# AUTOMATIC SEGMENTATION AND RECOGNITION OF UNCONSTRAINED HANDWRITTEN NUMERAL STRINGS

JAVAD SADRI

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

APRIL 2007

*Your file Votre référence*
*ISBN: 978-0-494-31139-4*
*Our file Notre référence*
*ISBN: 978-0-494-31139-4*

# Abstract

Automatic Segmentation and Recognition of Unconstrained Handwritten Numeral Strings

Javad Sadri, Ph.D.

Concordia University, 2007

Segmentation and recognition of handwritten numeral strings is a very interesting and challenging problem in pattern recognition. It also has a lot of important applications such as: postal code recognition, bank check processing, tax form reading, etc. In this thesis, a new system for the segmentation and recognition of unconstrained handwritten numeral strings is proposed. The system uses a combination of foreground and background features for the segmentation of touching numerals in strings. The method introduces new algorithms for the traversal of top and bottom foreground and background skeletons, and top and bottom contours of numerals. Then, it tries to locate all feature points on these skeletons and contours and alternatively match feature points from top to bottom (or bottom to top) of the images to build all possible candidate segmentation paths (so-called segmentation hypotheses). A novel genetic representation scheme is utilized in order to represent the space of all possible segmentation hypotheses. In order to improve searching and evolution of segmentation hypotheses and facilitate finding the ones with the highest confidence values of segmentation and recognition, this genetic framework utilizes contextual knowledge extracted from string images. A novel evaluation scheme based on segmentation and recognition scores is introduced in order to improve the evaluation of segmentation hypotheses and to enhance the outlier resistance of the system. In order to improve stability and plasticity of our system in the learning and recognition of numerals, a new algorithm for clustering of handwritten digits based on their shapes is proposed. Also, in order to improve the searching power of our system and its convergence, a new evolutionary algorithm based on genetic particle

swarm optimization (GBPSO) is proposed. Numerous experiments using images from well known databases of handwritten numeral strings such as CENPARMI, NIST NSTRING SD19, and our newly created databases of Farsi/Arabic numerals have been conducted in order to evaluate the performance of the proposed method. Experiments have shown that proper use of contextual knowledge in segmentation, evaluation and search greatly improves the overall performance of the system. This system shows superior results compared with those reported in the literature.

# Acknowledgments

I would like to sincerely thank my supervisors, Professor Ching Y. Suen and Professor Tien D. Bui for their guidance and their scientific support over the last few years during my Ph.D. studies and research projects. Their advice has always been a major source of inspiration for me.

Many thanks to Dr. Jianxiong Dong, Dr. Luiz Eduardo Soares Oliveira, Dr. Mohamed Cheriet, Dr. Nematollaah Shiri, and Dr. Cheng-Lin Liu for their advice, and encouragement.

I am grateful to all my friends at CENPARMI (Center for Pattern Recognition and Machine Intelligence at Concordia University) for their help and friendship that has enriched my studies during these years. CENPARMI, as a research center, has provided me with a very happy and rich environment to learn, think, criticize, be criticized, and share my ideas with other people, and to try new ideas. My special thanks at CENPARMI goes to Mr. Nicola Nobile for his technical assistance, and to Ms. Shira Katz for her editorial assistance.

I also would like to thank the members of my examining committee: Dr. David Doermann, Dr. Nawwaf Kharma, Dr. Tony Kasvand, and Dr. Adam Krzyzak for their kind advice and very helpful comments and discussions.

Most of all, I would like to thank my wife, Dr. Zari Dastani, who brought a peaceful and happy time for me and our two sons, and who gave me strength all through the challenges. Without her efforts and her patience it would have been impossible for me to come to Canada

and to continue my doctoral studies. This work is cordially dedicated to my wife, and our two curious sons: Iman, and Erfan.

The research in this thesis has been financially supported by awards and research grants from CENPARMI, Concordia University, The Power Corporation of Canada, NSERC (Natural Sciences and Engineering Research Council of Canada), and FQRNT (Fonds de Recherche sur la Nature et les Technologies de Quebec). I sincerely appreciate the support of all of them.

# Contents

# List of Figures

# List of Tables

# List of Acronyms and Symbols

| | |
|---|---|
| CENPARMI | Center for Pattern Recognition and Machine Intelligence |
| CC | Connected Component |
| CG | Center of Gravity |
| HMM | Hidden Markov Model |
| HSF | Handwritten Sample Forms |
| PDF | Probability Density Function |
| GA | Genetic Algorithm |
| IP | Intersection Point |
| OCR | Optical Character Recognition |
| SVM | Support Vector Machines |
| NN | Nearest Neighbor |
| KNN | K-Nearest Neighbor |
| MLP | Multi-Layer Perceptron |
| NIST | National Institute of Standards and Technology |
| MNIST | Modified NIST |
| PP | Post-Processor |
| RR | Recognition Rate |
| SE | Square Error |
| I | String Image |
| SD | Special Database |
| CSA | Component Slant Angle |

| | |
|---|---|
| SSA | String Slant Angle |
| UHNS | Unconstrained Handwritten Numeral Strings |
| RBF | Radial Bases Function |
| PSO | Particle Swarm Optimization |
| GBPSO | Genetic Based Particle Swarm Optimization |
| Rec. (%) | Recognition rate (%) |
| Err. (%) | Rejection rate (%) |
| Rej. (%) | Rejection rate (%) |
| $S^i$ | ith segmentation Hypothesis |
| $s^i_j$ | jth segment in Segmentation Hypothesis $S^i$ |
| $F(S^i)$ | Objective function |
| $\nu^i_j$ | Segmentation-recognition score |
| $V_i$ | Velocity vector of particle $i$ |
| $X_i$ | Position vector of particle $i$ |
| $\gamma^*$ | Parameters of our similarity measure |
| $\delta$ | Parameters of our clustering algorithm |
| $b(t)$ | Birth rate |
| $m(t)$ | Mortality rate |
| $r(t)$ | Overall birth rate |

# Part I

# Design and Implementation of The Modular System

# Chapter 1

# Introduction

Handwriting consists of artificial graphical signs and marks written by human on a surface such as paper, wood, metals, glass, rocks, etc. The purpose of handwriting is to register, transfer, or communicate news, messages, business information, ideas, etc. In fact, handwriting is a skill, learned by educated (or semi-educated) people and that is closely affected by their physical characteristics, age, personality, or mood. This can explain the great variability that exists in the handwritings of different people. In spite of this variability, human beings can still decipher most of the information in handwritings, accurately. For example, they can recognize handwritten numerals, letters, strings, and words, in spite of the very large variability in their shapes, sizes, writing styles, and their occlusions or noisy backgrounds. This ability of human beings is appreciated when faced with the difficulties of training computers to do the same task of recognizing handwritings. After more than 60 years of research in the area of pattern recognition and artificial intelligence, designing a general handwriting recognition system with capabilities similar to human beings still remains a big challenge and an open problem. It seems unlikely that a unique system that can simulate all human abilities in handwritten recognition will be built. So normally, the goal in designing handwritten recognition systems is to make computers perform as closely as possible to human beings in some aspects or subtasks of handwritten recognition.

Handwritten numeral strings are important part of handwritten documents, and recognition of these numerals has many potential applications, such as the recognition of postal codes

in mail, courtesy amounts in bank checks, revenue values in tax forms, and other numeric values in application forms. In this thesis, we consider the design and implementation of a system for segmentation and recognition of unconstrained handwritten numeral strings. There are many challenges in the problem of segmentation and recognition of unconstrained handwritten numeral strings. In this thesis, we discuss these challenges, and we present new solutions (algorithms) for them. Also, based on these solutions a new modular system for segmentation and recognition of unconstrained handwritten numeral strings is developed and tested.

## 1.1 Problem Statement

The focus of this thesis is the segmentation and recognition of Unconstrained Handwritten Numeral Strings (UHNS). By definition, unconstrained handwritten numeral strings are handwritten numerals with no limitations in their writings, e.g., UHNS are not written in separate boxes, nor written neatly, nor written with a special type of pen [9]. UHNS are normally found in real life applications such as courtesy amounts on bank cheques, postal codes on envelopes and numerical values on various business forms. Some examples of UHNS are shown in Figures 1, and 2. Generally, UHNS include isolated digits (with a lot of variations in their shapes, sizes or styles of writing), touching digits, overlapping digits, noisy or broken digits. Among challenges in the segmentation and recognition of UHNS, segmentation and recognition of touching or highly overlapped digits is the most difficult one.

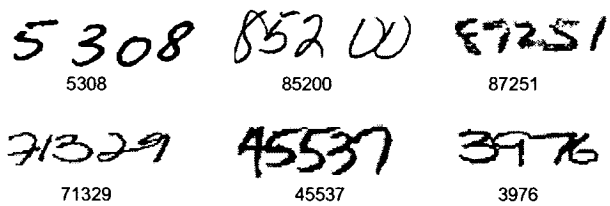



Figure 1: Some examples of unconstrained handwritten numeral strings from the NIST SD19 database.

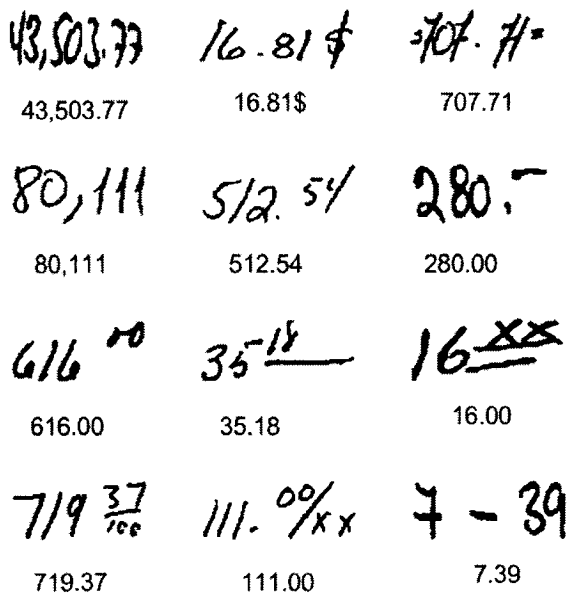



Figure 2: Illustration of variations in writing styles of courtesy amounts. Examples in this figure have been taken from CENPARMI courtesy amounts database.

Here, the challenge is to segment and recognize touching numeral strings with a very high accuracy. Touching numerals frequently occur in UHNS. For example, a study [10] on 1100 French bank checks showed that 6.2% of those checks contained connected digits. The distribution of 1189 connected numeral pairs is shown in Table 1. Here, 'X' represents any numeral, for example, 5X represents touching between numeral 5 and any other numeral. Another study [11] on 400 Brazilian checks showed that about 80% to 85% of successive zeros were connected; see Figure 3. In another research study, Wang et al. [12] showed that 84.6% of the touching cases occurred between two consecutive numerals, while the remaining occurred among three or more consecutive numerals. Because of the importance of handwritten numeral strings in real life applications, and also due to frequent occurrence of touching digits in these numerals, it is very important to develop effective and high performance methods in order to accurately segment and recognize UHNS.

Table 1: Distribution of 1189 touching cases in a sample of 1100 French bank checks

| Connected Pairs: | 1X | 2X | 3X | 4X | 5X | 6X | 7X | 8X | 9X | 0X |
|---|---|---|---|---|---|---|---|---|---|---|
| Occurrence (%) | 7.9 | 13.03 | 6.05 | 2.43 | 15.89 | 8.49 | 26.19 | 11.94 | 3.78 | 4.28 |

Figure 3: Example of touching successive zeros in numeral strings.

## 1.2 Goals of the Research

Many methods have been proposed for the segmentation and recognition of handwritten numeral strings such as in [9],[8],[13],[14], etc. However, it seems that present techniques are not yet adequate enough for modeling all the infinite variations of the handwritten strings. Many improvements are still required in order to build robust systems for real life applications. Our main goals in this research thesis are as follows:

- To introduce a method that improves building of segmentation candidates in order to lower as much as possible the number of outliers (refer to Figure 4) at the segmentation stage of UHNS.
- To introduce a method that improves outlier resistance (outlier rejection) of ordinary isolated digit classifiers in order to lower as much as possible the number of misclassifications at the recognition stage of UHNS.
- To design and implement a modular system for the segmentation and recognition of UHNS with a very high performance.

In general, we are going to introduce new algorithms and new techniques that can improve the accuracy of segmentation and recognition of unconstrained handwritten numeral strings.


Figure 4: Outliers are defined as samples of wrongly segmented isolated or touching digits (such as (a), and (b)). Example of outliers: (a) Over-segmented digits, and (b) Under-segmented touching digits. More examples of outliers (over/under segmented digits) and also more information about their detection and rejection will be presented in chapters 7 and 8.

We are going to use all possible sources of background and foreground information, and contextual knowledge from the string image in order to facilitate locating and searching of segmentation candidates and also to improve the evaluation of the corresponding separated digits. In other words, we are going to introduce segmentation algorithms that produce fewer outliers. Also, we are going to introduce evaluation algorithms that show higher outlier resistance in order to remedy insufficient outlier resistance of the ordinary isolated digit classifiers in UHNS. Finally, we are going to build a modular system for the segmentation and recognition of numeral strings which utilizes all these techniques, and we are going to evaluate and compare its performance with similar systems.

## 1.3 Contributions

The original contributions of this thesis can be summarized as follows:

- The introduction of a novel and efficient algorithm for slant correction of handwritten numeral strings. Using this algorithm, we investigated the statistical characteristics of slant angles in handwritten numeral strings. We also studied the effects of slant angle correction on the performance of of segmentation of handwritten numeral strings [15].

- For the first time, we introduced a novel and efficient algorithm for tracing the skeletons of digits and characters. This algorithm is similar to the algorithms that exist for tracing of contours or boundaries in image processing. Our skeleton tracing algorithm

is a general algorithm that can be used for tracing the skeletons of any 2D objects (planar shapes). This algorithm can be used for extracting structural features from skeletons and for exploring structures of unknown 2D objects [16, 17].

- The introduction of a novel and efficient algorithm for segmentation and separation of touching digits in unconstrained handwritten numeral strings. Our algorithm extracts new features, from both the backgrounds and foregrounds of string images, and it combines the information from the skeletons and contours of digits in order to build segmentation cutting paths [16, 18].

- The introduction of a new evaluation scheme (new scores) for evaluating segmentation hypothesis, based on the contextual information extracted from numeral strings. Our method can facilitate and improve detection of outliers (under/over-segmented components, in fact, wrongly segmented digits). This method compensates for the lack of outlier resistance and improves the weakness of ordinary isolated digit classifiers in rejecting outliers. In this thesis, these new scores are called segmentation scores, as opposed to recognition scores which are produced by ordinary isolated digit classifiers [17].

- We formulated (modeled) segmentation and recognition of unconstrained handwritten numeral strings as a global optimization problem. Our objective is to maximize both the segmentation and recognition confidences for any given image containing a numeral string (with any unknown length). We introduced a general framework for this optimization problem based on genetic (evolutionary) algorithms in order to find the optimal solution. We also introduced a novel genetic representation scheme that can be applied in searching for the optimal path in any Directed Acyclic Graph (DAG) using a genetic search [18, 17].

- In order to capture the great variability that exists in the shapes of handwritten digits and to improve the recognition accuracy of isolated digit classifiers, we introduced a new online clustering algorithm that clusters the shapes and the styles of writing

of digits. Our algorithm is an incremental unsupervised learning algorithm that can improve stability and plasticity of handwritten recognition systems [19].

- The introduction of a new class of evolutionary algorithms based on the combination and generalization of ideas from Genetic Algorithms (GA's) and Particle Swarm Optimization (PSO). This new class of evolutionary algorithms is called genetic-based particle swarm optimization (GBPSO) and it shows a higher performance and faster convergence compared to GA and ordinary PSO [20].

- Developing a new comprehensive standard database for research, training, evaluation, and testing of handwritten recognition systems for Farsi (Persian) language. This database is composed of six different sets of handwritten images including: isolated digits, isolated letters, numeral strings, legal amounts, dates, and Arabic isolated digits written by people who speak Farsi. These data sets can be used by researchers who want to investigate handwritten Farsi script recognition [21. 22]. (This was a joint research work with another graduate student).

- Applying developed algorithms including: segmentation, recognition, slant correction, and clustering on Farsi handwritten numeral strings. We investigated and evaluated the performances of these algorithms on Farsi handwritten numerals [23, 24, 25].

## 1.4 Outline of this Thesis

This thesis consists of three main parts. The first part of the thesis contains eight chapters. These chapters describe our main ideas, algorithms, and experiments for designing, implementation and testing of a new modular system for segmentation and recognition of unconstrained handwritten numeral strings. The second part of this thesis has three chapters which expand our research towards generalization and new applications of the algorithms and techniques introduced in the first part. Also in this part, we modify the algorithms that developed in the first part to be applicable for the numeral strings in other languages such as Farsi and Arabic. The third part of this thesis belongs to our conclusion. The details of the

chapters in this thesis are as follows: the current chapter (Chapter 1) outlines the problem, its definition, our goals, and our main contributions in this thesis. In Chapter 2, we present a review of the state-of-the-art techniques for handwritten numeral string segmentation and recognition. In Chapter 3, we present an overview of our system. In Chapter 4, we present the details of our pre-processing steps, such as noise removal, and slant correction. We also present statistical properties of slant angles in handwritten numeral strings. In Chapter 5, the details of the algorithms for generation of segmentation hypotheses are explained. In Chapter 6, the details of our representation and searching strategy are presented. Chapter 7 presents the the details of our evaluation scheme for segmentation hypotheses. We also introduce the segmentation score and the recognition score. In Chapter 8, the details of our experimental results are elaborated. We present the experimental results on all our developed algorithms on handwritten numeral strings. In Chapter 9, first the importance of plasticity and stability in handwritten digit recognition systems is discussed, and a new clustering algorithm for handwritten digits is presented. Then, using this clustering algorithm, a new method for improving plasticity and stability in handwritten digit recognition systems is presented. Chapter 10 presents a generalized model of evolutionary algorithms for search and optimization, based on Genetic Algorithms (GA) and Particle Swarm Optimizers (PSO). In Chapter 11, segmentation and recognition of handwritten numerals in Farsi and Arabic languages is presented. Finally, Chapter 12 draws the conclusion of this thesis and future works are discussed.

# Chapter 2

# Review of State-of-the-Art Techniques in Segmentation and Recognition of Handwritten Numeral Strings

The focus of this thesis is offline recognition of handwritten numeral strings. In this chapter, we briefly review some of the state-of-the-art techniques and developments for offline pre-processing, segmentation, feature extraction, and recognition of handwritten numeral strings in the literature. In general, a conventional system for offline segmentation and recognition of handwritten numeral strings has an architecture as shown in Figure 5. Here, first the function of each module is explained, and then we briefly review the literature on each of these functions (processes).

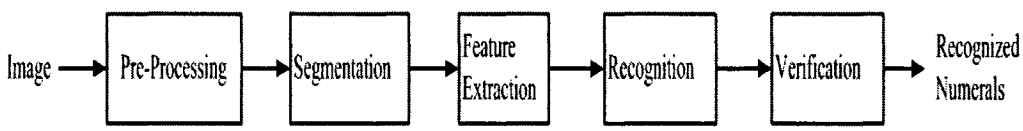



Figure 5: Block diagram of a general numeral string segmentation and recognition system.

- **Pre-processing:** This module may include some image processing operations such

as binarization of input images, noise reduction, smoothing, filling, slant correction, baseline detection, skew correction detection, or other image transformations such as normalization.

- **Segmentation:** This module introduces some segmentation cutting paths (segmentation hypotheses) in order to separate touching or overlapping digits in numeral strings.
- **Feature Extraction:** In this module, for each separated digit (or connected component), some features are extracted (to be used by the recognition module).
- **Recognition (Classification):** In this module, an isolated digit classifier (or a combination of isolated digit classifiers) assigns class labels and confidence values to segmented digits or connected components.
- **Verification:** In some systems, there is an optional verifier that verifies the decisions made by the recognizer(s). This module tries to reduce (resolve) the errors made by the recognizer.

## 2.1 Pre-processing of Handwritten Documents

Pre-processing steps are very important in order to improve the readability and the automatic recognition of handwritten document images. These steps make the segmentation and feature extraction processes more reliable and effective. In our method we only use smoothing and slant correction as pre-processing steps. Here, we review one of the methods found on the literature on slant correction.

The method presented by Britto et al. [3] is the only method found in the literature for slant correction of un constrained handwritten numeral strings. They modified, a word slant normalization method in order to improve the results for handwritten numeral strings. For each connected component (CC) in the string they calculated an slant angle. Using these slant angles and the contour length of each CC, Britto et al. [3] obtained a mean slant angle for the entire numeral string. Then, they used this average angle for the slant correction of the entire string.

## 2.2 Segmentation of Handwritten Numeral Strings

In this section, we review state of the art methods on segmentation of connected digits. Among earlier pieces of literature for the segmentation of handwritten numeral strings, one class of methods is based on extracting the profiles of touching digits [26]. Fujisawa et al. [26] proposed a method based on profile features. In this method, the segmentation path is constructed by computing the upper, and lower profiles of the connected components and analyzing the distance between these upper, and lower profiles (see Figures 6-a, b, and c).

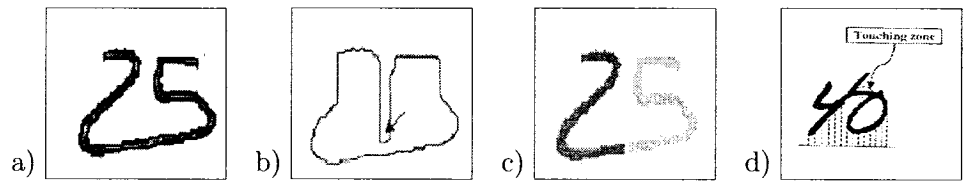



Figure 6: (a) A pair of touching digits, (b) Their profiles, (c) Their segmentation path, and (d) An example of a situation where a segmentation path cannot be found by using profile features.

Although some segmentation paths can be successfully produced by computing the upper profile and lower profile of the connected components, and analyzing the distance between these profiles, this approach fails when handwritings are skewed or strongly overlapped, because in these cases the proper segmentation points (or regions) cannot be reached simply by analyzing a vertical profile (see Figure 6-d).

Another class of methods for segmentation is based on analyzing the contour of a touching digits [13, 14, 27]. Strathy et al. [13] proposed a method for the segmentation of touching numeral strings based on an analysis of the contour valleys and mountains as feature points. In their method, by joining valley points and mountain points on the top and bottom contours, cutting paths are constructed, as illustrated in Figures 7-a, and b. Although contour points provide very important information for segmentation, in some cases when the two numerals touch each other through a straight line, segmentation methods based on contour fail (see examples in Figure 7-c). In these cases, there are no corresponding valley and mountain points in the touching regions.

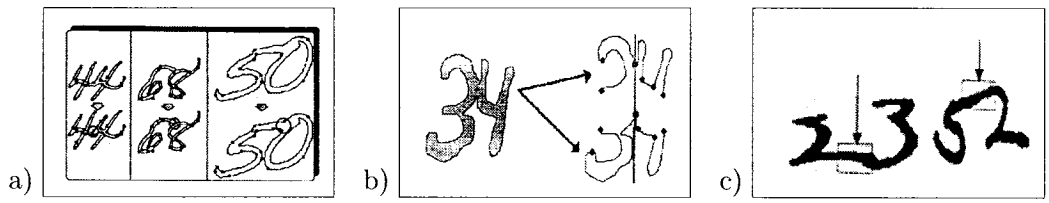



Figure 7: (a) Analyzing and extracting contour points, (b) Constructing the cutting path, and (c) Some cases where contour points cannot identify the touching regions.

Some other methods use features from the background and foreground skeletons such as Chen and Wang [2], and Lu et al. [1]. Chen and Wang [2], used background and foreground skeletons for the segmentation of connected components. They extracted feature points such as: fork, end, or corner points on the top, bottom, hole, or stroke segments of these skeletons (see Figure 8-a, to e). Then they matched feature points from top segments to bottom segments of the skeletons to construct the candidate segmentation paths (see Figure 9-a, b, and c). In their method, they tried to evaluate segmentation candidates based on some geometric features and rank them using a mixture of Gaussian probabilities in order to find the best segmentation path, or reject the candidate paths. In fact, these methods are very time consuming, and additionally skeletons usually generate protrusions. These protrusions sometimes cause confusion with actual fork and end points, and they may yield wrong candidate cutting path(s) (see examples in Figures 9-d, and e).

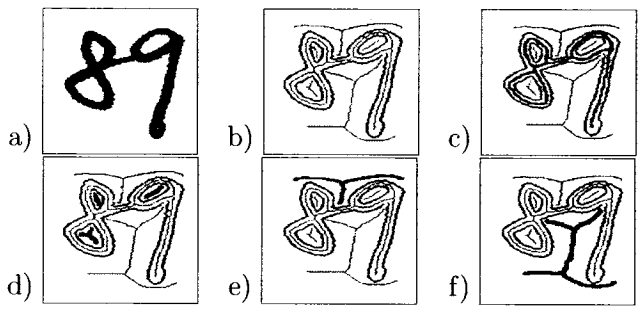



Figure 8: (a) Original image, (b) Foreground and background skeletons, (c) Stroke segment, (d) Hole segments, (e) Top background segment, (d) Bottom background segment.

Some other methods use structural and morphological features for segmentation, such as

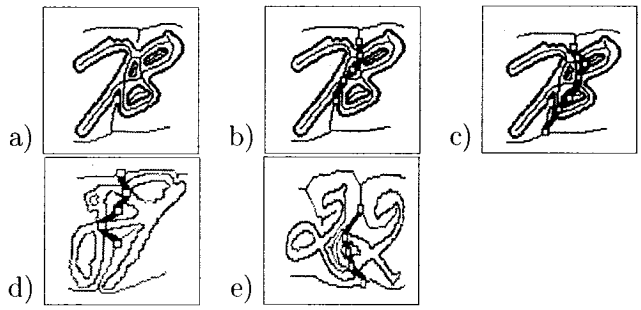



Figure 9: (a) Skeleton segments [(b) and (c)] Matching feature point for constructing candidate segmentation paths. [(d) and (e)] Wrong candidate segmentation paths caused by protrusions on the skeletons.

Yu and Yan [28, 29], and Kim et al. [14]. In [28], which is based on structural feature points, the first touching regions of the connected components are determined. Then, based on geometrical information of the special structural points, candidate touching points are pre-selected. Finally, after morphological analysis, partial recognition results are used for choosing cutting paths. The problem with this method is that it is very dependent on many heuristic rules based on structural features, and also it has to check for many structural features to select the appropriate ones.

Pal et al. [10] proposed a method based on a concept called water reservoir. If water is poured from the top or bottom of a numeral, regions of the numerals where water will be stored are defined as reservoirs (see Figure 10-a). In [10], they extracted three sets of features. The first set of features were extracted based on the reservoirs such as: the number of reservoirs, position of the reservoirs (top, middle, bottom), their sizes, their shapes, their centers of gravities (CG's), and their relative positions. The second set of features were topological features from closed loops such as: the number of closed loops, their positions, their center of gravities, and their relative heights. The third set of features were structural features based on the morphology of touched regions. Using a combination of these three sets of features and applying some heuristic rules, a decision was made for distinguishing isolated and connected components and building their segmentation paths. This method requires many computations for extracting a lot of structural features. Also, they will fail if there

is a breakage in the strokes of connected digits or if two digits have common portions (see Figure 10-b).

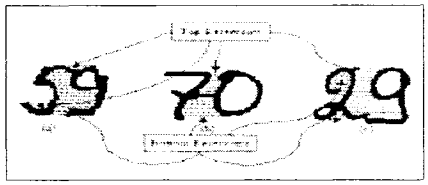
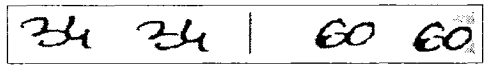
a)  b) 

Figure 10: (a) Top and bottom water reservoirs created by touching digits, (b)Some examples of errors produced by the segmentation.

Congedo et al. [30] and Punnoose [11], proposed some drop fall algorithms for segmentation, which can simulate an object such as a marble or a drop of water, falling from above the digits and sliding downward along their contours. The object falls downward until it gets stuck in a well, at which point it cuts through the contour and continues downward (see Figures 11-a, b, c, and d). These algorithms are very sensitive to starting points, and once a wrong starting point is selected they will fail to reach the touching regions (see example in Figure 11-e).

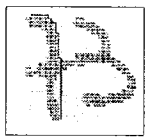
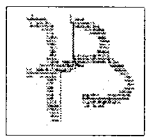
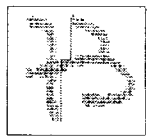
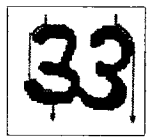
a)  b)  c)  d)  e) 

Figure 11: Drop fall algorithms: (a) Top left, (b) Bottom left, (c) Top right, (d) Bottom right, (e) Some examples of errors produced by a wrong starting point selected by the drop fall algorithm.

A similar algorithm was introduced by Shridhar and Badrelin [31], which is called Hit and Deflect Strategy (HDS). This algorithm starts at a peak of the bottom contour of the connected components and attempts to move upward. This algorithm follows a set of rules such as: whenever it hits a black pixel in the image it deflects, and it keeps changing its direction of motion until it reaches the top of the image. In fact, this method is only useful for separating not deeply touching handwritten numerals such as shown in Figure 12.

Figure 12: An example of HDS for segmentation.

Blumenstein, and Verma [32] proposed a strategy that generates a number of paths, and uses a neural network to select the correct paths. All paths are generated simultaneously using heuristic rules, and the best ones are selected by the neural network. This requires a large number of paths to be considered and they have a higher computational cost. Also, no new paths are generated in later stages so the system does not take advantage of possible feedback from the recognition module (in the latter stage). If the correct path was not generated in the original set, then the system fails. Even if the path is slightly off, confidence values will go down dramatically and the correct path will not be selected [11].

Martin [33] presented a system based on implicit segmentation for numeral string recognition which uses a sliding window to move incrementally across a string of text. A neural network attempts to recognize characters centered in the window. Ideally, each character is recognized when it gets to the center of the window, so explicit segmentation is not required. However, this approach needs a lot of computations for each position of the window.

In another system presented by Martin et al. [34], instead of sliding over the string, the window mimics the motion of the human eyes, and moves in bursts. The window moves forward, or sometimes if necessary backward in smaller jumps, called corrective jumps. These movements are controlled by a neural network, which learns to jump from one character to another while recognizing individual characters. This approach also entails exhaustive computation.

Oliveria et al. [35] presented a system based on a combination of different features for the segmentation and recognition of handwritten numeral strings. This system uses a combination of three types of features: contour, profile, and skeleton in order to build segmentation

paths (see Figures 13, and 14). This approach also entails a heavy computation.

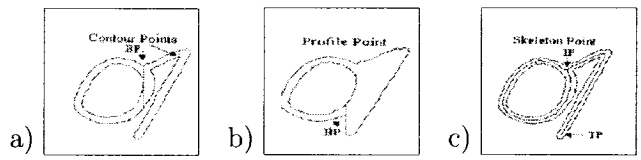



Figure 13: (a) Features from contour, (b) Features from profile, (c) Features from skeleton.

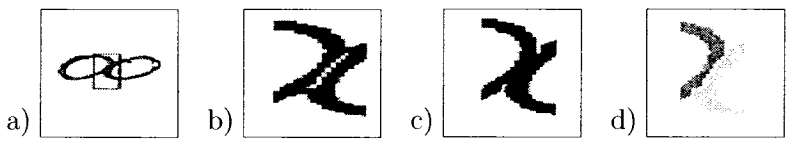



Figure 14: (a) Original image, (b), (c), and (d) Constructing segmentation path based on contour and skeleton feature points.

For the recognition of handwritten numeral strings, there have been two general approaches: segmentation-then-recognition, and segmentation-based-on-recognition [9]. In the first approach, the segmentation module provides a single sequence of partial images based on some heuristic rules, where each partial image should contain an isolated character which is submitted to a recognizer [10, 36, 31]. Since this technique does not use any contextual or recognition feedback (for segmentation), it shows its limits rapidly when the correct segmentation does not fit the predefined rules of the segmenter. In the second strategy, first the segmenter provides a list of segmentation hypotheses, and then the recognition module evaluates each hypothesis [9, 37, 38]. This approach gives better results and a higher reliability than the first one. However, it entails a higher computational cost, since it has to generate all the segmentation hypotheses and then compare their recognition results. In addition, in this latter method, the correct segmentation rate depends too much on the accuracy of the isolated digit classifier in the recognition of digits and rejection of outliers (wrongly segmented digits).

In general, we can classify segmentation and recognition methods of handwritten numeral strings according to Figure 15. As seen in this diagram, segmentation can be either explicit

or implicit. In the explicit methods, segmentation is accomplished explicitly prior to recognition in order to provide primitive segments (candidate digits) for the recognizer, such as [10, 16, 35]. In these methods, cutting paths are normally found from the contour, profile, background or foreground regions or skeletons of the string image. However, in implicit methods, segmentation is embedded in the recognition process and is performed simultaneously with recognition such as in [12, 39]. In these methods, the challenge consists of finding the best compromise between segmentation and recognition. As indicated in the literature, explicit segmentation methods for handwritten numeral strings tend to have a better performance than implicit methods [9, 2].

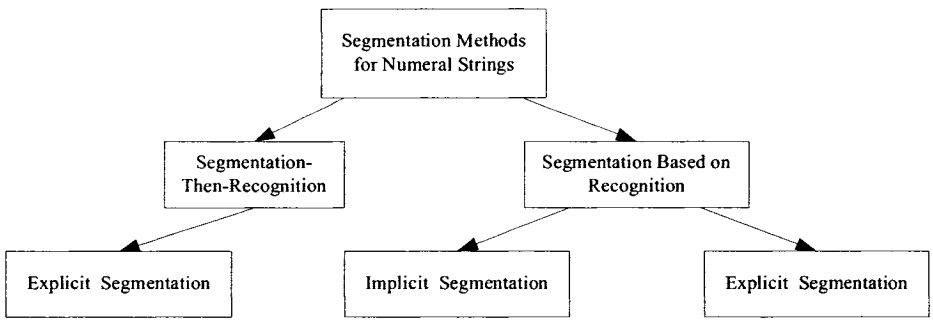



Figure 15: Classification of current methods for segmentation and recognition of numeral strings.

Similar to combining classifiers, there has been methods for combining string recognizers such as Ye et al. [40] proposed a paradigm of combining string recognizers. They introduced a frameworks for combination of multiple string recognizers. A parallel combination system, is implemented based on three independent alphanumeric handwritten string recognizers that act as black boxes. Among the different segmentation and recognition results provided by three handwritten string recognizers, they try to find the best segments based on a graph-based searching method. All factors such as the agreement of size, classification, and the position are converted into a measurement resulting in a soft decision. Their combination (so called *StrCombo*) could achieve a substantial improvement over any one of the individual recognizers. The problem with this method is the very high computational cost and we should have the original string recognizers before hand ready.

For more information on character string segmentation algorithms, see [41]. After segmentation of numeral strings and separating them into digits (or connected components) some features must be extracted for them for their recognition. In the next section, we review some of the-state-of-the-arts feature extraction methods.

## 2.3 Feature Extraction for Isolated Handwritten Numerals

Selection of a relevant feature extraction method is probably one of the most important decisions in designing any pattern recognition system for achieving a high recognition performance. For this reason, feature extraction has gained considerable attention in the field of handwritten recognition. A good survey about feature extraction can be found in [42, 43]. The literature on handwritten digit recognition shows generally three main classes of features: structural, statistical, and a combination of these two features.

