PRACTICAL CURSIVE SCRIPT RECOGNITION

DISSERTATION

Presented to the Graduate Council of the

University of North Texas in Partial

Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

By

Johnny Carroll, B.S., M.S.

Denton, Texas

May, 1995

PRACTICAL CURSIVE SCRIPT RECOGNITION

DISSERTATION

Presented to the Graduate Council of the

University of North Texas in Partial

Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

By

Johnny Carroll, B.S., M.S.

Denton, Texas

May, 1995

Carroll, Johnny, Practical Cursive Script Recognition. Doctor of Philosophy (Computer Science), August, 1995, 161 pp., 3 tables, 30 figures, references, 153 titles.

This research focused on the off-line cursive script recognition application. The problem is very large and difficult and there is much room for improvement in every aspect of the problem. Many different aspects of this problem were explored in pursuit of solutions to create a more practical and usable off-line cursive script recognizer than is currently available.

The scope of the project involved a complete solution to most aspects of the problem. Preprocessing was refined via a new thinning algorithm and a new Finite Induction (FI) based vectorization algorithm. Feature extraction was performed by extracting features from the singularity graph of the line drawing instead of the line drawing itself. The feature graph was designed to provide a very expressive, flexible, and efficient data structure so all existing features of a singularity graph can be easily scanned and associated locally. A new and powerful FI based character extraction mechanism was created and studied. Character extraction, word segmentation, and word classification were performed iteratively in light of the context of the lexicon using split n-gram indices to assist in word classification and search space reduction. The use of heuristics was employed and studied in the recognition of punctuation. Also, an adaptable system was designed so that the system could adapt to individual handwriting styles of experiment participants.

Another focus of this dissertation involved exploring how the pattern recognition technology known as Finite Induction could be employed in pursuit of applications of this nature. FI was a major contributor in two of the phases. FI technology was adapted for use and successfully employed in the line segmentation process and in the character extraction process.

An experiment was conducted which demonstrated that with reasonable training of the system and reasonable restrictions placed upon the writer of the cursive script, successful hand written cursive script recognition is feasible and usable systems are within reach.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Jan Carroll, for understanding, sympathy, support. motivation, and tender love during the work on the project. I also appreciate her endurance during months and years without the attention that she needed and deserved.

I would also like to thank my children, John Phillip and JoEtta Carroll, for enduring the long hours without their father while I was working on this project and for continuing to love and support me when I did not deserve it.

Also, mountains of thanks go to my parents, Arthur and LaVerne Carroll, who provided unending support and motivation to continue.

In addition, I really appreciate my research advisor, Dr. Paul Fisher, for his direction and motivation, the hours he spent as reviewer and editor, and the FI technology which was employed in this research.

Finally, I would like to say thanks to the members of my committee, Dr. C. C. Yang, Dr. Gerald Knezek, and Dr. Steve Tate, for the encouragement to finish and also for the hours of work they spent in scrutinizing the document.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

1.1.1 Pattern Recognition

In the broad field of Computer Science, the sub-field of **pattern recognition** has always been of great importance. In recent years, this importance has grown. If an application requires pattern recognition, then the application needs to examine a set of input data and determine if a certain pattern is present within the data; or, there may be a set of patterns and the application must search the input data to determine the presence of one or more of the patterns while retaining the knowledge of which pattern was recognized.

In the field of Artificial Intelligence, this process is known as **classification** and the part of the application that performs the classification is known as the **classifier** [Rich, 1983]. Actually an entire sub-field in AI known as expert systems is concerned primarily with pattern matching of various types. Rules are placed in expert system's rule bases along with certainty values and when an expert system notices that the input matches a pattern recognized by some of the rules and certainty values, it can produce a diagnosis of the situation described by the input and prescribe whatever actions are associated with the given rules [Forsyth and Rada, 1986].

Many applications exist that use pattern recognition to accomplish some or all of their goals. Applications exist that successfully perform object recognition in an image [Karbacher, 1990] [Jagadish and Ikeuchi, 1991], speech recognition [Lowerre and Reddy, 1980] [White, 1990], optical character recognition [Crawford 1991], target detection [Clark and Velten, 1991] [Sadjadi and Bazakos, 1991], illness diagnosis from a recognized pattern of symptoms (Mycin/Neomycim project) [Barr and Feigenbaum, 1981], and countless others.

The field of pattern recognition continues to grow almost exponentially in importance as the state of the art in computer hardware and software moves toward computers that have the capacity for:

- vision, where recognition of an object in the field of view is important,
- hearing sound and speech, where recognition of sounds and even spoken language is possible,
- reading, where information in documents, created on-line, can be extracted,
- reading, where information in documents created off-line and entered into the computer with a camera or scanner can be extracted; and,
- many others.

Much research is continuing in the area of pattern matching as the requirements become more and more demanding and the amount of data involved grows rapidly. Even though many issues in the general field of pattern matching are currently being addressed [Cantoni, et al., 1989] [Kyung, 1991] and innovative methodologies to perform general pattern matching are under development [Fisher and Case, 1984] [Tavakoli, 1986], much of the research currently centers around discovering ways to create new applications and enhance existing applications. However, the solutions to problems in the general field of pattern matching and the new methodologies can assist in the development of the new applications as is the case with this research.

1.1.2 Recognition of Human Readable Text

Over the past several years, an extensive amount of research has involved attempting to create a program that performs **computer reading**. The term reading is used in the loosest possible sense. For the purposes of this paper, computer reading refers to the process of examining the binary image representing an input document and determining which words are on the document. The other processes involved when a human performs reading such as the extraction of meaning from the words read are not examined. The task of performing reading, while seeming simple, is much more difficult than it appears.

As research has progressed, this general reading task has been separated into several categories according to constraints that are placed upon the task by the available hardware as well as the constraints that are imposed by the types of human readable text or script to be read by the computer program. Current hardware constraints and industry needs have effectively divided the possible solutions into two distinct categories:

- **On-Line** recognition; and
- **Off-Line** recognition.

The hardware requirements for each of these categories differ greatly.

In on-line recognition, the computer recognizes text or script as it is entered via an electronic pen and pad. The writing surface, commonly known as a **digitizing tablet**, typically has a resolution of 200 points per inch and is sampled at a rate of 100 times per second [Impevodo et. al., 1991]. This provides an environment from which pen velocity, stroke definition, and stroke ordering can be easily determined. This information, along with the actual pixel information, provides a rich alphabet of symbols, known as **ink**, to the on-line hand-written text or script recognizer.

Off-line recognition involves the use of a digital scanner or camera. After the writing or printing of a document is completed (usually on paper), the text or script is digitized by the scanner and this produces an image represented by a two-dimensional array of pixels. This array of pixels represents a white background (paper) with one or more line drawings made in the foreground in black. Each of the line drawings represents a word, letter, punctuation mark, or other symbol that may be present in text and possibly require recognition. Characters, text, script, or other patterns that may be present are recognized by the computer application using only the information provided in the pixels of the black and white image.

1.1.2.1 On-Line Recognition

With on-line recognition, symbols are recognized as they are drawn on a digitizing tablet which is also known as **electronic paper**. When used in conjunction with a pen based operating system or some pen user interface, an on-line recognizer can be extremely useful in many applications. At the present time, small computers called 'Personal Digital Assistants", or **PDA**s, which use a pen based operating system and pen based user interface have now become widely available [Reinhardt, 1994] [Andrews, 1994]. Users keep appointment calendars, order information for salesman, signature verifications, lists of phone numbers, and a wide variety of other information on these PDAs.

Interaction with the on-line pen driven machine involves pointing the pen at an object (possibly being viewed under the digitizing tablet), drawing some picture or character on the tablet, or writing on the tablet. The PDA then recognizes whatever message that the user is trying to convey and performs appropriate actions—data entry, etc. In the opinion of many, a windows type interface driven by a pen is much more convenient on some portable computers and at least as user friendly as one driven by a mouse and keyboard. A fairly recent survey of on-line handwriting recognition is given in [Tappert et. al., 1988]

Recent announcements and accompanying reviews of commercial pen-based notebook computer systems indicate that, although the leading edge of this technology lacks refinement, the potential for revenue is great [Baran, 1992] [Andrews, 1994].

1.1.2.2 Off-Line Recognition

As previously mentioned, off-line recognition involves the use of a digital scanner or camera. After the writing or printing of a document is completed (usually on paper), the text is digitized by a scanner or camera and an image of the document represented by pixels is produced, generally in a bit-mapped black and white or gray-scale format. Characters, text, or script are represented by black drawings in the foreground of a document that has a white background. The characters, text, or script is subsequently recognized by the computer using only the information contained in the image.

The task of reading documents prepared off-line has been separated into several categories of research according to the constraints that are imposed by the types of text or script to be read. These categories are listed below according to the perceived level of difficulty in their solutions ranging from relatively easy to very difficult. These are:

- machine generated (or typewritten) text [Impedovo et. al., 1991];
- hand-printed text and numerals [Cohen et. al. 1991];
- hand-written connected cursive script; [Srihari and Bozinovic, 1987]
- some combination of hand-written cursive script and hand-printed text and numerals [Parisse et. al, 1990],

Computer recognition of each of these has been studied, and at least partial solutions have been proposed for each.

Currently, the most important application for off-line recognition involves recognizing machine generated text. Recognition of this type of printed text involves the software technology known as Optical Character Recognition (OCR). The OCR technologies are well developed and many high quality commercial systems are now in existence.

In a typical OCR system, a two dimensional set of pixels comprises the input. The image is broken down into lines and then each line is broken down into characters. The internal form of each character is then fed into a preprocessor that performs smoothing, noise reduction, and size and orientation normalization. Each character is then classified or recognized using distinctive features extracted from the character's preprocessed internal form.

There are many problems that must be overcome by any high quality OCR software. It must be able to distinguish from a myriad of type faces or fonts. It must be able to handle noise and other categories of deformed images. It must be able to handle different sizes of characters. The list is much longer but these three problems alone should convince the reader that there are serious problems to be solved. The reader can refer to an excellent survey paper on the subject of OCR by Impedovo [Impedovo et al., 1991]. Intensive research in this area is on going [Kyung, 1991].

Other important applications of off-line recognition involve the recognition of hand printed text and handwritten cursive script. Software that performs off-line recognition of hand printed text is a generalization of the software that performs OCR. The same methodologies that solve the OCR problem must be present to solve this problem. However, recognition of hand printed text involves recognizing a possibly infinite number of different ways that the same character may be formed. These differences may be caused by the fact that more than one writer is involved, a single writer may change styles for some reason, the hard surface used to write on might be uneven, or any number of other possibilities.

The original goal of this research was to pursue reading of cursive script produced off-line. Algorithms that read off-line cursive script are a generalization of algorithms that read off-line hand printed text. Even though there has been a significant amount of research toward the solution of this problem, there are still many major problems to solve.

In the sections that follow, a detailed description of the research area is provided. As a convention, in the areas in which this research made significant contributions, the review of the specific literature in that area was postponed until Chapter II. Otherwise, a brief discussion of the relevant literature is included.

1.2 Off-line Recognition of Cursive Script

1.2.1 Difficulties In Performing Off-line Recognition

Extracting words from an image known to contain text involves two processes: recognition and contextual analysis. For images known to contain machine printed text or possibly even highly constrained handwriting or hand printing, the contextual analysis becomes less important. The contextual analysis is used to help make choices when the image processing encounters uncertainty or ambiguity.

1.2.2 Discussion of Application Constraints

In many instances, constraints are placed upon the writers involved with the creation of the image. These constraints can greatly simplify the problem of text recognition because they decrease the amount that different handwritings can vary. These constraints can take various forms including "pre-printed boxes to limit the size and location of characters, guidelines to specify location of words, suggestions for forming letters, suggestions for joining letters and ligatures in cursively written words, rigidly fixed syntax, and no spelling errors" [Cohen et. al., 1990] plus a host of other possibilities. The fewer constraints placed upon the writer, the more dependent the recognizer will be on the contextual analysis.

If text is totally unconstrained, the writing style and writing implements are not restricted.˙ Text appearance can vary according to the individual writing style, size and orientation of the text, writing implements used, and writing surface. A system that can recognize handwriting in the above unconstrained environment should also be prepared to work in an arena where digitization methodology and image thresholding could vary as well.

In the current state of the art, only when considerable constraints are placed on the images containing text does the process become tractable. [Cohen et. al. 1990] describe a project for the United States Postal Service (USPS) which attempts to recognize zip codes from envelopes. There are spatial layout conventions (not necessarily constraints) specified by the USPS to assist in the addressing of envelopes. Using the spatial layout contextual knowledge of how addresses are supposed to look and contextual knowledge about which states have what zip code, the goal was to examine the otherwise unconstrained image representing the envelope and to:

- segment out the zip code (discover which set of line drawings on the digitized image representing the envelope make up the zip code); and
- then classify the zip code (determine what the zip code was).

Even considering the seemingly simple specifications, at the time of the writing, their system had only managed to recognize about 80% of the 508 zip codes in the test data provided by USPS. Although this recognition process did have many obstacles such as widely varying interpretations of the spatial layout conventions and the fact that other sequences of numbers, such as box numbers and street addresses, can be part of an address, the project did not encounter some of the problems that are involved when attempting to recognize connected cursive script. Zip codes usually consist of five or nine disconnected numeric digits. Even though there is no hard and fast rule saying that digits in a zip code cannot accidentally touch each other, recognition of the digits generally does not require that digits are somehow extracted from a connected line drawing containing other characters or digits.

When the objective is recognizing unconstrained and possibly connected or disconnected handwritten text or script, the problem becomes much more difficult. Research in the area of recognition of handwritten cursive script generally concentrates in one or two emphasis areas with a considerable number of constraints placed on the input. Most of the systems do not even consider the spatial layout of the image and just concentrate on the recognition process. Possible constraints might include word connectivity constraints, size and normalization constraints, constraints on the formation of characters, constraints on how characters must be connected, constraints on the size of the recognizable lexicon, and various other constraints.

Figure 1.1 shows four possible word connectivity constraints for images assumed to contain text to be recognized. Figure 1.1a contains machine generated disconnected text. There are many OCR systems that can recognize disconnected machine generated text with a success rate of over 98%.

[Tappert, 1982] identified five separate cases that must be dealt with when concerned with the recognition of hand printed text or hand written cursive text. The categories he specified were: boxed discrete printed characters, spaced discrete printed characters, run-on discretely written/printed characters, pure cursive script writing, and mixed cursive and discrete characters. Figure 1.1b, 1.1c, and 1.1d show three of the cases.

Figure 1.1b contains disconnected hand printed text. The level of difficulty in the off-line recognition of hand printed text compared to machine generated text increases at least one order of magnitude. A large number of researchers are looking into this application. This disconnected constraint greatly simplifies the character extraction

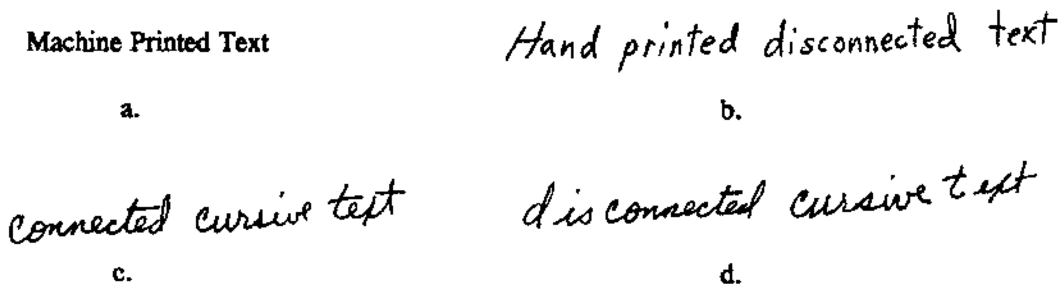


Figure 1.1. Connectivity Constraints.
Four Types of Connectivity Constraints.

requirement as each line drawing is considered a character, but complicates word segmentation as the beginnings and endings of words might not be obvious in the absence of other constraints like "a large distance between characters is required to assist in word segmentation" or "put words or characters in provided boxes".

Requiring that all words are totally connected cursive script completely removes the word segmentation problem but greatly increases the difficulty extracting characters. The line drawing representing a script word must be examined and the recognition of characters or character constructs must extracted from the script word.

Allowing words to contain both connected and disconnected script greatly complicates things. Words must be segmented in light of the fact that each drawing representing text might be one or possibly more characters. This problem is difficult for various reasons that are not obvious. The problems of ambiguity abound. For example, consider the word "amen". If the leading character "a" is disconnected from the rest of the word, then the word could reasonably be segmented as two words: "a" and "men". A further contextual analysis might require a syntax check to determine whether the correctly extracted characters represent the word "amen" or the two words "a" and "men".

Most research into cursive script recognition does not attempt to recognize capital letters since capital letters involve many more strokes and features than lower case letters. Further constraints might specify that all writing requires no normalization and the size of the letters may be restricted.

1.2.3 Contextual Analysis

The fact that human readers sometimes use vast amounts of contextual information in order to determine the correct stream of words is uncontested. Examples of such contextual information include the following:

- idiosyncrasies of a known individual's writing style;
- using phonetic pronunciation for mis-spelled words;
- spelling rules (generally i comes before e except after c);
- knowledge of syntax (if a verb is expected, then the search space of words is reduced);
- knowledge of the current semantics (knowing what a writer is trying to say cuts down the search space); and
- pre-knowledge of a constrained lexicon (if someone is to place the color of his/her eyes into a box, then the lexicon should only contain colors);

The use of contextual heuristics to assist in reading unconstrained text is an enormous amount of help to human readers, yet the process is not easily formalized and for the most part is not well understood. Researchers, including the author of this paper, have not pursued totally unconstrained text as of yet. [Cohen et. al., 1990], as described earlier, have pursued unconstrained text recognition in an extremely restricted domain.

In general, some types of context are widely used in the research. Use of a lexicon that must contain all the words that may be recognized is contextual information which is used in virtually all research dealing with text and cursive script recognition. Also, as real world applications employ some type of off-line character recognition, then heavy use of contextual knowledge of some type will generally be required.

1.2.4 General Methodology Used in Script Recognition

A block diagram representing the components of a system that reads off-line cursive script is given in Figure 1.2. The different phases of the system involve inputting a document via an optical scanner and the concomitant creation of an image file. Next, software applies a thresholding algorithm that maps the gray-scale or color image into black and white. Noise removal, smoothing, and other preprocessing is then applied to the image. Then the various lines drawings on the page are located and segmented into possible words, punctuation, and other items.

Once the segmentation and preprocessing are complete, features are extracted from each line drawing. The features discovered in the line drawing are used by the character construct extraction mechanism to discover which characters or character

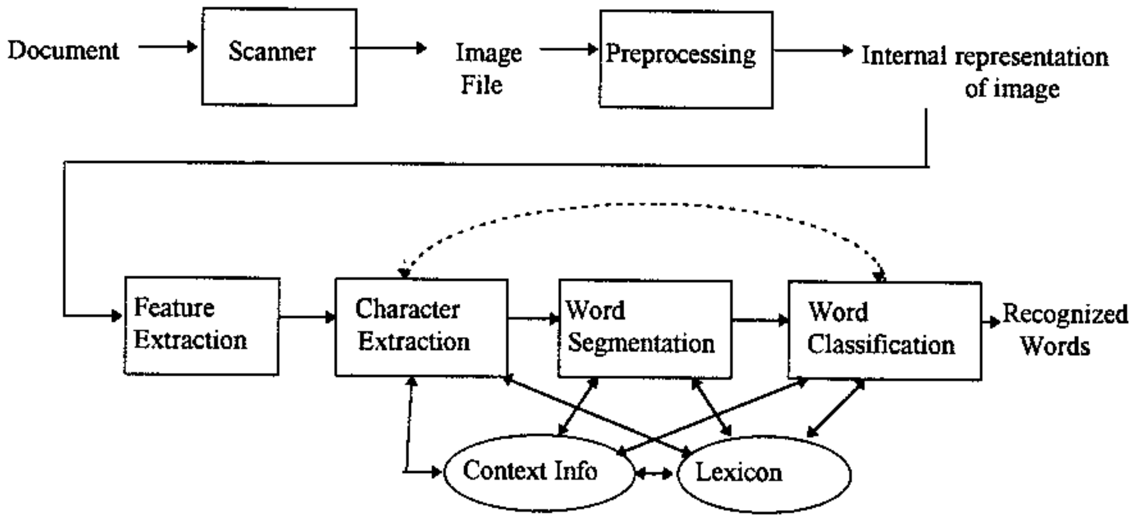


Figure 1.2. The Cursive Script Recognition Process
A Block Diagram of the Cursive Script Recognition Process

constructs make up the line drawing by comparing the statistics of the features obtained with the sets of sample features that were trained into the system at an earlier time.

At some point in this process, line drawings must be grouped in a manner so the characters contained in the drawings selected, represent a word. This is called the word segmentation process. Next, linguistic, contextual, or statistical information can be used to resolve ambiguities in the similarly shaped characters predicted. Finally, when the characters have been grouped either positively or tentatively into words, the characters in the word groupings must be classified into words.

1.2.5 Preprocessing the Input Image

Preprocessing is an important phase of any pattern recognition process. Similar types of preprocessing are performed on any input image for all applications of off-line text or script recognition including OCR applications, hand printed text applications, and hand written script applications. The main preprocessing techniques that are used include: thresholding, smoothing, thinning, normalization, and line segment approximation of the line drawings in an input image, along with segmentation of the document.

There are many different segmentation processes that are involved in the reading of text and script. An initial segmentation is performed to separate the drawings on the original document into text, script, graphics, etc. This is done so that subsequent processing which performs reading only need look at the text and script parts. Several researchers have addressed this type of segmentation including [Rosenfield and Thurston, 1971], [O'Gorman and Clowes, 1976], [Sun and Wee, 1982], [Haralick, 1978] and

[Horowitz and Pavlidis, 1974]. Of more direct interest to this paper is the work by [Fletcher and Kasturi, 1988] which is directly concerned with the separation of text from graphics in mixed text/graphics environments.

There are many other segmentation phases in a text or script reading system. With respect to disconnected text, segmentation is the isolation of characters. With respect to connected cursive script, segmentation is the isolation of words. With respect to partially connected cursive script, segmentation is still the isolation of words but it has become a much greater problem [Netvia, 1986].

Thresholding [Chanda et. al., 1986] [Bernsen, 1986] [Kahan et. al., 1987] is the action of modifying a gray scale image into a binary image. A thresholding function is determined according to the current gray scale values of the pixels in the image. This function is then applied to each pixel in the image.

Smoothing [Seun, 1982] removes noise and corrects minor flaws that are apparent in an input image. These algorithms are mainly responsible for the filling action which eliminates breaks, gaps, and holes in a line.

Thinning is the process of reducing a line in an image from several pixels wide down to a single pixel with the reduced image called a skeleton. Figure 1.3 shows the result of applying a thinning algorithm to an input character. Various thinning algorithms have been developed including sequential, parallel, and hybrid algorithms. The most common variety of thinning algorithms are based on iterative edge erosion techniques where a window is moved over the image and rules are applied to the contents of the window which may allow the deletion of some of the pixels in the window.

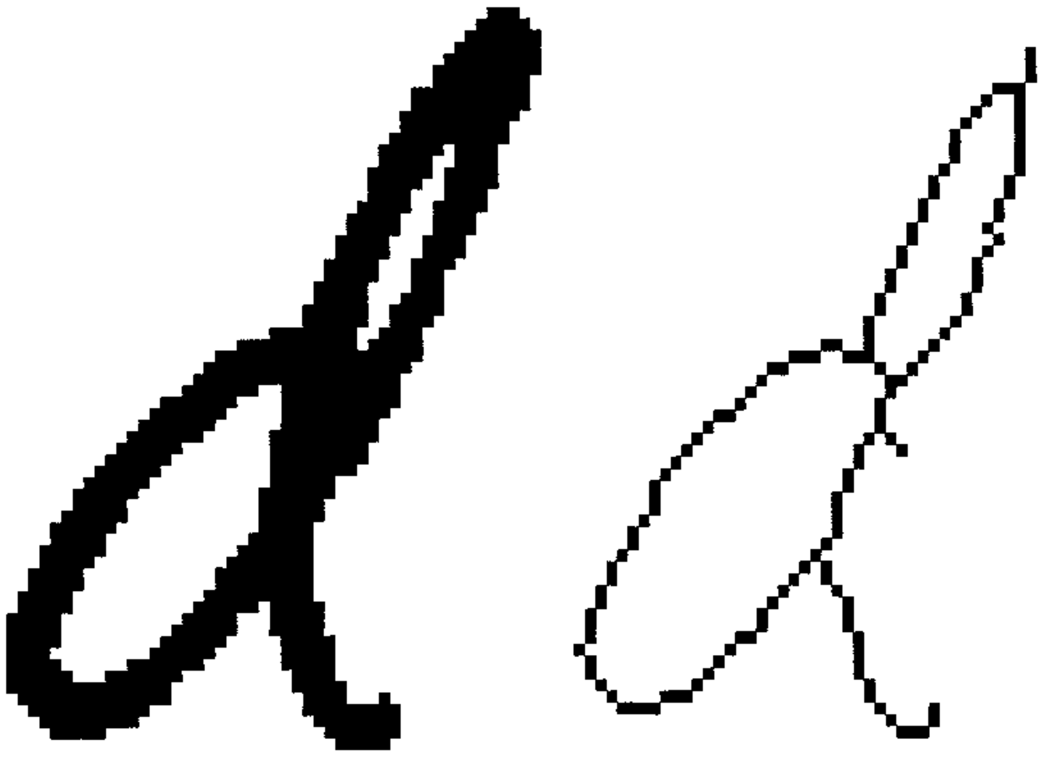


Figure 1.3. Example of Image After Thinning
An Original Image Representing a Lower Case d with its Skeleton (greatly enlarged)

It was discovered in the analysis phase for this research that most of the thinning algorithms in the literature produce skeletons that do not lend themselves well to thinning line drawings which encompass possibly much more than a single character. Even though there are many thinning algorithms in the literature, none of the simple ones produce a skeleton that is clean enough to be of much use. For this reason, a new thinning algorithm was created that produces smooth skeletons which retain virtually all the original line drawing's shape, connectivity, and end-points. A review of the literature dealing with thinning algorithms is provided in Chapter II.

Normalization algorithms perform corrections on the line drawing in an attempt to cause all characters, words, or other textual images to conform to some norm for the input [Nagy and Tuong, 1970] [Srihari and Bozinovic, 1987]. This includes algorithms which correct the slant of individual characters or words and algorithms which adjust the character sizes.

Vectorization, or line segment approximation, is the conversion of the pixels in a skeleton into a set of coordinates, which, when joined with straight line segments, form an approximation of the original skeleton. In this research, the result of the approximation is known as a singularity graph. This approximation is performed mainly to reduce the volume of data that must be processed. Figure 1.4 shows a thinned image and along with the drawing of the singularity graph which approximates the skeleton. A review of the literature dealing with vectorization algorithms is provided in Chapter II.

