparing variables with differing numbers of *dimensions*. For this reason, item b) above is most important.

The segmentation algorithm operates in two passes. On the first pass, a list of *Possible Segmentation Points* (PSPs) is produced which are defined by certain features of the script. This list is then reduced on the second pass by the application of a number of criteria to provide a list of *Definite Segmentation Points* (DSPs).

4.3.1. Possible Segmentation Points

A PSP is recorded at the DDP nearest to specific features of the script. These four features are as follows and illustrated in figure 4.5.

i) **Intersection** - Wherever a stroke crosses its own path to form a closed loop this is defined as an intersection. A PSP is marked at the DDP nearest to the intersection at the beginning of the loop and nearest to the intersection at the end of the loop.

ii) **Cusp** - Wherever there is a discontinuity in the slope of the stroke, ie where there is a sharp point, this is defined as a cusp. A cusp is treated as if it were an intersection with an infinitessimally small loop, and so has two superimposed PSPs with a zero length segment in between.

iii) **Point of Inflection** - Wherever there is a sign change in the angle of curvature, this is defined as a point of inflection. A PSP is recorded at each point of inflection.

iv) **End Points** - Each pen-down or pen-up is replaced by a *pseudo*-cusp with the corresponding two PSPs and zero-length segment.

4.3.2. PSP Deletion

This initial segmentation works well for carefully drawn words, but often fails in practice due to two features of everyday handwriting.

Firstly it is common for a number of loops to be drawn that overlap each other within a word. This causes a large number of *superfluous* intersections and loops to be identified which confuse the system.

Secondly straight lines within the script often wobble by more than the threshold set by the cleaning routines during preprocessing and thus produce *irrelevant* points of inflection with short segments between them. In figure 4.6, the important PSPs are indicated in black and the irrelevant PSPs in white. Methods are needed to alleviate these problems.

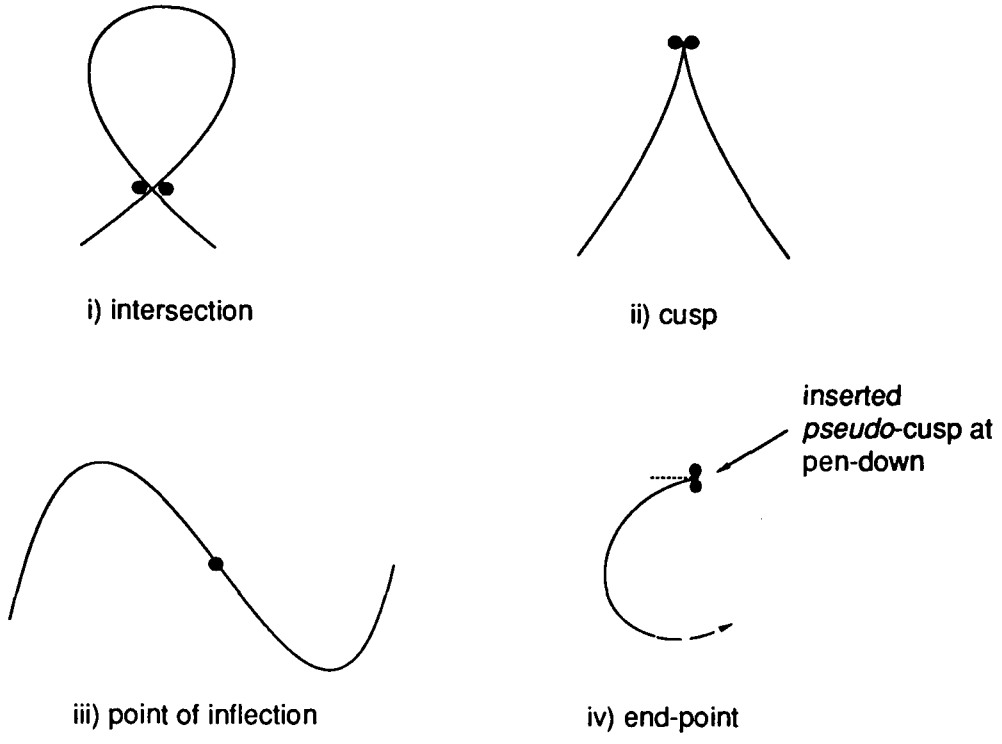


Figure 4.5 - Possible segmentation points

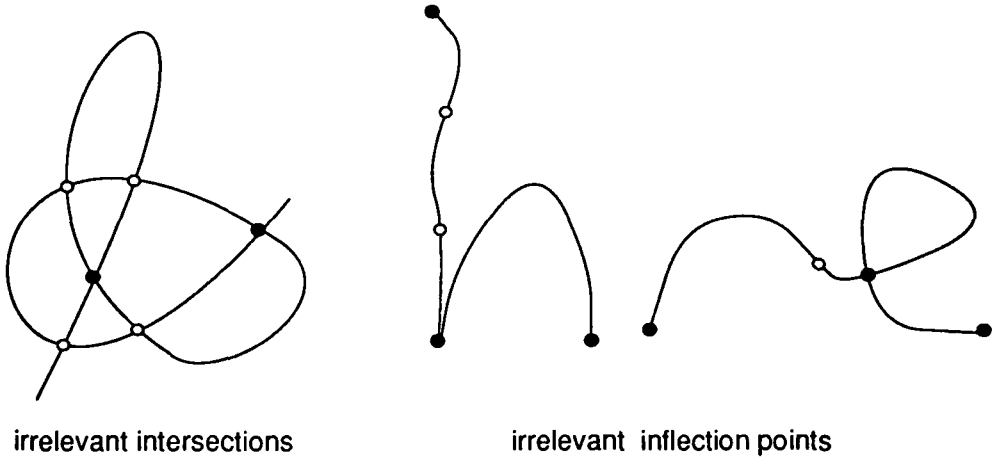


Figure 4.6 - Irrelevant segmentation points

4.3.2.1. Multiple Intersections

Consider the stroke sample between X and Y in figure 4.7a (perhaps a stylised letter 'b'). We can see that there are six intersections (A - F) between X and Y. However if we consider the first half of the stroke, between X and Z, there is

just one intersection at B (figure 4.7b). Similarly for the second half of the stroke, between Z and Y, there is just one intersection at F (figure 4.7c). This matches up with our theoretical view of a copy-book letter 'b' which consists of two cusps connected by a smooth curve (figure 4.8a).

It can now be seen that the stroke XY consists of the curve XB, the loop delimited by the intersection at B, the curve BZF, the loop delimited by the intersection at F, and the curve FY. The intersections at A, C, D and E are incidental and are not relevant in defining the constituent elements of the stroke. Figure 4.8b shows a stroke with the same underlying structure without the accidental intersections. We require a computational method for highlighting the relevant intersections within a piece of script.

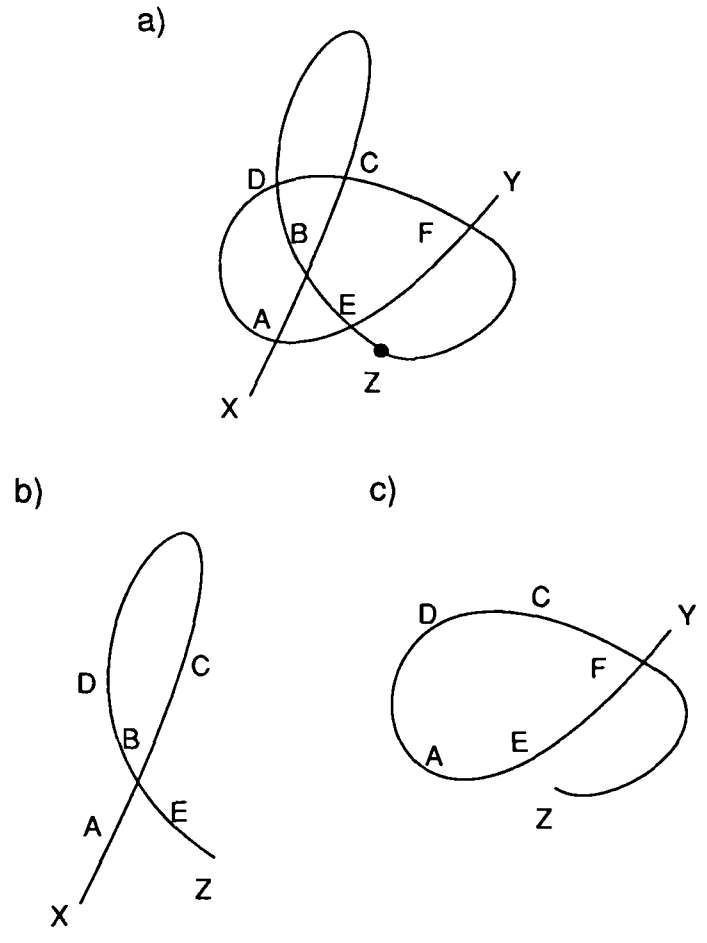


Figure 4.7 - Multiple intersections

A very simplified view of the motion of the pen during cursive writing is to consider that the fingers move the pen by small amounts in the horizontal and

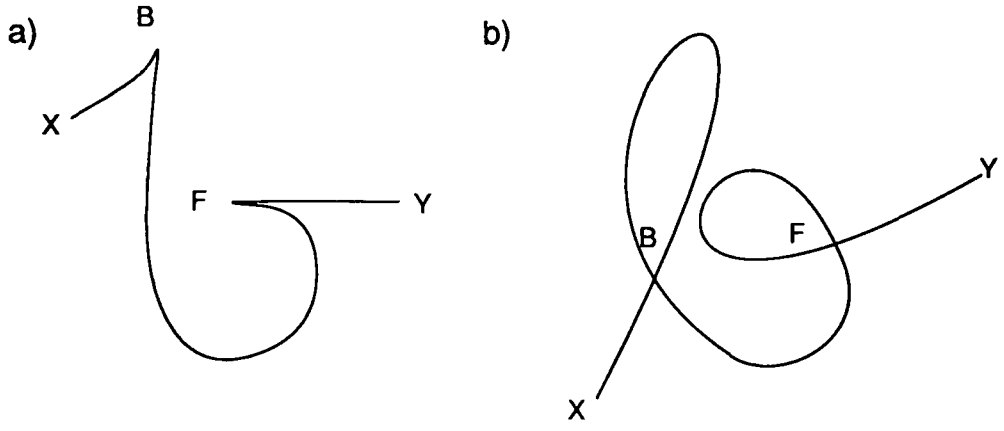


Figure 4.8 - Strokes with same underlying structure

vertical directions to form the characters, perhaps as they might form separated characters, and the whole hand, wrist and arm move slowly from left to right as the writing progresses[17,34,96]. Multiple intersections within a stroke are caused when the horizontal motion of the writing hand is fairly slow, so that the script overlaps itself. If the hand were moving faster in the horizontal direction, or the paper were moving from right to left, the strokes would not overlap. If we superimpose a time-motion in the horizontal or x direction, we can simulate this increased speed of hand movement. So if a stroke with multiple intersections is stretched along the x-axis with respect to time, by adding an increasing amount to the x coordinate of each DDP, the irrelevant intersections disappear leaving just the important intersections. Figure 4.9 shows an example being stretched by increasing amounts. This can perhaps be visualised by considering the writing as a piece of string so that the two ends could be pulled apart until there are no overlaps.

In the second pass of the segmentation algorithm, each segment that is delimited by an intersection, ie the loop part, is examined to see if any other intersections occur within it. If so, the local area around the intersection is incrementally stretched along the x-axis until no loops overlap. The remaining intersections are then allowed as Definite Segmentation Points.

The practical implementation also insists that no other PSP can exist within a loop, so there is a priority ranking for these PSPs. If an inflection point occurs within a loop, then the loop takes priority and the PSP at the inflection is deleted. If a cusp occurs within a loop, then the angle made by the entry and exit lines of

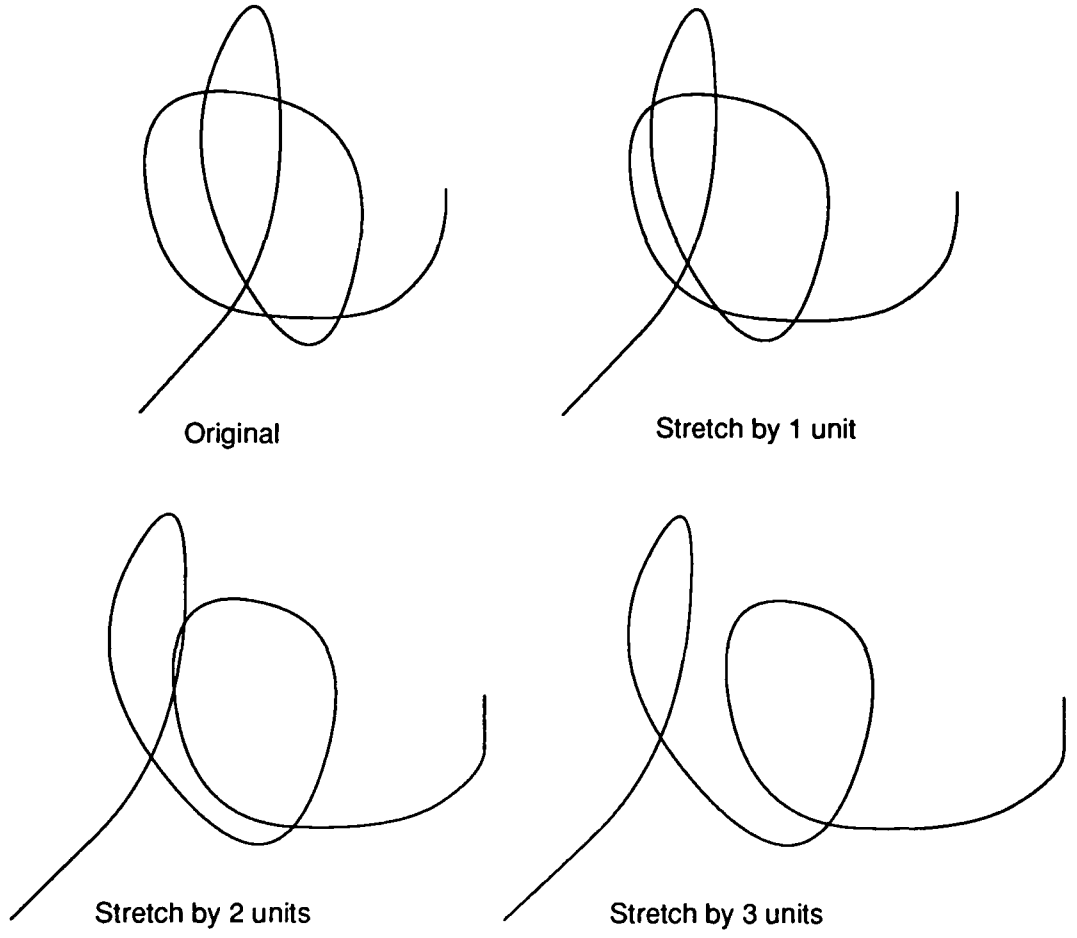


Figure 4.9 - Stretching process

the cusp (or its "tightness") is examined and compared with a threshold. A cusp that is sharper than the threshold takes precedence over the loop intersection, and a cusp that is less sharp than the threshold gives precedence to the loop intersection. The inappropriate PSP is then deleted.

4.3.2.2. Short Segments

Some of the (non-zero length) segments produced by the segmentation algorithm are very short, see examples in figure 4.6. Those segments whose length is below a threshold are marked as short during the first pass of the segmentation algorithm. If either the segment directly preceding or the segment directly following the short segment is not a loop or a cusp, then the segment is considered to be a wobble and the PSP at the point of inflection which defines the segment is deleted. The segment is then effectively added on to the preceding or following segment.

4.3.2.3. Serifs

Occasionally short segments occur just before a pen-up or immediately following a pen-down. These are referred to as *serifs* or flicks of the pen. They may be caused by inadequacies of the digitiser or may be deliberate strokes or flourishes within the user's style. They are usually not important for the letter formation and so are deleted by the system.

4.4. Segment Shapes

It can now be seen that the segments between the definite segmentation points fit into one of three categories (see figure 4.10) - either a closed *loop*, where the bounding DSPs occupy the same physical coordinates, but the connecting stroke is non-zero in length; or a *cusp*, where the bounding DSPs occupy the same physical coordinates, but the connecting stroke is zero in length; or a *hump*, where the bounding DSPs are separated.

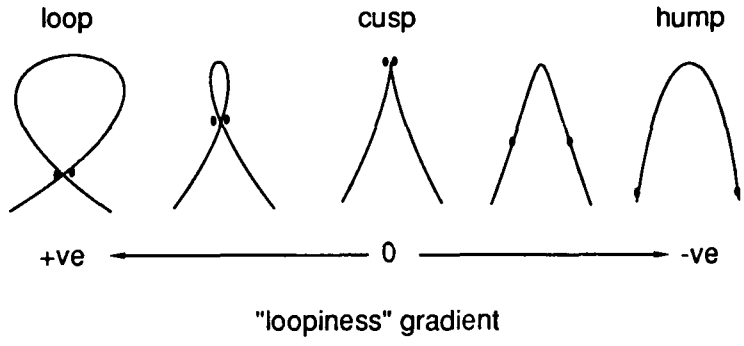


Figure 4.10 - Segment shapes

It can be seen from the diagram that these segments can be treated as the same shape but with differing amounts of "loopiness" and that one shape can evolve into another shape by either stretching or compressing the coordinates along the x-axis with respect to time, as in the technique used to remove multiple intersections described in section 4.3.2.1. By using this method of stretching or compressing along the x-axis locally with respect to time it is possible to adjust the graphical representation of a word to resemble its theoretical representation as required in section 4.1.

4.5. Suitability of the Segmentation Method

The segmentation algorithm reasonably satisfies the aims for a good segmentation outlined in section 4.2. Figure 4.11 shows a sample of the template database yielded when this segmentation is applied. The heavy lines indicate the segments, the light lines show connecting strokes.

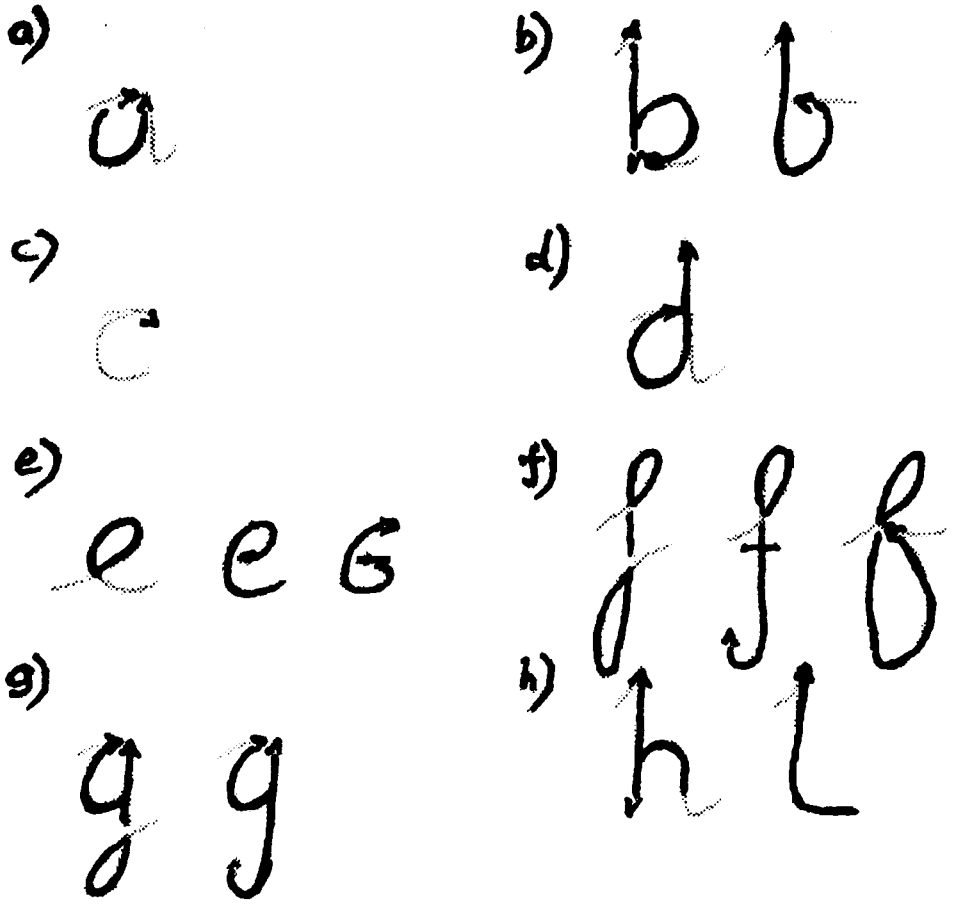


Figure 4.11 - Sample of the template database

Item 4.2 a) is satisfied, in that it is not possible to have more than one character between two segmentation points. Some letters with poorly defined beginnings and endings, eg v, w, etc, have to be carefully defined however to maximise the mapping of different styles to the same template. For example, some writers draw a letter v with an upward cusp at the beginning and end of the letter. Others draw the letter with a smooth curve in and out, while others use some

combination of these two styles. Since the middle part of the letter is the same in each case, we define this as the user-independent letter  and define special letter joins which will be user-dependent. Some letter combinations may not require a letter-join between them. A database of permissible letter/letter-join combinations is thus required.

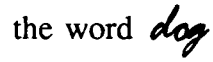
Item 4.2 b) is clearly satisfied since this was the most important consideration when developing the algorithm.

Item 4.2 c) appears to be well satisfied since key characteristics were chosen to define the segmentation points.

Item 4.2 d) is reasonably well satisfied. The segmentation produces consistent segments for well-written, correctly preprocessed data from individual users and across a number of users. This reduces the number of templates needed for each letter considerably.

A *pseudo*-cusp is inserted at each pen-up and pen-down in order to assist with the mapping of different styles to a single template. For example, if a letter  is written with a pen-down at the start of the character, insertion of a cusp will make the letter look like a letter written in the middle of a word.

4.5.1. Consistency of Segmentation

To test the consistency of the segmentation, the word  was written six times by ten writers and the script segmented using the algorithm. The words were collected in two sets of three, at the end of more substantial data collections to ensure that no extra care was taken over their formation. Each letter or letter-join was then identified to compare the segmentation. Figure 4.12 shows the different segmentations of the letters and table 4.1 shows their frequencies.

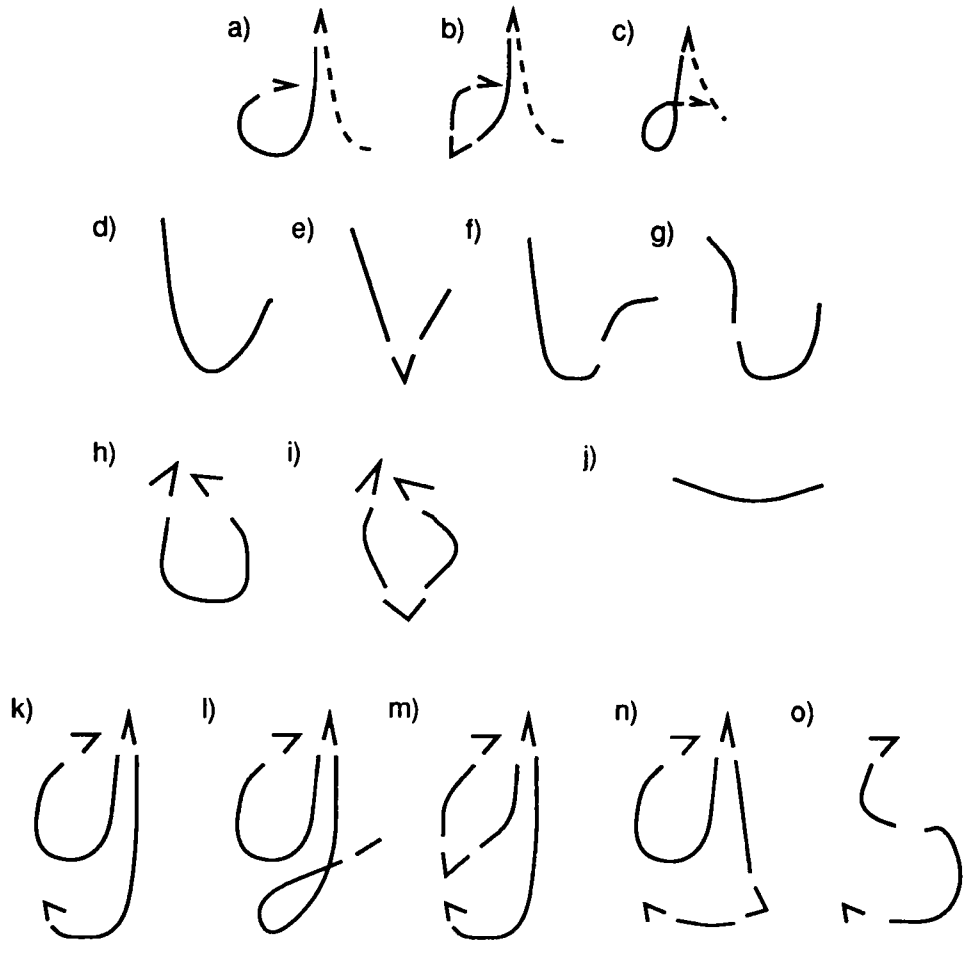


Figure 4.12 - Script segmentation

Letter/join	Figure 4.12	Frequency	%age
'd'	a)	53	88
	b)	4	7
	c)	3	5
d-o	d)	48	80
	e)	8	13
	f)	2	3
	g)	1	2
	none	1	2
'o'	h)	58	97
	i)	2	3
o-g	j)	59	98
	none	1	2
'g'	k)	41	68
	l)	11	18
	m)	5	8
	n)	2	3
	o)	1	2

Table 4.1 - Consistency of segmentation

Discussion

The main inconsistency occurs when a hump becomes thin and crosses the threshold of the cusp detector. The segments then appear to be totally different. For example, a letter 'a' can consist of three segments, but if the hump becomes thin then it consists of five segments (figure 4.13). This can be observed in figures 4.12a)/b), 4.12h)/i) and 4.12k)/m), and in reverse in figure 4.12k)/o).

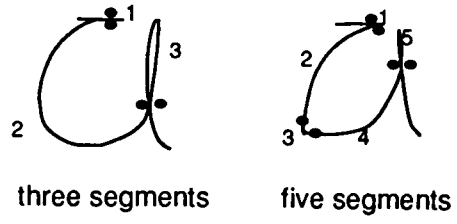


Figure 4.13 - Segmentation inconsistencies

This inconsistency could be reduced if, perhaps, the cusp detection threshold is adjusted for each user. To overcome this problem the current system allows these as different templates for the same letter, and this has proved desirable in a number of other cases, for example rounded 'v's and 'w's compared to pointed letters.

Some inconsistency occurs when script having a large amount of wobble is presented to the system. These segments are generally short, however, and are usually removed by the algorithm described above. An example that has not been removed can be observed in figure 4.12n). The two occurrences of this segmentation, however, occurred in the same sample from the same writer. Further examples from this writer may show that this segmentation is, in fact, consistent for this writer.

The segmentation appears to break down when attempting to resolve multiple intersections at the start or end of a stroke. Figure 4.14 gives some examples. In these cases the wrong intersections will be identified for deletion. Figure 4.12c) shows where this has occurred. An improved implementation of the stretching strategy might resolve this problem, for example working from right to left, rather than left to right.

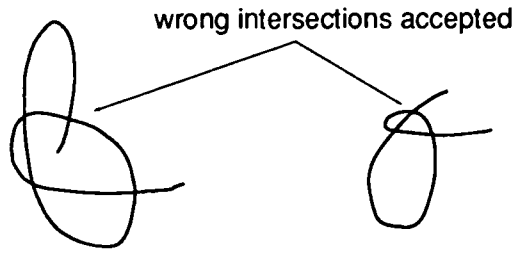


Figure 4.14 - Segmentation errors

Clearly, the segmentations for the letter 'g' in figures 4.12l) and 4.12k) are different forms of the letter. Separating these segmentations show that 4.12l) is 100% consistent, and 4.12k) is 84% consistent.

It can be seen that the letters are more consistently segmented than the letter-joins. This is probably due to more widely varying letter-join styles between writers, and within a single writer's script.

4.6. Feature Extraction

The coordinate data for each segment must now be reduced to a more manageable list of features. This list of features must adequately describe the segment so that no information is lost that might be necessary for the recognition process. By visually comparing the different graphical forms which the segment might take it can be seen that the following features may be useful. These are first described generally and then followed by details of those features extracted by the ORCHiD system.

The *position* of the segment - vertically with respect to the baseline and halfline, horizontally with respect to the neighbouring segments in some way. The vertical position is best measured at the point furthest from the bounding segmentation points, eg the top of a loop or at a cusp, as this is the least variable position across the different segment types. The horizontal position is more difficult to define in a consistent way.

The *shape* enclosed by the segment. This describes whether the segment is round, flattened or elongated. This can be represented numerically by measuring the *length* and *breadth* of the segment and made size independent by dividing the two to get the *aspect ratio*.

The *size* of the region enclosed by the segment. This can be represented by measuring its area.

The *direction* in which the segment is pointing.

The *"loopiness"* or amount of *stretching* to give the segment's position gradient in figure 4.10.

The *direction of rotation* in which the stroke was written - clockwise or anticlockwise.

With these points in mind, the following features are measured for each segment, each scaled with respect to the baseline-halfline separation. Figure 4.15

shows these features for each of the different possible segment shapes.

Vertical Position (*M*) The vertical position of the midpoint of the segment.

Angle (ϕ) The direction in which the segment is pointing. For a cusp, this is the direction in which the cusp is pointing. For a loop, it is the angle of the line joining the midpoint to the intersection. For a hump, it is the angle of the line joining the midpoint to the point halfway along the chord joining the segmentation points which define the segment.

Entry Angle ($\oplus$) The direction in which the pen was travelling on entry to the segment.

Exit Angle ($\ominus$) The direction in which the pen was travelling on exit from the segment.

Area The area enclosed by the segment. For a cusp, this value is zero. For a loop, this is the area enclosed by the loop. For a hump, this is the area enclosed by the segment and the chord joining the segmentation points which define the segment. The area is signed depending on the direction of the enclosing stroke, clockwise or anticlockwise.

Aspect Ratio (*t*/*d*) The ratio t/d, for the depth d and thickness t of the segment. For a cusp, this is defined to be unity (ie a perfect circle). For a straight line, this is defined to be a predefined large number (ie approaching ∞).

Chord Distance (c) The distance between the segmentation points which define the segment. This will be zero for cusps and loops.

Horizontal Displacement The horizontal distance between the midpoint of this segment and the previous but one segment. This was found by observation to be the most useful horizontal position information that could be easily measured.

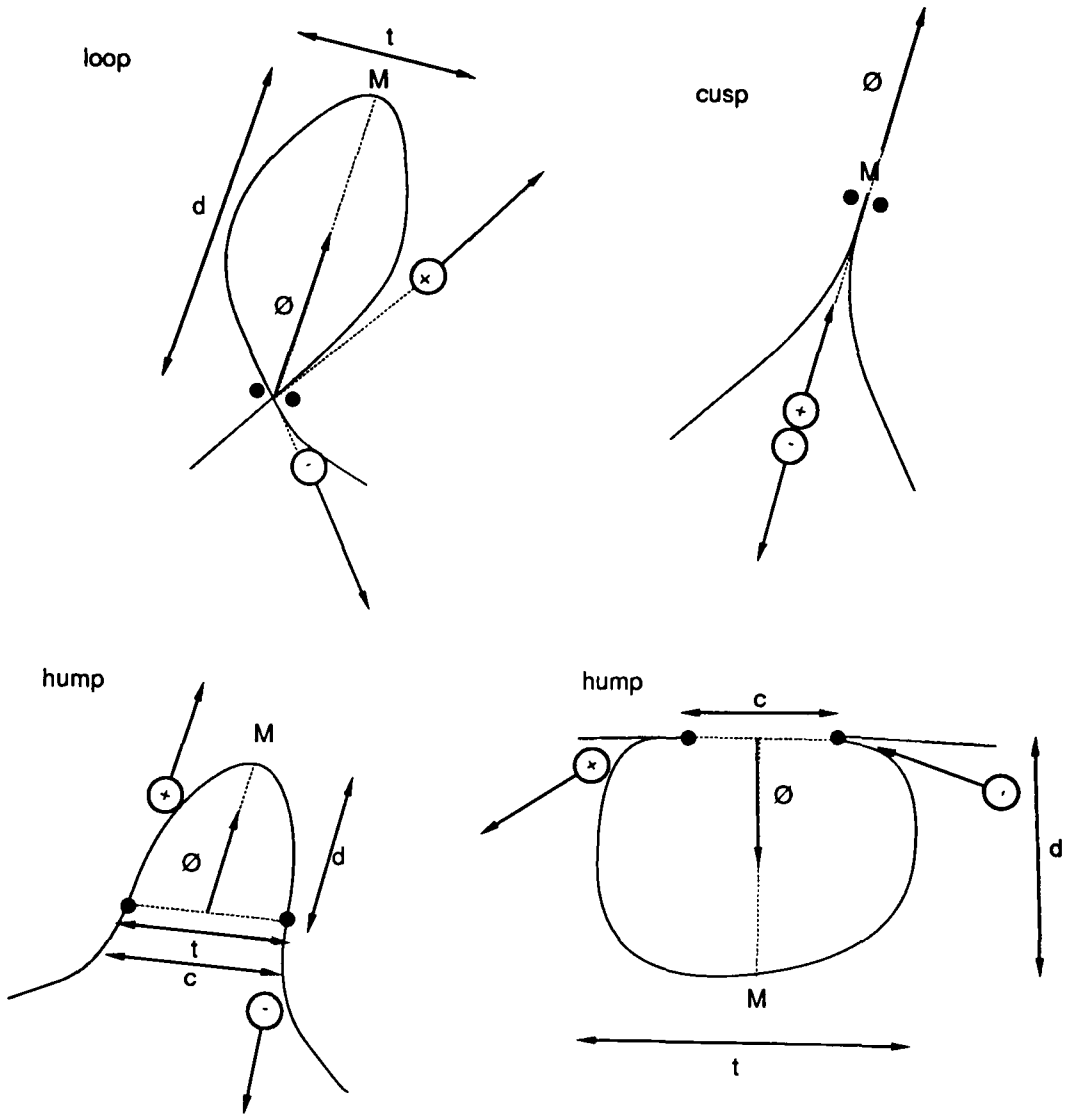


Figure 4.15 - Features measured

Figure 4.16 illustrates the variation of some of these features along the "loopiness" gradient.