Structural features are typically perceptual attributes of the character such as position and the number of their strokes, their bends, end points, branch points, loops, measures of concavity, and directional features [44, 45, 46, 47, 48, 49]. In structural handwriting recognition the characters are represented as unions of structural primitives. It is assumed that the character primitives extracted from handwriting are quantifiable, and that one can find the relations among them. Basically, structural methods can be divided into two classes: grammatical methods [31] and graphical methods [50, 51].

Statistical features are normally the results of global mathematical transformations on the image, for example, different sets of moments such as: Geometric moments, Invariant moments, and Zernike moments, statistics of chain codes, Fourier descriptors, and Wavelet coefficients (transforms) [52, 53, 54, 55, 56].

Also, in order to improve the reliability of the recognition systems many researchers have turned towards different combinations of structural and statistical features [57, 58]. Some researchers have also used deformable models [59] in order to make their systems more reliable against variations in the shapes of numerals.

Trier et al. [42] presented a comprehensive overview of feature extraction methods for off-line recognition of isolated characters. The feature extraction methods included: template matching, deformable templates, unitary image transforms, graph descriptions, projection histograms, contour profiles, zoning, geometric moment invariants, Zernike moments, spline curve approximations, and Fourier descriptors.

Liu et al. [43] compared state-of-the-art feature extraction techniques, which included the extraction of chaincode features, gradient features, profile structure features and peripheral directional features. They presented the comparisons of recognition performances among different types of these features.

Some methods such as [60, 61, 62] used the original gray-level or binary values of all pixels of the character images as features. In these methods, no transformation or explicit feature extraction involved. These values (features) are used as direct input to Neural Networks or Support Vector Machine (SVM) classifiers. In fact, in these cases, the Neural Network or the SVM can act as a feature extractor during the learning process, and all the variations among the charachter/digit shapes are handled by these classifiers.

Mitchell and Gillies [63] used mathematical morphology to extract concavity features from digits for their handwritten digit recognizers. A classification system was implemented by a symbolic model matching process.

Shi et al. [64] proposed a handwritten digit recognition system by using gradient and curvature features of the gray level images in order to improve the accuracy of handwritten digit recognition.

Krzyzak et al. [65] extracted features from the contours of numerals: 15 complex Fourier descriptors were extracted from the outer contours and simple topological features were extracted from the inner contours. These features were used as input of a three-layer Neural Network for classification.

Oliveira et al. [9] proposed specific concavity, contour-based feature sets for the recognition and verification of handwritten numeral strings.

Chen et al. [66] used a multiwavelet orthonormal shell expansion on the contour of handwritten numerals to get several resolution levels and their averages. They used the shell

coefficients as the features input and they input them into a multi-layer perceptron (MLP) neural network to recognize numerals.

## 2.4 Classification of Handwritten Isolated Digits

In this section, we review some of the state-of-the-art techniques in handwritten digit classification. Classifiers take feature vectors for each patterns as their input, then as their result they assign class labels (along confidence values) to those patterns (features). Classification is considered the next important step after feature extraction for the recognition of handwritten isolated digits. The literature shows many classification schemes, which have been applied for the recognition of handwritten isolated digits such as: Template Matching, Nearest Neighbor (K-Nearest Neighbor), Polynomial Discriminant Classifiers, Learning Vector Quantization (LVQ), Feed Forward Neural Networks such as Multi Layer Perceptron (MLP), and Support Vector Machines (SVM's) [67]. Some researchers have also used combinations of different classification algorithms (called multi-experts/multiple classifiers) in order to make recognition systems more reliable against variations in each numeral's shape [68, 69]. See [43, 59] for a good survey on classification techniques in handwritten recognition. Below, we briefly review some popular classification methods.

K-Nearest-Neighbor: In this classification method, for each pattern in the target dataset (test data set), the closest K pattern (the K nearest neighbors) in the training dataset is found. A distance measure such as Euclidean, or Hamming distance,... is used to find the neighbors. Based on the class label of the majority of these neighbors, the class label is determined for the test pattern [67, 70]. Using this approach in [71], good recognition results for handwriting recognition were reported. The problem with the K nearest neighbor classification is that it has a high computation cost and is inflexible. To avoid such a problem, some researchers in [72] proposed a faster K-NN method for handwritten recognition.

Bayesian Classifier: In Bayes classification, class prototypes are used in the training stage in order to estimate the class-conditional probability density function for a feature vector [67, 73]. Using Bayes' rule (Bayes' theorem), an unknown pattern (object) is assigned a class

label that achieves the highest (maximum) posterior probability.

Polynomial Discriminant Functions: The polynomial discriminant classifier assigns a pattern, for example, a digit to a class with the maximum discriminant value. These discriminant values are computed by a polynomial function on the components of the feature vector of the incoming pattern. Coefficients of the trained polynomial classifier implicitly represent learned class models [67, 74].

Hidden Markov Model (HMM): A Hidden Markov Model is a statistical model in which the system being modeled is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters [75]. Normally, HMM consists of a set of states and the transition probabilities between consecutive states. When HMM models are used for classification problems, and for each pattern class an individual HMM is constructed. Then for each observation sequence (each sequence of feature vectors), the likelihood for the sequence is calculated. The class in which the HMM achieved the highest probability, considerd as the winner which considered to be the class that produced the actual sequence of observations. HMM's are widely used in speech recognition and thier applications to handwritten recognition is also increasing [76, 77, 78, 79].

Artificial Neural Network (ANN) classifiers: Neural Networks due to their important advantages such as highly parallel mechanism (possible parallel implementation), excellent fault tolerance (good performance with noisy data), adaptation, and their excellent ability to learn from examples, have become increasingly popular in pattern recognition. ANN models are generally very powerful, and they have very nice theoretical properties. So, these models can be applied well to many real world applications. Different architecture of neural networks successfully have also been used in handwritten recognition [80, 81, 82, 83]. The most widely studied and used neural network in pattern recognition (and in handwritten recognition) is the Multi-Layer Perceptron (MLP) neural network. These networks can be trained based on gradient descent method, so-called the back propagation algorithm or generalized delta rule [84]. The MLP neural networks with back propagation algorithm are very powerful and they can easily be implemented. Other types of neural networks such as convolutional

neural networks are also proposed for handwritten recognition such as LeNet-5. These specialized architectures incorporate knowledge about the invariance of two dimensional objects (characters/digits) by using local connection patterns, and by imposing constraints on the weights of the network [85].

Support Vector Machines (SVM's): SVM's are introduced by Vapnik [86], and they are one of the most interesting recent developments in classification and machine learning. Because of their excellent generalization performance in learning, in the past years, SVM's have received increasing attention in machine learning and pattern recognition communities. Each SVM is basically a two-class (binary) classifier and multi-class classification is accomplished by combining multiple binary SVM classifiers. Many SVM classification systems have been developed for handwritten digit recognition and very promising results have been obtained such as [87, 88, 89], and also the method proposed by Dong et al. [90] for fast training of SVM's.

Fuzzy Logic or Fuzzy Sets: Fuzzy set models can represent degrees of truth or belonging to a set. In fact, fuzzy logic can encode imprecise knowledge and naturally maintains multiple hypotheses that result from the uncertainty and vagueness inherent in real world problems. Handwriting recognition also requires tools and techniques that recognize complex character patterns and represent imprecise, common-sense knowledge about the general appearance of characters, words and phrases, and fuzzy Logic or fuzzy Sets can provide these tools [91]. Also, combination of neural networks and fuzzy logic have been used in designing handwritten recognition systems. Neural networks can model highly nonlinear processes, and fuzzy logic can model imprecise knowledge. So, by combining the complementary strengths of neural and fuzzy approaches into a hybrid system, we can attain an increased recognition capability for solving handwriting recognition problems such as the systems in [91, 92, 93, 94].

Multiple Classifiers: Multiple classifiers or combination of classifier has attracted a lot theoretical and practical attention in pattern recognition community in the recent years. In general, the decision obtained by a combination of several classifiers seems to to be better (more reliable) than the decision obtained by the best individual classifier in the combination, however the computational cost will be higher. In handwritten recognition, there are

two main purposes for combining classifiers one is to increase recognition reliability another is to increase recognition accuracy [68, 69]. The idea of classifier combination appears in the literature under different names such as classifier ensembles [95, 96], classifier combination [69, 97], mixture of experts [98], classifier fusion [99], committees [100]. The literature shows many application of combining classifiers in handwritten recognition. Xu et al. [97] proposed four combining classifier approaches according to the levels of information available from the various classifiers. The experimental results showed that the performance of individual classifiers could be improved significantly. Huang and Suen [101, 102] proposed the Behavior-Knowledge Space method in order to combine multiple classifiers for providing abstract level information for the recognition of handwritten numerals.

Liu et al. [43] studied and compared the performance of different classifiers in isolated handwritten digit recognition. The classifiers were the k-nearest neighbor, three neural network classifiers, a learning vector quantization classifier, a discriminative learning quadratic discriminant function (DLQDF) classifier, and two support vector classifiers (SVCs). They compared the performance of these classifiers using different features such as chaincodes, gradient, profile, and peripheral direction contributivity.

Decoste and Scholkopf [103] proposed a method for the recognition of handwritten isolated digits. They incorporated prior knowledge about invariance of digits (as support vectors) under some transformations such as translation, rotation in the training procedure of Support Vector Machines (SVMs) as isolated digit classifier [104].

## 2.5 Verification Methods

Verification schemes are used in order to reduce the number of misclassifications, and also to increase the reliability of handwritten recognition systems in the classification stage. Zhou et al. [105] investigated some verification schemes, and they also proposed a verification model for handwritten numerals. They conducted experiments on both isolated and touching numerals. Their system has two layers of verification modules: class-specific verifiers and pair-wise verifiers. A class-specific verifier was designed to distinguish one class from all other

classes, for example: is the input '3'? A pair-wise verifier was used to verify the recognized characters from two different classes for example: is the input '7' or '1'?
Oliveira et al. [106] described the concept of levels of verification. They introduced two levels of verifications on handwritten numeral strings: high-level and low-level. The high-level verifier dealt with a subset of the classes in order to confirm or deny the hypotheses produced by the general-purpose recognizer. The low-level verifier dealt with meta-classes of the system (characters and parts of characters). The purpose of the low-level verifier was to determine whether a hypothesis generated by the general-purpose recognizer was valid or not.

## 2.6 Segmentation and Recognition of Handwritten Words

Although this thesis focuses on handwritten numeral string recognition and not on word recognition, here, we briefly review some literature on word recognition in order to show the important differences between segmentation and recognition of numeral strings and words. Segmentation and recognition of handwritten word is also an important topic of research in document recognition. Some applications of offline handwritten word recognition include recognition of legal amounts on bank checks, interpretation of handwritten addresses on pieces of mail, reading handwritten names/words on business forms, etc [107]. In general, handwritten words or strings may be cursive, purely discrete, touching discrete, or a mixture of these styles. Refer to Figure 16 to see some examples of these cases.

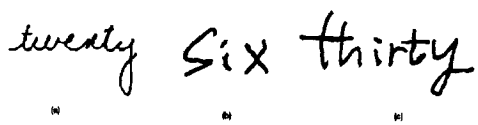



Figure 16: Examples of handwriting styles: (a) Cursive, (b) discrete, and (c) mixed.

Compared to numeral string recognition, word recognition applications normally include a lexicon which shows all the possible or meaningful words. The size of the Lexicon could be very small for example about 50 words or very large such 50,000 or more words. Although

words are normally complex patterns, and they contain great variability in handwriting styles, but handwritten word segmentation and recognition can becomes tractable when a lexicon of valid words is provided [108, 109]. The lexicon is usually dependent on the application domain. For example, in handwritten bank check processing there are exactly 33 different words that may appear in the so-called legal amount fields. However, the number of different courtesy amount values that can appear on the bank checks can be nearly infinite (in fact it depends to the maximum number of the allowed digits in the courtesy amount field, which is normally large).

From the earliest days of research in handwritten word recognition, two approaches to this problem have been identified. The first approach, often called the analytical approach, treats a word as a collection of simpler subunits such as characters, and proceeds by segmenting the word into these units, identifying the units and building a word-level interpretation using a lexicon. The other approach treats the word as a single, indivisible entity and attempts to recognize the word as whole. The latter approach is referred to as the word based or holistic approach. As opposed to the analytical approach, the holistic paradigm in handwritten word recognition treats the word as a single, indivisible object and attempts to recognize words based on features from their overall shapes. The holistic paradigm was inspired in part by psychological studies of human reading, which indicates that humans use features of word shapes such as length, ascenders, and descenders in reading; (see Figures 17 for some of the holistic features). A large body of evidence from psychological studies of reading points that humans do not, in general, read words letter by letter. Therefore, a computational model of reading should include the holistic method.

Because analytical approaches decompose handwritten word recognition into the problem of identifying a sequence of smaller subunits of individual characters, the main problems they face are:

- Segmentation ambiguity: deciding the boundaries of individual segments in the word image (see Figure 18)
- Variability of segment shape: determining the identity of each segment (see Figure 19)

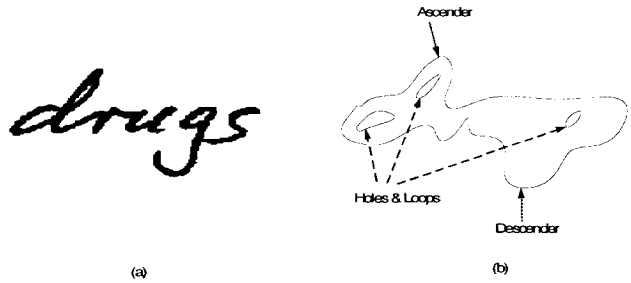



Figure 17: (a) Word image. (b) Word-shape features do not refer to individual characters and include "Length", "Ascenders", "Descenders", "Loops", etc.

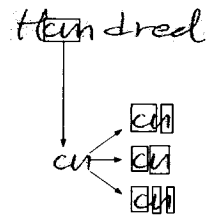


Figure 18: Ambiguities in segmentation: the letter(s) following "H" can be recognized as "w", or "ui" or "iu" or "iii".

Holistic approaches circumvent these problems in word recognition because they make no attempt to segment the word into subunits. Actually, holistic methods follow a two-step process: the first step performs feature extraction, and the second step performs global recognition by comparing the representation of the unknown word with those of the references stored in the lexicon. This scheme leads to two important practical consequences: firstly as letter segmentation is avoided and recognition is performed in a global way, these methods are usually considered to be tolerant to the dramatic deformations that affect unconstrained

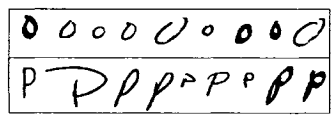


Figure 19: Large variability in shapes of handwritten characters ("O" and "P" in this example).

cursive scripts. Secondly, as they do not deal directly with letters but only with words, recognition is necessarily constrained to a specific lexicon of words.

Holistic approach has also been used for numeral string recognition. Zhou and Suen [110] presented a holistic approach for the recognition of pairs of touching digits. They considered recognition of touching pairs as a classification problem with 100 different classes, and they proposed a classifier based on error correcting codes that could handle this large number of classes. The problem with their holistic method is that it is limited to touching pairs of numerals, and it cannot be applied to unconstrained numeral strings with unknown lengths.

## 2.7 Some Results From the Literature

Here, in this section we summarize some results from the literature on segmentation and recognition of handwritten isolated numerals, numeral strings, and courtesy amounts on bank checks.

Table 2: Some results from the literature on isolated digit recognition.

| Approach and Year | Rec.(%) | Err.(%) | Rej.(%) | Database | Num. of test samples |
|---|---|---|---|---|---|
| Zhou et al. [111], 1999 | 90.0 | 1.0 | 9.0 | CENPARMI | 2,000 |
| Park et al. [112], 1998 | 98.2 | 1.8 | 0 | NIST SD3 | 53,301 |
| Ha and Bunke [113], 1997 | 99.5 | 0.5 | 0 | NIST SD3 | 173,124 |
| C. L. Liu and M. Nakagawa [114], 1999 | 98.45 | 1.55 | 0 | CENPARMI | 2,000 |
| Mayraz and Hinton [115], 2002 | 98.3 | 1.7 | 0 | MNIST | 10,000 |
| LeCun et al. [60], 1998 | 99.3 | 0.7 | 0 | MNIST | 10,000 |
| Lauer et al. [116], 2006 | 99.46 | 0.54 | 0 | MNIST | 10,000 |
| Simard et al. [117], 2003 | 99.60 | 0.40 | 0 | MNIST | 10,000 |

Table 3: Some results from the literature on segmentation of numeral strings.

| Approach | Recog. Rate (%) | Error Rate (%) | Reject Rate (%) | Database |
|---|---|---|---|---|
| Cheriet et al. [118] | 80.8 | 19.2 | 0 | 120 images of touching digits from their own collection |
| Lu et al. [1] | 97 | 3 | 28.6 | 3355 images of touching digits from NIST Database |
| Shi et al. [119] | 85.7 | 14.3 | 0 | 2579 US zipcode images from US Postal Database |
| Oliveira et al. [35] | 95.24 | 2.14 | 2.62 | 900 touching digits from Brazilian Bank Checks |
| Chen and Wang [2] | 96 | 4 | 7.8 | 4178 images of touching digits from NIST Database and 332 images from their own collection |
| Alhajj and Elnagar [120] | 97.8% | 2.2% | 0 | 4095 images of touching digits from their own collection |

Table 4: Some results from the literature on courtesy amounts recognition.

| Method | Rec.(%) | Err.(%) | Rej.(%) | Database |
|---|---|---|---|---|
| Lethelier et al. [121] | 60.0 | 40.0 | 0 | 10,000 French cheques |
| Suen et al. [122] | 62.0 | 1.0 | 37.0 | 400 Canadian cheques |
| Kaufmann and Bunke [123] | 79.3 | 20.7 | 0 | 1,500 Swiss cheques |

Table 5: Some results from the literature on numeral strings recognition.

| Approach | Rec.(%) | Err.(%) | Rej.(%) | Database |
|---|---|---|---|---|
| Britto et al. [76] | 90.33 | 9.67 | 0 | NIST NSTRING SD19 |
| Oliveira et al. [9] | 93.57 | 6.43 | 0 | NIST NSTRING SD19 |
| Liu et al. [8] | 96.74 | 3.26 | 0 | A subset of NIST NSTRING SD19 |

## 2.8 Discussion

In the present chapter, we reviewed some of the state-of-the arts techniques and their results related to the main areas of our research. By analyzing these techniques and results, we can draw some general conclusions:

- There have been a lot of progress on isolated digit recognition. Some papers reported recognition rates higher than 99% on isolated digits. However, still there is a big gap between the performance of human beings in reading on numeral strings and and machines.

- Generally the results on handwritten isolated digit recognition is higher than numeral string segmentation. This is due to the complexity of segmentation of numerals. In fact, when the problem comes to numerical string segmentation and recognition, normally performances go down due to problems such as connected (touching) digits, overlapping, and unknown length of the strings.

- Most of the papers in the literature focus on segmentation of isolated or touching digits and they do not consider segmentation and recognition of numeral strings with unknown lengths.

- It is difficult to carry out a complete comparison between different approaches, since

they have been tested on different databases and even with different number of testing samples from the same database.

- In applications such as recognition of courtesy amounts reliability of the system is much more important than recognition rate, so in these systems normally there is a higher rejection rates.

- On segmentation of digits segmentation-based recognition systems with explicit segmentation normally achieves better results than other methods.

- One of the main problems, in segmentation and recognition of numeral strings is the recognition (rejection) of outliers (over or under segmented digits). The main reason is that even rejection of outliers with ordinary classifiers is very difficult [8, 9].

## 2.9 Summary

In this chapter, we presented the state-of-the arts of the main areas related to the segmentation and recognition of handwritten numeral strings including: pre-processing, segmentation, feature extraction, digit recognition/classification, and verification. For each of these topics, we reviewed some of main important contributions in the literature. We also briefly reviewed some important techniques on word segmentation and recognition, and we highlighted the main differences between the word and numeral string recognition. In the next chapter, we will present an overview of our system and we will describe our methodology for segmentation and recognition of handwritten numeral strings.

# Chapter 3

# Overview of Our Methodology

In this chapter, a general mathematical model for the problem of segmentation and recognition of handwritten numeral strings is presented. Then, based on this mathematical model, an overview of our methodology for solving the problem of segmentation and recognition of handwritten numeral strings is shown.

## 3.1 Mathematical model

Segmentation and recognition of string images can be modeled as an optimization problem as follows: There is an input image $I$ (as shown in Figure 20), and it is assumed that $I$ contains a handwritten numeral string with an unknown length (unknown number of digits/connected components). We are looking for a segmentation or partitioning of image $I$ denoted by $S^i$ $(i = 1, \cdots, N)$, such that: $S^i$ has $m_i$ $(1 \leq m_i < \infty)$ partitions which are denoted by $s^i_j$ $(j = 1, \cdots, m_i)$, and they are subject to the following constraints:

$$I = s^i_1 \cup s^i_2 \cup \cdots \cup s^i_{m_i}, \qquad (I = \text{Input String}) \tag{1}$$

$$s^i_j \neq \phi, \qquad (\forall j = 1, \cdots, m_i) \tag{2}$$

$$s^i_j \cap s^i_k = \phi, \qquad (\forall j, k = 1, \cdots, m_i) \quad \text{and} \quad (j \neq k) \tag{3}$$

There also exists an evaluating function $(f)$, which assigns to each segment or partition $s^i_j$ of $S^i$, a segmentation-recognition score of $\nu^i_j$ and a class label of $c^i_j$. The value $\nu^i_j$ is a

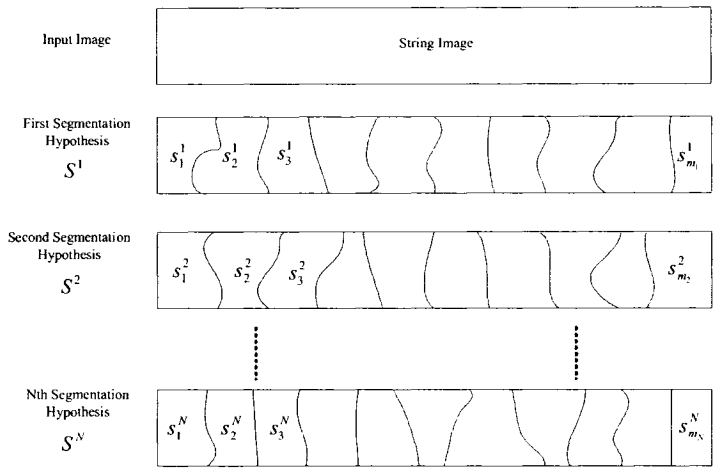



Figure 20: Illustrations of segmentation hypotheses for an input string image. $S^i$, $i = 1, \cdots, N$ shows the segmentation hypotheses or partitionings of the numeral string, and $s_k^i$ is the $k^{th}$ partition or segment of $S^i$.

measurement of the confidence (likelihood, or membership) of segment $s_j^i$ to be a valid digit in the class of $c_j^i$ as described below:

$$(c_j^i\ ,\ \nu_j^i) = f(s_j^i),\quad j = 1, \cdots, m_i,\quad \nu_j^i \in [0, 1],\quad c_j^i \in \{0, 1, 2, \cdots, 9\} \tag{4}$$

Among the set of all possible partitionings (segmentation hypotheses) of the image $I$ (shown in Figure 20), we are looking for the segmentation hypothesis or partitioning of $S^i$ which globally maximizes our objective function $F$ as follows:

$$F(S^i) \ = \ Min(\nu_1^i, \nu_2^i, \cdots, \nu_{m_i}^i) \tag{5}$$

In general, this optimization problem can be expressed as shown in Figure 21 and can be described as follows:

The segmentation or partitioning ($S^i$) which globally maximizes $F$ is called an optimal solution of the problem, and the sequence of class labels ($c_j^i,\ \ j = 1, \cdots, m_i$) corresponding

| | | |
|---|---|---|
| Maximize | $F(S^i)$, | $\forall i = 1, \cdots, N$, and $I =$ Input String |
| Subject to | $I = s_1^i \cup s_2^i \cup \cdots \cup s_{m_i}^i$, | $\forall s_j^i \in S^i$, $(j = 1, \cdots, m_i)$, $1 \leq m_i < \infty$ |
| | $s_j^i \neq \phi$, | $\forall j = 1, \cdots, m_i$ |
| | $s_j^i \cap s_k^i = \phi$, | $\forall s_j^i, s_k^i \in S^i$, $j \neq k$, and $j, k = 1, \cdots, m_i$ |

Figure 21: The general optimization model for our problem

to this optimal solution will be called the recognition result of the input handwritten numeral string. This recognition result is presented as follows:

$$c_1^i, c_2^i, c_3^i, \cdots, c_{m_i}^i \quad \text{(A numeral string with } m_i \text{ digits)}$$

In this general model, several candidate functions can be chosen for the objective function, such as those shown in Equations 6 to 9.

$$F(S^i) = Min_{j=1}^{m_i} \nu_j^i \quad \text{(Minimum Objective Function)} \tag{6}$$

$$F(S^i) = \prod_{j=1}^{m_i} \nu_j^i \quad \text{(Product Objective Function)} \tag{7}$$

$$F(S^i) = \sum_{j=1}^{m_i} \nu_j^i \quad \text{(Summation Objective Function)} \tag{8}$$

$$F(S^i) = \frac{1}{m_i} \sum_{j=1}^{m_i} \nu_j^i \quad \text{(Average Objective Function)} \tag{9}$$

In this thesis, we propose minimum objective function (Equation 6), and we try to maximize this function, in order to ensure that all the segments (partitions) of the optimum segmentation hypothesis receives enough high segmentation-recognition scores ($\nu_j^i$). Evaluation of segmentation hypothesis by combining their partial scores ($\nu_j^i$) through the minimum function also facilitates detection of the outlier patterns in those segmentation hypotheses. This will be explained in detail in Chapter 7.

## 3.2 Methodology

As seen in the previous section, we modeled segmentation and recognition of an input handwritten numeral string as an optimization problem. Therefore, in order to solve this problem, our proposed method contains several important stages. Figure 22 shows a general flowchart of these stages in our proposed system. In this section, we briefly review these stages.

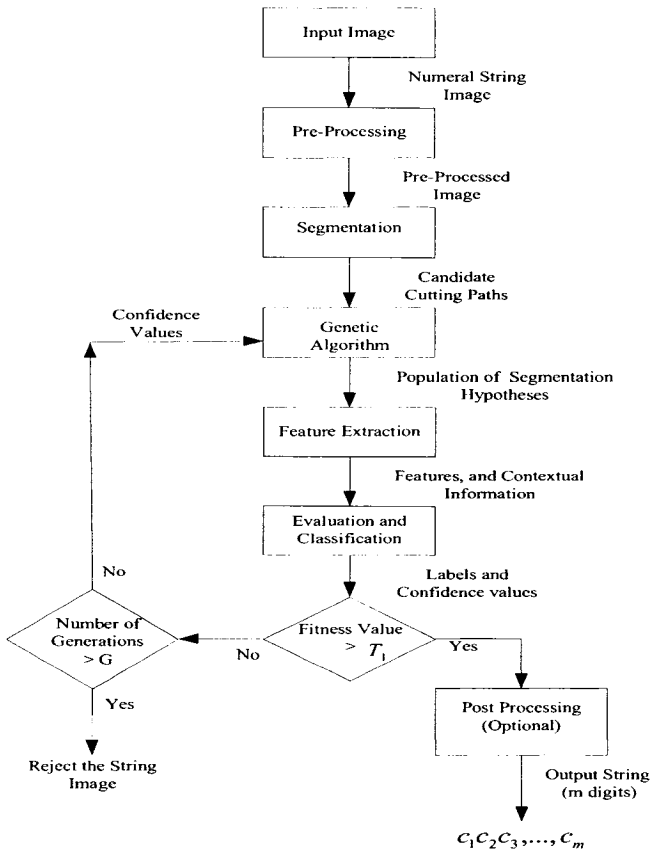



Figure 22: General flowchart of our proposed system for handwritten numeral string segmentation and recognition. $T_1$, and $G$ are two threshold values, which can be adjusted by the user of the system.

As seen in this flowchart, the first step in our system is inputting the image. During this step a handwritten numeral string with unknown length (unknown number of digits) is input into the system. In the second stage, the input image is pre-processed in order to prepare it for segmentation. The details of the pre-processing steps will be presented in Chapter 4. The next stage after pre-processing is segmentation. Segmentation is a very important step in our

system which its details will be described in detail in Chapter 5. The segmentation module in our system has two main goals: first to provide a set (super set) of best segmentation candidates (cutting paths) for over-segmentation of the input string image, second to produce less number of outlier patterns as much as possible. This module provides main building blocks (cutting paths) for generating segmentation hypotheses for the input string (as those cutting paths shown in Figure 20). After producing candidate cutting paths by the segmentation module, each combination (subset) of them can be chosen to form a possible segmentation hypothesis. In the next stage of the system, there is a search and optimization module which tries to find the best segmentation hypothesis for the input string based on some segmentation and recognition scores. In this stage, a Genetic Algorithm (GA) [124, 125] generates and searches the populations of segmentation hypotheses for the input numeral string. In fact, GA searches the space of all possible segmentation hypotheses, and tries to find the optimum hypothesis for the segmentation and recognition of the input numeral string. In the Feature Extraction stage, some features are extracted for each segment (partition) of each segmentation hypothesis. These features are used by the Evaluation and Classification module. In the Evaluation and Classification module, based on the features produced by the Feature Extraction module, all the segmentation hypotheses are evaluated. This module assigns a combination of scores to each segmentation hypothesis for its segmentation and recognition. Our GA uses these scores as confidence values to compare generated segmentation hypotheses also to find the optimum of those segmentation hypotheses. The details of our Genetic Algorithm, Feature Extraction and Evaluation method will be presented in Chapters 6 and 7, respectively.

## 3.3 Summary

In this chapter, a mathematical model for the problem of segmentation and recognition of handwritten numeral strings was presented. We modeled segmentation and recognition of handwritten numeral strings as a optimization problem. Based on this mathematical model, a general overview of our solution was shown. In the following chapters we will discuss the

details and the main stages of our system.

# Chapter 4

# Pre-Processing

Pre-processing is the first stage of processing in our proposed modular system. The goal of pre-processing is to prepare input images for segmentation and feature extraction. We assume that all the input images to the system are in binary (black and white) format, so no binarization process is involved in our system. In this chapter, two important pre-processing steps for handwritten numeral strings, which are essential in our modular system, are presented as follows: first, smoothing and noise removal, and second, slant correction. In the first step, as much as possible, images of handwritten numeral strings are smoothed and their noises are removed. In the second pre-processing step, in order to facilitate the segmentation task and also in order to make the resulting cutting paths as straight (vertical) as possible, the slants of the input numeral strings are corrected. Numeral strings are essential parts of documents such as: bank checks, postal codes (in mail), tax form. Slant correction can significantly improve their segmentation and recognition accuracy [3, 15].

## 4.1 Smoothing and Noise Removal of Handwritten Numeral Strings

Frequently, in handwritten document analysis and recognition systems, smoothing operations are applied in the initial image processing steps in order to reduce noises (clean the images) and to regularize the edges of characters, numerals, or words. In fact, by smoothing

operations, small gaps on the contours (or edges) of the characters are filled, or small bumps in their contour edges (or their edges) are removed. Normally, smoothing operations are implemented by filtering operations. Filtering is a neighborhood operation, in which the value of any given pixel in the target image (so-called target pixel) is determined by applying some algorithms to the values of neighboring pixels in the original image. In filtering methods, a pixel's neighborhood is defined as a set of pixels that are specified by their locations relative to the pixel of interest (center pixel). A neighborhood usually is defined by specifying a connectivity. There are two types of connectivity or neighborhoods (4 and 8), as shown in Figure 23. Similar to the methods in [9, 13], in our system, we use four 3 by 3 filter masks based on 8-neighborhood pixels in order to smooth the binary images of numeral strings. These filter masks are shown in Figure 24. In the rest of this section, we explain how we apply these filter masks for smoothing the input images.

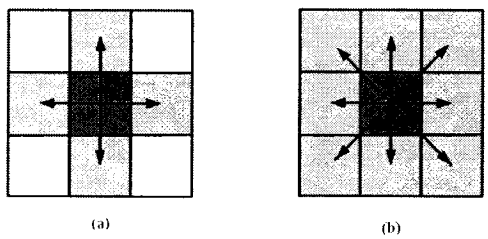



Figure 23: (a) 4-connected (or 4-neighborhood): pixels are connected if their edges touch. In other words, two pixels are neighbors if they are connected along the horizontal or vertical directions. (b) 8-connected (or 8-neighborhood): pixels are connected if their edges or their corners touch. In other words, two pixels are neighbors if they are connected along the horizontal, vertical, or diagonal directions.

The masks shown in Figure 24 are passed over the entire image in order to smooth it, and this process is repeated until no changes (removing or adding pixels) will take place in the target image. The masks are applied to the images sequentially as: a, b, c, and then d, and this process is repeated as follows: scanning of the image by each mask begins in the upper right corner of the image, and proceeds row by row moving downward. The pixel that falls in the center of each mask is considered as the target pixel. Pixels overlaid by any cell marked "X" are ignored. If the pixels under the cells marked by "=" all have the same color

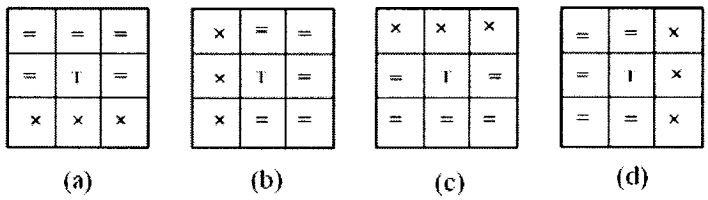



Figure 24: Examples of filter masks (3 by 3) which are used for smoothing binary document images, (b), (c), and (d) are rotated versions of the mask in (a) with 90 degrees.

(value), i.e., all are black (zeros), or all are white (ones), then the target pixel is forced to match them so that they will have the same color (value), otherwise the target pixel does not change. These masks are able to fill single pixel holes or to remove single pixel bumps in the edges of the characters or numerals, and they can remove single pixel noises (so-called salt and pepper noises). Figure 25 shows some of the result of smoothed handwritten numeral strings using these masks. As shown in these figures, artifacts such as lines that are one pixel wide can also be completely eroded from the input images.

## 4.2 Slant Detection and Correction of Handwritten Numeral Strings

After smoothing, slant correction is an important second step in the pre-processing stages of both handwritten numeral string and word recognition. By definition, the deviation of numerals, characters, or strokes from their vertical direction (which is normally found in handwritten texts), is called slant [126]. Figure 26 shows some samples of slanted handwritten numeral strings and words. The general purpose of slant correction is to reduce the variation of the script, shapes of characters and specifically to improve the quality of their segmentation candidates. These improvements, in turn, can yield a higher recognition accuracy or robustness in the automatic reading of handwritten documents. In addition, slant angle is an important feature in forensic examination and verification of official handwritten documents [127].

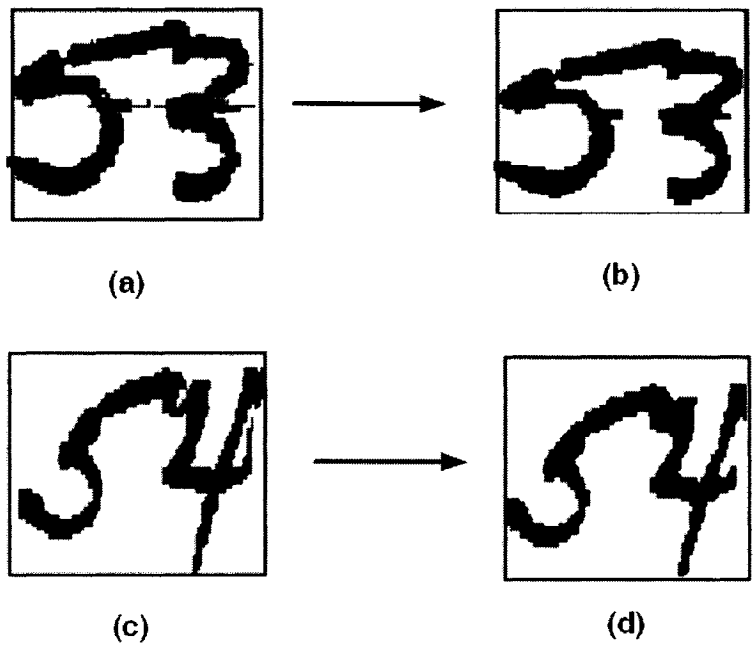



Figure 25: (a and c) original images, (b and d) smoothed images, respectively.