A new line segment approximation algorithm was also created for this system. It involves the use of the pattern recognition technology called Finite Induction (FI) [Fisher and Case, 1984]. FI is exploited to recognize patterns of pixels that should represent vectors, or line segments, and when it discovers a candidate, it marks the ends of the pattern within the thinned skeleton as critical points or singularities. A follow up pass performs a recursive descent traversal through the skeleton recording singularity and connectivity information. The singularity graphs of the skeletons produced in this fashion very closely approximate the actual shape of the skeleton. It is these singularity graphs

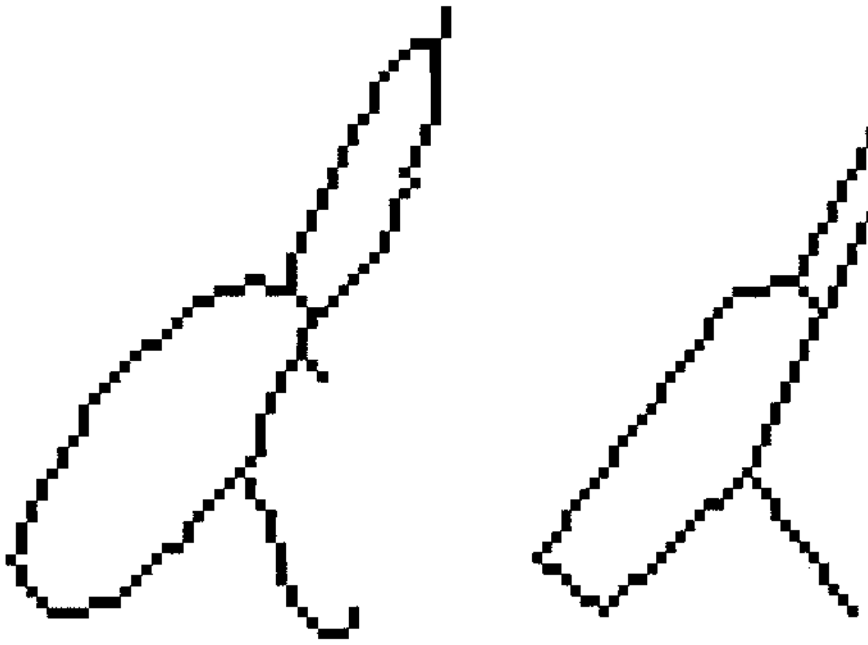


Figure 1.4. Example of Drawn Singularity Graph
A Skeleton for the Lower Case d and a Drawn Singularity Graph (greatly enlarged).

that are used as input to the feature extraction phase. The singularity graph drawn in Figure 1.4 was produced by the vectorization algorithm developed for use in this research.

1.2.6 Feature Extraction

Feature extraction algorithms involve examining the line drawings representing characters, words, or partial words and noticing the presence and location of features. Many different feature types have been proposed and many different algorithms have been examined that perform feature extraction. Some of the main feature extraction techniques are discussed in detail in Chapter II. A completely new feature extraction technique is proposed for this research. While most of the feature extracting algorithms were designed with either the disconnected printed text or connected cursive script in mind, the new feature extraction methodology can be easily modified for use in either domain.

1.2.7 Character Extraction

Character extraction involves recognizing multiple characters within a single line drawing. This is the only required algorithm for this project that is not also required in an OCR system or a hand printed text reading system. This is true because in the other applications, each drawing is assumed to represent only one letter, while in cursive script, each drawing may contain one or many letters.

In this system, after the preprocessing is complete, a singularity graph of the skeleton of an input drawing is available. This singularity graph is processed one singularity at a time, checking to see if a character begins at that singularity. In this phase, FI [Fisher and Case, 1984] is used to perform the extraction. The FI following algorithm is modified so that the algorithm itself predicts the required input during a recursive descent traversal over a localized area of the singularity graph starting at the selected singularity. The prediction is made in cases where multiple paths are available and the input predictor knows which path is expected by the FI ruling and chooses that path if it is available.

1.2.8 Character Extraction, Word Segmentation, and Word Classification

The initial character extraction pass should discover the presence of some characters. This set of characters consists of the highest confidence characters. They are maintained in a list of characters in what is referred to as a letter graph in this research. Note that in many cases, characters may partially overlap in the letter graph and in some cases, characters may overlap totally in the graph. As an example, consider a hurriedly written lower case 'i' where what would be the short upward stroke is really a short upward narrow loop. The 'i' will be extracted and so will an 'e'. The ambiguity will have to be solved later when more information is known. Many other similar cases exist.

When considering disconnected script, the word template is managed by the word segmentation sub-system. It selects which drawings go together to make a word. If the constraints are lenient, this job involves dealing with lots of ambiguity and is quite a difficult process. Most research on cursive script recognition requires that words be connected and ignore this problem.

1.3 In Pursuit of a Practical Cursive Script Recognizer

The major goal of this research was to analyze the difficulties of performing cursive script recognition and to solve enough of the problems involved to demonstrate that cursive script recognition is feasible, at least in certain constrained domains.

The word classification phase involves searching the word template and comparing the extracted characters against words in a lexicon. As extracted characters are placed in the word template, the search space for subsequent queries to the lexicon can be reduced. The final result of this phase is the list of classified words and rejected words. This is the end to which the entire reading process is pointed.

Another important primary goal was the experimentation with a pattern recognition technology known as Finite Induction to perform much of the actual recognition of the patterns involved. This involves mapping the information in the image into an appropriate input alphabet for an FI recognizer.

In the analysis phase it was determined that there were indeed major problems to solve. To prove that it is feasible, a working cursive script recognizer was constructed, including all of the phases mentioned in Figure 1.2. In experimental usage of the recognizer, a reasonable correct classification rate was observed. After each experiment phase, the classification rate improved because the system adapted to individual writers.

The specifications for the recognizer follow:

- Constraints placed upon the input image:
    a. cursive script to be recognized must all be roughly the same size script;
    b. the script must be written in a straight line across the page;
    c. cursive script words must be connected with the exception that capital letters can be disconnected to the left of the word to which it belongs and dots may be disconnected over letters;
    d. punctuation must be carefully written;

- Constraints placed upon the spatial layout of the script are that script must consist of lines written in a straight line across the page and well positioned on the page so that one line does not interfere with another;

- All phases of the cursive script recognition process should be examined in light of the diagram of Figure 1.2 so that the system should input an image file representing a document to be examined (following the above mentioned constraints) and a text document containing the classified text and punctuation should be produced;

- New methods of performing the steps of the cursive script reading process should be created when the previous state of the art is found lacking;

- The feature recognition system should be flexible and allow easy encoding and entry of new features for experimental purposes;

- The recognizer is to classify words from a limited lexicon; however, the lexicon management should be flexible as words might need to be added or deleted as requirements change;

- The system should be easily modifiable to operate in the realm of parallel computing with the new low cost SMP (Symmetric Multiprocessor) machines soon to become available;

- The recognizer should allow for an adaptation phase so that the recognizer might learn the idiosyncrasies of the personal writing style of a writer; and

- A reasonable level of noise and fuzziness is allowed in the original document images, but all images should be scanned on a similar scanner so all images suffer from the same deficiencies and enjoy the same advantages. Also the scanning resolution is 300 dots per inch.

The actual design involves all aspects of the cursive script reading process described in the previous sections. Major contributions were not made in every area but were made in many areas. Smoothing was not employed as the scanner used for the purposes of the experiment produced very clean line drawings; and the thresholding algorithms have already been well studied in the literature.

The system initially examines the mostly connected script across the page, segmenting each script item into a separate line drawing. Images are all assumed to contain line drawings representing cursive script as no graphics or other non-script information is allowed on the page. Segmenting mixed text and graphics was studied by [Fletcher and Kasturi, 1988].

There were contributions made in the area of preprocessing. This includes a new algorithm to perform thinning and a new FI based method to perform vectorization of an image.

A new method to perform feature extraction is another contribution. Other contributions include a new FI based method to perform character extraction and a methodology of classifying words with split n-gram tree indices into the lexicon.

[Kondo, 1990] has shown that even when writers write very carefully, there is no way to successfully contrast many characters from one writer to the next. This means that even if writers take extreme care, one writer's lower case 'f' could look exactly like another writer's lower case 'b". Various other similar situations exist. It is his contention that to be successful, an off-line cursive script recognizer must adapt itself to the idiosyncrasies of each individual writer. This approach was taken in this research even though there may be methods to use context to assist in the resolution of the ambiguity. Even though this context utilization is generally used by human readers, these approaches are not very well defined. The application of only a minimal amount of context was studied in this research.

Also, it was part of the original design specifications that the system be written with flexibility in mind with respect to the feature recognizer. Little research in this area was applicable to this project with respect to the choice of a set of features to extract in the feature extraction phase. It was decided to select a few simple features to start with, but leave room to easily expand the feature set, if the original set is found lacking.

An important goal of the project was to explore the capabilities of the theory of Finite Inductive Sequences [Fisher and Case, 1984] as a pattern matching mechanism in the arena of off-line processing of written documents. Finite Induction (FI) was used as the recognition mechanism in two of the sub-systems, the vectorization sub-system and the character extraction sub-system. A short description of the FI pattern recognition technology is provided in the Appendix and an expanded discussion can be found in [Fisher and Case, 1984] and [Tavakoli, 1986]

1.4 Organization of the Remainder of the Document

This dissertation contains seven more chapters:

- Chapter II discusses previous work in a literature survey. In this chapter a detailed review of the literature is provided for each area in which a significant contribution was made by this research.

- Chapter III discusses the Border Reduction Thinning Algorithm developed during this research. The algorithm itself is discussed with advantages and disadvantages analyzed.

- Chapter IV discusses the FI based vectorization algorithm developed during this research.

- Chapter V discusses the feature extraction algorithm developed during this research. Also discussed is the mechanism used to store the data base of recognizable features.

- Chapter VI discusses the FI based character extraction mechanism, word and punctuation segmentation, and Also discussed is the data structure used to store the lexicon for efficient searching.

- Chapter VII discusses the design and implementation of an experiment that demonstrates the effectiveness of the overall system. Performance measures are shown. The mechanism that provides adaptation of the system to the individual writer is also discussed.

- Chapter VIII is the conclusion. It offers concluding discussions, reviews the list of contributions made by this research, and mentions future directions that the research will take.

CHAPTER II

REVIEW OF THE LITERATURE

2.1 Overview

Most of the initial research on the topic of automatic cursive script recognition was in the area of on-line approaches and was closely tied into the study of how humans perform the task [Frishkopf and Harmon, 1961] [Eden, 1961] [Mermelstein and Eden, 1964]. Since then, the study of handwriting has moved into several directions, including:

- automatic signature verification [Plamondon and Lorette, 1989] [Liu, Herbst, and Anthony, 1979] [Sato and Kogure, 1982],
- modeling the motor control activities involved in handwriting [Dooijes, 1983] [Denier Van Der Gon and Thuring, 1965] [Plamondon and Lamarche, 1986],
- handwriting simulation [Denier Van Der Gon, Thuring and Strackee, 1962] [Vredenbregt and Koster, 1971],
- mathematical modeling of handwriting and the handwriting process [Morasso and Mussa Ivaldi, 1982] [Plamondon, 1989] [Schomaker, Thomassen and Teulings, 1989],
- cognitive modeling of handwriting [Teulings, Thomassen, and Van Galen, 1986] [Van Galen and Teulings, 1983] [Van Galen, Smyth, Meulenbroek, and Hylkema, 1989] [Van Galen, Meulenbroek, and Hylkema, 1986],
- parsing and recognition of multi-dimensional languages (for text recognition, parsing and recognition of 2 dimensional languages) [Rosenfield, 1979] [Inoue and Takanami, 1991] [Inoue and Takanami, 1994] [Aizawa and Nakamura, 1994]

- modeling and recognition of Chinese characters [Casey and Nagy, 1966] [Chen, Hsu, and Cheng, 1986] [Cheng and Hsu, 1991] [Huang and Huang, 1991], and
- applications of neural networks to script recognition [Le Cun et. al., 1989] [Guyon et. al, 1989] [Guyon et. al., 1991].

Research in the area of the direct reading of script has continued as the industrial need for high quality on-line and off-line readers has grown. [Earnest, 1962] presented an interesting off-line mechanism for the reading of handwritten cursive script involving global word characterizations, such as counting ascenders and descenders, to bypass the character segmentation phase. There were other similar important works for on-line recognition including [Farag, 1979] and [Bridle, Brown and Chamberlain, 1983]. The research of [Simon and Baret, 1991] considers an off-line mechanism which recognizes words without character segmentation. This research considers the pertinent information in a written word to consist of "singularities" which include forks, crossings, changes of directions, and extremities. If these singularities are removed from the line drawing, only an "oscillating wiggle", which can be discarded, remains. The word is recognized only from the restored arrangement of singularities within the word.

During the 1970's, some research was initiated in pursuit of off-line cursive word recognition. [Ehrich and Koehler, 1975] and [Sayer, 1973] studied aspects of character segmentation and the use of context to assist in resolving ambiguities.

Beginning in the early 1980's, interest began to pick up once again in on-line methodologies. [Burr, 1982], [Tappert, 1982], and many others studied the use of dynamic programming methodologies, also called elastic matching and/or time warping, as the recognition mechanism. Using this algorithm, a distance is calculated between a current template representing a character and each template in a character lexicon. The lexicon entry with the shortest distance is declared the correct classification. The distance between two templates is calculated by modifying one template in a series of simple transformations until it becomes the same as the other template. The distance is a function of the number and types of transformations made.

An on-line system, which allows users to train the system, was given in [Berthod and Ahyan, 1980]. The system used knowledge of letter formation from strokes using a syntactic-style approach. A brief sketch of an off-line system to recognize Roman cursive script was given in [Badie and Shimura, 1982]. An on-line system which ignored contour segmentation and character level feature extraction was given in [Brown, 1981]. Other early works on character recognition include [Frishkopf and Harmon, 1961], [Dutta, 1974], and [Stillman, 1974].

The interest in recognition of on-line handwritten print and cursive script is now intensifying, brought on by the availability of high quality digitizing tablets, industry requirements, and large markets for pen driven PDA's and other similar devices. More information about on-line recognition methodologies is available in the survey of recent research provided by [Tappert, Suen, and Wakahara, 1990]. [Higgins and Duckworth, 1990] describes an early electronic paper project providing an overview of the hardware and software mechanisms involved in on-line recognition, [Kadirkamanathan and Rayner, 1990] focuses on a methodology to segment on-line cursive script into strokes, and [Oulhadj et. al., 1990] describes a practical implementation of an on-line recognition system.

2.2 Off-Line Script Recognition

There are several recent efforts in the area of off-line cursive script recognition. A practical solution or even partial solution to the problem of computer reading of unconstrained text and script would be of great value to several industries. For example, a very important benefit would be the ability to automate aspects of the postal services around the world.

One of the more influential projects is described in [Bozinovic and Srihari, 1985], [Srihari and Bozinovic, 1987], and [Bozinovic and Srihari, 1989]. This project is characterized by the authors' view that script recognition is a "perception problem in which there exists a natural hierarchy of representation levels, each level identifiable with a conceptual entity: points, contours, shapes, letters, words, sentences, paragraphs, etc." [Srihari and Bozinovic, 1987].

Constraints are placed upon the input which require that words are carefully written in that each word is assumed to be a single well-framed binary-valued word image that is written in such a way that an upper, middle, and lower zone can be identified during the processing. After slant normalization on the input, character segmentation points are predicted by passing a vertical line over the input. Areas where there is only a single intersection point between the vertical line and the input are possible segmentation points. After character segmentation, words were predicted using a stack-decoding search algorithm with a trie-structured dictionary with a small lexicon of 1027 words. [Bozinovic and Srihari, 1989] added a depth of search heuristic to limit the computation.

The results of this project were quite good in that, with the small lexicon, a 70%+ accuracy level was obtained for correct selection in the top two choices.

Another project is detailed in [Aoki and Yamaya, 1986] and [Aoki and Yoshino, 1989]. This project views script recognition as a syntactic pattern recognition problem [Shaw, 1972] [Fu, 1976] [Bunke, 1992]. In this project, special grammars for each character to be recognized are hand encoded using a list of features as terminal symbols. After thinning, the original image is translated into a tree-type chain code representation of the image [Fu, 1976]. Then the chain code internal representation is compressed to remove some of the complexity. Using special heuristics, the tree representation of the line drawing is "easily" segmented into characters. A bottom up parse is performed and the results of this parse, along with some simple post-processing and a lexicon lookup, are used to perform the word classification.

The results of the project are hard to analyze. The experiment involved choosing 130 words from the lexicon and 3 experiment participants. Each participant wrote all 130 words and these words were supplied to the recognizer. A very good correct recognition ratio of 85.4% was obtained with an average recognition time of 180 seconds per word.

[Cohen, Hull, and Srihari, 1991] is a description of an approach for reading a block of hand-written text when there are only certain loose constraints placed upon the spatial layout and syntax of the text. A system which reads handwritten postal addresses, in particular the zip code, is described as an implemented instance of this approach. As described in Chapter 1 of this paper, this research emphasizes the use of the context of spatial layout and limited domain of words to assist in word recognition.

Another project which attempts to recognize the semi-unconstrained text of postal addresses is given in [Downton, Tregidgo and Kabir, 1991]. They present an algorithmic architecture for a 'high-performance optical character recognition (OCR) system for hand printed and hand written addresses". The architecture integrates syntactic and contextual post-processing with character recognition to optimize British postcode recognition performance. The strategy used involves extracting the postcode and then verifying its correctness by using information drawn from the rest of the address.

The rest of this chapter provides a review of the literature for specific areas where a significant contribution was made by this research.

2.3 Literature Regarding Present Contributions

The areas in which this research made a significant contribution include thinning, vectorization, feature extraction, character and word segmentation, and word recognition. This section is divided into five sub-sections in which the literature within each of the five areas is reviewed.

2.3.1 Thinning and Skeletonization

Thinning and skeletonization is an area of research in pattern recognition that has received an enormous amount of interest during the past several years. There have been literally hundreds of papers discussing thinning and aspects of thinning. Some of the important early efforts are discussed here along with some of the thinning methodologies that represent the state-of-the-art. Some good surveys of thinning methodologies can be found in [Smith, 1987], [Lam, Lee, and Suen, 1992], and in [Chen, 1993].

Some of the famous thinning algorithms in the literature are [Pavlidis, 1982], [Naccache and Shinghal, 1984], [Deutsch, 1981], [Arcelli, 1981], [Arcelli and Sanniti di Baja, 1985], [Zhang and Suen, 1984], [Hilditch, 1969] [Xu and Wang, 1987], and [Rosenfeld and Davis, 1984]. The preceding list represents only a few of the more famous algorithms.

There is also much research into the comparison of thinning algorithms [Lee, Lam, and Suen, 1994] [Zhang and Wang, 1994] [Plamondon, et. al., 1994]. Two of the better known and higher ranking algorithms, according to the above comparisons, were [Wang and Zhang, 1989] and [Kwok, 1988].

The Border Reduction Thinning algorithm, which is introduced in this research, is a very simple algorithm which produces a very high quality skeleton. It has not been compared and ranked with the more famous algorithms; however, it is anticipated that it will compete well in at least the quality of skeleton area.

2.3.2 Vectorization or Line Segment Approximation

Vectorization or line segment approximation involves the conversion of a list of pixel coordinates or chain-coded pixels into a vectorization graph. A vectorization graph is a small set of vertices and edges where the vertices represent critical points on the skeleton and the edges represent connection information so that when the critical points are plotted and the edges are drawn on the plot, the picture represented by the drawn vectorization graph is a close approximation of the picture represented by the original skeleton [Hung and Kasvand, 1983] [Jimenez and Navalon, 1982] [Ramer, 1972] [Slansky and Gonzalez , 1981].

This approximation is then used as input to the feature recognition engine which must extract features from the geometry or topology of the drawing [Nishida and Mori, 1992]. [Pavlidis, 1984] and [Pavlidis, 1986] proposed a line vectorization method in which a skeleton is constructed from a Line-Adjacency Graph (LAG) where only horizontal runs are used in the LAG.

In some instances, the vectorization process of a line drawing is only concerned with tracing a skeleton from an end/junction point to another end/junction point [Lam and Suen, 1988] [Ramer, 1972]. Each curve can be approximated by a polygon as in [Ramer, 1972]. In this fashion, mathematical methods for polygon matching can be used for feature extraction.

The methodology proposed by this research uses FI to determine the singularities (critical points) in a skeleton and a connection analysis to prepare a vectorization graph to make it convenient to use topological and structural information during the feature extraction process.

2.3.3 Feature Extraction and Character Segmentation

'It is generally accepted that feature extraction is one of the most difficult and important problems of pattern recognition" [Impedovo et. al., 1991]. In the area of off-line text and script recognition, most of the research has been directed toward recognizing features in the OCR arena. Many of these techniques are presented in this subsection.

2.3.3.1 Extracting Features From Template Matching and Correlations

Using this technique, an input character matrix, such as the one in Figure 2.1, is matched against a set of templates and the distance between the pattern and each template is calculated [Shimura, 1973] [Tubbs, 1989]. The pattern is then classified as the character represented by the template more closely matching the input pattern.

```
0000000000
0000000000
0011111000
0111111100
0110001110
0000000110
0000000110
0000001100
0000011000
0000110000
0011100000
0111111110
0111111110
0000000000
```

Figure 2.1. Simple Template Matching

This mechanism is easy to implement for OCR applications and it is used in many commercial products. The templates used varies from product to product from very simple templates and matching criteria to sophisticated templates with built-in truth tables or logical rules.

2.3.3.2 Extracting Features Using Statistical Distribution of Points

There are several techniques that fall into this category. The most widely used techniques involve moments and crossings [Tucker and Evans, 1974] [Cash and Hatamian, 1987].

In the *moments* technique, the features used are the moments of black pixels about a chosen center. The more natural moments include raw, central, and normalized moments. For a binary image, the raw moments are a function of the coordinates of each pixel in the image. They are calculated:

$$m_{pq} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} x^p y^q f(x,y)$$

where p, q = 0, 1, 2, …, ∞, M and N are the horizontal and vertical dimensions of the image and $f(x,y)$ is the pixel value at the point (x,y) in the image. The central moments depend upon the distances of points from the center of gravity of the character and are given by:

$$m'_{pq} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} (x-\bar{x})^p (y-\bar{y})^q f(x,y)$$

where $(\bar{x},\bar{y})$ are the coordinates of the center of gravity. The normalized central moments can be calculated by dividing the central moments by the moment of order zero. Raw moments generally produce poor recognition rates. Central moments, however, are much better and have an added bonus of being invariant to the translation of the image. Normalized moments are also invariant to the scale of the image.

Another popular technique in this category is the *crossing* technique [Calvert, 1970] [Holt, 1974] [Kwon and Lai, 1976]. In this technique, features are represented as count templates. Once an input character is framed, line segments are drawn in regular intervals over the character along certain directions. The directions generally being 0°, 45°, 90°, and 135° with respect to the horizontal axis. Features are represented by the number of times the black portion of the image is crossed by the line segments.

The crossing technique is widely used because it can be performed at high speeds as the algorithm is of low complexity. It is also tolerant to distortions and small stylistic variations as the crossing counts also encode topological and structural information.

There are several other proposed techniques that derive features from the statistical distribution of points in an image [Suen, Berthod, and Mori, 1980]. One such technique is known as *zoning*. In this technique, the frame containing the input character is divided into zones or regions and represents features as a template containing the densities of black pixels in each region. The features used by the *characteristic-loci* method are counts of the number of times that vertical or horizontal line segments intersect the black part of the line drawing for every white pixel in the background of the input character [Knoll, 1969] [Spanjersberg, 1974] [Downton, Tregidgo and Kabir, 1991].

2.3.3.3 Extracting Features Using Transforms and Series Expansion

These methodologies involve using Fourier descriptors [Person and Fu, 1977] and Fourier boundary descriptors S and T as introduced by [Granlund, 1972] and continued in [Krzyzak, Leung, and Suen, 1989]. These descriptors are interesting because of their invariance to scaling, rotation, translation of the character, and to shifts in the starting point [Impedovo, et. al., 1991]. The negative side of using these descriptors is that they have a smoothing effect and are insensitive to spurs along the boundary.

2.3.3.4 Extracting Features Via a Structural Analysis

There are many examples in the literature concerning extracting structural features, including [Iwata, Yoshido, and Tokunaga, 1978] [Parks et. al., 1974] [Baptista and Kulkarni, 1988] [Sue and Chen, 1976] [Aoki and Yoshino, 1989] [Cohen, Hull, and Srihari, 1991].

Structural features generally describe structural, topological, or geometrical properties of the input image. Such features include loops, strokes, curves, bays, end-points, line segment intersections, cups, hats, and commas. Topological information is used many times to assist in structural feature information. [Impedovo, et. al., 1991] provides an example similar to the example given in Figure 2.2. The mechanism can discern between lines and curves. The thinned image is divided into nine regions, A through I. The character is considered as a set of strokes where a stroke is a line (L) or a curve (C) which joins two vertices in the character. It is a curve if the following expression is true:

$$\left| \frac{\sum_{i=1}^{n} ax_i + by_i + c/\sqrt{a^2 + b^2}}{n} \right| > 0.69$$

otherwise it is a line. The equation of the line passing through the extremities of the stroke is $ax + by + c = 0$, (x_i, y_i) are the points of the stroke, and 0.69 was decided by experimentation. The features of Figure 2.2 are GLI and GCA which means that there is a line from G to I and a curve from G to A.

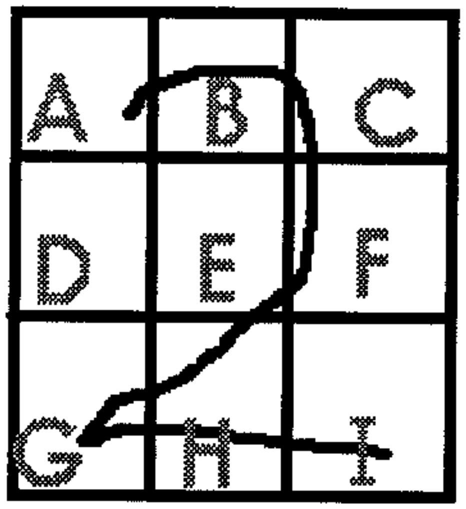


Figure 2.2. Example of Structural Feature Extraction

According to [Impedovo et. al, 1991], the main advantages of using structural features is their high tolerance to distortion and style variations, and moderate tolerance to rotation and translation. He goes on to say that features of this type are very difficult to extract and it is still a topic of research.

The work performed during this project continues the research into the extraction of structural features. The methodology used is quite successful. It is believed that the extraction of structural features is much more usable in the arena of connected cursive script recognition than the other methodologies. The other methodologies were designed to quickly recognize features in a low resolution image representing a single already segmented character. Structural features and topological information can be used to express information of a larger scale.

Note that, since a connected cursive script word might contain large numbers of features, an expressive mechanism to represent the features of a word is required to enable the character recognition engine to conveniently examine the word to perform character extraction and recognition. In this research, the vehicle chosen is known as the *feature graph*.

2.3.4 Character and Word Segmentation

Character segmentation is the process of determining what regions of a connected script word or partial word make up separate characters. This phase has not received a lot of research interest as most of the research was directed at OCR. In OCR applications, this phase does not exist since each line drawing represents exactly one character. However, this is a very important phase for any type of text recognition where two characters may touch and become one connected line drawing. [Tappert, 1982] discusses a simple taxonomy which describes the various levels of difficulty of performing character segmentation while attempting to read handwritten text and/or script. Level 1 is boxed discrete characters, level 2 is spaced discrete characters, level 3 is run-on discretely written characters, level 4 is pure cursive script, and level 5 is mixed cursive and discrete text. The last level is not necessarily totally unconstrained as requirements on word spacing and alignment may be present.

Much of the research about character segmentation involves trying to avoid it. As discussed above, each of [Earnest, 1962], [Farag, 1979], [Bridle, Brown, and Chamberlain, 1983] attempt to avoid character segmentation. [Simon and Baret, 1991] do something similar as an attempt is made to classify words by recognizing irregularities instead of characters.

[Srihari and Bozinovic, 1987] present an interesting mechanism to perform character segmentation. After normalizing a line drawing, which corrects any non-vertical slants in ascenders or descenders, a vertical line is passed over the image and any place in the line drawing in which the vertical line and the line drawing only have one intersecting point is assumed to be a probable segmentation point.

[Aoki and Yoshino, 1989] use special heuristics to assist in segmenting characters. Using these heuristics, not presented in the paper, the tree chain code representation of the line drawing is "easily" segmented into characters.

Many of the early text or script readers, such as those of [Srihari and Bozinovic, 1987] and [Aoki and Yoshino, 1989], assumed that it is possible to unambiguously segment a line drawing into characters. The other alternative is to assume that this is not possible and allow ambiguous segmentation [Bozinovic and Srihari, 1989], [Hayes, 1980], [Higgins and Whitrow, 1985], [Peleg, 1979], [Ford and Higgins, 1990].

In an ambiguous character segmentation system, a list of possible candidates is produced for each segment position, usually together with a value representing a "certainty weight". For example, if the connected cursive script word "man" was presented to the system, two or three options are possible for the second character. The second character may be an 'a' with a high certainty, 'o' with a lower certainty, and possibly even a 'u' with a still lower certainty.