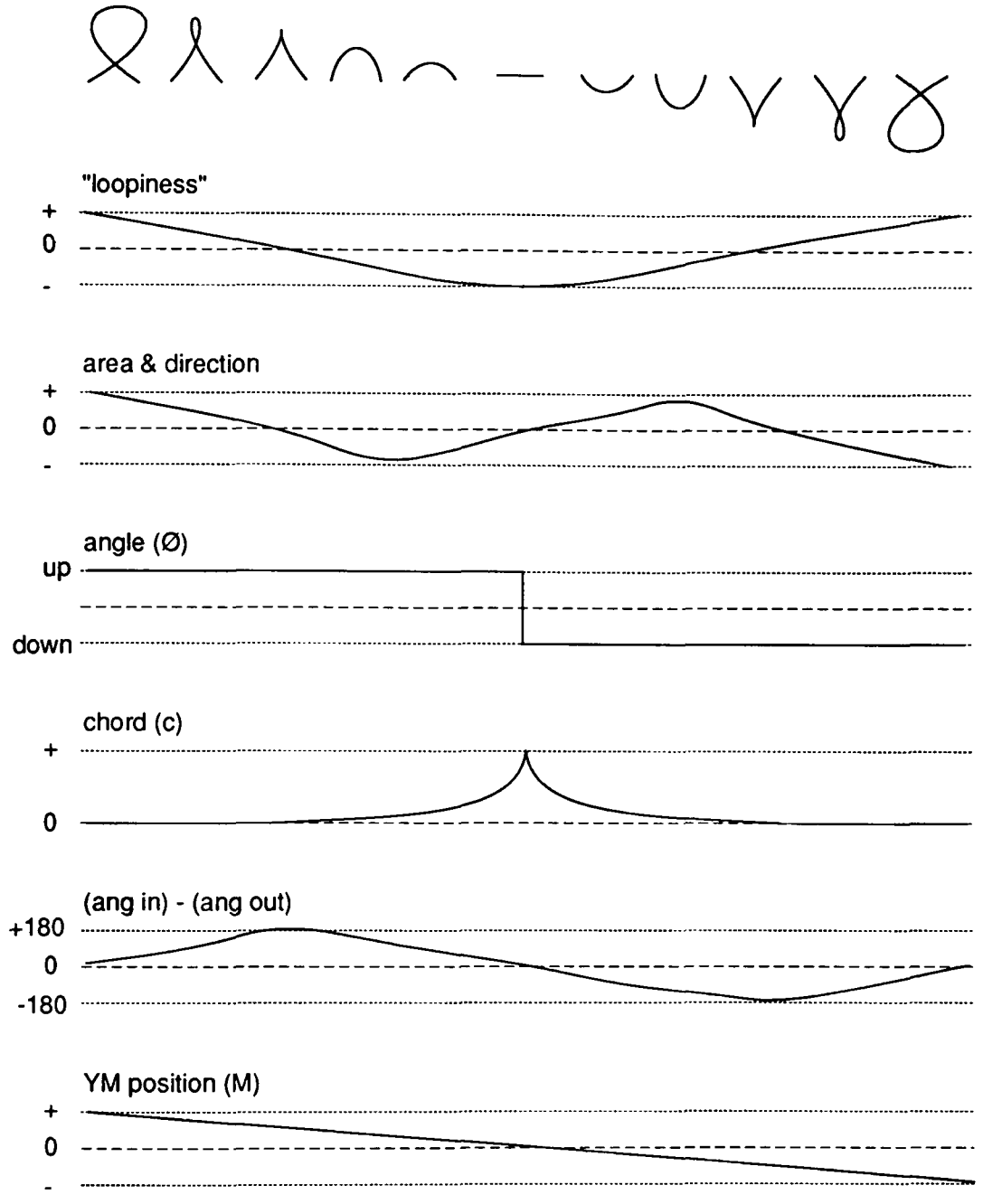


Figure 4.16 - Values of features

4.7. Independence and Normality of Features

The statistical template matching method described in Chapter 5 assumes that the features described above are normally distributed and are independent of each other. Results are given below for tests of normality for a number of these

features.

Clearly the features are not independent of each other. For example, the direction the segment is pointing in will be dependent on the angles of entry and exit to the segment. The area of the segment will be zero if the height and width are zero, and so on. It has been found by experiment, however, that all of the features add to the information about the segment, and help to differentiate in the template matching process. The assumption of independence is therefore used to reduce the complexity of computation of the multivariate normal distribution (see section 5.6).

There are a number of problems associated with proving normality of the feature values. The major problem is the large number of samples needed to achieve a significant statistical proof. For this reason, a non-parametric test was chosen to show normality, since the restrictions are generally less strict than for more formal methods. However these tests are consequently more conservative. The other problem occurs since the feature measurements are on a discrete scale so that the implementation can use integer arithmetic for efficiency. To compare this with a continuous normal distribution means that the test will again be conservative. Despite these restrictions, a number of features have shown reasonable supportive evidence that they are normally distributed.

The Kolmogorov-Smirnov test for normality has been used. This test involves comparing the cumulative distribution of the sample with the expected cumulative distribution function (cdf). The test statistic is the maximum distance between the two cdf's and this must be less than a tabulated value[67].

As an example, 32 samples of the loop of a letter 'e' were examined and the angle recorded (table 4.2).

Angle	9	24	40	41	43	45	48	50	51	52	55
Freq	1	2	1	1	1	1	1	2	1	1	3
Angle	59	60	61	63	66	68	69	71	76	79	88
Freq	2	2	1	1	1	1	5	1	1	1	1

Table 4.2 - Observed frequencies

The mean and standard deviation are estimated from these values as 56.2 and 16.7, respectively. The expected cumulative frequency is then calculated from tables to give the observed and expected frequencies in table 4.3.

Angle	9	24	40	41	43	45	48	50	51	52	55
Exp	0.1	0.9	5.3	5.8	7.0	8.1	10.0	11.4	12.1	12.8	15.1
Obs	1	3	4	5	6	7	8	10	11	12	15
Angle	59	60	61	63	66	68	69	71	76	79	88
Exp	18.2	18.9	19.7	21.1	23.1	24.4	24.9	26.0	28.3	29.3	31.3
Obs	17	19	20	21	22	23	28	29	30	31	32

Table 4.3 - Cumulative distribution values

The maximum distance between these functions can either be seen from the table, or by plotting the graphs (figure 4.17). This occurs when the angle is slightly less than 51 and is equal to 11.4 - 8 = 3.4. The test statistic D is equal to this value, scaled by the number of samples (32) = 0.1063. The 10% significant value for this test statistic is 0.1416, so there is no evidence to reject the hypothesis that the sample comes from a normal distribution.

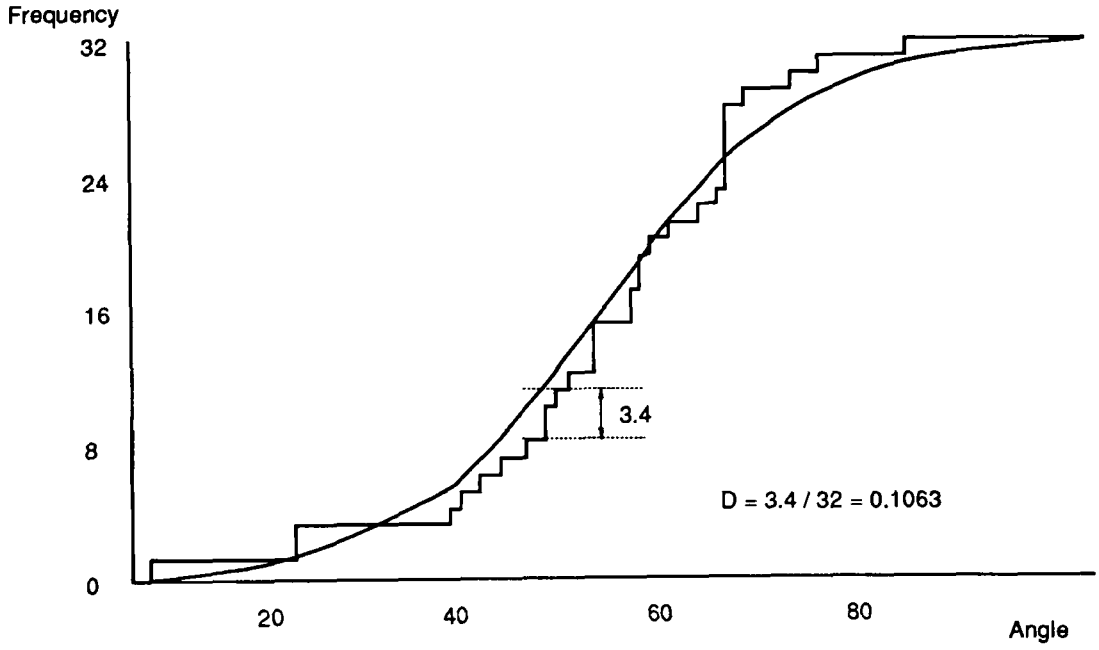


Figure 4.17 - Cumulative distribution functions

Table 4.4 shows the results from a number of tests. The "Templates" column shows which script letters the features were selected from. More than one letter was used to get sufficient samples, with the assumption that the segment being examined would be consistent across the different letters.

Feature	Templates sampled	Segment type	No. of samples	Reject level
Horiz disp	'e'	loop	32	5%
Vert pos	'b','d','h','k','l','t'	top	60	1.5%
Vert pos	'e'	loop	32	<1%
Angle	'e'	loop	32	>10%
Area	'a','g','o','q'	loop	39	>10%
Aspect ratio	'a','g','o','q'	loop	39	<1%
Chord dist	'a','g','o','q'	loop	39	2%

Table 4.4 - Normality test results

As can be seen from these results, some of the features (area, angle) appear to be normally distributed, whereas there is less evidence for the other features. The accepted features, however, take a wider range of values and so provide for a better test, as described above. Since most of the other features have not been completely rejected (>1% confidence level) it seems reasonable to assume that these too may be acceptable if a larger range of values were available.

To verify this, a discrete distribution was generated by taking values of the normal distribution at the midpoint of the discrete steps of the feature measurements. For the sample of 'e' loops (rejected in table 4.4) the expected values of the vertical position were calculated for the cdf of this distribution (table 4.5). Since the two discrete distributions coincide, the test statistic is simply the greatest difference between expected and observed frequencies. In this case the statistic (1.79 / 32) is 0.0599, which is not significant (the 10% significant level is 0.1416).

Vert pos	6	7	8	9
Observed	2	9	21	32
Expected	1.52	9.21	22.79	30.48

Table 4.5 - Discrete test

4.8. Implementation Details

Correctly preprocessed individual cursive words form the input to the segmentation routines as a stream of coordinates (or DDPs). The baseline and halfline are provided by the preprocessing routines, diacritical marks, ie dots and crosses, have been separated from the script, and pen-lifts within the word have been removed.

The feature extraction routine outputs a list of features measured for each segment of the word. The segments are listed sequentially and interspersed with any diacritical marks. For each segment, the output feature list is shown in table 4.6.

Internal Variable	Description
L	Left DDP number of the segment
R	Right DDP number of the segment
XM	Horizontal position of the middle of the segment
YM	Vertical position of the middle of the segment
ANG	Direction the segment is pointing in
^IN	Angle of entry to the segment
^OUT	Angle of exit from the segment
AREA	Area enclosed by the segment
MAXD	Height of the area enclosed
THICK	Width of the area enclosed
CHORD	Distance between start and end points

Table 4.6 - Segment features

There are a number of details to note, specific to the implementation. Many of these were for coding convenience, since the feature extraction was developed in Fortran at NPL.

- The features are scaled and the script translated such that the baseline occurs at $y = 0$, the halfline at $y = 10$ and the left-most part of the word is at $x = 0$.
- The sign of L and R is used to indicate a number of conditions. If L alone is negative, this segment is a dot; if R alone is negative, this segment is a cross; if they are both negative, this is a *short segment* which may be ignored depending on the type of segments on either side of it (see section 4.3.2.2 above). A dot or cross segment merely contains positional information. The pen-down and pen-up on either side can supply other information,

if required.

- The sign of AREA indicates if the stroke was written clockwise (negative) or anticlockwise (positive).
- If XM is flagged as negative, this indicates that this segment is an inserted *pseudo*-cusp at a pen-up or pen-down. The translation during preprocessing ensures that the actual value of XM is always positive.
- If THICK is flagged as negative, this indicates that the segment might be considered as a straight line. The actual value of THICK is always positive.
- The height MAXD and width THICK are divided (MAXD/THICK) to give the aspect ratio (ASPCT) for all calculations and this ratio is stored in the template files. (In practice the value stored is this ratio, multiplied by 10 and truncated.) If both numerator and denominator are zero, then this segment is a cusp, so the aspect ratio is set to that of a perfect circle, ie unity (ASPCT = 10). If only the denominator is zero, then the aspect ratio is set to a predefined large number to represent infinity.
- A value (DISP) is stored in the template to represent the horizontal displacement of the segment from its previous but one segment.

Example

There follows an example of how a sample of the word *they* was segmented (figure 4.18), and the features thus produced (table 4.7). The columns marked with '*' are those entries that are calculated from the other table entries, as described above.

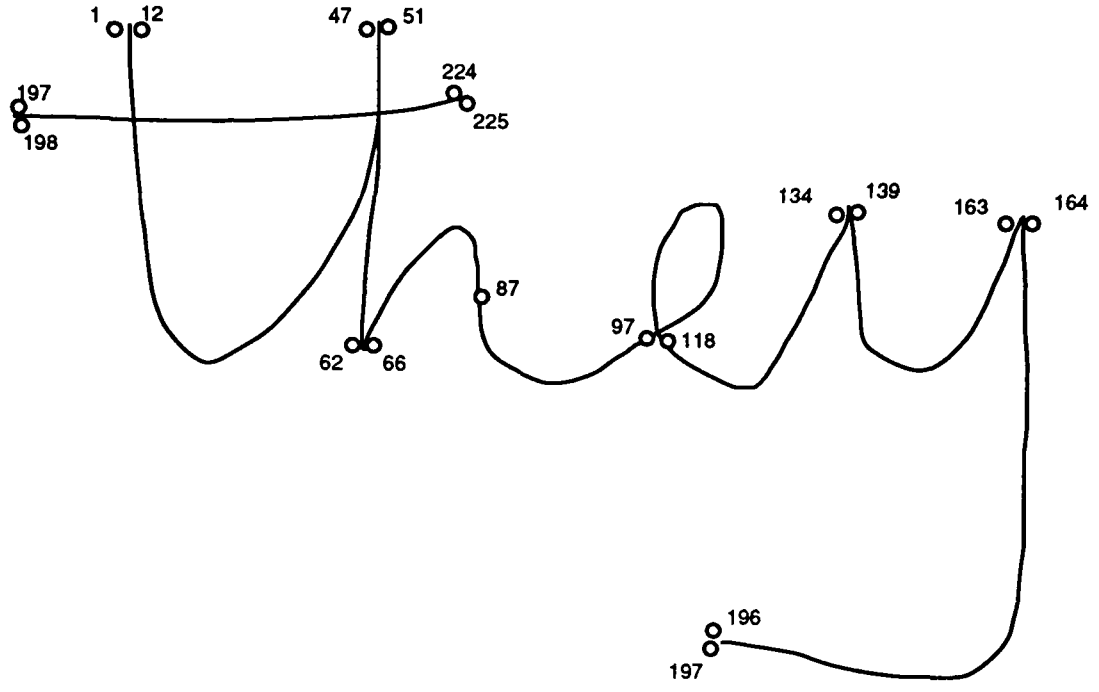


Figure 4.18 - Example segmentation

L	R	XM	DISP *	YM	ANG	^IN	^OUT	AREA	MAXD	THICK	ASPCT *	CHORD
1	2	-10	-	18	90	90	270	0	0	0	10	0
12	47	14	-	0	262	270	90	183	17	14	8	13
47	51	23	13	18	82	90	254	0	0	0	10	0
51	62	23	9	10	177	267	267	0	0	-16	250	16
62	66	22	-1	1	256	270	63	0	0	0	10	0
66	87	27	4	8	69	63	270	-24	6	4	7	5
87	97	32	10	0	243	288	27	11	1	9	90	10
97	118	41	14	9	59	27	270	21	8	3	4	0
118	134	45	4	3	319	270	90	43	4	12	30	12
134	139	48	7	9	90	90	270	0	0	0	10	0
139	163	50	5	1	252	270	104	66	8	10	13	9
163	164	57	9	8	90	90	270	0	0	0	10	0
164	196	56	6	-15	300	256	162	-200	13	25	19	28
197	197	-41	-16	-15	162	162	342	0	0	0	10	0
197	198	-1	-	13	180	180	0	0	0	0	-	0
199	-224	13	-	14	0	0	0	0	0	0	-	0
225	225	-26	-	15	9	9	189	0	0	0	-	0

Table 4.7 - Feature values for a segmented word

A brief explanation of this table appears below. The first column refers to the values of `L` and `R` for the particular segment.

`1-2` Pen-down - *pseudo*-cusp pointing upwards at `Y = 18`

12-47	Large area downward pointing hump touching the baseline
47-51	Upward cusp at Y = 18, 13 units right of the previous cusp
51-62	Straight line vertically down of length 16
62-66	Downward cusp, near to baseline, beneath previous cusp
66-87	69° upward clockwise hump approaching the halfline
87-97	243° flattened downward hump touching the baseline
97-118	60° upward pointing elongated loop approaching the halfline
118-134	319° downward pointing hump near the baseline
134-139	Upward pointing cusp near the halfline
139-163	252° downward pointing hump approaching the baseline
163-164	Upward pointing cusp near the halfline
164-196	Large area downward pointing clockwise hump through Y = -15
197-197	Pen-up - *Pseudo*-cusp pointing left at Y = -15
197-198	Pen-down - *Pseudo*-cusp pointing left at Y = 13
199-224	T-cross - at Y = 14
225-225	Pen-up - *Pseudo*-cusp pointing right at Y = 15

4.9. Conclusion

The segmentation method described here produces a consistent set of segments for a large number of writing styles. By providing a continuous measure that describes the deformation of everyday handwriting from ideal copy-book script it is possible to compare different handwriting styles that are based on the same copy-book style. The ORCHiD segmentation highlights the "loopiness" of the segments of the writing to provide this measure.

By mapping handwriting to a small number of copy-book styles, the template database used in the recognition system can be reduced, since a number of variations can be matched against the same template. This allows a computationally intensive matching routine to be used that would otherwise be impractical.

Features of each segment are measured, the distribution of which can be approximated by a multivariate normal distribution. These are then suitable for use within a statistical template matching process.

Chapter 5

Template Matching

This chapter describes the template matching algorithm used by the ORCHiD system. A number of template matching methods are described in section 2.5 and their advantages and problems are discussed. The method described here relies on a consistent segmentation, as described in Chapter 4. It produces an ambiguous set of candidate allographs with probabilities. Context can be used to reduce this ambiguous output (see Chapter 6).

5.1. An Introduction to Statistical Template Matching

It is first necessary to define some terminology. This will be illustrated with some simple examples from the field of separated, handwritten character recognition.

A *template* is a definition or computer representation of a real-world object (for example, a numerical description of what a letter 'd' looks like). An *allograph* is a subject specific or context specific topological structure of a character. So a template is used to represent an allograph.

A *sample* is data representing an object we wish to identify in terms of real-world objects (for example, some digitised script). This identification is carried out by comparing the sample with the templates. It may then be possible to make a discrete choice and *classify* the sample as being a particular object, but it is often desirable to allow some ambiguity in the classification. In this case it is necessary to calculate a *match probability* or *weighting* to indicate how well a sample matches a template.

A *feature* is some particular aspect of the data, whether it be the sample data or the template data, which can be measured. This measurement may be taken either on a continuous scale, for example the angle that a stroke makes with the x-axis, or on a discrete scale, for example it may be possible to make the statement "this is a cusp".

Let us consider a simple template matching example for a character recognition system where the feature of interest is the vertical position of cusps within the script. For simplicity we will assume that we have just two templates (see figure 5.1). Template A represents a lower case letter 'a' and its feature description is of two cusps with vertical position 10 (units). Template D represents a lower case letter 'd' and its description is of a cusp with vertical position 10 followed by a cusp with vertical position 20. If the sample consists of a cusp with vertical position 10 followed by a cusp with vertical position 13, then we may be inclined to deduce that the sample is a better match for template A than template D, since the vertical height of the final cusp of the sample (13) is closer to 10 than to 20.

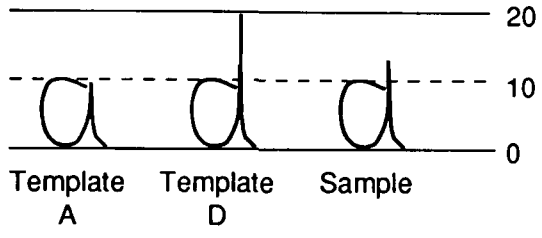


Figure 5.1 - A simple template matching problem

This deduction may not hold, however, if we are aware of some further statistical information about the vertical position of the final cusp in each of the templates. Consider a hypothetical experiment which examined a large number of samples of the letters 'a' and 'd'. Of 1000 samples of the letter 'a', the vertical position of the final cusp was found to vary only between 9 and 11, with most of the samples having a value of 10. However, of 1000 samples of the letter 'd', the vertical position of the final cusp varied between 13 and 25, with a mean value of 20. Our original deduction about the best-match template now seems less reasonable, since it is much less likely that the vertical position of the cusp in template A will exceed 11 and more likely that it will be variable in template D. Statistical theory must be applied to calculate the best match based on the distribution of the values of the feature for each template. We may know the actual distribution, but more than likely we will need to estimate the distribution from a large population of sample data.

Typically a large number of features are measured. The distributions of each of these features can be estimated separately, but it is often convenient to

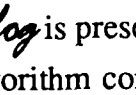
consider an array of the feature values to be a sample from a *multivariate distribution* covering the whole feature space. The number of features is then referred to as the number of *dimensions*. To reduce the theoretical complexities and computation of using generalised statistical distributions, it is usual to select features which yield values with a normal distribution so that the combined distribution is multivariate normal.

5.2. The Goodness of Fit Approach

In the classical statistical approach to template matching, sample data is compared to a template and a probability or weighting is produced representing how well the sample matches that template. The sample may be compared with several templates and the probabilities used to provide a ranked list of possible template matches for the sample data. The template might be described by the expected values of a number of features which could be extracted from the data, and the statistical distribution of values for these features may either be known, or can be estimated from a large population. A number of statistical theories are available which can be applied in this situation to provide a "goodness of fit" rating when comparing a sample to the expected multivariate distribution of the template. These may be based on traditional Bayesian theory[55], or on non-parametric statistics, such as Kolmogorov-Smirnov or Cramer von Mises[56]. The nonparametric tests are mostly favoured due to their simplicity.

When such a template matching approach is applied to cursive script recognition a common methodology is to attempt to segment the script at letter joins and match the segments to templates of characters. Some methods then take the closest matching letter in each position as the correct output, and combine these letters to form a word. Other methods allow for a number of possibilities in each letter position. These letter possibilities are then combined in every possible way to produce a list of possible output words. In this case the probabilities or weightings for each letter must also be combined to give a probability or weighting for each candidate word.

Illustration

Let us consider the following cursive script recognition problem as an example. The word  is presented to a segmentation based recognition system. The segmentation algorithm correctly segments the word at the letter boundaries and

passes the three resulting segments to the recognition phase. This recognition phase indicates that the first segment is either a letter *d* with probability or "goodness of fit" of 90% or a letter *a* with probability 10%; the second segment is either a letter *o* with probability 70% or a letter *a* with probability 30%; and the third segment is either a letter *g* with probability 60% or a letter *y* with probability 40%. It is then perfectly reasonable to multiply these probabilities in the usual way and say that there is a (90% × 70% × 60% =) 37.8% probability of the word being *dog* or a (90% × 30% × 40% =) 10.8% probability of it being *day* and so on for the other six possible combinations of letters.

The Problem with the Classical Approach

Cursive script recognition, however, poses a more complex problem. The segmentation points between letters are not only difficult to detect but are often ambiguous without the application of high level contextual information. There are two possible ways in which allowances can be made for this ambiguity. Either some form of transformation to the output can be applied which corrects any errors that may have occurred in the segmentation stage, for example mistaking *cw* for *au*, or ambiguous segmentation can be allowed with the assumption that the later stages of the recognition will resolve the ambiguity.

The first technique requires a detailed study of the performance of the recognition system to provide accurate statistics of the likelihood of various errors occurring. These errors may be *transliteration* errors, where a letter is misrecognised as another letter, or *segmentation* errors, where the segmentation has occurred in the wrong place and therefore caused recognition errors. It is then possible to take the output from such a recognition system and apply this statistical information to produce the most likely input word for the output produced. Sections 2.7.3-6 gives details of some of these methods.

The second technique retains as much information as possible throughout each stage of the recognition process and only rejects it when there is sufficient evidence, for example if a letter sequence does not make a valid word. In this way it is never necessary to make arbitrary binary decisions and no assumptions are necessary about likely errors that may have been introduced by the system. The ORCHiD system uses this second approach.

Illustration

Continuing with the previous illustration, if we allow an ambiguous segmentation method that provides for the strokes which form the letter *d* to be made up of a letter *c* and a letter *l*, then we encounter a problem with the template matching approach outlined in section 5.2. Using the same recognition module, it might return that the first segment matches the *c* template with a "goodness of fit" of 100% and the second segment is a 100% match for the *l* template. The problem that we encounter is that there is no way to compare the 90% letter *d* with, say, the 100% *c*.

Peleg[72] attempted to rationalise the match weights by constructing a graph of letter possibilities, and calculating probabilities normalised for each group of letters occupying the same space on the graph. Figure 5.2 shows his example letter graph for the word *look* with the possible letters and normalised probabilities. For the reasons described above, the probability calculations are flawed and should not be combined across the word to give a word probability.

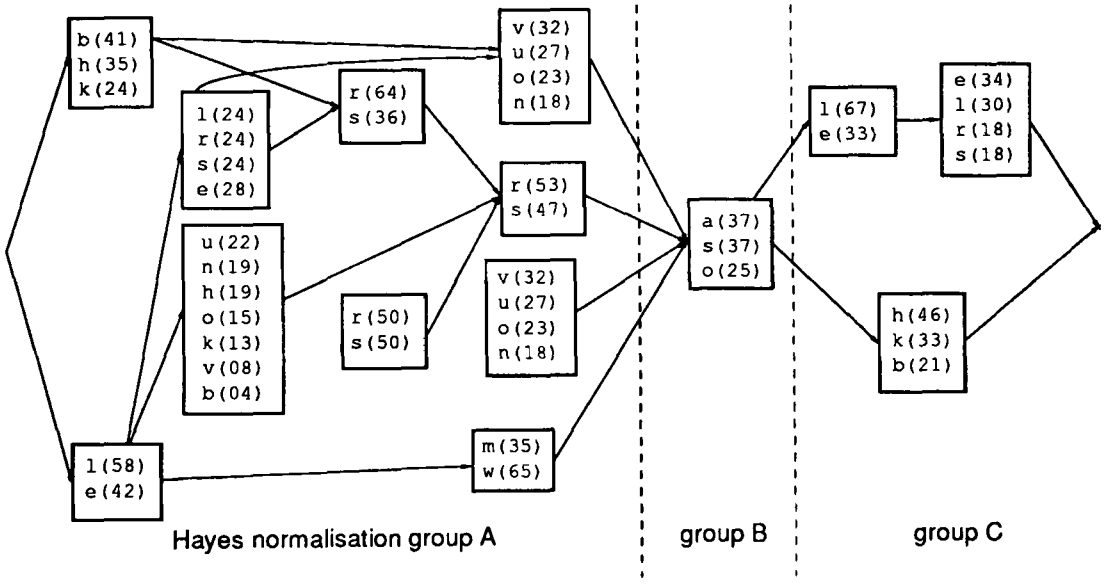


Figure 5.2 - Peleg's normalisation

Hayes[37] attempted to alleviate the problem by normalising across groups of letters that span a section of a word. In figure 5.2, the probabilities would be normalised within the three groups indicated. As letters are deleted from the graph (for contextual or other reasons) the probabilities are re-normalised. This method is still fundamentally flawed, for the same reasons.

The development of any form of statistical cursive script recognition system based on an ambiguous segmentation method must overcome the problem that there is no valid theory available for comparing samples taken from multivariate distributions with different numbers of dimensions. For instance letters may span differing amounts of the sample text and hence have different numbers of features associated with them. As stated above, for the word *dog* we might say that there is a 90% probability that the first letter is a *d* and a 10% probability of it being an *a*, but we may be 100% certain that the first part of the letter, when considered on its own, is a *c*. How can we now compare the *c* with the *d*? The only way that the comparison can make sense is if we consider the letter *d* with the letter combination *cl* taken together.

5.3. A Probabilistic Approach

In the example above comparing *d* with *cl*, the first part of the letter could equally belong to the *c*, *d* or *a*. (We will ignore other possibilities for the purposes of clarity of explanation.) So there is a 33% probability, say, that it belongs to any of these three letters. The second part of the letter involves comparing the ascender of the *d* with the *l* and the second part of the *a*. Assuming that the two true ascenders are equally likely, then they may have a probability of 45% each, say, and the *a* may have a probability of 10%. Combining these probabilities gives a probability for the *d* of 15%, for the *a* of 3.3%, for the *c* of 33% and for the *l* of 45%, but for the *cl* combined the probability is 15%, which can now be sensibly compared with the *d* and the *a*.

We can see that although the "goodness of fit" approach to template matching produces a meaningful value for how well a sample matches a template and can be used to compare the matches of equally sized templates, that value cannot be used to compare the match with a template which is of a different size or has a differing number of features associated with it. Despite this it is very common to find letter weights combined to produce a weighting for a word which is statistically meaningless.

The example above shows a method of comparing template matches which *is* valid in this situation provided the script between segmentation points is sufficiently small that no more than one template can lie within it, no matter how small that template.

This provision leads to the concept of using a set of *sub-letter* templates which represent the constituent parts of actual characters. The script can be segmented, the segments identified by comparison with the sub-letter templates and the templates combined to produce letters (and consequently words). In this way the start of the letters *a*, *c* and *d*, say, might correspond to the same sub-letter template.

A sub-letter template set does not provide for a readily trainable system, however, since the trainer (ie the ordinary user) must know the details of how each character breaks down into its constituent sub-letter templates in order to correctly train those templates. For a system that should require no specialist expertise to operate it, this is clearly unacceptable. The ORCHiD system solves these problems by using whole letter templates that are automatically segmented using the same method as the script sample. In this way each letter template is stored internally as a sequence of sub-letter template segments. So a template for the letter *c* may be made up of just one template segment, but the template for the letter *m* would be made up of several template segments.

The template matching process is carried out at the segment level, so that a segment of the sample data is compared with a segment of a template and a probability calculated (see section 5.7.1). A match weighting for the whole template can then be calculated by combining the probabilities of each of its segments. Section 5.2, however, showed that to sensibly compare two match probabilities, the matches must span the same portion of the sample script. The most convenient way to guarantee this is to combine the probabilities for every segment of a word to get a probability for that word.

[Example: Consider a sample word made up of five segments (ABCDE), and a template database containing two templates - template I, with two segments (PQ), and template II, with three segments (RST). There are clearly two ways that the templates can combine to span the word, either combination *a* - (PQ)(RST) or combination *b* - (RST)(PQ). Let Φ_{aI} be the match probability for template I in combination *a*, which is calculated as $\Phi_{AP}\Phi_{BQ}$, where Φ_{AP} is the match probability of comparing template segment P with sample segment A, and so on. It can be seen that it is not possible to compare any of the values Φ_{aI}, Φ_{aII}, Φ_{bI} or Φ_{bII} directly, but it *is* possible to compare the combined probabilities $\Phi_{aI}\Phi_{aII}$ with $\Phi_{bII}\Phi_{bI}$.]

5.4. Match Probability Calculation

Let us consider a simplified pattern recognition problem. If a sample X can come from one of just two known univariate distributions, a or b, what is the probability that X belongs to a? Assuming that there is no option of X belonging to neither, then the probability that X belongs to a is simply the ratio of the ordinate of the probability density function of a at X (simplistically referred to as the *height* of a at X) to the sum of the heights of the distributions a and b at X (see figure 5.3). (This is referred to as *normalising* the heights to 1.0.) This theory is applicable to any number of distributions, and also to multivariate distributions, provided they have the same number of dimensions.

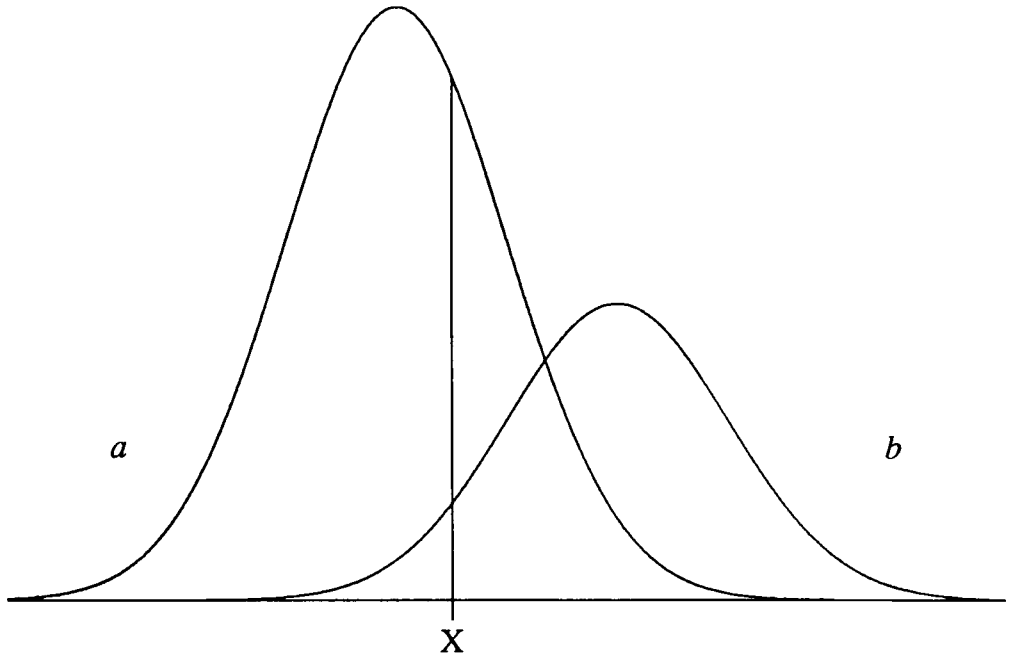


Figure 5.3 - Probability of X belonging to distribution a or b

In the script recognition problem, we can assume that the script must match letters or letter-joins along its whole length, since the writer is not trying to fool the system by making nonsensical marks or squiggles. (It may be necessary to allow for stylish flourishes at the beginning or end of a word which hold no meaning and for errors in script formation. This point will not be further discussed at this stage.) With this additional information about the input which will be presented to the system it is possible to calculate a more meaningful weighting than the "goodness of fit" approach, since that method yields a value which is

independent of other possible template matches. Provided we try to match every possible template in every position along the length of the script sample, we can calculate a real probability for a template occurring at any point within the sample, *conditional* on the fact that it must be one of these templates. By extending the theory described in the previous paragraph, the probability calculation involves calculating the height of the multivariate distribution for each template at the sample feature values and normalising all of these heights to 1.0 - see section 5.7 below.

As explained in section 5.3, we cannot use letter templates as the base unit for comparison between possible templates, but we can use the segments formed when the segmentation algorithm of Chapter 4 is applied to the templates. We therefore calculate match probabilities for each of the template segments by the same method as described above. Each template segment is compared with each sample segment and the height of the template distribution calculated. The segmentation algorithm provides us with segments which have the same number of features, and so their distributions have the same number of dimensions. The heights can now be normalised for each segment, and the resulting probabilities combined in the usual way to give a probability for each template as a whole.

Note that with this method a sample segment that matches a template segment very well will contribute a greater influence to the probability of the whole word than a sample segment that could equally match several template segments.