Many methods have already been proposed for slant detection/correction of handwritten words [126], [128]-[129]. However, there has been little research on slant correction of numeral strings. Only a few methods for slant correction of handwritten numeral strings such as [3]. or for typewritten numeral strings such as [130] can be found. Examples in Figure 26 show that digits in numeral strings are normally written separately, and compared to handwritten words (especially cursive words), numeral strings have a smaller number of touching components. Therefore, word slant correction methods are not directly or efficiently applicable to numeral strings.

Generally, methods for slant correction of words or numeral strings can be classified into two categories: uniform and nonuniform [129]. In uniform methods, the average slant angle of all the characters or components is estimated, and then uniform correction is applied to all the characters/components [126], [128], [131], [3], [130]. In nonuniform methods, a local slant angle is estimated in each horizontal position of the word, and then the slant is corrected nonuniformly in different positions or parts of the word [129]. For estimation of local slant

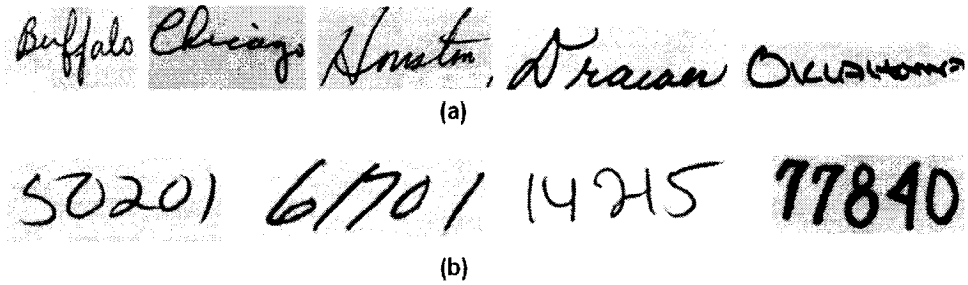


Figure 26: (a) Images of slanted handwritten words, and (b) Images of slanted handwritten numeral strings, selected from USPS-CEDAR CDROM1 Database.

angles, different techniques have also been proposed in the literature. Some techniques estimate slant angles based on the angles of non-horizontal strokes in the word [128],[132]. Others [133, 134] use projection histograms or statistics of the chain codes of contours in order to estimate the local slant angles [135],[131],[3]. Normally, these techniques of slant angle estimation are strongly based on heuristic rules or they have a high computational cost.

In this section, we contribute an efficient uniform slant correction method for handwritten numeral strings. In Chapter 8, we will compare our method with similar methods in the literature. In our method, a technique based on geometric features is utilized to estimate the local slant angles, and then a new technique is developed in order to find the average slant angle of all the connected components in the string. In the following subsections, we will describe the steps and the details of our slant correction method as follows: Component Slant Angle (CSA) estimation, String Slant Angle (SSA) calculation, and String Slant Angle (SSA) correction.

### 4.2.1 Component Slant Angle (CSA) Estimation

After smoothing the input images as explained in the previous section (4.1), component slant angle (CSA) estimation is the first step in our slant correction method. As shown in Figure 27, handwritten numeral strings are normally sets of connected components (CCs),

where each component of the string has its own slant angle and height.

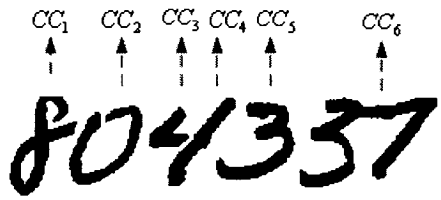



Figure 27: Handwritten numeral strings are sets of connected components (CC's) such as: isolated digits, parts of fragmented digits or touching digits.

In our CSA estimation algorithm, CC's are visited one by one from left to right (or vice versa), and for each CC a slant angle is estimated. The method in [130], which has been used for slant estimation of typewritten isolated digits, is generalized in order to estimate the slant of all CC's. Details are described as follows: First, each CC in the numeral string is circumscribed by a tilted rectangle. In other words, each CC of the string is surrounded independently by the following four straight lines ($l_1$ to $l_4$):

$$l_1: \quad y = x + \beta_1 \tag{10}$$

$$l_2: \quad y = x + \beta_2 \quad \text{where:} \quad \beta_1 < \beta_2 \tag{11}$$

$$l_3: \quad y = -x + \beta_3 \tag{12}$$

$$l_4: \quad y = -x + \beta_4 \quad \text{where:} \quad \beta_3 < \beta_4 \tag{13}$$

The details are shown in Figure 28. As seen in Figures 28-a, and in 28-b, denoted by $\theta$, a Component Slant Angle (CSA) can be estimated for each CC as follows:

$$\theta = \arctan(d/h) \tag{14}$$

The parameters $h$, and $d$, used in Equation 14, are illustrated in Figure 28, and they are explained as follows: $h$ is the height of the connected component, and $d$ is the horizontal distance from point P to point Q, where, points P, and Q are the intersections of the lines

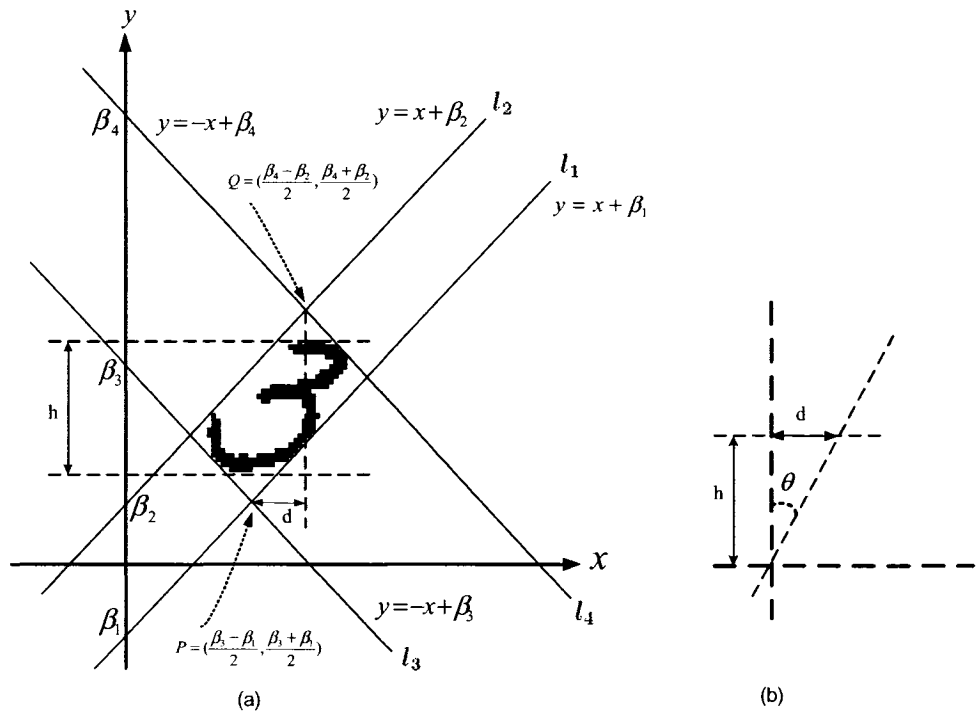



Figure 28: (a) A connected component (CC) is circumscribed by a tilted rectangle, where h is the height of the CC, and d is the horizontal distance from point P to point Q (points P, and Q are the intersections of the lines $l_1$, $l_3$, and $l_2$, $l_4$, respectively), (b) Based on the parameters h and d, a component slant angle ($\theta$) is estimated for the connected component.

$l_1$, $l_3$, and $l_2$, $l_4$, respectively. The horizontal distance from P to Q is projected on the $x$ coordinates, and it is measured algebraically as follows:

$$d = (\beta_4 + \beta_1 - \beta_3 - \beta_2)/2 \tag{15}$$

Since $h$ is the height of the CC, it is always a positive value. However, the sign of $d$ (horizonal distance) depends on the relative position of points P and Q, so it can have a negative or positive value. Therefore, corresponding to the left or right orientation of each CC, CSA ($\theta$) can have a negative or positive value, respectively. After estimating the values of CSA for all the components of the input numeral string, these values are used in order to calculate a string slant angle. The details will be explained in the next section.

### 4.2.2 String Slant Angle (SSA) Calculation

String slant angle (SSA) calculation is based on CSA values. Our observations and experiments on strokes/digits in handwritten numeral strings, and also our observations on the orientation of the straight objects, revealed that visually human beings pay more attention to the orientation of the objects which are higher than shorter. For example, people take notice more to the orientation of the Leaning Tower of Pisa than to the orientation of any tree around it, although the trees may have more of an inclination than the tower. Inspired by these observations, we defined a weighted average for the slant angle of the numeral strings (SSA's) based on the relative heights of their components, and their CSA's as follows: Assume there exists $N$ connected components in a numeral string, and also assume that the slant angle of each component ($\theta_i$, $1 \leq i \leq N$) has been estimated independently according to the method described in Section 4.2.1. A weighted average slant angle ($\Theta$) for the whole numeral string can be computed as follows:

$$\begin{aligned} \Theta &= \frac{\sum_{i=1}^{N}(h_i \cdot \theta_i)}{\sum_{i=1}^{N} h_i} \\ &= \sum_{i=1}^{N}\left(\frac{h_i}{\sum_{i=1}^{N} h_i}\right) \cdot \theta_i \\ &= \sum_{i=1}^{N} w_i \cdot \theta_i \qquad \text{where:} \quad w_i = \frac{h_i}{\sum_{i=1}^{N} h_i} \end{aligned} \tag{16}$$

In Equation 16 above, the parameters $\theta_i$, and $h_i$ are the component slant angle and the vertical height of the $i^{th}$ connected component, respectively. An illustration of $\theta_i$, and $h_i$ for each component was shown in Figure 28-b, and is shown in Figure 29-b. We denote the relative height of the $i^{th}$ component ($h_i / \sum_{i=1}^{N} h_i$) with ($w_i$), and consider it as the weight for its slant angle ($\theta_i$). We name $\Theta$ as the String Slant Angle (SSA), and we define it as the global slant angle for an entire handwritten numeral string. As seen in our method, each component of the string contributes to the value of $\Theta$ proportional to its relative height, with respect to the other components. In Equation 17, we summarize and write $\Theta$ (SSA) as a convex combination of all the components' slant angles (CSA's) in the numeral string. In this combination, components which are higher have bigger weights ($w_i$) for their slant

angles ($\theta_i$), compared to shorter components. Having calculated SSA, we use it in order to correct the slant of the numeral string. This will explain in the next section.

$$\Theta = \sum_{i=1}^{N} w_i \cdot \theta_i \quad \text{where: } 0 \leq w_i \leq 1, \quad \text{and} \quad \sum_{i=1}^{N} w_i = 1 \tag{17}$$

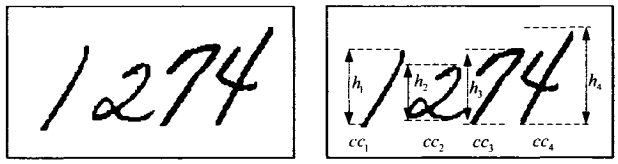



Figure 29: Illustration of the heights of the components: (a) Original numeral string, (b) Vertical heights of all the connected components (four CCs) are shown.

### 4.2.3 String Slant Angle (SSA) Correction

In order to correct the slant of a numeral string, its SSA ($\Theta$) is used. A shear transform [136] in the horizontal direction is applied to all the pixels of the string image, to shift them to the left or to the right (depending on the $\Theta$). The transformation expressions are given below:

$$x' = x - y \cdot \tan(\Theta) \tag{18}$$

$$y' = y \tag{19}$$

where $x$ and $y$ are horizontal/vertical coordinates of the pixels in the original image; $x'$, and $y'$ are the corresponding transformed coordinates. As Equations 18 and 19 show, this transformation only shifts the pixels of the image in the horizontal direction, proportional to their vertical height ($y$). If $\Theta$ is positive, the pixels will be shear transferred to the left, otherwise they will be shear transferred to the right. This transformation does not change the vertical height of the objects in the image. An example of applying our string slant correction method to a numeral string is shown in Figure 30. Several other examples will be shown in the next section.

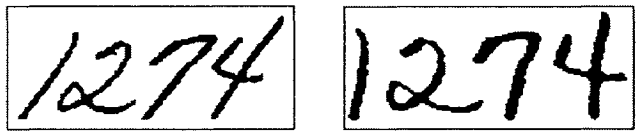



Figure 30: (a) Original numeral string, (b) Slant corrected by the proposed method.

### 4.2.4 Improving the Visual Quality by Slant Correction

In order to evaluate the effects of our slant correction method on the visual quality of handwritten isolated digits and numeral strings, 1600 images were randomly selected from four different databases (from each database we selected 400 samples), and we visually verified the result of slant correction on those samples. These databases included: Isolated handwritten digits (MNIST, and CENPARMI), and handwritten numeral strings (NSTRING SD-19, and CENPARMI courtesy amounts of bank checks). Our results are summarized in Figure 31. As shown in these figures, in nearly all the cases our slant correction method can improve the appearance or the visual quality of the images.

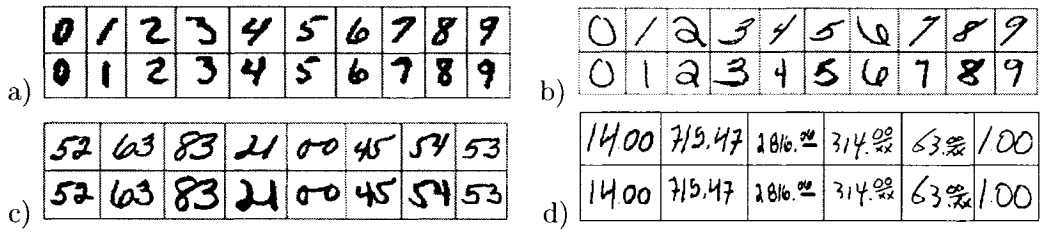



Figure 31: Top rows: Original images of numeral strings selected from different databases: (a) MNIST, (b) CENPARMI Isolated Digits, (c) NSTRING SD-19, (d) CENPARMI Courtesy Amounts. Bottom rows: Corresponding numeral strings slant corrected by our proposed method.

### 4.2.5 Statistical Characteristics of Slant Angles

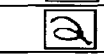
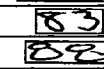
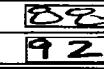
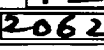
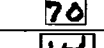
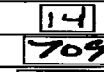
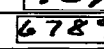
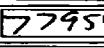
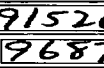
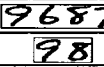
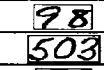
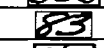
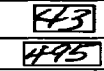
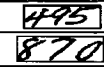
For the first time, we explore the statistical characteristics of slant angles in handwritten numeral strings, and we present some important statistics of slant angles. The method

presented in this chapter provides an efficient tool which enables us to measure and collect the SSA ($\Theta$) of many handwritten numeral strings very quickly. We conduct an exploratory data analysis on these values ($\Theta$) in order to obtain an insight into slant angels, uncover their underlying distribution, and extract some of their important statistics. Then, we look at the distribution of $\Theta$ over a very large number of handwritten numeral strings. In Table 6, slant angles of some numeral strings from the NIST NSTRING SD19 database (unconstrained handwritten numeral strings with various lengths) and from the CENPARMI database (unconstrained handwritten isolated digits) are presented and compared. In the first and third columns of this table, we show the original strings and their corresponding slant corrected strings. In the second column of this table, estimated string slant angles (SSA's) for the corresponding original strings are shown. The rows of this table have been sorted increasingly, based on the SSA's in the second column. As this table shows, when the absolute value of SSA is around 7 degrees or less than that, it is very hard to make a distinction between the original strings and their slant corrected images visually. Examples in Table 6 show that the bigger the absolute value of the SSA above 7 degrees, the easier it is to detect the slant in the string by the human eyes. In our analysis, we define slant angles with absolute values less than 7 degrees as relatively small (usually we ignore them).

In order to find the probability of occurrence (percentages) of these cases, we investigate the distribution of slant angles over a very large sample of handwritten numeral strings. In total, 8232 samples of handwritten numeral strings were randomly selected from the NSTRING SD-19 Database. These strings had different lengths from 2 to 10 digits per string, and they were written by different unknown individuals. The values of SSA's ($\Theta$'s) for all the numeral strings of this dataset (8232 samples) were computed, and their distribution was drawn as a histogram in Figure 32.

The histogram shows that the underlying distribution of string slant angles (SSA) in handwritten numeral strings is a near-normal distribution. In order to verify this conclusion, the normal probability plot of our data is shown in Figure 33. A normal probability plot is a graph of the cumulative probabilities of the data, using a specific plotting convention (special vertical scaling). This plot provides a better (graphical) test for normality than the

Table 6: Examples of digits, or numeral strings selected from NSTRING SD-19, sorted by their slant angles. Corresponding slant corrected strings are shown in the third column.

| Original String | SSA ($\hat{\Theta}$) of Original String (in degree) | Slant Corrected String |
|---|---|---|
|  | -22.66 |  |
|  | -20.73 |  |
|  | -18.03 |  |
|  | -10.45 |  |
|  | -8.53 |  |
|  | -7.82 |  |
|  | -5.66 |  |
|  | -4.37 |  |
|  | -1.2 |  |
|  | +0.7 |  |
|  | +1.4 |  |
|  | +2.04 |  |
|  | +4.04 |  |
|  | +6.05 |  |
|  | +7.21 |  |
|  | +8.22 |  |
|  | +9.46 |  |
|  | +14.60 |  |
|  | +18.01 |  |
|  | +22.71 |  |
|  | +25.65 |  |
|  | +28.87 |  |
|  | +34.74 |  |
|  | +38.53 |  |
|  | +39.02 |  |

histogram itself [137, 138]. As seen in Figure 33, the normal probability plot of our data is very close to a straight line. Therefore, according to [137], a near-normal distribution is a legitimate model to represent the distribution of slant angles.

The parameters of $\mu$ (mean), and $\sigma$ (standard deviation) of the normal distribution ($N(\mu, \sigma^2)$) in Figure 32, can be estimated as follows:

$$\mu = \frac{\sum_{j=1}^{M} \Theta_j}{M} \tag{20}$$

$$\sigma = \sqrt{\frac{\sum_{j=1}^{M} (\Theta_j - \mu)^2}{M - 1}} \tag{21}$$

Here, $M$ is the number of numeral strings in our dataset ($M = 8232$). After using the above

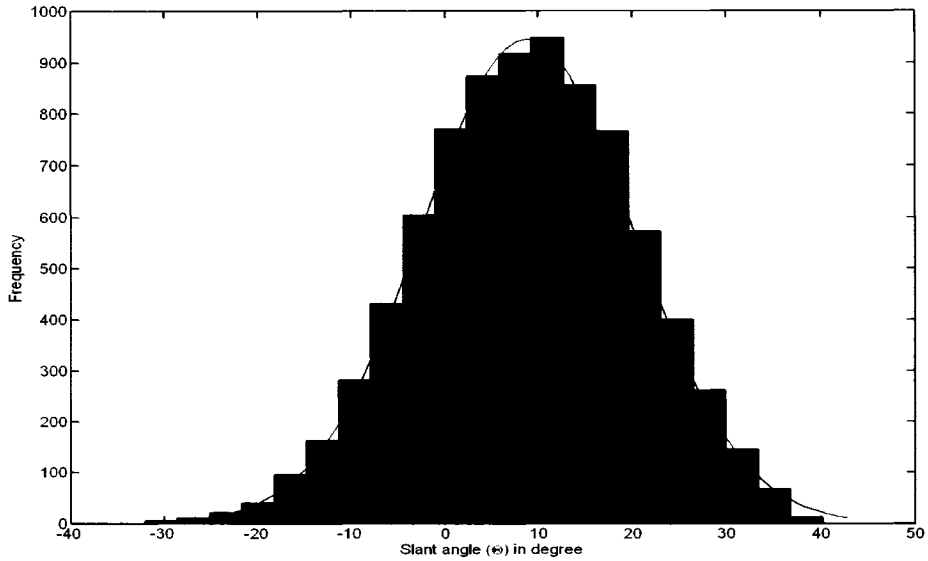



Figure 32: Histogram of the distribution of slant angles over 8232 samples of handwritten numeral strings with different lengths. This histogram shows a near-normal distribution. For comparison, it has been superimposed with a normal distribution.

estimations, we obtained $\mu = 8.7°$ and $\sigma = 11.4°$. So, the Probability Distribution Function (PDF) of SSA's over different numeral strings can be approximated as follows:

$$p(\Theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-1/2((\Theta - \mu)^2/\sigma^2)), \quad \text{where: } \mu = 8.7°, \quad \text{and} \quad \sigma = 11.4° \qquad (22)$$

As seen in Figure 32, the average value of a slant angles ($\mu$) of handwritten numerals is around 9 degrees to the right, and the range of the values of slant angle ($\Theta$) often changes from $-32°$ ($32°$ slant to the left) to $+41°$ ($41°$ slant to the right). In general, we consider the whole range of ($\Theta$) from $-90°$ to $+90°$. Using the approximation of PDF in Equation 22, we summarized some important statistics about the distribution of slant angles in handwritten numeral strings in Table 7. This table indicates that 22% of writers slant their handwritings to the left, and the rest (78% of writers) slant their handwritings (for numerals) to the right. Since the slant angles with absolute values less than 7 degrees are too difficult to detect by human eyes, we can say that about 35% of the handwritten numeral strings have very small SSA's, while the rest (about 65%) have a considerable slant to the left or to the right. This

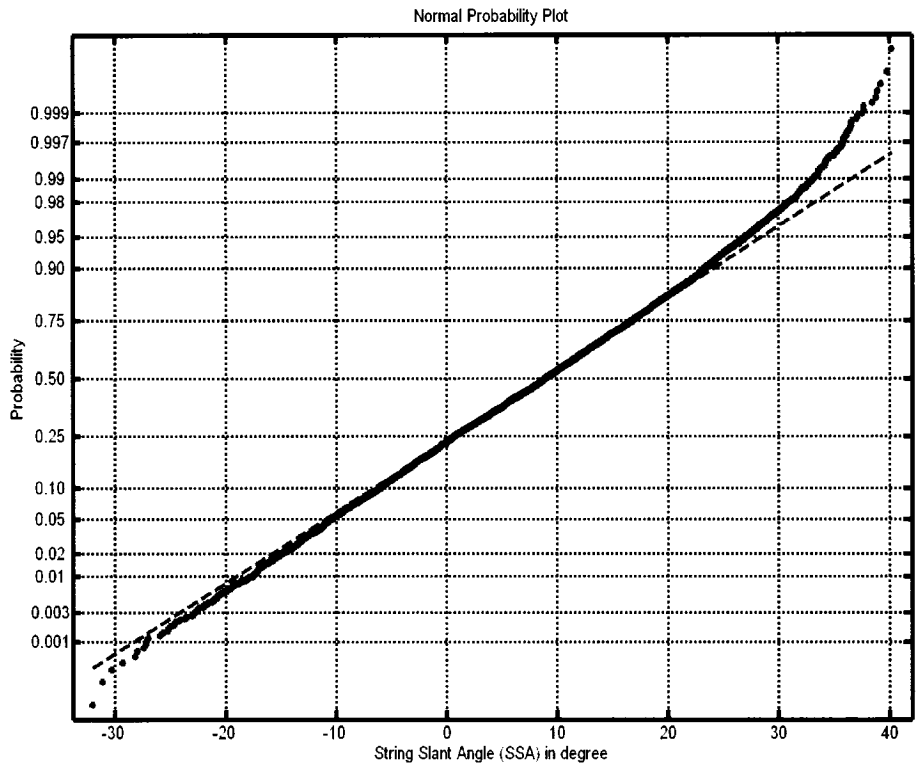



Figure 33: Illustration of a normal probability plot for the slant angles of 8232 samples of handwritten numeral strings. A normal probability plot is a graph of cumulative probabilities of the data, using a specific scale on the vertical access. If the data come from a normal distribution, the plot will appear to be near linear.

shows the importance of slant correction. In Chapter 8, we will investigate the effects of our slant correction method on segmentation of handwritten numeral strings, and we will compare our method with other methods found in the literature.

## 4.3 Summary

Pre-processing of handwritten numeral strings puts them in a better condition for further processing such as: segmentation, feature extraction, and recognition. In this chapter, two essential image pre-processing steps in our modular system were presented: smoothing (noise removal) and slant correction. Smoothing removes the small spots of noises and regularize the edges in the image. Slant correction straightens the orientation of the digits in numeral

Table 7: Some important statistics about the distribution of string slant angles (SSA) in handwritten numeral strings. (Note: $a + a' = 1$, and $b + b' = 1$).

| $\Theta$ (String Slant Angle) | $p(\Theta)$ (Probability) | |
|---|---|---|
| Slant to the left: $\Theta \in (-90°, 0°]$ | 0.22 | $(a)$ |
| Slant to the right: $\Theta \in [0°, +90°)$ | 0.78 | $(a')$ |
| Small or zero slant: $\Theta \in (-7°, 7°)$ | 0.35 | $(b)$ |
| Considerable or large slant: $\|\Theta\| \geq 7°$ | 0.65 | $(b')$ |

strings, and reduces the variation of their shapes. Here, we presented a new method for slant correction. Also, for the first time, we investigated the statistical distribution of slant angles in handwritten numeral strings, and we presented some statistics about slant angels. In the next chapter, we will present the details of our segmentation module.

# Chapter 5

# Segmentation Hypotheses Space: Generating Cutting Paths

In this chapter, an algorithm for segmentation (over-segmentation) of unconstrained handwritten numeral strings is proposed. We presented the basic idea of this algorithm (separation of pairs of touching digits) in [16]. Here, the algorithm is expanded such that it can segment numeral strings with unknown lengths. The goal of our segmentation algorithm is twofold: first, to introduce a super set of candidate cutting paths in order to over-segment input numeral strings, and second, as much as possible to produce a fewer number of outlier patterns. In fact, segmentation is a very complicated and difficult task, and there is a contradiction in its goal. For example, by introducing a set with a small number of cutting paths, there is a danger to lose or ignore some important cutting paths. On the other hand, by introducing a set with a large number of cutting paths, there is a danger to produce too many outlier samples (non-digit patterns which are produced by over-segmentation of digits). In general, outlier patterns are very difficult to evaluate or reject by isolated digit classifiers [9, 8, 17]. Also, too many unnecessary segmentations of numeral strings will increase the computational cost of the system very rapidly. So, in order to properly over-segment a numeral string, we require a set of cutting paths which gives us a high confidence that it contains all of the necessary cutting paths. At the same time, the set should not contain too many extra cutting paths which unnecessarily over-segment the components of the string. In our algorithm, in

order to reach a trade-off, a selected combination of contextual information from the string image and some local, global, foreground and background features for each component are extracted, and they are utilized to construct efficient cutting paths. The general block diagram of our segmentation module is shown in Figure 34. As seen, the segmentation module consists of two submodules: connected components analysis and splitting of touching digits. The details of the processes and algorithms in each of these submodules are described in the following sections.

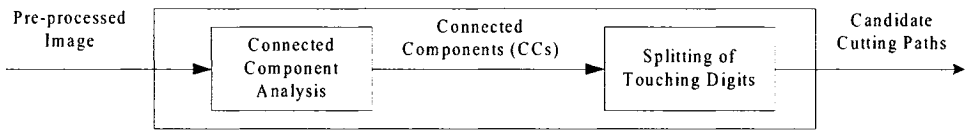



Figure 34: Block diagram of the segmentation module.

## 5.1 Connected Component (CC) Analysis

By definition, connected components (CC's) are groups of black pixels in the input image, which are connected (8-connected) [139]. Normally, numeral strings are composed of several connected components (CC's). Therefore, the first step in segmentation of a handwritten numeral string is detecting and labeling its connected components. In the CC analysis submodule, all the CC's of a pre-processed string are detected and labeled. In order to locate CC's, the algorithm starts scanning pixels of the image from left to right and top to bottom, and it finds groups of black pixels which are connected together. CCs, which are found, are ordered and labeled from left to right based on the horizontal position of their Center of Gravities (CG's). The center of gravity of each CC can be found by averaging the corresponding $x$ and $y$ coordinates of all the black pixels that belong to that CC. For each CC, its width, height and its bounding box information is also computed (as shown in Figure 35). This information will be used in subsequent stages of our modular system.

Our observations on the connected components of many handwritten numeral strings from Nstring SD-19 Database revealed that there are three possible types of connected components in a string image: pieces of broken digits, isolated digits, and touching components (or

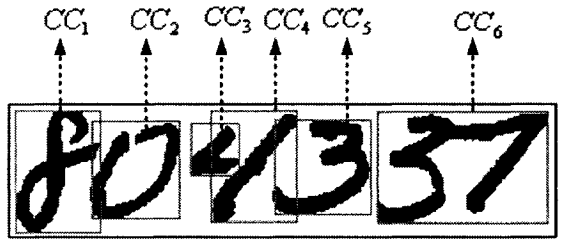



Figure 35: In the connected component (CC) analysis sub-module, all the connected components are detected, and they are labeled from left to right.

touching digits). Examples of these three component types can be seen in Figures 35, and 36. The first two types of CCs (36-a, and 36-b) do not require any segmentation; however, the third type of CCs (36-c) must be identified for segmentation (over-segmentation). In the splitting of touching digits submodule, all the incoming CCs are checked, and some cutting paths for their separation and their over-segmentation are introduced. The details of this process are explained in the next section.

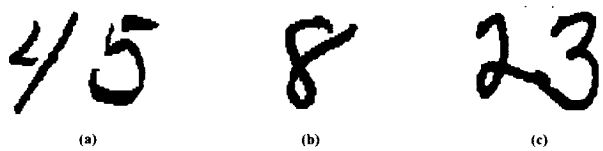



Figure 36: Three types of connected components found in numeral strings, (a) Parts of (broken) digits, (b) Isolated digits, and (c) Two (or more) touching digits/components.

## 5.2 Splitting of Touching Digits

In order to segment a handwritten numeral string, all its CCs must be separated from each other, and (if required) over-segmented by some cutting paths. Here, we call these cutting paths as candidate cutting paths. The touching digits splitting submodule is responsible for introducing those candidate cutting paths. In this section, we explain how the touching digits splitting submodule identify the locations of the candidate cutting paths, and how it constructs those cutting paths for separating and segmenting of digits.

Consider connected components of two sample numeral strings as shown in Figure 37, where the width of each CC is denoted by $w_{cc}$, and the global height of the string (height of the bounding boxes) is denoted by $H$. In observing many strings (such as those shown in this figure) from our database, we noticed that the third type of CC's (touching components/digits) normally have a longer width than the two other types with respect to the height of the string image. So, we employed a very helpful rule to avoid excess over-segmentation of the components: *If the width of a CC ($w_{cc}$) is less than 85% of the height of the image ($H$), then that CC is very likely to be a single digit (or a small piece of a digit), otherwise the CC is considered as a candidate of touching components.* Therefore, in the touching digits splitting submodule, all the CCs are checked based on this rule. CCs which are indicated as single digits or CCs which are small piece of digits, using this rule, are not over-segmented. They will only be separated from their previous or next CCs in the string. However, connected components which are identified as possible candidates of touching digits, will be over-segmented by utilizing this submodule.

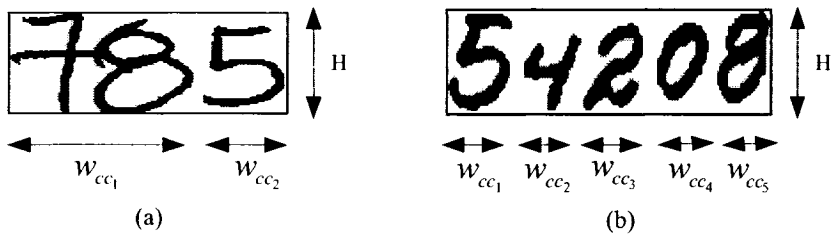



Figure 37: Connected components whose widths ($w_{cc}$) less than 85% of the height of the string image ($H$), often do not require any further segmentation. $CC_1$ in (a) needs over-segmentation, but other CCs in (a), and (b) do not require any segmentation.

As an example of applying this rule, component $CC_2$ in Figure 37-a and all the components in Figure 37-b will avoid being over-segmented. This preliminary segmentation step prevents over-segmentation of many isolated digits in the numeral strings, and it can prevent a great deal of computations. Also, this segmentation step avoids making many mistakes resulting from the over-segmentation of isolated digits. Most of the other methods in the literature (such as [8] and [9]) do not make any similar decision in their segmentation stage, so after segmentation they are faced with a lot of pieces of wrongly over-segmented digits (outliers).

Often, recognition or rejection of these pieces of digits in the later stages of the system by an isolated digit classifier is very costly or error prone. This will be explained in detail in Chapter 7

In the splitting of touching digits submodule, for segmentation of those CCs which are identified as candidates of touching digits, two types of important features are extracted: foreground features, and background features. The details of our new features, and the algorithm for construction of candidate cutting paths will be explained in the following three subsections.

### 5.2.1 Generating Foreground Features

Foreground features are those features which are extracted from the black pixels of the CC's. These features are very helpful for locating touching regions and for constructing cutting paths in CC's. In order to find foreground features, we introduce a new algorithm and a new concept called skeleton tracing. Our skeleton tracing algorithm is similar to contour tracing algorithms in [139, 140, 141], however, it traces the skeletons of 2D objects (instead of tracing their contours). Examples in Figure 38 show the difference between skeleton tracing and contour tracing. By tracing the skeletons of CC's (or any 2D objects), we can extract very helpful features from their structures, which can be used for their segmentation. Normally, it is not easy to extract these types of features by contour tracing. In the following paragraphs, we explain how our skeleton tracing algorithm works, and how this algorithm helps to extract foreground features for a CC.

In order to apply skeleton tracing to a CC, first its skeleton must be extracted by using a thinning algorithm [142, 143, 144] (see Figure 38-b). Then, on this skeleton, two points called starting and ending points (denoted by S, and E, respectively) are found, as described below. We start at the top-left corner of the skeleton image, and scan each column of the pixels from the top going downward, and we keep proceeding to the right until we encounter a black pixel on the skeleton. We declare this pixel as "the Starting point". In a similar way, we start at the top-right corner of the skeleton, and scan each column of the pixels from the top going downward, and we keep proceeding to the left, until we encounter a black

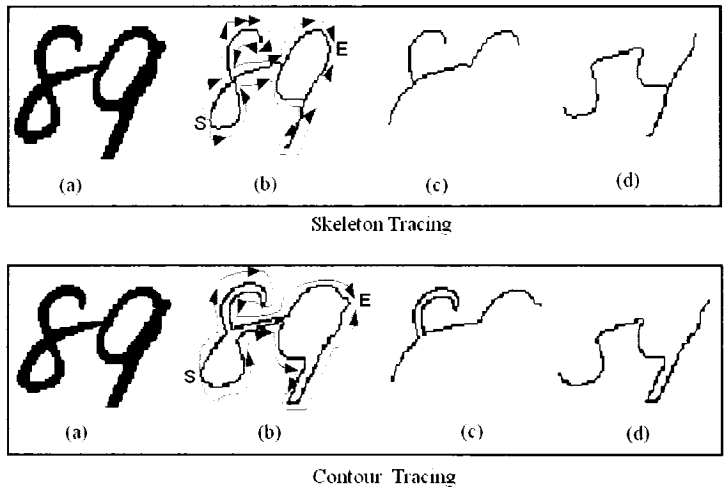



Figure 38: Skeleton and contour tracing: (a) Original image of a CC, (b) Skeleton (contour) is traversed from the starting point to the ending point in clockwise and counter-clockwise directions, (c) Top-skeleton (top-contour), (d) Bottom-skeleton (bottom-contour).

pixel. We declare this pixel as "the Ending point". Then, from the starting point (S), the skeleton is traversed in two different directions: first clockwise, and then counter-clockwise, until both traversals reach the ending point E), and they stop. We define the traversal in the clockwise direction as top-skeleton (Figure 38-c), and the traversal in the counter-clockwise direction as bottom-skeleton (Figure 38-d).