Word segmentation is the process of determining which line drawings on the input image represent characters, partial words, or words. Under some constraints, such as "there should be extra space between words" and "cursive script words should be totally connected", word segmentation is simplified. However, with totally unconstrained text as mentioned in Tappert's taxonomy, this becomes a much more difficult problem abounding in ambiguity. [Cohen et. al., 1991] points out that image layout is important for the word segmentation phase of the project. They demonstrate that it is not always easy to divide the input image into horizontal lines each containing script. If the text is unconstrained, then it may appear that some text should be segmented with the line above or below as writers preparing postal addresses do not always write in a straight line.

This research has constrained away much of the word segmentation problem by requiring words be completely connected script with the exception of the case of a word beginning with a disconnected capital letter. Even this small exception adds many difficult problems with ambiguity.

2.3.5 Word Recognition

The process of word recognition involves examining the output of previous phases that have preprocessed a line drawing and discovering exactly which word is represented. Post-processing is applied to the predicted characters (or whatever the output from previous phases) to make the choice and to insure the choice is a legal word in the system lexicon. One possible choice that must be available is that of a rejection, meaning that the characters do not represent a word known to the system.

2.3.5.1 Word Recognition with Unambiguous Segmentation

Word recognition is usually performed by using contextual information during a post-processing phase. If it can be assumed that characters can be unambiguously segmented, various techniques can be applied. Some of the more widely used techniques are discussed below.

2.3.5.1.1 N-Gram Techniques

This mechanism calculates the probability of all n-letter sequences occurring in text. These probabilities can then be used to predict the most likely candidate from the lexicon [Riseman and Ehrich, 1971].

2.3.5.1.2 Viterbi Algorithm

The algorithm described in [Viterbi, 1967] takes the predicted word and, using statistical information, calculates the most likely input word. The statistical information used includes statistical information of the sequence of letters in English, and likely errors from the recognition system. [Forney, 1973] provides a clear discussion of the theory involved, and [Neuhoff, 1975], [Riseman and Hanson, 1974] and [Hull and Srihari, 1982] have discussed applying the algorithm to text recognition.

This algorithm uses a "confusion" matrix of a priori probabilities that is observed from the activities of the recognition system, together with the transition probabilities between characters. The confusion matrix represents the probability that one letter may be mis-recognized as another. This value is stored in the node part of a graph data structure called a trellis along with the probability that the letter can be preceded or followed by any other character as edge labels on the graph.

A 26 x *l* trellis graph is constructed, where *l* is the length of the word, linking every letter with every other letter. By tracing a path through this trellis while combining the probabilities on the nodes and edges of the path, the probability that the traced word might have been the input word is calculated. This word is also the most likely path through the trellis graph and therefore is the most likely prediction of the input word.

2.3.5.1.3 Modified Viterbi Algorithm

The Viterbi algorithm always produces the most likely prediction for the input word. However, there is no guarantee that the word is in the system lexicon. [Srihari, Hull, and Choudary, 1983] and [Shinghal and Toussaint, 1979b] suppliment the straight Viterbi algorithm with a dictionary lookup to guarantee the presence of the word in the dictionary while [Shinghal and Toussaint, 1979a] describes another variant of the Viterbi algorithm that uses heuristics to limit search depth in the trellis as the graph is very large.

2.3.5.2 Word Recognition with Ambiguous Segmentation

An almost overriding problem with systems based upon the Viterbi Algorithm is that incorrect letter segmentation cannot be handled properly. For example, the word *duck* written in script might easily be mis-recognized as *cluck*. The Viterbi Algorithm fails as it requires the correct letter segmentation. A system allowing ambiguous letter segmentation retains all possible segmentation points within a line drawing.

The mechanism used by this research to represent the ambiguously segmented letters is that of a letter graph, as mentioned earlier and documented in [Ford and Higgins, 1990]. Two important practical techniques that are used to perform word recognition on such a letter graph are *binary n-gram graph reduction* and a *dictionary tree* mechanism.

2.3.5.2.1 Binary N-Gram Graph Reduction

Using this mechanism, a list of valid *n*-letter sequences for members of the dictionary of valid words is created. This list can be used to assist in the removal of invalid *n*-letter sequences from the current letter graph.

[Higgins, 1985] discussed in detail the use of binary *n*-grams and decided that the optimum length gram was four. This is because only 5% of the 4-grams are valid in English. Also the number of 4-grams is $26^4 = 456,976$ is not too large to reasonably store in memory as a binary array. The choice of using the 4-gram is quite reasonable as a large percentage of 3-grams are valid and 5-grams would provide only a small amount of extra information.

Reducing the graph to a word using the 4-gram data structure involves tracing through the letter graph and marking valid and invalid sequences of letters. Various implementation methodologies are provided in [Whitrow and Higgins, 1987] and an analysis of this mechanism compared with other mechanisms is given in [Ford and Higgins, 1990].

2.3.5.2.2 Dictionary-Tree Mechanism

Using this mechanism, a dictionary (simply a list of words) is encoded as a tree. This tree is an implementation of the *trie* structure given in [Knuth, 1973]. Consider the example in Figure 2.3 taken from [Ford and Higgins, 1990].

The words can be determined by searching left to right starting at the @ symbol and ending at a # symbol. [Bozinovic and Srihari, 1982] used a stack-decoding algorithm and a dictionary tree and [Bozinovic and Srihari, 1989] added a search depth heuristic to cut down on required computation. The use of this mechanism was compared to the use of binary *n*-grams in [Ford and Higgins, 1990].

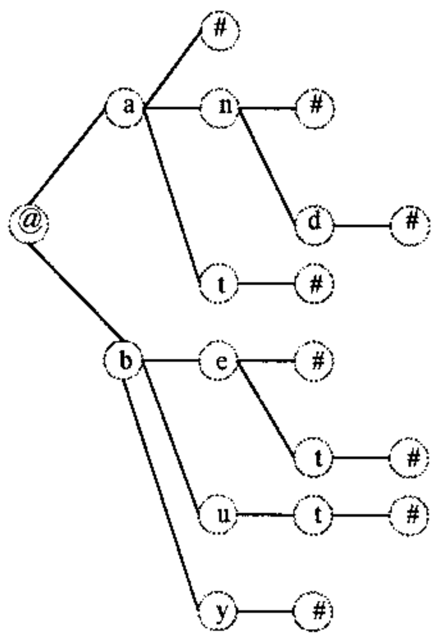


Figure 2.3. Example Dictionary Tree.
Tree for a very limited list of words that might be recognized.

The work done in this project uses something very similar to a dictionary tree when it is performing feature extraction. Once characters are extracted using FI and placed into the letter graph, a split *n*-gram technique is used where *n*-grams with wild cards are placed into search indices. These indices are used to help predict words where the letter graph does not specify a word from the lexicon with high enough confidence or where the letter graph is incomplete.

CHAPTER III

THINNING THE INPUT IMAGE

3.1 Review of Thinning

Thinning is the process of reducing the width of lines within a line drawing so that each line in the resulting line drawing will be one pixel wide and pixels at line intersections are kept at a minimum. The lines on the original line drawing will be of varying widths from three pixels wide to ten or so pixels wide depending upon ink flow and the resolution of the image. After an image is thinned, all lines should be exactly one pixel wide.

As with other types of pre-processing, information is lost when an image is thinned. The width of a line contains information about the ink flow and pen velocity as a slow pen can release more ink onto an document. Thinning can also introduce deformities, distortion, and other flaws into the shape of a skeleton.

In general, the process of thinning involves examining the pixels in an image containing a line drawing of connected black pixels on a white background and deleting black pixels until a skeleton remains. Generally, multiple passes are made over the image and in each pass, some pixels are deleted (set to white) in each pass and other pixels may be marked for later processing. The process of deleting pixels in an iterative fashion is known as iterative erosion.

The most famous of the algorithms [Pavlidis, 1982] moves a 3x3 two-dimensional window across the rows or down the columns and any neighborhood that meets certain criteria has a pixel removed from inside the window. Most of the other algorithms are similar.

Many different thinning algorithms have been proposed. Different thinning algorithms produce different kinds of skeletons and characteristic distortion. They can be classified into two general types: sequential and parallel algorithms. A parallel algorithm uses only the result from the previous pass or iteration to make decisions on whether pixels are removed. A sequential algorithm makes use of information obtained in the previous iteration and the current iteration to make pixel removal decisions.

Each of the various algorithms have their strong and weak points. Most were designed to meet the needs of a specific application and perform well in that application but perform poorly when applied to other applications. Survey studies [Chen, 1993] [Naccache and Shinghal, 1984] have indirectly analyzed the appropriateness of several of the more famous algorithms for use in recognizing hand-printed text or hand-written script.

For the skeleton to be useful to the application of concern, the thinning algorithm should:

- make sure that connectivity is maintained;
- make sure that end-points are maintained; and
- ensure that black pixels are stripped off symmetrically, so that the algorithm is isotropic.

During the early stages of this research, it was planned to use one of the thinning algorithms already existing in the literature. After searching the literature and experimenting with many of the thinning algorithms, it was determined that none of the famous algorithms produced skeletons that satisfy the three requirements mentioned above.

Some algorithms generate a good shaped thinned image, but with poor connectivity compared to the original image. This may be a critical fault when performing character recognition. Other algorithms do not maintain end-points very well and yet others leave the skeleton slightly deformed as pixels are not stripped off symmetrically. These problems can have quite negative effects upon the recognition process.

Most of the algorithms were invented for use in OCR applications where the resolution is much lower and line drawings are less complicated. Therefore a decision was made to create a new thinning algorithm tailored to the goals of connectivity maintenance, end-point maintenance, and guaranteed symmetric removal of pixels. The requirements for this algorithm were:

1. the algorithm must be simple;
2. the algorithm must guarantee connectivity in places where there is connectivity in the original image;
3. the algorithm must guarantee maintenance of end-points meaning that a line with an end-point should not be shortened by the stripping process; and
4. the algorithm must strip pixels from lines in a symmetric fashion so that no unexpected distortions are introduced.

The BRT Algorithm described in the next section fulfills the requirements.

3.2 Notation and the BRT Algorithm

Since the images that are of interest in this application are black and white images, it is assumed that each element of the drawing to be thinned is a black or white pixel. For the following discussion, it is assumed that a document exists in a binary image file. This document may be viewed as a drawing F_0 of connected black pixels residing on a background of white pixels.

The eight pixels neighboring any pixel p (points n_0 to n_7 in the following diagram) are defined to be the eight pixels surrounding pixel p.

$$\begin{array}{ccc} n_3 & n_2 & n_1 \\ n_4 & p & n_0 \\ n_5 & n_6 & n_7 \end{array}$$

Pixels n_0, n_2, n_4, and n_6 are known as the 4-neighbors of pixel p. Some researchers call these pixels the orthogonal neighbors [Beun, 1973]. All pixels $n_0 \ldots n_7$ are known as the *8-neighbors* of p. It is assumed that for any F_0 that contains a recognizable line drawing, a *skeleton*, $S(F_0)$, exists where each pixel $p \in S(F_0)$ resides on the medial axis of the drawing [Arcelli, 1985]. Informally, the medial axis is the set of all $p \in F_0$ where p is exactly in the "middle" of a line both distance wise and in orientation. The requirement for any thinning algorithm is that any pixel $p \in S(F_0)$ should be very near the medial axis.

A *skeleton* $S(F_0)$ of drawing F is said to be *4-connected*, if between any two black pixels p_0 to p_n in the skeleton, there exists a path $p_0, p_1, p_2, \ldots p_n$ where p_{i-1} is a 4-neighbor of p_i for all $1 \leq i \leq n$. A *skeleton* $S(F_0)$ of drawing F is said to be *8-connected*, if between any two black pixels p_0 to p_n in the skeleton, there exists a path $p_0, p_1, p_2, \dots p_n$ where p_{i-1} is a 8-neighbor of p_i for all $1 \leq i \leq n$. An *end-point* is defined to be a black point with at most one 8-neighbor.

Assume F_0 is the original connected set or multiple connected sets of black pixels in the original line drawing. Also define $B_0 \subseteq F_0$ to be the set of border pixels of F_0, where a *border pixel* is a black pixel that has one or more white 4-neighbors. Let $D_0 \subseteq B_0$ be the set of pixels that will be deleted in the 1st pass of the BRT algorithm. Further define $B_1 \subseteq F_1$ to be the set of border pixels in $F_1 = F_0 - D_0$. Similarly, $F_i \subseteq F_{i-1} \subseteq F_0$ is the connected set of black pixels remaining after i deletion passes have been made and $B_i \subseteq F_i$ is the set of border pixels of F_i. Likewise, $D_{i-1} \subseteq B_{i-1}$ is the set of black pixels deleted during deletion pass i where $F_i = F_{i-1} - D_{i-1}$.

A pixel $p \in B_i$ may be deleted during the $i+1st$ pass of the BRT algorithm if:

a. p is not an end-point
b. assuming p is deleted, then for all black 8-neighbors n_j and n_k of p where $n_j \in B_i \cup B_{i+1}$ and $n_k \in B_i \cup B_{i+1}$, n_j and n_k must be 8-connected.

and B_{i+1} is the partially completed border currently being marked. Using the above definition, $D_i = \{$all pixels $p \in B_i \mid p$ may be deleted in the $i+1st$ pass$\}$. Without loss of generality, assume that F_0 is a single set of connected black pixels in the original image. A high level specification of the BRT algorithm to thin F_0 may be simply stated:

```
$i = 0$
repeat
    $i = i + 1$
    $F_i = F_{i-1} - D_{i-1}$
until $B_i = B_{i-1}$
```

When the algorithm terminates $B_i = F_i = S(F_0)$. The technical details of the algorithm are given in Figures 3.3 and 3.4. In particular, the ordering of pixel selections for step b above is provided.

The BRT algorithm resembles the contour thinning of [Arcelli, 1980] where each B_i represents a figure contour. It is also related to the algorithm in [Arcelli and De Baja, 1985]. Those algorithms are parallel algorithms and are more complicated than the BRT algorithm. However, the parallel nature of the algorithms provide computational advantages. These algorithms are intended for thinning more than just line drawings representing handwriting and as such sometimes provide unexpected results.

The BRT algorithm, which is very simple and easy to code, produces an 8-connected skeleton because a skeleton of this type is more useful for the types of processing required later in this research. Another reason that 8-connected skeletons were produced is that 8-connected skeletons have fewer noisy branches or dendrites [Rosenfeld and Davis, 1976]. If it is desired to create a 4-connected skeleton, deletion criteria for the algorithm must be modified as follows. A pixel $p \in B_i$ may be deleted during the $i+1st$ pass of the BRT algorithm if:

a. p is not an end-point
b. assuming p is deleted, then for all black 4-neighbors n_j and n_k of p where $n_j \in B_i \cup B_{i+1}$ and $n_k \in B_i \cup B_{i+1}$, n_j and n_k must be 4-connected.

3.2.1 BRT Implementation

The thinning strategy used by the BRT Algorithm is quite simple. In brief, the strategy used is:

Loop through the following 2 steps until only a skeleton is left:
1. mark every pixel in the border of the line drawing;
2. delete every marked border pixel that can be deleted without deleting part of the skeleton.

A border pixel is a black pixel that is neighbor to a white pixel in some direction. The very simplicity of the strategy guarantees the symmetric removal of black pixels from the line drawing. The only real difficulty involves determining when you are about to delete a pixel in the skeleton.

Initially, the black and white image is read into a two dimensional array where each item in the array is stored in at least two bits. Therefore each item in the array can contain 4 different values, 0 through 3. If 0 is assigned to white and 1 is assigned to black, the values 2 and 3 can be used to mark border pixels. Figure 3.1 contains an image and its numeric representation. The name **q** is given to the two dimensional array where the image is stored.

Step 1 of the BRT algorithm involves marking every pixel in the border of the line drawing. A black pixel is considered to be in the border of the line drawing if it has at least one white 4-neighbor. Figure 3.2 shows the array **q** where the border pixels have been marked with a 2.

Step 2 involves deleting all border pixels that are not part of the skeleton. To determine whether a border pixel can be deleted or is part of the skeleton, each of the 8-neighbors must be examined. Therefore at each border pixel, a 3 by 3 window is centered on the pixel in question.

```
0000000000000000000000000000000
0000000000000111111000000000000
0000000000011111111110000000000
0000000000111100111111000000000
0000000001110000000011100000000
0000000011110000000001110000000
0000000011100000000001110000000
0000000011100000000001110000000
0000000011100000000001110000000
0000000011110000000011100000000
0000000001111000000111000000000
0000000000111000001110000000000
0000000000011110011110000000000
0000000000001111111100000000000
0000000000000111111100000000000
0000000000001111001110000000000
0000000000011110000111100000000
0000000000111100000011110000000
0001110001110000000001111111000
0001111111100000000000011111100
0000111110000000000000000111000
0000000000000000000000000000000
```

Figure 3.1. An Example Image and Its Numeric Representation

In contrast to most other thinning algorithms, the window is not moved across the rows of the image or down the columns. Once a pixel $p \in B_i$ is discovered and becomes the center of a window, it is guaranteed that at least one of the neighboring pixels $p_k \in B_i$ in the window is also a border pixel unless p is the last pixel from the current connected component of B_i to be considered. Using this information and the deletion criteria listed in Section 3.3, it is decided if the center border pixel p should be deleted or not. The window is then moved to be centered at one of the border pixels p_k which was a black 8-neighbor of the previous window center.

```
0000000000000000000000000000000
0000000000000222222000000000000
0000000000022122111220000000000
0000000000212200222212000000000
0000000002120000000021200000000
0000000021120000000002120000000
0000000021200000000002120000000
0000000021200000000002120000000
0000000021200000000002120000000
0000000021120000000021200000000
0000000002112000000212000000000
0000000000212000002120000000000
0000000000021220021120000000000
0000000000002112211200000000000
0000000000000211221200000000000
0000000000002112002120000000000
0000000000021120000212200000000
0000000000212000000021120000000
0002220002120000000002212222000
0002112222200000000000022111200
0000222220000000000000002220000
0000000000000000000000000000000
```

Figure 3.2. Example Image with its Border Marked.

In this fashion, once B_i is marked and the window is placed at some $p \in B_i$, the window is moved along the border from one pixel in B_i to the next, removing border pixels. Note, as the window is moved along the border, if a border pixel in the center of the window is deleted, some of the neighboring pixels in the window can be marked as new border pixels for the next iteration. Also, if the border pixel is not removed, that pixel must be marked as a border pixel for the next iteration. Border pixels that are marked in anticipation for the next iteration of the algorithm are marked with a different mark than the current border pixels.

At some point in time, the window will encounter a border pixel with no neighboring border pixels. This does not mean that the current iteration of the algorithm is complete. For example, in the image in Figure 3.2, the window would move all the way around the outside border removing border pixels first and then run out of neighboring border pixels (that could be discovered via window movement). The algorithm would then have to search further to discover the border pixels inside the middle of the loop. After the window has finished traversing the border pixels in the middle of the loop, then it will be time to move to the next iteration.

Figure 3.3 and Figure 3.4 give a pseudo-code version of the Border Reduction Thinning Algorithm where the legal_to_remove condition is defined in Section 3.3. The algorithm will make one or more iterations over the original image stored in array **q**. It will terminate when it makes an entire iteration and no black border pixels are deleted. When the algorithm falls out of the loop, the image stored in array **q**, containing only white and the latest border pixel value, has been thinned.

3.3 Analysis

The fact that this algorithm should produce skeletons where most pixels in the skeleton are very near the medial axis should be intuitively obvious. The boundary of the black pixels in the original line drawing is removed, then the boundary of the black pixels in the resulting line drawing is removed, in succession until only the skeleton remains.

```
mark all the border pixels in p to 2
initialize border_value = 2
initialize new_border_value = 3
initialize pixels_have_been_removed = true

loop while pixels_have_been_removed

    initialize current_border_pixel_found = true
    loop while current_border_pixel_found
        search array q for the upper/leftmost pixel containing: border_value
        if there was a q(i, j) = border_value
            set (x, y) to the coordinates (i, j)
            call routine:  window_around_border with x, y as parameters
        else
            current_border_pixel_found = false
        end if
    end loop

    if border_value = 2 then
        set border_value = 3
    else
        set border_value = 2
    end if

end loop
```

Figure 3.3. The BRT algorithm / Driver Portion

As the window proceeds moving from one pixel in B_{i-1} to the next deleting pixels, it is an easy task to mark the pixels in B_i. If a pixel p is deleted, then for any black pixels $n \notin B_{i-1}$ which are 4-neighbors of p, $n \in B_i$.

The only question remaining to discuss about the BRT Algorithm Implementation is: When is it legal to remove a pixel? A pixel may be removed, if after removal, it is

Window_around_border(x, y) [parameters - (x, y) of a border pixel]

```
initialize ran_out_of_border = false
loop while not ran_out_of_border

    establish 3x3 box around q(x, y)
    if q(x, y) has no non border black pixel 8-neighbors then
        if it is legal to remove q(x, y) then
            set q(x, y) = white
            set pixels_have_been_removed = true
        else
            set q(x, y) = new_border_value
        end if
    elseif it is legal to remove q(x, y) then
        set q(x, y) = white
        set pixels_have_been_removed = true
        mark all black 4-neighbors of q(x, y) to new_border_value
    end if
    search current box for neighbors of q(x, y) that contain value = border_value
    if a border pixel is found in a neighboring pixel
        set (x, y) to the coordinates of that pixel
    else
        set ran_out_of_border = true
    end if

end loop
```

Figure 3.4. Routine: Window_around_border
(legal_to_remove is defined in section 3.3)

guaranteed that the skeleton maintains connectivity and the end-points of lines are not whittled away.

Connectivity can be guaranteed in the skeleton if connectivity among neighboring pixels is maintained during deletion of a pixel. Assume the window is placed with its center over a border pixel candidate for deletion. Also assume that the window has other border pixels in it—generic border pixels including current border pixels (in B_i) and those marked for the next iteration (in B_{i+1}). Then, currently, there is connectivity between or among each generic border pixel in the window because a connection can go through the center pixel of the window. All that is required to guarantee connectivity in the skeleton is to allow removal of that center pixel only if after the deletion and subsequent marking of new border pixels, that connectivity still exists.

For example, consider the following 5 cases and assume the decision must be made concerning deletion of the current border pixel p (in the center):

```
. . b       . b *       . . .       t . b       . b .
. p *       . p *       . p .       . p *       . p .
t * *       . . t       t * b       b * *       t t .
Case 1      Case 2      Case 3      Case 4      Case 5
```

Assume **.** (a dot) represents white pixels, * represents black non_border pixels, **b** represents current border pixels with p representing the current window center border pixel (p, b $\in B_i$), and **t** is a border pixel marked for next iteration (t $\in B_{i-1}$). Also assume the pixel neighbor order:

$$\begin{array}{ccc} n_3 & n_2 & n_1 \\ n_4 & p & n_0 \\ n_5 & n_6 & n_7 \end{array}$$

In Case 1, if the center border pixel, p, is deleted, pixels in position n_0 and n_6 would become border pixels. This would leave generic border pixels in spots n_0, n_1, n_5, and n_6. There is a path from each border pixel to all others still, so the deletion of the center is allowed. (Pixel n_6 is connected to pixel n_0 by a diagonal.)

In Case 2, if the center border pixel is deleted, the pixel n_0 would be marked as a new border pixel. This would leave generic border pixels in spots n_0, n_2, and n_7. There is connectivity so the deletion is allowed.

In Case 3, if the center border pixel is deleted, the pixel n_6 would be marked as a new border pixel. This would leave generic border pixels in spots n_5, n_6, and n_7. There is connectivity so the deletion is allowed.

In Case 4, if the center border pixel is deleted, the pixels in positions n_0 and n_6 would be marked as new border pixels. This would leave generic border pixels in spots n_0, n_1, n_3, n_5, and n_6. The border pixel n_3 is not adjacent or connected to any of the other border pixels. If this deletion was allowed, there would be disconnectivity in the window and ultimately, this would cause disconnectivity in the skeleton as well.

Case 5 is a situation where most of the black pixels have been deleted and the window is searching the current border looking for any areas that may be left containing black pixels. This window contains only white and border pixels. It should be clear that if pixel p is deleted, then no new border pixels can be added and there will be a break in the skeleton. Note that connectivity among the remaining border pixels does not exist; therefore, the deletion would not be allowed.

In summary, all that is required to guarantee connectivity in the skeleton is to allow deletion of a pixel only if the remaining border pixels in the window are connected after the deletion. In other words, no pixel may be deleted that causes disconnectivity in the border whether the border is B_i or B_{i+1}.

To ensure that line end-points are not whittled away by the thinning process requires one additional check. For instance, assume that a line 5 pixels wide comes to an end-point. After the first thinning iteration is made and the outside border of the line is removed, the remaining line is three pixels wide. Note that the line has become shorter by one pixel. After the second thinning iteration is made and the outside border of the 3 pixel width line is removed, the remaining line is one pixel wide and it consists totally of border pixels. The next iteration, the window will move along this line and not delete any pixels until it reaches the end pixel. The window situation might be:

```
.  .  .
t  b  .
.  .  .
```

If the center pixel is deleted, then connectivity would still exist; but, if this is allowed, for this and each subsequent iteration, the thinning algorithm would excessively erode the line away each pass and it would continue because the loop terminates only when no pixels are deleted. In order to prevent this, simply require that: if a border pixel is to be removed, then there must be at least two remaining generic border pixels left in the window. With this additional requirement, removal in the above case would not be allowed.

Applying the pixel deletion criteria to the Figure 3.4 routine **Window_around_border** implies that the check:

it is legal to delete p(x, y)

is true when:

if the deletion of p(x, y) is allowed, there will be at least two remaining generic border pixels left and there will be connectivity among the remaining generic border pixels in the window

3.3.1 Skeleton Quality and Problems

The skeletons produced by the BRT thinning algorithm very closely approximate the original shape, retain the same connectivity, and maintain almost the same end-points as the original. These skeletons are much more appropriate for the application of hand-printed text and handwritten cursive script recognition than the existing algorithms studied. Note that the BRT algorithm is a sequential, multi-pass algorithm. The ever present trade-off between quality and execution speed is present.

The application of cursive script recognition as it is implemented in this research requires a very good skeleton or the recognition process will not perform well. For this reason, the small amount of extra computation that is required by the BRT algorithm is worth the trade-off. Figure 3.5 shows some originals and skeletons that are produced by the BRT algorithm.

The problem of dendrites, a problem with thinning in general, was not solved in the BRT algorithm. A **dendrite** or noisy branch is a short line existing in the thinned image but one that should not be present. After some experimentation with thinning algorithms, it was found that most of the thinning algorithms encounter dendrite problems.

The removal of dendrites from the skeleton is a particular sticky problem. It is not clear that it even should be done. How is it really known that the extra line is a dendrite and not an intended extra short line by the author of the original image? A good compromise might be that short lines of length less than some threshold are removed where the threshold is carefully chosen. This solution was chosen for the current research, but only applied in a postprocessing fashion in the line segment approximation phase. Large dendrites caused by large deformities in the original line drawing would cause problems with even human recognition.

Dendrites are caused by minor flaws in the original image. A usual cause is an ink smear on the original or a place were pixels that should be present have been noisily removed. The two cases are analyzed below. Consider a line in a drawing that has a tiny outgrowth of pixels such as the following, where periods represent white pixels and lower case *'s represent non-border black pixels in the line and b's represent border pixels.