5.5. Template Database Description

The template database stores a representation of every type of letter and letter-join recognisable by the system. The system is easily trainable for any style of writing or letter formation. This is achieved by using *data-driven* templates, whereby any number of samples of a character are averaged together to provide a new template (see Chapter 7). In this way a *fuzzy* picture of the character is used to compare against any new sample data, and a distance measure can be calculated. Samples which define the template can be taken from just a single user, to give an accurate personalised template, or from a population of users, to give a less accurate but more generalised template. It is therefore possible to have a user specific or a generalised recognition system simply by installing a different template database. As well as storing the mean values for the features of a template, the number of samples used and the standard deviation of the samples is also

available for use in the distance measure and for subsequent training.

This data-driven template approach allows any portion of a handwritten word to be defined as a template. A simplistic view might therefore be to define a set of templates for each letter, and combine those letters to form words. Unfortunately, this is not practical since letters are formed differently depending on the preceding and following letters[38]. It would therefore be necessary to record different types of letters for different positions within different letter combinations, and ensure the correct combination of letters at the word formation stage. Since it is possible to define any portion of the script as a template, it was decided that it would be easiest to define *letter-join* templates as well as letter templates. As a large number of letters either start or end in similar ways, it is not necessary to have 26×26 different letter-joins. Figure 5.4 shows those which were used. It is possible to ignore letter-joins completely, and simply flag those segments of script as *ligatures*. It was found, however, that these strokes can be useful, in some cases, to reject possible letters and letter combinations. More importantly, ignoring these ligatures would have also invalidated the statistical theory developed above.

The use of letter-joins as templates creates the need for two extra *pseudo-letters* to be included in the alphabet to indicate the start of a word (represented by '@') and the end of a word ('#'). Templates therefore exist for letter-joins from '@' to each letter, and from each letter to '#'.

As described above, a template may consist of a number of segments. For each segment the mean, standard deviation and number of generating samples can be calculated for a number of features from the information stored. The standard deviation provides an indication of the spread of samples for that particular feature, and so is an indication of its consistency, and hence its usefulness in distinguishing between templates. This is essential in the calculation of the *distance* of a sample segment from the template segment.

For each feature ORCHiD stores the sum of the samples, the sum of the squares of the samples and the number of samples. The mean and standard deviation can easily be calculated from this information. The number of samples making up a template could also be used as a measure of reliability for that template since it indicates how often it has been used. Perhaps a further enhancement of the template matching routine might be to reject templates from the personal template database when they have not been recently used.

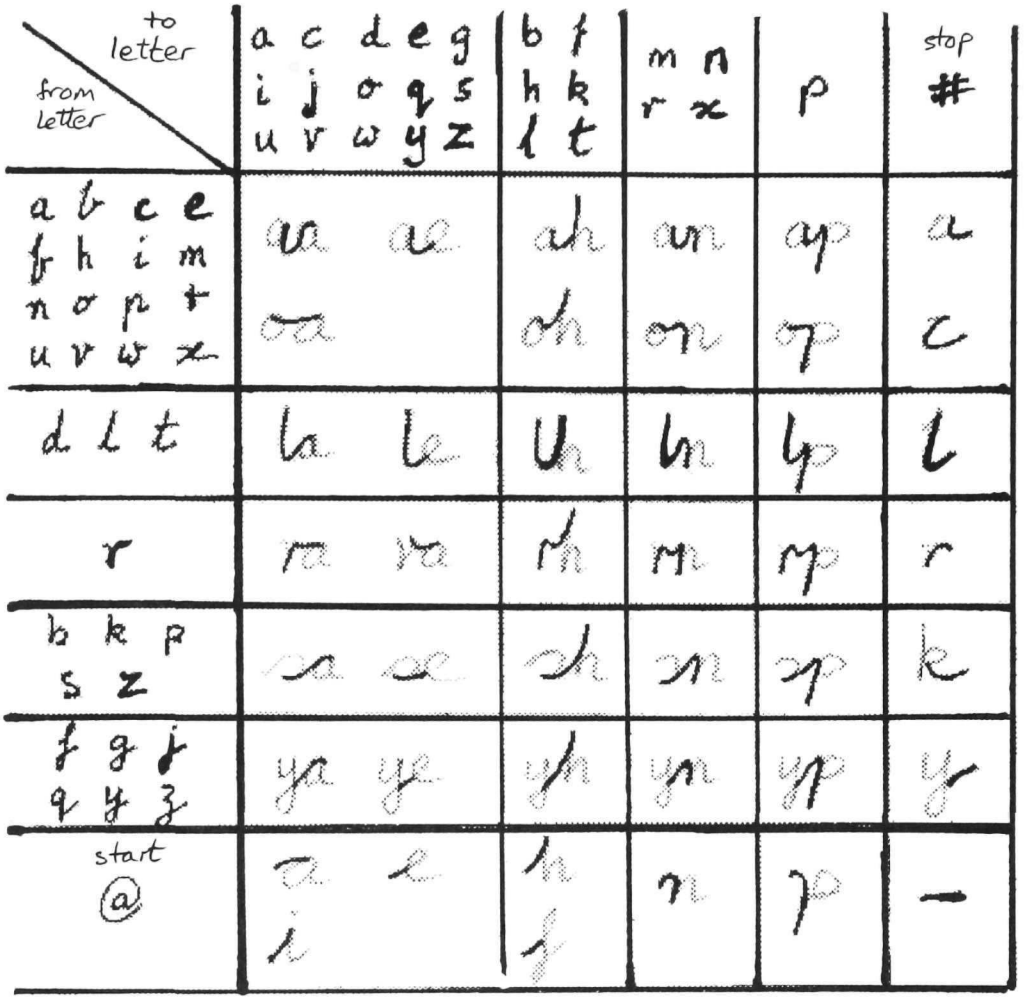


Figure 5.4 - Letter-joins

5.6. Distance Measure

A fixed number of features are measures for each segment. If we assume that each feature is distributed normally, and that all of the features of a segment taken together can then be represented by a multivariate normal distribution (see section 4.7), then the distance we require is the ordinate or *height* of the probability density function of the distribution at the sample segment values, see section 5.4. Normalisation of these distances across all of the other template segments then gives us the probability that it matches that template, given that it must match one of the templates.

Calculation of the ordinates of the multivariate normal distributions for each template for every sample segment would be extremely time-consuming, due to

the complexity of the equation (see below), so a simplification is necessary.

Multivariate Normal Distribution

If $\mathbf{X} = (x_1, x_2, \cdots, x_d)$ is a sample from a multivariate normal distribution with array of means $\mathbf{M} = (\mu_1, \mu_2, \cdots, \mu_d)$ and covariance matrix

$$\mathbf{V} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \cdots & \sigma_{1d}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \cdots & \sigma_{2d}^2 \\ \vdots & \vdots & \cdots & \vdots \\ \sigma_{d1}^2 & \sigma_{d2}^2 & \cdots & \sigma_{dd}^2 \end{bmatrix}$$

then the ordinate (or height) of the probability distribution function ϕ is given by,

$$\phi(\mathbf{X}) = \frac{e^{-\frac{1}{2}(\mathbf{X}-\mathbf{M})^t \mathbf{V}^{-1} (\mathbf{X}-\mathbf{M})}}{(2\pi)^{\frac{d}{2}} |\mathbf{V}|^{\frac{1}{2}}}$$

where d is the dimensionality, $(\mathbf{X}-\mathbf{M})^t$ indicates the transpose of the array $(\mathbf{X}-\mathbf{M})$, $|\mathbf{V}|$ is the determinate of the matrix $\mathbf{V}$, and $\mathbf{V}^{-1}$ is the inverse of $\mathbf{V}$.

If we assume, for simplicity, that the variables are independent (see section 4.7) so that the covariance matrix $\mathbf{V}$ is diagonal,

$$\mathbf{V} = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_d^2 \end{bmatrix}$$

then

$$\phi(\mathbf{X}) = \frac{e^{-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i-\mu_i)^2}{\sigma_i^2}}}{(2\pi)^{\frac{d}{2}} \prod_{i=1}^{d} \sigma_i} = \prod_{i=1}^{d} \frac{e^{-\frac{1}{2}\frac{(x_i-\mu_i)^2}{\sigma_i^2}}}{(2\pi)^{\frac{1}{2}} \sigma_i}$$

Normal Distribution

If x is a sample from a (univariate) normal distribution $N(\mu, \sigma^2)$, then

$$\phi(x) = \frac{e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}}{\sigma\sqrt{2\pi}}$$

So for the multivariate normal distribution

$$\phi(\mathbf{X}) = \prod_{i=1}^{d} \phi(x_i)$$

It can therefore be seen that, provided the features are independent of each other, the height of the multivariate distribution is simply the product of the individual univariate distributions. It is computationally expensive to calculate the ordinate of the normal distribution for every feature of every template segment, so an approximation is used. This is based on the ordinates of the standard normal distribution $N(0,1)$.

Standard Normal Distribution

If z is a sample from a standard normal distribution $N(0,1)$, then

$$\phi_N(z) = \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}$$

To calculate $\phi(x)$ where x comes from $N(\mu,\sigma^2)$ from the ordinate values of the standard normal distribution $\phi_N(z)$, make the substitution

$$z = \frac{x-\mu}{\sigma}$$

then

$$\phi(x) = \frac{\phi_N(z)}{\sigma}$$

For our purposes, the distribution is divided into fixed width portions either side of the mean (see figure 5.5). The ordinate in the centre of each portion is calculated and for any sample lying within that portion the central ordinate is returned as the approximation to the real ordinate. The ordinate of a normal distribution more than three standard deviations away from the mean is virtually zero (< 0.5% of the maximum ordinate at the mean), so this was selected as the maximum distance from the mean for which ordinates would be calculated. The intermediate section was divided into half standard deviation portions, since this provided a reasonable approximation (see below) and involved calculation of just seven ordinate values. The ordinate values are

$$\phi(\mu) = \frac{1}{\sigma\sqrt{2\pi}}, \quad \phi(\mu+\tfrac{1}{2}\sigma) = \frac{e^{-1/8}}{\sigma\sqrt{2\pi}}, \quad \phi(\mu+\sigma) = \frac{e^{-1/2}}{\sigma\sqrt{2\pi}}$$

and so on.

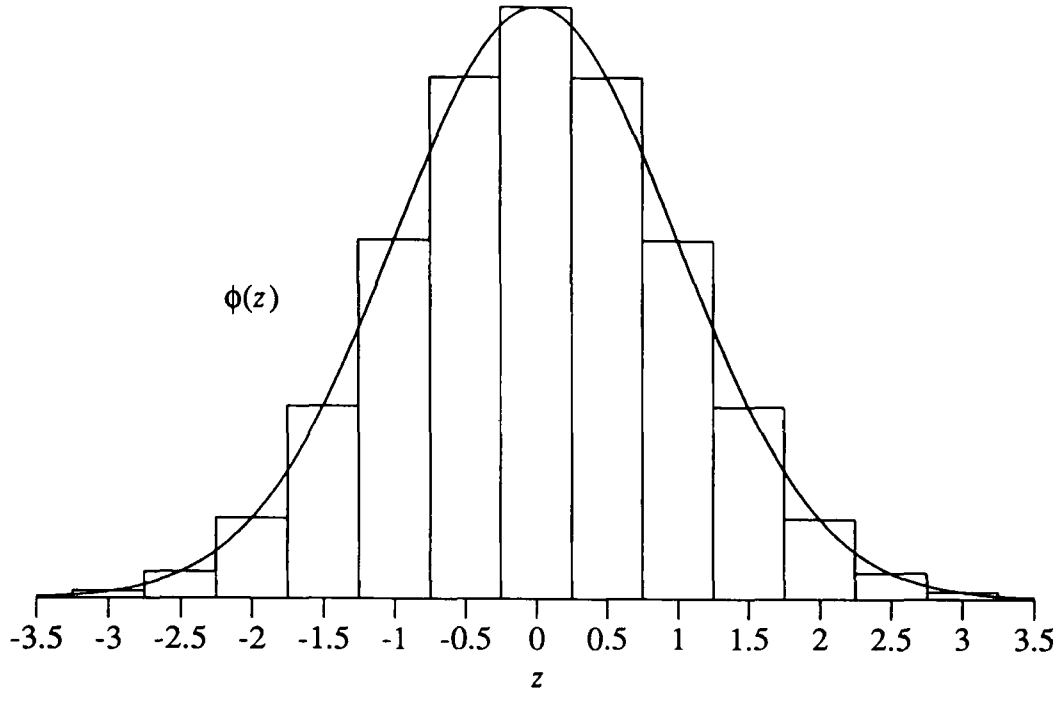


Figure 5.5 - Approximation to normal distribution

Closer approximations could have been used. This one was selected since it could be implemented extremely efficiently. This was important since this calculation is needed many times during the execution of the program. The next section shows that the approximation is, in fact, sufficient.

So for each feature, the distance of the sample from the template mean is calculated and scaled by the standard deviation. The relevant approximation to the standard normal distribution is looked up from the table and divided by the standard deviation. Each of the values for the features of a segment are then multiplied together to give the approximation to the ordinate of the multivariate normal distribution, which is the distance measure required.

Validity of the Approximation

It is possible to show that this approximation is reasonable if it can be shown that the area under the curve between the two bounding points of each portion ($\int\phi$) is approximately equal to the area under the corresponding histogram. Table 5.1, extracted from statistical tables, shows this to be sufficient. The average error across the range $-3.25<z<3.25$ is approximately 0.2%.

[Notation: $\Phi(z) = \int_z^\infty \phi(t)dt$]

z	0.0	0.25	0.50	.75	1.0	1.25	1.5
$\phi(z)$	.3989	-	.3521	-	.2420	-	.1295
$\Phi(z)$	.5000	.4013	-	.2266	-	.1056	-
z	1.75	2.0	2.25	2.5	2.75	3.0	3.25
$\phi(z)$	-	.0540	-	.0175	-	.0044	
$\Phi(z)$	.0401	-	.0122	-	.00298	-	.000577

Range	0 - .25	.25 - .75	.75 - 1.25	1.25 - 1.75	1.75 - 2.25	2.25 - 2.75	2.75 - 3.25
$\int\phi$	.0987	.1747	.1210	.0655	.0279	.00922	.002403
Approx	.0997	.1761	.1210	.0648	.0270	.00875	.002200
%age error	1.0	0.8	0.0	1.1	3.2	5.2	8.4

Table 5.1 - Errors in the approximation

Mathematical Summary

It is now possible to express formally the distance measure used, and the match probability between a sample segment and a template segment.

Let **X** be a sample segment feature array with d elements $(x_1, \cdots, x_d)$. Let there be n template segments $\mathbf{T}_1$ to $\mathbf{T}_n$ for comparison with the sample segment (each of dimension d), and assume that these templates come from a multivariate normal distribution. Let the feature elements of $\mathbf{T}_i$ be $(t_{i1}, \cdots, t_{id})$ and assume that each t_{ik} is normally distributed with mean μ_{ik} and standard deviation σ_{ik}, and the distributions are independent. Let the ordinate of the distribution of $\mathbf{T}_i$ at **X** be $\phi_{\mathbf{T}_i}(\mathbf{X})$ and the ordinate of the distribution of t_{ik} at x_k be $\phi_{t_{ik}}(x_k)$. Let $\phi_N(z)$ be the ordinate of the standard normal distribution $N(0,1)$ at z.

Then the distance measure between **X** and $\mathbf{T}_i$ is given by

$$\phi_{\mathbf{T}_i}(\mathbf{X}) = \prod_{k=1}^{d} \phi_{t_{ik}}(x_k)$$

$$= \prod_{k=1}^{d} \frac{\phi_N\left[\frac{x_k - \mu_{ik}}{\sigma_{ik}}\right]}{\sigma_{ik}}$$

5.7. Implementation of the Template Matching Process

The template matching process consists of several consecutive sub-routines - template comparison, segment normalisation and letter graph formation.

5.7.1. Template Comparison

In the template comparison stage, each segment of every template is compared to each segment of the sample word, and a distance measure calculated. For example in the simple case of figure 5.6 there are three templates R, S and T to be matched against the sample data D. D has been segmented into 5 segments, D_1-D_5, and R, S and T are made up of 1, 2 and 3 segments respectively (R_1, S_1-S_2, T_1-T_3). The following distances are then calculated - R_1 to each of D_1-D_5, S_1 to D_1-D_4, S_2 to D_2-D_5, T_1 to D_1-D_3, T_2 to D_2-D_4 and T_3 to D_3-D_5. In this way, each template has been examined in every possible segment position that it can occupy.

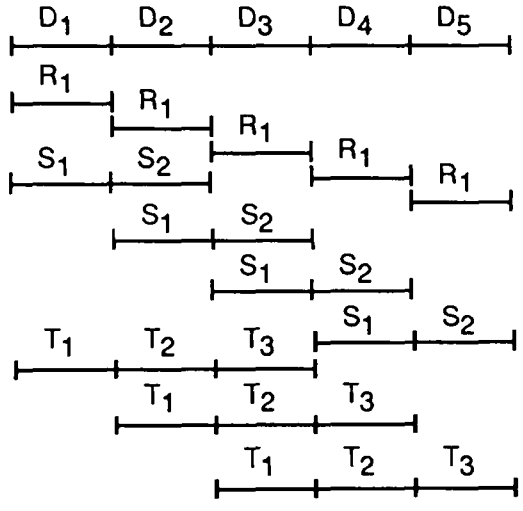


Figure 5.6 - Matching templates to data sample

The distance measure, which is the approximation to the height of the multivariate distribution of the template segment at the sample segment, is referred to as the *comparison score* for that segment of that template. These scores are recorded in a *template comparison array*, and this array is located at the appropriate segment position within the *sample word comparison array*. For example, the template comparison array for template S starting at segment 3 of the sample word would be $[\phi_{S_1}(D_3),\phi_{S_2}(D_4)]$, and the sample word comparison array is

shown in figure 5.7.

Seg 1	Seg 2	Seg 3	Seg 4	Seg 5
$\phi_{R_1}(D_1)$	-	-	-	-
-	$\phi_{R_1}(D_2)$	-	-	-
-	-	$\phi_{R_1}(D_3)$	-	-
-	-	-	$\phi_{R_1}(D_4)$	-
-	-	-	-	$\phi_{R_1}(D_5)$
$\phi_{S_1}(D_1)$	$\phi_{S_2}(D_2)$	-	-	-
-	$\phi_{S_1}(D_2)$	$\phi_{S_2}(D_3)$	-	-
-	-	$\phi_{S_1}(D_3)$	$\phi_{S_2}(D_4)$	-
-	-	-	$\phi_{S_1}(D_4)$	$\phi_{S_2}(D_5)$
$\phi_{T_1}(D_1)$	$\phi_{T_2}(D_2)$	$\phi_{T_3}(D_3)$	-	-
-	$\phi_{T_1}(D_2)$	$\phi_{T_2}(D_3)$	$\phi_{T_3}(D_4)$	-
-	-	$\phi_{T_1}(D_3)$	$\phi_{T_2}(D_4)$	$\phi_{T_3}(D_5)$

Figure 5.7 - Sample word comparison array

The matching template is referred to as a *candidate allograph* for that position in the sample word. A few statistical anomalies can produce spurious candidate allographs. Section 8.5.4 discusses these anomalies and their causes in more detail. Two criteria are examined to decide whether a candidate allograph has appeared because of one of these anomalies. Such candidate allographs are then removed.

1) Within a segment of a sample it is possible that one or two of the features may match the template very well, the rest matching very poorly. After calculation of the comparison score, it is possible that the good match(es) may outweigh the poor matches and produce a seemingly reasonable score. This score may, however, be a mathematical anomaly. (For example, consider eight features measured for a sample segment. If, by chance, one of these features matches the template feature exactly, and that feature distribution has a very low standard deviation, then the height value for that feature will be very great. If the other features match poorly, the height values for the other features will be very low. The comparison score may be reasonable,

though, due to the influence of the first feature, even though intuitively it would be said that this is a poor match overall.) To counteract this anomaly, a candidate allograph is rejected if a majority of a sample segment's feature values are more than three standard deviations away from the mean of the template feature values.

2) It is possible for the statistical method to allow candidate allographs which should be rejected for other reasons. Two basic elements of each segment are examined to see if a match between the template and the data is impossible. The area occupied by a script word is divided into horizontal strips, see figure 5.8. The vertical region in which each sample segment is located is compared with the expected region for the template segment. If the sample is located more than one region away from the expected region, the candidate allograph is deleted. The rotation of the segment is also compared. Provided neither sample nor template segment is a cusp, if the "clockwiseness" is opposite, the candidate allograph is rejected.

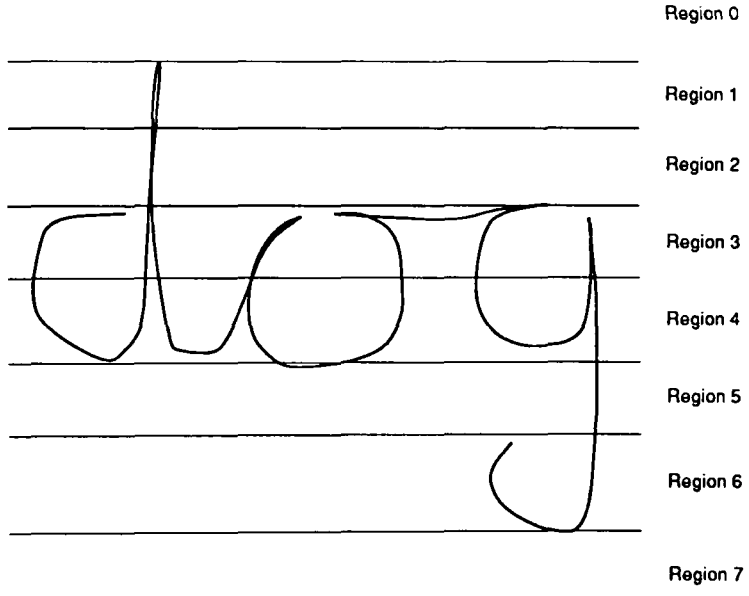


Figure 5.8 - Regions for vertical position of segments

5.7.2. Segment Normalisation

The template comparison scores are normalised to 1.0 across each column of the sample word comparison array to give *segment-normalised comparison scores*. These scores are equivalent to the probability that a sample segment

matches the template segment.

We can represent the normalisation process in the formal terms of the formulae derived above. If $\mathbf{X} \in \mathbf{T}_i$ means the sample segment $\mathbf{X}$ is an instance of the template segment $\mathbf{T}_i$, then the match probability P, conditional on the fact that $\mathbf{X}$ must match one of the templates, is given by

$$P(\mathbf{X} \in \mathbf{T}_i \mid \mathbf{X} \in \mathbf{T}_1, \cdots, \mathbf{T}_n) = \frac{\phi_{\mathbf{T}_i}(\mathbf{X})}{\sum_{j=1}^{n} \phi_{\mathbf{T}_j}(\mathbf{X})}$$

So, if we substitute the equation for $\phi_{\mathbf{T}_i}(\mathbf{X})$ from above we get

$$= \frac{\prod_{k=1}^{d} \frac{\phi_N\left[\frac{x_k - \mu_{ik}}{\sigma_{ik}}\right]}{\sigma_{ik}}}{\sum_{j=1}^{n} \prod_{k=1}^{d} \frac{\phi_N\left[\frac{x_k - \mu_{jk}}{\sigma_{jk}}\right]}{\sigma_{jk}}}$$

It should be noted that this normalisation decreases the influence on the word probability of a segment with a number of equally matched candidate allographs and increases the influence of a segment with a single outstandingly good match. In this way the influence of each segment on the word probability is limited and the possibility of a single, large segment comparison score overwhelming the influence of the other segments is minimised.

There is clearly a heavy computational requirement to implement this template matching method. (If there are n templates in the database, each template consisting of s segments on average, and there are d segments in the sample data, then approximately $n \times s \times d$ comparison scores must be calculated.) To reduce the computation needed, a candidate allograph is rejected at this stage if any of its segment normalised comparison scores fall below a threshold. This threshold is calculated by experiment to provide a reasonable execution time without deleting any correct candidate allographs.

The segment-normalised comparison scores are now combined across each candidate allograph by multiplication to give a *weighting*. It should be noted that because these weightings are dependent on the number of segments in an allograph, they cannot be used for comparisons between different size candidate

allographs.

5.7.3. Letter Graph Formation

A directed graph is constructed to represent how each candidate allograph may connect with every other allograph, see figure 5.9. This graph can then be traced to produce a list of *candidate words* that might match the sample script.

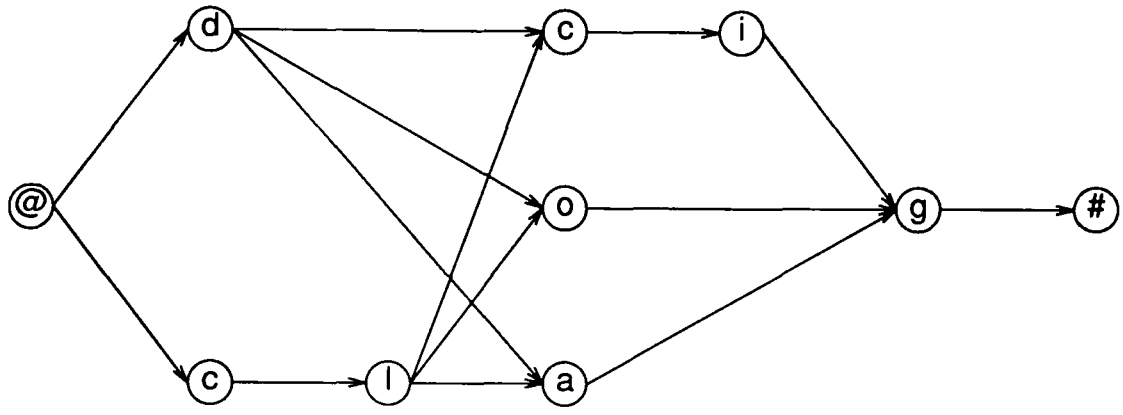


Figure 5.9 - A directed letter graph

The template matching routine provides a list of candidate allographs, for each of which is recorded its weighting, calculated from the normalised comparison scores for each segment of the allograph, and its location within the sample word. The location of the allograph is indicated by the sequence number of the segmentation points bounding it. Each candidate allograph is placed onto the graph and connected (with pointers) to those allographs that may precede or follow it, as defined by their locations.

On each node of the graph is stored the name of the allograph together with its weighting. Each path within the graph can be traced (from left to right in figure 5.9) to produce a candidate word and a probability calculated for how well the sample matches that candidate word. This probability is calculated simply by multiplying the weightings of the candidate allographs making up the candidate word. Since the number of segments across the whole word is the same, no matter which combination of allographs is used, these probabilities satisfy the conditions discussed in section 5.2 and can thus be used to directly compare the different candidate words. Using these probabilities the candidate words are ranked.

With a system that retains a large amount of ambiguity, such as ORCHiD , the letter graph can become very large, with many nodes. This is especially the case since the candidate allographs can represent letter-joins as well as letters. Although in theory the graph tracing process described in Chapter 6 does not drastically suffer with large letter graphs, it is a recursive process which in practice requires a finite amount of time for each call of the recursion. As the letter graph gets extremely large, this extra execution time can become considerable, especially if the amount of stack required for the recursive calls approaches the amount of memory available, causing excessive page faulting.

To reduce the size of the graph, a *template-join table* has been constructed which contains a list of which letter-joins are allowable between letters, whether letters can join directly to another letter without a join, which templates must be located on a pen-up or pen-down, and so on. As each candidate allograph is placed on the graph, any other allograph which may connect to it is checked in the template-join table to ensure the connection is permissible. As well as reducing the size of the letter graph, this method also increases the accuracy of the recognition system by removing inadmissible allograph sequences that will not be removed by word-level context.

5.8. Conclusion

A statistical template matching routine has been developed which compares a sequence of contiguous sample segments with a database of templates. Each segment is described by a set of numerical features and these allow a probability of match to be calculated. The match probabilities, when combined across the whole word, provide a valid weighting for comparison between different possible combinations of templates.

The template database is constructed very simply, by storing the average feature values from any number of samples of a template. This enables training to be straightforward (see Chapter 7). At present only lower case letter templates have been used but, in theory, capital letters could easily be included by defining extra templates.

The process is computationally expensive but shows the value of a data-driven approach to recognition. There are a number of problems associated with this method, however, and these are discussed further in section 8.5.4. Section 9.3.5 discusses how this method might be combined with different recognition

approaches to produce more consistent results.

A letter graph is produced as the output from this section which can be traced to produce a ranked list of candidate words that match the sample. There will be a large number of these words since the method only deletes possibilities that are extremely unlikely. Many of these words will not be valid in the language so it is necessary to verify each of the candidate words against a lexicon or dictionary.

Chapter 6

Candidate Word Verification

This chapter discusses the use of word level context to reduce a set of ambiguous template matches and verify the output. Some methods previously used are discussed in section 2.7. A ranked list of valid candidate words is produced and this is further reduced by application of diacritical mark information and *a priori* word probabilities.

6.1. Output from Segmentation Based Recognition Systems

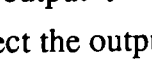
The output from typical segmentation based text recognition systems can fall into one of two main categories:

(a) a single word can be produced consisting of the most likely letters which span the written sample; or

(b) a list of possible letters in each segmentation position can be produced, together with a weighting of their likelihood.

Recognition systems belonging to type (a) have made a binary decision at this stage as to where letter segmentation occurs - this will be referred to as *fixed letter-segmentation*. Those belonging to type (b) retain all of the possible segmentation points and will be referred to as *ambiguous letter-segmentation*. No decision has been made at this stage as to exactly where the letters occur in the written word.

Example

With a fixed letter-segmentation recognition system, the most likely word is produced as output. For example, if the word  was presented to the system, the letter matching algorithm may make a mistake in the second letter position and produce the output  . It is then the task of any contextual post-processor to correct the output and produce the most likely real word.

An ambiguous letter-segmentation system produces a list of possible candidates in each segment position, together with a *certainty weight* to indicate the

closeness of the match. For example, if the word *dog* was presented to the system, it may produce two options for the second character, perhaps an *a* with high certainty weight of 60 or an *o* with a lower certainty weight of 40.

In cursive script recognition systems, individual characters are permitted to run into each other, with the problem that segmentation of the script into letters is ambiguous. For example, if the word *dog* is written cursively, there is the possibility that the word may be *clog*, where the segmentation after the first letter is ambiguous.

One way to represent the output from such a system is as a directed graph (Hayes[37], Peleg[72], Higgins and Whitrow[39]). Figure 6.1 shows a simplified letter graph that might be produced by a recognition system acting on the input script *dog* . The '@' symbol represents the start of a word and the '#' symbol represents the end of a word. The graph is traversed from left to right, yielding a list of all possible combinations of letters that the original data might represent, ie *dccg*, *dog*, *dag*, *clccg*, *clog*, *clag*. The certainty weights can be attached to each letter on the graph and combined as the graph is traced to produce a ranking for each word that is produced.

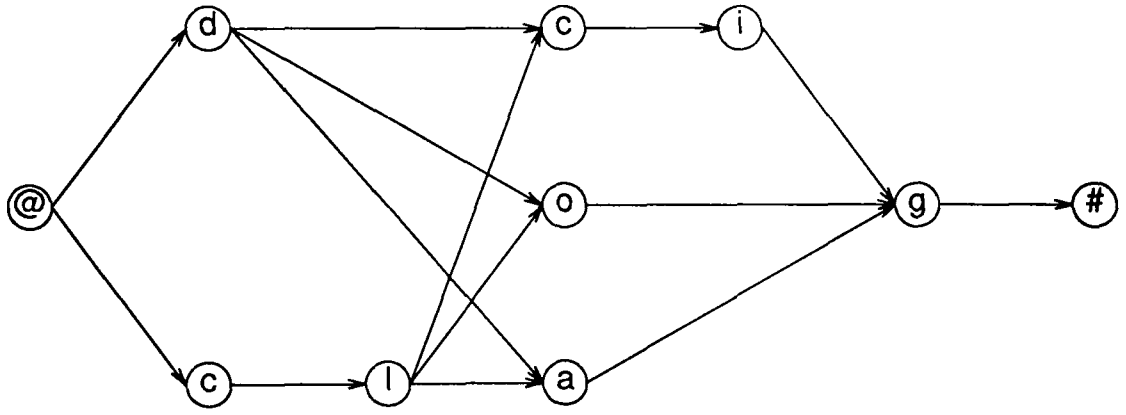


Figure 6.1 - A simple letter graph

Such a letter graph may produce a large number of words if there is a lot of ambiguity, and many of these words will not be valid. It is therefore necessary to develop an efficient method of verifying each of these words.

6.2. Use of Contextual Information from Dictionary Sources

Since a cursive script recognition (CSR) system will usually be required to recognise real words, rather than random sequences of letters, it is possible to define a dictionary or lexicon of valid words. Information can be extracted from this and used to restrict and verify the output from the system. If such a lexicon is used, however, it is essential that every word that is required to be recognised is included within it.

6.2.1. Fixed Letter-Segmentation

Most of the past approaches to using contextual information have been based on constructing a post-processor for the output from fixed letter-segmentation recognition systems. The output from such a system is a single word, consisting of the most likely letters that span the sample script. The contextual disambiguation process takes that word as its input and returns the most likely written word as its output, using contextual and other information to make this decision. Several techniques are discussed in Chapter 2 most of which are based on the Viterbi Algorithm.

6.2.2. Ambiguous Letter-Segmentation

The problem with all of the systems based on the Viterbi Algorithm is that they do not allow for incorrect letter segmentation. For example if the word  is misrecognised as  then the algorithm cannot produce the correct word and the system will fail. An ambiguous letter-segmentation retains all of the possible segmentation points within the letter graph representation.

For an ambiguous script recognition system, the contextual disambiguation process takes a letter graph as its input, and produces as its output a list of words which have been checked, in some way, against a lexicon of dictionary information. One simple way to do this is to trace every path through the graph and check each resulting word against a list of valid words. This is not practical since the letter graph is usually very large with many paths. Even the most efficient dictionary searching algorithms will take a significant amount of processing time to verify a very large list of words.

Commercial spell-checking software cannot be used since, for efficiency, the algorithms they employ usually rely on the input words either being correct, or

very close approximations to real words. Often a hash-coding is used on the word to be checked and a flag set in the hash-table if that word is valid. The hash-code is usually designed so that encodings of slight mis-spellings will not collide with correct spellings, but completely random sequences of letters may often be accepted as valid words. The UNIX spell facility is an example of such a system.

A much more efficient and reliable technique is required.

6.3. Binary N-Gram Graph Reduction

This technique makes use of the existence or non-existence of *n*-letter sequences in English. Higgins[38] reported that four is the ideal length of gram to use, since only approximately 5% of 4-grams are valid in English, and the number of possible grams, $26^4 = 456\,976$, can be reasonably stored as a binary array occupying just under 56 Kbytes of memory. A much larger percentage of 3-grams are valid, and 5-grams would require about 1.5 Mbytes of memory for the binary array with little gain in context.

The letter graph is supplemented by adding an extra start and stop node at the beginning and end of the graph. This is so that the opening two and three letter sequences can be checked using the same 4-gram approach.

The graph reduction process can be implemented in many different ways, see Whitrow and Higgins[102]. An efficient technique has been selected, which uses a similar letter graph structure and tracing algorithm as will be discussed in section 6.4, for the purposes of comparison with the dictionary tree search algorithm. This technique involves recursively tracing each path through the letter graph, maintaining pointers to the last four letters accessed. At each step the current four letter sequence is compared against the list of valid 4-grams. If the sequence *is* valid, then the three arcs connecting the four letters are *marked.* The three arcs are marked differently depending on which letter positions within the sequence they connect (first two, middle two or last two letters). An arc which has been in each of the three connecting positions of a four letter sequence is flagged as *used.* After the traversal is complete, the graph is stripped of all *unused* links, the arc-markings are cleared, and the process repeated until no more links are removed. In this way a much reduced letter graph is produced, which can be more readily checked against a dictionary using a straightforward approach.