Examples in Figure 39 illustrates how we can obtain useful features of top/bottom-skeletons for segmentation of CC's. When the skeleton tracing algorithm traverses the top/bottom skeletons, it looks for intersection points (IPs), which are visited on the skeletons. IPs are points that have more than two connected branches (in Figure 39-b, they are denoted by ○). Corresponding to each visit of any IP on the skeletons, there is an angle where its bisector can be found (Figure 39-c). The intersections of these bisectors with the outer contour of the connected component are obtained, and are denoted by □ in Figure 39-d. These points are called foreground feature points. In fact, in our algorithm, bisectors map IP points on the outer contour of the connected components in order to form foreground feature points. As shown in Figure 39-d, these feature points are very close to their corresponding IP. Therefore, foreground feature points can specify important information about the touching regions and the location of touching strokes.

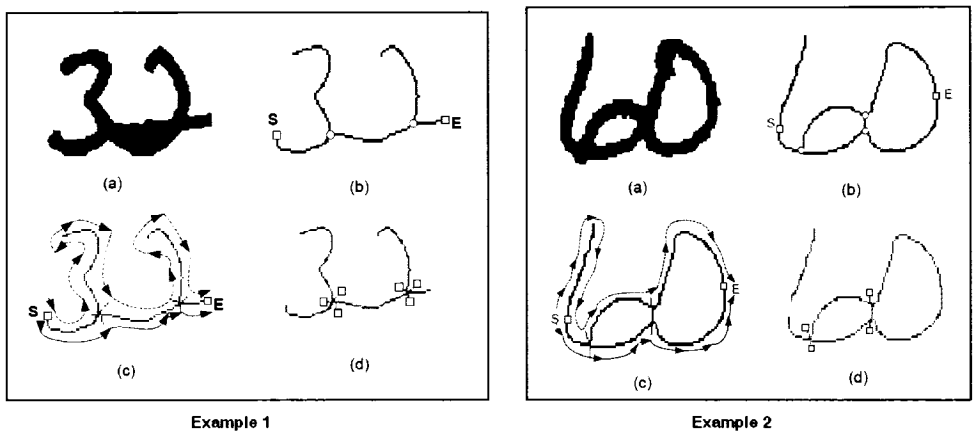



Figure 39: In both examples 1 & 2: (a) Pre-processed image, (b) Foreground skeleton, starting point (S), ending point (E) are depicted by □, and intersection points (IPs) are depicted by ○, (c) From starting point (S), the skeleton is traversed in two different directions (clockwise: dashed arrows, and counter clockwise: dotted arrows) to the end point (E), (d) Mapping of intersection points on the outer contour by bisectors to form foreground-features (denoted by □ ).

### 5.2.2 Generating Background Features

For over-segmenting of CC's, in addition to foreground features, we utilize background features. Background features are those features which are extracted from the surrounding pixels and inner pixels (those inside the holes and inner parts) of the CC's. Lu et. al. [1] and Chen and Wang [2] utilized the skeleton of the background to extract background features. In their methods, they considered all surrounding and inner pixels of CCs as background region (see Figure 40-b). However, in our method background region is obtained by combining vertical top and vertical bottom projection profiles (see Figures 40-e). In fact, we applied a new definition of the background regions for CC's, and we conducted a new method of finding background features. In our method, vertical top and bottom projection profiles of each CC are found, as illustrated in Figures 40-c and 40-d. In these profile images, black pixels are considered as the background regions. By combining background regions of the two profile images, we can show the overall background region in our method (see Figure 40-e). The skeletons of the background regions in top and bottom profile images are also extracted, which are shown in Figure 40-f and Figure 40-g, and they are called top-background-skeleton and bottom-background-skeleton, respectively.

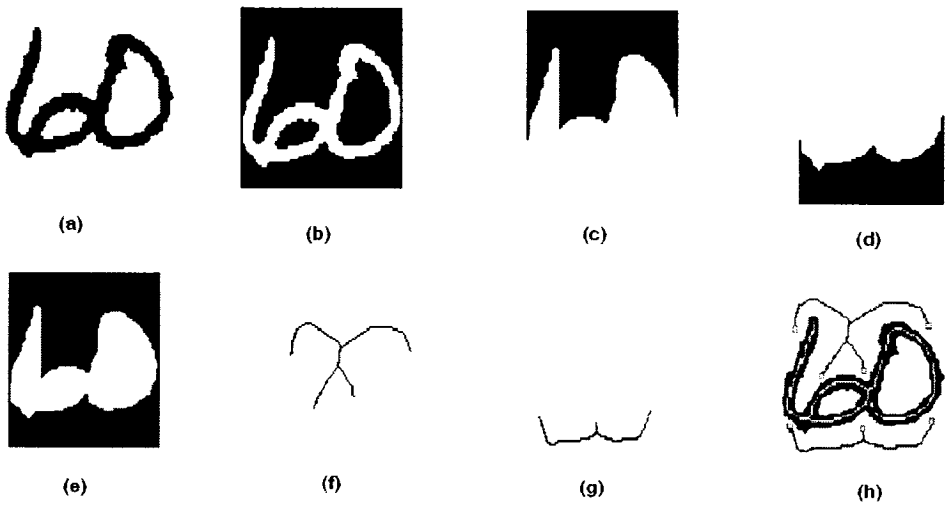



Figure 40: (a) Pre-processed image, (b) Background region in [1] and [2] (black pixels are considered as background), (c) Top projection profile of (a), (d) Bottom projection profile of (a), (e) Combining top and bottom-projection profiles in (c) and (d)(here black region is considered as background in our method), (f) Top-background-skeleton (skeleton of black region in c), (g) Bottom-background-skeleton (skeleton of black region in d). (h) Background features are denoted by □.

In order to specify background features, on each background skeleton (top/bottom), end points are found. End points are points which have only one black neighbor pixel, and they are denoted by □ in Figure 40-h. The first and the last end points of each background skeleton will not be used (they are ignored). Compared to the methods of [1], and [2], the background regions in our method only covers the essential part of the background of the CC's. Results of our experiments (Chapter 8) show that our feature points are more informative and stable with respect to the variation of the shapes of the touching digits. Also, in our method, usually a smaller number of feature points in the background regions are obtained. This facilitates decision making for the construction of cutting paths. In the next section, using both foreground and background features, we describe how to construct candidate cutting paths for touching CC's.

### 5.2.3 Constructing Candidate Cutting Paths

Using the example in Figure 41, we explain the details of the algorithm for constructing of candidate cutting paths for touching CC's. All the feature points extracted from the foreground and background, are denoted by □ in Figures 41-a, and 41-b, respectively. These feature points from top to bottom, or from bottom to top, are assigned and connected together alternatively to construct all possible segmentation paths, according to the following rule: Two feature points, A and B, are matched and assigned together if condition (23), described below, is met.

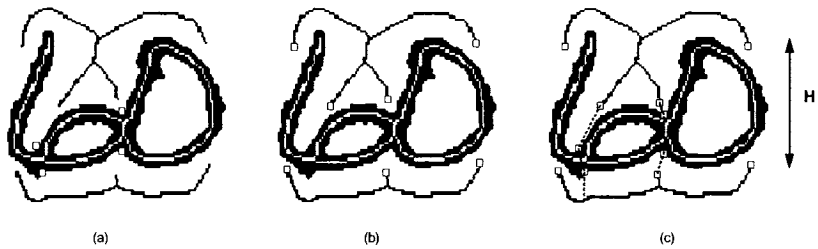



Figure 41: (a) Foreground feature points denoted by □, (b) Background feature points denoted by □, (c) Feature points from the background and foreground (from top to bottom or bottom to top) are matched and assigned together to construct possible cutting paths.

$$|x_A - x_B| < \alpha \cdot H, \quad \alpha \in [0.25, 0.5] \tag{23}$$

Here, $x_A$ and $x_B$ are the horizontal coordinates of A and B, respectively. $\alpha$ is a constant, which is selected as equal to 0.4 in our experiments, and $H$ is the vertical height of the string image. The flowchart in Figure 42 shows the details of the process of construction of segmentation paths. This process is repeated until all the cutting paths for over-segmentation of the CC's in the string are constructed.

More results of generating candidate cutting paths (for short, we say cutting paths) by our segmentation algorithm (in handwritten numeral strings) will be presented in Chapter 8, and its performance will be compared with similar algorithms in the literature. In the next section we explain how to generate segmentation hypothesis from cutting paths.

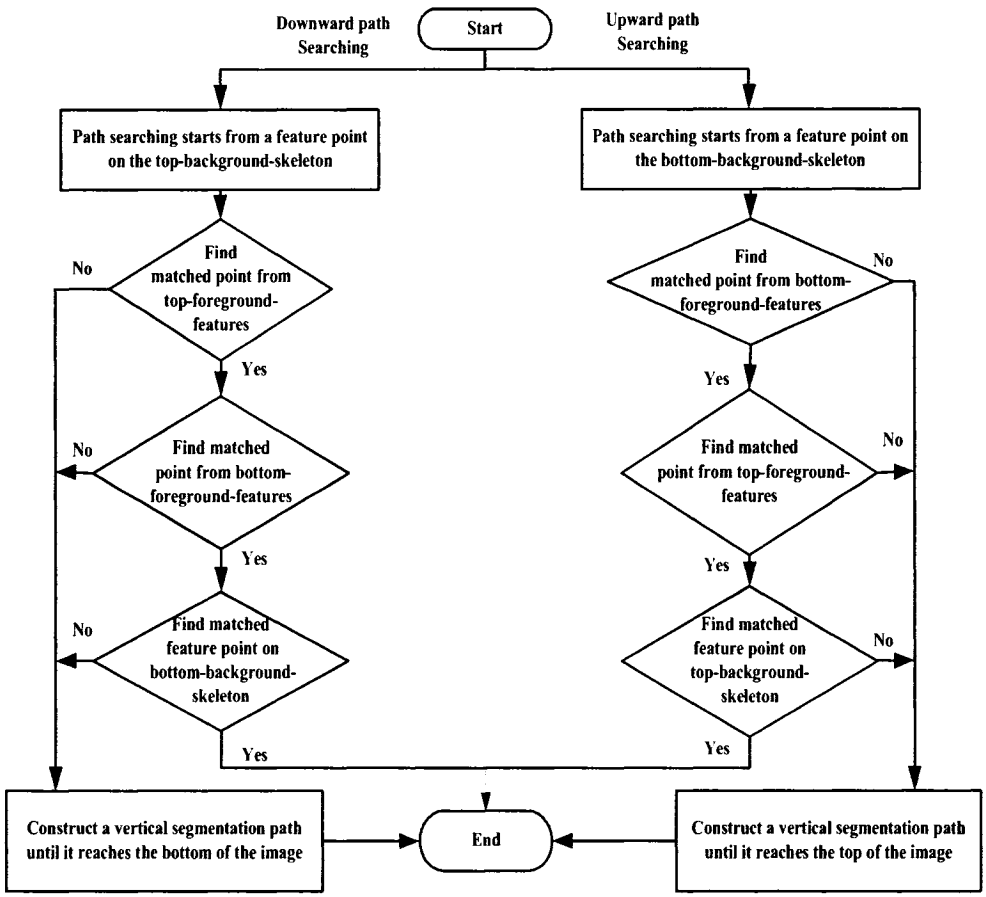



Figure 42: Flowchart of downward/upward searching for constructing segmentation paths.

## 5.3 Combining Cutting Paths for Generating Segmentation Hypotheses

After generating of all cutting paths for an input string, those cutting paths will be put together. So, a set (super set) of candidate cutting paths for that string is formed. Then, these cutting paths are ordered and labeled from left to right based on the geometric location of their centers of gravity (an example is shown in Figure 43). As shown in this example, our segmentation algorithm is an over-segmentation strategy. Therefore, its output normally provides some redundant cutting paths. Over-segmentation of a numeral string is defined successful if it can introduce a super set of candidate cutting paths containing $n$ cutting paths ($0 \leq n < \infty$) which includes the optimum cutting paths for that string (the optimum

required cutting paths for each string are not yet known at this stage). Each combination of cutting paths is called (is defined) a segmentation hypothesis. In the next chapter, we will explain how to represent all segmentation hypotheses generated from the set of candidate cutting paths, and how to search for the optimum segmentation hypothesis.

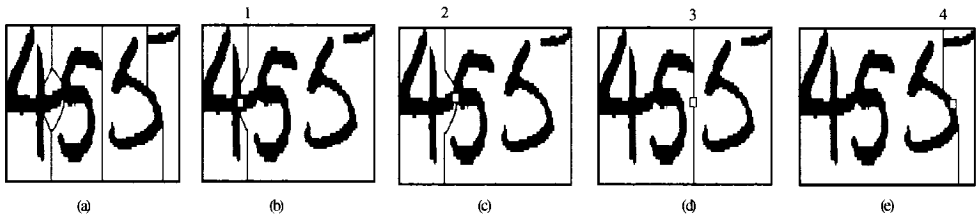



Figure 43: (a) A numeral string is over-segmented by our segmentation algorithm (here, total number of cutting paths $n$=4). (b), (c), (d), and (e) Cutting paths are ordered from left to right (based on the horizontal position of their centers of gravity), and they are labeled by integer numbers from 1 to $n$. The center of gravity (CG) of each cutting path is denoted by □.

## 5.4 Summary

In this chapter, we presented the details of our algorithm for the segmentation of unconstrained handwritten numeral strings. In our segmentation algorithm, all connected components are detected and several segmentation cutting paths for their over-segmentation are constructed. For construction of cutting paths, our algorithm utilizes a combination of foreground and background features. We presented new methods for extracting these features and also for constructing candidate cutting paths. In the next chapter, we will show how to represent and search the space of all possible segmentation hypotheses. Also, we will explain how to search the space of segmentation hypotheses in order to find the optimum hypothesis.

# Chapter 6

# Segmentation Hypotheses Space: Representation and Searching

In the previous chapter, we showed how our segmentation algorithm generates a set of candidate cutting paths in order to over-segment an input numeral string. Using this set of candidate cutting paths, in this chapter, we show how to generate and represent all segmentation hypotheses for an input numeral string. We introduce a representation and a searching method based on an evolutionary approach (Genetic Algorithm or GA) in order to find the optimum segmentation hypothesis in the space of all possible segmentation hypotheses. The details of our representation and searching (optimization) method will be presented in the following sections.

## 6.1 Representation of Segmentation Hypotheses

Efficient representation is a very important step in searching for the optimum segmentation hypothesis among all possible segmentation hypotheses for an input numeral string. Here, we describe our representation scheme and we compare it with similar methods found in the literature.

After our segmentation algorithm splits (over-segments) an input string image into a sequence of primitive images (sequence of segments), the next task is to produce all possible

sequences of these primitive images. Each sequence of these primitive images (segments) is called a segmentation hypothesis for the input string image. In order to find the optimum segmentation, the next important step is to generate and represent the space of all possible segmentation hypotheses. A graph representation is used in order to show the segmentation hypotheses. Then, segmentation hypotheses will be assessed and searched in order to find their optimum.

Segmentation graphs (candidate lattice graphs) have been widely used for representation of all segmentation hypotheses in both word and numeral string segmentation and recognition [8, 9, 145]. In Figure 44, a segmentation graph which shows all possible segmentation hypotheses for a numeral string is shown. Oliveira et al. [9] and Liu et al. [8] used segmentation graphs to represent segmentation hypotheses for segmentation/recognition of handwritten numeral strings, however, they used heuristic rules to remove or reject some of the unlikely edges (or paths). For example, in [8] researchers made some assumptions about the maximum number of connected components in a path from S to E. They used some heuristic rules for the grouping of broken components, or they put constraints on the widths or heights of the candidate patterns in each edge of the graph. Since these kinds of heuristic rules are normally biased to the training/verification sets of the system, we decided not to use any heuristic rule for rejection of the paths in our segmentation graphs, and we kept all the edges in these graphs. After some investigations on segmentation graphs of many handwritten numeral strings, we found a general representation for these graphs, and we also discovered that our graphs show certain properties. These properties have not been used (or mentioned) in the previous studies on numeral strings such as [9, 8, 2]. In this section, and in the next one, these properties will be proven in some theorems, starting with a description of our graph representation and notations.

In describing the properties of segmentation graphs, we utilize the same notations that we used in the previous chapter. For example, $n$ always shows the total number of cutting paths which have been produced by our segmentation algorithm. We also assume that the cutting paths have already been ordered from left to right and labeled by 1 to $n$. Here,

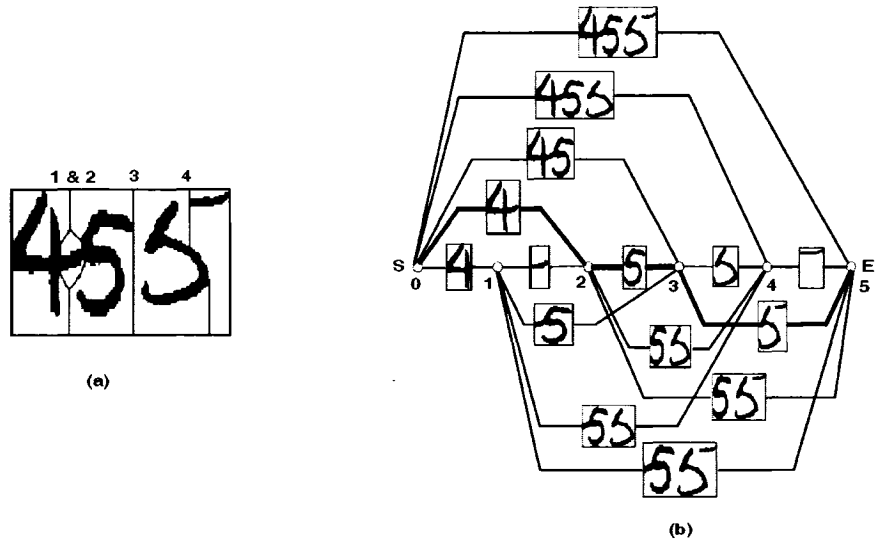



Figure 44: (a) The original numeral string and its segmentation cutting paths, (b) Segmentation graph for the numeral string in (a), where each node except for S and E, corresponds to a cutting path in (a), and each edge represents a possible segment in (a). Terminal nodes (starting and ending nodes) are denoted by S, and E, respectively. Each path from S to E is a segmentation hypothesis; the optimum path from S to E is depicted by a thick path line. In (a), there are $n = 4$ candidate cutting paths. Similarly, there are $n = 4$ non-terminal nodes, and $e = 15$ edges in the graph. The total number of segmentation hypotheses (paths from S to E) is 16.

for construction of our segmentation graph (Figure 44-b), each non-terminal node (non-terminal vertex) corresponds to a cutting path in Figure 44-a, so non-terminal nodes are assigned the same labels (integers from 1 to $n$) as their corresponding cutting paths. There are two extra nodes which are called starting (S) and ending nodes (E). Since nodes S and E do not have any corresponding cutting path, they are labeled by 0 and $n+1$, respectively. S is placed on the left, and E is placed on the right side of the graph. Edges, in segmentation graphs, represent pieces of the digits or sequences of the primitive images (CC's). In our representation, in order to build legitimate segments of the input string, each node of the graph is only connected to all of its following nodes (not to its previous nodes). Each path from S to E shows a sequence of possible segments of the numeral string; in fact each path from S to E represents a segmentation hypothesis. Therefore, the space of all possible segmentation hypotheses for an over-segmented string can be represented by the set of all possible paths from S to E in the segmentation graph. We can say, there is a one to one

correspondence between the set of all paths from S to E in the graph, and the space of all possible segmentation hypotheses of the string. In the rest of this section, we provide the proof of some general properties for our segmentation graphs.

**Theorem 6.1.1.** *Segmentation graphs are Directed Acyclic Graphs (DAG's).*

*Proof.* As explained in the construction of our segmentation graph, each node $i$ is only connected to all of its following nodes. So, according to our representation scheme a node $i$ is connected to node $j$, if and only if $i < j$ and $i, j \in \{0, 1, 2, 3, \ldots, n+1\}$. Therefore, all the edges of the graph are directed, and they start from any node $i$ and end at any node $i+1$, $i+2$, $i+3$,..., $n+1$. This means that there will be no circles, closed circuits (or loops) in our graphs. Therefore, our segmentation graphs are always Directed Acyclic Graphs. See an abstraction of a segmentation graph in Figure 45.

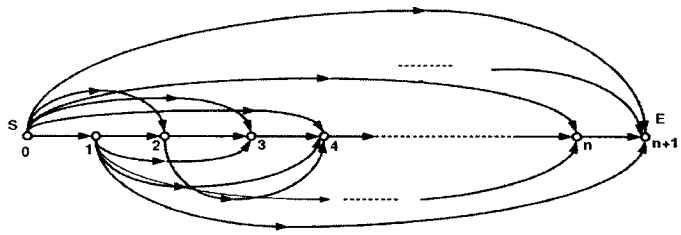



Figure 45: A general segmentation graph. Nodes 0 and $n+1$ represent the terminal nodes of S and E, respectively. Other nodes represent non-terminal nodes (corresponding to cutting paths) which are ordered from left to right and labeled from 1 to $n$.

□

**Theorem 6.1.2.** *The underlying graph of a segmentation graph is always a complete graph.*

*Proof.* According to the structure of the segmentation graphs and Theorem 6.1.1, we can say that each pair of the nodes $i$ and $j$ (when $i \neq j$) are connected such that if $i > j$ there is an edge from $i$ to $j$, and if $j > i$ there is an edge from $j$ to $i$. Since each pair of distinct nodes are connected by exactly one edge, if we ignore all the arrows on the edges in Figure 45, the remaining graph will be a complete graph. □

**Theorem 6.1.3.** *The total number of edges $e$ (the total number of possible segments) in a segmentation graph with $n$ non-terminal nodes ($n$ cutting paths) is:* $e = \frac{1}{2}(n^2 + 3n + 2) = O(n^2)$.

*Proof.* According to the structure of the segmentation graphs, they have $n$ non-terminal nodes corresponding to $n$ cutting paths, and two extra nodes corresponding to terminal nodes of S and E. So we can say that the total number of nodes in segmentation graphs ($v$) is always $v = n + 2$. Also, according to Theorem 6.1.2, there is exactly one and only one edge between any distinct pair of the nodes in the segmentation graph (underlying graph is complete). So, the number of edges ($e$) can be written as:

$$
\begin{aligned}
e &= \frac{1}{2}v(v-1) && (24)\\
&= \frac{1}{2}(n^2 + 3n + 2), \qquad \text{(since } v = n + 2\text{)} && (25)\\
&= O(n^2), \qquad \text{(Big-Oh notation)} && (26)
\end{aligned}
$$

□

The conclusion of Theorem 6.1.3 is very important, because it shows that in order to construct a segmentation graph and assign weights (evaluation scores) to all the edges of the graph, we have to invoke an evaluation function (such as a digit classifier) $O(n^2)$ times to produce the weights for all the edges (segments). In Section 6.2, we will also prove that the total number of paths from S to E in segmentation graphs, or the total number of segmentation hypotheses is always $2^n$. These facts verify the critical role of $n$ (the number of cutting paths) in the computational cost or complexity of the problem. So, if a segmentation algorithm (carelessly) produces a large number of redundant cutting paths for over-segmentation, it can heavily affect the computational cost of searching for the optimum hypothesis. In the next section, we will present a searching method which utilizes our representation scheme.

## 6.2 Searching Segmentation Hypotheses Space

Using an exhaustive search to find the optimum path (optimum segmentation hypothesis) in the segmentation graph, while $n$ is large, entails a very high computational cost, and it will be quite unfeasible. Many researchers used Dynamic Programming (DP)[146, 147] to find the optimum segmentation hypotheses in the segmentation graphs of the numeral strings. For example, in [8], authors evaluated different paths in the graph by accumulating dissimilarity measures, produced by the isolated digit classifiers, along each path. Then, they used a DP search to find the optimal path in the graph. Since lengths of the numeral strings were unknown a priori, experiments showed that the results of the search were biased towards the short strings or short paths of the graph. Authors in [8], also attempted to use the average path scores with respect to the path lengths. Since the average path scores were not monotonic, it turned out that the DP search did not find the global optimum path based on the average path scores. Their results showed that under some constraints of the evaluation scores or the corresponding objective functions (e.g. non-monotonicity), the search does not guarantee finding the global optimum segmentation hypothesis. In order to relax the constraints on the evaluation scores of the segments of numeral strings and the corresponding objective functions, in [17, 18] we proposed a Genetic Algorithm (GA), as an alternative searching strategy for finding the optimum segmentation hypothesis in long numeral stings. In the next subsections, we will present the details of our genetic algorithm.

### 6.2.1 Genetic Algorithm (GA)

Genetic Algorithms (GA's) are known as robust search techniques for solving optimization problems in complex spaces which are not continuous, or their objective functions are multi-modal (not monotonic) [124, 125]; (see Figure 46). GA's are computationally simple and at the same time powerful. They also have powerful operations such as selection or reproduction, crossover, and mutation, which can evolve the initial population of the candidate

solutions to a better population of solutions in terms of the average fitness. However, successful application of genetic algorithms into a new problem domain generally is very dependent on the information representation, operator definition, and also on the corresponding evaluation scheme for that domain. Our information representation scheme (chromosome structure) and the operator definition are explained in this chapter, and the details of our evaluation scheme will be described in the next chapter.


Figure 46: Genetic algorithms can be used for global optimization of multi-modal objective functions in very complex search spaces.

### 6.2.2 Genetic Representation Scheme

In this section, we explain how to represent all segmentation hypotheses of a numeral string by binary chromosomes. As seen in Figure 44-b, all the paths from S to E in the segmentation graph can be obtained by alternatively activating or deactivating the cutting paths in Figure 44-a (or the corresponding graph nodes in Figure 44-b). In general, for example in Figure 45, in each path from S to E, some nodes (cutting paths) are present (active) and some are absent (inactive). This implies that each path from S to E in the graph (or equivalently each segmentation hypothesis) can be represented by a binary chromosome with $n$ genes. Each gene corresponds to a cutting path or a non-terminal node in the segmentation graph. In this representation, nodes S and E are not taken into account, since they are always present in all the paths. Here, we prove a theorem about the number of paths from S to E in segmentation graphs.

**Theorem 6.2.1.** *The total number of paths from the starting point (S) to the ending point (E), in a segmentation graph with $n$ non-terminal nodes is $2^n$.*

*Proof.* As explained in our genetic representation scheme, each path from S to E in the segmentation graph with $n$ non-terminal nodes can be represented by a binary chromosome (or binary string) with $n$ genes ($n$ bits), and vice versa. So, if a gene in the chromosome is 1 then the corresponding node is present on the path, and if a gene is 0, then the corresponding node is not present on the path. A binary chromosome with $n$ genes, has $2^n$ possibilities, therefore, there are $2^n$ corresponding paths from S to E in a segmentation graph. □

**Corollary 6.2.2.** *The total number of segmentation hypotheses for an over-segmented numeral string with n cutting paths is* $2^n$.

*Proof.* Each path from S to E in the segmentation graph corresponds to a segmentation hypothesis for the numeral string, and also according to Theorem 6.2.1, the total number of such paths is $2^n$. Therefore, the total number of corresponding segmentation hypotheses is also $2^n$. □

An example of applying Theorem 6.2.1 (and Corollary 6.2.2) is shown in Figure 47, where the segmentation algorithm produced $n = 9$ cutting paths. In this example, each segmentation hypothesis is encoded as a binary chromosome with a length of $n = 9$ genes. If a gene is 1, then the corresponding cutting path is active (denoted by a solid line), so it will be considered in the segmentation of the image. On the other hand, if a gene is 0, then the corresponding cutting path is inactive (denoted by a dashed line), so it will not be considered in the segmentation of the image. Simply, we can see that the total number of segmentation hypotheses is $2^9 = 512$.

Considering the above representation scheme for chromosomes, we can say that each path (chromosome) in the segmentation graph (from S to E) contains a unique sequence of some edges (segments). During our genetic algorithm, these segments of the chromosome are evaluated and they receive weights (segmentation and recognition confidence scores). The details of the computation of weights (scores) for edges will be presented in the next chapter (Chapter 7). The weights of the edges are saved in an upper triangle matrix such as matrix $A$ shown in Figure 48, and they are reused during the evaluation of other chromosomes in our GA. The idea of saving these weights and reusing them for the evaluation of the

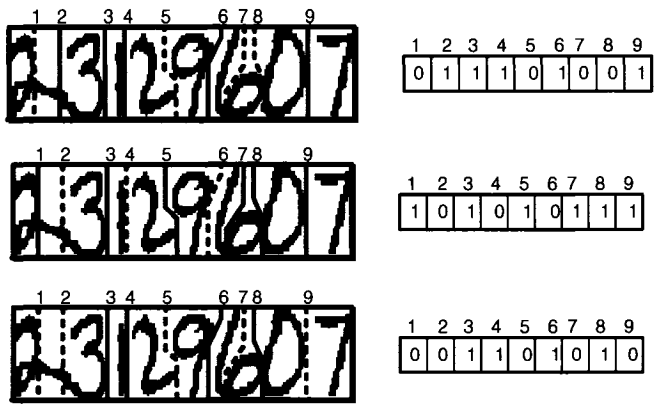



Figure 47: Each cutting path is presented by a gene in a binary chromosome (of length $n$ = total number of cutting paths produced by the segmentation algorithm). Each segmentation hypothesis is encoded as follows: dotted lines (inactive cutting paths) are not considered in the segmentation of the string, and they are encoded as zeros in the chromosome. Solid lines (active cutting paths) are coded as ones, and they are considered in the segmentation of the string. Each segment of the image is defined as a portion of the image from one active cutting path (solid line) to the next active cutting path (solid line). Each segment can contain either a valid digit or an outlier (over or under-segmented pattern).

future chromosomes (offsprings) is very similar to the idea of memoization and it avoids re-evaluation (re-computation of weights) of edges. Memoization (storing and reusing earlier results) is very general and helpful idea in optimization, and it is also the main idea of dynamic programming [146, 147]. The algorithm in Figure 49 shows the details of our memoization process. In the next subsection, we describe the design of our genetic operators in order to produce new generations of chromosomes (offsprings).

### 6.2.3 Genetic Operations

Representation of all segmentation hypotheses by binary chromosomes enables us to efficiently use all genetic operations such as: selection, crossover, and mutation [124, 125]. Figure 50 shows examples of these operations, and they are explained in the following paragraphs.

Selection is a process by which the chromosomes ranked higher in the current generation are given higher chances to act as parents for the next generation. During each iteration of GA, a proportion of the current population is selected to breed a new generation. Chromosomes

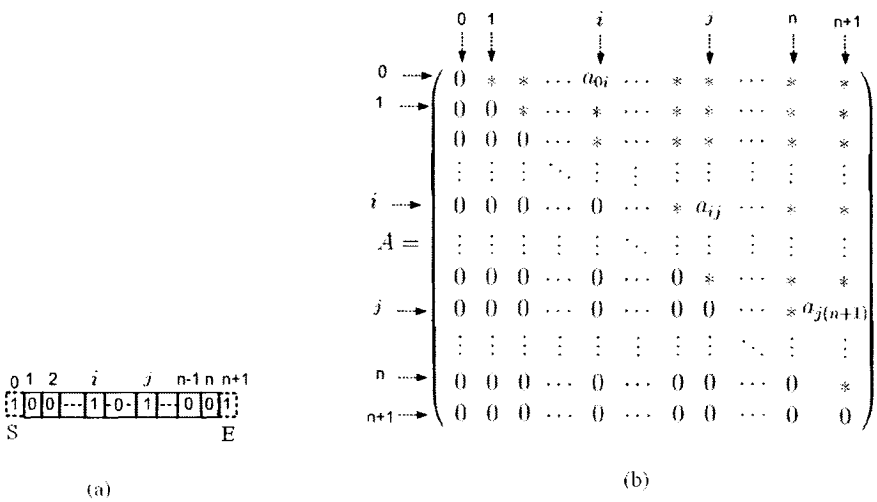



Figure 48: (a) An example of a chromosome of $n$ genes; in this example, we assume all those genes except $i$, and $j$ are equal to zero (S and E are not part of this chromosome representation and each of them is always assumed to be equal to one). So this chromosome shows a path in the segmentation graph from S to E containing three edges (segments): from node S to $i$, from $i$ to $j$ and from $j$ to E. (b) Matrix $A$ is a data structure that saves all the weights (confidence scores) of all the edges in the segmentation graph such as: $a_{0i}, a_{ij}$, and $a_{j(n+1)}$.

are selected through a fitness-based process, where fitter solutions are typically more likely to be selected. For implementation of the selection, we used a fitness-proportionate selection (based on roulette-wheel [124, 125]). The details of measuring the fitness of chromosomes will be explained in the next chapter.

The crossover operator produces two offsprings (two candidate segmentations) by recombining the genetic structure of the two parents. So, it is possible that by combining two good chromosomes, we can produce a better chromosome (solution). Crossover operation

```
Var A = An upper triangle matrix of size (n + 2)by(n + 2)
function evaluate (edge(i, j))
    if A(i, j) is empty
        A(i, j) = Compute-weights-for(edge(i, j))
    else
        return A(i, j)
end
```

Figure 49: Memoization of the weights of all segments (edges).