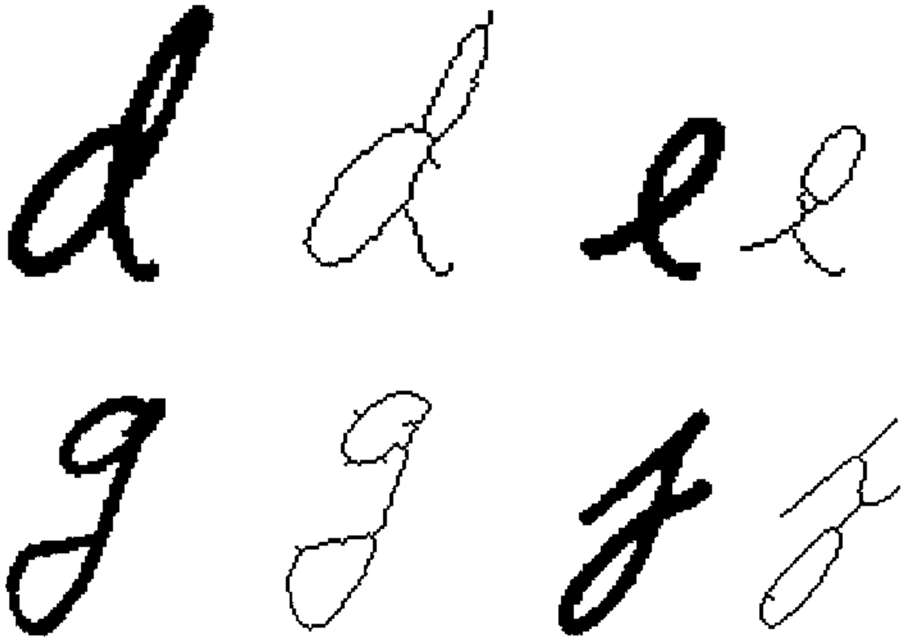


Figure 3.5. Example BRT Skeletons
Several original images and the corresponding skeletons

```
. . . . . . . b . . . . . .
. . . . . . . b b b . . . .
b b b b b b b * * * b b b b
* * * * * * * * * * * * * *
```

During the next iteration of the BRT algorithm, the algorithm would thin to the following:

```
. . . . . . . b . . . . . .
. . . . . . . b b b . . . .
. . . . . . p * * * b b b b
t t t t t t * * * * * * * *
```

where the t's represent border pixels for the next iteration and p represents the current center of the window. The algorithm would delete p, mark its 6-neighbor and 0-neighbor as new border pixels. The window will then move to be centered at the pixel represented by the bold **b** in the above diagram. This window center was chosen because it is the only current border pixel still surviving. This border pixel would then be deleted as it fulfills all the requirements of deletion producing the following:

```
. . . . . . . b . . . . . .
. . . . . . . . b b . . . .
. . . . . . . t * * b b b b
t t t t t t t * * * * * * *
```

The algorithm would then choose the now bold **b** above. It could not remove that border pixel because it is an end-point. Ultimately after a couple of iterations, these pixels would create a short line, a dendrite, in the skeleton.

The other type of dendrite is caused by missing pixels usually near a turn in a loop. Consider the following circumstance representing the turn in a loop:

```
. . . . . . . x . . . . . .
. . . . . x x x x x . . . .
. . . . x x x x . x x . . .
. . . x x x x x x x x x . .
. . x x x x . . . x x x x .
```

Notice the missing black pixel in the line. The pixels around that missing pixel become border pixels by default. As the thinning algorithm reduces the outer border and the inner loop border, it also thins around that border. One iteration produces the following:

```
. . . . . . . . . . . . . .
. . . . . . . X X . . . . .
. . . . . X X . . X . . . .
. . . . X X . X X X X . . .
. . . X X . . . . . X X . .
```

After another pass, the final extra pixels will be removed:

```
. . . . . . . . . . . . . .
. . . . . . . X X . . . . .
. . . . . . X . . X . . . .
. . . . . X . X X X . . . .
. . . . X . . . . . X . . .
```

So the ultimate result would be the correct loop with a tiny extra loop at a turn.

These dendrite problems plague thinning algorithms and, in many instances, they can create problems in the recognition process. This is especially true if the recognition methodology is based on structural analysis of the skeleton (as this research is based) and the structure has an extra line or loop in it.

One possible solution to this problem involves a post-processing pass to the thinning algorithm to remove dendrites. However, this extra work was not needed in this research since the next phase of image pre-processing, the line segmentation approximation phase discussed in the next chapter, eliminates the problems.

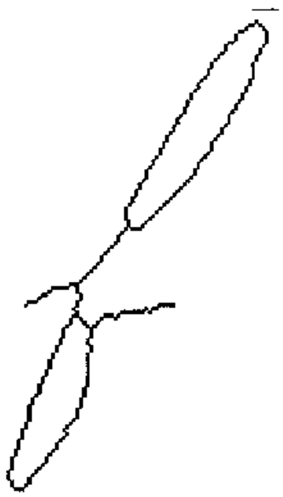
CHAPTER IV

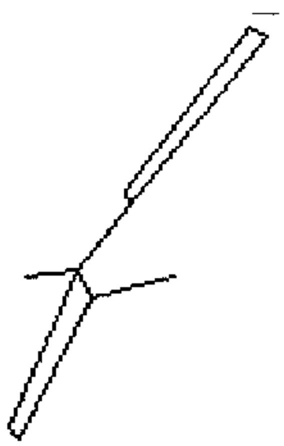
LINE SEGMENT APPROXIMATION

4.1 Overview

Line segment approximation, or vectorization, involves the conversion of an input skeletal line drawing, such as might be output from the BRT algorithm, into a set of points and edges. These points and edges, when drawn on a two-dimensional graph, approximate, in a piece-wise fashion, the shapes of the line drawing represented by the skeleton. Figure 4.1 shows a thinned line drawing, its line segment approximation that was created with the algorithm described in this chapter, along with a drawn version of the line segment approximation.

The thinning process greatly reduces the amount of data in an original line drawing while maintaining information about the shape and connectivity. The main purpose of line segment approximation is the further reduction of data that must be processed in later stages of application processing. If high quality line segment approximations, or **singularity graphs**, such as that in Figure 4.1, can be accomplished, the amount of data can be greatly reduced with only a small loss in shape information. For example, the singularity graph shown in Figure 4.1 has only 10 singularity points and 11 edges which join the points. In this case, a **singularity** point can be thought of as an end-point for one or more of the graph edges.





```
Image:00  (005, 005) to (080, 139)

        sing_start=00  n_sings=10
        path_start=00  n_path_items=11
        loop_start=01 n_loops=02

Singularities
NDX    (j,  i)       stype  nloop         #
00     (040, 063)      i      01         001
01     (039, 058)      f      01         001
02     (075, 005)      f      01         001
03     (080, 008)      f      01         001
04     (025, 084)      i      01         002
05     (029, 093)      i      01         002
06     (053, 086)      e      00
07     (008, 139)      f      01         002
08     (005, 135)      f      01         002
09     (010, 086)      e      00


Singularity Path
edge    fr-to     theta
00.     00-01     01.768
01.     01-02     00.974
02.     02-03     05.743
03.     03-00     04.084
04.     00-04     04.092
05.     04-05     05.131
06.     05-06     00.284
07.     05-07     04.284
08.     07-08     02.214
09.     08-04     01.197
10.     04-09     03.274


Loops
loop nbr    center      singularities
01         (033, 059)   000--003--002--001
02         (113, 016)   004--008--007--005
```

Figure 4.1. An Example Singularity Graph
The skeleton of character 'f', its singularity graph, and its drawn singularity graph.

In this research, the singularity graph is used to extract features. Many other methods have been proposed. For example, some methods

- analyze the entire line drawing without thinning [Srihari and Bozinovic, 1987]
- analyze the thinned figure [Aoki and Yamaya, 1986] [Aoki and Yoshino, 1989] or
- analyze the outer and inner contours of the line drawing [Cohen et. al., 1991].

These methods must examine possibly hundreds or thousands of pixels to recognize individual features. However, examining only the edges and nodes in a singularity graph can be quite a dramatic improvement over these methods.

The disadvantage to using singularity graphs is that during the vectorization process some information is lost and deformities may be introduced. It is extremely important to obtain a singularity graph that very closely approximates the original skeleton and simultaneously reduces the data to allow efficiency in the recognition process.

4.2 Notations and Definitions

A **singularity** in a curve of a skeleton corresponds to a pixel *p* which is the optimum point to subdivide the curve into two line segments such that when the segments are drawn, the segments more closely approximate the given curve than any other possible choice of singularity. Any intersection point in a line drawing, as well as an end point, are also considered singularities.

A **singularity graph** for a skeleton S, *SG(S)*, is a graph where

- the **nodes** represent the set **N** of all singularities of S and each node is labeled with an (x, y) coordinate indicating the location of the singularity in the image;
- the **edges** represent a set **E** of connecting line segments that connect nodes.

such that when a two-dimensional drawing is made using the (x, y) coordinates that label each of the nodes in N, the set of drawn edges E resembles the original skeleton.

The greater the number of singularities, the more closely the drawn singularity graph can approximate the skeleton. In the extreme case where the set of singularities N represents the entire set of pixels in the skeleton and there are edges between any two 8-

connected pixels, the singularity graph would be a duplicate of the original skeleton. The problem with using this singularity graph is that there is, in essence, no data compression. If one out of every two pixels are selected to be nodes in the singularity graph, then data compression is 50% with very little loss of information.

The desired algorithm produces a singularity graph that chooses singularities at irregular intervals in an effort to model the length of the pen strokes that created the drawing with the exception of intersection singularities and end-point singularities. Therefore, edges will have differing lengths but the lengths will more closely model the pen strokes. This problem has been addressed in the literature and was discussed in Chapter II.

4.3 Methodology

The process used to create a singularity graph in this research involves two phases:

1. Marking the points in the skeleton where the singularities exist; and
2. Examining the skeleton and creating the singularity graph
    - with nodes for each singularity marked in step 1 and labeled with singularity pixel coordinates; and
    - with edges connecting nodes determined by connectivity information present in the skeleton.

Marking the points in the skeleton where singularities exist is the most difficult phase. Singularities must be selected such that the number of singularities chosen is minimized, yet the shape of the drawn singularity graph created must closely resemble the skeleton. The trade-off is that the more singularities chosen, then the more closely the resulting drawn singularity graph can resemble the skeleton; conversely, the fewer chosen, the less the drawn singularity graph will resemble the skeleton. The trick is to choose just the right number.

Once the singularities are marked, the skeleton must be processed to determine the nodes and edges of the graph. The mechanism used is a recursive traversal of the skeleton starting at a known singularity such as an intersection or end-point. A node for the beginning singularity is inserted into the graph and the traversal process begins. The traversal recursively follows along the skeleton, in a specified order of travel, keeping track of which node represents the last singularity it encountered, called the current singularity. When another singularity is discovered, a new node is inserted into the singularity graph along with an edge from the current singularity to the new singularity. Then the new singularity is marked as the current singularity and the traversal continues.

When a situation is encountered that involves a recursive backup, such as reaching an end-point in the skeleton or reaching an intersection where all exiting lines have already been traversed, then the traversal algorithm returns to a place on the skeleton where it can continue. The singularity, that is returned to, is then marked as the current singularity and the traversal continues.

4.4 Marking Singularities with FI

The method used to mark singularities is a variation of a pattern recognition technique called Finite Induction (FI) [Fisher and Case, 1984]. A short introduction to FI is presented in the Appendix. It is a generalized pattern matching technique that is a mathematical pre-algebra and can be very effective as is demonstrated in this research.

Using FI, patterns that the system has learned (has been trained with) are stored in **FI rulings** in a process called **FI factorization**. A ruling is similar to a context free grammar and can automatically be generated by FI factorization. Once a database of one or more FI rulings has been acquired, the FI rulings can then be used to recognize patterns from input data. The recognition process is called **FI following**. The activity of performing FI following on input data according to an FI ruling produces a "closeness to a match" measurement called an **FI residual**. The FI residual, or the length of the residual, may be used to measure the closeness of the pattern in the input data and the pattern that is represented by the FI ruling. A short residual length generally implies a very close match between the input pattern and the pattern represented by the FI ruling.

For this application, a short target pattern must be recognized as present in a long stream of data. For this reason the actual residual length can not be used as a closeness measurement. What is used is the local **residual density**. In this research, the local residual density is defined as the length of the residual in a contained local area where the size of the local area to be checked is maintained in the exemplar and is a relative measurement dependent upon the height of the text. Actually, the residual density is calculated and maintained for three contained local areas of the skeleton, where each local area is of a different length. If the density of the residual in any of the local contained areas ever gets low enough, then a match is signaled.

4.4.1 Input to the FI Following

In previous work, a chain-graph mechanism was used to represent the shape of a skeletal figure by specifying directions of movement when following along the skeleton [Aoki and Yoshino, 1989]. For example, if a skeleton contains the following set of pixels (where . represents white and x represents black):

```
. . . . . . . X .
. . . . . . X . .
. . . . X X . . .
X X X X . . . . .
```

and directional indicators are specified by the following diagram:

```
3 2 1
4 p 0
5 6 7
```

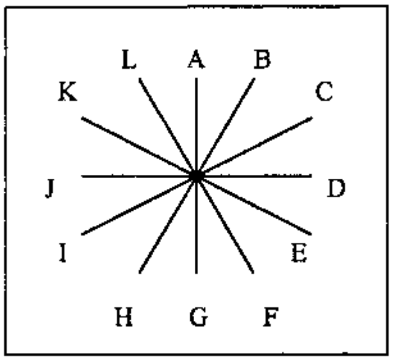
Directional Indicators

where each number in the perimeter of the diagram represents a direction from pixel p, then from the given leftmost starting point, the chain graph 0-0-0-1-0-1-1 represents the skeleton information above.

To begin the singularity search, a recursive traversal of the skeleton is begun from a known singularity pixel (at an end-point in the skeleton or at a line intersection). The traversal then recursively moves over the skeleton providing chain-graph type directional indicators to the FI following process as it continues. The traversal continues until there is no more skeleton to process.

4.4.2 Using FI Following to Mark Singularities

A set of twelve FI rulings was created to represent straight lines in various directions. Twelve directions were chosen to match the directions in the following diagram:



where each angle, ∠ AB, ∠ BC, ∠ CD, etc. are each 30°. Assume that O is the mid-point in the above diagram, sets of pixels in the equivalent of each segment OA, OB, OC, OD, … OL were used in FI factoring for 12 FI rulings. Each of the twelve FI rulings will be used to recognize any curve in a line drawing that travels approximately in the direction Oi for 'A' ≤ i ≤ 'L'. These rulings were created only once during the development of the algorithm to mark singularities

Given that the FI rulings for each vector type are known, the algorithm to mark the singularities is quite simple. Begin a traversal at an already known singularity. This will be at an intersection or an end point as singularities automatically exist in these places. The chain code directional indicators from the skeleton traversal will be provided by the traversal as input to the FI following process. Then the FI following process is initiated.

As the traversal continues, whenever the FI following process determines a "match" using the local residual density indicator, a singularity is marked in the skeleton. The FI following process continues until the skeleton has been completely traversed. The FI following process over a skeleton must be repeated for each FI ruling in the above ruling set to determine all singularities.

As the FI following proceeds, the residual density of three local areas is maintained, a long, a medium, and a short area. If the density gets below the threshold for any of the three areas, then a singularity is marked by placing a marker into the image. The density thresholds were carefully selected by hand and work exceptionally well. The marker can be any color pixel other than black or white in the already mentioned array that contains the line drawing.

4.4.3 Creation of the Singularity Graph

Once the singularities are marked within the skeleton, one more pass is needed to create the singularity graph. As described earlier, a recursive traversal is started at an important singularity, usually an intersection singularity or an end-point singularity. A node is created in the singularity graph labeled with the (x, y) coordinates of the initial singularity. The traversal recursively follows along the skeleton, in a specified order of travel, keeping track of the last singularity that it passed. It is referred to as the current singularity.

When another singularity is discovered, a new node is inserted into the graph along with an edge that connects the current singularity with the new singularity. This new singularity is a point in the skeleton that one of the following passes had marked as a singularity or it is an intersection or end-point in the skeleton. The new singularity is then established as the current singularity and the process continues. When a situation is encountered where the traversal can not continue, such as reaching an end-point in the skeleton or reaching an intersection where all exiting lines have already been traversed, then the traversal returns to a recursion point in the skeleton, and takes a remaining untraversed path.

A recursion point is a place in the skeleton that is at the intersection of two lines. Previously during this pass, the traversal had encountered this point in the skeleton. There were two or more paths that could be selected to continue. The traversal had selected one of the paths and had continued in that direction. When the traversal encounters a spot in the skeleton where it cannot continue, such as an end-point or an intersection point where all other paths have previously been processed, it will return to this recursion point. If there still exists an untraversed path from this point, the traversal will continue in that direction; otherwise, it will return to the next previous recursion point. When it returns to the initial recursion point and there are no other paths to take, the FI following is complete along with the singularity graph. See Figure 4.1 for an example containing a thinned letter 'f', the drawn singularity graph for the thinned letter 'f', and a listing of the singularity graph in tabular form.

4.4.4 Final Analysis and Comments

Once the vectorization process is completed, a singularity graph is available. The nodes of the graph are labeled with the (x, y) coordinates for the singularities with respect to the skeleton. Along with the (x, y) coordinates for each singularity, the type of singularity is also recorded. Singularity types are: **end-point** singularities, **intersection** singularities where lines cross, and **flow-through** singularities which represent the singularities selected in a curved line used to approximate the curve with line segments. The edges of the singularity graph consist of a list of (s_1, s_2) pairs where s_1 and s_2 represent the singularities that are the end-points of the edges. Also for each edge (s_1, s_2) in the set of edges, an (r, θ) field is maintained such that if (x, y) is the vector representing the s_1 component of the (s_1, s_2) singularity pair, and (x', y') is the rectangular coordinates of the vector represented by (r, θ), then (x + x', y + y') equals the (x, y) coordinate of s_2.

The above process produces very high quality vectorized approximations of input skeletons. Figure 4.2 shows several thinned images with their drawn vectorized approximations. The quality of the singularity graph is good enough for the singularity graph to be used as the basis of the feature extraction process instead of the skeleton itself. Note that the drawn images in Figure 4.2 only approximate the singularity graph and that imperfections are caused by the enlargement process and the quality of the line drawing routine.

Using this method to perform vectorization has a trade-off. The high quality singularity graphs produced by this method require 13 passes over the input skeleton, one

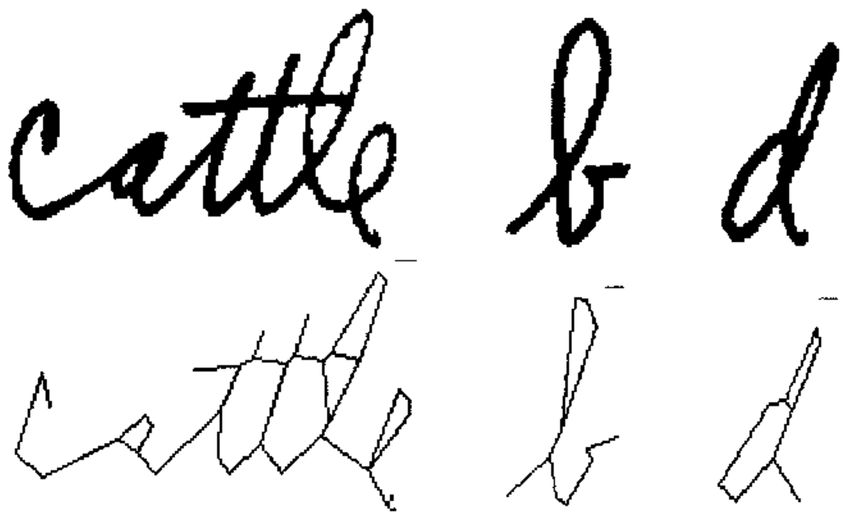


Figure 4.2. More Examples of Drawn Singularity Graphs
Original line drawings (greatly enlarged)and the (similarly enlarged) drawn

for each ruling to mark the singularities and one to create the set of singularity graph nodes and edges. Other algorithms discussed in Chapter II only required one pass. However, the other algorithms performed much more work per pass than this algorithm, and, even if more passes were involved, it was decided that a high quality singularity graph was worth the extra effort.

4.5 Post-processing the Singularity graph

The dendrites produced by the thinning process have to be addressed at this point. An easy method to recognize and remove dendrites is to look through the singularity graph and find edges where the r coordinate of the (r, θ) component is less than an established dendrite length threshold. If one singularity in the (s_1, s_2) pair is an end singularity and the other is an intersection singularity, then the edge can be removed from the set of edges and the node representing the end singularity of the pair can be deleted.

Other simplifications are also possible. As an example, consider the following situation in a skeleton where s represents a singularity and an x represents a black pixel in the skeleton:

```
s1
x
x
x
x   s3
s2 x
x
x
s
```

There are three edges in the set of edges for this singularity graph, (s_1, s_2), (s_2, s_3), (s_2, s_4). Obviously the (s_2, s_3) edge is a dendrite. Also the (x, y) coordinates for each singularity is known, so it is easy to remove the dendrite from the skeleton. However, since later processing only involves the singularity graph, there is no longer a need for the skeleton. So simply remove the (s_2, s_3) edge from the singularity graph, combine the vectors (s_1, s_2) and (s_2, s_4) into a single vector (s_1, s_4), and delete the two singularity nodes s_2 and s_3.

As one final note, the mechanism to build a singularity graph builds the graph as a di-graph. It is an easy matter to add extra edges in the graph to create an undirected graph at a later time. In this case, for each edge (s_i, s_j) in the singularity graph, (s_j, s_i) must be added as a new edge.

CHAPTER V

FEATURE EXTRACTION

5.1 Background

This phase of the application involves examining the input, consisting now of a singularity graph containing a vectorized approximation of a skeleton, and attempting to discover the presence of features. These features will be used later as input to the recognition engine which finds higher level features—characters and character constructs.

The feature extraction phase is considered by many to be one of the most difficult phases in the entire area of pattern recognition [Impedovo et. al., 1991]. It has been widely studied with respect to OCR and handprinted text recognition (see Chapter II) but much of that work is not directly applicable to recognition of cursive script. In OCR and handprinted text recognition, sets of features at this level are extracted for use in classifying entire small line drawings representing characters. In this research into connected cursive script recognition, sets of features at this level are extracted for use in extracting larger features called character constructs in large line drawings representing words or partial words. The character construct features are used, in turn, in classifying the large line drawings representing words and word constructs.

If features are to be recognized within a large line drawing representing several characters, then a method of managing large numbers of features is required so that the features and their topological organization are conveniently available during the character extraction phase. In this fashion, a subset of features in the total set of features must be used to recognize a character construct within the large line drawing.

For example in OCR, once features are extracted, character recognition is straightforward. Feature extraction many times involves discovering the feature vector for an input line drawing (character). If the character contains one hump, two loops, a slash and a zero count for other features, the feature vector might be (1,0,0,2,0,1). To classify the character, these feature vectors are compared against known exemplar feature vectors for each character that may be recognized possibly using a Euclidean or weighted Euclidean distance as the measurement:

$$D_j^E = \sqrt{\sum_{i=1}^{N} w_i \left(F_{ji}^L - F_i^I\right)^2}$$

where D_j^E represents the Euclidean distance between the input character and feature *j* in the library *L* of known feature vectors. F_{ji}^L is the *i*th feature in the *j*th vector in library *L*. F_i^I is the *i*th feature of the input vector, w_i is the weight applied to feature *i* as some features may be more reliable than others. *N* is the number of features. The character would be classified as the one with the smallest distance.

Feature extraction in OCR and handprinted text recognition reflects the requirements of the classification process used. Feature extraction in cursive script recognition must do the same. In cursive script recognition, a feature count vector is of little use. The relative position of the features in the line drawing, or singularity graph for the line drawing, is important—not just the feature counts. If unambiguous character segmentation were possible, it would be less important. However, assuming fairly unconstrained script, assessing where characters begin and end in the line drawing involves large amounts of ambiguity. An efficient mechanism for organizing features, which may overlap, so that they can be easily scanned and associated locally with other features is important. In this research, the **feature graph**, which is output from this phase, was the mechanism chosen. The mechanism used to perform character extraction is discussed in Chapter VI.

The vectorization phase, that was discussed in Chapter IV, can also be considered part of the feature extraction phase. Each edge and singularity of the singularity graph can be considered a feature. In this case, the features recognized in this phase are joined together with the singularity graph of the skeleton to create an expanded singularity graph, known as the feature graph. The feature graph is then used as input to the higher level feature recognition engine which recognizes the characters and character constructs.

Many different kinds of features are present in cursive script and handprinted text. It is important to consider very carefully the kinds of features that will be extracted. It is obvious that the types of features useful for cursive script recognition differ from those used in OCR and handprinted text recognition. The problem is to extract features which will enable the system to discriminate efficiently between and among characters and character constructs in later phases.

The main requirements for a feature extracting mechanism in this research are:

- the mechanism has to be computationally efficient;
- the recognition mechanism must be very thorough in that it recognizes every feature for which it has been trained;
- the mechanism must be flexible in that it may easily be trained with new features and new shape possibilities for existing features; and
- the data structure created to contain the features that are discovered must be an efficient mechanism for organizing features so that they can be easily scanned and associated locally with other features.

5.2 Features Extracted

After a close examination of the alphabet, especially the lower case alphabet, a small set of features was selected for extraction. This set consists of several kinds of loops, a feature that looks like a c, a feature that looks like a backwards c, a feature called a hump which is part of an m, and a feature that looks like a u. These features along with FI rulings of the feature graph for each character construct will be used to extract characters. These features were purposefully chosen to keep the number of features small. See Figure 5.1 for a hand drawn visual image of the features chosen.

The features were selected to enable a high correct classification rate for lower case letters. The same features were used in the description of upper case characters with less success. It was for this reason that the design of the feature extraction mechanism includes the flexibility for the implementor to add features as new and desirable features are discovered. Use of these new features must be coordinated with the character extraction mechanism that will use the features to build new abstractions for recognizable characters.

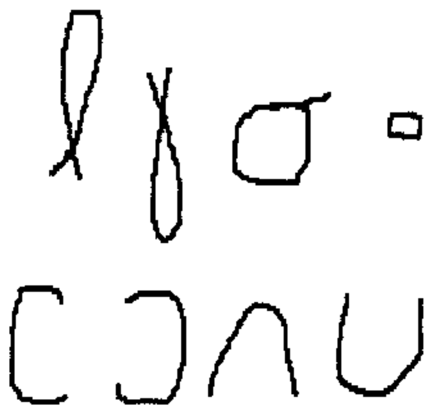


Figure 5.1. Features Extracted (hand drawn).
Features include a long narrow up loop, long narrow down loop, round loop, small round loop, c-type, backwards c type, hump, and cup.

The data structure used to store exemplar features is described in Section 5.3.2.2. The structure can store and efficiently access large numbers of features, with many different variations on each feature.

5.3 Feature Extraction Mechanism

The features discovered by the feature extraction mechanism are divided into two categories reflecting the method used to extract the features. There are loop type and non-loop type features. A loop is any set of vectors in the singularity graph where there is a path from a singularity back to the same singularity. The other four features fall into the non-loop category.

5.3.1 Loop Extraction Methodology

The types of loops extracted were subdivided into categories. The two main categories were: single intersection loop and multiple intersection loop. Single intersection loops involve loops that have exactly one intersection point. Consider Figure 5.2.a containing a cursive 'l' (lower case L) and a cursive 'y'. Both are single intersection loops. The 'l' contains a single intersection loop with direction 80° from the intersection point. The y contains a single intersection loop with direction 260° from the intersection point. The loop directions may be used for heuristic purposes during the later recognition process. The length, width, and center of a loop is also recorded so that the loops can be further subdivided into long loops and round loops. A multiple intersection loop is shown

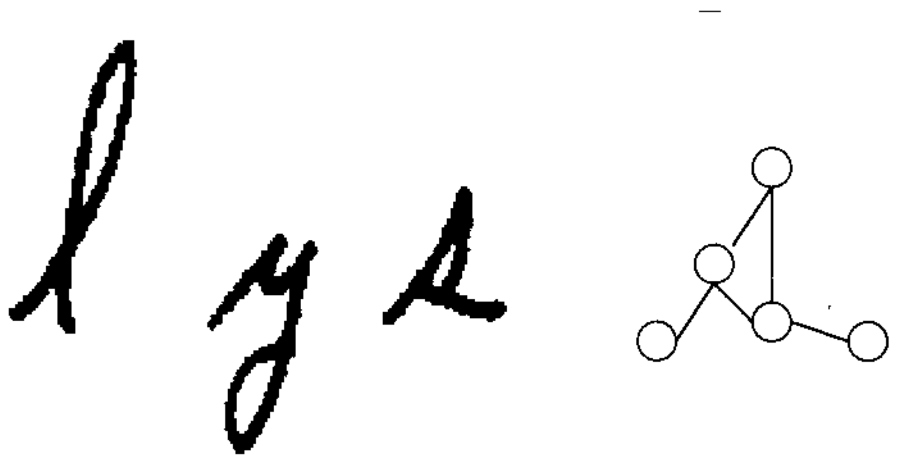


Figure 5.2. Examples of Loops
Single intersection and multiple intersection loops.

in Figure 5.2.b in the letter 's'. A drawn singularity graph for the character containing the loop of Figure 5.2.b is displayed in Figure 5.2.c. Notice how in the singularity graph for 's', there are two intersection singularities in the loop. An intersection singularity is any node incident with more than two edges.