6.3.1. Speed of Reduction

The speed of graph reduction for the technique described above is dependent on the number of possible paths through the graph, which is in turn dependent on the length of the word and on the *confusion level* of the letter graph. This is defined to be the number of letter options at each letter position.

To test the speed of reduction of this technique in a controlled way, simulated letter graphs were generated for the word "test". The confusion level at each letter position could be varied and was uniform across the graph. (This is not the case in a real letter graph.) The plot in figure 6.2 shows how the time taken for the reduction greatly increases as the confusion level increases, as expected.

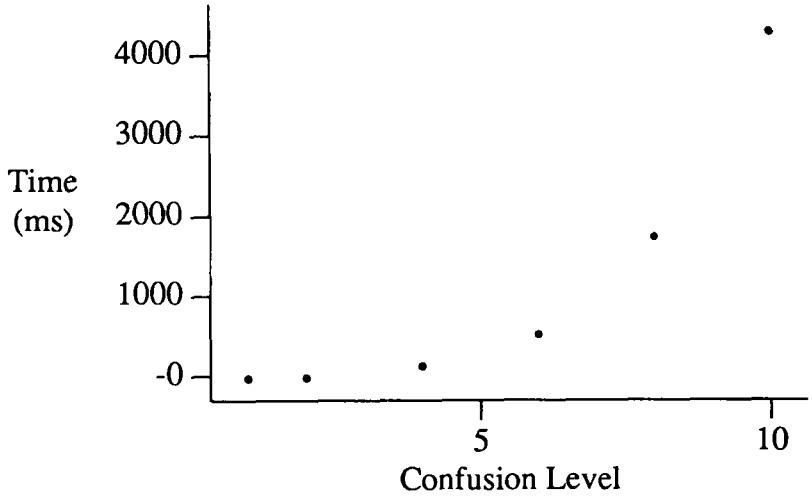


Figure 6.2 - 4-gram reduction time *vs* confusion

6.3.2. Output

4-gram graph reduction produces a much reduced letter graph, but on traversal invalid words will still be produced. In a sequence of letters each individual gram may be permissible but this will not guarantee that the whole word is valid. Also, an arc in the letter graph can only be deleted if *no* valid gram uses it. A remaining arc will then still allow invalid grams to pass through it. (Figure 6.3 shows a subgraph of a letter graph, where the gram *ebcf* may be invalid but *abcd* valid. The arc marked '*' must therefore remain in the graph, so the invalid gram will still be present when the graph is traced.) It is therefore still necessary to check each word against a dictionary to guarantee validity. This will incur the overhead of additional processing time.

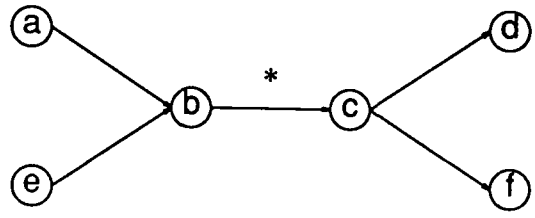


Figure 6.3 - Arc deletion

The *n*-gram approach is a fast graph reduction technique with small memory requirements but the overheads required to verify the final list of candidate words can be large. A new approach was investigated to see if it would be more successful[28].

6.4. Tree-Based Dictionary Lookup

A dictionary or word-list is restructured in the form of a tree, based on the *trie* structure suggested by Knuth[54]. This is shown pictorially in figure 6.4, where the tree represents the word list {a, an, and, at, be, bet, but, by}. Each of these words can be found by tracing a path from left to right. The '@' symbol represents the start of a word and the '#' symbol represents the end of a word.

The ORCHiD system efficiently traces a letter graph, such as that shown in figure 6.1, using a recursive procedure. Such a procedure might take the head-node of the graph on which it is to act as its parameter and call itself recursively, passing each subgraph that the head-node points to as a parameter, in turn. This will carry out a depth-first trace of the graph. Using a tree structure, the dictionary can be traced simultaneously. As each arc of the letter graph is traced, if the corresponding arc exists in the dictionary tree, then the word is valid *up to that point.* When the end-of-word marker is reached in the graph and tree, the word traced out exists in the dictionary. An invalid word in the letter graph will be rejected *as soon as* an arc cannot be found in the dictionary tree. It is not necessary to continue tracing the graph past this point. This is advantageous since it limits the time taken to search the dictionary, and allows the dictionary to be very large without seriously reducing the performance.

By using these similar data structures, the letter graph and dictionary tree complement each other perfectly. The graph can be traced extremely quickly and efficiently, without following any irrelevant paths.

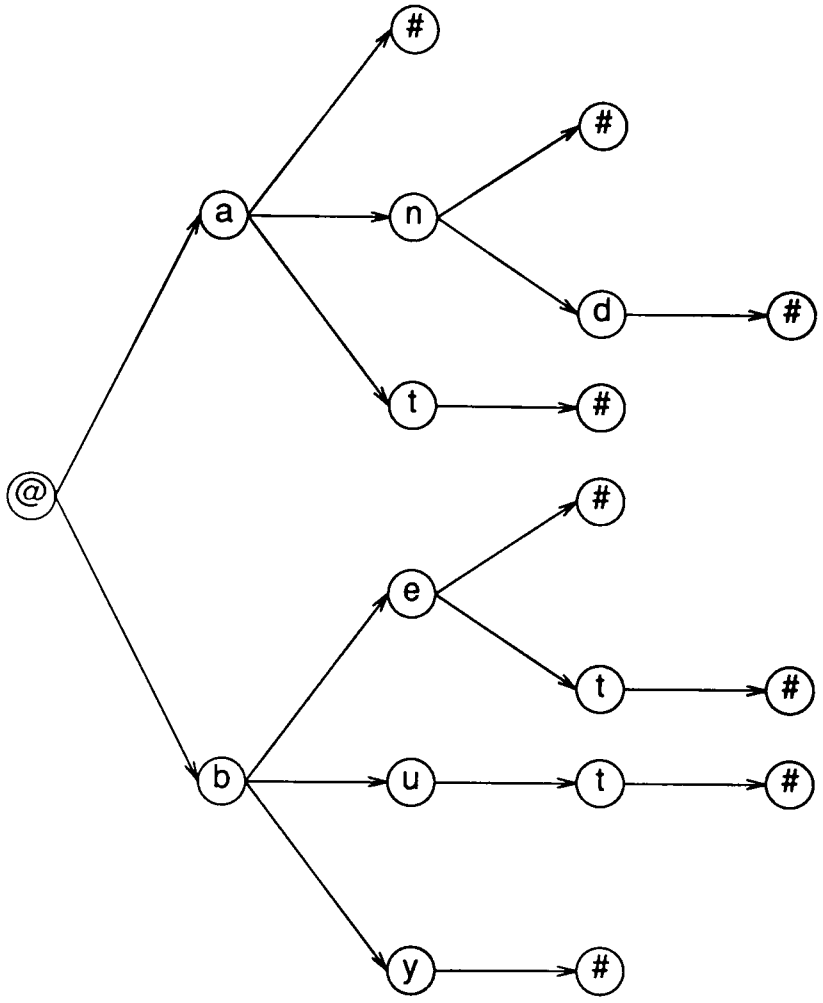


Figure 6.4 - Dictionary tree - pictorial representation

All valid words that exist in the letter graph can be produced in this way and become candidate words (with a weighting as described in section 5.7.3). No information is thrown away at this stage which may prove useful later on if, for example, more than one recognition routine is used in parallel and the outputs combined.

6.5. Computer Representation of the Dictionary Tree

A dictionary structured in this way can be accessed very quickly. It has the disadvantage of requiring a large amount of memory, since the data structure remains resident while the program is running. This is probably the reason for its rejection as a suitable method in the past, but as computer memory is becoming

larger and more readily available this is no longer a major problem. A number of different ways to construct this data structure within a computer program are considered below.

6.5.1. Discussion of Possible Data Structures

Knuth[54] suggests a static data structure for the dictionary tree. This consists of a flag to indicate whether a word can end here and an array of 26 element integer arrays together with 26 boolean flags. Each column position points to the column containing the next letter in a valid word, and a corresponding flag to indicate if a word can end at that point. This array is very sparse and would take up a large amount of memory. Nowadays, dynamic memory allocation allows more efficient data structures to be used.

A simple dynamic data structure might consist of a tree of linked nodes, where each node contains 26 pointers to possible successor nodes (see figure 6.5). In this particular layout, the letter is implied by the position of the pointer in the array. Such a structure will be extremely sparse and have huge memory requirements.

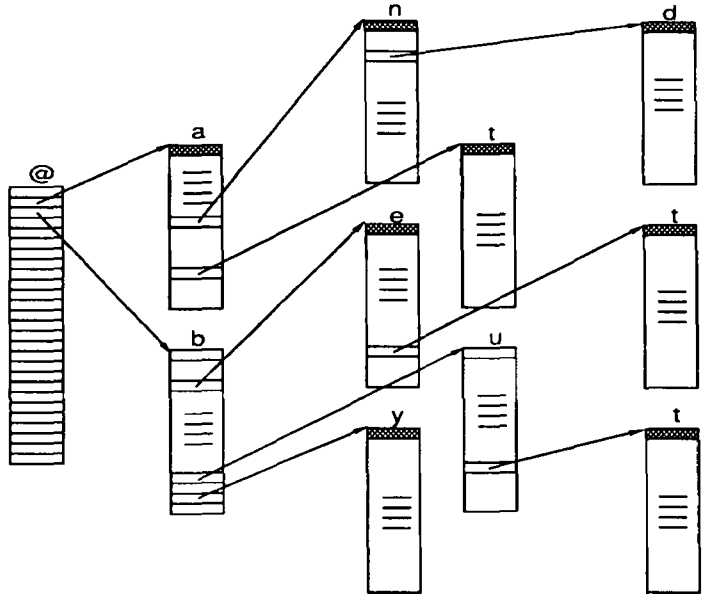


Figure 6.5 - Dictionary tree - simple implementation

A more compact dynamic structure is shown in figure 6.6. In this diagram each large square represents a node of the tree consisting of a key letter, at a certain level, and two pointers. The upper pointer points to a list of those letters, at

the next level, that are permitted to follow this letter. The lower pointer points to the remaining list of letters at the current level which can occupy the same letter position.

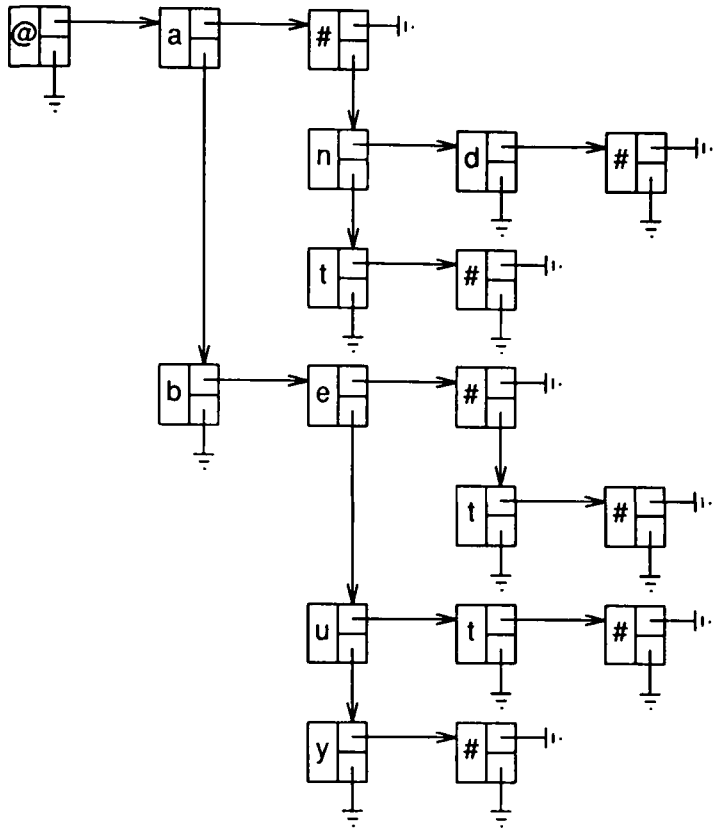


Figure 6.6 - Dictionary tree - compact implementation

Figure 6.7 shows the implementation for a letter graph which complements the dictionary tree implementation above. These two data structures can be traced simultaneously and implemented efficiently with very similar code. In the diagram a node consists of a letter and a pointer to a list of arcs which, in turn, point to possible successor nodes.

6.5.2. Possible Enhancements

The final choice of internal representation involves consideration of the trade off needed between access speed and memory requirements. There are several different structures which can be used depending on the most important criterion.

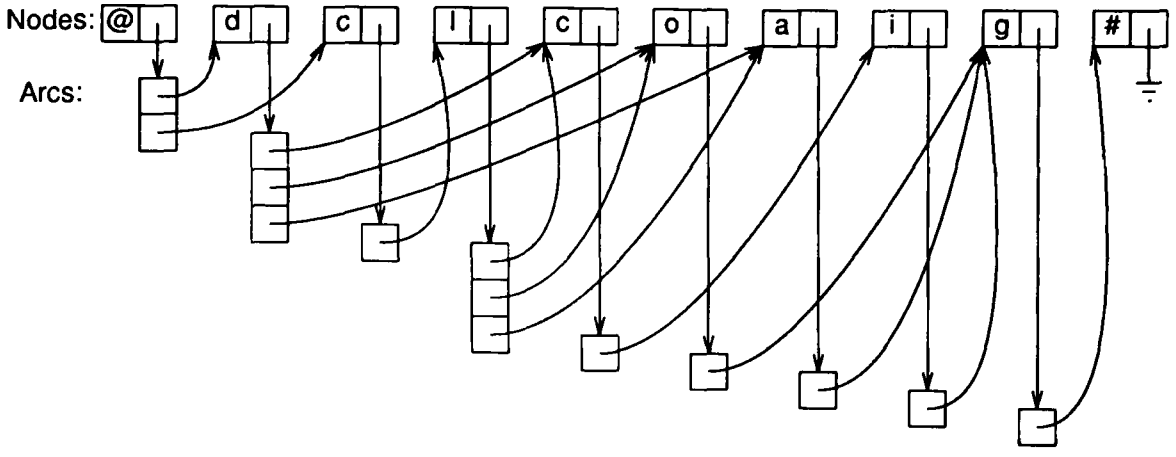


Figure 6.7 - Letter graph implementation

6.5.2.1. Speed

The list of letters, at the next level, which can follow a particular letter in the dictionary tree is defined as its *sub-list*. The search method for a structure similar to figure 6.6 can be speeded up by reducing the time taken to search for a letter or *search-key* in a sub-list. The sub-list of letters can be ordered in several ways. If the letters are in a random order, it is necessary to search to the end of the list if the search-key cannot be found. However, if the ordering is known it is only necessary to search as far as the expected position of the letter.

Several different orderings are possible. Alphabetical ordering has the advantage of allowing the ASCII code to imply position in a list. Sorting on frequency of occurrence in English, or frequency of generation from the recognition system may produce a faster search, but may require extra memory to store a representation of the order. Actual speed will be dependent on the letter graph itself; a graph which contains many valid words will take a different amount of time to trace than a similarly sized graph with many invalid words. For example, a list optimised for finding a commonly occurring letter will not be optimised to show that an infrequent letter is not present - in fact, it will be the worst possible ordering.

6.5.2.2. Size

The size of the data structure can be reduced by using *tail-end compression.* If the end of a word is unique and is not common with any other word, then the dictionary structure can be reduced by constructing a special *end-of-word* node which contains the rest of the word as a string. This removes the need for extra nodes and pointers (see figure 6.8). The disadvantage of this structure is that the dictionary cannot easily be maintained. For example, to add an extra word may involve unpacking the tail of an existing word, adding the new word, then repacking the two new tails. Also, since the code needed to search a tail will be different to that needed to search the tree, switching lookup code would degrade the speed of lookup.

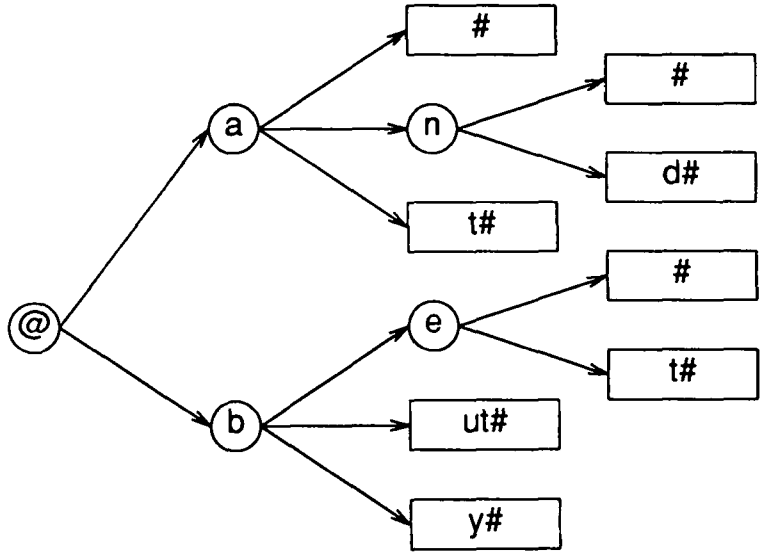


Figure 6.8 - Dictionary tree - tail end compression

6.5.2.3. Recognition

This technique was originally developed for use as a post-processor for a cursive script recognition system. Often handwritten script samples have very poor quality sections where either the word has been written badly, or has been poorly digitised. A recognition system may not be able to identify, in any way, what has been written in these sections and may insert a *wild card,* representing every possible character, into the letter graph in that place. Alternatively, it may be clear that a segment has, for example, a descender, but no other information is

obvious, so a *descender subset wild card*, representing every letter with a descender, could be inserted. If the poor section occurs at the end of the word, the performance of the tree structure described above will not be adversely affected, since most of the paths through the letter graph will be rejected before the highly ambiguous section is reached. However, if it is at the beginning of the word, then there will be a large amount of ambiguity at the beginning of the graph, resulting in a large amount of wasted computation.

If the dictionary tree was also structured in reverse, starting at the end of words and working towards the beginning, it would be possible, in these cases, to verify the letter graph backwards, and so speed up the process. Similarly, if both ends of the original data sample are poor, it may be possible to work from the middle outwards. An ideal data structure might have multiple linkages, starting at each letter position, so that the dictionary search would always begin at the least ambiguous part of the graph. It is intended that the properties of such a structure might be investigated in future implementations but the data structures will be very large.

It should be noted that a dictionary lookup system based on a letter graph will fail if the correct letter is not in the letter graph - hence the need for wild-cards when no letters are obvious candidates. It is not always clear where the wild card should be inserted, however (see section 6.11). A fixed letter-segmentation recognition system does not suffer from this problem since it is possible to change one letter to another.

6.6. Implementation Selected

While the suitability of the data structure was assessed, the implementation selected for ORCHiD used a simple approach, for ease of coding. The data structure is an exact representation of figure 6.6. Lists are ordered alphabetically, and the tree is linked only from start to finish. After verification that the method was suitable, it was intended that it would be refined but it was discovered that the performance of this prototype was more than adequate for the system. The size of the data structure was within the limits of available memory, even with a large dictionary, and the execution time was fast, even for very large letter graphs. (Some of the more complex generated test graphs could be checked in much less than one second, but the number of paths could not be counted even with several hours of computing time.)

To test the performance of this implementation, a 210 000 word dictionary was used. This is probably much larger than would be needed in a practical system. A substitution set was used to generate simulated letter graphs of possible output from typed input words. A test set of 2100 letter graphs was verified using the system running on a Sun 3/160 with 8 Mbytes of main memory, timeshared with other users. The average real time needed per word was 0.45 seconds, and the actual CPU time used by the system averaged 0.19 seconds per word. In a practical system the lookup process would be quicker, since the times quoted above include routines to set up the data structure for the dictionary, to time the program and to calculate and print various analytical results.

6.7. Size of Data Structure

The size of the dictionary data structure can be measured by counting the number of nodes in the tree. Experiments have been carried out using different sized word lists to investigate the effect of dictionary length against data structure size. As can be seen from the graph in figure 6.9, this is roughly a linear function of dictionary size.

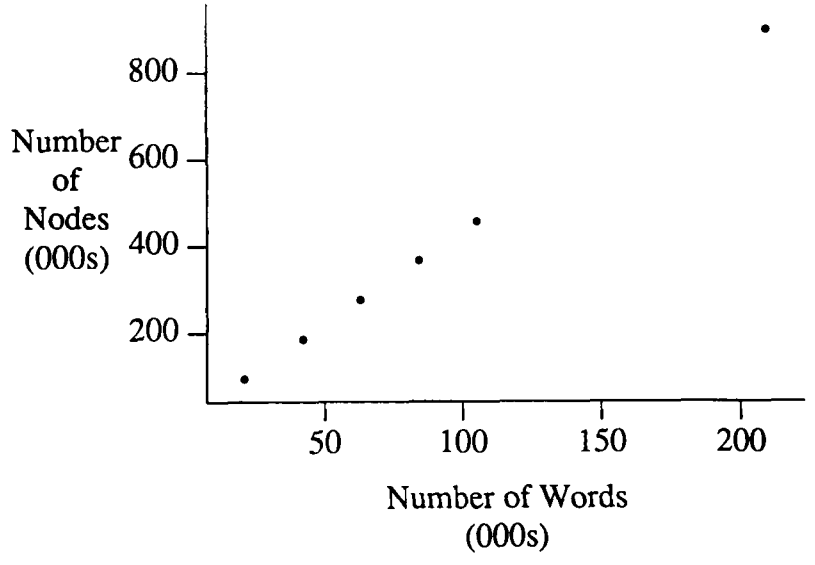


Figure 6.9 - Size of data structure *vs* dictionary size

It seems that a dictionary of 60 000 words will probably be sufficient for a usable script recognition system. This would allow for a vocabulary of about 20 000 root words with plurals and verb endings. (The Lancaster-Oslo/Bergen corpus of British English[46] consists of samples of everyday text, with a total of 5

million words. 50 000 individual words were identified in these texts.) Assuming that a node consists of a character and two pointers, it can be seen from the graph that such a dictionary would require approximately 2.4 Mbytes of memory in our implementation. This is not an unreasonable amount of memory to expect in modern computer workstations.

The size of the data structure for a particular dictionary is dependent on the *compactness* of the dictionary, ie whether the words have common roots. Adding a word which has the same root as a word already in the dictionary increases the size by less than adding a unique word since fewer extra nodes are needed. This means that the dictionary can contain all participles of verbs, plurals etc without drastically increasing its size, avoiding the need to construct these words by a rule-driven system.

To investigate the effect of compactness on dictionary size, two sub-dictionaries of 21 000 words were selected from a word list of 210 000 words; one by selecting the first word from each group of 10, the other by selecting the first 10 words from each group of 100. As expected, the first of these was less compact and used 181 138 nodes, the other more compact dictionary used only 98 981 nodes.

6.8. Comparison with Binary 4-Gram Graph Reduction

The graph in figure 6.10 shows the time taken for the graph reduction and the dictionary lookup, plotted against confusion level of the input graph, again for the word "test". It can be seen that with very small graphs the two techniques are comparable, but, as the ambiguity increases, the dictionary lookup is considerably faster. It should also be remembered that the 4-gram output still needs to be verified against a dictionary.

The ORCHiD template matching routine, described in Chapter 5, retains a large amount of ambiguity within the letter graph it produces. This often produces a confusion level of 10 or more. It can be seen that with this level of confusion the dictionary tree method is far superior to the 4-gram method.

6.9. Implementation Details

A few practical details of the implementation are discussed below, together with the differences between the theoretical and the actual implementation. The template matching stage produces a graph of candidate allographs, but does not

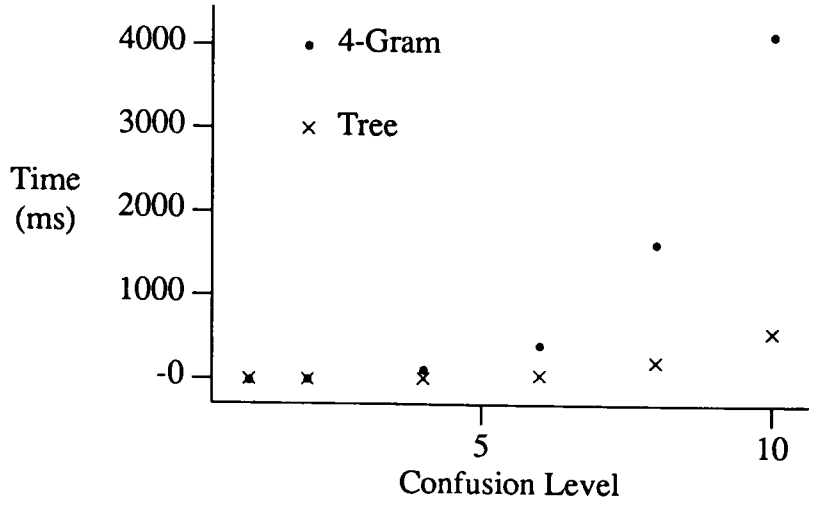


Figure 6.10 - Comparison - lookup time *vs* confusion level

produce wild-card entries if the quality of match is poor.

6.9.1. Letter and Letter-Join Graph

The graph produced by the recognition stage contains letter-joins as well as letters on the nodes of the graph. This greatly increases the size of the graph. It is possible to construct the dictionary tree including letter-joins and simply use the lookup algorithm as it stands. This, however, would greatly increase the memory requirements and the system would become unwieldy. It would also be difficult to incorporate the contextual information indicating which allographs are permitted to connect to which other allographs.

A template-join table is used to identify which templates are allowed to connect to which other templates. For example, some letters may join directly to other letters without any linking stroke and some letters must join to other letters with specific letter-joins. As the graph is traced, the table can be consulted to see if two templates can be joined, and if so the dictionary tree can be checked.

To further improve the efficiency of the dictionary lookup stage, however, the template-join table is checked as the letter graph is being constructed, rather than as it is being traced. In this way, the table need only be accessed once for each arc of the graph, rather than every time the arc is traversed. If a letter template can link to another template already on the graph, then it is placed on the graph. If a letter-join template can link to a letter template on the graph, and can

be followed by another letter, then the probability weight for the letter-join is placed on the *arc* joining the letters. In this way a true letter graph is constructed, with additional probabilities for the connecting strokes located on the arcs of the graph.

6.9.2. Multi-Pass Dictionary Search

The dictionary search routine is applied to the letter graph and the output automatically examined. If all candidate words produced have a probability below a threshold then the dictionary search routine is applied again with wild-card letter-join templates inserted between each letter. These wild-cards are used since letter-joins often contain few distinguishing features and are sometimes not identified at the recognition stage.

The use of two passes improves the speed of the system, since most words are identified in the first pass, yet allows for greater accuracy if no word is an outstanding match.

6.9.3. Diacritical Marks

The information about diacritical marks, such as dots and crosses, stored during the preprocessing is used to reduce the output list and increase the accuracy of the system. Firstly, any letter that *requires* a dot or cross is checked to see if such a mark is nearby in the sample word. If not, then that letter is removed from the letter graph before the dictionary search takes place. Secondly, the candidate word list produced by the dictionary lookup is checked to ensure that every word contains the same number of diacritical marks as were detected in the script.

This information can only be used with care since most people are not particularly accurate in the location and formation of these marks, if they remember them at all. A number of common problems can be identified.

- Frequently a dot will become extended into a line, or a line will be very short and recognised as a dot. It is therefore not possible to insist that a 't' has a line through it and an 'i' has a dot above it.
- The diacritical mark will often not be directly above its corresponding letter. It is important to allow some tolerance in the position of these marks.
- A double 't' within a word is usually crossed with one stroke. Allowance must be made for this.

Different writers place marks on different letters. Some cross an 'f', for instance, while others do not make a separate line through a 't'. For this reason, parameters can be passed to the ORCHiD system to indicate the level of importance to be placed on these marks depending on the writer.

6.9.4. Word Frequency

A dictionary of 60 000 words is used for this system. This provides a reasonable vocabulary but includes a number of less common words which may appear in the output list. In order to improve the accuracy of any recognition system it seems reasonable to increase the weight associated with those items that are more likely to be presented to the system. We know the frequency of word occurrence in English from a number of studies[46,57]. We can therefore attempt to incorporate the *a priori* probabilities of a word occurring into our statistical probability produced by the recognition system.

6.9.5. Reduction of Candidate Word List

The candidate word list contains a number of words ranked by their probability. A practical CSR system, which could be incorporated within other applications, will need to present the user with either a single word, or a very small number of options (if it is not possible to decide between them). It is therefore necessary to greatly reduce the output list to just a few words.

Words that are similar or ambiguous in appearance are recognised with similar probability weightings, and so appear close together in the ranked list. The top words in the list are examined. If the probability associated with the top word is considerably higher than its nearest contender, that word is presented as the correct word. If, however, the probability of the next word is relatively close to the top word, then a group of words is presented. The group consists of all those words at the top of the list whose probabilities are close to their neighbours.

6.10. Higher Level Context

It is possible to apply even higher level context to further improve recognition rates. The candidate words can be compiled into a *word graph* which represents an entire sentence or phrase. Sentence and phrase semantics and syntax can be applied to reduce this word graph to a smaller list of *candidate sentences*, which can be ranked in the same way as the letter graph produces a ranked

list of candidate words. This technique can also be used to remove word segmentation ambiguity that can arise in some handwriting styles which have very separated characters.

This context is not within the scope of this thesis but will be further discussed in Chapter 9.

6.11. Conclusion

Whereas a large amount of work has been done in the field of contextual post-processing of text recognition systems, not many of the techniques suggested successfully address the problems caused by ambiguous letter-segmentation, common to cursive script recognition. The tree-based dictionary lookup technique described here is an ideal application of letter and word context for verifying the output from text recognition systems and allows for ambiguous letter-segmentation of the script. It is extremely fast and efficient, can be used with a large dictionary or word list and produces all possible output words, with no loss of information that can occur when arbitrary cut-offs are applied. The dictionary contains every word which can be recognised by the system, without the need for prefix and suffix generation. It is simple to add extra words to the dictionary without greatly increasing its size. The technique is especially appropriate for very ambiguous letter graphs, typically produced by cursive script recognition systems.

The performance is superior to n-gram graph reduction techniques, and is in fact a superset of the n-gram approach, since all possible values of n are effectively applied during the recursive traversal.

An efficient implementation of the dictionary tree has been produced. Currently the output is presented to the user, along with the image of the preprocessed word, for verification. This verification can then be used to further train the system (see Chapter 7).

In theory, wild card substitution can be used in areas where the script is very poorly written or badly digitised. These can be of great value and help to improve the recognition rates, especially where there is a small area of uncertainty within a script word. In this implementation, wild card letter-joins are inserted into the letter graph if no reasonable candidate words can be found from the original graph. Wild card letters are not used, however. With a sub-letter segmentation method, as described in Chapter 4, it would be necessary to insert a

wild card segment and then relate that segment to candidate allographs that contain it. This is a non-trivial task.

Chapter 7

Training

This chapter discusses the methods used to train the ORCHiD system, either to a single user or to multiple users. The system can be automatically trained with minimal human intervention. There are a number of problems associated with automatic training however and these are discussed below.

7.1. The Need for Training

As has been discussed in section 1.3, a completely untrained cursive script recognition (CSR) system is desirable but very difficult to develop. To produce a practical system with maximum recognition rates and minimal adjustment of writing style on behalf of the user, a recognition system that can be trained in some way is necessary. A trained system has the added advantage that it can also more easily allow for more unusual writing styles.

The problem with a trained recognition program is that there is a considerable period of time while the training process is being completed during which the system cannot be used. This problem is especially acute if CSR is the only data input method within a larger computer system. With no other method of data entry the whole system is inoperable until training is complete. Other factors must be considered with a trained system. If the training period is lengthy the user may get frustrated and reject CSR in favour of another data entry method. If, however, the training session is not sufficiently comprehensive, the recognition accuracy may not be acceptable to the user, who might again reject the system in favour of a more reliable method. ORCHiD attempts to solve this problem by supplying a system that can be used almost immediately, yet which can also be trained whilst it is in use, if desired to give improved recognition rates.

7.2. Automatic and Manual Training

With the majority of trainable separated character recognition systems a training session is performed before the system is used. This training session requires the user to supply a number of samples of each character that the system is to recognise. There may, however, be a method for on-going adaptation by adding characters to the training set while the system is in use. This is a *manual* training approach since the user defines the structure of a character.

A CSR system cannot easily be trained in this way. Either the user would have to supply samples of every word which might be written, or indicate to the system where each letter starts and finishes within a cursive word. The first approach is clearly impractical for systems which must recognise words from a large vocabulary. The second approach relies heavily on the user specifying the letters correctly. Since the definition of letters, in terms of the recognition system, is heavily dependent on the underlying recognition model, such as segmentation and feature extraction, this requires the user to have detailed knowledge of how the system works. This is clearly impractical for anything other than a development system.

It may be possible to request the user to provide samples of each letter separately and build a CSR system trained on these samples. The problem with this approach is that most people write characters differently in cursive writing to separated hand-printing and even form letters differently in different parts of a word. Observations have shown that this inconsistency occurs even if the user is consciously attempting to write uniformly.

An *automatic* training system, which decides on where each individual template lies within a word and trains it accordingly, greatly simplifies the training as far as the user is concerned. If desired, it also allows for continuous training all of the time that the system is being used.

7.3. Theory of Automatic Training

In statistical terms, the training of a template is equivalent to estimating the parameters (mean and standard deviation) of the distribution that the template represents. Bayesian theory shows that with increasing numbers of samples these parameters can be estimated more accurately. In order to do this, however, the samples must be correctly identified, ie the system must be *manually* trained. The theory associated with this *supervised learning* is then straightforward[20].

As discussed above, an *automatically* trained system is preferable. This is referred to as *unsupervised learning*, since the precise classification of the samples presented is not known prior to training. The theory underlying this approach is complex, especially if no *a priori* assumptions are made about possible classification of the samples. A *decision-directed* approximation is commonly used where an automatic classifier attempts to label the samples[20]. These labels are then used to train the classifier itself. Further samples are labelled by the classifier and the training is repeated until sufficient recognition accuracy is obtained.

There are two main advantages to this approach. Firstly the classifier can be quickly designed using a small number of labelled samples and then allowed to "fine-tune" itself on a much larger set of unlabelled samples. Secondly, if the system is to recognise patterns that may evolve slowly over a period of time, it can be constantly updated and honed.

The main drawback of such a system is that a classification error will cause the *classifier* to be incorrectly trained. There are two possible outcomes from this mistake - either sufficient samples will be correctly identified to "cancel out" the incorrect sample (on average), or the error will be more likely to occur and further contaminate the classifier. The only way to reduce the likelihood of this error occurring is to ensure the initial classifier is reasonably accurate.

Despite the possible problems with the decision-directed approximation, most experimental evidence shows that this procedure works well in practice. The ORCHiD training stage is an implementation of such a technique. The initial classifier uses of a set of prototype or general templates to label samples, and these templates are modified during the subsequent training period. The system described by Teulings *et al*[94], however, does not require a general template database since it generates the prototype templates automatically by judicious selection of training words (see section 2.9).