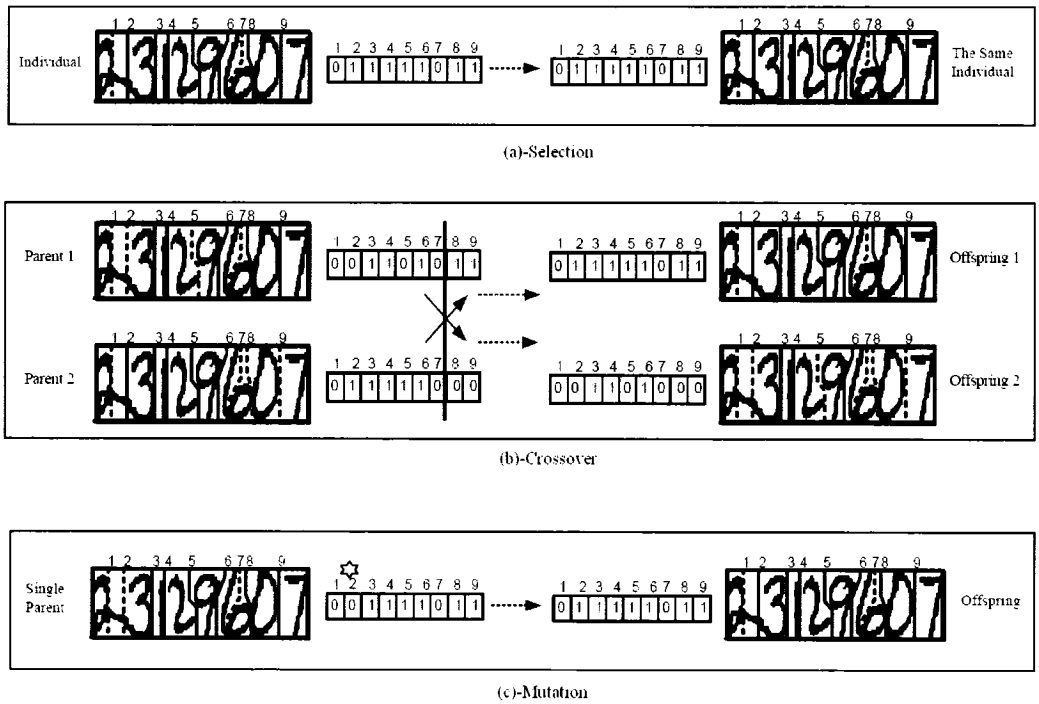



Figure 50: (a) Selection: very high fitness chromosomes are given a better chance to reproduce themselves, and to be present in the new generations or to act as parents. (b) Crossover: pairs of chromosomes are selected based on their relative fitness in order to perform crossover at a random position. In this example, crossover happens between the locations of the seventh and eighth genes in the two parent chromosomes, and it produces two new offsprings. (c) Mutation: single chromosomes are selected based on their relative fitness in order to be mutated at a single random position. In this example, mutation happens on the second cutting path (second gene in the parent chromosome), and it produces one new offspring.

can be single point or k-points. In crossover, after choosing one single point (or k points) in the parent chromosomes, the genetic information of the two parents between the crossover point(s) are swapped, and two new offsprings are generated.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next generation. Mutation is implemented by a random alteration of one or more genes in a single parent. A common method of implementing a mutation operator involves generating a random variable for one or any chosen gene in a chromosome. This random variable indicates whether or not a particular gene(s) will be altered. Mutation in GA can help the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other. In the next subsection

we give an overview of our GA approach.

### 6.2.4 Our Genetic Algorithm Approach

In our implementation, we applied very simple versions of the genetic operations, such as random single-point crossover, and a random single-point mutation on our binary chromosomes [124, 125]. As seen in the examples in Figure 50, genetic operations in general are able to improve current segmentation hypotheses and change them into new offsprings with higher fitness/confidence levels. Also, they show that the results of the genetic operations in our method will always yield valid segmentation hypotheses (valid paths in the graph). Therefore, our GA repeatedly can produce new generations (populations) of the segmentation hypotheses. In each iteration of our GA, there is a population of chromosomes (segmentation hypotheses), which is evaluated and it goes through the genetic operations in order to generate the next population. This process is continued until the highest fitness chromosome is found or the algorithm reaches the maximum number of generations; a general pseudo code of our algorithms is shown in Figure 51. For more details on genetic algorithm or genetic operators, refer to [124] and [125].

```
Generate an initial population of chromosomes.
Evaluate the fitnesses of the chromosomes in the population.
Repeat
        Select best (fitter) chromosomes to reproduce.
        Breed new offsprings, using crossover and mutation operations.
        Evaluate the fitnesses of the offsprings.
        Replace worst ranked part of the population with the offsprings.
Until terminating condition occurs.
```

Figure 51: Pseudo code of our GA.

## 6.3 Summary

In this section, first we introduced a representation scheme for segmentation hypotheses in the numeral string. We explained the application of the segmentation graph (candidate

lattice) in order to show all the possible segmentation hypotheses for an over-segmented handwritten numeral string. We also introduced a searching method based on Genetic Algorithms (GA's) in order to find the optimum segmentation hypothesis in the space of all possible segmentation hypotheses. We described our representation scheme for the chromosomes and the genetic operators of GA. Our representation scheme and the operator definition are very efficient and general. In fact, our representation scheme is able to generate (or represent) all the paths in any Directed Acyclic Graph (DAG), and our genetic operators are also applicable on all the paths of the graph. Therefore, our method can easily be applied for searching any searching space modeled by DAG's. In the next section, we will evaluate our chromosomes (segmentation hypotheses/or paths in the segmentation graphs) in order to find their optimum one(s) by GA.

# Chapter 7

# Segmentation Hypotheses Space: Evaluation

In the previous chapter, using segmentation graphs, we showed how to represent all the possible segmentation hypotheses for an input numeral string. In order to be able to compare different segmentation hypotheses and in order to find the optimum segmentation hypothesis, all those hypotheses must be evaluated (ranked). Evaluation of segmentation hypotheses (paths in the segmentation graph) is a very important and difficult stage in numeral string recognition. At this stage, we have to rank each segmentation hypothesis based on some confidence scores (measures). Due to uncertainty in the segmentation process, segmentation hypotheses may include isolated digits with different sizes or shapes, over-segmented (or broken) digits, or under-segmented (not correctly separated) digits. In the literature, over-segmented and under-segmented parts of the digits are called outliers (out-of-class or non-digit patterns). A segmentation graph which presents all the segmentation hypotheses for a numeral string was shown in Figure 44 in Chapter 6. In that figure, each segmentation hypothesis was represented by a path from starting node (S) to ending node (E). As shown in that figure, some paths contained outliers which were produced during the segmentation process. At the evaluation stage, all the paths which contain at least one outlier pattern must be rejected or ranked with very low confidence scores. In the first section of this chapter, we briefly review some of the problems with the current methods for the evaluation (ranking

or measuring the confidence) of segmentation hypotheses for numeral strings, and in the following sections, we will describe our evaluation method. In Chapter 8, we will compare our evaluation method with the current methods found in the literature.

## 7.1 Needs for New Evaluation Schemes

Unlike handwritten word recognition, in the recognition of handwritten numeral strings there is no dictionary information or linguistic context available. So graphical models such as Hidden Markov Models (HMM) [148, 149], which are frequently used in word recognition, simply cannot be used to evaluate segmentation hypotheses in a candidate lattice for a numeral string (in some applications such as zip code recognition, there are some linguistic models used, but we do not consider these special applications here). In general, it is assumed that the constituent patterns in a segmentation hypothesis (or in a path) are independent from each other. Therefore, the majority of the researchers in the area of numeral string recognition only use the outputs of single character classifiers (similarity, dissimilarity, or probability measures) and assign scores independently to each constituent pattern (edge) of a path in a segmentation graph [8, 9, 150]. They combine these scores (by product, summation, average, etc.) to find a total score for ranking different segmentation hypotheses. However, our experiments show that there are many cases where isolated digit classifiers are not able to evaluate or reject outliers in handwritten numeral strings. For example, all numeral strings in Figure 52 have been over-segmented by the segmentation module, and the resulting outliers are very similar to real digits. In these cases, it is very likely that an isolated digit classifier assigns higher confidence values to the outliers instead of to the correct candidates and misclassifies outliers as valid digits. For instance, in Figure 52-a, it is possible that the wrong candidate (020) receives a higher recognition score than the correct candidate (20), or in Figure 52-b, it is possible that the wrong candidate (100) receives a higher confidence score than the correct candidate (60) by using only an isolated digit classifier and so on.

The main reason for these kinds of mistakes can be explained as follows: isolated digit classifiers are trained to classify their inputs to one of the ten classes ($0-9$). These classifiers

(a) (b) (c) (d)

Figure 52: Relying only on the recognition scores of the isolated digit classifiers, and without contextual information, it is more likely that most of the numeral strings shown in this figure will be misrecognized with very high confidence values. For example, in (a), 20 will be misrecognized as 020, (b) 60 will be misrecognized as 100, (c) 9 will be misrecognized as 01, and in (d) 6 will be misrecognized as 10.

normally classify input digits one by one independent of the previous or the next components in their input. In other words, isolated digit classifiers normally do not pay attention to the (geometric) context of the digits/components in the numeral string images, such as relative positions, relative widths, or relative heights of the digits in the string. Therefore, if a high precision isolated digit classifier is fed with outliers, which are very similar to valid digits, it will very likely output valid digit class labels with high confidence values for those outliers, instead of rejecting them. In order to remedy this weakness of the isolated digit classifiers, and still use them with high reliability in numeral string recognition systems, we propose the use of a novel method based on contextual information which uses a new set of scores called segmentation scores. These scores can support isolated digit classifiers, and can help them to avoid making many mistakes in numeral string recognition systems. In our method, each segmentation hypothesis is assigned two types of scores: segmentation and recognition scores. In the rest of this section, we define our notations and templates, and then in the following sections, we describe the details of our segmentation and recognition scores and the method for their combination.

We use the template in Figure 53 for defining and explaining segmentation and recognition scores. This figure shows a typical segmentation hypothesis for a numeral string. This segmentation hypothesis consists of $m_i$ segmentation regions (so-called segments or partitions). We assume that each region or segment contains a valid isolated digit or an outlier. During the evaluation process, each segment is assigned two different scores: a segmentation score

and a recognition score, denoted by $s\_score$ and $r\_score$, respectively. Computation of these two scores for each region will be explained in detail in the next two sections.

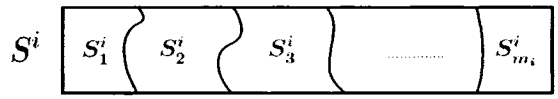



Figure 53: A segmentation hypothesis or chromosome, which is denoted by $S^i$, consists of $m_i$ regions (segments) ($1 \leq m_i < \infty$). Each segment is bounded by two consecutive active cutting paths. A combination of two scores (segmentation score and recognition score) are used to evaluate (measure the quality of) each segment in a segmentation hypothesis. By combining these scores, a total confidence value ($0 \leq \text{conf}(S^i) \leq 1$) is assigned to this segmentation hypothesis. Based on this confidence value we can rank and compare different segmentation hypothesis in order to find the best one.

## 7.2 Segmentation Scores

Segmentation scores ($s\_score$) are tools that help contextual knowledge to be taken into account in the evaluation and selection of the segmentation hypotheses. These scores are expressed as degrees of membership that show to what degree each connected component (or segment) in a numeral string can be fit into a bounding box of a valid digit. Segmentation scores are calculated based on two other component scores: position-confidence ($p\_conf$), and aspect-ratio-confidence ($a\_conf$). Computations of $p\_conf$, $a\_conf$, and $s\_score$ are accomplished according to the Equations 27, 28, 29, 30, and 31, listed below, and the parameters which are used in these equations are illustrated for the third segment (or third CC) of the numeral string in Figure 54. As seen in this figure, the position-confidence ($p\_conf$) evaluates the position-ratio ($p\_rat$) of each CC in the string image. The position-ratio ($p\_rat$) is defined by Equation 27, and it shows the relative height and vertical position of the bounding box of each CC in the string image. If a CC is very small, and its position is very close to the top or bottom of the bounding box of the image, it is assigned a low value of $p\_conf$. Similarly, aspect-ratio-confidence ($a\_conf$) evaluates the relative aspect-ratio ($a\_rat$) of a CC, which is defined here as the ratio of the width of a CC ($w_{cc}$) to the height of the string image ($H$). If the width of a CC is greater than 85% of the height of the string ($H$), it is

assigned a low value of $a\_conf$. The membership functions of position-confidence ($p\_conf$) and aspect-ratio-confidence ($a\_conf$) are plotted in Figure 55.

$$p\_rat = \frac{\max(h_t, h_b)}{H} \tag{27}$$

$$p\_conf(p\_rat) = \begin{cases} 1 & \text{if } \; p\_rat < \alpha \\ & \\ \exp(-k(p\_rat - \alpha)) & \text{if } \; p\_rat \geq \alpha \end{cases} \quad , \alpha = \tfrac{1}{3} \quad , k = 4 \tag{28}$$

$$a\_rat = \frac{w_{cc}}{H} \tag{29}$$

$$a\_conf(a\_rat) = \begin{cases} 1 & \text{if } \; a\_rat < \beta \\ & \\ \exp(-t(a\_rat - \beta)) & \text{if } \; a\_rat \geq \beta \end{cases} \quad , \beta = 0.85 \quad , t = 1.45 \tag{30}$$

$$s\_score = \min\Big(p\_conf(p\_rat) \; , \; a\_conf(a\_rat)\Big) \tag{31}$$

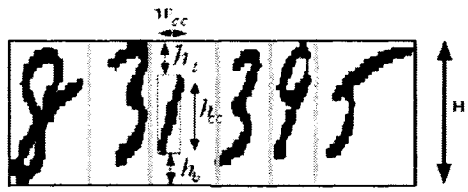
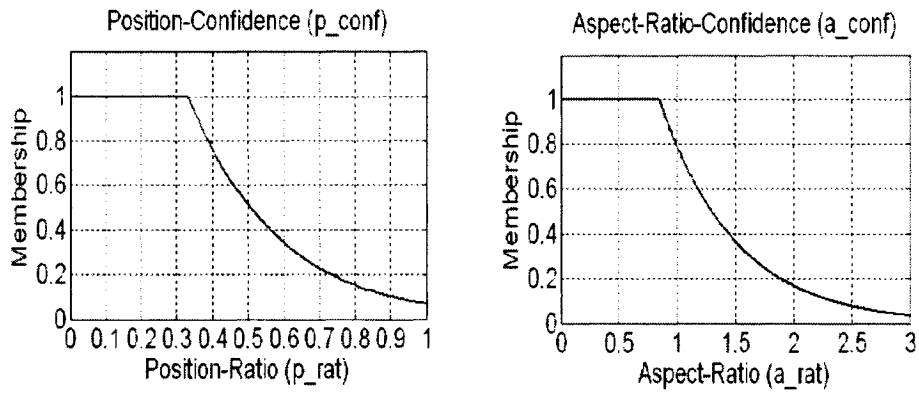



Figure 54: Illustration of parameters used in the computations of position-ratio ($p\_rat$), and aspect-ratio ($a\_rat$) for the third segment of the string. For each segment (or CC) in the image, we can compute $p\_rat$, and $a\_rat$, and then we can calculate $p\_conf$, and $a\_conf$. Finally, $s\_score$ can be computed for each segment (or CC) according to Equation 31.

Based on the definition of $s\_score$, we can verify that valid isolated digits or valid segmented digits in a numeral string will receive a much higher $s\_score$ than invalid outliers such as those shown in Figure 52. In fact, rejection of most outliers by relying only on recognition scores (like the methods in [9, 150, 8]), is very error prone. However, by using our segmentation scores, those outliers can be detected easily and rejected. In order to compute an overall




(a) (b)

Figure 55: (a) Graph of position-confidence ($p\_conf$) membership function, (b) Graph of aspect-ratio-confidence ($a\_conf$) membership function.

segmentation score for a segmentation hypothesis, after computing the $s\_score$ for all of its segments, a $total\_s\_score$ can be computed, according to Equation 32. In order to detect an outlier in a segmentation hypothesis we apply the following rule: if the $total\_s\_score$ of a segmentation hypothesis is lower than a threshold of $T_2 = 0.45$ (which is determined empirically), we can conclude that it contains at least one outlier segment. Therefore, that segmentation hypothesis will be rejected, and the recognition scores ($r\_score$) of its segments will not be computed (in fact those scores will be set to zero). This also helps to avoid a great deal of computations (which are required to evaluate/reject patterns in an invalid segmentation hypothesis). For those remaining segmentation hypotheses, of which their $total\_s\_score$ values are greater than the threshold ($T_2$), recognition scores ($r\_score$) are computed. The details of computing recognition scores will be described in the next section.

$$total\_s\_score(S^i) = \min\left(s\_score(s_1^i)\ ,\ s\_score(s_2^i)\ ,\ \ldots\ ,\ s\_score(s_{m_i}^i)\right) \tag{32}$$

## 7.3 Recognition Scores

In addition to segmentation scores, we use recognition scores to reject the remaining outliers, and also to rank the segmentation hypotheses. By using an isolated digit classifier,

recognition scores and class labels are assigned to the segments in the segmentation hypothesis. In our system, any isolated digit classifier (or a combination of classifiers), can be utilized. Achieving a very high accuracy for isolated digit classification is not the goal of our research, since this goal has already been investigated in the literature (see [43], for a survey on state-of-the-art techniques). Here instead, our goal is to improve the outlier resistance of the isolated digit classifiers, by using contextual knowledge, and to study its effect on the improvement of segmentation and recognition of numeral strings. According to [43] and [90], MLP Neural Networks (MLPs) and Support Vector Machines (SVMs) with radial basis function kernels (SVM_rbf) normally have higher recognition rates in handwritten recognition than other classifiers with the same features. Therefore, due to the efficiency in implementation and satisfaction in the performance of MLPs and SVMs with rbf kernels, we implemented our system, using these two classifiers.

The general structure of our recognition module is depicted in Figure 56. The details of preprocessing and feature extraction are shown in Figure 57. Figure 57-b shows that the slants of all the segments in a segmentation hypothesis are corrected, and the size of all the segments are normalized into a matrix of size 45 by 45 [151]. Then, for feature extraction, the skeleton of each normalized segment is taken [142](Figure 57-c), and it is divided into 15 by 15 zones, such that each zone is a window of 3 by 3 pixels. If there is at least one black pixel in a zone, its center pixel is set to black; otherwise its center pixel will remain white. Then, all the pixels in a zone except for the center pixel will be removed. Two examples of this transformation are shown in Figure 57-e, and in Figure 57-f. Since different arrangements of the black pixels inside a zone (a window of 3 by 3) are replaced by just a single black or white pixel in the center, this transformation can greatly reduce the variations of the pixels in the skeletons of handwritten digits. Afterwards, the basic shapes of the digits are extracted, as in Figure 57-d. This image has 15 by 15 pixels, which is considered as a feature vector, and it is fed into the classifiers (MLP or SVM) for classification.

Here, we briefly describe the structure of our classifiers, while the details of their training are presented in Section 8.4. Our MLP classifier has 3 layers (225 neurons in the input layer, 85 neurons in the hidden layer, and 10 neurons in the output layer), and all the neurons in

our network are sigmoidal neurons [84]. By using the back propagation learning algorithm [84], our MLP classifier is trained to produce labels and confidence values in the range of 0, and 1 for the segmented digits. For constructing our SVM classifier, we use 10 binary SVMs, one for separating each digit class from the other classes (so-called one-against-all strategy). Each binary SVM classifier uses a radial basis function kernel, and it produces an algebraic output as follows: a positive value corresponds to the target class samples (+1), and a negative value corresponds to the non-target class samples (−1). Similar to the methods in [150], and [152], these output values are mapped into confidence scores in the range of 0, and 1. The sigmoid function in Equation 33 is used for this mapping, where $f(x)$ is the output of a binary SVM for the feature vector $x$, and $g(x)$ is the corresponding confidence value ($\in (0, 1)$).

$$g(x) = \frac{1}{1 + \exp(-f(x))} \tag{33}$$

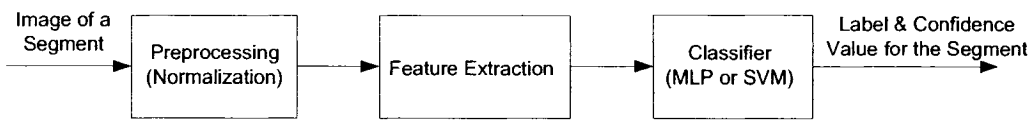



Figure 56: The general structure of our recognition module. After pre-processing and feature extraction, a trained classifier assigns class labels and confidence values to each segment in the segmentation hypotheses.

As mentioned above, each of our classifiers (MLP or SVM) has 10 outputs, corresponding to 10 classes ($0 - 9$), and each output produces a confidence value between 0, and 1 (so-called recognition score, denoted by *r_score*). For each classifier, the class which receives the highest recognition score is the winner, and it determines the label for the input segment. Recognition scores of all the segments in a segmentation hypothesis are combined based on Equation 34, in order to produce a *total_r_score*. Having a low *total_r_score* for a segmentation hypothesis indicates that it contains at least one outlier segment, or a segment which our classifier cannot recognize very well.

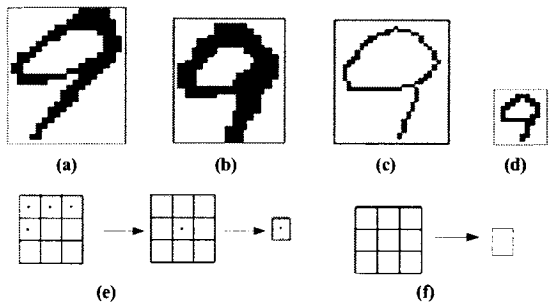



Figure 57: An example of pre-processing and feature extraction: (a) Original image, (b) Pre-processed, slant corrected, and normalized image (45 by 45 pixels), (c) Skeleton of part b, (d) Reducing the resolution of the skeleton in horizontal and vertical directions by 1/3; the resulting image is considered as a feature vector, (e, and f) Two examples of the transformation which is used to reduce the variability of the pixels on the skeleton. In (e), all black pixels inside the windows are represented by a single pixel in the center of the window; In (f), all white pixels surrounding the center pixel are removed.

$$total\_r\_score(S^i) = \min\left(r\_score(s_1^i)\ ,\ r\_score(s_2^i)\ ,\ \ldots\ ,\ r\_score(s_{m_i}^i)\right) \tag{34}$$

## 7.4 Overall Confidence Score

In the two previous sections (7.2, and 7.3), we showed how to compute the *total_s_score* and *total_r_score* for each segmentation hypothesis. By combining these two scores, we are able to compute a confidence value for each segmentation hypothesis ($S^i$) based on Equation 35, below. This confidence value expresses the total segmentation-recognition confidence for a segmentation hypothesis (or a chromosome $S^i$). It is used by our genetic algorithm to evaluate the individuals in the population of segmentation hypotheses. Having a low confidence value for a segmentation hypothesis indicates that it contains at least one segment with a very low *s_score*, or a very low *r_score* (or both), and in all these cases that segmentation hypothesis must be rejected. In fact, this confidence value is a measure of quality for segmentation hypotheses. Based on this measure, we can rank and compare (evaluate) different segmentation hypotheses in order to find the best and optimum ones.

$$Conf(S^i) = \min\left(total\_s\_score(S^i)\ ,\ total\_r\_score(S^i)\right) \tag{35}$$

## 7.5 Summary

In this section, we presented a new evaluation method for segmentation hypotheses of a numeral string. Our evaluation method is based on a combination of two sets of scores: segmentation and recognition scores. Segmentation scores enable us to take into account the contextual information from the string image in the evaluation of segmentation hypotheses. Our recognition scores is based on confidence scores from isolated digit classifiers. For each segmentation hypothesis these two scores are computed and combined in order to produce a total confidence score (fitness for a chromosome) which is used by our searching algorithm (GA).

# Chapter 8

# Experimental Results and Discussion

In the previous chapters, we introduced our modular system for segmentation and recognition of handwritten numeral strings. Our system has several important modules including: pre-processing (slant correction), segmentation, genetic algorithm, evaluation and classification. The details of the algorithms for each of these modules were described in the previous chapters. In this chapter, we show our experimental results with these algorithms, and we compare them with similar algorithms in the literature. Also, we present the overall performance of our system on segmentation and recognition of handwritten numeral strings and we compare it with the similar systems. Finally we present our discussion and conclusion. All of the test images used for the experiments in this chapter are taken from two standard handwritten databases as follows: NIST NSTRING SD19 (unconstrained handwritten numeral strings) and CENPARMI (unconstrained handwritten isolated digits). These standard databases have been frequently used for testing and comparing the performances of different algorithms (or systems) on handwritten numeral string segmentation and recognition [8, 9, 43, 76].

## 8.1 Comparison of Slant Correction Methods

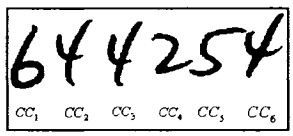
Slant correction is a very important step in the pre-processing module of our system (presented in Chapter 4). Our slant correction method is a uniform slant correction method.

Here, first we compare our slant correction method (as a uniform method) with nonuniform slant correction strategies, and then, we compare the running time of our slant correction method with a similar uniform slant correction method found in the literature. For our comparisons in this section, we randomly selected 8 sets of 125 numeral strings (in total 1000 strings); each set contained strings with different lengths (one set of touching digits and 7 sets of strings with lengths $1, 2, 3, 4, 5, 6,$ and 10 digits). These strings were taken from the NSTRING SD-19 database .

### 8.1.1 Comparison of Our Method With Nonuniform Methods

In nonuniform slant correction methods, for each part of the string (CC) a slant angle is estimated, and its slant is corrected independently [129]; however in uniform methods, the average slant angle of all the CC's is calculated, and then uniform correction is applied to the whole string. Here, we compare these two methods. In Figure 58, two samples of numeral strings from our data set are shown, where for each string, all of the CCs and their corresponding slant angles and heights are listed. According to our method, SSA ($\Theta$) has also been calculated for each of these two strings. As seen in Figure 58-a and b, in $string_1$, component $CC_4$ has a slant to the left and $CC_5$ has a slant to the right. If we correct the slant of $string_1$ nonuniformly, these two components will move towards each other (because of shear transform), and as they become closer, they may even touch or overlap. In Figure 58, another string ($string_2$) is shown. In $string_2$, component $CC_2$ nearly has no slant, and $CC_3$ has a large slant angle to the right. Therefore, during nonuniform slant correction, $CC_3$ will move to the left towards $CC_2$ (because of shear transform), and their distance will become smaller; again there is a possibility for touching or overlapping digits. The results of nonuniform slant correction of these two strings, along with many other numeral strings are illustrated in Figure 59, and they are compared with the results of our method (as a uniform method). As this figure shows, nonuniform slant correction of both $string_1$, and $string_2$ will produce touching components. As examples in Figure 59 show, in nonuniform slant correction of numeral strings, sometimes the distance between components becomes much smaller, which yields touching or overlapping digits, and sometimes the distance between

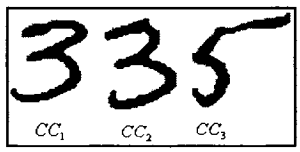
components becomes unexpectedly large. Also, when broken digits or fragments exist in the strings, nonuniform slant correction can produce distortions in the strings.




(a) String$_1$

| $CC_i$ | $\theta_i$ | $h_i$ (in pixels) |
|---|---|---|
| $CC_1$ | 8.07° | 67 |
| $CC_2$ | 8.13° | 70 |
| $CC_3$ | 12.04° | 68 |
| $CC_4$ | −7.94° | 43 |
| $CC_5$ | 27.01° | 52 |
| $CC_6$ | 21.03° | 69 |
| SSA | $\Theta = 12.04°$ | — |

(b) CCs, and their corresponding slant angles and heights for string$_1$




(c) String$_2$

| $CC_i$ | $\theta_i$ | $h_i$ (in pixels) |
|---|---|---|
| $CC_1$ | 10.84° | 47 |
| $CC_2$ | 0.6° | 52 |
| $CC_3$ | 28.01° | 54 |
| SSA | $\Theta = 13.42°$ | — |

(d) CCs and their corresponding slant angles and heights for string$_2$

Figure 58: Two examples of original handwritten numeral strings are shown, where for each string, all of the CCs, and their corresponding slant angles and heights are listed. For each string, SSA ($\Theta$) has been calculated.

### 8.1.2 Comparison of Our Method With Another Uniform Method: Running Time

Here, we compare our slant correction method with another uniform method. Method [3] is the only slant correction method that was found in the literature for slant correction of handwritten numeral strings. This method uses statistics of chain codes of contours of CC's (digits) in order to estimate slant angles ($\theta_i$'s) in numeral strings. Our experiments on 1000 numeral strings showed that both our method and [3] produced very similar slant corrected numerals without any distortions, however there was a big difference between their running time, which is shown here. The average running time of these two methods for numeral

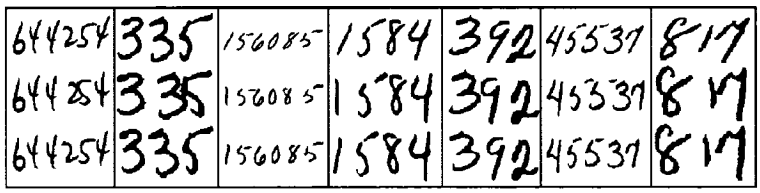


Figure 59: Comparison of nonuniform and uniform slant corrections (our method). For each string, top row: original image, middle row: nonuniform slant corrected, bottom row: uniform slant corrected (by our method) are shown. Nonuniform correction of the slants of CC's in numeral strings can yield many distortions such as touching cases and overlapping CC's. The first two columns show the results of $\text{String}_1$ and $\text{String}_2$ from Figure 58 , respectively

strings with different lengths is compared in Figure 60-c. In the next paragraph, we analyze the time complexity of these two algorithms.

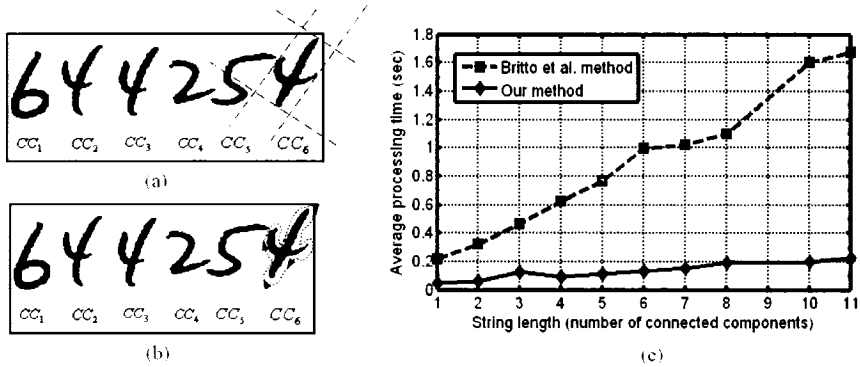



Figure 60: (a) and (b) Two examples of slant estimation for $CC_6$: (a) Our method, (b) Method in [3], (c) Comparison of the average running time between our slant correction and method in [3] for all numeral strings in our data set (1000 strings from NSTRING SD-19 database). Experiments showed that, on average, our method is around 6.5 times faster than the method used in[3].

If we assume that all the CCs in the strings have exactly the same shape, size, and orientation (strings such as $99\cdots99$ or $44\cdots44$), under this assumption, analysis of both methods shows that both will have a linear time complexity with respect to the number of CCs in the strings (running time will be a linear function of string length). However, in practice, CCs in the strings have different shapes, sizes, and orientations, so as a result of our experiments shown in Figure 60-c, the running time of these two algorithms is not exactly linear with respect

to the string length. Also, in our algorithm, the slant angle of each CC ($\theta_i$) is estimated by finding its bounding box (circumscribing the CC by four straight lines as shown in Figure 60-a). It can be verified that this process takes a very short and nearly constant time for all the connected components in the strings. However, in method [3], the values of $\theta_i$ for CC's are estimated by tracing their contours and finding the statistics of their chain code elements in specified directions such as: $45°, 90°, 135°$ (see Figure 60-b). It can be verified that this process heavily depends on the details of the shapes of the connected components, and on the smoothness of the contours. So, normally this process takes a much longer time compared to our component slant angle ($\theta_i$) estimation. This can explain why there is a big difference between the running times of the two algorithms.

In order to have a fair time comparison of these two methods, both were implemented in a Matlab programming environment with the same programming style, and both programs were executed on the same machine having a Pentium(R) 4 Processor with 2.40 GHz speed. Experiments were conducted on the same set of numeral strings used in Section 8.1 (1000 strings randomly selected from NSTRING SD-19 database ). Because of fragmentation and touching cases, the number of CCs in these strings varied from 1 to 11 components per string (there was no string with 9 CCs in our selected set). Our experiments showed that our slant correction method on average is around 6.5 times faster than the method used in [3]. In the next section, the effects of our slant correction on segmentation are shown.

## 8.2 Effects of Slant Correction on Segmentation

In this section, we investigate the effects of our slant correction method on the segmentation of handwritten numeral strings. Here, for our study, we consider the effects of slant correction on the location of foreground and background feature points for segmentation and also on the construction of the candidate cutting paths. In Chapter 4–Section 4.2, we showed that about 65% of the handwritten numeral strings (in NIST NSTRING SD19) have considerable slant angles ($|\Theta| \geq 7°$). In this section, for our experiments, we randomly selected a set of 1000 images of touching pairs of digits from the NSTRING SD-19 database. For each touching

pair, we estimated $\Theta$ (SSA) based on our proposed method in Chapter 4. Observing the values of $\Theta$ for these touching pairs showed that only a subset of them containing 631 pairs, had considerable slant angles ($|\Theta| \geq 7°$). We applied our segmentation algorithm (presented in Chapter 5) to all the touching pairs in this subset (631 slanted pairs) under two different situations: in the first situation, we did not correct the slant of the touching pairs. In the second situation, we applied the slant correction as a pre-processing step before segmentation to correct the slant of the touching pairs. Comparing these two situations showed that slant correction did not affect the position of the foreground feature points significantly. However, in the majority of cases, it greatly changed the position of the background feature points. Changing the background feature points, in turn, can affect the construction of the candidate cutting paths. Experiments showed that in 92.40% of the slanted cases, slant correction could improve the candidate cutting paths. Two examples of these cases are shown in Figure 61, and they are described below:

Figures 61-$A$ and 61-$A'$ show the segmentation results of the same touching pair by the our algorithm in two different situations: without slant correction ($A$), and with slant correction ($A'$), respectively. Comparison of Figures $A$-b and $A'$-b shows that slant correction does not move the position of the foreground feature points significantly. However, comparison of Figures $A$-c and $A$-d with Figures $A'$-c and $A'$-d shows that the background regions and also background feature points are greatly affected by the slant correction. Comparison of Figures $A$-e and $A$-f with Figures $A'$-e and $A'$-f shows that construction of the cutting path has been improved by slant correction. Figures 61-$B$ and 61-$B'$ also show another example where slant correction has improved construction of the segmentation candidates.

Experiments also showed a few cases where slant correction does not improve or affect the results of segmentation significantly. An example of these cases is shown in Figures 62-$C$ and 62-$C'$. In this example, the segmentation algorithm was able to find the correct candidate cutting paths in both situations. Comparison of Figures 62-$C$ and 62-$C'$ shows that there is no significant difference in the background regions and segmentation candidates before and after slant correction.

Our visual inspection on the results of segmentation of touching pairs showed that in 92.40%

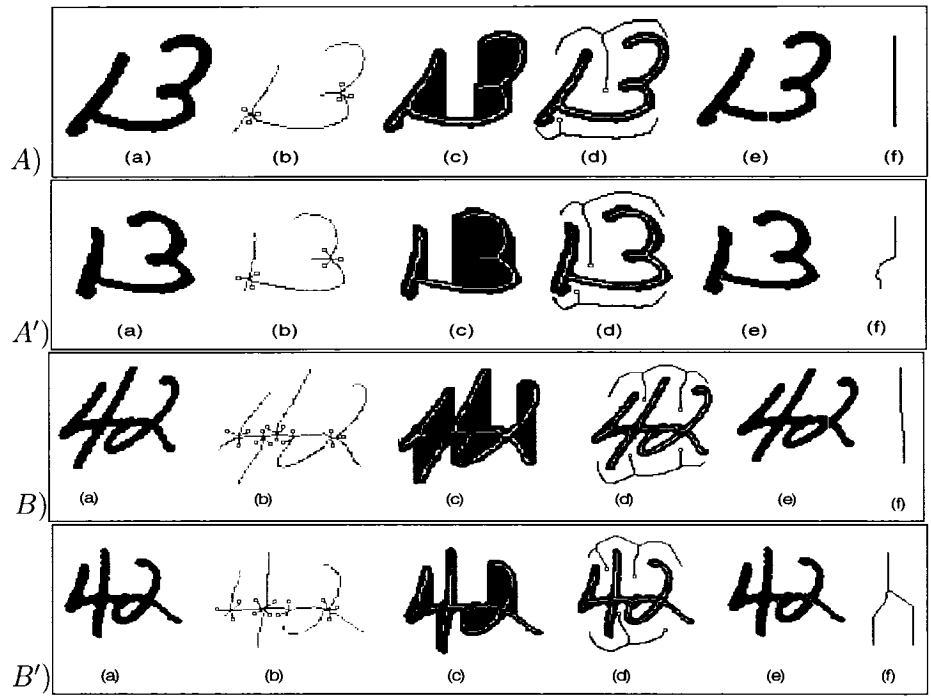



Figure 61: $A$ and $B$: images before slant correction, $A'$ and $B'$: images after slant correction: (a) Original images without slant correction (in $A$ and $B$), with slant correction (in $A'$ and $B'$), (b) Foreground features denoted by □, (c) Combining vertical projection profiles from top and bottom to specify background regions, (d) Taking background skeleton, to extract background features (denoted by □), (e) Image after segmentation, and (f) Candidate cutting path(s) without slant correction (in $A$ and $B$), and using slant correction (in $A'$ and $B'$). In $B$ the segmentation algorithm was not able to find the correct segmentation candidate. However, in $B'$ it was able to do so.

of the slanted pairs, slant correction improves the segmentation candidates of touching digits, and also in 7.60% of the slanted cases, slant correction cannot improve the construction of the segmentation paths significantly. Table 8 summarizes these results. As a result of our experiments, we recommend that slant correction should be added as a pre-processing step to the segmentation algorithms, especially to those algorithms that utilize background information such as [2, 16].