For the purposes of this research, a loop is any set of vectors in the singularity graph where there is a path from a singularity in the loop back to the same singularity. For example, the letter d shown in Figure 5.3 has six loops, 3 loops that do not contain any other loops, 1 loop that subsumes the top two loops, 1 loop that subsumes the bottom two loops, and 1 loop that subsumes all other loops. As will be discussed later, each loop is considered a feature and is placed in the feature graph.

The process for the recognition of loops involves traversing the singularity graph looking for situations where there is a path from a singularity back to that singularity. Given a singularity graph, a two dimensional connection matrix form of the graph is

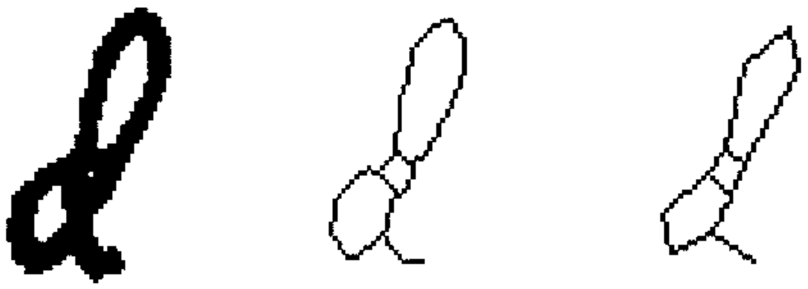


Figure 5.3. An Example of Subsumed Features
The original version, thinned version, and drawn singularity graph of a 'd' containing 6 loops.

created containing singularity connection information. A traversal of the graph is done stacking singularities as they are found. If a singularity is found that is already on the stack, a loop has been discovered. The loop is recorded, the stack is popped back to the last intersection singularity placed on the stack, another direction is chosen (if possible) from that singularity and the process continues. If no other edges from that intersection singularity can be found, then the stack is popped again to the next intersection singularity in the stack and so on.

The different types of loops are distinguished heuristically by checking the known height of the script with the length and width of the loop. If the length of the edges are "long" relative to the known height of the script and the width is much smaller than the height, a long narrow loop exists. If the length and width of the loop is near the same, then a round loop exists which can be either large, regular, or small depending upon the length of the edges relative to the known height of the script. The loop direction is the angle of the vector that points from the loop intersection point to the singularity in the loop that is the furthest Euclidean distance from the intersection singularity.

5.3.2 Extracting Non-Loop Features

After the pass is made over the singularity graph to extract loops, another pass is made to extract non-loop features. As mentioned above, the non-loop features extracted include the "c" shape, the "backwards c" shape, the "hump" shape, the "u" shape or cup shape.

5.3.2.1 Design of Non-Loop Feature Extractor

As this phase is the most difficult and time consuming phase of similar projects in the literature, steps were taken to make this phase a time efficient phase. Figure 5.4 contains three different edge groupings that represent a "c" shape. It should be apparent that there are many other sets of edges that might be placed together to obtain a "c" shape. The method chosen to extract features involves manual training of the feature recognizer and storing the various possible edge direction/length combinations in an efficiently accessible data structure.

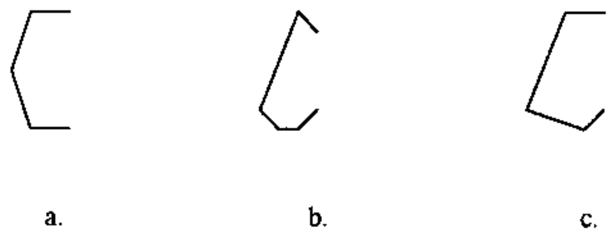
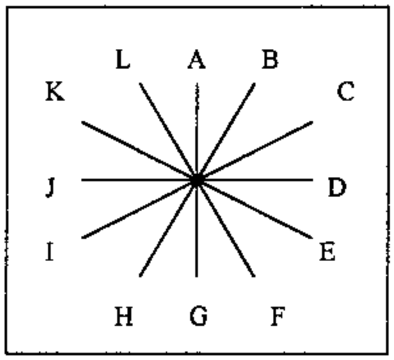


Figure 5.4 Example Feature Exemplars
Three different singularity graph edge patterns for representing the feature "c"

During early experimentation, many different edge combinations for each feature were discovered. These approximate edge combinations were translated into an alphabet suitable for storing into the data structure. The alphabet involves the approximate direction of the edge as a vector beginning at the source singularity and an approximate length of the vector, either short, medium, or long. The directions are illustrated in the following diagram:



The lengths were S for short, M for medium, and L for long. Therefore, the “c” feature in Figure 5.4.a is represented as JS, HM, FM, DS. Figure 5.4.b is represented by the sequence LS, HL, FS, DS, BS, and Figure 5.4.c is JS, HL, EM, BS. The definitions for the 3 variations on the “c” type shape depend upon the starting singularity being the upper right end-point. Each time a new possible representation for a c shape was discovered, it was included in the list. The same process was followed for each of the features. The resulting set of feature patterns are stored in an efficiently accessible data structure which is placed in memory during the extraction process. As the number of features to be recognized here is only four, then the entire feature extraction data structure has less than 160 entries which averages a little less than 40 entries per feature.

5.3.2.2 Feature Base Data Structure

The list of features is stored in a variant of the *trie* structure suggested in [Knuth, 1973]. It is similar to the dictionary tree suggested in [Ford and Higgins, 1990] except that it is used as an efficient mechanism to store patterns for features extracted from singularity graphs instead of patterns of letters for words in a dictionary. As possible edge combinations for features are discovered, they are inserted into this trie called the **exemplar feature tree**. An example subset of the exemplar feature tree is shown in Figure 5.5.

Figure 5.5 shows three different possibilities for subsets of singularity graphs for the feature "c" taken from Figure 5.4. Notice at the right side that there are dotted

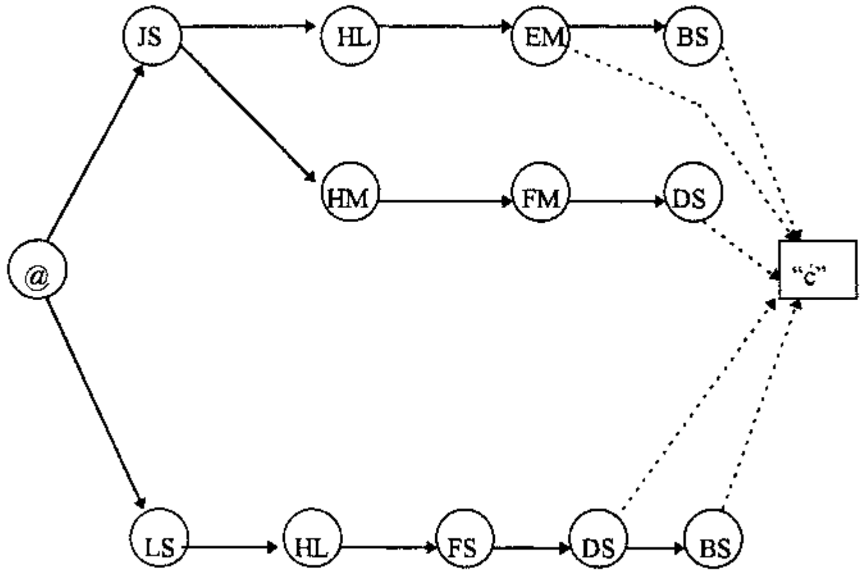


Figure 5.5. A Subset of an Exemplar Feature Tree. This represents three different edge combinations representing the letter "c".

branches that point to a box containing the c. Each dotted line represents a recognition point for the feature "c". Each dotted line is implemented as a confidence value containing the experience of the feature recognition system. Compared to the total number of c features that have been recognized, the dotted line specifies the number of times this recognition point was used.

5.3.2.3 The Process of Extracting Features

The process of extracting features from an input singularity graph is fairly simple given the exemplar feature tree. As a global traversal over the singularity graph progresses, each singularity is assumed to be the beginning of a feature. A localized traversal is begun starting at the currently selected singularity directed by the exemplar feature tree. The direction and length of each edge of the singularity graph connected to this singularity becomes the first edge in a predicted feature. If there is a feature in the exemplar feature tree beginning with this edge, then the local traversal continues, otherwise a failure is signaled and traversals, starting at the other edges, are begun. As a traversal progresses, each possible edge in the path is compared with a corresponding edge in the exemplar feature tree and the traversal continues as long as there are matches. A success is reported when correct feature marker is discovered at a node in the exemplar feature tree. This situation is equivalent to discovering a dotted line in Figure 5.5. A failure is reported when the next edge in the input singularity graph does not match any edge in the exemplar feature tree.

For example, consider Figure 5.6 which represents a sub-graph of a singularity

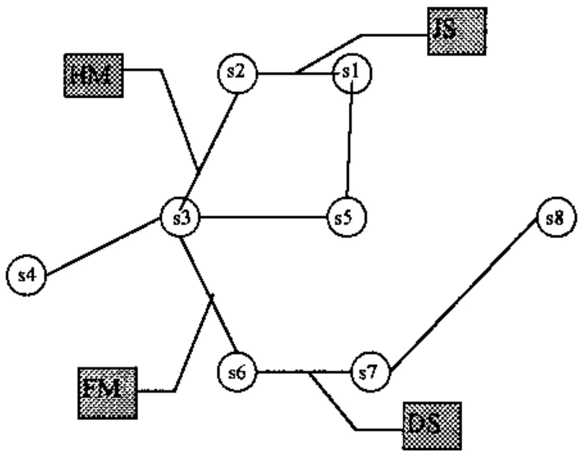


Figure 5.6 Example Labeled Singularity Graph
Subset of a singularity graph labeled showing the feature ‘c’.

graph where the circles represent singularities and the gray boxes represent annotations. Several features are present in the graph. These include:

• s3 - s2 - s1 - s5	hump
• s2 - s3 - s5 - s1	cup
• s2 - s3 - s6 - s7 - s8	cup
• s2 - s1 - s5 - s3	backwards c
• s1 - s2 - s3 - s5	small round loop

among others. Assume the traversal has reached singularity s1 and the (s1, s2) edge is next to be tested and assume the exemplar feature tree given in Figure 5.5. (s1, s2) represents a JS. There is at least one feature that begins with a JS so the local traversal can continue. (s2, s3) represents a HM. There is a path in the exemplar feature tree for an HM so the local traversal continues. (s3, s4) is a failure because no path exists in the exemplar feature tree for an IM next. However, (s3, s6) represents an FM which is present and followed by (s6, s7) representing a DS. If the traversal reaches this node in the exemplar feature tree, then a "c" feature has been discovered.

5.4 The Feature Graph

The feature graph is an extension of the singularity graph. To review, the singularity graph contains a set of nodes, which represent singularities, and edges, which represent connections between singularities. The singularity nodes are labeled with:

- the (x, y) coordinates representing the actual location of the pixel to which this singularity is associated, within the skeleton; and
- the type of the singularity—intersection singularity, end singularity, and flow-through singularity.

Each edge is labeled with an (r, θ) which represents a vector length and direction from the source singularity to the destination singularity. Note that a singularity graph is an undirected graph and any edge (s_i, s_j) in the graph represents an edge from s_i to s_j and from s_j to s_i.

In the feature graph, every feature discovered is simply added to the singularity graph as a new singularity. This "feature" singularity node is labeled with the (x, y) coordinates of the relative center of the feature with respect to the singularities of which it consists. It is also labeled with a feature type—long narrow down loop, long narrow up loop, round loop, small round loop, c-type, backwards c-type, hump, or cup. As part of adding a feature node, edges are added so that any singularity (or feature) adjacent to any component of the new feature is connected to the new feature as well, with the exception of any singularities that are contained in the new feature. These new edges are labeled in the same fashion as before, with an (r, θ) which represents a vector from the source singularity/feature of the edge to the destination singularity/feature. However, there is one additional edge label, an edge-type label. There are four types of edge labels in a feature graph:

- a "regular" edge—this represents an edge where at least one end-point is a simple singularity (not one of the features);
- an "adjacent" edge—this represents an invisible edge between two features such that the two features are adjacent (each feature contains some singularities in the other feature but one feature does not totally contain the other feature);
- a "subsume" edge—this represents an invisible edge between two features such that the source feature contains all the singularities and edges in the destination feature; and
- a "subsumed" edge—this represents an invisible edge between two features such that all singularities of the source feature are also part of the destination feature.

For example, consider an example cursive letter "o" in Figure 5.7.a. The singularity graph might be drawn as in Figure 5.10.b where the circles represent

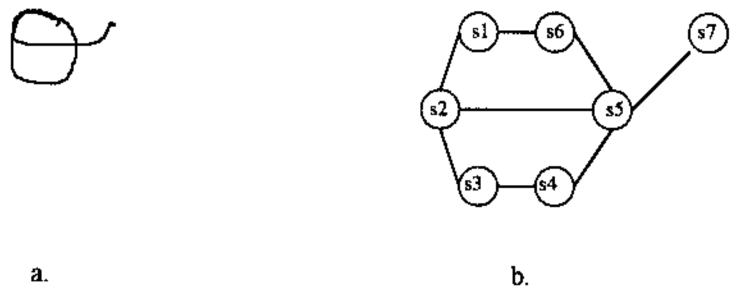


Figure 5.7 Feature Graph Example

singularities. The singularity graph would have nodes N= {s1, s2, s3, s4, s5, s6, s7}. The undirected edges would be {(s1, s2), (s2, s3), (s3, s4), (s4, s5), (s5, s6), (s1, s6), (s2, s5), (s5, s8)}. Features discovered include:

• s1-s2-s5-s6,	cup (f1), hump(f2), c(f3), back-c(f4), loop(f5)
• s2-s3-s4-s5,	cup(f6), hump(f7), c(f8), back-c(f9), loop(f10)
• s1-s2-s3-s4-s5-s6,	cup(f11), hump(f12), loop(f13)
• s1-s6-s5-s4-s3.	back-c(f14)
• s6-s1-s2-s3-s4	c(f15)

So nodes in the feature graph would be N = {s1...s7, f1...f15} where si...sj or fi...fj signifies all singularities or features numbered between i and j. Edges added would be:

- regular—(s3, f1...f5), (s4, f1...f5), (s1, f6...f10), (s6, f6...f10), (s7, f1...f15)
- adjacent—(f1...f5, f6...f10)
- subsume—(f11, f1..f10), (f11, f12...f15), (f12, f1...f11), (f12, f13...f15), (f13, f1...f12), (f13, f14...f15)
- subsumed—(f1...f10, f11), (f12...f15, f11), (f1...f11, f12), (f13...f15, f12), (f1...f12, f13), (f14...f15, f13)

When loops are involved, lots of new features are added. This is especially true when one loop subsumes another as in the above example. In practice, when features are discovered in a singularity graph and a feature graph is created, if a large sized original line drawing containing a word with many cursive characters is encountered, the feature graph can have as many as 300 to 500 feature/singularities. This means that the feature graph connection matrix contains 500x500 entries. This is quite a large array considering the labeling involved. In general, however, the matrix involved is very sparse. Even though the current implementation did not do so, this matrix could have been implemented using a sparse matrix data structure [Horowitz and Sahni, 1983] to minimize storage requirements.

5.5 Analysis and Comments

In the original proposal for this research, it was planned to use a variety of FI to perform this feature extraction. However, after the creation of the singularity graph, the design of the feature recognizer utilizes patterns that make up features which are very short and absolute. FI technology is badly under-utilized when the patterns are short and absolute. Also, one of the key advantages of FI is that FI can provide more "fuzzy" type recognition in that the residual provides a measurement of how close the new pattern is to the pattern trained into the ruling. This simple pattern matching requires absolute matching and does not require the fuzzy matching strong point of FI. For this reason, the exemplar feature tree directed search mechanism was used for the advantage it provides which is the speed of execution.

Nonetheless, the feature recognition phase quickly and easily finds all features present in the singularity graph that match features stored in the exemplar feature tree. The feature graph that is created by the feature extraction phase provides a flexible environment for the recognition of characters in the next phase.

CHAPTER VI

CHARACTER EXTRACTION, WORD SEGMENTATION, AND WORD CLASSIFICATION

6.1 Background Discussion

The next phase after thinning, vectorization, and feature extraction deals with three important topics: character extraction, word segmentation, and word classification, each of which is considered a worthy research area in the literature as discussed in Chapter II. This research treats them together as they are closely related and they work together to produce the desired word classifications.

The output from the feature extraction phase is a list of feature graphs where the contents of each graph is as discussed in Section 5.5. There is one feature graph for each line drawing assumed to contain a word or partial word to recognize. The construction of the feature graph involved

- thinning the line drawing into a skeleton,
- vectorizing the skeleton into a singularity graph, and
- extracting features from the singularity graph and producing the feature graph.

Character extraction involves examining a feature graph and discovering characters and character constructs in the graph. A character or character construct is a special feature that may, in general, be much more complicated than one of the simple features that currently exists in the feature graph.

Because of the extra complexity, the same methodology used to extract features from the singularity graph can not be used to extract characters from the feature graph. The process of extracting features from a singularity graph involved examining a feature tree containing exemplars for all possible features. An exact match of singularity patterns in the feature tree with singularity patterns in the input singularity graph was required. This can be done because the number of singularities and edges in any feature is small.

On the other hand, the number of singularities and edges in a character can be very large and the chances of an exact match are almost nil. For this reason, feature graphs are constructed and a pattern matching methodology that can produce an approximate match is used. An FI ruling base was constructed, with one ruling for each exemplar character construct trained into the system. The FI rulings are used via FI following to examine the feature graph for approximate matches with the exemplars represented by FI rulings in the FI ruling base.

The phase of word segmentation involves examining the list of feature graphs for likely combinations that might be segmented together to make a word. This is necessary because words are not always totally connected line drawings, as defined in the research specifications presented in Chapter I. Although it is planned to relax this constraint in the near future, in the current form of this research, cursive script words must be connected with only a couple of exceptions. The exceptions involve dots over letters, punctuation, and capital letters which begin words. If a capital letter is extracted and is disconnected, it is assumed to begin the word to its right or be a word in its own right.

The knowledge of which feature graphs represent capital letters and which represent short words is not available at the time the character extraction process begins. This requires that the word segmentation, even with the constraints, must at least wait until after an initial pass at character extraction takes place and generally must wait longer.

The word classification phase involves examining the list of extracted characters and determining which word in the vocabulary is the most likely match. When a character extraction is made, the extracted character and a confidence measurement (representing the expected correctness of the extraction) is placed in a data structure known as a letter graph, described in Section 6.5.2. The word classifier examines the most likely of the characters in the letter graph with respect to the confidence measurements and the characters in the words of a lexicon query.

The character extraction, word segmentation, and word classification phases are performed in an iterative fashion until at some point in time the most likely match is discovered. Figure 6.1 shows a graphical view of each phase in the process.

6.2 Character Extraction

The character extraction phase, like the vectorization phase, involves the use of FI to assist in the extraction. For each character construct that can be recognized, an FI ruling is prepared. During the training process, several exemplars for each character construct are FI factored to create several FI rulings. Each FI ruling recognizes one of those many exemplars. Also, any time during the use of the system, if a user desires a new

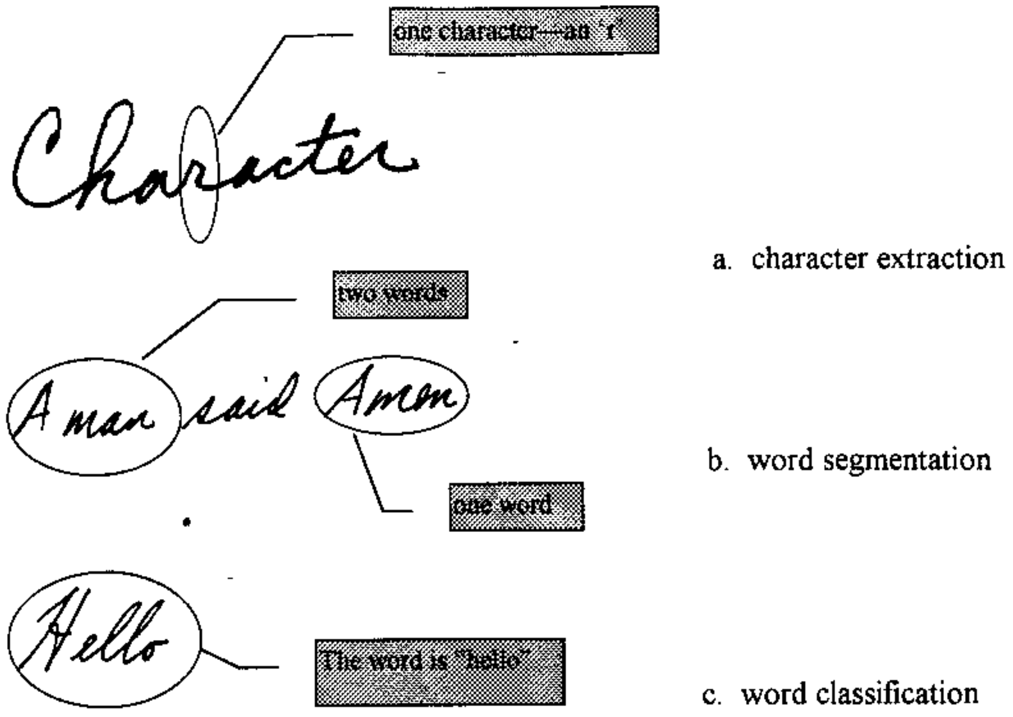


Figure 6.1. A Graphical View of Chapter VI Phases
The processes involved with character extraction, word segmentation, and word classification

exemplar letter construct, an FI ruling can be prepared at that time and added to the FI ruling data base. This process is known as adaptation.

Each FI ruling is created by FI factoring the feature graph representing an exemplar character construct after the feature extraction phase. The feature graph, as described in Chapter V, includes both regular singularities and features as nodes. From this point, however, this paper will refer to both as features when there is no ambiguity.

The FI ruling, therefore, involves features, as well as directional indicators representing edges.

Consider the example of the singularity graph for a lower case letter q in Figure 6.2.a. Assuming that the singularity graph is visually drawn in Figure 6.2.c, then the feature graph would consist of Nodes = {s1..s9, f1..f5} where:

s1-s2-s3-s4 = f1 (c-type) = f2 (back-c)
= f3 (cup) = f4 (hump)
= f5 (round loop)

with edges ={(s1, s2) - EM, (s1, s3) - GM, (s3, s4) - DS, (s2, s4) - BM, (s2, s5) - GL, (s5,s6) - EM, (s6, s7) - BS, (s7, s8) - LM, (s8, s9) - CM} unioned with {(s5, f1...f5) - all

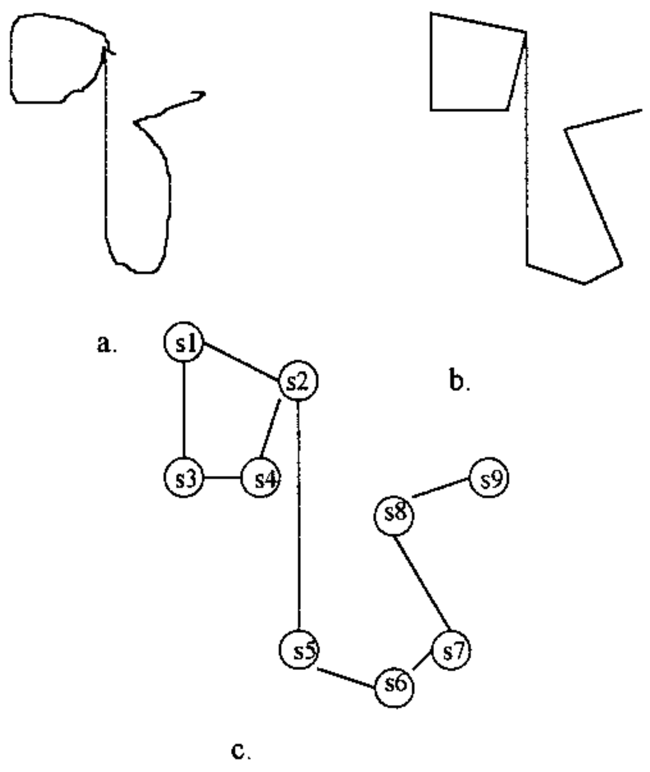
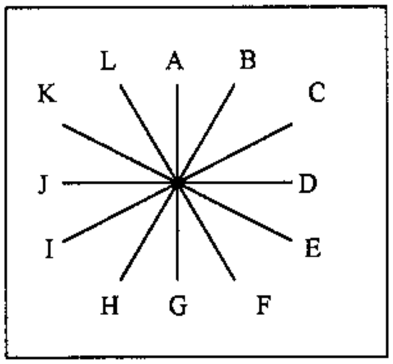


Figure 6.2. Vectorized Singularity Graph for a lower case 'q'.

LL's} unioned with the feature to feature edges {(f1, f2...f5), ,(f2, f3...f5), (f3, f4...f5), (f4, f5)}. Edge directions are taken from the following diagram:



with lengths specified as S(hort), M(edium), L(ong), and Z(ero Length). In this case, each feature to feature edge has a zero length and therefore the directions do not matter. This is not true in general. The feature to feature nodes are set up so that every edge with the loop feature f5 as the destination is a subsumed edge and each edge with the loop feature f5 as the source is a subsume edge. All other feature to feature edges are subsumed edges. Note that the edge between the singularities s1 and s2 is an undirected edge. The directional pointer and length is given as EM, direction E length M. If the other side of this edge is considered (s2, s1) then the reverse directional pointer would be KM. A similar situation exists for all other edges with non-zero length.

6.2.1 Creating the FI ruling base

If the above 'q' was to be used as an exemplar, then the factorization would start at s5, because it is the feature in the feature graph with the largest degree (or the one connected to the largest number of non-zero length edges). The input to the FI factorization would be the string: *, i, AL, i, KM, f, GM, f, DS, f, BM, *, i, LL, rl, *, i, LL, cup, *, i, LL, c, *, f, LL, back-c, *, i, LL, cup, *, i, LL, hump, *, i, EM, f, BS, f, LM, f, CM, f, *. Note, s5 in the singularity graph was a flow-through singularity as it is only connected to two other singularities. But in the feature graph, s5 is an intersection feature because it is connected to all the non-intersection features which means that it has a degree greater than 2.

The '*' is the end-of-path marker. Each time it is encountered, it signals return to the intersection feature immediately after the previous '*' and resume traversing. In this case, the 'i' after each '*' represents s5 which is the initial intersection feature.

A standard FI factorization is performed with one exception. It is assumed that there can be many rules *i → X, if i is the start feature. For example, rule *i → AL is one rule, *i → LL is another rule, *i → EM makes the third rule with antecedent *i. This is illegal in FI in general, but is very convenient here and is facilitated by allowing one symbol look ahead if the symbol to the right of the * is the start feature (s5 in the above case).

Each exemplar is FI factored as described above. An FI ruling is created for each exemplar. These FI rulings are stored in an efficiently accessible data structure with various key indices. The main key index includes

- the type of the starting feature,
- the degree of the starting feature, and
- the directional indicators and length for each edge out of the start feature.

Other indices have key fields including: the character construct that the ruling represents, the ruling start feature, and the identification of the author of the script. Having an index key containing the author of the exemplars allows an individualized ruling base. These indices, along with combinations, are used to cut down the search space during the extraction process.