7.4. Training the Personal Template Database

As described in section 5.5, the ORCHiD template database consists of statistical information about the population of samples presented to the system. This statistical information consists of the sum of feature values, the sum of squares of feature values and the number of samples so far included. From this information it is possible to calculate the mean and standard deviation of the feature values to

be used during template matching. To adjust a template during training simply requires the addition of a sample or samples of the template to the template database, and the relevant statistical calculations. The complex part of the training stage is therefore the automatic identification of templates and their location within the script - the *allograph segmentation.*

The training routine used utilises the same basic procedures as the recognition system so that training can be carried out during use. The writing is collected, preprocessed, segmented and the features extracted in exactly the same way as during recognition.

Two template databases can be accessed during the training routine - the user's personal database and the general database. By superimposing the personal database on top of the general database, a complete set of possible templates can be constructed. The sample features are matched against this combined database using the same algorithms as the recognition stage, but with less severe thresholds so that more candidate allographs are produced. These allographs are constructed into a letter graph, as in section 5.7.3.

The correct classification of the script word (the *identified word*) is known to the training routine - either it has been supplied by the user during the recognition verification stage, or the system prompted the user to write that particular word during an initialisation process. The training routine can now apply this information to identify and locate the correct templates within the sample word. The dictionary lookup routine described in Chapter 6 is applied to the letter graph but with just one word in the dictionary tree - the identified word. The highest weighted path through the graph is assumed to be correct allograph segmentation and the templates are thus trained.

7.5. The Training Session

There are two stages to training the system. The first stage involves the system selecting the particular styles of character formation that the writer uses from a generalised set of characters. This subset is then further trained to the individual during the second stage of training by adjusting the template feature values.

7.5.1. User Initialisation

During the user initialisation stage a personal database is constructed that contains a template or templates for every letter of the alphabet.

The system prompts the user to write a small number of words that contain every character in the alphabet. These sample words are preprocessed, segmented and features extracted. Candidate allographs are produced using the general template database and a letter graph is constructed. *Wild-card* letter-joins are inserted at each segment of the sample word to ensure that templates are not missed. The allograph segmentation is identified in the way described above using the training word that the user was requested to write in the dictionary tree.

The correctly identified templates are now copied across to the personal template database and trained with the new sample data. The statistical information stored in the general database is scaled during copying so that it appears as if only a few samples have been used to train the template. In this way the new sample data, specific to the user, will quickly influence the average feature values of the template.

Any words for which the system has failed to identify the allograph segmentation correctly are requested again from the user, until the system has recognised each word at least once.

7.5.2. Continual Training

Once initialisation is complete and the user's personal database contains at least one template for each letter, further training of the templates can take place. As well as improving the letter templates, the letter-join templates are identified and trained during the continual training process.

The user writes any number of words from the dictionary. The recognition system is applied to these sample words. The user is shown the results and asked to verify the correctly recognised words, or type in any corrections for erroneous recognition. The system then reapplies the recognition routines using the personal and the general database combined to produce the letter graph. The correct allograph segmentation is identified using the information provided during verification and the personal database trained accordingly.

At present this recognition system is a stand-alone demonstrator, but this method of verification and continual training could be incorporated into any system without the user being unduly aware of it. For example, as soon as the user

has accepted and saved a page of recognised text this can be assumed to be verification and training can take place.

7.6. Success of Training

The automatic training routines work effectively, and greatly simplify the training process. There are, however, two main problems with this automatic training approach.

The problem associated with decision-directed learning, described in section 7.3, occurs within this system. Allograph segmentation is not always correct and consequently templates are incorrectly trained. Observation of the templates over a period of training shows that the errors are usually cancelled out by correct training, but occasionally a spurious template is produced. There are no simple cures to this problem other than to delete the adulterated template from the database when it has been identified. This identification is difficult during normal usage, especially for the naive user. Section 9.3.1 also discusses this problem.

If a template is not in the general or the personal database, perhaps because it is an unusual allograph, then it must be identified and trained manually. This is a difficult process for the untrained user, as has been mentioned above. A solution might be to investigate the fully automatic approach of Teulings (described in section 2.9) that requires no initial knowledge about character formation. The drawback with using this technique for training is that a much longer training period is needed before the system can be used. Perhaps a hybrid method, combining these two techniques, would provide the best remedy, where the general database approach is used to train most letters, and Teulings' approach for unrecognised letter formations.

7.7. Conclusion

The simple format used to store templates for letters and letter-joins provides for an easily trained recognition system. An unusual allograph can be catered for by incorporating several samples into a new template. Other symbols could be trained other than lower case characters, for example capital letters, digits etc.

The automatic allograph segmentation enables the system to train itself without any intervention by the user. This is not without its own problems, notably the problem of incorrect classification described above, however these do not

seem to seriously affect the final results of the training. Some possible enhancements to the training routine are discussed in section 9.3.2.

Chapter 8

Results

This chapter contains details of performance tests carried out on the ORCHiD system. Details are provided of recognition rates and speed of execution of the system, together with an analysis of these results and a discussion of some of the causes of errors.

8.1. Discussion of Performance Testing

There are a number of problems associated with adequately testing the recognition performance of a CSR system. The most important problem to overcome is the collection of a sufficiently large amount of high quality data to provide significant results. Any volunteer providing sample data must be prepared to spend a considerable length of time writing with the equipment. This is especially the case with an interactive trainable system since the data collected must be processed before collection of the next set of data. This lengthens the collection process.

To test a trained system in a reliable way, it is necessary to collect two sets of sample data, one for training and one for testing. Some research papers quote results where the test set was also used to train the system. This is unreliable, since a practical CSR system will always be required to recognise unseen script.

8.2. Experimental Details

Three sets of tests have been carried out. These check the performance of the system with no training, after minimal training to a particular writer using a small sample of writing, and after a more extended training period.

8.2.1. Hardware

Samples of handwriting were collected using a Pencept Penpad opaque digitising tablet, connected to a Sun 3/140 via a serial line. This was selected as the input device since it has the least intrusive and most accurate pen action, and

provided a familiar feel to the writer.

Special forms were used to collect the data (figure 8.1 is a reduced image of a completed form). These were ruled with baseline, full-height-line and halfline to assist the NPL preprocessing software by ensuring some uniformity. Each word was read to the subject one at a time, at a reasonable pace, to ensure a natural handwriting style. The subjects were asked to write relatively neatly and an example line of script was shown at the top of the collection form for guidance.

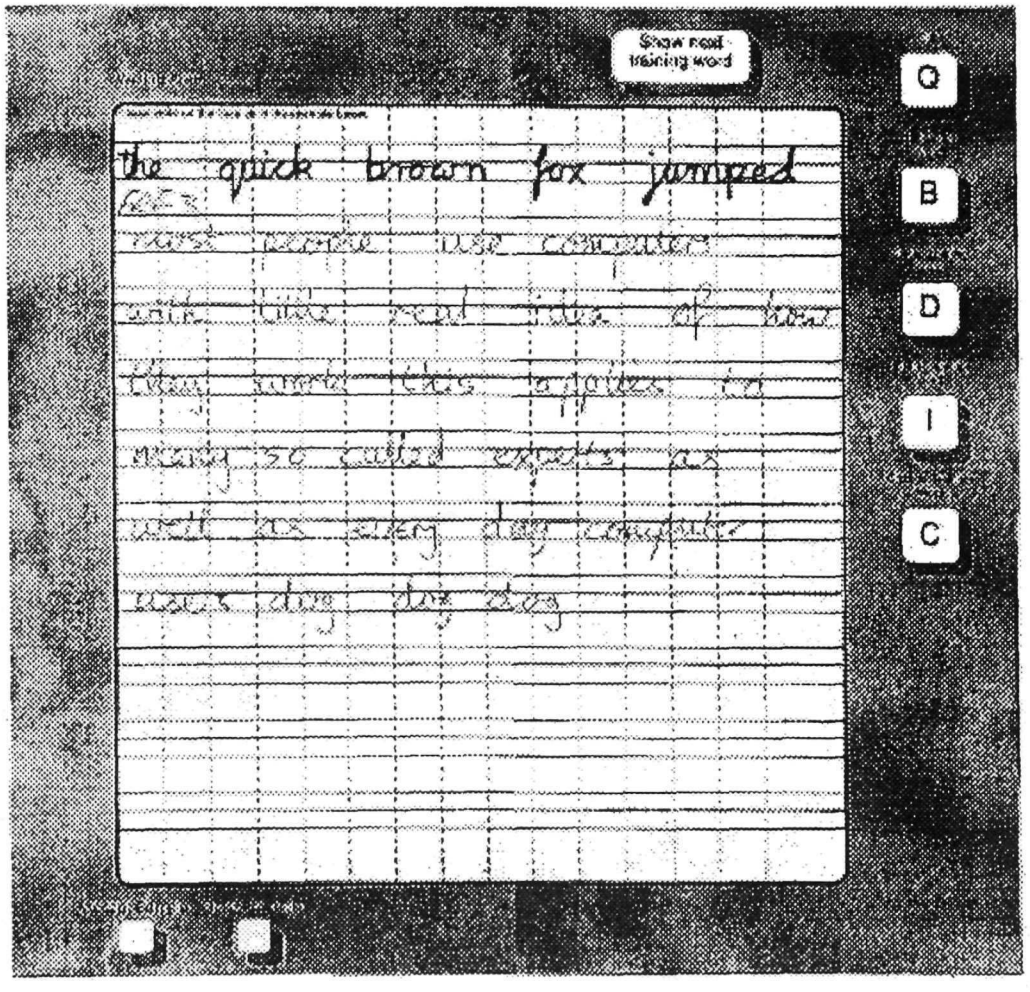


Figure 8.1 - A handwriting sample

The data was collected, inspected, preprocessed and the preprocessed data verified and corrected on a Sun 3/140 workstation. The recognition processing was carried out on a Sun 4/330 fileserver.

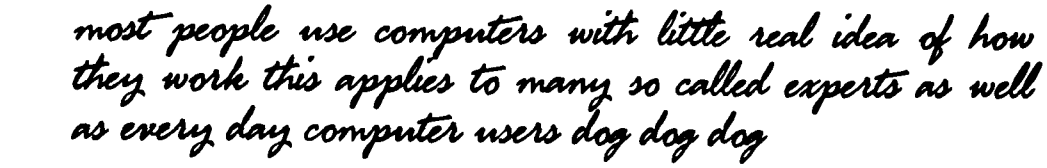
8.2.2. Samples Collected

Four samples of handwriting were collected from eleven writers. The samples consisted of two sets of words which were written twice.

Samples A1 and A2 consisted of the words :-

the quick brown fox jumped over a lazy dog pack my box with five dozen liquor jugs

Samples B1 and B2 consisted of the words



most people use computers with little real idea of how they work this applies to many so called experts as well as every day computer users dog dog dog

Sample set A was chosen to include every letter of the alphabet at least once. Sample set B provided a list of more common words within the context of a sensible sentence structure. The repeated word 'dog' was used to check the consistency of the segmentation and the normality of the extracted features (see sections 4.5.1 and 4.7).

8.2.3. Template Databases

The general template database has been constructed by presenting large numbers of handwriting samples to the system from a variety of writers. This database is probably not exhaustive, but includes many of the more common writing styles. Personal databases have been constructed and trained from this general database.

8.2.4. Verification of Preprocessing

This was carried out by hand before recognition. Errors in preprocessing, such as incorrect word segmentation or incorrect baseline identification, were corrected using the verification software. Any poorly digitised images were discarded.

The consistency of occurrence of diacritical marks varies greatly between users. The writers were not forced to make these marks, since this would affect their natural flow of writing. For this reason the information provided by the diacritical marks was not used in the first two sets of tests.

8.2.5. Definition of Successful Recognition

The only *true* definition of successful recognition must surely be that the correct word is identified and is the most strongly weighted of all of the candidate words. As has been explained in previous chapters, the ambiguity of cursive handwriting implies that 100% recognition is unlikely for single words taken in isolation, even for human readers. For this reason, we define successful recognition to be that the correct word appears towards the top of the ranked candidate word list. This is a common practice for workers in this field.

Results are quoted for successful recognition if the correct word is within the top ten words of the candidate word list, together with some results for recognition in the first position.

8.2.6. Dictionary

A dictionary containing 60 000 words was used to verify the output. This contained most of the words likely to be written, including plurals and verb conjugations as well as word roots (see section 6.7).

8.3. Test Routines and Results

As stated in section 8.1, it is very costly and time-consuming to gather large quantities of data to exhaustively test an interactively trained system. It is therefore necessary to devise tests that will show the trends of such a system and then prove its potential performance with a smaller data set. The tests below show the performance of the untrained system on a large quantity of data (test I), the trend in performance for the system after minimal training (test II, III and IV), and the performance after further training (test V). Since the trained system requires some of the sample data for the training process, the later tests are, of necessity, based on smaller test samples.

For each test, details are given below of the test sample, the training sample and the template database used. The expected results are briefly discussed, followed by the observed recognition rates. More specific details for sample B2 are also given, which are discussed later. An analysis of all of these results is given in section 8.4.

8.3.1. Test I - Untrained Recognition

The recognition system was applied to all of the samples using the general template database. This shows the average recognition rates for the system with no training. There will be a wide variety of rates between different samples depending on the number of non-standard letter formations used. Clearly if a subject uses an unusual allograph that is not in the general template database then the system cannot recognise a word containing that allograph.

Sample	Rate(%) for writer											Average
	1	2	3	4	5	6	7	8	9	10	11	rate(%)
A1	94	76	53	59	65	73	71	76	44	71	82	**69**
A2	82	82	53	53	76	59	82	77	50	71	56	**69**
B1	89	85	63	81	67	78	85	96	72	85	85	**80**
B2	85	89	67	78	72	77	85	81	63	81	81	**78**
Overall	88	84	59	70	70	73	82	85	60	78	78	**75**

Table 8.1 - Test I results

For sample B2, the average position of the word in the output list was 1.509, and the correct word was top of the list 61% of the time.

8.3.2. Test II - First Training Session

A personal template database was created for each subject. This was created automatically by the system (Chapter 7) using sample A1 to identify and train each template. The database was then amended as necessary by hand where the automatic training system had not succeeded in identifying a template (section 7.6). This minimally trained system was applied to samples A2, B1 and B2.

Test II provides recognition rates when a personal database is used that has been trained once on a limited set of letters and letter-joins taken from sample set A. The recognition rate will be *reduced* compared with test I, since a number of the writer's templates will not have occurred in the initial training set and so will not have been transferred to the personal database. This will be especially apparent for samples from set B where the particular letter-join templates will not be included in the database.

Sample	Rate(%) for writer											Average
	1	2	3	4	5	6	7	8	9	10	11	rate(%)
A2	89	76	53	59	59	53	76	62	38	65	63	**63**
B1	48	56	44	48	26	63	56	59	36	59	63	**50**
B2	52	44	44	67	20	54	48	70	44	59	65	**51**
Overall	59	56	46	58	32	57	58	64	40	61	64	**53**

Table 8.2 - Test II results

For sample B2, the average position of the word in the output list was 1.790.

8.3.3. Test III - Second Training Session

Sample A2 was used to further train the personal template database for the subject. The training set was created automatically but *not* adjusted by hand. The system was then applied to samples B1 and B2.

Test III provides recognition rates when a personal database is used that has been further trained using a limited set of letters and letter-joins taken from sample set A. There may be a slight improvement on the results of test II, due to improved training of the letter templates, but the letter-join templates used in set B, but not set A, will still not be incorporated into the personal database.

Sample	Rate(%) for writer											Average
	1	2	3	4	5	6	7	8	9	10	11	rate(%)
B1	52	59	59	59	52	70	67	67	44	59	78	**61**
B2	59	52	59	67	36	62	52	70	48	59	73	**58**
Overall	56	56	59	63	44	66	59	69	46	59	75	**59**

Table 8.3 - Test III results

For sample B2, the average position of the word in the output list was 1.688.

8.3.4. Test IV - Third Training Session

Sample B1 was used to further train the personal template database for the subject. The training set was created automatically but *not* adjusted by hand. The system was then applied to sample B2.

Test IV provides recognition rates after a training session with a set of words that contain the same letters and letter-joins (sample B1). The rates should now be considerably improved on test III since most of the letter-joins used in set B should now be incorporated into the personal template database. This improvement in recognition should continue with further training.

Sample	Rate(%) for writer											Average
	1	2	3	4	5	6	7	8	9	10	11	rate(%)
B2	85	85	63	78	76	73	70	96	63	78	85	**78**

Table 8.4 - Test IV results

For sample B2, the average position of the word in the output list was 1.462, and the correct word was top of the list 64% of the time.

8.3.5. Test V - Extended Training

A number of subjects interactively trained the system for a short period of time. Sample set B2 was used to test the accuracy of the system after this training session. Diacritical mark information was used by the recognition system if this was consistent for that writer.

Test V provides recognition rates after a further training session. The recognition rates should now be much improved on untrained recognition. The personal database for writer 1 was trained using a much larger collection of data and gives an indication of the performance that might be expected of the system when in constant use. Training should not only improve the likelihood of the correct word appearing in the output list but should also improve its position within the list, with the ideal being that the correct word should appear at the top.

Table 8.5 shows recognition rates for four subjects, both with and without training. The occurrences when the correct word was top of the output list are also shown.

Writer	Untrained system		Trained system	
	Top 10 rate(%)	Top word	Top 10 rate(%)	Top word
1	88	74	96	85
2	84	72	89	78
10	78	67	89	78
11	78	58	88	81
average	82	68	**91**	81

Table 8.5 - Test V results

The average position of the correct word was 1.240.

8.4. Analysis of Results

The results from test I show that the average untrained recognition rate for the system is 75%, though there is quite a considerable variation between different subjects (59-88%).

To compare the recognition rates before and during training it is necessary to consider sample B2 alone, since this is the only sample used throughout all of the tests (table 8.1).

Test	Rate(%)	First position rate(%)	Ave position
I	**78**	61	1.509
II	**51**	39	1.790
III	**58**	44	1.688
IV	**78**	64	1.462
V	**91**	81	1.240

Table 8.6 - Recognition rate comparison - sample B2

Table 8.6 shows that the untrained recognition rate is reasonable (78%). The recognition rate drops, as expected (section 8.3.2), between tests I and II, but with the introduction of training from samples taken from set B1, with the same words as set B2 (test IV), the rates climb considerably, back to the level for untrained recognition. (Figure 8.2 shows the individual results for each subject.) In fact the rate for 1st position recognition and the average position are improved on the untrained rate showing that the recognition is improving with training, as expected.

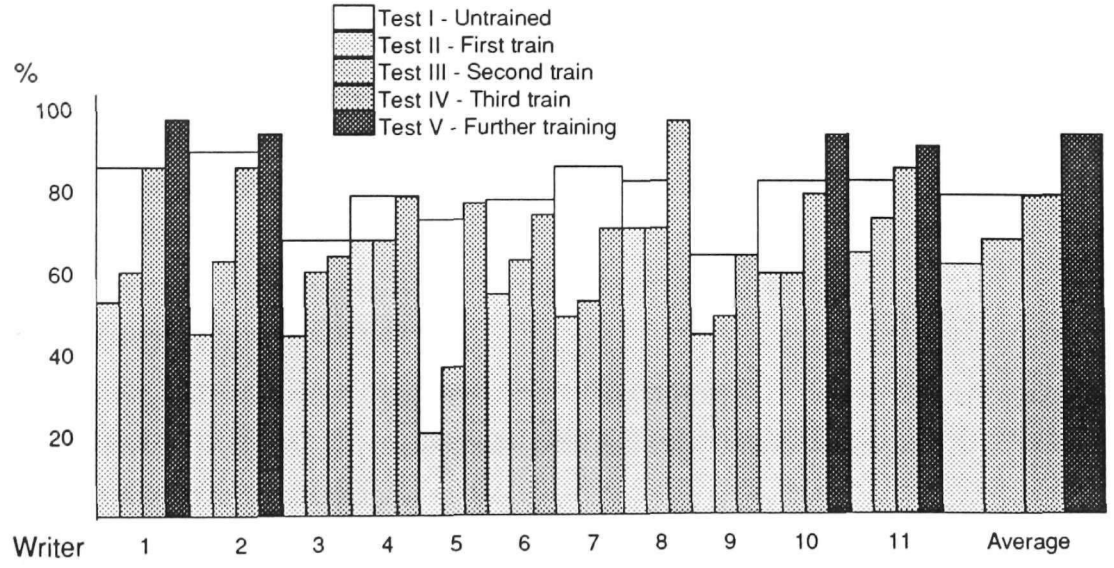


Figure 8.2 - Recognition results - sample B2

After some further training, the accuracy improves considerably (test V) to produce 91% recognition on average. This rate can be improved further with extra training. The template database for writer 1 was trained for a much longer period and achieved 96% recognition for sample B2, with 85% recognised in first place.

These results appear to follow the expected trends discussed in sections 8.3.1-5. The system can achieve good recognition rates after just a small amount of training and very good rates with further training. The exact performance of the system, however, is dependent on the consistency of the writing style.

8.5. Description of Errors

Recognition errors occur due to a number of reasons. These are described below and the occurrence rates are given in section 8.6.

8.5.1. Stylistic or Writing Errors

i) Within a sample of a subject's handwriting it is common for mistakes to be made during word formation, especially in longer words. With a segmentation based system, which looks for individual letters within a word, it is very difficult to allow for these errors. This may not be the case with a whole word recognition system, say, which looks at the overall shape of the word, rather than carefully analysing each section.

ii) Some writers have pronounced stylish flourishes within their script. These can be trained into the ORCHiD system, but will not be present in the untrained system. Those samples that showed a marked improvement in recognition after training (figure 8.2, writers 5 & 8) generally featured such flourishes.

8.5.2. Preprocessing Errors

iii) The results of the preprocessing phase are verified before the recognition phase is initiated but occasionally incorrectly preprocessed script is passed on due to human error.

iv) The algorithm for joining separated strokes within a word is very elementary - the individual strokes are simply connected by a straight line. This is sufficient for most purposes, but with very separated handwriting styles this is an additional source for errors to occur. Ideally, the script should be

reconnected in an intelligent way with curves reflecting the directions that the pen is moving at pen-up and pen-down. If pen motion information is available from the digitiser when not in contact with the writing surface, this could be used to reconnect the strokes.

v) The detection of diacritical marks (dots and crosses) is not error-free. Sometimes the classification is ambiguous (a short line, for instance). Sometimes a diacritical mark is not detected at all and the stroke is reconnected into the word, causing errors in the recognition.

Preprocessing errors iii), iv) and v) could be reduced with further development of the preprocessing routines. As these areas were not within the scope of this research, recognition errors due to these problems can be ignored in the analysis of this work.

vi) Another problem associated with preprocessing is the problem of handwriting of inconsistent height. The detection of the halfline is made more difficult if tall letters are not appreciably larger than short letters, and in some writing styles the height of letters varies considerably. Any handwriting recognition system will be hindered to some extent by this factor.

8.5.3. Segmentation Errors

vii) The inconsistencies of segmentation described in section 4.5.1 can cause misrecognition. It should be noted that even though there are inconsistencies between the practical segmentation and the theoretical segmentation, the errors *are* in fact consistent within themselves. For example, in the case illustrated in figure 4.13, if a cusp *is* detected at the baseline, giving rise to five segments instead of three, that segmentation *is* consistent with similarly segmented strokes. Most of the problems can thus be rectified by the inclusion of extra templates for the other possible segmentations.

viii) The segmentation occasionally fails due to errors in the implementation.

8.5.4. Template Matching Errors

ix) The templates are constructed simply by averaging a number of samples from each class. Match probabilities are calculated using the estimated distribution of these samples. A number of errors can be introduced in this way, primarily as a result of approximating the distribution of the features as multivariate normal. Firstly, the distribution of a template feature may be

adequately approximated by a normal distribution, but in practice limits may be placed on its possible range of values, for example the vertical position of an ascender should never fall below the halfline of a word. Secondly, one sample feature may match the template very well. If the estimated distribution for that feature has a low standard deviation, the comparison score will be greatly influenced by that feature, possibly producing a high comparison score even if the other features match poorly. This may occur especially if only a few samples have been used to generate the template.

Generally, these errors will not result in a correct template match being rejected but in an incorrect template match being accepted. This will reduce the weight associated with the correct template match, however, due to the normalisation process.

x) Theoretically, the template matching algorithm will not reject a correct match since all of the ambiguities should be retained (ie every letter should occur in every position in the letter graph, even if its probability is very low). In practice, for efficiency, cut-offs are required to reduce the computation necessary by reducing the number of template matches retained in the system. If a correct template match has a low probability, it is therefore possible that it may not appear in the candidate allograph list.

xi) If a particular letter formation is unusual, it is possible that a template may not exist in the database for it. The system will fail unless that template is introduced and trained.

8.5.5. Dictionary Lookup Errors

xii) No errors occur due to the dictionary lookup itself (excluding words not in the dictionary). If the correct candidate allographs are included in the letter graph the correct word will be output, together with a reliable probability based on the probabilities of its constituent components. It has been noted, however, that letter-joins are less reliably identified than letters. Errors can occur, therefore, if the system is running with high thresholds to increase its speed of operation. In this situation, wild-card letter-joins are not included in the letter graph, so words may be missed if the less consistent letter-joins are not correctly identified.

xiii) A number of errors have been caused due to mistakes in the template-join table, that specifies which templates may connect to which others. These are

especially noticable with very separated handwriting styles, where the stroke reconnection algorithm has introduced spurious data.

8.6. Error Rates

The B2 samples recognised by the untrained system (test I) were examined to identify the cause of any errors. Out of these samples (266 words), 164 words (62%) were correctly identified with the highest probability. The other 102 (38%) are detailed in table 8.2.

Error type	No of words	%age	Error group
Word near top	27	9.9	A
Omitted letter template	23	8.4	C
Omitted join template	20	7.6	C
Poor template match	13	4.9	C
Stylish flourish	6	2.3	A
No wild-card search	3	1.1	A
Failed segmentation	3	1.1	B
Preprocessing failed	3	1.1	A
Join-table error	2	0.8	B
Writing error	2	0.8	A

Table 8.7 - Error rates

The different error types are discussed below. Error group A consists of those errors which may be ignored since either the correct word is weighted very close to the top word, or the error is not caused directly by the ORCHiD system. Error group B consists of those errors caused by mistakes in the programming or implementation, and error group C consists of those errors due to non-training of the database.

Word near top

The probabilities of words in this group were sufficiently close to the top-ranked word for no clear decision to be made between them.

Omitted letter template

The correct letter template was not present in the candidate allograph list. This could be due to causes x) or xi) identified above.

Omitted join template

The correct letter-join template was not present in the candidate allograph list. This could be due to causes x) or xi) identified above. This number does not include those omitted letter-joins inferred by the use of wild-cards. Wild-card letter-joins are only inserted with a maximum length of four segments, so longer omitted joins will be counted here.

Poor template match

The correct candidate allograph occurred in the allograph list, but with a low probability. This may be due to cause ix).

Stylish flourish

An unusual style was not included in the template database (cause i).

No wild-card search

There were a large number of segments in the word, such that including all wild-card letter-joins would have caused lengthy processing (cause xii) above). If this were done, however, the word would be correctly identified.

Failed segmentation

The segmentation failed due to errors in the implementation (cause viii).

Preprocessing failed

An error occurred in the preprocessing that was not identified during the verification stage. This could have been due to cause iii), iv) or v).

Join-table error

The correct templates were identified but errors in the template-join table caused the word to be rejected (cause xiii).

Writing error

The word was incorrectly formed (cause ii).

8.6.1. Discussion

It can be seen from these results that, for this sample, 62% of words were identified with the highest probability, 15.2% of words were either close to the top-rank or could be ignored due to some allowable error (group A), 1.9% of errors were due to implementation mistakes (group B), and 20.9% of errors were due to lack of training or some other unidentified error (group C). It can be seen from the test results of the trained system that this last group can be reduced by at least a half after training to bring overall recognition rates up to >90%.

8.7. Speed

Various factors affect the speed of recognition of the ORCHiD system.

The preprocessing speed depends on the amount of sample data collected. The NPL routines preprocess a large block of data rather than a single word at a time. Preprocessing time is proportional to the amount of data being processed. Consequently sample data less than the size of a block is processed more quickly.

The template matching speed is affected by the number of segments in a word (s) and the number of templates in the database (t). Since the speed is proportional to the number of template comparisons, and the number of comparisons is approximately equal to $s \times t$, this is a linear function. The matching routine is therefore quicker when trained, since less templates will be in the template database.

The dictionary lookup speed is less easily determined since the major factor is the number of successful matches between the letter graph and the dictionary. On average, the number of entries in the candidate allograph list and the number of words in the dictionary will affect the speed, especially since larger data structures may occupy more than the physical memory available to the processor and cause page faults. It is therefore preferable to keep the size of these structures to a minimum. The letter graph is kept smaller by the use of a probability threshold

below which a candidate allograph is rejected, and by only inserting wild-card letter-joins if the probabilities associated with the output do not exceed another threshold. It should be noted, however, that it is sometimes possible for a small letter graph and dictionary to produce more matches than a large one, and hence take a longer lookup time.

The speed of recognition of the trained system was tested with a large set of data containing 107 words of varying length (2-13 letters). The process of template matching, dictionary lookup and output sorting took 629 seconds on a time-shared Sun 4/330, including file access for sample data and writing output, but excluding template and dictionary database loading. This gives an average of 5.88 seconds per word. The thresholds were set such that 90% recognition was achieved.

The system could not be described as running in real-time on the current hardware, but processor speeds are constantly improving and it seems likely that there will soon be sufficient power available to process even very complex recognition systems within real time. The ORCHiD system was developed as a research platform and so was not designed with speed of execution as its main criterion. For example, the system stores intermediate data in files that can be examined after processing. This requires many disc accesses during the recognition process, slowing down the overall performance time.

8.8. Summary

The untrained recognition system achieved an average recognition rate of 75% using a large test sample (960 words). After a short initialisation period for a particular writer, the system can be automatically trained during use to improve the recognition accuracy. After training, recognition rates of 96% were achieved. Average processing time after preprocessing for the trained system is 5.88 seconds per word.

Chapter 9

Conclusions and Further Work

9.1. General Conclusions

The ORCHiD system developed during this research is a practical implementation of a trainable recognition system for lower case cursive script. Handwriting data is collected online and preprocessed, with the baseline and halfline detected, by the NPL routines. After segmentation, segment features are measured and compared with a database of templates. Possible letters and letter-joins are combined to produce possible words which are verified against a lexicon of valid words. Although the system presently recognises only lower case script, the template matching routine could be easily extended to include recognition of upper case characters.

A summary of each section of the research is given below, together with an indication of which areas are innovative.

9.1.1. Segmentation

The basic elements of the segmentation method were first described by Berthod[2] and Higgins[38]. The suggestions encompassed by these works have been fully developed and extended in this thesis to provide a practical segmentation method of direct relevance to cursive script recognition (CSR). A number of problems have been resolved by examination of the underlying structure of the writing. The way in which different features of script evolve into other features (as writing accuracy deteriorates) has led to the development of the x-axis stretching technique. This resolves the problem of multiple intersections within the script, and provides a measure for comparing graphically different segments based on the same underlying letter formation.

The segmentation is very consistent across a number of different handwriting styles, enabling a reduced number of templates to be stored.

9.1.2. Features

A number of feature measurements are taken for each segment. There is a fixed number of features per segment allowing a valid statistical template matching approach since the dimensionality of the array of vectors for each segment is the same. The distribution of the features can be approximated by a multivariate normal distribution, for ease of calculation.

9.1.3. Template Matching

A new algorithm for template matching has been developed, based on a probabilistic method using conditional probabilities. The use of segmented templates enables valid statistical combinations across a word and allows comparison between different possible words.

9.1.4. Dictionary Lookup

The use of a tree structure, first proposed by Knuth[54], was rejected for a number of years due to the large requirements of computer resources. This research has shown that in an efficient implementation this structure is ideal for reducing the number of candidate words, and verifying them simultaneously.

9.1.5. Training

The system has been designed in such a way that the adjustment of templates to train the system to an individual writer is a very simple process. A new template can be added by presenting a number of samples. Training can be automatic, and concurrent with system use. Initial training can be kept to a minimum before the system can be used.

9.1.6. Important Aspects of this Research

The most important developments described in this thesis include:-

- the use of the *copy-book model* to reduce the number of templates required (section 4.1);
- the x-axis stretching technique to compare handwriting samples that have evolved from the same copy-book base-style (section 4.4);
- the use of data-driven templates providing a simple method for the addition and training of unusual allographs (section 5.5);

- the calculation of a statistically valid probability of matching for a candidate word (section 5.6); and
- the very efficient dictionary lookup method (sections 6.4 and 6.5).

9.2. Recognition Rates

The system achieves fair recognition rates for untrained recognition (75%) and good recognition rates after some training (91% on average). It is difficult to compare results between different research groups due to the differences between systems and experimental results quoted (section 2.10). Table 2.1 shows results quoted for some of the research teams. The work of Higgins (85-93%)[38], Wright (82-94%)[104] and Ouladj (94%)[71] appear as the best quoted results to date, due to the less restricted test data used. It should be noted that Wright's figures are for untrained recognition, Higgins used test samples from only two writers and Ouladj's work uses a very small dictionary. The recognition rates reported in this thesis seem to be at least comparable with these results.

9.2.1. A Discussion of Required Recognition Rates

At present no system has been able to achieve anywhere near 100% recognition without the application of syntactic and semantic contextual knowledge. Even with the use of higher level context, 100% recognition has not been achieved, since this is only of value if the correct word is contained in the candidate word list. An omitted word cannot usually be inferred automatically. With this in mind it is necessary to ask if 100% recognition will ever be attained, or indeed is it necessary before CSR will be used in earnest?

Let us consider some of the possible uses of CSR. Handwriting will not be used as a mass text entry medium, since a skilled typist can type much faster than a neat handwriter can write. It could be argued that in the long term speech recognition may well be available for the non-keyboard user to enter large amounts of text into a computer. So when would handwriting be used? It would be used for many of those applications where we currently use handwriting on paper, but in conjunction with a portable electronic paper notebook. For example one could take notes in situations where a keyboard or speech input would be obtrusive or impractical. Diagrams and sketches that have been prepared with pen-based drawing packages could be annotated without having to set down the pen. (One of the problems with current mouse-driven systems is that the mouse

has to be released to type at the keyboard.) It should be noted that even if speech recognition were available it is not necessarily the best text input method, since we do not speak in the same way as we write.

Within the applications listed above it is unlikely that the user will wish to use correct sentence construction, so the gains in recognition accuracy achievable from the use of sentence context may be lost. We can, however, still see a use for a CSR system that is not 100% accurate in an electronic paper environment. Such an environment will provide an editing mode which allows for the rapid correction of mistakes by simple gestures, eg crossing out a word. If the CSR system were combined with such an editing environment, then it would be possible to simply correct for any misrecognition during proof-reading by touching the incorrect word and having the next likely word appear instantly, say. In this way, the small amount of misrecognition can be easily corrected.

9.3. Further Work

There are a number of related areas which would benefit from further investigation. Some of the areas detailed below are of direct application to the work described in this thesis while others are concerned with cursive script recognition in general, especially with respect to producing a feasible commercial system.

Improvements of direct relevance to this work include developing an enhanced set of general templates, for the untrained recogniser, and devising a more efficient training scheme. CSR systems in general may be improved with the use of an accurate Electronic Paper (EP) device for data collection, the use of higher level context, when appropriate, and development of a combined recognition system, with a number of different recognition methods providing a combined result.