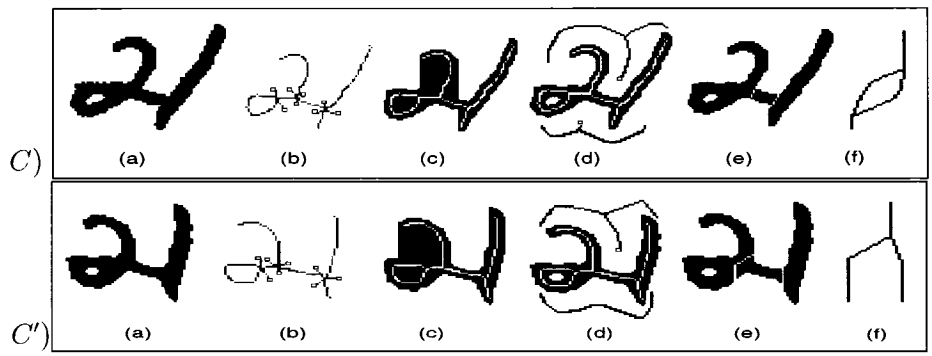



Figure 62: $C$: image before slant correction, $C'$: image after slant correction: (a) Original image without slant correction (in $C$), with slant correction (in $C'$ (b) Foreground features denoted by □, (c) Combining vertical projection profiles from top and bottom to specify background regions, (d) Taking background skeleton, to extract background features (denoted by □), (e) Image after segmentation, and (f) Candidate cutting path(s) without slant correction (in $C$), and using slant correction (in $C'$).

Table 8: Distribution of segmentation cases which were improved or not improved by slant correction.

| Segmentation of slanted cases | Number | Percent(%) |
|---|---|---|
| Segmentation cases improved by slant correction | 583 | 92.40% |
| Segmentation cases not improved by slant correction | 48 | 7.60% |
| Total slanted cases ($\|\Theta\| \geq 7°$) | 631 | 100% |

## 8.3 Testing the Segmentation Algorithm

The function of our segmentation module is to over-segment an input numeral string by constructing some cutting paths. In the experiments with segmentation algorithm, we examined the performance of our segmentation module independent of other parts of the system (without using classification information). In order to have a better comparison of our algorithm with similar algorithms in the literature, we conducted two sets of experiments. In the first set, we took 5000 touching pairs of digits from the NIST Database, and we input them to the algorithm. Experiments showed that in 96.5% of the touching cases, our algorithm was able to produce the correct cutting path. In the next set of the experiments, we randomly selected 1800 images, from the NIST NSTRING SD19 Database (for each string length of:

2, 3, 4, 5, 6, and 10, we took 300 string images). The lengths of the strings were not given to our algorithm, and they were considered unknown. After inputting these images, we looked at the output of the segmentation module, to verify the results. We defined segmentation (or over-segmentation) of a numeral string as successful if the set of cutting paths produced for that numeral string contained all the necessary cutting paths. Otherwise, segmentation was considered unsuccessful. A visual analysis revealed that in 98.04% of the cases, our segmentation algorithm could successfully over-segment the numeral strings. Figures 63 and 64, respectively, show some successful and unsuccessful segmentation results produced by our segmentation algorithm.

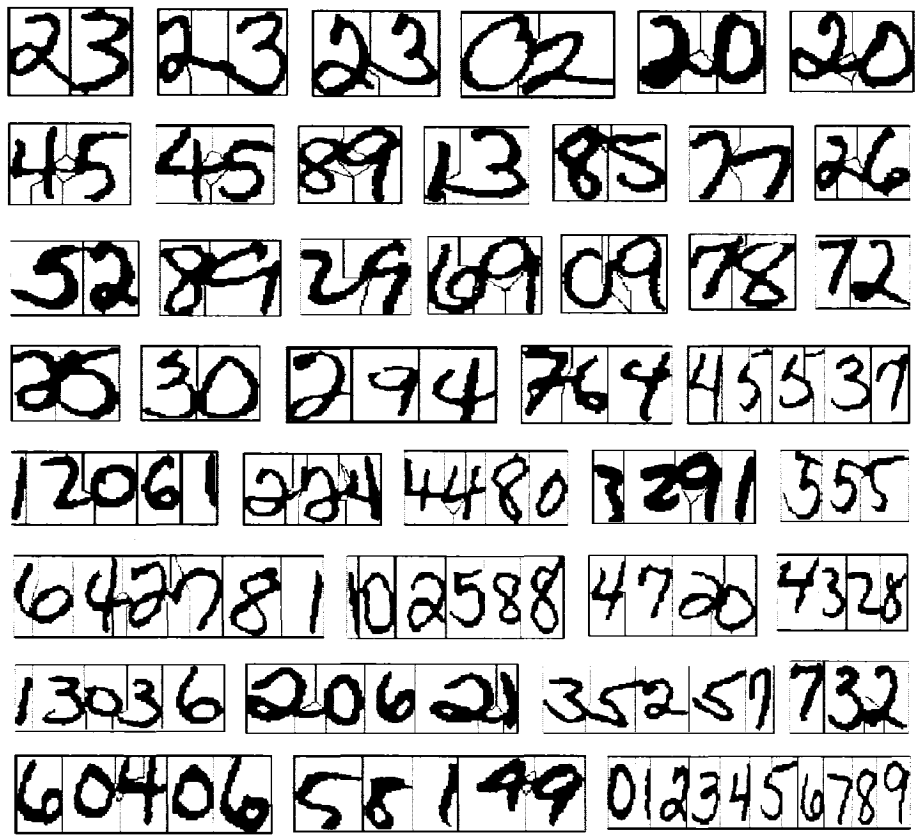


Figure 63: Examples of successful segmentations (or over-segmentations) produced by our segmentation algorithm. Here, we do not use any recognition information. Bounding boxes contain real life string images from the Nstring SD19 Database with different lengths (2 to 10 digits). Each cutting path goes from the top to the bottom of the bounding box or vice versa. As shown in this figure, some cutting paths have two or more branches.

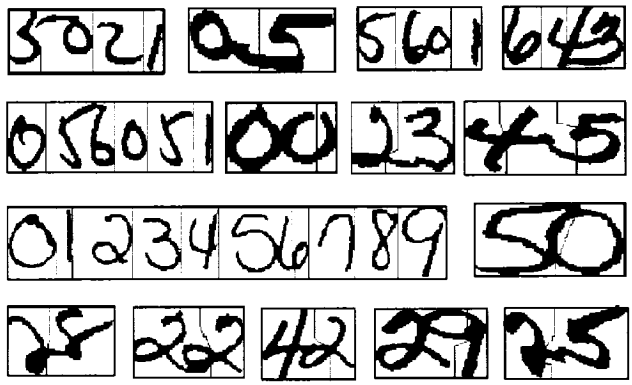


Figure 64: Examples of unsuccessful cases of segmentation (without using any recognition information) by our segmentation algorithm.

As shown in Figure 64, most unsuccessful segmentations were due to a large amount of overlaps between the connected components in the strings. Most of the algorithms in the literature (such as [2],[13],[10],[35], etc.) were proposed to split only touching pairs of digits, and they were not used for the segmentation of longer numeral strings (with unknown lengths). In contrast, our algorithm assumes an unknown number of isolated or touching components in the numeral strings. In addition, the majority of the segmentation algorithms in the literature use recognition information to select the best cutting paths. By contrast, the goal of our algorithm is to over-segment the numeral strings by introducing a super set of best cutting paths, and it does not make any final decision about those cutting paths. Also, note the methods such as [2, 1, 153] have very high rejection rates. Although our algorithm uses neither recognition information for rejection (or acceptance), nor information about the lengths of the strings, Table 9 shows that the results of our segmentation algorithm compares favorably to similar algorithms found in the literature such as [1, 2, 35, 119, 153].

### 8.3.1 Collecting Outlier Patterns

After producing all the segmentation hypotheses for our numeral strings, we were able to collect a set of over and under-segmented components. These components are called outliers

Table 9: Performance Comparison of Different Segmentation Algorithms

| Approach | Correct Rate (%) | Error Rate (%) | Reject Rate (%) | Database | Recog. info. |
|---|---|---|---|---|---|
| Chi et al.[153] | 95.1 | 4.9 | After 32.7 | 3355 images of touching digits from NIST Database | yes |
| Chi et al.[153] | 89.2 | 10.8 | After 2.8 | 3355 images of touching digits from NIST Database | yes |
| Cheriet et al.[118] | 80.8 | 19.2 | 0 | 120 images of touching digits from their own collection | no |
| Lu et al.[1] | 97 | 3 | After 28.6 | 3355 images of touching digits from NIST Database | yes |
| Lu et al.[1] | 92.5 | 7.5 | After 4.7 | 3355 images of touching digits from NIST Database | yes |
| Shi et al.[119] | 95 | 5 | 0 | 495 zipcode images from CEDAR CD-ROM Database | no |
| Shi et al.[119] | 85.7 | 14.3 | 0 | 2579 US zipcode images from US Postal Database | yes |
| Oliveira et al.[35] | 98.5 | 1.5 | 0 | 900 touching digits from Brazilian Bank Checks | no |
| Oliveira et al.[35] | 95.24 | 2.14 | 2.62 | 900 touching digits from Brazilian Bank Checks | yes |
| Chen and wang[2] | 96 | 4 | After 7.8 | 4178 images of touching digits from NIST Database and 332 images from their own collection | yes |
| **Our approach** | 96.5 | 3.5 | 0 | 5000 images of touching digits from NIST Database | no |
| **Our approach (Over-segmentation)** | 98.04 | 1.96 | 0 | 1800 images from NIST NSTRING SD19 Database (touching and non-touching strings with lengths of: 2,3,4,5,6, and 10 digits) | no |

(out-of-class or non-digit) samples. In total, 1640 outlier samples were collected: 1154 samples of over-segmented outliers and 486 samples of under-segmented outliers. Some of these outlier samples are shown in Figure 65. These outliers were used for two different purposes. Firstly, by using them we could adjust the parameters ($\alpha$, $k$, $t$, and $\beta$) in Equations 28 and 30 in Section 7.2 (for our segmentation scores). Secondly, these outlier samples were utilized to improve the training of our classifiers, explained in the next section.

## 8.4 Implementation and Testing of the Classifiers

Before using MLP and SVM to classify the input digits, they must be trained by samples in a database (training set). Compared to other similar handwritten databases, samples in the CENPARMI database show more variations in their shapes and styles. We therefore selected this database for the training of our isolated digit classifiers. Some samples

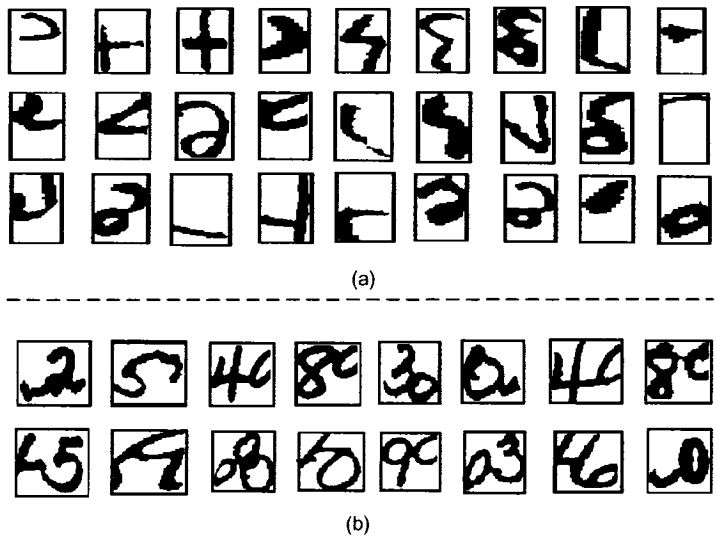



Figure 65: Outlier Samples: (a) Over-segmented samples, (b) Under-segmented samples.

of the CENPARMI isolated digit database are shown in Figure 66. Using the CENPARMI handwritten isolated digit database and the collected outlier samples, our MLP and SVM classifiers were trained and their parameters were adjusted. The CENPARMI database has 4000 training samples (400 samples per digit) and 2000 testing samples (200 samples per digit). Researchers in [8] and [154] showed that adding outlier (or non-character) samples to the training set of isolated digit classifiers (such as MLP, SVM, etc.) yields higher performances in the classifier's rejection of outliers. Therefore, in order to improve the outlier resistance of our classifiers, 1640 outlier samples (produced in the previous stage of our experiments) were added to the training samples of the CENPARMI database. However, unlike [8], and [154], we did not require a very large number of outlier samples to train our classifier to become outlier resistant. This is because most of the outlier samples were handled/rejected by our segmentation scores (which were introduced in Chapter 7–Section 7.2), and not by our isolated digit classifiers (MLP and SVM).

After some trial training of our classifiers (MLP, and SVM) the number of neurons in the hidden layer of the MLP was set equal to 85 (it has 225 and 10 neurons in the input and output layers, respectively). The parameters $\sigma$ and $C$ of the SVM_rbf classifier were also set equal to 1.15 and 10, respectively. The performances of our classifiers after training

on the CENPARMI isolated handwritten digit database are presented in Table 10. As it shows, our MLP, and SVM classifiers could reach the recognition rate of 98.03% and 98.90% on the testing set of the CENPARMI isolated digit database. Table 10 also shows that the performances of our two classifiers on the testing set are a little lower than the results reported by [43] (using SVM_rbf). Although our isolated digit classifiers have a little lower performance in isolated digit recognition than [43], our system shows outlier resistance and a very good overall performance. We will discuss this issue and compare the performance of our system with other systems in the recognition of numeral strings in Sections 8.6 and 8.8, respectively.

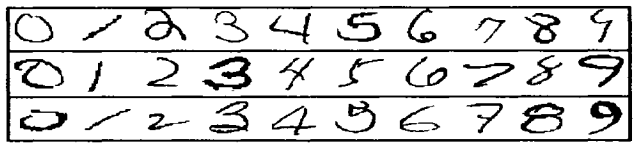


Figure 66: Some samples of the CENPARMI isolated handwritten digit database.

Table 10: Performance evaluation of our classifiers on CENPARMI isolated digit Database. (N.A. stands for not applicable.)

| Database: CENPARMI | Our Approach | | | | | | Approach in [43] | | |
|---|---|---|---|---|---|---|---|---|---|
| | MLP | | | SVM_rbf | | | SVM_rbf | | |
| | Rec. (%) | Err. (%) | Rej. (%) | Rec. (%) | Err. (%) | Rej. (%) | Rec. (%) | Err. (%) | Rej. (%) |
| Training Set (4000 Samples+ 1640 Outlier samples) | 99.81 | 0.19 | 0 | 99.92 | 0.08 | 0 | N.A. | N.A. | N.A. |
| Testing Set (2000 Samples) | 98.03 | 1.97 | 0 | 98.90 | 1.10 | 0 | 99.05 | 0.95 | 0 |

## 8.5 Adjusting Parameters of our Genetic Algorithm

Genetic algorithms have several parameters such as probability of reproduction ($P_r$), probability of crossover ($P_c$), probability of mutation ($P_m$), size of the initial population ($M$),

and maximum number of generations ($G$). Before using GA, these parameters must be adjusted. In order to adjust these parameters the same samples of numeral strings that were used in testing of segmentation module (1800 numeral strings) were also used to adjust the parameters of our genetic algorithm (GA). After some trial adjustment, the parameters of our GA were determined as shown in Table 11. In this table, $n$ is the number of cutting paths generated by our segmentation algorithm. For short numeral strings where $n$ is less than or equal to 5, the total number of possible segmentation hypotheses ($2^n$) will be less than or equal to 32. In these cases, our search algorithm conducts an exhaustive search on the population of segmentation hypotheses, instead of a GA search, but in the other cases (where $n \geq 5$) it conducts a GA search.

As shown in the flowchart in Figure 22, our genetic search (GA) starts with a random population of chromosomes, where each chromosome has $n$ binary genes. Then it searches for the optimum solution by updating the population using genetic operations. Our GA terminates in two cases: first, it terminates if a solution is found with desired fitness (a solution where its confidence is greater than a threshold of $T_1$; $T_1$ is shown in Figure 22). Second, it terminates if the number of iterations (generations) reaches the maximum number generations ($G$), and no other good solution has been found. For the former case, the algorithm accepts the string, and it outputs the labels and corresponding confidence values for the string. For the latter case, the algorithm rejects the input string. In our experiments, the rejection level of the system was controlled by adjusting the acceptance/rejection threshold ($T_1$). In our experiments, a typical value for $T_1$ was set equal to 0.80. In the next section, we will present the overall numeral string recognition results produced by our system.

Table 11: Parameter values for our genetic algorithm.

| $P_r$ | $P_c$ | $P_m$ | $M$ | $G$ |
|---|---|---|---|---|
| 0.4 | 0.55 | 0.05 | 32 | $2^{\lfloor n/2 \rfloor}$ |

## 8.6 Numeral String Recognition Results

Many researchers have used the NIST NSTRING SD19 Database in order to evaluate their segmentation/recognition systems. For example, NIST NSTRING SD19 Database (hsf_7 series) was used by researchers in [9], [150], and [76]. Authors in [8], also used NIST NSTRING SD19 for some of their experiments, but they only reported the results on numeral strings with lengths 3 and 6, and they used fewer string images (1471 images, for each string length). In order to compare our results with all these researchers, we also used the NIST NSTRING SD19 Database for our experiments, and we used the same test images and string lengths used in [9], [150], and [76] (reference [8] used only a subset of those images). Table 12 summarizes the segmentation/recognition rates of our system on numeral strings of lengths 2, 3, 4, 5, 6, and 10 digits. In this table, we report the results of our system with two different isolated digit classifiers: MLP, and SVM under two different conditions: first, without using segmentation scores, and using only the recognition scores (columns 3-6); second, using segmentation scores in addition to the recognition scores (columns 7-10). In the former case, segmentation hypotheses are evaluated based on their recognition scores produced by the isolated digit classifiers (MLP, or SVM), without using any contextual information. In the latter case, in addition to the recognition scores, contextual information (segmentation scores) are also used to evaluate the segmentation hypotheses. The results of these two cases at zero rejection level for the MLP and SVM classifiers, can be compared from columns 3 to 7 and 6 to 10 of Table 12, respectively. This comparison shows that using segmentation scores (contextual knowledge) improves the results of the numeral string recognition system. The average improvement for the case of MLP is 7.85%, and for the case of SVM it is 5.21%. For simplicity, we also provide the results only for the MLP classifier at two different levels of errors (less than 1%, or 0.5%) by changing the acceptance/rejection threshold ($T_1$). As seen in Table 12, on average, our system can correctly segment, and recognize 95.28% (in case of MLP), and 96.42% (in case of SVM) of the test samples from NIST NSTRING SD19 (hsf_7) Database at a zero rejection level. In Table 13 a summary of our best results using

segmentation scores taken from Table 12 is compared with the best results of similar approaches found in the literature. This table shows that our results using segmentation scores compare favorably to those existing in the literature [8, 9, 150, 76].

Table 12: Recognition rates of our system on the numeral strings from NSTRING SD19, using two different classifiers: MLP and SVM, and in two different situations: without and with segmentation scores (contextual knowledge).

| Col# 1 | Col# 2 | Col# 3 | Col# 4 | Col# 5 | Col# 6 | **Col# 7** | Col# 8 | Col# 9 | **Col# 10** |
|---|---|---|---|---|---|---|---|---|---|
| | | Without Segmentation Scores | | | | **With Segmentation Scores** | | | |
| String | Number | MLP | | | SVM | **MLP** | | | **SVM** |
| Length | of Strings | 0%(rej) | 1%(err) | 0.5%(err) | 0%(rej) | **0%(rej)** | 1%(err) | 0.5%(err) | **0%(rej)** |
| 2 | 2370 | 92.06 | 87.05 | 85.56 | 95.05 | **97.87** | 92.20 | 90.77 | **98.94** |
| 3 | 2385 | 89.39 | 84.79 | 83.04 | 91.43 | **96.43** | 90.06 | 87.18 | **97.23** |
| 4 | 2345 | 88.20 | 83.63 | 81.00 | 91.07 | **95.17** | 87.91 | 85.04 | **96.16** |
| 5 | 2316 | 85.75 | 81.91 | 79.05 | 88.05 | **94.91** | 86.97 | 84.38 | **95.86** |
| 6 | 2169 | 85.64 | 81.02 | 78.69 | 88.69 | **94.26** | 83.01 | 80.41 | **96.10** |
| 10 | 1217 | 83.51 | 79.75 | 78.13 | 86.13 | **93.01** | 81.12 | 79.05 | **94.25** |
| Average Rates | — | 87.43 | 81.56 | 80.91 | 90.07 | **95.28** | 86.88 | 84.47 | **96.42** |

Table 13: Comparison of the recognition rates of our approach (using segmentation scores) with similar approaches on the test samples of NIST NSTRING SD19. (Reference [8] used only a subset of the test samples for their experiments.)

| Col# 1 | Col# 2 | Col# 3 | Col# 4 | Col# 5 | Col# 6 | Col# 7 | **Col# 8** | **Col# 9** |
|---|---|---|---|---|---|---|---|---|
| String | Number | Results | Results | Results | Results by [8] | | **Our Approach** | |
| Length | of Strings | by [76] | by [9] (MLP) | by [150] (SVM) | MLP | SVM | **MLP** | **SVM** |
| 2 | 2370 | 94.8 | 96.88 | 97.67 | — | — | **97.87** | **98.94** |
| 3 | 2385 | 91.6 | 95.38 | 96.26 | 95.60 | 96.82 | **96.43** | **97.23** |
| 4 | 2345 | 91.3 | 93.38 | 94.28 | — | — | **95.17** | **96.16** |
| 5 | 2316 | 88.3 | 92.40 | 94.00 | — | — | **94.91** | **95.86** |
| 6 | 2169 | 89.1 | 93.12 | 93.80 | 95.58 | 96.74 | **94.26** | **96.10** |
| 10 | 1217 | 86.9 | 90.24 | 91.38 | — | — | **93.01** | **94.25** |
| Mean | — | 90.33 | 93.57 | 94.57 | — | — | **95.28** | **96.42** |
| Standard | — | ±2.8 | ±2.3 | ±2.2 | — | — | ±1.7 | ±1.6 |
| Deviation | — | ▷○ | • | * | — | — | ▷• | ○* |

Paired T-test analysis between different columns: ▷ (col#3 and col#8) P Value= 0.0001, ○ (col#3 and col#9) P Value = 0.0001, • (col#4 and col#8) P Value= 0.0015, * (col#5 and col#9) P Value = 0.0006.

Some successful and unsuccessful segmentation-recognition results of our system on handwritten numeral strings from NSTRING SD19 are also shown in Figures 67 and 68, respectively. As these two figures indicate, there are four possible outcomes for the output of a handwritten numeral string segmentation-recognition system: correct segmentation-correct recognition, wrong segmentation-wrong recognition, correct segmentation-wrong recognition,

and wrong segmentation-correct recognition. The first and the last of these cases produce correct recognition results.

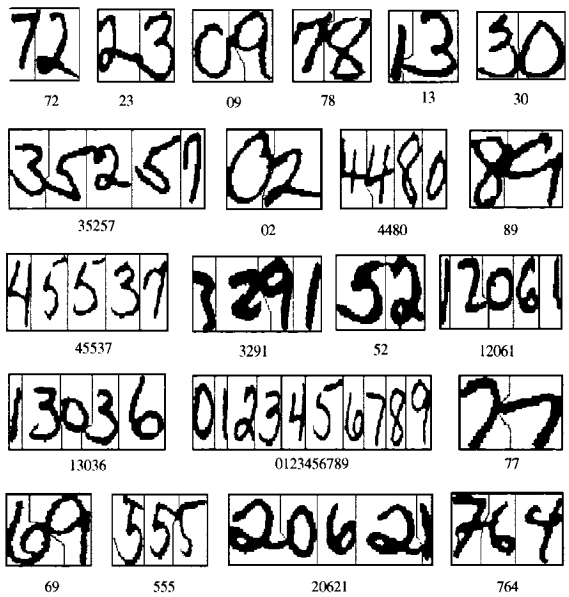



Figure 67: Successful examples of segmentation/recognition produced by our system. Numeral strings are taken from NSTRING SD19 Database.

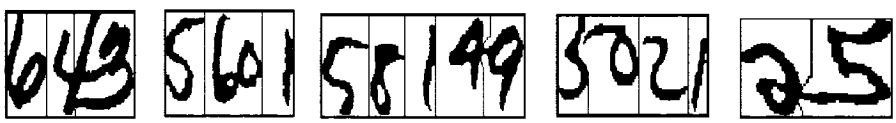



Figure 68: Unsuccessful examples of segmentation/recognition produced by our system. (a, and b) wrong segmentation-wrong recognition, (c) correct segmentation-wrong recognition, (d, and e) wrong segmentation-correct recognition

## 8.7 Statistical Analysis

In order to verify the statistical significance of the improvement of our results in Table 13 over other methods in that table, a statistical analysis using GraphPad InStat Software (Version

3.05) was performed. A paired T-test was applied and one-tailed P value less than 0.05 considered for a significant differences. These comparisons have been shown at the bottom of Table 13. This analysis has been performed on the results reported by Britto et al. [76] and Oliveira et al. [9, 150], and the results by Liu et al. [8] were not included (because they did not use all the samples in the database). The data in Table 13 passed the normality test, and the results based on 95% confidence interval and 5 degree of freedom were extremely significant for both our MLP and SVM classifiers.

## 8.8 Discussion

In this section, we discuss the results that we obtained in our experiments with handwritten numeral strings (Section 8.6), and we compare them with similar methods reported in the literature. There are two recent examples of numeral string recognition systems in the literature where researchers focused on improving the isolated digit classifiers in order to improve the overall performance of the system. First, researchers in [9] used MLP neural networks to function as an isolated digit classifier and as verifiers for the recognition of handwritten numeral strings, and then in [150] they substituted these MLP neural networks for Support Vector Machines with Radial Basis Function Kernels (SVM_rbf). Their results are shown in columns 4, and 5 of Table 13, respectively. Comparison of these two columns in Table 13 shows that although SVMs have a higher generalization power (and a higher computational cost) than neural networks in isolated digit recognition [43], substituting MLPs for SVMs could improve the performance of a numeral string recognition system by an average of around 1%. Secondly, researchers in [8] also studied the performances of various isolated digit classifiers on handwritten numeral string recognition and various training methods of those classifiers. They also found that using different isolated digit classifiers or using different strategies for their training can yield different performances in handwritten numeral string recognition. Comparison of their results for two sample classifiers (MLP Neural Network, and SVM_rbf) in columns 6, and 7 of Table 13, show that SVM_rbf has about $1.16\% - 1.22\%$ higher performance in numeral string recognition than MLP Neural

Networks. These results are consistent with the results of [9, 150, 50]. These two examples of systems for handwritten numeral string recognition show that there is a possibility to improve the performance of a numeral string recognition system slightly, by enhancing the isolated digit classifier module. However, in our experiments, unlike [9], [8], and [150] we kept our isolated digit classifiers (MLP and SVM) unchanged, and we only added the segmentation scores (as a source of contextual knowledge) to the system, and we were able to increase the performance of our system, to about 7.85% in the case of MLP, and 6.35% in the case of SVM , which are very significant improvements. The reason for this improvement can be explained as follows: isolated character classifiers (such as SVM, MLP, etc.) are designed, and trained in order to recognize single and independent components one at a time, so normally they do not use the contextual knowledge (such as comparisons of adjacent components, or comparison of relative sizes or relative positions of different components) in their decisions. Not using contextual knowledge, which is easily available in numeral strings, can cause isolated digit classifiers (even with a very high recognition power) to misrecognize many outliers, and to treat them as valid digits. This can explain why boosting the isolated digit classifier in a numeral string recognition system is not the most effective solution in order to improve the overall performance of the system.

Researchers in [8] also showed that training the isolated digit classifiers with many outlier samples (in addition to the original training samples), can improve the outlier resistance of those isolated digit classifiers, and subsequently this can improve the performance of those classifiers in handwritten numeral string recognition systems. However, our experiments showed that there were many outlier samples that could not be recognized/rejected by isolated digit classifiers, no matter whether those classifiers have been trained or not by the outlier patterns. Also, in order to train classifiers with outlier samples, a set with a very large number of outlier patterns is required. Currently, such a standard set of outlier samples is not available. Therefore, researchers have to use their own collected outlier samples.

In summary, in systems which do not use segmentation scores (sources of contextual knowledge), correct segmentation-recognition of numeral strings only relies on the accuracy and the outlier resistance of the classifier. So, these systems are very dependent on the structure

and training method of the isolated digit classifier. In comparison, in our system, correct segmentation-recognition of numeral strings depends on two different factors: first the accuracy/outlier resistance of the classifier, and secondly, the segmentation scores. Since segmentation scores are independent of the classifier and they come from another source (contextual information), in our system the classifier has a less crucial rule in the performance, and this can also improve the reliability of the system.

## 8.9 Summary

In this chapter, we presented the experimental results of our modular system, and we compared them with similar systems in literature. All of our experiments in this chapter were conducted on handwritten isolated digits or numeral strings from the CENPARMI and NIST NSTRING SD19 databases. Here, first we compared our slant correction algorithm with other slant correction algorithms, and we showed the effects of slant correction on segmentation of handwritten numeral strings. Then, we compared the performance of our segmentation algorithm and our isolated digit classifiers with other segmentation algorithms and isolated digit classifiers in the literature. Adjusting the parameter of our system was also explained. Finally, we compared and discussed details of our experimental results with similar methods.

# Part II

# Towards Generalization and Applications

# Chapter 9

# Incremental Learning: Plasticity and Stability in Handwritten Digit Recognition Systems

In Chapters 7 and 8 (Sections 7.3 and 8.4, respectively), we designed and used MLP Neural Networks and SVM classifiers for the recognition of segmented digits and in order to produce recognition scores. Although Neural Networks (NN's) and Support Vector Machines (SVM's) both have a very good generalization power [43, 90], and they have received a great deal of attention over the last decade in many application domains, they both have a low incremental learning capacity [155]. In many applications such as character or digit recognition, new data and new styles of writing characters/digits are continuously introduced and added to already huge databases. Therefore, we require the introduction of recognition systems that operate incrementally that are able to learn new data.

In this chapter, in order to capture the very large variability that exists in the shapes and styles of handwritten digits, and in order to learn the new shapes/styles of digits incrementally, we introduce a new clustering algorithm. Our clustering algorithm is an incremental unsupervised learning algorithm that is able to continuously learn new shapes of digits from the same class (plasticity) and at the same time, it is able to remember the shapes of the digits that it has learned previously (stability) [19]. By applying our clustering algorithm

and by using a K-Nearest Neighbor classifier, we have designed a system that shows stability and plasticity in handwritten digit (character) recognition.

## 9.1 Needs for Plasticity and Stability in Digit Recognition Systems

The determination of a set of representative prototypes to be used by a pattern recognition system (or classifier) is a very challenging design step [156]. In the case of handwritten character recognition, finding representative prototypes can be very complex, first because of the very large variability that is exhibited by handwritten characters. Second, people's handwriting styles normally change with time; therefore, new data samples/shapes of writing for characters/digits are introduced and continuously added to already huge databases. One way to tackle this challenge is to introduce handwritten recognition systems that adapt to new writing styles without forgetting the previously learned ones. Therefore, we require machine learning techniques in order to automate this task. Conventional machine learning techniques (such as conventional Neural Networks and SVM's), in order to learn new patterns, must be retrained. In these machines, learning new patterns involves modifying all previously trained weights and adjusted parameters, and normally by retraining (learning) new prototypes, these machines forget the old learned prototypes. We need machine learning techniques (classifiers) which do not require retraining the entire set of weights (parameters) when a new pattern is learned. However, in conventional learning machines, in order to maintain the general performance of the system at a relatively high level, the retraining process should involve all the historical samples besides the new samples. So, we have to combine those samples into one huge data set and use it for retraining of the system. This is not efficient in terms of both time and space, and it is considered as one of the main challenges in the design of evolutionary, efficient and robust handwritten character recognition systems.

In pattern recognition (or machine learning), the ability to learn new patterns continuously is called plasticity, and the ability not to forget the previously learned patterns is called

stability [157]. There is a conflict between plasticity and stability. Simply speaking, there is always a question of how to keep learning new things without forgetting or destroying previously learned information [157, 158]. Adaptive Resonance Theory (ART) was proposed in order to resolve the plasticity-stability dilemma in machine learning [157]. In fact, all classifiers/learning systems should solve the plasticity-stability dilemma. Due to the challenges mentioned in the previous paragraph, we need handwritten character recognition systems that exhibit plasticity and stability in their learning. However, our literature survey showed that there has not been much research on implementing plasticity and stability in handwritten character recognition systems. In this chapter, in order to achieve plasticity and stability in handwritten digit recognition systems, we introduce a new clustering algorithm based on ART. We also introduce a new set of similarity measures that can compare similarities of the shapes of handwritten digits. Using our features, similarity measure, and clustering algorithm, handwritten digits are clustered based on their shapes, and the centers of different clusters for each digit are found. These centers of clusters (basic shapes) will function as representative prototypes which can efficiently model variations in each digit class. Finally, a K-Nearest Neighbor (K-NN) classifier, utilizing our similarity measure, is used to assign class labels to new samples based on their similarity to the prototype shapes of each digit. This system is able to incrementally learn the new shapes/styles of handwritten characters (shows plasticity), without forgetting the previously learned ones (shows stability).

In the next subsections, we will describe the details of our feature extraction, similarity measure, and our proposed clustering algorithm. Finally, we will show our experimental results.

## 9.2 Feature Extraction and Similarity Measures

Feature extraction and similarity measures are two essential components of any clustering method. They will be described in this section and applied in Section 9.3.

### 9.2.1 Feature Extraction

For feature extraction, we used the method that was presented and used in Chapter 7 (in Section 7.3). This feature extraction method is briefly reviewed here. Our method tries to reduce the variation in the shape of a digit through smoothing, slant correction, and normalization, and it utilizes a two-dimensional representation for the shapes or structures of digits. The details of preprocessing and feature extraction are illustrated in Figure 69.

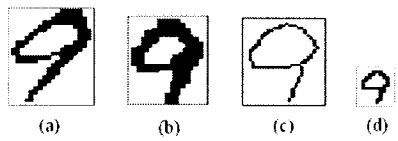



Figure 69: (a) Original image, (b) Smoothed, slant corrected, and normalized image (45 by 45 pixels), (c) Skeleton of part b, (d) Reducing the resolution of the skeleton in horizontal and vertical directions by down sampling (1/3). The resulting (15 by 15 pixels) image in (d) is considered a (2D array) representation of the structure and style of writing of a digit.