Fields in the record containing the FI ruling contain the FI ruling itself, the author of the script, the character that the FI ruling recognizes, a copy of the input specifying the graph walk, and a vector of rule fire counts—one for each rule in the ruling. The rule fire counts specify how many times each rule in the FI ruling fired during the FI factorization of the exemplar. These counts will be used later in a calculation to determine the "closeness to a match" indicator for the FI ruling.

The fact that there are many different rulings that represent each character construct is a change from the flavor of other research projects that use structure and topological information to represent the internal form of a pattern to extract. In those projects, a single ""grammar" was prepared and the input was parsed according to the grammar. In this research, there are several possible FI rulings for each character construct to be extracted and rulings can be added at any time via adaptation. There are two reasons that this is possible. The first reason is that during the character extraction process, the search space of FI rulings is severely restricted, as described in the Section 6.5. The second reason is that rulings are quite small and the FI followings are localized in scope. This means that even if there are many FI rulings in the search space, the processing time is still small.

6.2.2 Performing Character Extraction

To extract characters and character constructs from an input feature graph, a walk is performed over the feature graph and an attempt is made to extract a character at each feature in the feature graph. A key is prepared containing the feature type, the degree of the feature, and the directional indicators for each edge incident with the start feature. The directional indicators are placed in ascending order within the key. A lookup is done in the FI ruling base for each FI ruling with that key. Generally a very small number of rulings are obtained. A localized FI following is performed using the copy of the exemplar input graph walk to direct the FI following in the input feature graph.

During FI following, the mechanism which provides input examines the input graph walk provided with the exemplar, and if possible, chooses a similar path in the input feature graph. If a path in the feature graph cannot be found, then input items from the feature graph are skipped until the '*' indicator is found. The traversal backtracks to an intersection feature and continues. Each time a rule fires, the count for that rule is incremented.

The rule fire counts are calculated during FI following of the local area in the feature graph. These rule fire counts are compared against the exemplar rule fire counts provided with the exemplar FI ruling that was used by the FI following. A distance is calculated between the input feature graph and the character represented by the ruling. The distance D_j^E in the following:

$$D_j^E = \sqrt{\sum_{i=1}^{N} w_i \left(R_{ji}^B - R_i^I\right)^2}$$

represents a weighted Euclidean distance between the rule fire count vector of the exemplar and the rule fire count vector of the localized FI following. The index j represents which ruling in the FI ruling base was used to perform FI following. The index i represents the number of the rule within the ruling. So, R_{ji}^B represents the rule fire count for rule i in ruling for exemplar j of the FI ruling base. R_i^I represents the rule fire count for rule i in the FI following of the input feature graph. The value w_i is the weight applied to rule i as some rules may be more reliable than others; i. e., where the consequent of the FI rule is a loop. N is the number of rules in the ruling. The exemplar character or character construct will be recognized if the distance is lower than a threshold for the FI ruling (which is stored along with the FI ruling in the FI ruling base).

6.3 Word Segmentation

In the general problem of cursive script recognition where both connected and disconnected script is allowable, word segmentation is a difficult problem. In these cases, the breaks between line drawings do not necessarily indicate a break between words. The problem is analogous to connected speech recognition where, in general, there may be no breaks between words and a break may represent a pause while saying a word. Also, if words may be disconnected, in many instances, characters may be disconnected. Another problem exists where there are no guidelines provided to the author of the script and the words are not written in a straight line. In this case and in other cases, it is difficult to segment the input into words on different lines as lines and even words may bend and turn unexpectedly. Another problem is apparent when the case where a descending loop on one line may intersect an ascending loop on the line below is considered. The solution of the general word segmentation problem is not within the scope of this research.

As specified in Chapter I, the word segmentation problem is limited to allowing for dots over letters, disconnected capital letters, and reasonable punctuation. The general rule is that a disconnected capital letter begins the word represented by the line drawing to its right. Even with these constraints, there are problems. A capital 'A' can begin a sentence as the word A, or the beginning of the word Another, or even some capitalized word not at the beginning of a sentence. In this case, contextual heuristics can be used to help in the word segmentation.

Even with the constraints placed on the input, the word segmentation phase must wait until after the first pass at character extraction. If a dot is discovered, it can be added to the singularity graph representing the line drawing physically placed directly below it on the page, if it is "close enough". If not, and there is possible punctuation near and above it on the page, it could be placed inside that singularity graph. There is always the possibility that the dot represents a period and belongs only to itself.

An extracted capital letter is assumed to begin the word to its right, for the time being, unless the extracted capital letter is an A or an I. In this case, heuristics must be used to help with word segmentation and these heuristics must wait until the character extraction process is almost over and word classification is taking place. Even in cases where all characters are recognizable, ambiguity may require even further contextual heuristics which is beyond the current scope of this research. As an example, consider the script for the name 'Ivan" where the 'I' part is disconnected from the drawing representing the 'van" part. This is possibly two words, 'I" and 'van". It might also be the single name "Ivan".

6.4 Word Classification

Word classification involves a context constrained search of the word space during character extraction. A lexicon exists in an efficiently accessible data structure with various key indices to assist in the lookup. For the purposes of this research, only the first five characters in the words of the lexicon are used in the indices.

There are several indices maintained with the lexicon. Each index corresponds to a possible situation that might exist after the initial pass at character extraction. Consider the following sequence of keys to indices into the lexicon where L_i represents a letter in position i.

L_1 L_2 L_3 L_4 L_5	- 5 letters known with confidence
* L_2 L_3 L_4 L_5	- 4 letters known with confidence where the wild card '*'
L_1 * L_3 L_4 L_5	represents an unknown sequence of characters
L_1 L_2 * L_4 L_5	
L_1 L_2 L_3 * L_5	
L_1 L_2 L_3 L_4 *	

The sequence continues where only three characters are known and there is one or possibly two corresponding wildcards in what is approximated to be the first five characters of the word.

These indices represent limited n-gram type information. In fact, the indices are referred to as split n-gram indices in that they represent not only characters that can go together sequentially, but they represent characters that can go together with wild card characters or character sequences separating them. A split n-gram index setup like this is very convenient and useful but the indices require a large amount of memory, especially if the indices were not limited to the first 5 characters of the words.

The extraction, segmentation, and word classification process work together in an iterative fashion. As mentioned before, after an initial extraction pass is made, then all characters extracted are placed in a letter graph and a lexicon lookup is performed. This lookup will determine a target set of words that are possible with the current characters. Then an attempt is made to construct each word in the target list by using letters in the letter graph and by doing further character extractions from the feature graph. Contextual knowledge says that, if any word is selected, one of the words from the lexicon search list must be the choice. As new characters are extracted, the target list of words is narrowed. This process continues in an iterative fashion until the feature graph is classified as a word from the lexicon, or is rejected.

6.5 Overall Methodology

When a new subject image—assumed to contain words to classify—is encountered, all relevant preprocessing is performed and the end result of all the pre-processing is a feature graph. This feature graph contains features, including regular singularities, and features. Regular singularities include intersection singularities, end singularities, and flow-through singularities. Features include loops, c , back-c, cup, and a hump feature.

Then in an iterative fashion, the character extraction, word segmentation, and word classification phases are performed. As characters are extracted, new word segmentation and classification is attempted. If a word is not predicted with a strong enough confidence, the information discovered in the attempted classification is used to further constrain the search space for the next pass at character extraction.

6.5.1 Character Extraction

The algorithm of the initial pass of the character extraction process is given in Figure 6.3. As the process continues, each feature in the feature graph is checked to see if a character begins at that feature. During the process, for any particular feature S and edge (directional indicator) directed away from S, a query is performed returning all FI rulings in the FI ruling base that begin with a feature that has the same feature type as S, has the same degree, and is followed by the same directional indicator list.

The record containing each of the FI rulings found in the query also contains the graph walk which was made during the factoring of the exemplar. This graph walk is used to control an FI following in the input feature graph where the next input symbol to the following is suggested by the ruling graph walk.

As an example, assume there is a rule: AM f BL i $\rightarrow$ DM and the current state of the FI following has AM f BL i in the FI shift register. Also assume that the current feature is an intersection feature and there are two possible directional indicators that

```
for each feature, S, in the feature graph
    for each edge directed away from feature S
        query the FI ruling data base for each ruling that begins
                with a feature of the same type as the current
                feature, the same degree and
                 and one which has the same following directional
                indicator list (one index contains this ordering)
        for each FI ruling obtained in the above query
                using a localized directed walk of the feature graph as input,
                        perform an FI following trying to match with high
                        confidence the "area" around this feature as the
                        character construct represented by this FI ruling
                if a match is discovered, insert the character, with its
                        confidence number into the letter graph for this
                        feature graph in its approximate position.
        after all the character candidates have been extracted, decrease the
                confidence number of those candidates which may be subsumed
                by others (for example a lower case script L might be subsumed
                by a lower case D, or a lower case B, or a lower case K.)
```

Figure 6.3. Character Extraction Algorithm

might come next during the traversal. One of the indicators is an AL and the other is a DM. The graph walk would specify the selection of DM and the FI following process continues. The process continues until this exemplar is recognized at this feature—or not recognized.

Each ruling returned from the query is used to perform a similar directed following. The rule fire count distance returned from each represents a confidence measure concerning how close the localized area around the starting feature in the input feature graph came to matching the pattern trained into the ruling, or how close it came to matching the character represented by the ruling.

When it is decided that a character construct has actually been recognized, then a marker is inserted into a letter graph at a position approximating its actual position in the line drawing. At any place in the letter graph, there may be a wild card, one, or several characters. These characters are used as likely candidates during word classification.

6.5.2 The Letter Graph

Much of the past work performed in the character extraction area in the literature involved unambiguous character segmentation of a connected word [Srihari and Bozinovic, 1987]. If characters in connected words can be unambiguously segmented, then the problem of character extraction from connected script can be handled much like disconnected handprinted text where the types of characters recognized are script characters instead of print characters. However, in unconstrained script, unambiguous segmentation is not very realistic. It requires various forms of normalization to map the input into a form that makes character segmentation feasible. At best, it is error prone and if writers do not take care, it is infeasible.

Since unambiguous segmentation is not a real possibility, then some type of scheme allowing ambiguous segmentation is required. What was selected here is a letter graph concept [Peleg, 1979], [Hayes, 1980], [Higgins and Whitrow, 1985], and [Ford and Higgins, 1990]. For example, consider the word *dip*. The cursive form of *dip* might be considered the word *clip* as the ‘d’ could be interpreted ‘cl’. A simple letter graph representing the situation is given in Figure 6.4.

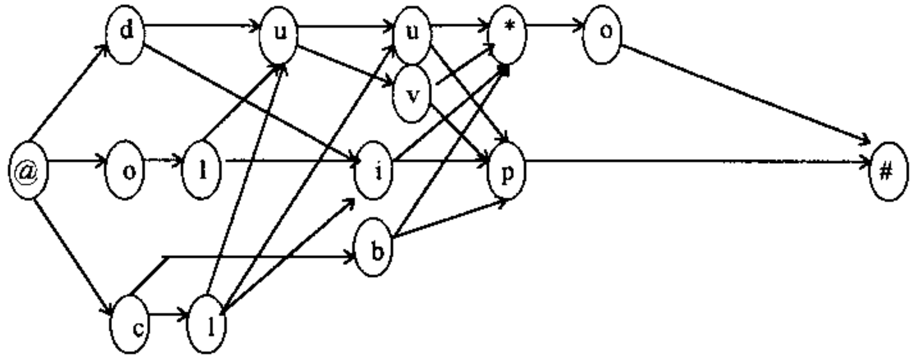


Figure 6.4. Example Letter Graph.
Letter graph for the word *dip* as it may have been written cursively

It is obvious from looking at the example graph for so simple a word, that there is an abundance of ambiguity. Section 6.5.4 discusses reducing the letter graph to get rid of impossible combinations of letters using an n-gram and split n-gram lookup technique.

6.5.3 Word Segmentation

The word segmentation phase involves a simple heuristic driven scheme where dots, comma shapes, straight vertical strokes, and the upper part of a question mark are combined, or not combined, to create predicted punctuation and where dots, capital letters, and a large feature graph are combined to create predicted words.

The creation of punctuation involves only the placement of the punctuation type features on the subject image page. A dot thins down to a very short skeleton which vectorizes to a very small singularity graph with two singularities and one edge. A comma preprocesses down to a feature graph almost as small as a dot, or with two edges each about the length of a dot edge.

It is the word segmentation phase that attempts to classify the punctuation. The punctuation includes periods, commas, semi-colons, colons, exclamation marks, question marks, apostrophes, and double quotes. The heuristics used involve mainly the spatial layout of the simple singularity graphs across the page.

The dot symbol involves more heuristics in this phase than any other symbol. It is part of a colon, semi-colon, question mark, exclamation mark, and sometimes part of an apostrophe and double quote as writers get in a hurry. A dot by itself can be a period or a dot over some character in a word depending upon its placement.

Segmenting words in this research also involves only very simple heuristics. After a capital letter is extracted with a high confidence, it is added to the letter graph of the feature graph to its right on the subject image page (if any, as it may be at the end of the line). If a high confidence level word recognition happens, then the capital letter is assumed to be segmented properly. If a rejection or low confidence level recognition happens and if the capital letter is an I or an A, then the I or A is classified and the classification process for the feature graph to the right is restarted. If the capital letter is something other than an I or an A, then the rejection or low confidence level recognition stands.

6.5.4 Word Classification

This phase involves searching the letter graph which is prepared by the character extraction phase and the word segmentation phase. In the first pass of character extraction, all of the high confidence characters should be extracted and placed in this letter graph. Using this high confidence set of characters and approximate positions, an index is chosen into the lexicon and a lookup is performed. A target set of words is selected. It is from this target set of words that the most likely candidates for a successful classification should come.

If the target set of words is empty, then a problem exists. If it is not possible to perform a new lookup or this lookup returns no target words, then a rejection is a possibility. At this point the system relaxes the FI following constraints somewhat and tries all the original FI rulings again. If the list of target words is empty still, then a rejection occurs.

When the first character extraction pass is complete, the letter graph may have a list of high confidence characters as well as other letters with lower confidence values. The original list of target words is created by using only the highest confidence characters in the letter graph and performing a lookup with those characters in their respective positions. For example, if the letter graph's highest confidence characters are a 'j' in character 1 of the word, an 'a' in character 2, a wild card, and a 'g' in the next character , then the key 'jag" would be used in the L_1 L_2 * L_3 * lookup index. This would return all words in the lexicon with those three letters in character positions 1, 2, and 4 or 5 of the word.

In general, at least three characters are desired to perform the lookup. However, in many cases, only one or two characters have a confidence value above the threshold required. If this occurs, then a larger number of words might be selected as target words. If no characters have a confidence value above the threshold, then the threshold is lowered.

When the set of target words contains more than one word, then the problem becomes selecting the correct target word. Each target word contains the high confidence characters in the specified letter graph positions. For any target word, the remaining characters might possibly be already in the letter graph as lower confidence characters. Each target word is checked by attempting to construct the word from the characters in the letter graph. If a target word contains a letter not in the letter graph, then a short character extraction pass is performed on the feature graph with the FI rulings selected as described earlier with the exception that only FI rulings representing the missing letter may be selected. These same FI rulings had been tried in earlier passes, and the recognition had failed. So, in this pass, the FI following "match" criteria is relaxed to allow for extraction of characters that do not match with a confidence value as high as originally desired. If the missing letter is found at one or more places in the feature graph by this latest character extraction, then this character and its confidence value is placed into the letter graph for later use.

This latest character extraction short pass was made during the process of attempting to construct a target word from the list of characters in the letter graph. If the new pass added the required character in the proper place of the letter graph, then successful construction can continue. If not, then an attempt is made to extract the other missing characters from this target word anyway. This is done because in the final analysis, it is possible that none of the target words can be completely constructed from the letter graph and extra extraction attempts. However, if one of the target words only has one wild card space or two wild card spaces for a long word, it might still be the best guess.

After a construction is attempted for each of the target words, hopefully only one target word is fully constructed. Many times, however, this is not the case. Generally, at least two target words are almost completely constructed. They are ordered as first choice, second choice, etc. by the sum of the confidence values of each of the characters extracted.

The confidence values for characters in the letter graph are calculated during the character extraction. The character extraction phase involved performing FI following on localized areas of the feature graph. The FI rule fire count length from the FI following is the "closeness to a match" indicator. Some characters extracted and placed into the letter graph will be a closer match than others. The lower the rule fire count distance, the higher the confidence value. Using the confidence values calculated in this fashion, there is no problem picking a first choice and second choice.

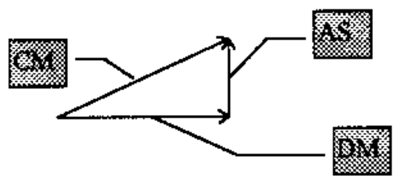
6.5.5 FI Constraint Relaxation

If the feature graph has an area in which no characters can be extracted with a high enough confidence value, there are two ways to manage the problem. First, the choice can be made to lower the confidence threshold. This choice is a viable choice when there are several alternate choices in the letter graph some having confidence values near the threshold.

However, if there are no reasonable choices and there is an area of the feature graph that should contain characters, then relaxation of the FI following constraints is used. Three mechanisms were experimented with to relax constraints in situations when an FI following comes up with no high confidence character extractions. They include:

1. accepting directional indicators that are up to 30° off,
2. using only the directional indicator in FI rulings that normally require a directional indicator/vector length combination, and
3. allowing the combination of two edges in a feature graph to match what corresponds to one edge in an exemplar FI ruling.

In a feature graph, features are connected with labeled edges that specify a length and a directional indicator. For example, two features might be connected by an CM edge. The C is a directional indicator according to the diagram given in Section 6.2. The first relaxation specified above involves allowing an expected CM to be matched with an BM or a DM. The second relaxation scheme specified above involves allowing an expected CM to be matched with a CS or a CL, where the length of the vector is not used (S = short, M = medium, and L = long). The third relaxation scheme tried involves allowing two vectors separated by a flow-through singularity in the input feature graph to match up with one vector in the exemplar FI ruling. For example, a DM vector connected to a flow-through singularity connected to an AS vector is allowed to match a CM vector as in



The third relaxation mechanism provides the cleanest type of relaxation. This is because the vectorization process provides small extra edges around intersection singularities. Matches determined after using this type of relaxation have a higher confidence value than do matches provided with other relaxation techniques.

In the implementation, the first two relaxation techniques mentioned above were employed together. This means that if a CM vector is expected by the exemplar ruling, then any B vector or any D vector would be accepted. This has a possible down side in that in many instances very deformed characters are accepted as matches for exemplars. Other times, incorrect matches are made.

The process of relaxing constraints on FI following attempting to extract characters is an area in which much research is still needed. For the purposes of the experiment, described in Chapter VII, this process resulted in many incorrect matches where a rejection would have been more appropriate. However, it is true that many correct matches were made that would have been rejections had it not been for an extra pass over the feature graph using a relaxed FI following.

6.6 Analysis of the Classification Scheme

The classification scheme described above entailed 3 phases that work together to provide a classification. The phases were character extraction, word segmentation, and word classification. The start of this process depends upon a previous phase that provides a singularity graph of the original subject line drawing.

Even though the process works pretty well, compared to similar projects, there is still much room for improvement. The feature graphs of the various line drawings across the page are segmented into punctuation and words. The punctuation is classified with heuristics built in to the code. This methodology of classifying punctuation is very error prone. It would be much better to generalize the word classification phase to classify punctuation "words" also. However, this would mean building the heuristics into that code.

It is apparent that when humans read disconnected script they employ large numbers of heuristics to perform their word classification as well as secondary context. If we are to build classifiers that classify script as well as humans can, the system will have to learn and manage heuristics. This means that heuristics can not be built into the code but will somehow have to be encoded into data for use by the classifier.

Even though much of the information is not currently used, much information is built into the current feature graphs that can be used as heuristics. For example, dots in the vicinity of an 'i' like or 'j' like character increase the likelihood that that character is indeed an 'i' or 'j' character. Even though dots are segmented to join the singularity graphs of words in the close vicinity below the dot, dots are not used in the current implementation for heuristic purposes

It is not clear at the present time how to encode these heuristics into a heuristics base that might allow the use of heuristics without having to encode the heuristics into the program code. If each of these human type heuristics were to be included inside the program code, a code nightmare would exist. For example, the heuristic "One way to distinguish between a script ‘q’ and a script ‘g’ is that in the descending loop, the direction of the vector from the middle of the loop to the loop intersection feature is generally greater than $\frac{\pi}{2}$ for a ‘q’ and less than $\frac{\pi}{2}$ for a ‘g’." There are many, many of these heuristics used by humans when they interpret script. It would be fairly easy to insert code into the program which will use the above heuristic to help in determining the difference between a g and a q. What is needed is a methodology to allow the heuristics to be encoded as data and have the system learn, or at least be trained, to use the heuristics.

As will be shown in the next chapter, even without the wide use of heuristics for classification, the results are encouraging.

CHAPTER VII

TRAINING AND EXPERIMENTATION

7.1 System Setup and Training

The initial steps to setting up the system involved training the character extractor and setting up the lexicon. The original line drawings for the original data base of FI rulings for characters and character constructs are from an old children's spelling book called **My Word Book** [Breed and Rogers, 1954]. The set of characters in Figure 7.1 had

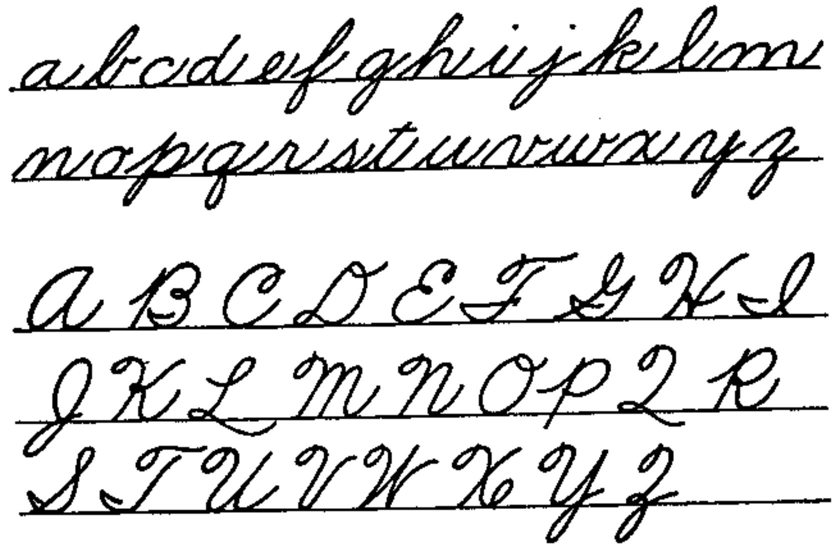


Figure 7.1. Original Training Alphabet. It was taken from **My Word Book** [Breed and Rogers, 1954].

the underline removed and was carefully touched up to connect the disconnected components of the capital ‘F’ and ‘T’. An image file was created. It was preprocessed with thinning, vectorization, and feature extraction. This produced a list of 54 feature graphs. That is two more than the number of letters as there were two dots over the lower case ‘i’ and ‘j’. The dots were ignored and 52 FI rulings were created by FI factoring the feature graphs; one FI ruling for each letter.

Several other sets of letters were drawn by this author and three other subjects. Each of these alphabets had FI rulings created after careful preprocessing. These letters were carefully and smoothly drawn to ensure that the FI rulings for the generic letters were high quality. The other subjects in this group, known as the model makers, did not participate in the experiment. Their function was simply to participate in the careful construction of the FI ruling base.

After each of the model makers had written the alphabet three times, they were instructed to write several words. The words were dissected into letters and the FI rulings for these character constructs were carefully constructed and placed into the FI ruling base. The motivation behind using letters extracted from words was our conjecture that people tend to construct their letters differently in words than when they write out the alphabet. This seems to be true for certain letters and it is assumed without proof that letters extracted directly from words more accurately reflect the letters used in the construction of words than do letters written in disconnected alphabets.

The FI rulings for these letters were placed in a generic FI ruling base. In this case, generic means that these rulings are part of the FI ruling space for any and all of the experiment participants. Each of the three experiment participants were required to also write a disjointed alphabet out three times and write out the same words that the model makers wrote three times. For each participant, an individual FI ruling data base was created that consisted of the character constructs (usually letters) from the letters of the disjointed alphabets that were produced and the letters extracted from the words written. In this way, each participant had his/her own individual FI ruling base reflecting his own personal handwriting style.

The lexicon was extracted from a 6,000 plus word data base. In this experiment, 1026 words were chosen and stored as described in Chapter VI into the efficiently accessible data structure. Two distinct groups of words were chosen. In one group, words were dissimilar to all other words. In the other group, similar words were chosen to test how well the similar words could be distinguished.

It is important to realize that the first real world uses of technology such as this will involve use in limited lexicon domains. For example, one example that is currently being heavily investigated by [Cohen, Hull, and Srihari, 1991] involves recognizing United States postal addresses. That particular paper mainly reports on an experiment involving off-line recognition of zip codes. Their results, even though seemingly somewhat unimpressive, make a very good contribution in the area of unconstrained word segmentation. They study much more general word segmentation requirements than is covered in this research. In their attempt to segment out the last line of an address and the zip code, they manage to correctly recognize or reject about 75% of the zip codes on the sets of envelopes in their experiment. Even in this heavily constrained arena, it is difficult to manage the zip code segmentation. They propose to use a lexicon of city and state names to assist in recognition of these zip codes. If a city name can be recognized, then the context of that city can assist in the recognition of the zip code.

If context can be used as heuristics, then recognition of words within special contexts can be greatly facilitated. For example, if, on some forms, a salesman must write a description of an item, knowing the small lexicon that might be involved in such a description will greatly assist in the recognition of the word in the description.

In this study, the context involved is totally contained in the lexicon. The small vocabulary of 1026 words in many cases might accurately reflect the total needs of certain applications. Ultimately, however, before a product such as this would be viable in many real world applications, a much larger lexicon of around 10,000 plus words would probably be required.

7.1.1 Experiment Design

After the training process was completed and each of the three experiment participants had completed their portion of the setup training, there was a generic set of FI rulings for each letter and an individual set of FI rulings for each letter and character construct for each of experiment participants. The fact that there are FI rulings for letters and character constructs for individuals involves the fact that, in some cases, a reasonable segmentation of words into specific characters was impossible and that certain constructs in words can generally be tied to a combination of letters. FI rulings were constructed for these character constructs and it was noticed later that sometimes these constructs greatly facilitate the recognition process.

For each of four phases of the experiment, four pages of script were selected according to the rules described in Chapter I. There were nine paragraphs, plenty of punctuation, where exactly 300 words of the lexicon were used, some more than once. The nine paragraphs were written twice by each experiment participant. In the first writing pass, all words contained only lower case characters and were totally connected. In the second writing pass, a large percentage of the words were required to begin with an upper case character and many of these, but not all, were disconnected.

Each of the script pages were originally totally blank and they were clipped on the top of other pages which provided visible guidelines which assisted the participants in writing the words the correct size and helped them write straight across the page.

After each phase of the experiment was completed, there were many words classified or rejected successfully; and, there were many words that were not classified correctly or correctly rejected. For each of the failed words, the letters and character constructs were manually extracted very carefully and new FI rulings for these new models were inserted into the individual FI ruling base for each participant. This was done because of the conjecture that the next time the same individuals wrote the words again, there would be a greater probability of successful classification. The fact that this training is very time consuming and quite pains-taking is a weakness of the system as it currently stands.

Each participant was required in each of four days to write the two sets of nine paragraphs. The classification results are given in Table 7.1 and Table 7.2. Table 7.1 gives the performance results for the test input involving only punctuation and lower case letters. Table 7.2 shows the corresponding results where the test involved both upper and lower case letters.