9.3.1. Improved Letter Template Database

The accuracy of the current untrained system is limited by the quality of the generalised set of templates. The system is prone to error if a spurious template has accidentally been introduced into the system. The current template database was constructed simply by incorporating a large number of samples as they were encountered. This was mainly carried out automatically and so it is likely that a few inaccurate templates have been introduced (see section 7.6).

It would be a useful exercise to construct a theoretical set of ideal templates to use as the general template database. This might be carried out by considering what would be reasonable values for each of the features of a template segment. Some of the features would be easy to specify theoretically (vertical position, for example) whereas others may have to be calculated from a large number of carefully identified samples (area, for example). In this way it could be ensured that no spurious templates are introduced.

By using such a carefully selected general template database, the recognition rates should be improved for careful handwriting that is relatively standard. This improvement will occur since the less common templates will no longer influence the probability of the correct templates. Writers with more unusual styles will be excluded, however, without some initial training.

9.3.2. Training

The automated training routine described in Chapter 7 works reasonably well, but suffers from a fundamental flaw. If a writing style contains a peculiar allograph that does not exist in the general template database, there is no method to automatically insert it into the personal database. The allograph must be identified and included by hand. This, at present, requires more detailed knowledge of the system than is desirable for an average user.

A less restrictive method of automated training is that described by Teulings *et al*[94] detailed in section 2.9. This method selects templates by requesting the user to write similar words and analysing the differences between the extracted features. This has the advantage that no previous knowledge of possible template formations is necessary. All templates can be deduced by judicious selection of a training set of words. The problem with this approach is that the training set will need to be quite large to allow every template to be extracted, and hence the training session will be lengthy.

A more reasonable method might be to combine these two approaches. The method described in this thesis could be used initially. Any templates that have not then been successfully identified could invoke a Teulings training session with a smaller set of training words that concentrated only on the missing templates.

Another area associated with the template database that should be investigated is the continuous assessment of the database. As a personal database is

continually trained, it will increase in size as additional variations in letter formations are encountered. This will cause the system to slow down as an increasing number of templates must be matched against each sample.

To reduce this problem it would be desirable to have a process which assesses the database and removes any unwanted templates. These might include:-

- templates that have not been used for a long time, for instance if the user's style has evolved and changed;
- templates that have only occurred rarely, as might be produced in error by the automatic training routine;
- templates that are not significantly different from another template. During a training period it is possible that an allograph cannot be classified with any of the current templates and so it is stored as a new template. In the course of further training, all of the templates may be adjusted and this new template might be trained so that it is now sufficiently similar to another template that they may be amalgamated.

9.3.3. Data Collection and Preprocessing

A number of benefits will be gained by using an EP device (section 1.2.2) for data collection.

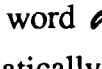
- A more accurate sample of script will be collected. With a traditional opaque digitising tablet and monitor the writer cannot easily tell if the script is being successfully digitised - for example the writing may be too fast, or be written without sufficient pressure to trigger the pen switch, or there may be hardware glitches. It is therefore necessary for the writer to watch the monitor screen as the writing is taking place. This is a very unnatural way to write and there is often a time-lag between the digitiser recording the data and the output on the graphical display, causing coordination problems for the writer. With the direct feedback of an EP device, the writer will immediately see the data that the machine is recording, and will know if the digitising process is successful. If necessary the pen trace (electronic ink) could be thickened to force the user to write sufficiently large for successful digitising. (Compare this with using a thick nibbed felt pen which naturally makes people write bigger.)

- At least in early systems, the handwriting could be constrained to aid the recognition hardware and software in such a way that is not intrusive to the user. For example, guide lines could be drawn on the writing surface for the user to write on and as a guide to the size of writing. The spacing of these lines might be user adjustable, but would assist the preprocessing software in locating the various vertical regions of the script. Guide lines should not be intrusive since real paper is commonly ruled to assist writing.
- An EP device provides instant feedback to the user on the machine's interpretation of gestures drawn onto the device and the consequent actions that it has carried out. Just as the characters appearing on a conventional computer display indicate to a typist any miskeying, so any misrecognised script will appear very quickly on the EP display. Since users of EP will (probably) want the machine to work for them, it seems only natural that they will try to accommodate the machine by adapting their writing style, at least to some extent. (This process may even take place subconsciously.) This user adaptation, together with the system learning a writer's style, might considerably improve recognition rates.

Further experiments to evaluate all of these different aspects should be carried out when suitable hardware (and software) becomes available.

9.3.4. Context

The dictionary verification method described in this thesis is an efficient implementation of a word level contextual disambiguation process. The output from this process, however, is still ambiguous, resulting in a number of possible candidate words for each sample word. Higher levels of orthographic context, such as the syntax and semantics of phrase and sentence structure, will further improve the recognition rates within certain applications, by removing unlikely word combinations. Ambiguous word segmentation may also be resolved at this level. This is an area of Artificial Intelligence (outside the scope of this thesis) which should be studied further[51,79].

The use of context has its problems, however, since it enforces a structure on the writer. For example, insisting on valid dictionary words prevents the user from writing the word  (perhaps a coded filename) since this might be recognised and automatically interpreted as 'dog', say. The use of sentence level context prevents the user from writing  , since the

system might automatically substitute the word 'dog' for 'clog' as this is a more meaningful interpretation. It would therefore be impossible to use such a system to write this paragraph!

There are also dangers in relying too heavily on context to correct for poor recognition. If a letter has been completely missed within a word during the recognition phase then simple dictionary context cannot find the correct word. Similarly, if a word has not been recognised within a sentence, a sentence level context approach cannot *verify* the sentence. It may be possible to insert a wild card letter, for example, into a word and produce a small list of possible values for that letter by comparison with a dictionary, but it is much less practical to generate a list of words to complete a sentence. To resolve this type of problem we must use the methods described in sections 2.7.3-6 where substitution sets of likely errors are examined to decide what word or sentence was written.

Context can be a great assistance in allowing for the shortfalls of the recognition phase, but should not be relied on in preference to improving the recognition phase.

9.3.5. A Pooled Method Approach

The data-driven recognition system described in this thesis has shown that such a method can be used for unconstrained script recognition, but it is always prone to produce the occasional peculiar output due to mathematical anomalies (see section 5.7.1). At the other extreme, a completely rule-based system could be used which defines general rules for letter or word formation independent of writing style. However, this approach suffers from peculiarities that occur within the wide variety of handwriting styles in common usage. No matter how carefully a rule has been devised, there will always be an exception or an ambiguity to be resolved. It seems evident that no recognition system based on just one of these techniques will be completely successful. In the short term, a successful CSR system will need to be:-

a) easily trainable, so that the user does not need to learn a new skill to operate the device; and

b) subject to a number of built-in, all-encompassing rules to remove any peculiar anomalies from the output.

The best way of achieving this may be to have a number of different recognition modules that may independently analyse the data and then pool the output

from each of the modules to produce an aggregate set of output. For efficiency it may be possible to use a simple but fast method to quickly analyse the data and, if this does not produce conclusive results, to bring in more complex recognition modules.

One method which could take advantage of these ideas is a *blackboard system*[25,44]. This type of system contains a shared data area, the *blackboard*, where a number of different processes (*knowledge sources*) can locate data on which they can operate, and where they place their output.

An efficient CSR system could be developed using a blackboard approach to continually reduce a lexicon of candidate words by examining smaller and smaller features of a word. For example, ascenders and descenders might be used to reduce the dictionary of valid words. Another process might then examine this word list to decide what other features of the word would most profitably be considered next. Some of these processes might be trainable to a particular user, others might contain fixed general rules that must be satisfied. In this way, the problems of using any one fixed method (discussed above) may be alleviated by incorporating both data-driven and rule-based systems.

9.3.6. Feedback

As each stage of the recognition process is completed, information is produced that may have been of use to the previous stages of recognition, had that information been available. For example, if one of the stages in the recognition process cannot produce reasonable output, then it is possible that its input data is suspect. If each of the preceding processes are adjusted in turn, perhaps by moving a threshold, until reasonable output can be achieved, the overall recognition rates of the system may be improved.

Future recognition systems (especially blackboard systems) may be able to make use of this continual flow of information backwards and forwards between each individual process. Each process will need to retain details of each action it has applied to the data, so that it can undo and reapply the action using the additional information supplied to it. A few possible examples follow. If a peculiar set of features is extracted from the data, this may indicate that the preprocessing has erred and should be corrected before continuing with the recognition process. If a word is identified, then the letter segmentation is clear and can be passed back to the segmentation phase for future reference. The dictionary lookup phase

could be dynamically linked to the template matching phase and pass back information as to which template to try to match next.

Within the ORCHiD system, some feedback is currently used by the template matching routine, since the templates are adjusted during training to incorporate previously observed samples. The dictionary lookup routine uses feedback to improve its performance by only including wild-card letter-joins if the output word probabilities are low. A closer "feedback-loop" between the template matching stage and the dictionary lookup stage would be useful. As a candidate allograph is rejected by the dictionary verification, the relevant probabilities for the other allographs occupying the same segments could be recalculated. This is similar to the method used by Hayes[37] (see section 5.2) but, unlike Hayes' method, this will still produce valid probabilities conditional on the other possible allographs in the letter graph. This was not incorporated within the system due to implementation difficulties and time constraints.

9.4. Closing Remarks

A working CSR system has been produced. The techniques developed as part of the ORCHiD system could provide an important constituent element of any future CSR development. In particular, two fundamental principles for CSR have been identified and successfully incorporated during development. Firstly, the principle of retaining ambiguity wherever possible is extremely important within a CSR system. Secondly, an adaptive system that can be trained to an individual will remain a necessity for any practical system in the near future and may provide a useful security measure for personal identification in the long term.

This system has shown that these principles can be adhered to within an effective CSR system. The untrained ORCHiD system achieves acceptable recognition rates with fairly neat handwriting (75%) and it is particularly successful after training, when recognition rates of 96% were achieved.

Bibliography

1. Baecker, R. M. and Buxton, W. A. S., "A Historical and Intellectual Perspective," in *Readings in Human-Computer Interaction*, Morgan Kaufmann Publishers, 1987.
2. Berthod, M., "On-Line Analysis of Cursive Writing," in *Computer Analysis and Perception - Visual Signals*, ed. R. De Mori, vol. 1, pp. 55-81, CRC Press.
3. Berthod, M. and Ahyan, S., "On Line Cursive Script Recognition: A Structural Approach with Learning," CH1499-3/80/0000-0723S00.75c.1980 IEEE, pp. 723-725, 1980.
4. Boes, U., Doster, W., Fogaroli, G., Lobl, H., and Maslin, G., "The Role of Paper in the Automated Office," Esprit Project 295 - The Paper Interface.
5. Bozinovic, R. and Srihari, S. N., "A String Correction Algorithm for Cursive Script Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, no. 6, pp. 655-663, November 1982.
6. Bozinovic, R. and Srihari, S. N., "Off-Line Cursive Script Word Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 68-83, January 1989.
7. Brocklehurst, E. R., Ford, D. M., and Symm, H. J., "Feature Extraction for Cursive Script Recognition," NPL Report DITC 130/88, 1988.
8. Brocklehurst, E. R., "The NPL Electronic Paper Project," NPL Report DITC 133/88, 1988.
9. Brocklehurst, E. R. and Stevens, M. J., "Software Modules for Electronic Paper," NPL Report DITC 127/88, 1988.
10. Brocklehurst, E. R. and Kenward, P. D., "Preprocessing for Cursive Script Recognition," NPL Report DITC 132/88, 1988.
11. Brocklehurst, E. R., "The NPL Electronic Paper Project," *International Journal of Man-Machine Studies*, vol. 34, pp. 69-95, 1991.
12. Brown, M. K. and Ganapathy, S., "Cursive Script Recognition," Proc Int Conf on Cybernetics & Society, pp. 47-51, Cambridge, MA, USA, October 1980.
13. Brown, M. K. and Ganapathy, S., "Preprocessing Techniques for Cursive Script Word Recognition," *Pattern Recognition*, vol. 16, no. 5, pp. 447-458, 1983.

14. Burr, D. J., "Designing a Handwriting Reader," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 5, pp. 554-559, September 1983.
15. Cox, M. G. and Harris, P. M., "An Algorithm for Loop Detection in Cursive Script Recognition," NPL Report DITC 120/88, 1988.
16. Davis, M. R. and Ellis, T. O., "The Rand Tablet: a Man-Machine Graphical Communication Device," *Proc FJCC*, pp. 325-331, 1964.
17. Dooijes, E. H., "A Description of Handwriting Dynamics," Simulation of Systems '79, pp. 727-731, 1980.
18. Doster, W. and Oed, R., "Word Processing with On-Line Script Recognition," *IEEE Micro*, pp. 36-43, 1984.
19. Duda, R. O. and Hart, P. E., "Experiments in the Recognition of Hand-Printed Text: Part II - Context Analysis," *AFIPS Conference Proceedings*, vol. 33, pp. 1139-1149, 1968.
20. Duda, R. O. and Hart, P. E., in *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.
21. Earnest, L. D., "Machine Recognition of Cursive Writing," Information processing 1962 (Proc. IFIP Congr.), pp. 462-466, 1962.
22. Eden, M. and Halle, M., *The Characterization of Cursive Writing*.
23. Ehrich, R. W. and Koehler, K. J., "Experiments in the Contextual Recognition of Cursive Script," *IEEE Transactions on Computers*, vol. 24, no. 2, pp. 182-194, February 1975.
24. Elliman, D. G. and Lancaster, I. T., "A Review of Segmentation and Contextual Analysis Techniques for Text Recognition," *Pattern Recognition*, vol. 23, no. 3/4, pp. 337-346, 1990.
25. Engelmore, R. and Morgan, T., in *Blackboard Systems*, Addison Wesley, 1988.
26. Farag, R. F. H., "Word-Level Recognition of Cursive Script," *IEEE Transactions on Computers*, vol. 28, no. 2, pp. 172-175, February 1979.
27. Ford, D. M., Higgins, C. A., and Brocklehurst, E. R., "The Electronic Paper Project," *Proceedings of the Third International Symposium on Handwriting and Computer Applications*, Montreal, 1987.
28. Ford, D. M., "Cursive Script Recognition using Dictionary Information," Internal Report, September 1987.
29. Ford, D. M. and Higgins, C. A., "Electronic Paper Project Final Report - Cursive Script Recognition," Nottingham University Computer Science Technical Report TR0018, 1988.
30. Ford, D. M. and Symm, H. J., "Segmentation and Reduction Analysis for Cursive Script Recognition," NPL Report DITC 131/88, 1988.
31. Forney, G. D. Jr., "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268-278, March 1973.

32. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computing*, vol. 10, pp. 260-268, June 1961.

33. Frishkopf, L. S. and Harmon, L. D., "Machine Reading of Cursive Script," 4th London Symposium on Information Theory, pp. 300-316, 1961.

34. Gon, J. J. Denier van der and Thuring, J. P., "The Guiding of Human Writing Movements," *Kybernetik*, vol. 2, pp. 145-148, 1965.

35. Gould, J. D. and Alfaro, L., "Revising Documents with Text Editors, Handwriting-Recognition Systems, and Speech-Recognition Systems," *Human Factors*, vol. 26, no. 4, pp. 391-406, 1984.

36. Harmon, L. D., "Automatic Recognition of Print and Script," *Proceedings of the IEEE*, vol. 60, no. 10, pp. 1165-1176, October 1972.

37. Hayes, K. C., "Reading Handwritten Words Using Hierarchical Relaxation," *Computer Graphics and Image Processing*, vol. 14, pp. 344-364, 1980.

38. Higgins, C. A., "Automatic Recognition of Handwritten Script," PhD Thesis (CNAA), Brighton Polytechnic, 1985.

39. Higgins, C. A. and Whitrow, R., "On-Line Cursive Script Recognition," First IFIP Conference on Human-Computer Interaction - INTERACT 84, pp. 139-143, 1985.

40. Higgins, C. A. and Duckworth, R. J., "The PAD (Pen and Display) - A Demonstrator for the Electronic Paper Project," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, pp. 111-132, World Academic Press, 1990.

41. Higgins, C. A. and Ford, D. M., "Stylus Driven Interfaces - An Evaluation of Electronic Paper," CCTA Report, February 1991.

42. Hull, J. J. and Srihari, S. N., "Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, no. 5, pp. 520-530, September 1982.

43. Apple Computer Inc, in *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley, 1987.

44. Jagannathan, V., Dodhiawala, R., and Baum, L. S., in *Blackboard Architectures and Applications*, Academic Press, 1989.

45. James, P., *The Structure of Cursive Handwriting*, October 1975.

46. Johansson, S., *Manual of Information to Accompany the Lancaster-Oslo/Bergen Corpus of British English, For Use with Digital Computers*, University of Oslo, 1978.

47. Kadirkamanathan, M. and Rayner, P. J. W., "A Scale-Space Filtering Approach to Stroke Segmentation of Cursive Script," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, pp. 133-166, World Academic Press, 1990.

48. Kashyap, R. L. and Oommen, B. J., "An Effective Algorithm for String Correction Using Generalized Edit Distances - I. Description of the

Algorithm and Its Optimality," *Information Sciences*, vol. 23, no. 2, pp. 123-142, 1981.

49. Kashyap, R. L. and Oommen, B. J., "An Effective Algorithm for String Correction Using Generalized Edit Distances - II. Computational Complexity of the Algorithm and Some Applications," *Information Sciences*, vol. 23, no. 3, pp. 201-217, 1981.
50. Keenan, F. G. and Evett, L. J., "Lexical Structure for Natural Language Processing," Proc. First Int Workshop on Language Acquisition, IJCAI-89, 1989.
51. Keenan, F. G., "The use of linguistic knowledge in a handwriting recogntion system," Internal Report, Nottingham Polytechnic, 1990.
52. Kim, J. and Tappert, C. C., "Handwriting Recognition Accuracy versus Tablet Resolution and Sampling Rate," 7th Int Conf on Pattern Recognition Vol. 2, pp. 917-918, 1984.
53. Kim, Y. T., "Syntax Directed On-Line Recognition of Cursive Writing," TR4-8 AD 770-241, University of Utah, Salt Lake City, 1973.
54. Knuth, D. E., in *Sorting & Searching, The Art of Computer Programming*, vol. 3, pp. 481-487, Addison-Wesley, 1973.
55. Krishnaiah, P. R. and Kanal, L. N., "Classification, Pattern Recognition and Reduction of Dimensionality," in *Handbook of Statistics*, vol. 2, North-Holland Publishing Company, 1982.
56. Krishnaiah, P. R. and Sen, P. K., "Nonparametric Methods," in *Handbook of Statistics*, vol. 4, North-Holland Publishing Company, 1984.
57. Kucera, H. and Francis, W. N., *Computational Analysis of Present-day American English,* Brown University Press, Providence RI.
58. Leedham, C. G. and Chan, W. L., *Recognition of Unconstrained, Separated Handwritten Characters,* Department of Electronic Systems Engineering, University of Essex, 1988.
59. Lindgren, N., "Machine Recognition of Human Language, Part III-Cursive Script Recognition," *IEEE Spectrum*, pp. 104-116, May 1965.
60. Lowrance, R. and Wagner, R. A., "An Extension of the String-to-String Correction Problem," *Journal of the ACM*, vol. 22, no. 2, pp. 177-183, 1975.
61. Maier, M., "Separating Characters in Scripted Documents," Eighth International Conference on Pattern Recognition, pp. 1056-1058, Paris.
62. Meeks, M. L. and Kuklinski, T. T., "Measurement of Dynamic Digitizer Performance," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, World Scientific Press, 1990.
63. Mermelstein, P. and Eden, M., "A System for Automatic Recognition of Handwritten Words," Proceedings - Fall Joint Computer Conference, 1964 Vol. 26, pp. 333-342, 1964.

64. Miletzki, U., Doster, W., Fogaroli, G., Lobl, H., and Moulds, P., "Paper Interfaces for Office Systems," ESPRIT '86: Results and Achievements, pp. 373-387, Elsevier Science Publishers B.V. (North-Holland), 1987.

65. Miller, G. M., "Real-Time Classification of Handwritten Script Words," Information Processing 71, pp. 218-223, North-Holland Publishing Company, 1972.

66. Morasso, P., "Understanding Cursive Script as a Trajectory Formation Paradigm," in *Graphonomics: Contemporary Research in Handwriting*, ed. H. S. R. Kao, G. P. van Galen, R. Hoosain, pp. 137-167, Elsevier Science Publ., 1986.

67. Neave, H. R., *Elementary Statistics Tables,* George Allen & Unwin, 1981.

68. Neisser, U. and Weene, P., "A Note on Human Recognition of Hand-Printed Characters," *Information and Control*, vol. 3, pp. 191-196, 1960.

69. Neuhoff, D. L., "The Viterbi Algorithm as an Aid in Text Recognition," *IEEE Transactions on Information Theory*, pp. 222-226, March 1975.

70. Okuda, T., Tanaka, E., and Kasai, T., "A Method for the Correction of Garbled Words Based on the Levenshtein Metric," *IEEE Transactions on Computers*, vol. 25, no. 2, pp. 172-177, February 1976.

71. Ouladj, H., Petit, E., Lemoine, J., Gaudaire, M., and Lorette, G., "A Prediction-Verification Strategy for Automatic Recognition of Cursive Handwriting," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, pp. 187-206, World Scientific Press, 1990.

72. Peleg, A., "Ambiguity Reduction in Handwriting with Ambiguous Segmentation and Uncertain Interpretation," *Computer Graphics and Image Processing*, vol. 10, pp. 235-245, 1979.

73. Peterson, J. L., "Computer Programs for Detecting and Correcting Spelling Errors," *Communications of the ACM*, vol. 23, no. 12, pp. 676-687, December 1980.

74. Pobgee, P. J., "A Prototype System for Interactive Input of Cursive Information," NPL Report DITC 125/88, 1988.

75. Pollock, J. J., "Spelling Error-Detection and Correction by Computer - Notes and Bibliography," *Journal of Documentation*, vol. 38, no. 4, pp. 282-291, 1983.

76. Rengger, R. E., "Exploring Experimentally Derived Usability Metrics by a Laboratory Evaluation of Electronic Paper," NPL Report DITC 119/88, 1988.

77. Riseman, E. M. and Ehrich, R. W., "Contextual Word Recognition Using Binary Digrams," *IEEE Transactions on Computers*, vol. 20, no. 4, pp. 397-403, April 1971.

78. Riseman, E. M. and Hanson, A. R., "A Contextual Postprocessing System for Error Correction Using Binary n-grams," *IEEE Transactions on Computers*, vol. 23, no. 5, pp. 480-493, 1974.

79. Rose, T. G., "The use of natural language semantics as an aid to handwriting recognition," Internal Report, Nottingham Polytechnic, 1991.

80. Sakoe, H., "Two-Level DP-Matching - A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 6, pp. 588-595, December 1979.

81. Sankoff, D., in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, ed. D. Sankoff & J. B. Kruskal, Addison-Wesley, 1983.

82. Shinghal, R. and Toussaint, G. T., "A Bottom-up and Top-down Approach to Using Context in Text Recognition," *International Journal of Man-Machine Studies*, vol. 11, pp. 201-212, 1979.

83. Shinghal, R. and Toussaint, G. T., "Experiments in Text Recognition with the Modified Viterbi Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 184-192, April, 1979.

84. Shinghal, R., "A Hybrid Algorithm for Contextual Text Recognition," *Pattern Recognition*, vol. 16, no. 2, pp. 261-267, 1983.

85. Smith, P. and Williams, J., *New Nelson Handwriting*, Thomas Nelson and Sons.

86. Srihari, S. N., Hull, J. J., and Choudari, R., "Integrating Diverse Knowledge Sources in Text Recognition," *ACM Transactions on Office Information Systems*, vol. 1, no. 1, pp. 68-87, January 1983.

87. Stevens, M. J., "A Text Editor Driven by Hand-Drawn Symbols," NPL Report DITC 124/88, 1988.

88. Suen, C. Y., Berthod, M., and Mori, S., "Automatic Recognition of Hand-printed Characters - The State of the Art," *Proc of the IEEE*, vol. 68, no. 4, pp. 469-478, April 1980.

89. Takeda, Y., Tanabe, N., Aono, Y., and Tokumitsu, Y., "A New Data Tablet Superimposed by a Plasma Panel," *Innovative Telecommunications*, vol. 4, pp. G5.4.1-5, 1981.

90. Tappert, C. C., "Cursive Script Recognition System by Elastic Matching," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 765-771, November 1982.

91. Tappert, C. C., "Adaptive On-Line Handwriting Recognition," 7th Int Conf on Pattern Recognition Vol. 2, pp. 1004-1007, 1984.

92. Tappert, C. C., Fox, A. S., Kim, J., Levy, S. E., and Zimmerman, L L., "Handwriting Recognition on Transparent Tablet Over Flat Display," SID International Symposium Digest of Technical Papers, pp. 308-312, 1986.

93. Tappert, C. C., Suen, C. Y., and Wakahara, T., "On-Line Handwriting Recognition - A Survey," Proc 9th International Conference on Pattern Recognition, pp. 1123-1132, 1988.

94. Teulings, H., Schomaker, L. R. B., Gerritsen, J., Drexler, H., and Albers, M., "An Online Handwriting-Recognition System based on Unreliable Models," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, pp. 167-185, World Scientific Press, 1990.

95. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, vol. IT-13, pp. 260-269, April 1967.

96. Vredenbregt, J. and Koster, W. G., "Analysis and Synthesis of Handwriting," *Philips Technical Revue*, vol. 32, pp. 73-78, 1971.

97. Wagner, R. A. and Fischer, M. J., "The String-to-String Correction Problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168-173, 1974.

98. Ward, J. R. and Phillips, M. J., "Digitiser Technology: Performance Characteristice and the Effects on the User Interface," *IEEE Computer Graphics and Applications*, pp. 31-44, April 1987.

99. Welbourn, L. K. and Whitrow, R. J., "A Gesture Based Text and Diagram Editor," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, World Academic Press, 1990.

100. Wells, C., Evett, L., Whitby, P., and Whitrow, R., "The Use of Orthographic Information for Script Recognition," in *Computer Processing of Handwriting*, ed. G. Leedham & R. Plamondon, World Academic Press, 1990.

101. Wells, C., Evett, L., Whitby, P., and Whitrow, R., "Fast Dictionary Look-up for Contextual Word Recognition," *Pattern Recognition*, vol. 23, no. 5, pp. 501-508, 1990.

102. Whitrow, R. and Higgins, C. A., "The Application of n-Grams for Script Recognition," Proceedings of the Third International Symposium on Handwriting and Computer Applications, pp. 92-94, Montreal, Canada, July 1985.

103. Wong, K. H. and Fallside, F., "Dynamic Programming in the Recognition of Connected Handwritten Script," Second Conference on Artificial Intelligence Applications, pp. 666-670, Miami Beach, USA, 1985.

104. Wright, P., "Algorithms for the Recognition of Handwriting in Real-Time," PhD Thesis (CNAA), Trent Polytechnic, 1989.

Glossary

Active Book	Example of *EP*.
Allograph	Topological structure of a character.
Baseline	See figure 1.8.
Candidate allograph	Possible character identified by template matching routine.
Candidate word	Possible output word identified by recognition system.
Centre-line	See figure 1.8.
Copy-book	Handwriting teaching book.
CSR	*Cursive script* recognition.
Cursive script	Natural, connected handwriting (section 1.3).
Data-driven system	Handwriting samples define *templates* (*cf rule-based system*).
DDP	Digitiser data point - the time-ordered list of coordinates provided by the *digitiser*.
Decision-directed	Method for automatic *training* (see section 7.3).
Descender-line	See figure 1.8.
Diacritical mark	Cross through 't', dot over 'i', for example.
Digitiser	Pen-driven computer input device.
DSP	Definite segmentation point.
Electronic Paper	Interactive display/digitiser operated with a stylus (section 1.2.2).
EP	*Electronic Paper*.
Feature	Descriptive value of a segment (section 3.8).
Freeman coding	Method for describing a stroke (section 2.5.2).
Full-height-line	See figure 1.8.
Halfline	See figure 1.8.
HCI	Human computer interface.
Interactive Tablet	*EP*
Ligature	*Letter-join*.
Letter-join	Portion of script connecting letters (figure 5.4).

N-gram *N* letter sequence occurring in a lexicon of words (section 6.3).

Noise Errors in data caused by hardware inadequacies (section 1.4.1).

Normalisation Standardisation and scaling of a sample (section 1.4.1).

NPL National Physical Laboratory, Teddington.

Off-line recognition Data collected statically, after writing (section 1.3).

On-line recognition Data collected dynamically, in real-time (section 1.3).

ORCHiD On-line Recognition of Connected Handwriting Demonstrator described in this thesis.

PAD Pen And Display - NPL EP demonstrator.

PC Personal computer.

Pen-down Point where pen touches writing surface.

Pen-up Point where pen leaves writing surface.

Preprocessing Adjustment of raw data to remove *noise* and *normalisation* (section 1.4.1).

PSP Possible segmentation point.

Postprocessing Use of additional information to improve recognition (section 1.4.3).

Rule-based system Predefined rules identify different characters (*cf data-driven system*).

Segmentation Division of script into smaller units for recognition (section 1.4.2).

Serif Stylistic or accidental pen-flick at beginning or end of a *stroke*.

SP Segmentation point.

Stroke Portion of script between *pen-down* and *pen-up*.

Template Internal representation of a character (section 5.1).

Training Adjusting recognition system for a particular user (Chapter 7).

Trie Data re*trie*val tree (section 2.7.7).

VA *Viterbi algorithm*.

Viterbi algorithm Method for optimally tracing a graph (section 2.7.3).

Wild-card A "match-all" character (section 6.5.2.3).

WIMP Window, Icon, Mouse and Pointer computer interface.

Appendix A

A Simple Recognition Demonstrator

A.1. Introduction

Early on in the development of the project it was decided that a simple demonstration system would provide a useful illustration of the concept of cursive script recognition, and prove the feasibility to a limited extent. A system was developed based on the work completed to date.

At this stage in the development of the recognition process, features were extracted from the script which could be described as graphical features, ie global features evident from examining the "image" of the script. With this type of information available, it was decided that a whole word recognition approach would be most profitable in producing a reasonable result for demonstration purposes.

A limited dictionary of accepted words was analysed and sorted according to the global features which were expected. The features extracted from the sample script were then compared against the dictionary to produce the output list of candidate words.

A simple real-time demonstrator was produced which was publicly demonstrated on the BBC TV programme *Tomorrow's World* (March 1988). It would recognise a fairly fixed style, but it was found that most people could adapt their style to achieve at least some recognition.

A.2. Hardware

This system was implemented on several combinations of hardware. Since the aim of this system was to produce real time recognition, the speed of processor and digitiser provided a number of constraints on the capabilities of the recognition software.

Perq with Opaque Tablet

The software was originally developed on a Perq 2 workstation, running PNX UNIX , and the Perq proprietary window system. This worked successfully but was slow due to the limitations of the Perq's central processor unit. The maximum usable dictionary size was 300 words, with the digitiser producing 60 coordinates per second over a parallel port.

Perq Connected to PAD

A prototype hardware, called the PAD[40] was developed for the Electronic Paper Project. Performance with this hardware was comparable with the Perq and opaque tablet. The host computer received only about 20 coordinates per second from the digitiser due to the slow speed of the RS232 connection. The low coordinate rate meant that the preprocessing had to be modified.

Sun 3 Workstation and Optical Mouse

The software was ported to a Sun 3 workstation, under the SunView window system. This was a successful implementation working much faster than the Perq. An 800 word dictionary was installed, and this produced a reasonable response. The optical mouse produces 40 coordinates per second.

Sun 3 connected to PAD

The PAD was integrated directly with the Sun via the VME bus. This was the optimal configuration for the system. The PAD digitiser produced about 50 coordinates per second.

A.3. Preprocessing

A single word was presented to the system. The raw data was preprocessed to straighten, deskew and smooth the handwritten sample. The baseline and halfline were detected. Details of the preprocessing are available in Brocklehurst and Kenward[10].

A.4. Feature Extraction

A number of primitive features were detected by the feature extraction routines at this time. These included pen-ups, pen-downs, cusps, intersections, straight lines, humps, i-dots and t-crosses. From these primitive features, a

number of basic features were identified, namely pen-ups, pen-downs, cusps in four directions (up, down, left and right), i-dots, t-crosses and four types of loop (or near loop). These loops, or closures, included not only loops delimited by intersections but also strokes which nearly meet (as in a letter 'o'). They could either be written clockwise or anti-clockwise and pointing upwards or downwards.

These basic features were reduced further to produce a more consistent set of features that could be used for recognition. Two processes were used to define this feature set. Firstly the dictionary was analysed to see which features would help to reduce the possible list of candidate words. Secondly the extracted features were examined for consistency when the same letters or word were written.

Analysis of the dictionary subject to different feature sets showed that the most useful features were ascenders, descenders, i-dots and t-crosses. (Experimental details are available in Ford and Higgins[29]). The information about position of cusps provided further reduction of the dictionary, as did the information about loops between the baseline and halfline (loops outside this region are either ascenders or descenders).

It was discovered that the direction of cusps occurring in the region between the baseline and halfline was not very consistent. For example, the initial cusp of a letter 'c' could be written pointing upwards, rightwards, or downwards. The occurrence of cusps is also ambiguous. Loops can also be ambiguous if the ends do not actually meet. Downward pointing clockwise loops that occurred between the baseline and halfline were found to be very inconsistent, since they were generally poorly written cusps.

Combining the above information led to the features of table A.1 being used within this system.

Feature	Description	Example letters
ascender	stroke rising above halfline	b d f h k l t
descender	stroke descending below baseline	f g j p q y z
o-loop	downward pointing, anti-clockwise loop	a d g o q
b-loop	upward pointing, clockwise loop	b k p
e-loop	upward pointing, anti-clockwise loop	e
i-dot		i j
t-cross		t
wild	none of the above	c m n r s u v w x

Table A.1 - Features extracted

Any basic feature that occurred considerably above the halfline was defined as an ascender (excluding i-dots and t-crosses), and any considerably below the baseline was defined as a descender. Any basic feature that did not fit into any of these categories was defined as wild.

An estimate of the number of letters in the word was calculated by counting the centre-line crossings and dividing by the average number of crossings per letter. Each of the features detailed above was identified together with its approximate letter position within the word and a likelihood weighting.

These features do not provide a unique description of the word, since they are ambiguous in their definition. For this reason more than one feature could be reported at any location within the word. For example, a cusp may be a tightly written e-loop and so both these possibilities would be reported with appropriate weight. A loose, non-closed o-loop might not be a loop at all, in which case it would be reported as wild.

Details of the implementation of the feature extraction routines are given in Brocklehurst *et al.*[7]

A.5. Dictionary Lookup

The list of possible features is constructed into a directed graph, similar to the letter graph described in section 5.7.3, but with weighted features attached to each node instead of letters.

The dictionary is coded into features using a substitution set for each letter. In this way, ideal letters are described in terms of features but each letter is then specified in a fixed style. Two data structures are constructed from this coded dictionary. Firstly a dictionary tree is produced using features instead of letters in a similar way to the letter tree described in section 6.4. Secondly a number of short lists of dictionary words are produced sorted on word length, i-dot count and t-cross count. These can be accessed quickly by a three dimensional array of pointers.

A comparison of the graph and tree is carried out producing a list of possible feature-coded words. These are then verified by cross-checking in the array-indexed lists. The estimate of word length was found to be in error by up to two letters in either direction, so this was allowed for at this stage. Allowances were also made for missing diacritical marks, for example, i-dots becoming elongated into t-crosses, and vice versa, and the use of a single stroke to cross a double 't'.