### 9.2.2 Similarity Measures

Similarity or dissimilarity (distance) measures also play an important role in clustering, pattern recognition and classification. Researchers have investigated similarity/dissimilarity (distance) measures for a century, and currently many similarity/distance measures are available in the literature, such as: Euclidian distance, Inner product, Cosine Similarity, Hamming distance, Rogers-Tanimoto similarity [159], etc. Rogers-Tanimoto similarity can measure similarity between two binary feature vectors, so it can be used to measure the similarity of the shapes of digits in our representation. We adopted Rogers-Tanimoto similarity measure and we modified it a little bit by adding some adjustable weights for its terms which is explained as follows:

$$S_{R-T}(X,Y) = \frac{X^tY + \overline{X}^t\overline{Y}}{X^tY + \overline{X}^t\overline{Y} + 2(X^t\overline{Y} + \overline{X}^tY)} \quad \text{(Rogers-Tanimoto Measure)} \qquad (36)$$

$$MS_{R-T}(X,Y) = \frac{\alpha X^t Y + \beta \overline{X}^t \overline{Y}}{\alpha X^t Y + \beta \overline{X}^t \overline{Y} + \gamma(X^t \overline{Y} + \overline{X}^t Y)} \quad \text{(Our Modified Measure)} \qquad (37)$$

Here $X$, and $Y$ are two binary feature vectors (where zeros stand for white pixels, and ones stand for black pixels), and $X^tY$ stands for the inner product of the two vectors. $\alpha$, $\beta$, and $\gamma$ are adjustable non-negative weights (credits) where at least one of them must be non zero. Each term in Rogers-Tanimoto similarity has a meaning as follows: $X^tY$ shows the positive matching (black to black matching pixels); $\overline{X}^t\overline{Y}$ shows the negative matching (white to white matching pixels); $X^t\overline{Y}$, and $\overline{X}^tY$ show non-matching cases between two input vectors (black to white pixels or white to black pixels, respectively). Rogers-Tanimoto similarity measure always assigns a score value between 0 and 1 for the similarity of two pattern vectors. If two pattern vectors, $X$ and $Y$, are exactly the same, their similarity, based on this measure, will be equal to 1. Whenever the two pattern vectors are completely dissimilar (for example, logical complements of each other), their similarity measure is equal to 0. In addition to the above properties, our modified measure in Equation 37 introduces a bigger family of similarity measures which enables us to apply different weights or credits for positive matching (black to black pixels matching) and negative matching (white to white pixels matching), or larger weights for non-matching cases (in the denominator) in order to penalize those non-matching cases (and to obtain more restricted similarity measures). We can say that Rogers-Tanimoto similarity is a special case of this family where both $\alpha$ and $\beta$ are equal to 1, and $\gamma$ is equal to 2. In our modified version, the greater the ratio of $\gamma^* = \frac{\gamma}{\max(\alpha,\beta)}$, the more restricted the similarity measure (more sensitive to non-matching cases) will be, and the lower the ratio $\gamma^* = \frac{\gamma}{\max(\alpha,\beta)}$, the more flexible the similarity measure (less sensitive to non-matching cases) will be. If we choose $\alpha = \beta$ ($\neq 0$), we can write our similarity measure as follows, which has only one control parameter, denoted by $\gamma^*$:

$$MS_{R-T}(X,Y) = \frac{X^t Y + \overline{X}^t \overline{Y}}{X^t Y + \overline{X}^t \overline{Y} + \gamma^*(X^t \overline{Y} + \overline{X}^t Y)} \qquad \text{(where } \gamma^* \geq 0) \qquad (38)$$

In the next section, we explain our clustering algorithm.

## 9.3 Clustering Algorithm

Data clustering is one of the most traditional and important issues in computer science. Cluster analysis aims at discovering groups and identifying meaningful distributions and patterns in large datasets. Numerous clustering algorithms have been proposed in the literature, and some methods such as K-Means clustering are very popular [67]. See [160] for a recent survey of clustering algorithms. In recent years, due to emerging complex applications such as data and text mining, data clustering has attracted a new round of attention [158]. In many applications, such as handwritten recognition or data mining in dynamic environments, databases are growing. Designing efficient and robust clustering algorithms that can cluster patterns incrementally is very challenging. As a result, few works for the development of incremental clustering algorithms have been found in the literature [158]. ART clustering algorithm (ART neural network) is one of the algorithms that has incremental learning capabilities. ART is a type of unsupervised learning algorithm that tries to assign an input pattern into one of the stored categories depending on which stored pattern it most resembles. If the input pattern does not match any stored pattern within a system parameter $\delta$ (called vigilance factor; here, we refer it as minimum similarity threshold), a new category is created, and no stored pattern is destroyed or changed. We took the basic idea of this algorithm, and we modified it as follows. In our method, in order to improve the generalization in learning, whenever the centers of the neighboring clusters (categories) become very close to each other, (based on the chosen similarity or distance measure), they will be merged into one cluster as illustrated in Figure 70.

The pseudo code of our algorithm is shown in Figure 71, where the minimum similarity threshold ($0 \leq \delta \leq 1$) is the only parameter that must be adjusted before running the algorithm. In each cluster, each pattern (member of the cluster) must have a similarity at least of $\delta$ to the center of that cluster, in other words a pattern is assigned to a cluster with maximum similarity if that similarity is greater than $\delta$. Otherwise, a new cluster is created for that pattern. Unlike offline clustering algorithms (such as K-Means), here, we do not

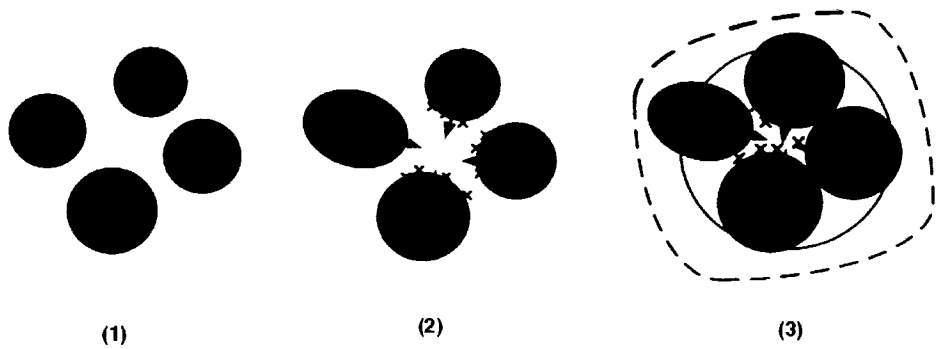



Figure 70: New incoming data may push the centers of some previously learned clusters towards each other. By merging these (smaller) clusters, gradually bigger clusters are emerged (more general concepts or better generalizations are achieved).

specify the number of clusters a priori, and we do not require all the patterns to be present at the beginning of the clustering process. Also, unlike K-Means, in order to find the centers of clusters, our algorithm scans (visits) all the input patterns only once, and it does not iterate over the input patterns several times. These properties of the algorithm make it suitable for the clustering of incoming online patterns (incremental learning).

**Clustering Algorithm**

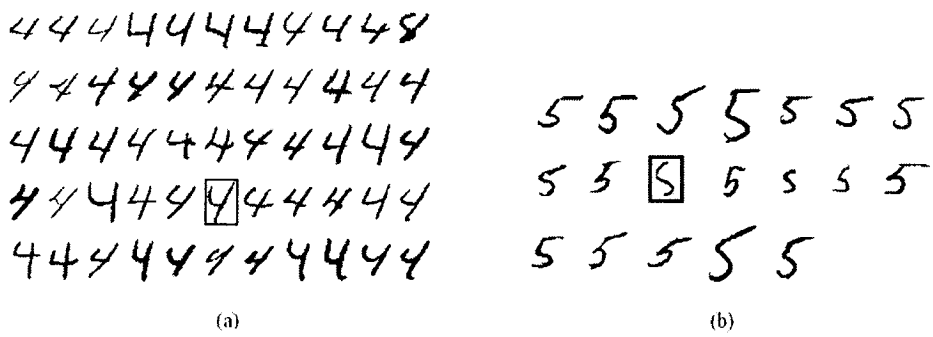
1 Adjust the minimum similarity threshold ($0 \leq \delta \leq 1$).
2 Cluster_centers_list = $\phi$.
3 Read the next pattern.
4 Find the most similar cluster center to the pattern in cluster_centers_list with similarity greater than $\delta$.
If found: assign the pattern to that cluster; Adjust cluster center; If this new center becomes close (with similarity greater than $\delta$) to any other center, then merge the corresponding clusters as one cluster; Update the cluster_centers_list.
If not found: build a new cluster, and insert the input pattern into the cluster_centers_list as a new cluster center.
5 Repeat steps in 3-4 for all the input patterns.
6 Output cluster_centers_list.

Figure 71: Our clustering algorithm.

## 9.4 Experimental Results

For our experiments on handwritten isolated digits, similar to Chapter 8, we used the CENPARMI isolated digit database. As we mentioned in that chapter (8), handwritten samples in the CENPARMI database show more variations in their shapes, compared to similar handwritten databases. This database has 4000 training samples (400 samples per digit) and 2000 testing samples (200 samples per digit). Here, for implementing our similarity measure, we considered equal weights for black to black and white to white matching pixels. Therefore, we used Equation 38 for computing the similarity. After some initial adjustments, we set $\gamma^*$ (parameters of our similarity measure) equal to 3.5, also we set $\delta$ (minimum similarity threshold in our clustering algorithm) equal to 0.75. We applied our clustering algorithm to all training samples of each digit in the CENPARMI database (400 training samples per each digit). Afterwards, we found the centers of the clusters for each digit. We considered those centers as candidate prototypes for the shapes of digits. Two examples of clusters found by our algorithm, (one for digit 4 and one for digit 5) are shown in Figure 72. The centers of these clusters, have also been identified by rectangular bounding boxes. In our method, the center of a cluster is a pattern that has a basic shape which contains most of the common structural parts (common features) of the other patterns in its corresponding cluster. By adding new samples to a cluster, its center can change. In fact, after adding a new sample to a cluster, the algorithm updates the center to a pattern which has maximum similarity to all the patterns of that cluster.

Figure 73 shows all the cluster centers obtained for digits 4 and 2. Table 14 shows the number of all the cluster centers obtained per digit class. With the above parameter values of $\gamma^*$ and $\delta$, our clustering algorithm was able to find, in total 345 clusters of different shapes, which is much smaller than the number of training samples found in the CENPARMI database (4000). This greatly reduces the computation time, and memory space required for the system, yet it maintains the high performance, using fewer prototypes.



(a) (b)

Figure 72: Two samples of clusters obtained by our algorithm. (a) One of the clusters obtained for digit 4, this cluster has 55 samples. (b) One of the clusters obtained for digit 5, this cluster has 19 samples. For both these clusters our clustering algorithm has identified one of the patterns as the center (representative or prototype, shown in a rectangular bounding box). Here, the center of a cluster is defined as a basic shape which contains most of the common structural parts of the other patterns in that cluster.

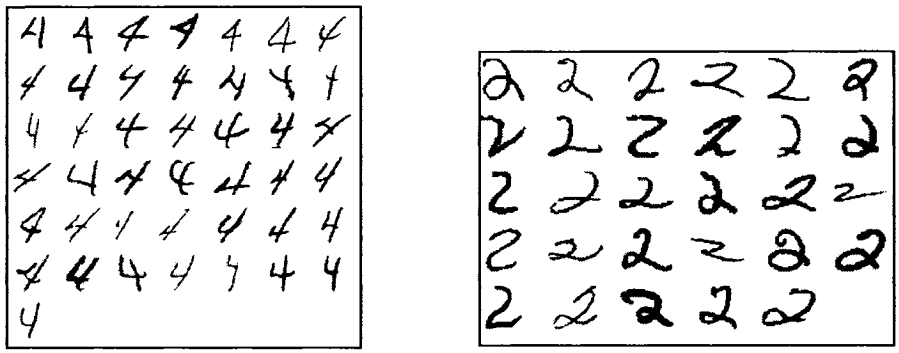
Table 14: Number of clusters (prototypes) per digit class in the training set of the CENPARMI database.

| Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Clusters | 12 | 4 | 29 | 47 | 43 | 32 | 42 | 44 | 57 | 35 | 345 |

The classifier used in our experiments was based on the K-Nearest Neighbor rule: an input digit is compared against all the prototypes (centers of all the clusters for all digits) and the most K similar ones vote for its classification. In our experiments, we took K =3. We applied our classifier to all testing samples (200 samples per class) in the CENPARMI digit database, and the results for all the classes are shown in Table 15. Our overall recognition result on the CENPARMI test set was 98.75%. In Table 16 this result has been compared with the result that reported by Lu et al. [43] using the CENPARMI database. Reference [43] reported one of the highest results in the literature on CENPARMI isolated handwritten digit database.

Table 15: Recognition rates per digit class and overall recognition rate in the testing set of the CENPARMI database.

| Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Recog. Rate (%) | 99.55 | 99.33 | 98.74 | 98.33 | 97.68 | 99.01 | 99.10 | 98.15 | 98.79 | 98.84 | 98.75 |



(a) (b)

Figure 73: (a) Centers of clusters (43 prototypes) found for digit 4, and (b) Centers of clusters (29 prototypes) found for digit 2 in the training set of the CENPARMI isolated handwritten digit database.

Table 16: Performance evaluation of our system in isolated digit recognition on CENPARMI handwritten isolated digit database.

| | Our approach (Our clustering + KNN) | | | Approach in [43] (SVM_rbf) | | |
|---|---|---|---|---|---|---|
| Dataset | Rec.(%) | Err.(%) | Rej.(%) | Rec.(%) | Err.(%) | Rej.(%) |
| CENPARMI Training set | 99.83 | 0.17 | 0 | N.A. | N.A. | N.A. |
| CENPARMI Testing set | 98.75 | 1.25 | 0 | 99.05 | 0.95 | 0 |

As this table shows, the results using our method here is a little bit lower than the results obtained by SVM (using rbf kernels)[43]. However, our goal here was not to reach the maximum recognition rate, but instead to design a system which shows plasticity and stability (high ability for incremental learning) in handwritten isolated digit recognition [19].

## 9.5 Summary

In this Chapter, we considered the problem of incremental learning in the handwritten recognition systems. We presented a generalized set of similarity measures for comparing the shapes of handwritten digits (characters), and we presented a new clustering algorithm for improving plasticity and stability of handwritten character recognition systems. Our clustering algorithm, for new shapes of different digits (or new shapes of the same digits) which are dissimilar, creates new clusters. Our clustering algorithm, adaptively allows the

learned clusters to be merged to make bigger clusters (broader concepts). Also it is able to automatically determine the optimal number of clusters in the input data. Unlike off line clustering algorithms, our method scans the input data just once, so it is normally faster, and it can be used for clustering on line input data. We built a system which is able to incrementally learn new handwritten styles (shapes) of characters, without forgetting the previously learned ones (shows plasticity and stability in handwritten digit recognition).

# Chapter 10

# A Genetic Based Particle Swarm Optimization (GBPSO) Model

In Chapter 6, we proposed an evolutionary method (based on GA's) for searching and optimization of segmentation hypothesis. In this chapter, considering basic properties of GA's and another important evolutionary algorithm (so-called Particle Swarm Optimizer (PSO) [4, 5]), an extension for evolutionary optimization algorithms is proposed. Here, a general model which is called Genetic Based Particle Swarm Optimization (GBPSO) model is developed. In the original GA or PSO methods, the size of the population (swarm) is fixed. In our model, we introduce birth and death as two general population operations (instead of regular evolutionary operations) in order to make the population dynamic. Similar to the natural species that their birth and mortality rates change over time, our model allows oscillations in the size of the populations due to such changes. In our model, as part of its history, the swarm (population) remembers records of its good (high fitness) particles, even after their deaths. Compared to the original Genetic Algorithms and PSO models, our strategy proposes a more natural simulation of the social behavior of intelligent animals. The experimental results show that compared to the original PSO, our GBPSO model can reach broader domains in search spaces and can converge faster in very high dimensional and complex environments. In this chapter, first we give some background information about PSO optimization, then we present the details of our proposed GBPSO as a general evolutionary

optimization model, and finally we show our comparison results.

## 10.1 Particle Swarm Optimization

Here, we give a short description of the main concepts for the Particle Swarm Optimization (PSO) technique. A detailed description of PSO can be found in [4, 5, 161]. PSO was originally designed and introduced by Eberhart and Kennedy in 1995 [4, 5]. The PSO algorithm is an adaptive algorithm, which involves simulating the social behavior of a group of bees, birds or a school of fish. In PSO, individuals (each one is referred to as a particle) fly through a multidimensional search space. Each particle is represented by a point (vector) in the multidimensional search space, and a population of particles (so-called swarm) adapts by moving stochastically towards previously successful regions of the space. A particle swarm has two primary operations: velocity update and position update. At every iteration, each particle accelerates towards the particle's best previous position and the global best position (or the local best position in its neighborhood). In each generation, each particle updates its velocity vector based on its current velocity, the distance from its best previous position, and the distance from the global best position (or the local best position). The new velocity vector of each particle is then used to update its new position vector in the new generation. The PSO process is then iterated a fixed number of times, or until a minimum error based on some performance index is achieved. It has been shown that this simple model can deal with difficult optimization problems efficiently [161]. Some researchers compared PSO and GA's and they have shown that PSO shares the ability of genetic algorithms to handle arbitrary nonlinear functions, but with a much simpler implementation [162]. Some other researchers have also shown that the hybrid GA and PSO possess better abilities to find the global optimum than the standard GA and PSO algorithms [163].

There are two versions of the PSO algorithm: continuous and discrete. The original PSO described above has been developed for solving continuous optimization problems. However, lots of practical engineering problems are formulated as discrete combinatorial optimization problems. In addition, researchers frequently model continuous domain problems in binary

(discrete) terms, and they solve those problems in discrete high dimensional spaces, featuring qualitative distinctions between levels of variables. Typical examples of discrete optimization include problems which require the ordering or permutation of discrete elements, such as scheduling or routing problems. Although PSO have been used recently for solving optimization problems in different domains, the overall situation still seems to be as it was in 2001 [164]. At that time, Kennedy and Eberhart wrote in their book [161]: "...we are looking at a paradigm in its youth, full of potential and fertile with new ideas and new perspectives...researchers in many countries are experimenting with particle swarms...many of the questions that have been asked have not yet been satisfactorily answered". In 1997, Kennedy and Eberhart introduced a discrete (binary) version of PSO for discrete optimization problems [165]. Our literature survey has shown that compared to continuous PSO, binary PSO has not been used extensively for optimization, and it is still at the beginning stages of its development and research.

Although evolutionary algorithms such as GA and PSO have been used extensively, to the best of our knowledge there has been no research on general variable size population models of evolutionary algorithms, and most of the research has focused only on special cases of variations in the populations such as saw-tooth population model in [6]. In this chapter, we propose a general population model for evolutionary algorithms, and we utilized it in order to improve the performance of original PSO model. In our expansion, we add important operations called birth and death to the PSO model, in order to make the population more dynamic. Our model allows a variable population size of PSO, such that the size of the population (swarm) can oscillate or change as a result of the changes in the birth and mortality rates. Also, we add history over generations to our PSO model, so that the model remembers the records of its high fitness individuals even after their death or removal of those individuals from the population. Our proposed GBPSO model can be considered as a unified generalized (hybrid) model for both Genetic Algorithms (GA's) and Particle Swarm Optimization methods [20]. So, it can exhibit advantages of both methods in the searching of complicated search spaces or the optimization of difficult objective functions. We have also applied our proposed GBPSO for optimization of some standard test functions, and

have compared its performance with the original PSO method.

In the next two sections, we will present some background information about two important versions of particle swarm optimization: continuous and binary versions.

## 10.2 Continuous Particle Swarm Optimizer

In this section, we describe the basic concepts and the equations used in PSO model [4, 5, 161]. Like GA, PSO is a population-based search and optimization algorithm, however, in PSO all the members of the population (particles) move around and search for food (the optimum). So, in PSO each particle in the search space has a current location, direction and velocity of movement. Assume that the search space for an optimization problem is d-dimensional. The $i^{th}$ particle of the population (swarm) can be represented by a d-dimensional position vector $X_i = (x_i^1, x_i^2, \ldots, x_i^d)$. The rate of changes in position (velocity) for the $i^{th}$ particle is represented by another d-dimensional vector $V_i = (v_i^1, v_i^2, \ldots, v_i^d)$. The best previously visited position of the $i^{th}$ particle is denoted as $P_{ibest} = (p_i^1, p_i^2, \ldots, p_i^d)$. Also assume that g is the index of the individual in the swarm which currently has the best fitness, therefore its position is denoted by $P_{gbest} = (p_g^1, p_g^2, \ldots, p_g^d)$. For each generation, the particles are manipulated according to the following equations for their velocity and position:

$$v_i^k = v_i^k + c_1\varphi_1(p_i^k - x_i^k) + c_2\varphi_2(p_g^k - x_i^k) \quad (39)$$

$$x_i^k = x_i^k + v_i^k \quad (40)$$

where $c_1$ and $c_2$ are positive constants, and $\varphi_1$ and $\varphi_2$ are random numbers, uniformly distributed between 0, and 1. Some researchers have shown that setting each of $c_1$ and $c_2$ equal to 2 gets the best overall performance. Figure 74 shows the vector representation of these equations. The new velocity and position vectors in Equations 39, and 40 may be updated to take some constraints into account. The most common one for the velocity vector is to check that the maximum velocity in each dimension is within a specified range: $[-v_{i_{max}}^k, v_{i_{max}}^k]$. The most common constraint for a position vector is the interval constraints:

the new position vector has to be within a given interval (continuous, or binary) to be considered inside the search space. The pseudo-code of the basic PSO procedure is given as follows in Figure 75. The PSO procedure described here essentially handles continuous optimization problems. In the next section, we will look at discrete PSO.

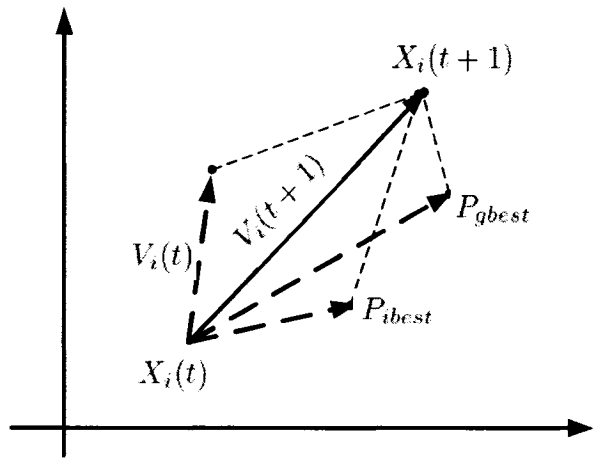



Figure 74: Vector representation of Equations 39 and 40.

## 10.3 Discrete (Binary) Particle Swarm Optimizer

Kennedy and Eberhart developed a discrete binary version of PSO for binary problems [165]. They proposed a model wherein the probability that a particle will decide "yes" or "no" ("true" or "false"), or will make some other binary decision, is a function of personal and social factors as follows:

$$Prob(x_i^k = 1) = f(x_i^k, v_i^k, p_i^k, p_g^k) \quad (41)$$

$$Prob(x_i^k = 0) = 1 - Prob(x_i^k = 1) \quad (42)$$

Here, $Prob(x_i^k = 1)$ is the probability that a particle will choose 1 for its next movement in the dimension $k$ ($k \in \{1, 2, 3, \ldots, d\}$), and $Prob(x_i^k = 0)$ is the probability that a particle

**PSO Procedure**

For each particle in the population
Initialize the particle (its position and its velocity)
End

Repeat
For each particle
Compute its fitness value. If its fitness value is better than its previous best fitness ($P_{ibest}$) set the current value as its new $P_{ibest}$.
End

Find the particle with the best fitness ($P_{ibest}$) and consider it as the best particle of the population ($P_{gbest}$)

For each particle
Compute its velocity: based on Equation 39
Update its position : based on Equation 40
End
Until the maximum number of iterations or stopping criteria are attained.
End PSO Procedure

Figure 75: Pseudo code of original PSO Algorithm.

will choose 0 for its next movement in the dimension $k$. The parameter $v_i^k$ (predisposition for each particle) is calculated similarly to the continuous case based on Equation 39, and it will function as a probability threshold to make one of the two decisions (0 or 1). If $v_i^k$ is higher, the individual is more likely to choose 1, and lower values will yield to the choice of 0. Such a threshold needs to stay in the range of $[0, 1]$. The sigmoidal function shown in Equation 43, and in Figure 76, maps the interval of $v_i^k$ to a range of $[0, 1]$:

$$s(v_i^k) = \frac{1}{1 + \exp(-v_i^k)} \tag{43}$$

Final binary decision making is based on the following rule:

$$x_i^k = \begin{cases} 1 & \text{if} \quad \rho_i^k < s(v_i^k) \\ \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

Here, $\rho_i = [\rho_i^1, \rho_i^2, \rho_i^3, \ldots, \rho_i^d]$ is a random vector which is chosen from a uniform distribution in the interval of $[0, 1]$. As shown in Figure 76, we can limit $v_i^k$ so that $s(v_i^k)$ does not

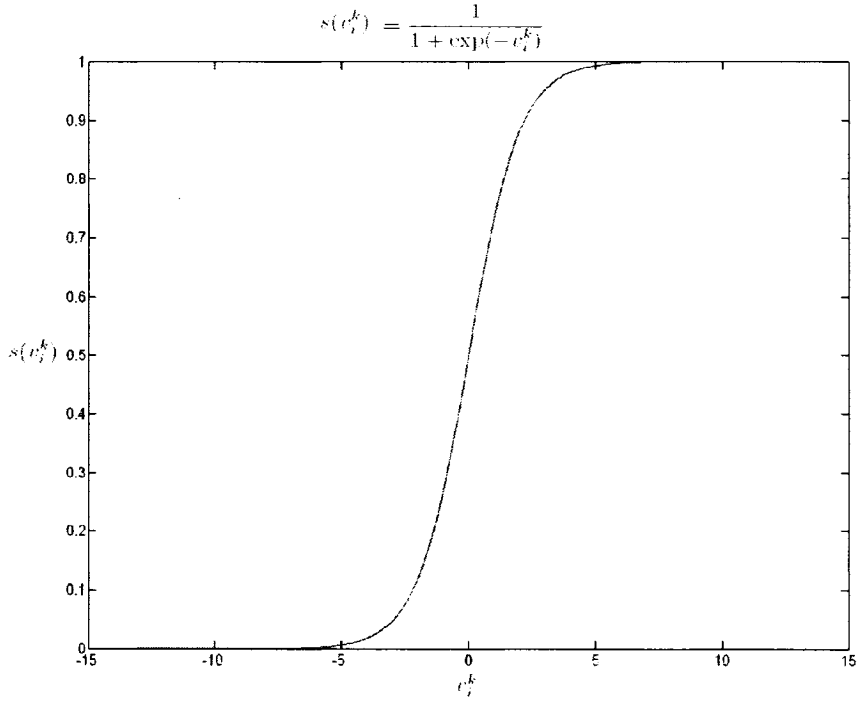



Figure 76: Sigmoid function of: $s(v_i^k) = \frac{1}{1+\exp(-v_i^k)}$

approach too closely to 0 or 1. A constant parameter $V_{max}$ can be set at the start of the algorithm to limit the range of $s(v_i^k)$. Normally, $V_{max}$ is set within the range of $[-4.0, +4.0]$, so that there is always at least a chance (probability $\geq 0.0180$) that a bit state change can occur. The entire PSO algorithm of the binary version is almost the same as that of the basic continuous version in Section 10.2, except that it uses the above decision rule. In the next section, we will introduce our GBPSO model.

## 10.4 Genetic Based Particle Swarm Optimizer

Our Genetic Based Particle Swarm Optimizer (GBPSO) is a generalized evolutionary algorithm (a novel generalization of both PSOs and GAs). In our model, in order to improve the dynamism of the population (swarm), and to increase the exploration power of its particles, we introduce birth and mortality rates into the population [20]. In each generation,

particles (chromosomes) are updated similarly to particles in the PSO method, based on the rules and equations explained in Sections 10.2 and 10.3. However, in each generation after updating all the particles in the swarm, new individuals (children) are added to the swarm by birth operations, and also some particles of the current swarm die, and they are removed from the swarm, by death operation(s). In addition to birth and death rates in our model, we also maintain a history that keeps the information of high performance particles which have been removed from the swarm because of death operation. Dynamic swarm, birth and death operations, and history of the swarm are the main components of our GBPSO model, described in the following subsections.

### 10.4.1 Modeling of a Dynamic Swarm

In this section, we introduce the equations that model our dynamic swarm. Assume that at each generation $t$, the swarm has a birth and a death rate (mortality rate). The Birth rate (which is denoted by $b(t)$) is defined as: the number of births per unit population per unit of time. The death rate or mortality rate (which is denoted by $m(t)$) is defined as the number of deaths per unit population per unit of time. Here, both birth and death rates are considered as non-negative values ($b(t) \geq 0$ and $m(t) \geq 0$). The overall birth rate of the swarm (which is denoted by $r(t)$) can be defined as follows:

$$r(t) = b(t) - m(t) \qquad \text{where: } \forall t \quad b(t),\ m(t) \geq 0 \tag{45}$$

Although, $b(t)$ and $m(t)$ both are non-negative rates, $r(t)$ can be positive, zero, or negative. The population size at generation $t$ is denoted by $P(t)$. The overall birth rate $r(t)$ can affect the size of the population at the next generation ($P(t+1)$) relative to the current size of the population ($P(t)$), as follows:

$$\begin{aligned}
&\text{If } r(t) > 0, \text{ then } P(t+1) \text{ is greater than } P(t)\\
&\text{If } r(t) = 0, \text{ then } P(t+1) \text{ is equal to } P(t)\\
&\text{If } r(t) < 0, \text{ then } P(t+1) \text{ is less than } P(t)
\end{aligned}$$

In order to derive the differential equation, which specifies the population (swarm) size, we consider how the population will change in a small time interval from $t$ to $t+\Delta t$. The change in the population size is directly proportional to the birth and mortality rates, $\Delta t$, and the population size itself, as follows:

[change in population size] = [births] -[deaths]

$$P(t+\Delta t) - P(t) \approx b(t).P(t).\Delta t - m(t).P(t).\Delta t \tag{46}$$

If we divide both sides of Equation 46 by $\Delta t$, and let $\Delta t$ approach zero, we can get the following differential equation, which models the size of the population (swarm) in different generations:

$$\begin{aligned} P^{'}(t) &= (b(t) - m(t))P(t) \\ P^{'}(t) &= r(t)P(t) \end{aligned} \tag{47}$$

This differential equation has a unique solution as follows, where $c$ is a constant that depends on the initial size of the population at time $t = 0$ (denoted by $P(0)$).

$$P(t) = e^{R(t)}, \quad \text{where:} \quad R(t) = \int r(t)dt + c \tag{48}$$

Without loss of generality and in order to be able to find the solution to Equation 48, we have considered four different cases for the population and its birth rate ($b(t)$) and mortality rate ($m(t)$) as follows:

### Case I: Population with Constant Size

If both $b(t)$ and $m(t)$ are chosen to be equal such that $b(t) = m(t)\ \forall t \geq 0$, then $r(t) = 0$, therefore, according to Equation 48, the population will be a constant size as follows:

$$P(t) = P(0) = \text{Constant} \tag{49}$$

In particular, if $b(t) = m(t) = 0$, then we have the conventional PSO model (constant population size with no births and deaths). So, the original (conventional) PSO model is a special case of our GBPSO model.

**Case II: Population with Exponential Changes in Size**

If both $b(t)$ and $m(t)$ have different values such that $b(t) = c_1$ and $m(t) = c_2$ and $c_1 \neq c_2$, then $r(t)$ and $P(t)$ will be defined as follows:

$$r(t) = b(t) - m(t) = c_1 - c_2 \tag{50}$$

$$P(t) = e^{(c_1 - c_2)t + c} = P(0) \cdot e^{(c_1 - c_2)t} \tag{51}$$

In this case, the population size will increase or decrease exponentially (depending on the relative values of $b(t)$ and $m(t)$).

**Case III: Population with Exponential-Sinusoidal Changes in Size**

If both $b(t)$ and $m(t)$ have general forms as follows:

$$b(t) = \alpha_1 \cos(\omega t + \theta_1) + \beta_1, \qquad \text{where: } \beta_1 \geq \alpha_1 \geq 0 \tag{52}$$

$$m(t) = \alpha_2 \cos(\omega t + \theta_2) + \beta_2, \qquad \text{where: } \beta_2 \geq \alpha_2 \geq 0 \tag{53}$$

Here, $\alpha_1, \alpha_2, \beta_1, \beta_2,$ and $\omega$ are positive real numbers and they are normally chosen to be very small; and $\theta_1, \theta_2 \in [0, 2\pi]$. The condition $\beta_j \geq \alpha_j$ guarantees that both $b(t)$, and $m(t)$ will have non-negative values in all generations. If $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$ and $\theta_1 = \theta_2$, this case will become Case I above. If $\alpha_1 = \alpha_2 = 0$, we have constant birth and mortality rates and this case will become Case II above. In the general form, by substituting $b(t)$, and $m(t)$ from Equations 52 and 53 into Equation 45, we can find $r(t)$ as follows:

$$r(t) = \alpha \cos(\omega t + \theta) + \beta \tag{54}$$

Here, $\beta = \beta_1 - \beta_2$, and $\alpha$, and $\theta$ can be easily derived from $\alpha_1, \alpha_2, \theta_1,$ and $\theta_2$ (refer to Appendix C). After replacing the $r(t)$ in the Equation 48, we can find the solution for the differential equation, as follows:

$$P(t) = P(0)e^{\left[\frac{\alpha}{\omega}\sin(\omega t+\theta)+\beta t-\frac{\alpha}{\omega}\sin(\theta)\right]} \tag{55}$$

Considering this model for the population, we can simulate many different situations for our population. For example, if both $\alpha$, and $\beta$ are equal to zero, the population will always be fixed at $P(0)$ (conventional population models). If $\alpha$ is equal to zero, the size of the population will increase or decrease exponentially, corresponding to the sign of $\beta$ (+ or -, respectively). If $\beta$ is equal to zero, the size of the population will oscillate starting from the initial size ($P(0)$). If $\alpha$, and $\beta$ are both non-zero, the size of the population will oscillate, and it will increase (or decrease) exponentially. In all of these cases, parameter $\omega$ determines the speed (frequency) of the oscillations. The two examples below graphically show these situations.

**Example 1:**

In this example, we assume that the initial size of the population is equal to $P(0) = 30$, and we also assume that the birth and mortality rates of the population ($b(t)$, and $m(t)$) have been chosen as the following functions:

$$b(t) = 0.012\cos(0.03t + \pi/12) + 0.5 \tag{56}$$

$$m(t) = 0.019\cos(0.03t + \pi/3) + 0.5 \tag{57}$$

Oscillations or changes of these rates are shown in Figure 77-(a). Using the parameter values in the above equations of $b(t)$, and $m(t)$, we can obtain $\alpha = 0.0135$, $\beta = 0$, and $\theta = 25\pi/16$, which are required for the population function in Equation 55. By replacing these values, we can obtain the population size as a function of $t$ as shown in Equation 58. The graph of this function is also shown in Figure 77, where the size of the population is initialized at 30, and it oscillates while its minimum is fixed around 30, and its maximum is fixed at 73.