Participant 1	Trial 1		Trial 2		Trial 3		Trial 4	
Code	Count	%	Count	%	Count	%	Count	%
1	162	44.5	190	52.2	204	56.0	211	58.0
2	49	13.5	69	19.0	73	20.1	81	22.3
3	129	35.4	101	27.7	79	21.7	65	17.9
4	24	6.6	4	1.1	8	2.2	7	1.9
5	25	35.2	31	43.7	36	50.7	41	57.7
6	46	64.8	40	56.3	35	49.3	30	42.3
Participant 2	**Trial 1**		**Trial 2**		**Trial 3**		**Trial 4**	
Code	Count	%	Count	%	Count	%	Count	%
1	184	50.5	197	54.1	216	59.3	224	61.5
2	61	16.8	78	21.4	74	20.3	82	22.5
3	101	27.7	79	21.7	66	18.1	53	14.6
4	18	4.9	10	2.7	8	2.2	5	1.4
5	23	32.4	29	40.8	34	47.9	41	57.7
6	48	67.6	42	59.2	37	52.1	30	42.3
Participant 3	**Trial 1**		**Trial 2**		**Trial 3**		**Trial 4**	
Code	Count	%	Count	%	Count	%	Count	%
1	180	49.5	201	55.2	215	59.1	222	61.0
2	58	15.9	75	20.6	78	21.4	85	23.4
3	115	31.6	82	22.5	63	17.3	50	13.7
4	11	3.0	6	1.6	8	2.2	7	1.9
5	30	42.3	38	53.5	44	62.0	46	64.8
6	41	57.7	33	46.5	27	38.0	25	35.2

where code 1 $\rightarrow$ correctly classified first choice, code 2 $\rightarrow$ correctly classified second choice
code 3 $\rightarrow$ incorrectly classified from lexicon, code 4 $\rightarrow$ incorrectly rejected from lexicon
code 5 $\rightarrow$ correctly rejected from rejection list, code 6 $\rightarrow$ incorrectly classified from rejected list

Table 7.1. Results from Experiment / No Capital Letters

In the nine paragraphs that each participant was required to write, there were 364 words from the lexicon with some words used more than once. Three hundred words from the lexicon were used at least once. There were 71 words used that were not in the lexicon. Also there were 45 punctuation marks.

Participant 1	Trial 1		Trial 2		Trial 3		Trial 4	
Code	Count	%	Count	%	Count	%	Count	%
1	146	40.1	155	42.3	159	43.7	173	47.5
2	53	14.6	55	15.1	54	14.8	66	18.1
3	153	42.0	140	38.5	130	35.7	110	30.2
4	12	3.3	14	3.8	21	5.8	15	4.1
5	20	28.2	22	31.0	22	31.0	31	43.7
6	51	71.8	49	69.0	49	69.0	40	56.3
Participant 2	Trial 1		Trial 2		Trial 3		Trial 4	
Code	Count	%	Count	%	Count	%	Count	%
1	158	43.4	167	49.5	180	49.5	189	51.9
2	59	16.2	69	19.0	71	19.5	76	20.9
3	145	39.8	115	31.2	101	38.3	95	26.1
4	2	0.5	13	3.6	12	3.3	4	1.1
5	22	31.0	25	35.2	28	39.4	33	46.5
6	49	69.9	46	64.8	43	60.6	38	53.5
Participant 3	Trial 1		Trial 2		Trial 3		Trial 4	
Code	Count	%	Count	%	Count	%	Count	%
1	140	38.5	153	42.0	158	43.4	171	47.0
2	55	15.1	50	13.7	55	15.1	70	19.2
3	159	43.7	145	39.8	133	36.5	110	30.2
4	10	2.7	16	4.4	18	4.9	13	3.6
5	26	36.6	26	36.6	30	42.3	35	49.3
6	45	63.4	45	63.4	41	57.7	36	50.7

where code 1 → correctly classified first choice, code 2 → correctly classified second choice
code 3 → incorrectly classified from lexicon, code 4 → incorrectly rejected from lexicon
code 5 → correctly rejected from rejection list, code 6 → incorrectly classified from rejected list

Table 7.2. Results from Experiment / Capital Letters Included

7.2 Performance Evaluation

The results shown in Tables 7.1 and 7.2 describe the system performance during the summer of 1994. The performance results are not impressive by today's OCR standards. However, the performance is very near expectations. The initial intentions were to demonstrate that the system could be trained to perform better and better. This is ultimately what was shown in the experiment.

Compared to other projects of a similar nature, the requirements for this project involved a high level of difficulty as the system attempted to recognize punctuation, perform word segmentation for certain types of disconnected words, recognize capital letters, and adapt to the individual handwriting style of each experiment participant. The system was successful in each arena.

In the experiment where no capital letters and special word segmentation were involved, the correct classification rate started between 55% and 65% for correct classifications on the first or second choice. By the fourth attempt, the classification rate improved to over 80% for all participants. Each participant had a correct recognition rate of around 60% for the first choice in the fourth attempt. This is compared to between 44.5% and 50.5% in the first attempt.

In the part of the experiment where capital letters and special word segmentation was involved, the correct classification rates were not as good. However, in each case the classification rates improved in the later trials. The poor recognition rates for words in this experiment were caused by the fact that capital letters are much more difficult to recognize correctly as they involve much more detailed drawings. Also, the feature set chosen including loops and the c, back-c, u, and hump shaped features were more prevalent in lower case letters than in upper case letters. Using the methodology chosen for this research, successful classification at high percentages recognizing upper case cursive script will have to involve a richer set of features and a greater use of heuristics.

The system used only static heuristics in its attempt to recognize punctuation. It did not attempt to learn to recognize punctuation. The success rate for recognition of punctuation is given in Table 7.3. Recall that in the nine paragraphs that were written for each experiment, there were 45 punctuation marks. However, each time the participants had to write the same nine paragraphs twice, making 90 punctuation marks in all. In each

Participant 1	Trial 1	Trial 2	Trial 3	Trial 4.
Code	%	%	%	%.
1	73	78	71	72
2	23	18	24	22
3	4	4	5	6
Participant 2	Trial 1	Trial 2	Trial 3	Trial 4.
Code	%	%	%	%.
1	77	76	75	77
2	20	18	19	21
3	3	6	6	2
Participant 3	Trial 1	Trial 2	Trial 3	Trial 4.
Code	%	%	%	%.
1	80	80	83	84
2	18	16	14	12
3	2	4	3	4

where code 1 $\rightarrow$ correctly classified punctuation, code 2 $\rightarrow$ incorrectly classified punctuation
code 3 $\rightarrow$ incorrectly rejected punctuation

Table 7.3. Punctuation Recognition Results

trial in Table 7.3, the percentages demonstrate correctness or incorrectness of the recognition process for all 90 punctuation marks.

The success rates are fairly high especially for the one experiment participant who took extra care when writing the punctuation. There was little or no improvement from one trial to the next and the recognition rate was determined by how much care the participant took when he/she wrote the punctuation.

CHAPTER VIII

CONCLUDING REMARKS

8.1 Overview

Major advances take place almost daily in the field of pattern recognition. These advances will have far reaching effects on the world and society as the technology improves. Pattern recognition of some form or another will be the foundation of applications such as voice recognition, computer vision and object recognition, target detection for military applications, optical character recognition, on-line handprinted text recognition, and off-line cursive script recognition, among many others.

This research focused on the off-line cursive script recognition application. The problem is very large and difficult and there is much room for improvement in every aspect of the problem. Many different aspects of this problem were explored in pursuit of solutions to create a more practical and usable off-line cursive script recognizer than is currently available.

The scope of the project involved a complete solution to most aspects of the problem. Preprocessing was refined via a new thinning algorithm and a new FI based vectorization algorithm. Feature extraction was performed extracting features from the singularity graph of the line drawing instead of the line drawing itself. The feature graph was set up to provide a very expressive, flexible, and efficient data structure so all existing features of a singularity graph can be easily scanned and associated locally. A new andpowerful FI based character extraction mechanism was created and studied. Character extraction, word segmentation, and word classification were performed iteratively in light of the context of the lexicon using split n-gram indices to assist in word classification and search space reduction. The use of heuristics was formally employed and studied in the recognition of punctuation. Also, an adaptable system was designed so that the system could adapt to individual handwriting styles of experiment participants.

Another focus of this dissertation involved exploring how the pattern recognition technology known as Finite Induction could be employed in pursuit of applications of this nature. FI was a major contributor in two of the phases. FI technology was adapted for use and successfully employed in the line segmentation process and in the character extraction process. The major strong point of FI is that during FI following, the FI residual, the local residual density, and rule fire count distance provide a usable measure of closeness between the input source pattern and the pattern trained into the FI ruling with which the FI following was performed.

8.2 Achievements

This research has demonstrated that some off-line cursive script recognition applications are feasible within constraints. It employed a combination bottom-up and top-down approach to the word segmentation and word classification problem as character extraction was performed to assist in word segmentation and word classification; and, then the context provided by the word segmentation and word classification process assisted in an attempt to further extract characters. The experiment that was conducted demonstrated that with reasonable training of the system and reasonable restrictions placed upon the writer of the cursive script, successful hand written cursive script recognition is feasible and usable systems are within reach.

The specific contributions made by this research include:

- The Border Reduction Thinning Algorithm; The skeletons produced by the BRT thinning algorithm are very smooth and have very near the same shape as the original. They retain the same connectivity and very near the same end-points as the original. These skeletons are much more appropriate for the application of hand-printed text and hand written cursive script recognition than the other algorithms studied.

- The FI directed line segment approximation algorithm also referred to as the vectorization algorithm. Via utilization of the FI technology, this phase produces very high quality vectorized approximations of input skeletons. The quality of the singularity graph is such that the singularity graph may be used as the basis of the feature recognition process instead of the skeleton itself.

- The modification of the standard FI mechanism to allow for its usage in the vectorization algorithm. The input to the FI following was a recursive descent traversal over the skeleton and the recognition mechanism in the FI following is the local FI residual density instead of the residual length or the residual itself.

- A feature trie structure was used to contain a dictionary of feature descriptions. Use of this mechanism allowed easy and exhaustive extraction of features contained in a singularity graph as long as the feature was defined in the feature trie.

- The feature graph was created. The feature graph is an extremely flexible and expressive data structure that allows all features present in the singularity graph to be stored in such a manner that they can be easily scanned and associated locally to assist in character extraction.

- The FI directed character extraction mechanism. The algorithm examines the feature graph and at each feature, checks the entire FI ruling base for possible candidate characters that might start at that singularity.

- The modification of the standard FI mechanism to allow for its usage in the character extraction algorithm. The input to the FI following is a localized recursive descent traversal where the ruling itself predicts the next input symbol in the situation where the current input is an intersection singularity and there are two or more possibilities for the next input symbol.

- The lexicon or context directed word segmentation and word classification algorithm. The character extraction phase works together with the word segmentation and word classification phase in an iterative fashion where each phase provides information to the other phases generally in the form of reduced search spaces. Anytime new information is discovered, the search space in the lexicon is reduced and only possibilities from this reduced search space are pursued.

- Split n-gram indices were used to assist in word classification. These indices were used to assist in classifying words and reducing the search space when making secondary attempts at character extraction.

- An adaptation mechanism which allows the recognizer to adapt to a personal handwriting style. This feature was proven by experiment to increase the correct classification and rejection of words as the training progresses.

Each of the above mentioned contributions are important in their own right, but just as important is the fact that each was implemented and a system now exists that performs cursive script recognition. Experiments using the cursive script recognition system showed that the cursive script application is feasible even though a very difficult application. The system is undergoing revisions and testing and therefore improving on an almost daily basis.

8.3 Future Work

The immediate future research efforts will involve the creation of an automated learning facility for the character extraction phase. As it existed in the experiment, much pains-taking manual intervention was required to enable the newly discovered singularity graphs for the characters to be inserted into the FI ruling base as new FI rulings. An implementation of an automated learning facility is nearing completion.

An interactive Microsoft Windows™ based user interface for the system is also near completion. It will work together with the automated learning facility which also employs a similar windows based user interface.

Other improvements will involve:

- re-implementing the system on a more powerful computer to increase system performance;
- creating a variant of this system designed to recognize hand printed connected or disconnected text;
- refining the feature set to be of greater use during recognition of capital letters;
- increasing the size of the lexicon to a size that would be of interest to commercial applications;
- setting up a heuristics base or heuristics implementation methodology which will allow heuristics to play a greater role in the recognition process;
- using greater amounts of context to assist in word classification including grammatical context, positional context, and understanding the fact that different boxes on a form containing script might each require different lexicons for context;
- experiment with more general word segmentation to allow more generally disconnected script;

The impact that a high-quality off-line cursive script recognizer would have in various industries and applications might be very large. Research is ongoing and system performance and capabilities are improving almost daily. The goal is to soon have a product which is usable in real world applications.

APPENDIX

AN INTRODUCTION TO FINITE INDUCTIVE SEQUENCES (FI)

APPENDIX

AN INTRODUCTION TO FINITE INDUCTIVE SEQUENCES (FI)

A.1. Detailed Introduction

This section introduces the concepts associated with Finite Inductive Sequence Processing (FI) and explains the associated terminology and the capabilities. The FI process is a mathematical technique for dealing with large amounts of data which represent objects of interest. In comparison to the present techniques of processing data, statistical techniques tend to utilize too little of the data, while typical mathematical techniques tend to deal with too much of the data. FI is a technique which is a compromise between these two approaches. The FI premise is that the occurrence of a symbol in some data stream such as a pixel in an image is dependent upon some number of previous symbols (similar to a Markov Process), potentially, a very large number of such symbols. Because this number can be very large, FI provides a mechanism for reducing this number of *antecedent* symbols to an a priori number of symbols which can be very small, say four or five symbols. The advantage of this technique will be shown later in this section, but the impact is a considerable reduction in data storage requirements for the database of specific patterns of interest. Using this capability, FI also provides a very powerful technique for recognition of patterns of particular interest.

Not all representations or sequences of symbols are reducible in the FI sense. Random sequences of symbols or sequences of symbols that have very little differentiation between symbols are examples of nonreducible sequences or sequences whose reducibility is not of interest. For the most part, representations of all real objects are reducible in the FI sense. This reducibility is highly desirable because it allows automatic discarding of data that is superfluous, or data that makes no contribution in the understanding (representation) of a pattern of interest. FI provides an automatic technique to organize data in a format which is immediately useful for recognition.

The first concept important in the area of image recognition is to reduce the image or pattern to some representation which encapsulates the features of interest, preserves the geometry of the image, and provides a format which is suitable for easy manipulation. To illustrate this, consider a square as the image of interest. The following three alphabets encapsulate certain features of the square.

ALPHABET 1: Topological alphabet consisting of the number of intersections between the object and ten horizontal and ten vertical lines. The values for the object would then be:

1 2 2 2 2 2 2 2 2 2 1; 1 2 2 2 2 2 2 2 2 2 1

Note: if the box is larger than 9 pixels on a side, and if the first and last line fell on a box boundary, then the representation of the box is as given.

ALPHABET 2: Geometric representation would consist of the number of angles, the size of the angle, their arrangement, and the order and type of the sides. The values for this type of alphabet would be:

1 1 0 2 1 0 3 1 0 4 1 0

Note: for this alphabet, the 1, 2, 3, 4 indicate a right angle in the first quadrant, in the second quadrant, etc. The 1 0's indicate the length of the ray beginning from that angle and going to the next angle.

ALPHABET 3: Edge following representation would provide output from the follower as it moves around the edge. Such an alphabet would be:

1 1 1 1 1 1 1 1 1 1 7 7 7 7 7 7 7 7 7 7 5 5 5 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3 3 3

Note: here the symbols represent the direction of travel for the edge follower. There are eight directions that a 3x3 edge follower can move. The directions are 0° = 1, 45° = 2, 90° = 3, 135° = 4, 180° = 5, 225° = 6, 270° = 7, and 315° = 8.

The most difficult aspect of recognition using FI is the selection of the alphabet. As the above alphabets have shown, the two-dimensional image was represented as a one-dimensional string of symbols, and in the first alphabet, the representation could fit several distinct shapes. FI does not address the alphabetic representation, but the representation selected can be two-dimensional components such as small arrays, multiple strings; that is, input streams coming simultaneously from more than one source or any other selected arrangement. The process of pattern recognition after representation can then be summarized as shown by Figure A.1. In this figure, the unknown image (a rectangle of a certain size) is to be matched with the correct object in the space of known objects.

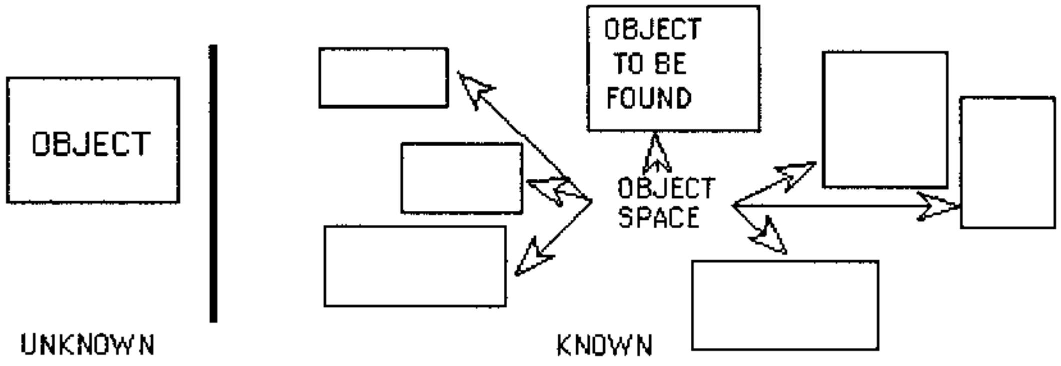


Figure A.1.
Process of Recognition: Mapping from Unknown to Known

The FI system provides the following capabilities:

- a method for representing the known objects in a concise form;
- a method for identifying the rank order of best matches from the space of possibilities;
- a recognition speed O(n) where n is the number of symbols representing the input object and not the size of the known object space;
- a concise pair of algorithms making up FI where the implementation in C requires 500 lines of code for both algorithms;
- a technique that is easily executed on a PC, even for complex patterns.

The FI process together with the creation of an alphabetic representation is shown in Figure A.2.

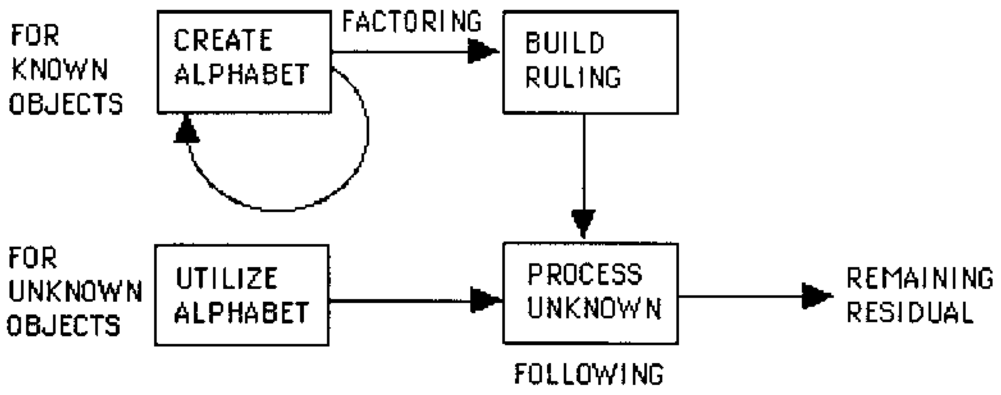


Figure A.2.
FI Process Representation

The important features shown in Figure A.2 are the two algorithms, FACTORING and FOLLOWING and their relationship. Associated with these two algorithms are two data structures: RULING and RESIDUAL. The RULING results from the application of FACTORING to the input strings representing the known patterns. The RESIDUAL results when the algorithm FOLLOWING is applied to the string representing the unknown patterns using the RULING. Figure A.3 portrays this second process of FOLLOWING applied to an unknown pattern, then creating a RESIDUAL.

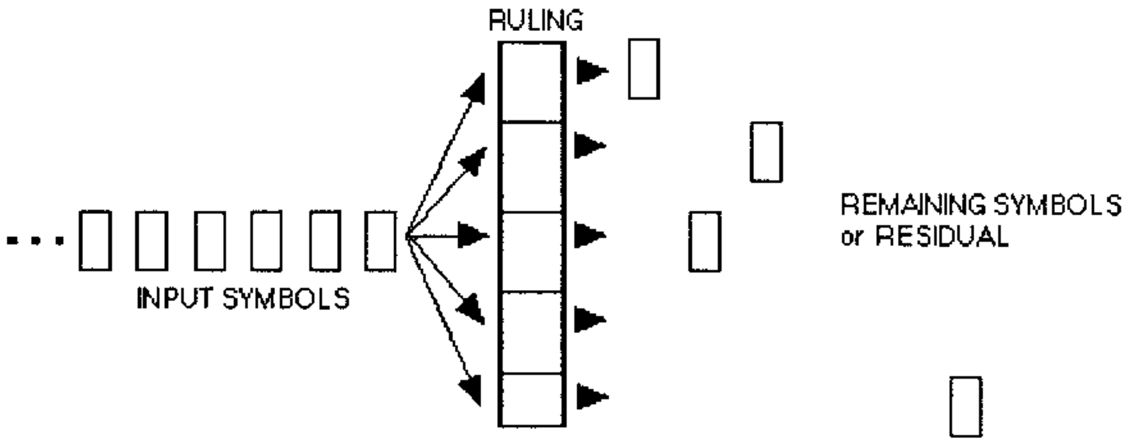


Figure A.3.
FOLLOWING Applied to Input String
to Produce a RESIDUAL Using a RULING

Figure A.3 portrays the relationship between the input string and the output or RESIDUAL. If the RESIDUAL and the input string are identical, then the RULING contains no knowledge of the patterns represented by the input string. If the RESIDUAL is empty, then the RULING contains a representation of the pattern represented by the input string. In Figure A.3, there is a relation between the input string and the RESIDUAL, as the RESIDUAL is a subset of the input string. A measure of closeness between the input string and the information contained in the RULING is the number of symbols contained in the RESIDUAL under two conditions: first, the RULING only contains data from one object; or second, if the RULING contains data from more than one object, the data which makes that object unique from all of the other objects is identifiable in the RULING.

A.2 The FI Definition

With the introduction given in Section A.1, we now provide a somewhat more formal definition of FI. Only one of the requisite theorems is stated, and the proof is not included.

Def—Let A be a finite set of symbols called the alphabet and let S be a sequence (called a string) of these symbols which may be finite or infinite. The sequence S is Finitely Inductive (FI) if the choice of a letter at any particular position depends (immediately) only upon the choices of letters at the previous n positions. The least such n is called the inductive base of the sequence S.

Def—Let S be an FI sequence, and let the pair (w,p) consist of a word w over the alphabet A and the letter p be in the alphabet such that (i) w occurs at least once as a substring (subsequence of contiguous entries) of the sequence; and (ii) wherever w occurs as a substring there is a succeeding entry and it is p.

Note: The pair (w,p) is normally written w —> p and is called an implicant; w is called the antecedent, and p is called the consequent.

Def—An implicant is in reduced form if its antecedent contains no proper terminal segment that is an antecedent of another implicant of the sequence.

These definitions are immediately generalizable to families of sequences. The pair (w,p) is an implicant of the family if its antecedent occurs at least once as a substring of one of the sequences. Whenever w occurs as a substring of one of the sequences, the following symbol will be p.

The following observations can be easily noted:

- Any finite sequence is FI;
- For any finitely inductive sequence, the inductive base is the maximal length of the antecedents in its reduced form implicants;
- Any periodic FI sequence has a period less than or equal to k^n, where k is the number of letters in the alphabet and n is the inductive base;
- If an FI sequence has inductive base n and an alphabet of k letters, then k^n is an upper bound for the number of its reduced form implicants.

Def—A table of implicants is a finite table of pairs of the form w —> p where w is a word over a given alphabet and p is a letter in the same alphabet.

A table of implicants satisfies the following conditions: (i) no antecedent of one entry in the table is a terminal segment of the antecedent of any other entry in the table; (ii) without losing generality we can suppose that every sequence has a special symbol called the start symbol, then the only way the start symbol may occur in the table is as the first symbol in an antecedent. The inductive base for such a table is the maximal length of the antecedents.

Def—Given an FI sequence S, there is a finite sequence of tables of implicants T_i called a RULING; such that for each i from 1 to L, T_i is called the table of implicants for level i.

In order to motivate understanding, an example of an FI sequence and the associated RULING is shown. The example will show how the sequence of tables (RULING) is constructed. The process of obtaining this RULING is called FACTORING, and this is the first of the two FI algorithms. The inductive base for the string shown in the example is four (as known from processing this example), and it will be reduced to two without the loss of data.

EXAMPLE: The alphabet will be the symbols A, B, C, D, and the string will be infinitely periodic to the right, each period being separated from the next by a colon (:).

AABABCABCD:AABABCABCD:...

Step 1 — Form table T_1 of the RULING from all Implicants whose Antecedents are two or less symbols (this can be any value, but since the inductive base of the string is four, a value less than four will produce a series of tables). Place all Consequents of nonselected Implicants in a new string called the RESIDUAL for the level 1 table T_1, preserving their relative order.

Level 1:	D -> A; DA -> A; AA -> B; BA -> B; CA -> B
Residual:	A CA CD:A CA CD:...

Step 2 — Apply the factoring strategy to the Residual of the previous level. Form a new table T_2 of Implicants and select from those all that meet the Antecedent length criteria (two or less). Form a RESIDUAL. If the RESIDUAL is empty or the factoring process terminates due to the inability to formulate new Implicants, end the process.

From the RESIDUAL string above we get:

D -> A; DAC -> A; A -> C; CAC -> D

Selecting those which meet the length criteria we have:

Level 2:	D -> A; A -> C
Residual:	A D: A D:...

Step 3 — Repeat step 2 except form the table T3:

Implicants:	D -> A; A -> D
Level 3:	D -> A; A -> D
Residual:	None

<u>Example Ruling Tables</u>

LEVEL 1	LEVEL 2	LEVEL 3
D -> AD -> A	D -> A	
DA -> A	A -> C	A -> D
AA -> B		
BA -> B		
CA -> B		

From the example there are several considerations that should be noted:

- The individual tables are combined into the RULING, and are indicated as levels in the RULING;
- The results of FACTORING any finite string will yield one of many RULINGs depending upon the inductive base chosen, and the method that is selected for forming the RESIDUAL (that is for 'pushing out' the symbols that will be removed in the next level);
- The RESIDUAL is empty after FACTORING every FI string, but when FACTORING a family of FI strings the process will terminate with a non-empty RESIDUAL associated with each string in the family;
- The inductive base for the example is 2-1-1, or simply 2.

We now present a theorem and one observation dealing with FI sequences. We have noted that every finite sequence is FI; that is, it is reducible to a collection of implicants. Not every finite sequence has a RULING consisting of a sequence of tables. For example, sequences which are pseudo-random have implicants whose antecedent lengths are all equal. This brings us to the observation:

Observation. If S is a sequence, finite or infinite, and the implicants of S do not have uniform antecedent length, then using the FACTORING technique, there is an associated RULING representing the sequence S, such that the inductive base of the RULING is uniformly less than a fixed a priori value for each table in the RULING.

Def—An autonomous RULING is a RULING which generates a sequence using a particular starting symbol of length 1.

Theorem. The inductive base of the sequence generated by an autonomous RULING is exponentially longer than the inductive base of each level of the RULING in the following sense: if the alphabet has k letters, the inductive base of all levels is bounded by b and the number of levels is L, then the inductive base of the generated sequence is potentially as long as $k^{(b-1)(L-1)}$.

The implication of the observation and theorem is in representing sequences derived from such things as images, there is a significant possibility for reduction in storage associated with the objects that are represented by the RULING resulting from the FACTORING process. If the number of implicants for a finite string are O(m) where the length of the string is m, then using implicants of four or five symbols is far more concise than using implicants of arbitrary order. This can be seen from the example. The original sequence consisted of ten symbols and the resulting RULING consists of nine implicants whose antecedent length is one or two. The original implicant length included several with lengths of four symbols.

The process of FOLLOWING utilizes the RULING(s) developed by the FACTORING algorithm. Essentially, the FOLLOWING process is one of applying the implicants in each level of the RULING to the incoming, unknown string and when a match is found between the antecedent in the RULING and a substring in the input string, then the consequent in the input string can be deleted. If each level proceeds in a sequential manner (this can be pipelined for speed), then the remaining symbols form the RESIDUAL. This RESIDUAL can then be used in the pattern recognition task. For example, suppose that there are several RULINGs and the unknown string is processed by each RULING. The RULING producing the smallest (in length) RESIDUAL will be the best representation of the unknown entity.