Three values are combined to give a certainty weighting for a particular word - the combined value of probabilities from each of the segments in the word, the difference in actual and estimated word length, and the combined distance of the letter positions of actual features from their estimated positions. This allows the output list of candidate words to be ranked.

Two dictionaries were installed, with 300 and 800 words. The most common words in English usage have been selected, the 800 word dictionary accounting for about 60% of all usage. In theory it is possible to install any size of dictionary, but this would be at the expense of speed of execution of the program. The accuracy of the results would also be reduced, since more words would be permissible.

A.6. Example

The word *the* was written into the system. Figure A.1 shows the screen displaying the raw data collected from the digitiser, the preprocessed word and the candidate words output from the system.

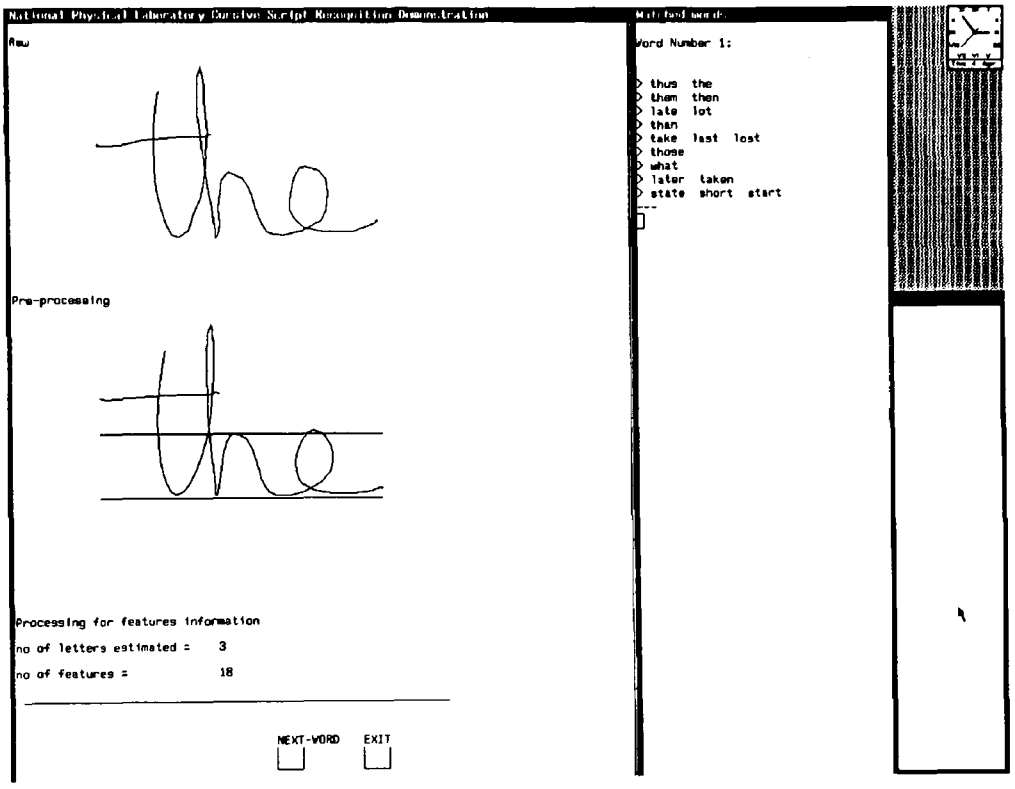


Figure A.1 - An example screen

The following list of features was extracted.

Feature	Height	Position	%	L	R	Basic Feature
ascend	23	1	100	1	2	pen-down
o-loop	13	2	29	2	26	
wild	13	2	71	2	26	
ascend	27	2	99	26	27	up cusp
wild	14	2	100	27	38	
wild	0	2	99	38	39	down cusp
wild	6	2	95	39	48	
b-loop	6	2	42	39	57	
wild	6	2	58	39	57	
o-loop	7	3	41	48	65	
wild	7	3	59	48	65	
wild	6	3	100	57	65	
e-loop	4	3	90	65	65	
o-loop	6	3	47	65	78	
wild	6	3	53	65	78	
wild	3	3	100	78	80	pen-up
t-cross	17	1	100	80	92	

`Feature` details the type of feature to be looked for in the dictionary.

`Height` is the scaled vertical position.

`Position` is the estimated letter position within the word, based on the estimate of word length and the horizontal position.

`%` is the weighting associated with the feature.

`L` and `R` are the coordinate pair numbers of the start and end of the feature. These are used to determine which features connect to which other features.

`Basic Feature` indicates what basic feature (if any) defined this feature.

These features were then combined and any matching words in the dictionary displayed in ranked order of likelihood. The output list can be seen in figure A.1 under the heading "Matched Words".

A.7. Evaluation and Conclusions

This system expected writing in a style loosely based on the author's own hand, but most people had few problems in conforming to this style. It could be tailored to slightly differing styles by installing different coded dictionaries. A large proportion of users could write and have their script successfully recognised, but there were problems with the preprocessing, especially detection of the base-line of short words and the half-line in non-linear and non-uniform writing. The writing samples were completely unconstrained in this system. The system was slow in operation and response time increased with a larger dictionary.

Despite its limitations, this system served its goal of demonstrating the feasibility of cursive script recognition. The underlying methods were, however, unsuitable for further extension to a more sophisticated system.

Appendix B

Example Processing

This appendix shows the processing of an example word through the ORCHiD system.

B.1. Preprocessed Data

Figure B.1 shows the collected sample after preprocessing and correction of any errors.

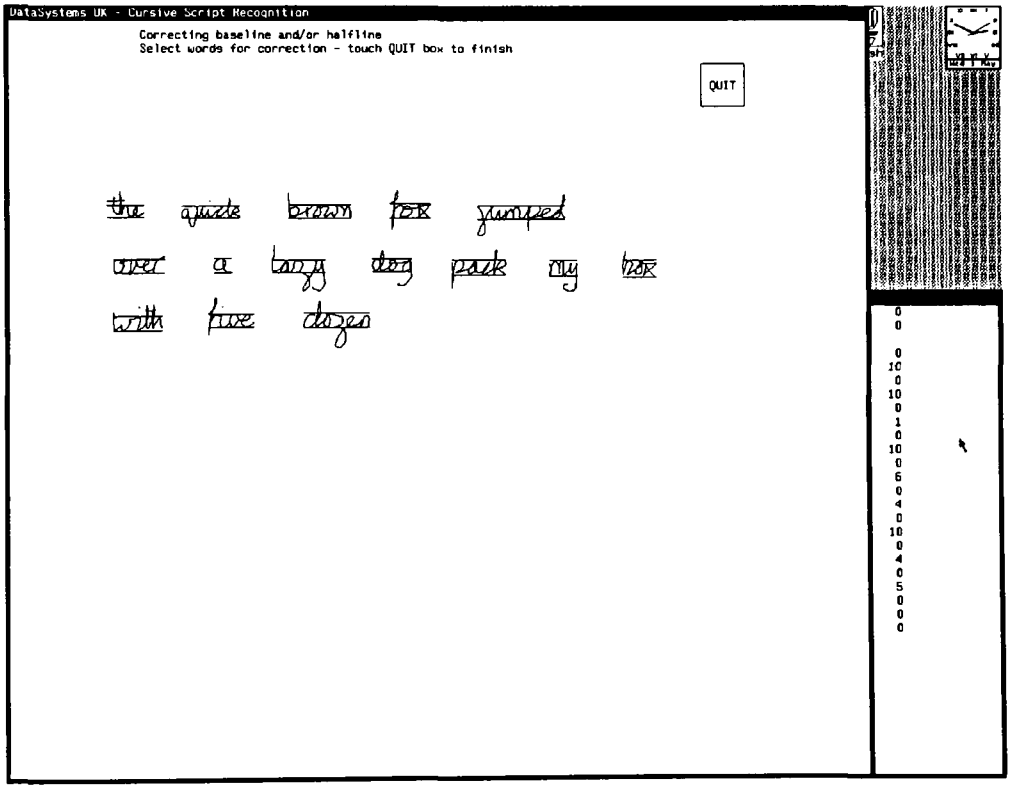


Figure B.1 - Preprocessed data

B.2. Features Extracted

The script is segmented and features are measured for each segment. These are shown in table B.1.

L	R	XM	DISP	YM	ANG	^IN	^OUT	AREA	MAXD	THICK	ASPCT	CHORD
1	2	-4	-	14	124	124	304	0	0	0	10	0
-3	-10	5	-	15	45	304	80	0	0	0	-	0
10	34	8	-	3	277	270	90	56	14	6	4	6
34	36	9	5	20	90	90	270	0	0	0	10	0
36	50	10	2	11	185	275	275	0	0	-17	250	17
50	51	10	1	2	270	270	90	0	0	0	10	0
51	69	12	2	9	90	79	256	-23	6	4	7	4
69	80	16	6	1	246	279	37	8	2	4	20	5
80	93	21	9	8	72	37	315	8	6	1	2	0
93	102	23	7	1	292	315	37	5	1	6	60	6
103	103	-26	5	3	37	37	217	0	0	0	250	0
103	104	-1	-	12	180	180	0	0	0	0	-	0
105	-118	6	-	12	0	0	0	0	0	0	-	0
119	119	-12	-	12	351	351	171	0	0	0	-	0

Table B.1 - Extracted features

A full description of these feature values is given in section 4.8. A brief description identifying the entries specific to this example is given below.

1-2	Cusp at top of 't' (pen-down)
3-10	Short segment which is deleted
10-34	Join between 't' and 'h'
34-69	The letter 'h'
69-80	Join between 'h' and 'e'
80-93	Loop of the 'e'
93-103	Tail of the 'e' (pen-up)
103-119	Cross through the 't'

B.3. Templates Matched

The segment features are compared against the template database and a list of candidate allographs produced, together with a probability for the match.

i.1	1	2	691	i.2	6	7	279
l.1	1	2	1096	u.2	6	8	233
t.1	1	2	1318	o.3	6	9	146
d.2	1	4	550	aa2	7	8	199
ah1	2	3	399	c.1	8	9	108
lh1	2	3	542	e.1	8	9	196
l.1	3	4	1426	i.1	8	9	206
t.1	3	4	1630	u.2	8	10	172
h.1	3	7	539	o.2	8	11	129
la1	4	6	128	a#2	9	11	364
				a#6	10	11	171

Table B.2 - Candidate allographs

Table B.2 shows a selection of the allographs matched for the example word. The first column indicates the template name - "i.1" meaning template 1 for a letter 'i', "ah1" meaning template 1 for a letter join between an 'a' shape and an ascender (see figure 5.4). Columns two and three indicate the segment positions of the beginning and end of the allograph within the sample word - the first allograph "i.1" occupies segment 1, for example. The fourth column is the scaled match probability for the template - see section B.4 for details of its calculation.

The information that there is a t-cross at the beginning of the word is used to eliminate any 't' allographs at the end of the word. This ensures that all 't's are crossed.

The actual candidate allograph list is much larger than this, containing some 450 elements. The letter graph is now constructed from this list.

B.4. Letter Graph

A letter graph is dynamically constructed from the candidate allograph list. A letter allograph is placed on the graph if a valid letter-join allograph exists which can precede it and join it to an entry already on the graph. The probability for the letter allograph is placed with it on the node and the probability for the letter-join allograph is placed on the connecting arc (see figure B.2). By ensuring letter/letter-join connectivity at this stage, the graph tracing routine is much more efficient.

Match probabilities are calculated for each segment individually. These are combined by simple multiplication to provide probabilities for larger portions of the script. This means that the probability for a two segment portion of script will

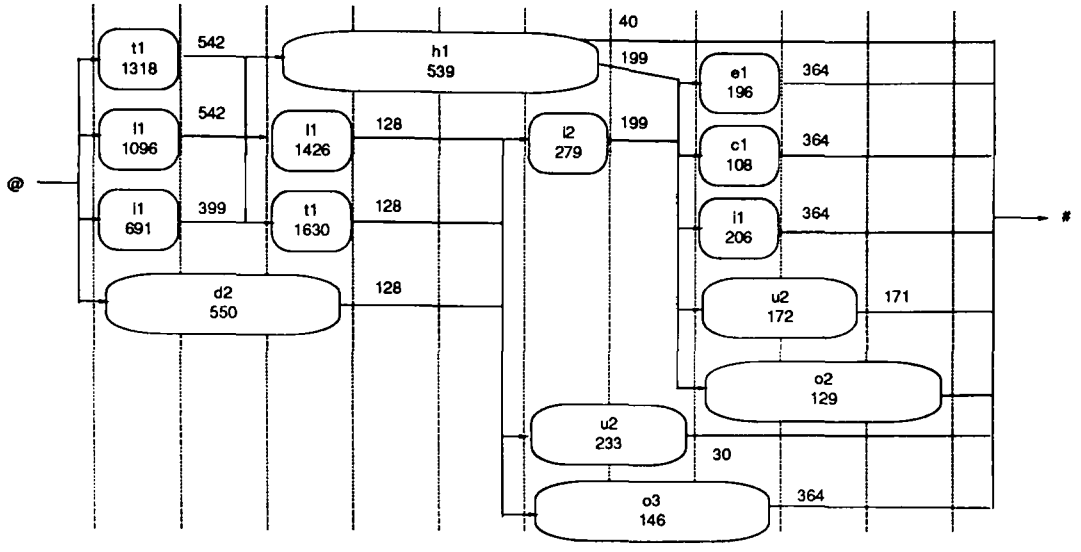


Figure B.2 - Letter graph

more than likely be less than for one segment. (If each segment matches with probability 0.3, say, the two segment probability will be $0.3 \times 0.3 = 0.09$.) For efficiency, it is often desirable to ignore candidate allographs with very low probabilities. In order to compare these, the probability is scaled by taking the nth root, where n is the number of segments. (This is the geometric mean of the probabilities.) The fourth column of table B.2 contains this geometric mean (multiplied by 10 000).

B.5. Candidate Words

The letter graph and dictionary tree are now traced simultaneously to provide a list of candidate words. A match probability is calculated for each word (see below). Figure B.3 shows the output that is presented to the user for verification. In a practical system, the word *the* is clearly the most likely, and the other possibilities would not be presented.

The word probability is calculated by multiplying the probabilities for each individual segment of the word. For the purposes of display to the user, the geometric mean is again calculated. This is the first number displayed in figure B.3. The number in parentheses is the ratio of the actual probability to the probability of the first word in the list. This is the value used to decide whether more than one candidate word should be displayed to the user. (Extra candidate words have been displayed in this example for clarity of explanation.)

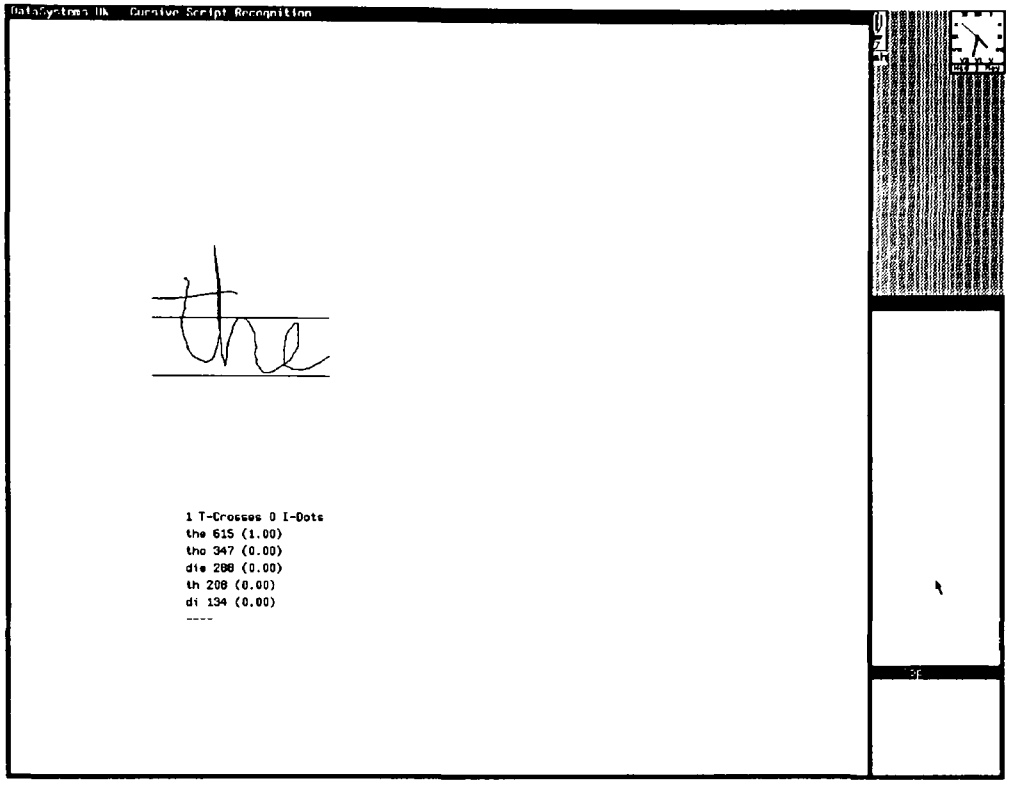


Figure B.3 - Output

For example, the word *tho* would be made up of t1(1318), join(542), h1(539), join(199), o2(129). Its probability is therefore calculated as

$$.1318 \times .0542 \times .0539^4 \times .0199 \times .0129^3 = 2.576 \times 10^{-15}$$

The geometric mean, multiplied by 10000 is therefore

$$(2.576 \times 10^{-15})^{1/10} \times 10000 = 347.6$$

The probability of any word that is among the 800 most frequently used in the English language is increased by a small factor, depending on its frequency. This ensures that more common words will be recognised more often. The probability for the word *the* has therefore been increased, since this is the most likely word.

A check of the number of diacritical marks in each candidate word is made to ensure compatibility with the recognised number of marks. Allowance is made, however, for ambiguity in the recognition of these marks, hence the appearance of the word *die* in the output list.

B.6. Training

After verification of the correct word by the user, the personal template database can be adjusted to include the new data. This provides an automatic method for training the system.

Appendix C

Published Papers

Copies of relevant papers published by the author follow.

THE ELECTRONIC PAPER PROJECT

D M Ford & Dr C A Higgins
Department of Computer Science
University of Nottingham
England

E R Brocklehurst
National Physical Laboratory
Teddington
England

1. INTRODUCTION

The National Physical Laboratory (NPL) has originated and led a coordinated UK research effort designed to exploit the potential of a novel form of computer interface. This interface consists of a high resolution, large area, flat panel display overlain with a transparent digitiser which is activated by a stylus. The combination of such a display and digitiser offers the opportunity to develop a new human-computer interface which subsumes and exceeds the mouse, light pen and touch panel.

Work has been in progress for five years. A demonstrator has been built which requires only the natural strokes of pen on paper as its input. Initial experiments with the demonstrator have shown the tremendous potential of such an interface.

This paper documents the concept of *Electronic Paper*, together with the hardware developments and ergonomic factors involved. This is followed by a description of the research and software developments made by the team. The main areas of research currently in progress are hand-printed character and symbol recognition, cursive script recognition and free-hand editing of text. Future work will include the detailed study of the ergonomic aspects related to the device, together with improvements to the hardware.

2. THE CONCEPT OF *ELECTRONIC PAPER*

Electronic Paper will consist of a flat panel display that can be written on with a stylus. The movement of the stylus is traced onto the screen, and the gestures interpreted and understood. *Electronic Paper* will be more than an input/output device - it will also include intelligence at a functional level. An interaction with the machine will involve pointing to an object, drawing a symbol or character, or writing. This leads to a very *user-natural* interface, since compound actions can be expressed very simply with a few handwritten symbols. For example, the handwritten instruction to move a paragraph merely requires the gesture of drawing an arrow from within the paragraph to its destination. With a conventional word processor it is necessary to specify the beginning of the paragraph, the end of the paragraph, select the move command from a range of options, and specify the destination - a large amount of data must be supplied for what should be an obvious action.

Electronic Paper offers the potential for people to interact with a machine without the need for training or recourse to instruction manuals. The manual for *Electronic Paper* will consist of just a small number of pages, with pictures of the few symbols necessary to operate the device.

There are several obvious uses for *Electronic Paper*, particularly in an office environment.

- The interface can be used as a jotting pad for notes and sketches. These can then be tidied up, sent electronically to another site for corrections or annotations, and returned for printing or inclusion in other documents.
- It can be used for the interactive manipulation and annotation of digitally encoded documents and images.
- The executive working outside normal office hours can produce documents that would normally be sent to a secretary.
- Two devices connected to each other via a telephone line could be used for remote, on-line, visual communications, enabling distant parties to converse and simultaneously illustrate their ideas with free-hand sketches.

3. HARDWARE DESIGN

The hardware for *Electronic Paper* relies on two crucial pieces of technology. The first, the transparent digitiser, has already been developed sufficiently to be incorporated in a flat panel display. The second is the flat panel display itself. The Japanese efforts to develop flat screen TV should ensure their availability at competitive prices.

The *PAD* (*P*en *A*nd *D*isplay), a prototype, has been built which demonstrates the potential of the interface. The hardware was specified as the result of joint discussions between NPL, the Central Computer and Telecommunications Agency (CCTA) of the British Government, and DataSystems UK Ltd. It has been designed and built by DataSystems UK Ltd.

The *PAD* consists of a flat AC plasma display panel, the Thomson TH7617, and a Scriptel SPD-1212 transparent digitiser. These are controlled by a Motorola 68000 processor. The display is 310mm (12 in) square, with a resolution of 3.4 pixels per mm (86 per inch). The digitiser has a resolution of 0.025mm (0.001 in) and an accuracy of 0.64mm (0.025 in). The prototype is at present desk-bound but it is hoped that the final product will be about 300mm (12 in) square, and 50mm (2 in) deep. Future designs should be light enough to be carried or used on the lap. The size and weight is very dependent on current flat screen technology.

The prototype operates in a *stand-alone* (executive toy) mode, where it simply mimics the operations of pen on paper, or it can be driven by a host computer via a serial line. Work is in progress to link the *PAD* directly to a SUN workstation, via the VME bus. This will enable the *PAD* to act as another window to the SUN operating system, or to replace the current CRT and mouse completely.

4. ERGONOMIC FACTORS

From initial observations of volunteers using the *PAD* it appears that this interface has considerable potential. Most users can get used to using the device after just a few minutes of practice, and are soon able to manipulate the stylus with considerable accuracy.

Problems that must be considered before construction of a final product include parallax, reflection, stylus design, screen colour, contrast and the presentation of the device itself and the information on the screen. The parallax problem is the most severe at present due to the construction method of the *PAD*. It was built using "off-the-shelf" components. The digitiser is 6mm (0.24 in) thick and the display glass with filter is 9mm (0.36 in) thick, leading to a total glass thickness of about 15mm (0.6 in) between the display pixels and the surface. Consequently, standing directly over the display increases the accuracy of pointing. The thickness of the digitiser could be reduced, since it was originally designed to be self-supporting.

Some work has already been done on the problems of parallax on a similar device, see Tappert *et al.*[1]

5. SOFTWARE

Work is in progress on many of the modules concerned with the project, at several UK sites. These include NPL, and the Universities of Nottingham and Essex. Several programs have been completed and transferred onto the *PAD* for testing, including a free-hand text editor, for correction of documents, and table and histogram drawing programs. Projects are underway researching into hand-printed character recognition and cursive script recognition. When all of the individual modules are complete, they will be fused together by a control program which will take the input data stream of coordinates and distinguish between cursive script, single characters, punctuation, symbols, editing corrections or diagrams.

5.1. Free-hand Editing

Two editors have been developed at present. A simple editor was developed at Trent Polytechnic for small corrections to unrecognised cursive script, for example to join incorrectly segmented words. The second, developed at NPL, is a more powerful general text editor for correction of typescript. This recognises commonly used correction marks, as specified by BSI standard BS5261(1976), and makes the necessary adjustments to the text. Thus, for example, a word may be deleted simply by striking it out with a line.

5.2. Tables and Histograms

Programs have been developed which enable tables and histograms to be roughly sketched free-hand on the *PAD*. These are then tidied up, lines are straightened, corners joined, and after editing with simple deletion and insertion symbols, the final diagram saved to file.

5.3. Hand-printed Character Recognition

Work is continuing at Essex University on recognition of any single character on the QWERTY keyboard, together with the option of a small number of user-definable characters, for example mathematical symbols.

5.4. Cursive Script Recognition

This is being developed at two sites. NPL are developing the pre-processing and feature extraction routines. Nottingham University are researching into whole word recognition, based on the extracted features, and dictionary lookup techniques. Much work is being channelled into appropriate feature set selection to find a compromise between easily recognised features at the feature extraction level, and features which are of most use to distinguish between words at the dictionary lookup level. See Higgins.[2]

Most researchers into cursive script recognition use a database of samples on which to test their routines. This database is usually collected on an opaque tablet, with either a real time visual check on a VDU screen, or is checked after collection. This has proved inconvenient and cumbersome, the subject requires a substantial practice time with the equipment, and the data so collected is unnatural and unsatisfactory. The interactive feedback provided by *Electronic Paper*, with the "ink" displaying the digitised image on the writing surface, means that the volunteer can dramatically improve the standard of his/her writing. In the near future we will be implementing a crude recognition system on the *PAD*. This will then be used to carry out experiments to discover to what extent a subject will adapt his/her handwriting style to assist the machine in recognition. For example, if the subject writes the word "had", and the computer recognises this as "naa", he/she may increase the height of ascenders. If this correction comes naturally, it will increase our chances of successful recognition.

6. CONCLUSIONS

NPL has shown the feasibility of *Electronic Paper* as an input/output device. Several modules are complete and are being field-tested; the recognition projects are making progress. The natural way in which a user can interact with the computer holds great potential for future developments, especially in a modern, electronic office environment.

References

1. TAPPERT, C. C., FOX, A. S., KIM, J., LEVY, S. E., AND ZIMMERMAN, L. L., "Handwriting Recognition on Transparent Tablet Over Flat Display," SID International Symposium, pp. 308-312, May, 1986.
2. HIGGINS, C. A., "Automatic Recognition of Handwritten Script," PhD Thesis, CNAA, 1985.

This paper was presented at the Third International Symposium on Handwriting and Computer Applications, Montreal 1987.

Computer Processing of Handwriting
Eds. R. Plamondon & C. G. Leedham

A TREE-BASED DICTIONARY SEARCH TECHNIQUE AND COMPARISON WITH *N*-GRAM LETTER GRAPH REDUCTION

DAVID M FORD and COLIN A HIGGINS

Department of Computer Science, University of Nottingham, Nottingham NG7 2RD,United Kingdom

Segmentation methods are commonly used in automatic handwriting recognition systems. Pre-processed input data is divided into smaller segments, which are then recognised, perhaps by comparing with pre-defined templates, and these recognised pieces recombined to form the required output. Recognition of any one segment is often ambiguous, resulting in a substitution set of possible templates. The segmentation itself may also be ambiguous.

Higher level context can be used to resolve some of this ambiguity by defining a set of valid output which can be expected from the system. This context can be applied at various levels ranging from legal letter sequences, to valid words, to correct sentence semantics. The letter and word context can be applied using a purely computational approach, whereas the sentence level semantics require techniques from the field of Artificial Intelligence.

This paper discusses the application of letter and word context, and in particular demonstrates a dictionary lookup technique which is applicable to many forms of automatic script recognition. A comparison is made between this method and a binary *n*-gram approach.

1. Introduction

In many syntactical and statistical handwriting recognition systems, unique classification is impossible and an ambiguous substitution set may be produced. This ambiguity can often be resolved if contextual information at a higher level can define a limited set of valid output. A dictionary or lexicon of this information can then be constructed, and the list produced by the recognition process reduced and validated. Often very large substitution sets are produced which need to be checked quickly and efficiently against a large

dictionary. This is especially the case with cursive script recognition systems.

A number of different techniques have been proposed to make use of contextual information.

Statistical information about the transition probabilities between letters can be used to remove or reduce unlikely letter sequences and produce the required output (e.g. Neuhoff (1975); Riseman and Hanson (1974); Hull and Srihari (1982)). This is often referred to as a *bottom-up* technique. These techniques are usually very efficient, but do not guarantee valid output words.

Another approach assumes that the written word comes from a fixed dictionary and the nearest matching word provides the required output (e.g. Duda and Hart (1968)). This is often referred to as a *top-down* technique. These techniques always produce a valid output word but can be inefficient to implement.

A number of hybrid methods have been suggested to balance out the advantages and disadvantages of the two approaches (e.g. Srihari *et al.* (1983); Shinghal and Toussaint (1979)). Some of these techniques are briefly discussed below.

A technique incorporating the advantages of both top-down and bottom-up approaches is presented here. This appears to be ideally suited to input data which can be represented as a directed letter graph, for example as is produced by cursive script or character recognition systems. Each node of the letter graph contains information concerning the match of a segment to a particular template, together with links to any possible succeeding nodes. The data structures used for the dictionary and graph are directly comparable, leading to a very efficient technique. An algorithm is designed and implemented, and experiments are reported which investigate the characteristics and suitability of this technique. Different internal structures for the dictionary are discussed and compared.

Experiments have been carried out to compare the performance of this algorithm with a method which has been used previously to reduce the letter graph using binary n-grams. Results are presented which show that the new technique is usually much faster than a 4-gram system, and a manifold increase in performance is achieved with very large graphs.

2. Typical Output from Segmentation Based Text Recognition Systems

There are two main approaches to the problem of text recognition. The first involves attempting to recognise a complete word as a whole unit - *whole word recognition.* The second involves segmenting the word into smaller units which are then recognised - *segmentation based recognition.* Only the second of these will be discussed in this paper.

A typical segmentation based text recognition system operates in several distinct stages. Firstly, the raw data, whether it be a scanned image or a time-ordered sequence of coordinates from an on-line digitising tablet, is pre-processed to smooth, straighten and normalise the script. Secondly, a list of possible segmentation points is produced and thirdly, features are extracted from the segments which allow recognition to take place. The segmentation and subsequent recognition may be achieved at the letter level, or at the sub-letter level, where more than one segment may be combined to produce a single letter.

The letter-matching routine can fall into one of two main categories:

(a) those which produce a single word as output consisting of the most likely letters which span the written sample;

(b) those which produce a list of possible letters in each segmentation position together with a weighting of their likelihood.

Recognition systems belonging to type (a) make a binary decision at this stage as to where letter segmentation occurs - we will refer to this as a *fixed letter-segmentation.* Those belonging to type (b) may retain all of the possible segmentation points for future use and will be referred to as *ambiguous letter-segmentation* (no decision has been made at this stage as to exactly where the letters occur in the written word).

Example

With a fixed letter-segmentation recognition system, the most likely word is produced as output. For example, if the word *dog* was presented to the system, the letter matching algorithm may make a mistake, for instance, in the second letter position and produce the output *d-a-g.* It is then the task of any

contextual post-processor to correct the output and produce the most likely real word.

An ambiguous letter-segmentation system produces a list of possible candidates in each segment position, together with a *certainty weight* to indicate the closeness of the match. For example, if the word *dog* was presented to the system, it may produce two options for the second character, perhaps an *'a'* with high certainty weight of 60 or an *'o'* with a lower certainty weight of 40.

In cursive script recognition systems, individual characters are permitted to run into each other, with the problem that segmentation of the script into letters is ambiguous. For example, if the word *dog* is written cursively, there is the possibility that the word may be *clog*, where the segmentation after the first letter is ambiguous.

One way to represent the output from such a system is as a directed graph (Hayes (1980); Higgins and Whitrow (1985); Peleg (1979)). Figure 1 shows a simplified letter graph that might be produced by a recognition system acting on the input word *dog*. The '@' symbol represents the start of a word and the '#' symbol represents the end of a word. The graph is traversed from left to right, yielding a list of all possible combinations of letters that the original data might represent, i.e. {dcig, dog, dag, clcig, clog, clag}. The certainty weights can be attached to each letter on the graph and combined as the graph is traced to produce a ranking for each word that is produced.

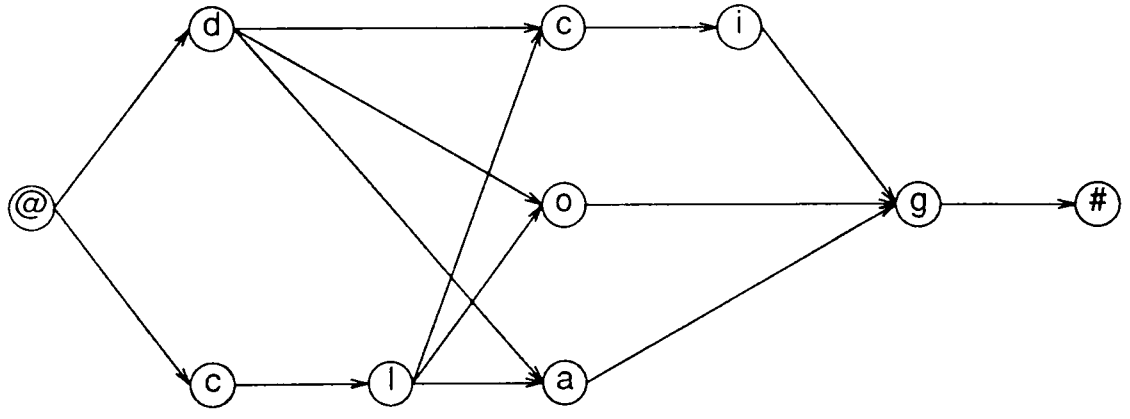


Figure 1 - A simple letter graph

3. Use of Contextual Information from Dictionary Sources

3.1 Fixed Letter-Segmentation

Most of the approaches to using contextual information have been based on constructing a post-processor for the output from fixed letter-segmentation recognition systems, ie the output from the recognition stage is a single, most likely word. The contextual disambiguation process takes a *word*, consisting of the most likely letters that span the word, as its input and returns the most likely written word as its output. There are several techniques which have been widely reported, the most common are listed below.

3.1.1 N-Gram Techniques

The probability of any individual n-letter sequence occurring can be calculated by examining large pieces of text. These probabilities can then be used to calculate the most likely written word given the output word from the recognition system, (see Riseman and Ehrich (1971)).

3.1.2 Viterbi Algorithm

The Viterbi Algorithm (VA) takes the output word from the recognition system and, using statistical information on the sequence of letters in English and likely errors from the recognition system, calculates the most likely input word. Viterbi (1967) first described the algorithm. Forney (1973) provides a thorough tutorial introduction to the theory behind the algorithm. Neuhoff (1975) described how it could be applied to the problem of text recognition. Various authors have discussed its application, including Riseman and Hanson (1974) and Hull and Srihari (1982) who compared its performance against a binary n-gram approach.

The VA makes use of a *confusion matrix* of *a priori* probabilities observed from the recognition system, together with the transition probabilities between characters. In other words, the probability that a given letter may be mis-recognised as another letter is calculated and stored, together with the probability that it can be preceded or followed by any other character.

A $26 \times l$ node trellis is constructed, where l is the length of the word, linking every letter with every other letter (see Figure 2). On the nodes of the trel-

lis are the confusion probabilities, and on the arcs are the transition probabilities. [Notation: The confusion probability that a written letter X will be mis-recognised as a Y is written P(X | Y). The transition probability that a letter X may be followed by a letter Y is written P(X - Y).] By tracing a path through this trellis, and combining the probabilities on the arcs and nodes included in the path, the probability that the traced word could have been the original input word can be calculated. The algorithm finds the most likely path through this trellis, so yielding the most likely interpretation of the input word.