There are several types of FOLLOWING, and these are called BLIND, EXACT, and REPLACEMENT FOLLOWING. In BLIND FOLLOWING, the antecedents need only match and the consequent is deleted whether it matches or not. In EXACT FOLLOWING, the antecedents and the consequent must match before deletion can occur. In REPLACEMENT FOLLOWING, if the consequents do not match, then the input string consequent is converted to the consequent of the RULING consequent. This prevents the perturbation of a symbol propagating through the deletion process.

If a RULING is to contain more than a single object, then Figure A.4 depicts a technique that can be used for recognition. Figure A.4 shows a RULING containing three objects. The number four is for illustration purposes only, and in general the number would be far larger.

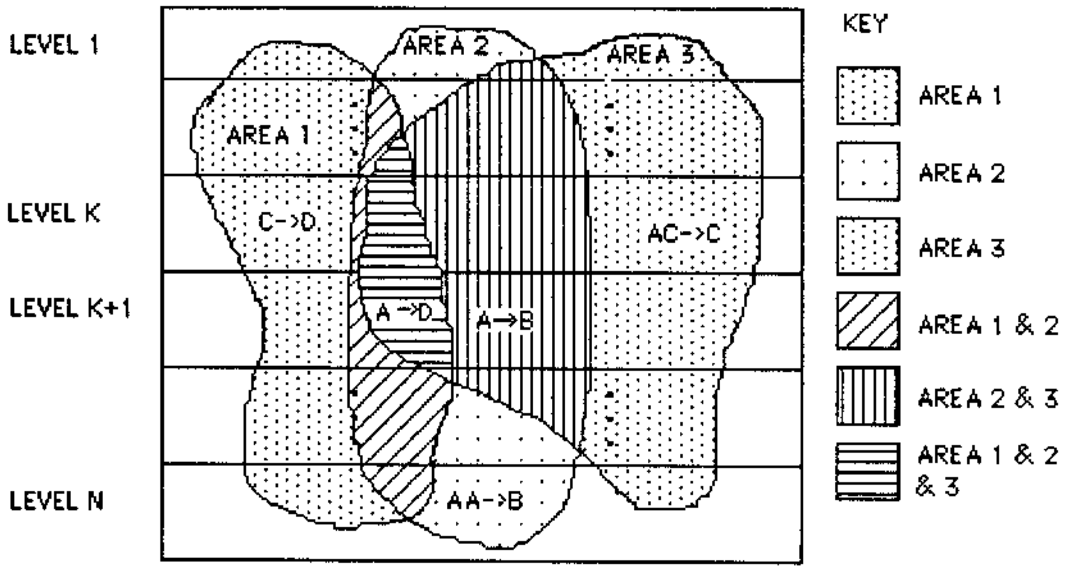


Figure A.4.
RULING Containing Implicants from Three Objects

If the KEY shown in Figure A.4 was a collection of counters instead of area indicators, and the utilization of any implicant matched during the FOLLOWING process would cause the appropriate counter to be incremented, then at the end of the FOLLOWING, the highest counters would indicate the appropriate match between the unknown and known objects. In reality, the use of the overlapped areas of the implicants could be discarded; however, it is possible that these counters could indicate a measure of strength between objects recognized. The implicants found in each area would have commonalties; that is, an implicant may belong to more than one known object. By adding an indicator to the implicant to denote the appropriate counter to increment, the counts can be obtained directly during the FOLLOWING process.

In considering the FOLLOWING process, we note that the performance time for FOLLOWING only depends upon the length of the incoming, unknown string and not upon the complexity of the RULING or the number of implicants contained in the RULING.

REFERENCES

Aizawa, K. and Nakamura, K., "Path Controlled Graph Grammars for Syntactic Pattern Recognition," in *Parallel Image Analysis and Processing*, Eds. K. Inoue, A. Nakamura, M. Nivat, A. Saoudi, and P. S. P. Wang, World Scientific, 1994. pp. 71-86.

Andrews, D. "PDA Companies: We've Only Just Begun," *Byte Magazine*, **19**, 5, May 1994, pg. 34.

Aoki, K. and Yamaya, Y., "Recognizer With Learning Mechanism for Hand-written Script English Words," *Proceedings of the 8th Int. Conf. Patt. Rec.*, Paris, 1986, pp. 690-692.

Aoki, K. and Yoshino, K., "Recognizer for Handwritten Script Words Using Syntactic Method," in *Computer Recognition and Human Production of Handwriting* , R. Plamondon, Suen, C. Y., and Simner, M. L. (eds.), World Scientific, 1989, pp. 5-18.

Arcelli, C. "Pattern thinning by Contour Tracing," *Computer Graphics and Image Processing*, **17**, 1981, pp. 130-144.

Arcelli, C. and Sanniti di Baja, G., "A Width Independent Fast Thinning Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-7**, 4, July, 1985, pp. 463-474.

Badie, K. and Siimura, M., "Machine Recognition of Roman Cursive Script," in *Proceedings of the Sixth International Conference on Pattern Recognition*, Munich, F. R. G., 1982, pp. 28-30

Baptista, G. and Kulkarni, K., "A High Accuracy Algorithm for Recognition of Handwritten Numerals," *Pattern Recognition*, **21**, 4, 1988, pp. 287-291.

Baran, N., "Rough Gems: First Pen Systems Show Promise, Lack Refinement," *Byte Magazine*, **17**, 4, 1992, pp. 212-222.

Barr, A. and Feigenbaum, E. (Eds.), *The Handbook of Artificial Intelligence, Vol 1*, Kaufmann, Los Altos, CA., 1981.

Cash, G. and Hatamian, M., "Optical Character Recognition by the Method of Moments," *Computer Vision Graphics Image Processing*, **39**, 1987, pp. 291-310

Chanda, B., Chaudhuri, B. B., and Dutta Mayumder, D., "Some Modified Algorithms for Graylevel Threshholding," *International Conference on Pattern Recognition*, 1986, pp. 884-986.

Chen, C. "A Survey of Thinning Algorithms", Master's Project, Sam Houston State University, Huntsville, TX, 1993.

Chen, M., Hsu, W., and Cheng, F., "An Application of the Hough Transform to the Recognition of Handwritten Chinese Characters," *Proc. Int'l Computer Symposium*, Taiwan, Dec., 1986, pp. 719-727.

Cheng, F., and Hsu, W., "Research on Chinese OCR in Taiwan," in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 139-164.

Clark, L. and Velten, V., "Image Characterization for Automatic Target Recognition Algorithm Evaluations," *Optical Engineering*, **30**, 2, February 1991, pp. 147-153.

Cohen, E., Hull, J., Srihari, S., "Understanding Handwritten Text in a Structured Environment: Determining Zip Codes From Addresses," in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 221-261.

Crawford, Walt, "Catching Pictures, Catching Words: Low-Cost Scanning and Optical Character Recognition," *Library HI TECH*, **9**, 1, Jan. 1991, pp. 91-112.

Deutsch, W. S., "Thinning Algorithms on Rectangular, Hexagonal, and Triangular Arrays," *CACM*, **15**(9), 1972, pp. 827-837.

Denier Van Der Gon, J. and Thuring, J. "The Guiding of Human Writing Movements," *Biological Cybernetics*, **2**, 1965, pp. 145-148.

Denier Van Der Gon, J., Thuring, J. and Strackee, J., "A Handwriting Simulator," *Phys. Med. Biol.*, **6**, 1962, pp. 406-414.

Dooijes, E., "Analysis of Handwriting Movements," *Acta Informatica*, **54**, 1983, pp. 99-114.

Downton, A., Tregidgo, R. and Kabir, E., "Recognition and Verification of Handwritten and Hand-Printed British Postal Addresses," in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 265-291.

Dutta, A., "An Experimental Procedure for Handwritten Character Recognition," *IEEE Transactions on Computing*, **23**, 1974, pp. 536-545.

Earnest, L., "Machine Recognition of Cursive Writing," *Information Processing: Proceedings of the IFIP Congress '62*, 1962, pp. 462-465.

Eden, M., "On the formalization of Handwriting," in *Proceedings Symposiom on Applied Mathematics*, **12**, 1961, pp. 83-88.

Ehrich, R. and Koehler, K., "Experiments in the Contextual Recognition of Cursive Script," *IEEE Trans. Comput.*, **2** (2), 1975, pp. 182-194.

Farag, R., "Word Level Recognition of Cursive Script," *IEEE Transactions on Computers*, **28** (2), 1979, pp. 172-175.

Favita, J. and Srihari, S. N., "Recognition of Cursive Words for Address Reading,"*Fourth Advanced Technology Conference*, Washington, D. C., Nov 1990.

Fisher, P. S. and Case, J. H., "Long-Term Memory Modules," *Bulletin of Mathematical Biology*, **46**, 2, 1984, pp. 295-326.

Fletcher, L. A., and Kasturi, R., "A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images," *IEEE Transactions on Pattern Anal. and Machine Intelligence* **10**, 6, 1988, pp. 910-918.

Ford, D. and Higgins, C., "A Tree-Based Dictionary Search Technique and Comparison with N-Gram Letter Graph Reduction," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. G. Leedham, World Scientific, 1990, pp.291-312.

Forney, G., "The Viterbi Algorithm," *Proceedings of the IEEE*, **61**, 3, 1973, pp. 268-278.

Forsyth, R. and Rada, R., *MACHINE LEARNING Applications in Expert Systems and Information Retrieval*, Ellis Horwood Limited Pub., 1986.

Frishkopf, L. and Harmon, L., "Machine Reading of Cursive Script," in *Information Theory*, Ed. C. Cherry, Butterworth Pub., London, 1961.

Fu, K., "Tree Languages and Syntactic Pattern Recognition," in *Pattern Recognition and Artificial Intelligence*, ed. C. Chen, Academic Press, 1976.

Guyon, I., Albrecht, P., Le Cun, Y., Denker, J. and Hubbard, W., "Design of a Neural Network Character Recognizer for a Touch Terminal," *Pattern Recog.*, **24**, 2, 1991.

Guyon, I., Poujaud, I., Personnaz, L., Dreyful, G. Dender, J. and Le Cun, Y., "Comparing Different Neural Network Architectures for Classifying Handwritten Digits," *Proceedings Int'l Joint Conf. on Neural Networks, Volume II*, Washington DC, 1989, IEEE, pp. 127-132.

Frishkopf, L. and Harmon, L., "Machine Reading of Cursive Script," *Information Theory*, C. Cherry (ed.) Butterworth, London, 1961.

Granlund, G., "Fourier Preprocessing for Hand Print Character Recognition," *IEEE Transactions on Computing*, **21**, 1972, pp. 195-201.

Haralick, R. M., "Statistical and Structural Approaches to Texture," *Proc. of the 4th International Joint Conference on Pattern Recognition*, Kyoto, 1978, pp. 45-69.

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

Higgins, C., "*Automatic Recognition of Handwritten Script*," Ph. D. Thesis, CNAA, 1985.

Higgins, C. and Duckworth, R., "The PAD (Pen and Display) - A Demonstrator for the Electronic Paper Project," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. G. Leedham, World Scientific, 1990, pp. 111-131

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

Hilditch, C., "Linear Skeletons from Square Cupboards," *Machine Intelligence IV*, Eds. B. Meltzer and D. Mitchie, Elsevier, New York, 1969, pp. 403-420.

Holt, A., "Algorithm for a Low-Cost Hand Print Reader," *Computer Design*, Feb, 1974, pp. 85-89.

Horowitz, E. and Sahni, S., "Fundamentals of Data Structures," *Computer Science Press*, Rockville, MD, 1983, pp. 51-62.

Horowitz, S. L. and Pavlidis, T., "Picture Segmentation by a Directed Split-and-Merge Procedure," *Proc. of the 2nd International Joint Conference on Pattern Recognition,* Copenhagen, 1974, pp. 424-433.

Huang, J. and Huang, P., "Machine Printed Chinese Character Recognition based on Linear Regression," in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 165-173.

Hull, J. and Srihari, S., "Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* **4**, 5, 1982, pp. 520-530.

Hung, S. and Kasvand, T., "Critical Points on a Perfectly 8- or Perfectly 6- Connected Thin Binary Line," *Pattern Recognition,* **16**, 1983, pp. 297-306.

Inoue, K. and Takanami, I., "A Survey of Two-Dimensional Automata Theory," *Inform. Sci.*, **55**, 1991, pp. 99-121.

Inoue, K. and Takanami, I., "Characterization of Recognizable Picture Languages," in *Parallel Image Analysis and Processing*, Eds. K. Inoue, A. Nakamura, M. Nivat, A. Saoudi, and P. S. P. Wang, World Scientific, 1994. pp. 87-94.

Impedovo, S., Ottaviano, L., and Occhinegro, S, "Optical Character Recognition—A Survey," in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 1-24.

Iwata, K., Yoshida, M., and Tokunaga, Y., "High-Speed OCR for Handprinted Characters," *Proc. 4th Int'l Conf. on Pattern Recognition*, Nov., 1978, pp. 826-828.

Jagadish, H. V. and O'Gorman, L., "An Object Model for Image Recognition," *Computer,* **22**, 12, December 1989, pp. 33-41.

Jimenez, J., and Navalon, J., "Some Experiments in Image Vectorization," *IBM Journal of Research and Development*, **26**, 1982, pp. 724-734.

Kahan, S., Pavlidis, T., and Baird, H. S., "On the recogntion of hand printed characters of any font and size," *IEEE Trans. Pattern Anal. Machine Intelligence* **9**,2 1987 pp. 274-288.

Karbacher, S., "Associative Object Recognition by Hierarchic Template Matching," *Optical Engineering*, **29**, 12, December 1990, pp. 1449-1457.

Kadirkamanathan M. and Rayner P., "A Scale-Space Filtering Approach to Stroke Segmentation of Cursive Script," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. G. Leedham, World Scientific, 1990, pp. 133-166

Knoll, A., "Experiments with Characteristic Loci for Recognition of Hand Printed Characters," *IEEE Trans. on Computing*, **18**, Apr, 1969, pp. 366-372.

Knuth, D. "Sorting and Searching, The Art of Computer Programming," **3** Addison-Wesley, 1973, pp. 481-487.

Kondo, S., "On Determining Distinctive Features of Handprinted Characters using Statistical Techniques," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. G. Leedham, World Scientific, 1990, pp. 207-219.

Krzyzak, A., Leung, S. and Suen, C., "Reconstruction of Two-Dimensional Patterns by Fourier Descriptors," *Machine Vision and Applications*, Springer-Verlag, New York, Inc., 1989, pp. 123-140.

Kwok, P., "A Thinning Algorithm by Contour Generation," *Communications of the ACM*, **31**, 11, 1988, pp. 1314-1324.

Kwon, S. and Lai, D., "Recognition Experiments with Hand Printed Numerals," *Proc. Joint Workshop on Pattern Recognition and Artificial Intell.*, June, 1976, pp. 74-83.

Kyung, H. A., "Concurrent Pattern Recognition and Optical Character Recognition," Ph. D. Dissertation, University of North Texas, Denton, TX, 1991.

Lam, L. and Suen, C., "Structural Classification and Relaxation Matching of Totally Unconstrained Handwritten Zip Code Numbers," *Pattern Recognition*, **21**, 1988, pp. 19-31.

Lam, L., Lee, S. and Suen C., "Thinning Methodologies—A Comprehensive Survey," *IEEE Trans. on Pattern Anal. Mach. Intell.*, **14**, 9, 1992, pp. 869-885.

Le Cun, Y., Jackel, L., Boser, B. Denker, S. Graf, H. Guyon, I. Handerson, D., Howard, R. and Hubbard, W., "Handwritten digit recognition: Application of Neural Network Chips and Automatic Learning," *IEEE Communication Magazine*, Nov., 1989, pp. 41-46.

Lee, S., Lam, L. and Suen, C., "A Systematic Evaluation of Skeletonization Algorithms," in *Thinning Methodologies for Pattern Recognition*, Eds. C. Suen and P. Wang, World Scientific, 1994, pp. 239-261.

Liu, C., Herbst, N. and Anthony, N., "Automatic Signature Verification: System Description and Field Results," *IEEE Trans. on Syst., Man, and Cybernetics* **9**, 1 1979 , pp. 35-38.

Lowerre, B. and Reddy, R., "The HARPY Speech Understanding System," in *Trends in Speech Recognition*, ed. W. Lea, 1980, pp. 340-360.

Mermelstein, P. and Eden, M., "Experiments on Computer Recognition of connected handwritten words," *Inf. Control*, 7, 1964, pp. 255-270.

Morasso, P. and Mussa Ivaldi, F., "Trajectory Formation and Handwriting: A Computational Model," *Biol. Cybernetics*, **45**, 1982, pp. 131-149.

Naccache, N. and Shinghal, R., "SPTA: A Proposed Algorithm for Thiining Binary Patterns," *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-14**, 3, May 1984, pp. 409-418.

Nagy, G. and Tuong, N., "Normalization Techniques for Handprinted Numerals," *CACM*, **13**, Aug., 1970, pp. 475-481.

Netvia, R., "Image Segmentation," in *Handbook of Pattern Recognition and Image Processing*, Academic Press, Inc., 1986, pp. 215-231.

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

Nishida, H. and Mori, S., "Structural Analysis and Description of Curves by Quasi-Topological Features and Singular Points," in *Structural Document Image Analysis*, Eds. H. S. Baird, H. Bunke, and K. Yamamoto, Springer-Verlag, 1992.

O'Gorman, F. and Clowes, M. B., "Finding Picture Edge through Collinearity of Feature Points," *IEEE Transactions on Computers* **25**, 1976, pp. 449-456.

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

Parisse, C., Rosenthal, V., Imagache, A., Andreewsky, E. and Cochu, F., "A Task Oriented Approach to Reading and to Handwritten Text Recognition," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. G. Leedham, World Scientific, 1990, pp. 313-335.

Parks, J., Bell, D., Watson, R., Cowin, G., and Olding, S., "An Articulate Recognition Procedure Applied to Handprinted Numerals," *Proc. 2nd Int'l Joint Conference Pattern Recognition*, Aug., 1974, pp. 416-420.

Pavlidis, T., *Algorithms for Computer Graphics and Image Processing*, Computer Science Press, 1982, pp. 195-214.

Pavlidis, T., "A Hybrid Vectorization Algorithm," *Proc. 7th Int'l Conference on Pattern Recognition*, Montreal, 1984, pp. 490-492.

Pavlidis, T., "A Vectorizer and Feature Recognizer for Document Recognition," *Computer Vision Graphics Image Processing*, **35**, 1986, pp. 111-127.

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

Persoon, E. and Fu, K., "Shape Descrption using Fourier Descriptors," *IEEE Trans. Syst. Man Cybern.*, 7, 1977, pp. 170-179.

Plamondon, R. "Handwriting Model Based on Differential Geometry," in *Computer Recognition and Human Production of Handwriting* , R. Plamondon, Suen, C. Y., and Simner, M. L. (eds.), World Scientific, 1989, pp. 179-192.

Plamondon, R. and Lamarche, F., "Modelization of Handwriting: A System Approach," in *Graphonomics: Contemporary Research in Handwriting*, eds. Kao, Van Galen, Hoosain, Elsevier Science Publishers B. V., Amsterdam 1986, pp. 169-183.

Plamondon, R. and Lorette, G., "Automatic Signature Verification and Identification: The State of the Art," *Pattern Recognition*, **22**, 2, 1989, pp. 107-131.

Plamondon, R., Suen, C., Bourdeau, M. and Barriere, C., "Methodologies for Evaluating Thinning Algorithms for Character Recognition," in *Thinning Methodologies for Pattern Recognition*, Eds. C. Suen and P. Wang, World Scientific, 1994, pp. 283-306.

Ramer, U., "An Iterative Procedure for the Approximation of Plane Curves," *Computer Graphics and Image Processing*, **1**, 1972, pp. 244-256.

Reinhardt, A., "Motorola's Envoy First to Run Magic Cap," *Byte Magazine*, **19**, 5, May 1994, pg. 34.

Rich, E., *Artificial Intelligence*, McGraw-Hill, Inc., New York, NY 1983.

Rimmer, Steve, *Bit Mapped Graphics*, Windcrest/McGraw Hill Publishing, Blue Ridge Summit, PA, 1990.

Riseman, E. and Ehrich, R., "Contextual Word Recognition Using Binary Digrams," *IEEE Trans. on Computers*, **20**, 4, April 1971, pp. 397-403.

Riseman, E. and Hanson, A., "A Contextual Post-processing System for Error Correction Using Binary N-grams", *IEEE Transactions on Computers*, **23**, 5, 1974, pp. 480-493.

Rosenfeld, A., "Picture Languages (Formal Methods for Picture Recognition)," Academic Press, New York, 1979.

Rosenfeld, A. and Davis, L. "A Note on Thinning," *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-6**, March 1976, pp. 226-228.

Rosenfeld, A. and Thurston, M. ""Edge and Curve Detection for Visual Scene Analysis," *IEEE Transactions on Computers* **20**, 1971, pp. 562-569.

Sadjadi, F. and Bazakos, M. "A Perspective on Automatic Target Recognition Evaluation Technology," *Optical Engineering*, **30**, 2, Feb. 1991, pp. 141-146.

Sayer, K., "Machine Recognition of Handwritten Words," *Pattern Recogn.*, **5** (3), 1973, pp. 213-228.

Sato, Y. and Kogure, K., "On-line Signature Verification based on Shape, Motion, and Writing Pressure," *Proc. of the Sixth Int'l Conference on Pattern Recognition*, Munich, Oct. 1982, pp. 823-826.

Schomaker, L., Thomassen, A., and Teulings, H., "Computation Model of Cursive Handwriting," in *Computer Recognition and Human Production of Handwriting* , R. Plamondon, Suen, C. Y., and Simner, M. L. (eds.), World Scientific, 1989, pp. 153-177.

Shaw, A., "Picture Graphs, Grammars, and Parsing," in *Frontiers of Pattern Recognition*, ed. S. Watanabe, Academic Press,1972.

Shimura, M., "Multicategory Learning Classifiers for Character Reading," *IEEE Trans. Syst. Man, Cybern.*, **3**, Jan., 1973, pp. 74-85.

Simon, J. and Baret, O., "Regularities and Singularities in Line Pictures", in *Character and Handwriting Recognition, Expanding Frontiers*, Ed. P. S. P. Wang, World Scientific, 1991, pp. 57-77.

Slansky, J. and Gonzales, V., "Fast Polygonal Approximation of Digitized Curves," *Pattern Recognition*, **12**, 1981, pp. 327-331.

Smith, R., "Computer Processing of Line Images—A Survey," *Pattern Recognition*, **20**, 1987, pp. 7-15.

Spanjersberg, A., "Combinations of Different Systems for he Recognition of Handwritten Digits," *Proc. 2nd Int'l Joint Conf. on Pattern Recog. and Artificial Intell.*, Aug, 1974, pp. 208-209.

Srihari, S. N. and Bozinovic, R. M., "A Multi-Level Perception Approach to Reading Cursive Script," *Artificial Intelligence*, **33**, 1987, pp. 217-255.

Srihari, S., Hull, J., and Choudary, R. "Integrating Diverse Knowledge Sources in Text Recognition," *ACM Transactions on Office Information Systems*, **1**, 1, 1983, pp. 68-87.

Shinghal, R. and Toussaint, G., "Experiments in Text Recognition with the Modified Viterbi Algorithm," *IEEE Trans. of Pattern Analysis and Machine Intelligence*, **1**, 2, 1979a, pp. 184-192.

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

Stillman, R., "Character Recognition Based on Phenomological Attributes: Theory and Methods," Ph. D. Dissertation, MIT, Cambridge, MA. 1974.

Sue, T. and Chen, Z., "Skeletal Chain Code Approach to Recognition of Handwritten Numerals," *Proc. National Computer Symposium*, Dec., 1976, pp. 4.15-4.26.

Suen, C. Y., "Distinctive Features in Automatic Recognition of Hand Printed Characters," *Signal Process.* **4**, Apr. 1982, pp 193-207.

Suen, C., Berthod, M. and Mori, S., "Automatic Recognition of Hand Printed Characters—The State of the Art," *Proc. IEEE*, **68**, 4, 1980, pp. 469-487.

Sun, C. and Wee, W., "Neighboring Gray Level Matrix Textural Classification," *Computer Vision and Graphics Image Processing* **23**, 1982, pp. 341-352.

Tappert, C., "Cursive Script Recognition by elastic matching," *IBM Journal of Res. Dev.*, **26**, No. 6, 1982, pp 765-771.

Tappert, C., Suen, C., and Wakahara, T., "On-line handwriting recognition—A Survey," *Proc. of the 9th Int'l Conf. on Pattern Recognition*, **2**, 1988, pp. 1123-1132.

Tappert, C., Suen, C., and Wakahara, T., "The State-of-the-Art in On-line Handwriting Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, **12**, 1990, pp. 787-808.

Tavakoli, N., "A New Approach to Pattern Recognition", Ph.D. Dissertation, Kansas State University, Manhattan, Kansas, 1986.

Teulings, H., Thomassen, A., and Van Galen, G., "Invariants in Handwriting: The Information Contained in a Motor Program," in *Graphonomics: Contemporary Research in Handwriting*, eds. Kao, Van Galen, Hoosain, Elsevier Science Publishers B. V., Amsterdam 1986, pp. 305-315.

Tubbs, J., "A Note on Binary Template Matching," *Pattern Recognition*, **22**, 4, 1989, pp. 359-365.

Tucker, N. and Evans, F., "A Two-step Strategy for Character Recognition using Geometrical Moments," *Proc. 2nd Int'l Joint Conf. on Pattern Recognition*, Aug, 1974, pp. 223-225.

Van Galen, G., Meulenbroek, R., and Hylkema, H., " On the Simultaneous Processing of Words, Letters, and Strokes in Handwriting: Evidence for a Mixed Linear and Parallel Model," in *Graphonomics: Contemporary Research in Handwriting*, eds. Kao, Van Galen, Hoosain, Elsevier Science Publishers B. V., Amsterdam 1986, pp. 5-20.

Van Galen, G., Smyth, M., Meulenbroek, R., and Hylkema, H., "The Role of Short-Term Memory and the Motor Buffer in Handwriting under Visual and Non-Visual Guidance," in *Computer Recognition and Human Production of Handwriting*, Eds. R. Plamondon, C. Suen, M. Simner, World Scientific, 1989, pp 253-271.

Van Galen, G. and Teulings, H., "The Independent Monitoring of Form and Scale Factors in Handwriting," *Acta Informatica*, **54**, 1983, pp. 9-22.

Viterbi, A., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. of Information Theory*, **IT-13**, 1967, pp. 260-269.

Vredenbregt, J. and Koster, W., "Analysis and Synthesis of Handwriting," *Phillips Tech. Rev.*, **32**, 1971, pp. 73-78.

Wang, P. and Zhang, Y., "A Fast and Flexible Thinning Algorithm," *IEEE Transactions on Computers*, **38**, 5, 1989, pp. 741-745.

White, G., "Natural Language Understanding and Speech Recognition," *Communications of the ACM*, **33**, 8, August 1990, pp 72-82.

Whitrow, R. and Higgins, D., "The Application of n-Grams for Script Recognition," *Proc. of the Third Int'l symposium on Handwri5ing and Computer Applications*, Montreal, Canada, July 1987, pp. 92-94.

Xu., W. and Wang, C., "CGT: A Fast Thinning Algorithm Implemented ona Sequential Computer," *IEEE Trans. on Syst. Man Cybernetics*, **17**, 5, 1987, pp. 847-851.

Zhang, T. and Suen, C. "A Fast Parallel Algorithm for Thinning Digital Patterns," *Communications of the ACM*, **27**, 3, Mar 1984, pp. 236-239.

Zhang, Y. and Wang, P., "Analytical Comparison of Thinning Algorithms," in *Thinning Methodologies for Pattern Recognition*, Eds. C. Suen and P. Wang, World Scientific, 1994, pp. 263-282.