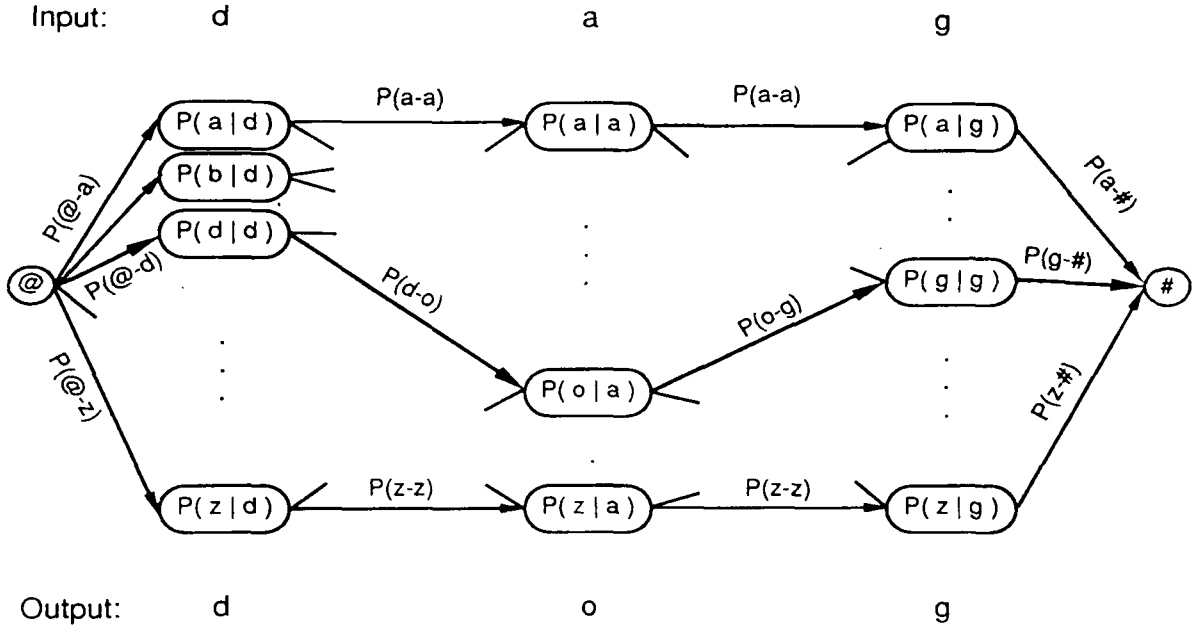


Figure 2 - Viterbi Algorithm Trellis

3.1.3 *Dictionary Viterbi Algorithm*

The VA produces the most likely interpretation of the written word, but does not guarantee that the word exists. The Dictionary Viterbi Algorithm (DVA), developed by Srihari *et al.* (1983), is an improvement on the VA making use of a simultaneous full dictionary search, in place of the transition probabilities, to ensure that only a valid word is produced. (A similar technique was used by the HARPY speech understanding system, using words and sentences in place of letters and words, see Lowerre and Reddy (1980).) The dictionary lookup uses a *trie* structure to store the dictionary. This is described in Section 3.2.2.

3.1.4 Modified Viterbi Algorithm

Shinghal and Toussaint (1979a) describe another variant of the VA, called the Modified Viterbi Algorithm (MVA). Here a heuristic depth of search d is set by the user so that only the d most likely probabilities in each letter position are checked. The computational overheads of the VA are thus reduced, as only a $d \times l$ trellis needs to be traced, but the performance degrades as d is reduced.

3.1.5 Predictor-Corrector Algorithm

Shinghal and Toussaint (1979b) further improve on the MVA by combining it with an efficient dictionary lookup algorithm (DA). This is called the Predictor-Corrector Algorithm (PCA).

The dictionary is partitioned into sub-dictionaries of same-length words. Each sub-dictionary is then sorted by *value*, where the *value* is calculated by combining the transition probabilities of the letters of the word. This *value* was found to be nearly always unique for any sub-dictionary. A binary search is used to see if the output word from the MVA exists in the dictionary. If so, then that is taken as the required output. If not, then the DA is employed. A *score* is calculated for the nearest f words to where the output word was expected. f is a heuristic set by the user. The *score* is calculated by combining the transition probabilities between the letters with the confusion probabilities. The word with the largest *score* is the required output.

Shinghal (1983) describes a further enhancement to the PCA, where the n most likely words from the MVA are checked in decreasing order to see if they exist in the dictionary. If none of them exist, the DA is employed. The value of n is determined by experiment.

3.2 Ambiguous Letter-Segmentation

The problem with all of the systems based on the VA is that they do not allow for incorrect letter segmentation. For example if the word *dog* is misrecognised as *clog* then the algorithm cannot produce the correct word and the system will fail. An ambiguous letter-segmentation retains all of the possible segmentation points within the letter graph representation.

For an ambiguous script recognition system, the contextual disambiguation process takes a letter graph as its input, and produces as its output a list of

words which have been checked, in some way, against a lexicon of dictionary information. One simple way to do this is to trace every path through the graph and check each resulting word against a list of valid words. This is not practical since the letter graph is usually very large with many paths. Even the most efficient searching algorithms will take a significant amount of processing time to verify a very large list of words. (Commercial spell-checking software cannot be used since, for efficiency, the algorithms they employ usually rely on the fact that the input words will either be correct, or very close approximations to real words. Completely random sequences of letters may often be accepted as valid words, eg UNIX† spell.) A much more efficient technique is needed.

Two main practical techniques are described and compared below.

3.2.1 *Binary N-Gram Graph Reduction*

A list of valid n-letter sequences is produced by analysing a dictionary of allowable words. This list can be used to reduce the size of the letter graph by removing invalid portions. Section 7.1 gives details of one such method.

3.2.2 *Dictionary Tree*

A dictionary or word-list is restructured in the form of a tree, based on the *trie* structure suggested by Knuth (1973). This is shown pictorially in Figure 3, where the tree represents the word list {a, an, and, at, be, bet, but, by}. Each of these words can be found by tracing a path from left to right. The '@' symbol represents the start of a word and the '#' symbol represents the end of a word.

The *trie* can be used as an efficient structure to store a dictionary for a variety of applications, but is especially applicable for contextual post-processing of script recognition systems. The DVA proposed by Srihari et al. (1983) used a dictionary *trie* at the same time as the Viterbi lattice is traced to guarantee that a valid word is found. Bozinovic and Srihari (1982) combined a stack-decoding search algorithm with a *trie*-structured dictionary with a small dictionary of 1027 words. Bozinovic and Srihari (1989) used a similar approach but added a depth of search heuristic to limit the computation needed.

---

† UNIX is a trademark of Bell Laboratories

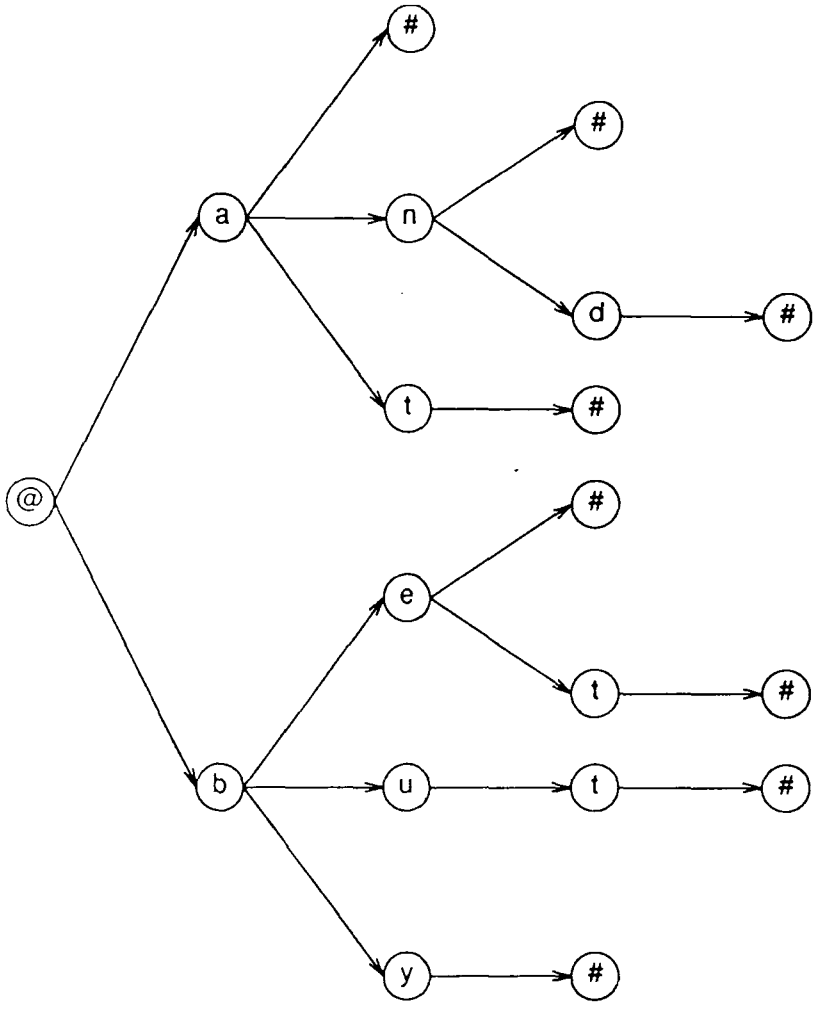


Figure 3 - Dictionary Tree - Pictorial Representation

Our method shows that a letter graph, such as that shown in Figure 1, can be efficiently traced using a recursive procedure. Such a procedure might take the head-node of the graph on which it is to act as its parameter and call itself recursively, passing each sub-graph that the head-node points to as a parameter. This will carry out a depth-first trace of the graph. Using a tree

structure, the dictionary can be traced simultaneously. As each arc of the letter graph is traced, if the corresponding arc exists in the dictionary tree, then the word is valid *up to that point.* When the end-of-word marker is reached in the graph and tree, the word traced out exists in the dictionary. An invalid word in the letter graph will be rejected *as soon as* an arc cannot be found in the dictionary tree. It is not necessary to continue tracing the graph past this point. This is advantageous since it limits the time taken to search the dictionary, and allows the dictionary to be very large without seriously reducing the performance.

All valid words that exist in the letter graph can be produced in this way with a weighting of their likelihood. No information is thrown away at this stage which may prove useful later on, for example, if more than one recognition routine is used in parallel and the outputs combined.

4. Computer Representation of the Dictionary Tree

A dictionary structured in this way can be accessed very quickly. It has the disadvantage of requiring a large amount of memory, since the data structure remains resident while the program is running. This is probably the reason for its rejection in the past, but as computer memory is becoming larger and more readily available this is no longer a major problem. We consider a number of different ways to construct this data structure within a computer program.

4.1 Discussion of Possible Data Structures

Knuth (1973) suggests a static data structure for the dictionary tree. This consists of a flag to indicate whether a word can end here and an array of 26 element integer arrays together with 26 boolean flags. Each column position points to the column containing the next letter in a valid word, and a corresponding flag to indicate if a word can end at that point. This array is very sparse and would take up a large amount of memory. Nowadays, dynamic memory allocation allows more efficient data structures to be used.

A simple dynamic data structure might consist of a tree of linked nodes, where each node contains 26 pointers to possible successor nodes (see Figure 4). In this particular layout, the letter is implied by the position of the

pointer in the array. Such a structure will be extremely sparse and have huge memory requirements.

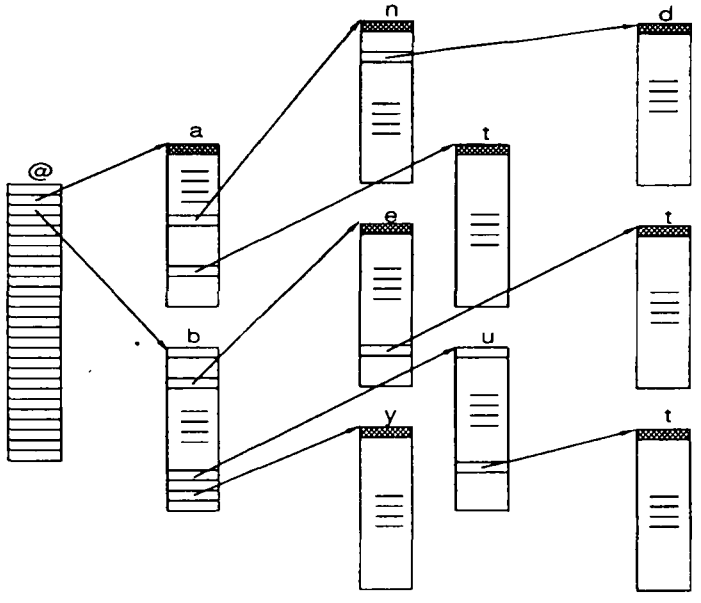


Figure 4 - Dictionary Tree - Logical Representation

A more compact dynamic structure is shown in Figure 5. In this diagram each large square represents a node of the tree consisting of a key letter, at a certain level, and two pointers. The upper pointer points to a list of letters, at the next level, that are permitted to follow this letter. The lower pointer points to the remaining list of letters, at the current level, which can occupy the same letter position. This diagram is a representation of the tree shown in Figure 3.

Figure 6 shows an internal representation for the letter graph in Figure 1. A node consists of a letter and a pointer to a list of arcs which, in turn, point to possible successor nodes.

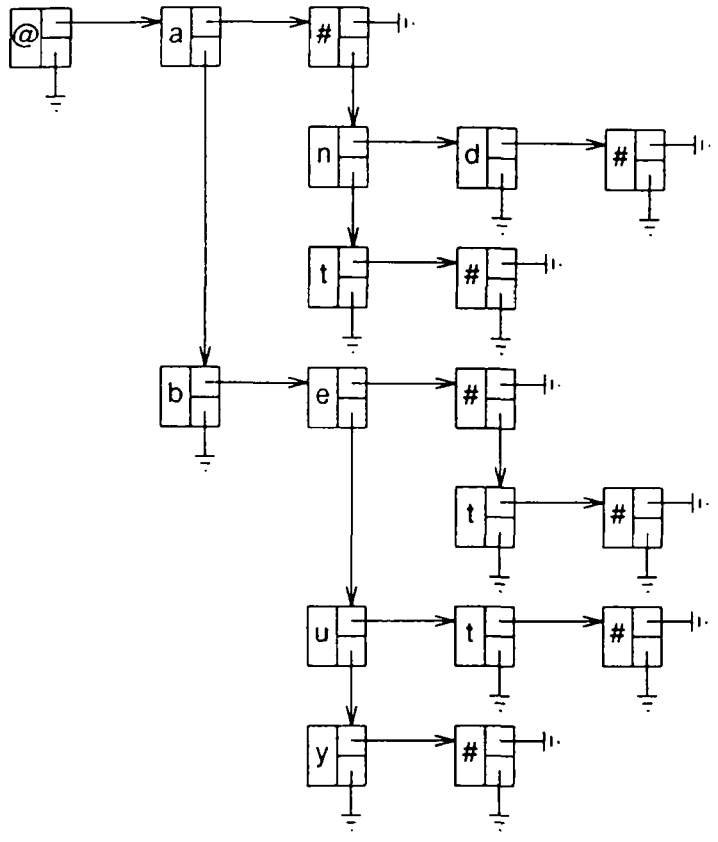


Figure 5 - Dictionary Tree - Physical Implementation

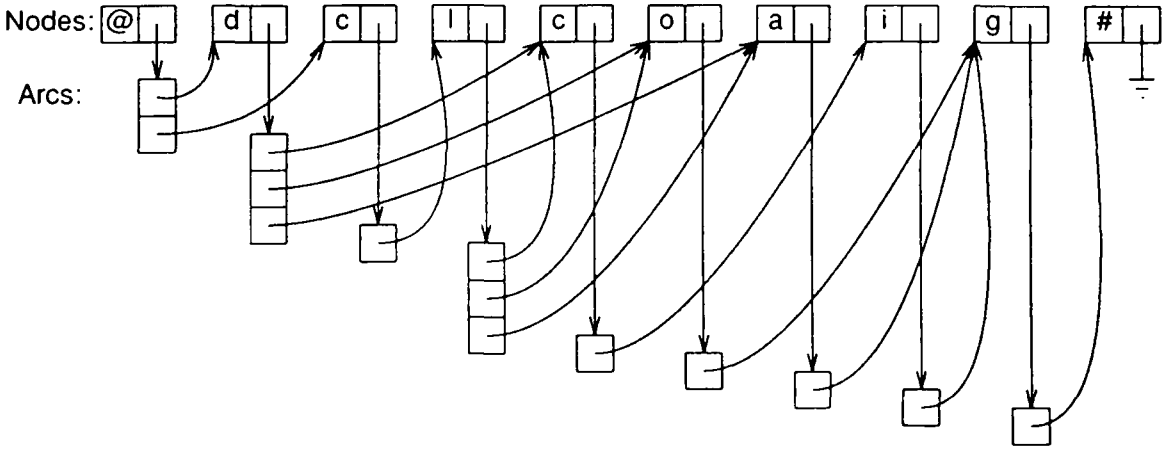


Figure 6 - Letter Graph - Physical Implementation

4.2. Possible Enhancements

The final choice of internal representation involves consideration of the trade off needed between access speed and memory requirements. There are several different structures which can be used depending on the most important criterion.

4.2.1 Increased Speed of Access

We define the list of letters, at the next level, which can follow a particular letter in the dictionary tree as its *sub-list*. The search method for a structure similar to Figure 5 can be speeded up by reducing the time taken to search for a letter in a sub-list. The sub-list of letters can be ordered in several ways. If the letters are in a random order, it is necessary to search to the end of the list if the key cannot be found. However, if the ordering is known it is only necessary to search as far as the expected position of the letter.

Several different orderings are possible. Alphabetical ordering has the advantage of allowing the ASCII code to imply position in a list. Sorting on frequency of occurrence in English, or frequency of generation from the recognition system may produce a faster search, but may require extra memory to store a representation of the order. Actual speed will be dependent on the letter graph itself; a graph which contains many valid words will take a different amount of time to trace than a similarly sized graph with many invalid words. For example, a list optimised for finding a commonly occurring letter will not be optimised to show that an infrequent letter is not present in fact, it will be the worst possible ordering.

4.2.2 Reduced Size of Data Structure

The size of the data structure can be reduced by using *tail-end compression*. If the end of a word is unique and is not common with any other word, then the dictionary structure can be reduced by constructing a special *end-of-word* node which contains the rest of the word as a string, without the need for extra nodes and pointers (see Figure 7). The disadvantage of this structure is that the dictionary cannot easily be maintained. For example, to add an extra word may involve unpacking the tail of an existing word, adding the new word, then re-packing the two new tails. Also, since the code needed to search a tail will be different to that needed to search the tree, switching lookup code would degrade the speed of lookup.

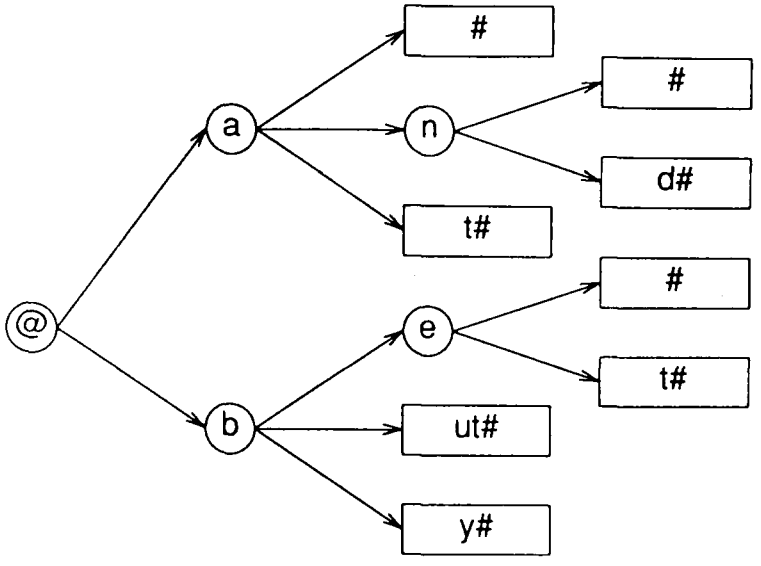


Figure 7 - Dictionary Tree - Tail End Compression

4.2.3 *Improved Performance*

Our technique was originally developed for use as a post-processor for a cursive script recognition system. Often handwritten script samples have very poor quality sections where either the word has been written badly, or has been poorly digitised. A recognition system may not be able to identify, in any way, what has been written in these sections and may insert a *wild card*, representing every possible character, into the letter graph in that place. Alternatively, it may be clear that a segment has, for example, a descender, but no other information is obvious, so a descender subset wild card, representing every letter with a descender, could be inserted. If the poor section occurs at the end of the word, the performance of the tree structure described above will not be adversely affected, since most of the paths through the letter graph will be rejected before the highly ambiguous section is reached. However, if it is at the beginning of the word, then there will be a large amount of ambiguity at the beginning of the graph, resulting in a large amount of wasted computation.

If the dictionary tree was also structured in reverse, starting at the end of words and working towards the beginning, it would be possible, in these

cases, to verify the letter graph backwards, and so speed up the process. Similarly, if both ends of the original data sample are poor, it may be possible to work from the middle outwards. An ideal data structure might have multiple linkages, starting at each letter position, so that the dictionary search would always begin at the least ambiguous part of the graph. It is intended that the properties of such a structure will be investigated in future implementations.

It should be noted that a dictionary lookup system based on a letter graph will fail if the correct letter is not in the letter graph hence the need for wild-cards when no letters are obvious candidates. A fixed letter-segmentation recognition system does not suffer from this problem since it is possible to change one letter to another.

5. Implementation Selected

For our implementation, we initially chose a simple approach, for ease of coding, to ascertain whether this data structure was suitable in practice. The data structure is an exact representation of Figure 5. Lists are ordered alphabetically, and the tree is linked only from start to finish. It was discovered that its performance and size were perfectly adequate for our requirements, even with very large ambiguous letter graphs.

To test the performance of this implementation, a 210,000 word dictionary was used. This is probably much larger than would be needed in a practical system. A substitution set was used to generate simulated letter graphs of possible output from typed input words. A test set of 2,100 letter graphs was verified using the system running on a SUN 3/160 with 8 Mbytes of main memory, timeshared with other users but with light loading. The average real time needed per word was 0.45 seconds, and the actual CPU time used by the system averaged 0.19 seconds per word. In a practical system the lookup process would be quicker, since the times quoted above include routines to set up the data structure for the dictionary, to time the program and to calculate and print various analytical results.

6. Size of Data Structure

The size of the dictionary data structure can be measured by counting the number of nodes in the tree. Experiments have been carried out using differ-

ent sized word lists to investigate the effect of dictionary length against data structure size. As can be seen from the graph in Figure 8, this is roughly a linear function of dictionary size.

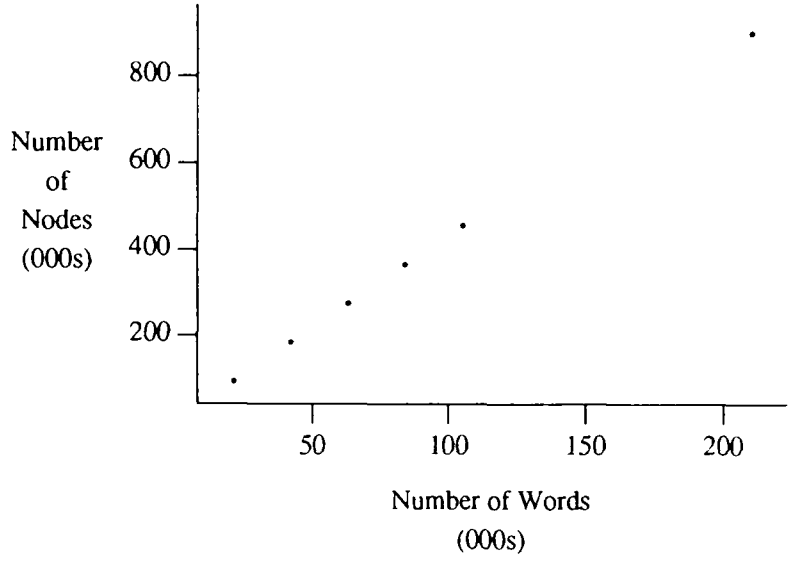


Figure 8 - Size of Data Structure vs. Dictionary Size

For example, it seems that a dictionary of 60,000 words will probably be sufficient for a usable script recognition system. (This would allow for a vocabulary of about 20,000 root words with plurals and verb endings.) Assuming that a node consists of a character and two pointers, it can be seen from the graph that such a dictionary would require approximately 2.4 Mbytes of memory in our implementation. This is not an unreasonable amount of memory to expect in modern computer workstations.

The size of the data structure for a particular dictionary is dependent on the *compactness* of the dictionary, ie whether the words have common roots. Adding a word which has the same root as a word already in the dictionary increases the size by less than adding a unique word. This means that the dictionary can contain all participles of verbs, plurals etc without drastically increasing its size, avoiding the need to construct these words by a rule-driven system.

To investigate the effect of compactness on dictionary size, two sub-dictionaries of 21,000 words were selected from a word list of 210,000 words; one by selecting the first word from each group of 10, the other by selecting the first 10 words from each group of 100. As expected, the first of these was less compact and used 141,138 nodes, the other more compact dictionary used only 98,981 nodes.

7. Comparison with Binary 4-Gram Graph Reduction

7.1 The Graph Reduction Process

This technique makes use of the existence or nonexistence of four-letter sequences in English. Higgins (1985) reported that four is the ideal length of gram to use, since only approximately 5% of 4-grams are valid in English, and the number of possible grams, 26^4 = 456,976, can be reasonably stored as a binary array occupying just under 56 Kbytes of memory. A much larger percentage of 3-grams are valid, and 5-grams would require about 1.5 Mbytes of memory for the binary array without much gain in context.

The letter graph is supplemented by adding an extra start and stop node at the beginning and end of the graph. This is so that the opening two and three letter sequences can be checked using the same 4-gram approach.

The graph reduction process can be implemented in many different ways, see Whitrow and Higgins (1985). We have selected an efficient technique, which uses a similar letter graph structure and tracing algorithm to the dictionary tree search algorithm, for the purposes of comparison. This technique involves recursively tracing each path through the letter graph, maintaining pointers to the last four letters accessed. At each step the current four letter sequence is compared against the list of valid 4-grams. If the sequence *is* valid, then the arcs connecting the letters are marked. The three arcs are marked differently depending on which letters of the sequence they connect. An arc which has been in each of the three connecting positions of a four letter sequence is marked as *used*. After the traversal is complete, the graph is stripped of all *unused* links, the arc-markings are cleared, and the process repeated until no more links are removed. In this way a much reduced letter graph is produced, which can be more readily checked against a dictionary using a straightforward approach.

7.1.1 Speed of Reduction

The speed of graph reduction for the technique described above is dependent on the number of possible paths through the graph, which is in turn dependent on the length of the word and on the *confusion level* of the letter graph. This is defined to be the number of letter options at each letter position. The graph in Figure 9 shows how the time taken for the reduction greatly increases as the confusion level increases. The graph shows the time taken when reducing various possible letter graphs generated from the input word "test", with increasing ambiguity at each letter position.

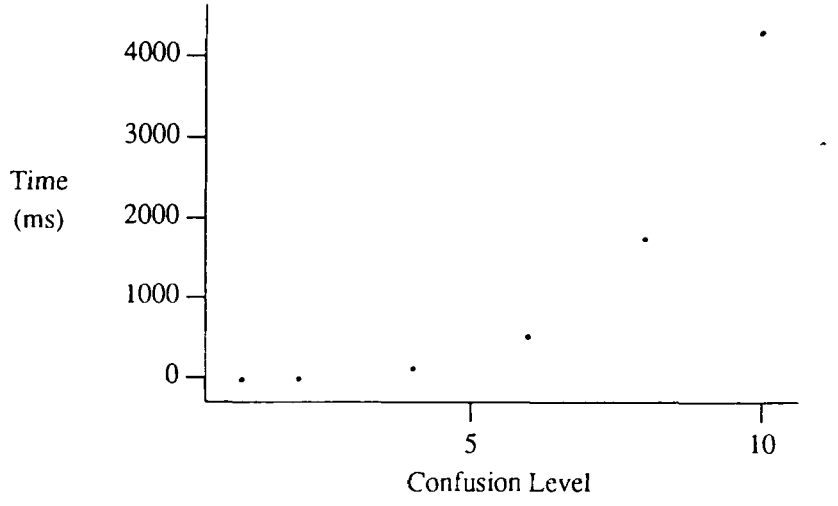


Figure 9 - 4-Gram Reduction Lookup Time vs. Confusion

7.1.2 Output

4-gram graph reduction produces a much reduced letter graph, but this will usually still contain invalid words when all of the paths are traced. In a sequence of letters each individual gram may be permissible but this will not guarantee that the whole word is valid. Also, an arc in the letter graph can only be deleted if *no* valid gram uses it. A remaining arc will then still allow invalid grams to pass through it. (Figure 10 shows a sub-graph of a letter graph, where the gram *ebcf* may be invalid but *abcd* valid. The arc marked '*' must therefore remain in the graph, so the invalid gram will still be pre-

sent when the graph is traced.) It is therefore still necessary to check each word against a dictionary to guarantee validity. This will incur the overhead of additional processing time.

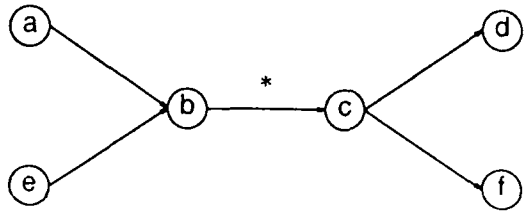


Figure 10 - Arc Deletion

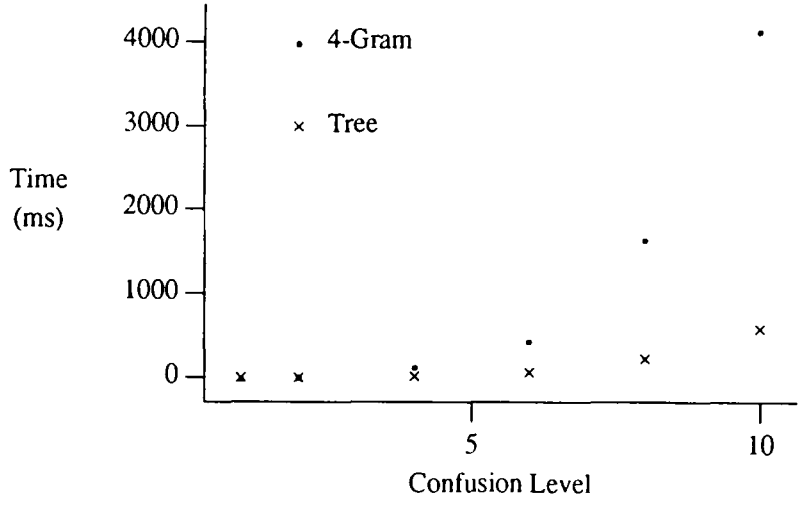


Figure 11 - Comparison - Lookup Time vs. Confusion Level

7.2 Speed Comparison with Dictionary Tree

The graph in Figure 11 shows the time taken for the graph reduction and the dictionary lookup, plotted against confusion level of the input graph, again for the word "test". It can be seen that with very small graphs the two techniques are comparable, but, as the ambiguity increases, the dictionary lookup is considerably faster. It should also be remembered that the 4-gram output still needs to be verified against a dictionary.

8. Conclusions

Whereas a large amount of work has been done in the field of contextual post-processing of text recognition systems, not many of the techniques suggested successfully address the problems caused by ambiguous letter-segmentation, which are common with cursive script. The tree-based dictionary lookup technique described here is an ideal application of letter and word context for the output from text recognition systems and allows for ambiguous letter-segmentation of the script. It is extremely fast and efficient, can be used with a large dictionary or word list and produces all possible output words, with no loss of information that can occur when arbitrary cut-offs are applied. The dictionary contains every word which can be recognised by the system, without the need for prefix and suffix generation. It is simple to add extra words to the dictionary without greatly increasing its size. The technique is especially appropriate for very ambiguous letter graphs, typically produced by cursive script recognition systems. Wild card substitution can be used in areas where the script is very poorly written or digitised.

The performance is superior to n-gram graph reduction techniques, and is in fact a super-set of the n-gram approach, since all possible values of n are effectively applied simultaneously.

References

Bozinovic, R. and Srihari, S.N., "A String Correction Algorithm for Cursive Script Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4** no. 6, November 1982, pp. 655-663.

Bozinovic, R. and Srihari, S.N., "Off-Line Cursive Script Word Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11** no. 1, January 1989, pp. 68-83.

Duda, R.O. and Hart, P.E., "Experiments in the Recognition of Hand-Printed Text: Part II - Context Analysis," *AFIPS Conference Proceedings*, **33**, 1968, pp. 1139-1149.

Forney, G.D.Jr., "The Viterbi Algorithm," *Proceedings of the IEEE* **61** no. 3 March 1973 pp. 268- 278.

Hayes, K.C., "Reading Handwritten Words Using Hierarchical Relaxation," *Computer Graphics and Image Processing* **14** 1980 pp. 344-364.

Higgins, C.A. and Whitrow, R., "On-line Cursive Script Recognition," *First IFIP Conference on Human-Computer Interaction - INTERACT 84*, 1985, pp. 139-143.

Higgins, C.A., "Automatic Recognition of Handwritten Script," PhD Thesis, CNAA, 1985.

Hull, J.J. and Srihari, S.N., "Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4** no. 5 September 1982 pp. 520-530.

Knuth, D.E., in *Sorting & Searching, The Art of Computer Programming*, **3** Addison-Wesley, 1973, pp. 481-487,.

Lowerre, B. and Reddy, R., "The HARPY Speech Understanding System," in *Trends in Speech Recognition*, ed. W. Lea, 1980, pp. 340-360.

Neuhoff, D.L., "The Viterbi Algorithm as an Aid in Text Recognition," *IEEE Transactions on Information Theory* March 1975 pp. 222-226.

Peleg, A., "Ambiguity Reduction in Handwriting with Ambiguous Segmentation and Uncertain Interpretation," *Computer Graphics and Image Processing* **10** 1979 pp. 235-245.

Riseman, E.M. and Ehrich, R.W., "Contextual Word Recognition Using Binary Digrams," *IEEE Transactions on Computers* **20** no. 4 April 1971 pp. 397-403.

Riseman, E.M. and Hanson, A.R. "A Contextual Post-processing System for Error Correction Using Binary n-grams," *IEEE Transactions on Computers*, **23** no. 5 1974 pp. 480-493.

Shinghal, R. and Toussaint, G.T., "Experiments in Text Recognition with the Modified Viterbi Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1** no. 2 1979a pp. 184-192.

Shinghal R. and Toussaint, G.T.,"A Bottom-up and Top-down Approach to Using Context in Text Recognition," *International Journal of Man-Machine Studies* **11** 1979b pp. 201-212.

Shinghal, R., "A Hybrid Algorithm for Contextual Text Recognition," *Pattern Recognition* **16** no. 2 1983 pp. 261-267.

Srihari, S.N., Hull, J.J. and Choudari, R., "Integrating Diverse Knowledge Sources in Text Recognition," *ACM Transactions on Office Information Systems* **1** no. 1 January 1983 pp. 68-87.

Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory* **IT-13** April 1967 pp. 260-269.

Whitrow, R. and Higgins, C.A., "The Application of n-Grams for Script Recognition," *Proceedings of the Third International Symposium on Handwriting and Computer Applications*, Montreal, Canada, July 1987, pp. 92-94.