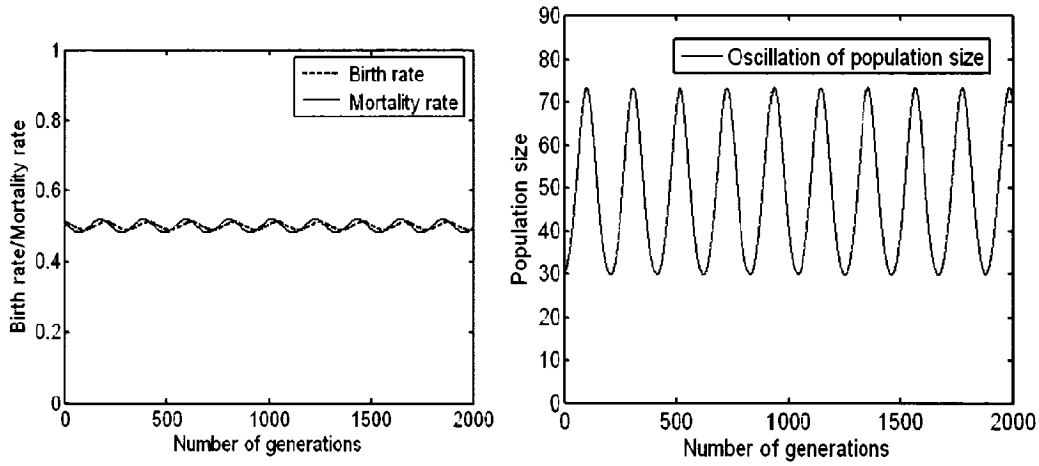



Figure 77: (a) Oscillation of the birth and mortality rates in Example 1, (b) Oscillation of the population size in Example 1.

$$P(t) = 30e^{\left[0.45\sin(0.03t+\frac{25\pi}{16})-0.45\sin(\frac{25\pi}{16})\right]} \tag{58}$$

**Example 2:**

In this example, again we consider the same initial size for the population ($P(0) = 30$), and we choose slightly different birth and mortality rates for the population ($b(t)$, and $m(t)$) from those in the first example, as follows:

$$b(t) = 0.012\cos(0.03t + \pi/12) + 0.5 \tag{59}$$

$$m(t) = 0.019\cos(0.03t + \pi/3) + 0.4999 \tag{60}$$

Oscillation of these rates are shown in Figure 78. Using the parameter values in the above equations of $b(t)$, and $m(t)$, we can obtain $\alpha = 0.0135$, $\beta = 0.0001$, and $\theta = 25\pi/16$. The population function is shown in Equation 61, and the graph of this function is also shown in Figure 78. As shown in this figure, the size of the population is initialized by 30 and it is oscillating, while its minimum, maximum, and the amplitude of its oscillations are slowly

increasing. This population model can be useful in cases where, after many generations, we want to gradually increase the number of particles in the swarm (adaptively using more particles to search a complicated landscape).

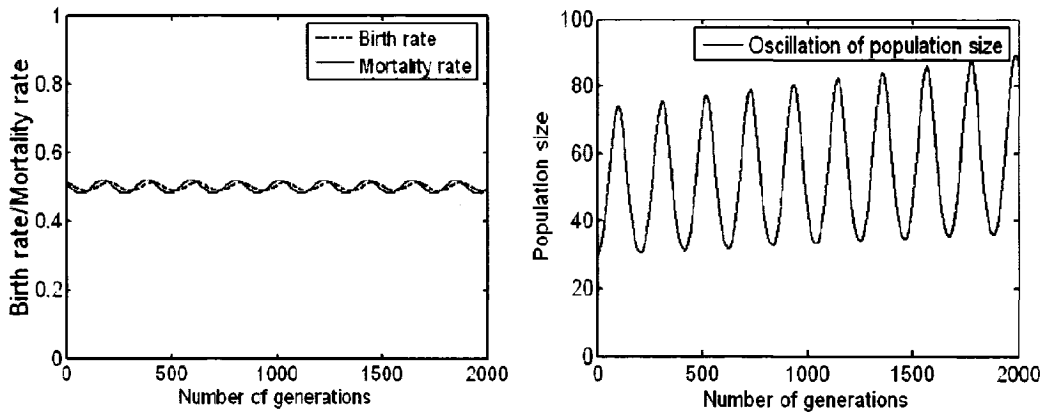



Figure 78: (a) Oscillation of the birth and mortality rates in Example 2 (these oscillations are slightly different from Example 1), (b) Oscillation of the population size in Example 2 (here, the population size is gradually increasing).

$$P(t) = 30e^{\left[0.45\sin(0.03t+\frac{25\pi}{16})+0.0001t-0.45\sin(\frac{25\pi}{16})\right]} \tag{61}$$

### Case IV: Population with Linear Changes in Size

If $g(t)$ is a positive valued ($\forall t \geq 0 : g(t) > 0$), and if it is differentiable ($\forall t \geq 0 : g'(t)$ exists), in this case it is possible to choose $b(t)$ and $m(t)$ such that:

$$r(t) \;=\; b(t) - m(t) = \frac{g'(t)}{g(t)}, \qquad \forall t \geq 0 \tag{62}$$

In this case, according to Equation 48, $P(t)$ can be calculated as follows:

$$P(t) \;=\; e^{\ln(g(t))+c} = e^c \cdot g(t) = k_1 \cdot g(t) \quad (k_1 : \text{constant, depends on } P(0)) \tag{63}$$

In this case, the population form is dependent on the form of the function $g(t)$. For example, if $g(t)$ is linear, polynomial, or exponential, the form of $P(t)$ also will be linear, polynomial, or exponential, respectively. Consider the following example of a linear $P(t)$,

**Example 3:**

Assume that $b(t)$ and $m(t)$ have been chosen as follows, and the initial population size is $P(0) = 20$.

$$b(t) = \frac{18}{10t + 20} \tag{64}$$

$$m(t) = \frac{8}{10t + 20} \tag{65}$$

then $r(t)$ will be calculated as follows:

$$r(t) = b(t) - m(t) = \frac{10}{10t + 20} \tag{66}$$

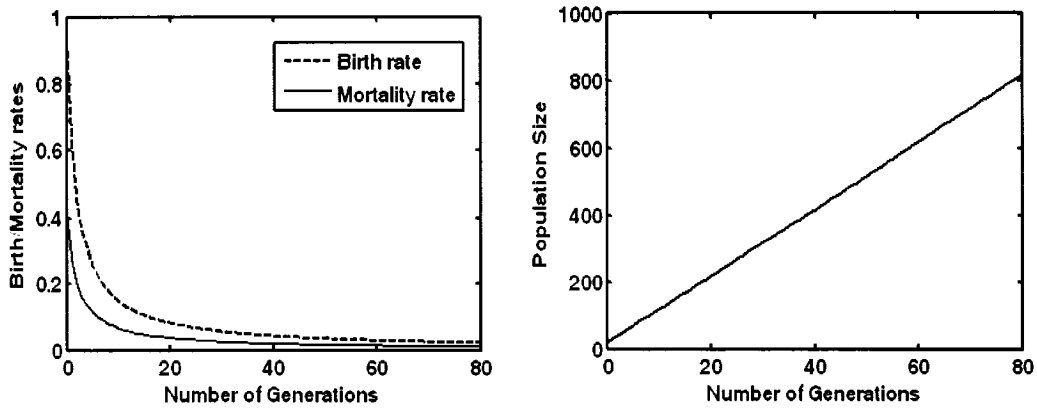
According to Equation 48, $R(t)$ and $P(t)$ can be calculated as follows:

$$R(t) = \ln(10t + 20) + c \tag{67}$$

$$P(t) = e^{\ln(10t+20)} = 10t + 20 \tag{68}$$

Figure 79 shows the graph of the changes of $b(t)$, $m(t)$ and $P(t)$ for this example. As seen in this figure the population size starts from 20 and it grows linearly (increases by 10 in each generation).

As shown in Cases I to IV, in our GBPSO model, the population size can change in different forms, and we are able to simulate any form of population function, such as constant, exponential, sinusoidal, linear, etc. In the following sections, we will model birth and death processes, and the history of the population.




(a) (b)

Figure 79: (a) Birth and mortality rates in Example 3, (b) Population size changes linearly in Example 3.

### 10.4.2 Modelling of Birth Operations

In our GBPSO model, unlike PSO, there are birth operations. At each generation $t$, a subset of the particles in the swarm are randomly selected, in order to produce new children (birth). The particles in the selected subset are given equal chances to mate or mutate in order to produce new children. The new children will be included in the swarm and new information records (data structures) for them will be created and initialized. Here, unlike conventional Genetic Algorithms, the new children do not replace their corresponding parents, nor do they compete with their parents. In fact, here, the relationship between parents and children is some kind of collaboration or cooperation for reaching a shared goal.

At each generation $t$, the number of particles which are added to the swarm as new born particles are proportional to the birth rate $b(t)$ of the swarm. For example, at generation $t$, the number of particles that are born is calculated as follows:

$$num\_of\_new\_children = \lfloor P(t).b(t) \rfloor \tag{69}$$

Similar to genetic algorithms [124, 125], here, there are two different ways of producing new children: crossover and mutation. So, we can assume that we have two partial birth rates:

one birth rate for the children produced by crossover $b_c(t)$, and another birth rate for the children produced by mutation $b_m(t)$, such that:

$$b(t) = b_c(t) + b_m(t), \quad \text{where} \quad b_c(t) = k_2.b_m(t) \tag{70}$$

Normally, the percentage of children produced by crossover is higher than the percentage of the children produced by mutation. Here, we assume that $k_2 \in [1, 10]$ is a constant ratio of $b_c(t)$, and $b_m(t)$ (as a typical value in our experiments we took $k_2 = 3$). Having $k_2$ and $b(t)$, we can compute $b_c(t)$, and $b_m(t)$. Therefore, we can compute the number of parents for producing crossover and mutated children separately, as follows:

$$num\_of\_crossover\_children(t) = \lfloor P(t).b_c(t) \rfloor \tag{71}$$

$$num\_of\_mutated\_children(t) = \lfloor P(t).b_m(t) \rfloor \tag{72}$$

$$crossover\_parents(t) = \left\lfloor \frac{\lfloor P(t).b_c(t) \rfloor}{2} \right\rfloor \tag{73}$$

$$mutated\_parents(t) = \lfloor P(t).b_m(t) \rfloor \tag{74}$$

So, from the population at each generation $t$, the number of $crossover\_parents(t)$ (pairs of particles) will be randomly chosen in order to produce crossover children (each pair of parents produces two crossover children). Also, $mutated\_parents(t)$ particles will be randomly chosen in order to produce mutated children.

### 10.4.3 Modeling of Death Operation

In our GBPSO model, unlike PSO, there is death operation. At each generation $t$, a subset of the particles in the swarm are randomly selected (without considering their age or fitness), in order to be removed from their swarm (death). In fact, this process simulates removing those individuals in nature from their population which die for many different reasons such as: disease causing death, natural disasters, sudden changes in the environment, lack of food, or being hunted by predators, etc. The number of particles in the selected subset for death at each generation is proportional to the death rate $m(t)$ of the swarm. For example, at

generation $t$, if the size of the population is denoted by $P(t)$, then the number of particles that must have died is calculated as follows:

$$num\_of\_death(t) = \lfloor P(t).m(t) \rfloor \tag{75}$$

In order to simulate death, at each generation $t$, the $num\_of\_death(t)$ particles from the current swarm will be randomly chosen, and they will be marked as dead particles (their records can be physically removed from the swarm, as well). The particles in the selected subset are logically considered as dead, so there will be no further updates for their records (their positions and their speeds are not updated). In the next section, we will explain how a summary of the information of dead particles is saved in the history of the swarm.

### 10.4.4 Modeling the History of the Swarm

Unlike the original PSO, there is a history in our GBPSO algorithm, which saves the information of the best individuals which have died. At each generation, records of those particles which are marked as dead can be physically removed from the swarm. At the same time, the history of the swarm is updated to keep the information on the best particles(s) that have dead. Therefore, if the best particle ($P_{gbest}$: particle with the highest fitness) is among those selected for death, its record will be kept in the history. In this way, the swarm can save the knowledge gained during past generations. In fact, at any generation, live particles of the swarm can use the knowledge, and the experiences of the best particle ($P_{gbest}$), even if it has died. This is a bit similar to elitism in Genetic Algorithms (GA's). Elitism in GA's ensures that the genes or information on the best individual(s) is always copied to the subsequent population, and in this way, good individuals will not be accidentally lost when combined with other individuals [6]. However, in our method, we keep the information on dead good individuals (high performance particles which died) in the history, not in the swarm.

### 10.4.5 Pseudo Code of Our GBPSO Algorithm

In this section, we put all the main elements of our GBPSO model (dynamic swarm, birth and death operations, and history) together, and we show a pseudo code of our algorithm. The general pseudo code of our GBPSO algorithm is shown in Figure 80. In the next section, we show the experimental results with our algorithm.

```
Initialization;
while (not terminated)
{
t = t + 1;    // increment number of generations
for i = 1 : Number of particles   // (for all particles of (P(t))
{
V_i(t+1) = V_i(t) + c_1 * rand() * (P_ibest - X_i(t)) + c_2 * rand() * (P_gbest - X_i(t));
X_i(t+1) = X_i(t) + V_i(t+1);
Fitness_i(t) = f(X_i(t));   // f() is the fitness or objective function
if required, update P_ibest;
}   // end of for
do the birth operation;    // Number of births: P(t) * b(t)
do the death operation;    // Number of deaths: P(t) * m(t)
update the swarm size (P(t))  // update the Number of particles
update P_gbest and history of the swarm (if required);
}   // end of while
Output the best particle found.
```

Figure 80: Pseudo code for our GBPSO Algorithm.

## 10.5 Experimental Results

In this section, we will compare the performance of our proposed GBPSO method with the original PSO proposed by Kennedy and Eberhart [165]. For comparison, we investigated both methods on the minimization of a set of four standard test functions: Sphere, Rosenbrock, Griewangk, and Rastrigrin. Each of these four functions has a global minimum value equal to zero. The expressions of each of these functions for $N$ input variables are given below, and their graphs for 2 input variables are shown in Figure 81.

$$f(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{x_i^2} \qquad \text{(Sphere)} \tag{76}$$

$$f(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{N-1}} (\mathbf{100}(\mathbf{x_{i+1}} - \mathbf{x_i^2})^{\mathbf{2}} + (\mathbf{x_i} - \mathbf{1})^{\mathbf{2}}) \text{ (Rosenbrock)} \tag{77}$$

$$f(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{4000}} \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{x_i^2} - \prod_{\mathbf{i=1}}^{\mathbf{N}} \cos \frac{\mathbf{x_i}}{\sqrt{\mathbf{i}}} + \mathbf{1} \qquad \text{(Griewangk)} \tag{78}$$

$$f(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} (\mathbf{x_i^2} - \mathbf{10}\cos(\mathbf{2}\pi\mathbf{x_i}) + \mathbf{10}) \qquad \text{(Rastrigrin)} \tag{79}$$

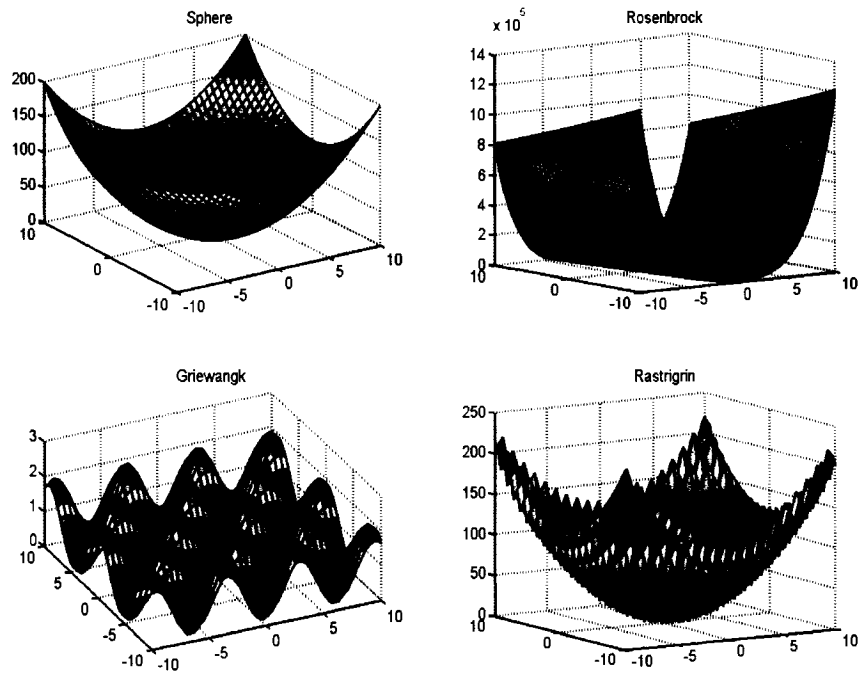



Figure 81: Graphs of four standard test functions for evolutionary algorithms (Sphere, Rosenbrock, Griewangk, Rastrigrin) in two dimensions (two input variables) [4, 5, 6].

These functions have been used by many researchers as benchmarks for evaluating and comparing different optimization algorithms [4, 5, 6]. Here, $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_N})$, is a position vector in the $N$ dimensional search space. Like PSO or GA's, in our GBPSO, each particle is represented by a vector of $N$ values (a point in $N$ dimensional search space) or we can say

by a chromosome which has $N$ genes. In our experiments, the range of the particles in each dimension was set equal to $[-50, 50]$. We tested both PSO, and GBPSO methods in finding the global minimum of these functions, in both low and high dimensional search spaces such as: $N = 3, 15, 75,$ and 150 dimensions.

In our experiments, with GBPSO, the birth rate ($b(t)$) and mortality ($m(t)$) rate of the population were set by Equations 80 and 81 below, and the size of the population was initially set at $P(0) = 1$. Considering these settings, the size of the population in GBPSO changed according to Equation 55, which is shown as a graph in Figure 82.

$$b(t) = 0.15\cos(0.6t + \pi/12) + 0.43531 \quad (80)$$

$$m(t) = 0.15\cos(0.6t + \pi/3) + 0.361 \quad (81)$$

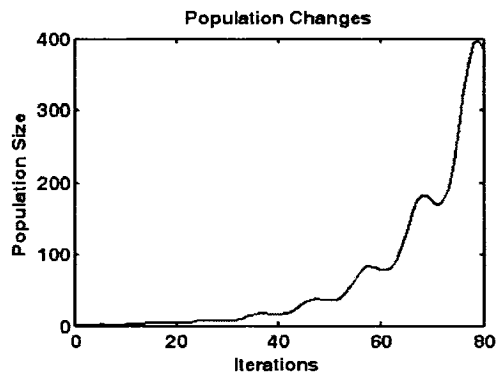



Figure 82: Population changes for our GBPSO model in one run (80 iterations).

For each function and for each value of $N$ (dimension of the space), we ran PSO and GBPSO algorithms 10 times, and each run looped 80 generations. As mentioned above, and as shown in Figure 82, the number of particles in our GBPSO model started at one, and it gradually increased, such that its maximum finally reached 397 particles (after 80 generations/iterations). To have a better comparison, we took the population size of 400 for the ordinary PSO (which is slightly bigger than the maximum number of particles in our GBPSO model), and also of 800 (which is around two times of the maximum number of particles in our GBPSO model).

In each run, we looked at the best optimum value of objective functions (test functions) found by both algorithms, and we took the average of the best optimum values found for each algorithm during 10 runs. Results are summarized in Table 17 for each dimension. The convergence of the original PSO and GBPSO algorithms for the case of N=150 (a very high dimensional space) is also compared in Figure 83. As Table 17 and Figure 83 show in nearly all the cases our GBPSO coverage faster (in terms of number of generations) and find a better minimum.

| Method | Population Size | Sphere | Rosebbrock | Griewangk | Rastringrin |
|---|---|---|---|---|---|
| PSO | 400 (fixed) | 1.22E-013 | 9.15E-011 | 9.91E-007 | 0.9950 |
| PSO | 800 (fixed) | 3.88E-015 | 3.00E-012 | 4.99E-009 | 3.64E-009 |
| GBPSO | 1→ 397 | 2.15E-014 | 2.52E-008 | 2.24E-013 | 3.92E-011 |

(a) Dimension $N = 3$

| Method | Population Size | Sphere | Rosebbrock | Griewangk | Rastringrin |
|---|---|---|---|---|---|
| PSO | 400 (fixed) | 3.23E-004 | 23.59 | 0.0543 | 56.72 |
| PSO | 800 (fixed | 3.48E-005 | 14.43 | 0.0197 | 51.73 |
| GBPSO | 1→ 397 | 7.29E-012 | 13.83 | 6.96E-013 | 2.69E-010 |

(b) Dimension $N = 15$

| Method | Population Size | Sphere | Rosebbrock | Griewangk | Rastringrin |
|---|---|---|---|---|---|
| PSO | 400 (fixed) | 371.55 | 5.12E+005 | 1.0951 | 1.67E+003 |
| PSO | 800 (fixed | 258.87 | 1.30E+005 | 1.0873 | 1.41E+003 |
| GBPSO | 1→ 397 | 3.44E-011 | 73.78 | 6.21E-011 | 8.84E-008 |

(c) Dimension $N = 75$

| Method | Population Size | Sphere | Rosebbrock | Griewangk | Rastringrin |
|---|---|---|---|---|---|
| PSO | 400 (fixed) | 1580 | 2.38E+006 | 1.3315 | 2.5E+003 |
| PSO | 800 (fixed | 756.91 | 2.12E+006 | 1.26 | 1.6E+003 |
| GBPSO | 1→ 397 | 5.38E-011 | 148.78 | 2.25E-013 | 2.2E-008 |

(d) Dimension $N = 150$

Table 17: Comparison of the best average minimum found by original PSO and our GBPSO method for (a) $N = 3$, (b) $N = 15$, (c) $N = 75$, (d) $N = 150$ dimensions in 80 iterations

## 10.6 Discussion

One of our main goals in this research was motivating Evolutionary Algorithm (EA) researchers to further investigate varying population sizes and to show the important positive

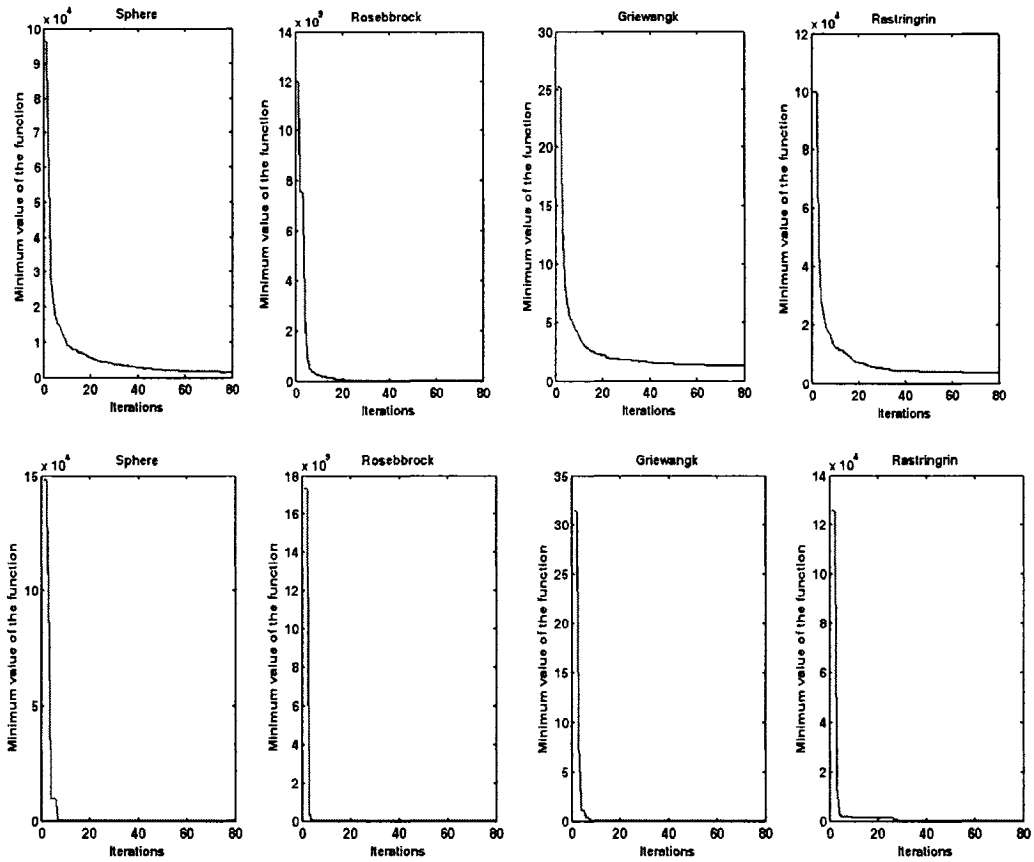



Figure 83: Comparison of the convergence of original binary PSO, and GBPSO on four different test functions for N=150 dimensions.

effects of births and deaths in the population in evolutionary algorithms. In most EA publications, the population size is considered to be a fixed parameter. However, there are biological and experimental arguments to explain why adjusting the population size is more promising. Normally, in natural environments, population sizes of animals change and tend to stabilize around appropriate values according some natural resources and the capacity of ecosystems [166], [167]. Here, for the first time (to the best of our knowledge), we proposed and simulated an effective, general and easy to use population model that can be utilized in all evolutionary algorithms (such as PSO, GA's...). Table 18 compares some basic features of GBPSO with two other important evolutionary algorithms: GA and PSO.

Our experimental results showed that introducing birth and death operations and utilizing

Table 18: Comparison of the main features of three evolutionary optimization algorithms: GA, PSO, and GBPSO.

| Property of the Algorithm | GA | PSO | GBPSO |
|---|---|---|---|
| Population based | Yes | Yes | Yes |
| Updating the population in each generation | Yes | Yes | Yes |
| Evolutionary operators (crossover, mutation,...) | Yes | No | Yes |
| Interaction between particles | No | Yes | Yes |
| Memory (each particle remembers and learns from its past) | No | Yes | Yes |
| Learning (particles can learn from other particles and history) | No | Yes | Yes |
| Birth/Death operations | No | No | Yes |
| Variable population size (General population model) | No | No | Yes |

variable size population model can greatly improve the performance and convergence of EA's in optimization, especially in high dimensional or complicated search spaces. As shown by the results in Table 17 and Figure 83, GBPSO converges much faster, and finds a much better optimum than ordinary PSO for the standard test functions in very high dimensional search spaces, however because of the birth and death operations (updating the population and the history in each generation) it needs more computations compare to PSO.

## 10.7 Summary

In this chapter, we explored the extension of the particle swarm optimization (PSO) model by including features from Genetic Algorithms in order to improve its performance in searching and optimization. The main contribution of our model is to generalize PSO and Genetic Algorithms, and to put them into a unified framework. A second contribution was to propose a simple model for variable population size models of evolutionary algorithms through the use of birth and death operations.

In our future research, we intend to apply our proposed approach (GBPSO) to a variety of real-world problem domains such as searching and optimization of large graphs, and segmentation of numeral strings in different languages such as Latin, Arabic and Farsi. We also intend to explore how this model may adaptively can use more information from the

past history of the swarm in order to control and adjust its birth and mortality rates in the future generations.

# Chapter 11

# Segmentation and Recognition of Handwritten Numerals in Farsi

Segmentation and recognition of unconstrained handwritten numeral strings is one of the main challenges in the automatic processing of many scripts (languages) in the world. Although several methods have been proposed for the recognition of isolated handwritten Farsi (Persian) digits, such as [23, 168, 169, 170]..., to the best of our knowledge, no method has been reported for the separation of touching digits and the construction of segmentation paths in handwritten Farsi numeral strings. In this chapter, a brief history of the evolution of Farsi script is presented and the historic connections between Farsi and Arabic Scripts are shown. Similarities and dissimilarities between Farsi digits (used in Iran) and Indian digits (used in Arab countries) from the Optical Character Recognition (OCR) point of view are shown [22, 25]. Due to a lack of standard databases for OCR research on Farsi, we created a novel standard comprehensive database for Farsi script to facilitate the OCR research in Farsi [21]. Our database includes different sets for isolated digits, letters, numerical strings, legal amounts (used for cheques), and dates. Also, for the first time, we applied Support Vector Machines (SVMs) for the recognition of isolated handwritten Farsi numerals [23], and we tested our segmentation method for separating touching handwritten digits in Farsi [24].

## 11.1 An Introduction to Farsi (Persian) Script and its relation to Arabic

Farsi (Persian) and Arabic are two important cursive scripts used mainly in the Middle East and some other neighboring countries. Farsi is the main language used in Iran and Afghanistan, and it is spoken by more than 110 million people, including some people in Tajikistan, and Pakistan. Arabic is spoken in all Arab countries, both in the Middle East and in Africa, and it is used by 234 million people worldwide. In western countries, it is commonly thought that Farsi and Arabic scripts are the same. However, apart from major differences in the two languages, minor yet important differences exist in their alphabets and their styles of writing of their scripts. Due to these differences, a system adjusted for automatic recognition of one script might not perform well for the other one. While much research on Arabic recognition has been published and introduced internationally, most of the research in Farsi recognition has been presented only in Farsi Journals and Iranian conferences [22, 25]. To the best of our knowledge, research on handwritten Farsi script recognition has not yet been widely introduced to the research community and compared to English script, it is still at the beginning stage. In the rest of this section, we briefly review the history of Persian script and its historic connection to the Arabic script, and review evolution of the Persian script over three periods: Old, Middle, and Modern.

### 11.1.1 Old Persian Script (550 to 330 B.C.)

Old Persian script was a cuneiform type script dating from the time of the Achaemenid dynasties in Persia (6th-4th century B.C.) [22, 25, 75]. In that script, characters were made of strokes, which could be impressed upon soft materials by a stylus with an angled end. Figure 84 shows the alphabet and numbers used in Old Persian script.

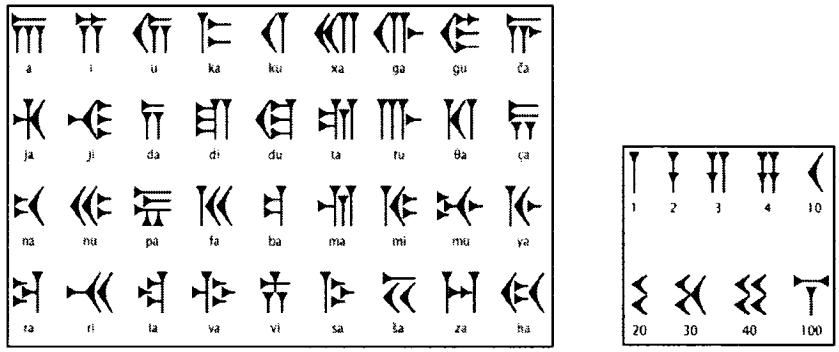


Figure 84: Old Persian (cuneiform type): alphabet and numbers.

### 11.1.2 Middle Persian (300 B.C. to 900 A.D.)

Middle Persian includes the Iranian dialects as they appeared from about 300 B.C. to about 900 A.D. Middle Persian is generally called Pahlavi (a derivative of the old Persian word 'Parthian') [22, 25, 75]. It was the language of quite a large body of Zoroastrian literature, the state religion of the Sassanids in Iran. An example of Pahlavi Script is shown in Figure 85.

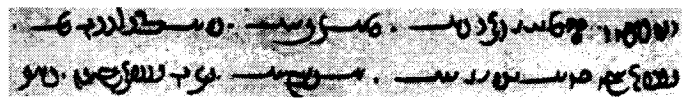


Figure 85: A sample of Pahlavi (Middle Persian) script.

### 11.1.3 Modern Persian Script (after 7th century)

After the fall of the Sassanids and conquest of Persia (Iran) by the Arab Muslims (in the 7th century - 650 A.D), Pahlavi script gradually gave way to the Arabic script [22, 25, 75]. The introduction of Islam brought a massive infusion of loaned words from Arabic to the Persian language. The Arabic alphabet was gradually adopted as the Persians' new alphabet, as the script of politics, religion, and education. Figure 86 shows Modern Persian (Farsi) alphabet. An example of modern Persian script is also shown in Figure 87.

| Isolated | Initial | Medial | Final | Roman | Name | Isolated | Initial | Medial | Final | Roman | Name |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ا | ا | ـا | ـا | á | alef | ص | صـ | ـصـ | ـص | ṣ | sád |
| ب | بـ | ـبـ | ـب | b | be | ض | ضـ | ـضـ | ـض | ḍ | zád |
| پ | پـ | ـپـ | ـپ | p | pe | ط | طـ | ـطـ | ـط | ṭ | tá |
| ت | تـ | ـتـ | ـت | t | te | ظ | ظـ | ـظـ | ـظ | ẓ | zá |
| ث | ثـ | ـثـ | ـث | th | se | ع | عـ | ـعـ | ـع | ‘ | ayn |
| ج | جـ | ـجـ | ـج | j | jim | غ | غـ | ـغـ | ـغ | gh | ghayn |
| چ | چـ | ـچـ | ـچ | ch | che | ف | فـ | ـفـ | ـف | f | fe |
| ح | حـ | ـحـ | ـح | ḥ | ḥe | ق | قـ | ـقـ | ـق | q | qáf |
| خ | خـ | ـخـ | ـخ | kh | khe | ک | کـ | ـکـ | ـک | k | káf |
| د | د | ـد | ـد | d | dál | گ | گـ | ـگـ | ـگ | g | gáf |
| ذ | ذ | ـذ | ـذ | dh | zál | ل | لـ | ـلـ | ـل | l | lám |
| ر | ر | ـر | ـر | r | re | م | مـ | ـمـ | ـم | m | mím |
| ز | ز | ـز | ـز | z | ze | ن | نـ | ـنـ | ـن | n | núm |
| ژ | ژ | ـژ | ـژ | zh | zhe | و | و | ـو | ـو | v/ú | váv |
| س | سـ | ـسـ | ـس | s | sin | ه | هـ | ـهـ | ـه | h | he |
| ش | شـ | ـشـ | ـش | sh | shin | ی | یـ | ـیـ | ـی | y/i | ye |

Figure 86: Farsi alphabet derived from Arabic alphabet, the letters which have been indicated by arrows were added to Arabic alphabet in order to form Farsi alphabet.

### 11.1.4 Farsi Handwritten Digits and Numeral Strings

Farsi and Indian (Hindi) digits look very similar; however, there are minor but important differences between their handwritten digits [23]. Indian (Hindi) digits are used in most Arab countries for writing numeral strings. Farsi digits are used mainly in Iran, and they normally form 13 classes of shapes because of the two different ways of writing the numerals 0, 4, and 6. However, Indian (Hindi) digits normally form 11 classes because of the two different ways of writing the number 3. Also the number 5 is written differently in these two scripts. Figure 88 compares Arabic, Farsi and Indian (Hindi) handwritings of digits. Note that Arabic digits (used mainly in Latin and English speaking countries), are written differently from Farsi and (Indian) Hindi digits. In Figure 89, samples of numeral strings written in Farsi, Indian, and Arabic digits can be seen and compared.

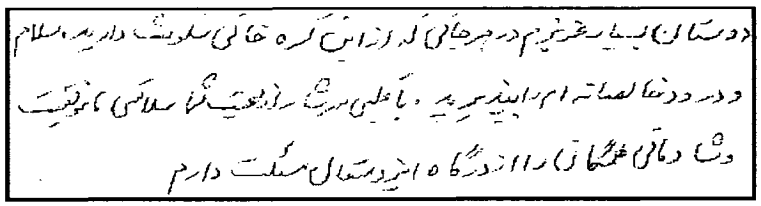


Figure 87: A sample of Modern Persian script.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (a) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 3 | 4 | 6 |
| (b) | ۰ | ۱ | ۲ | ۳ | ٤ | ٥ | ٦ | ٧ | ٨ | ٩ | ٥ | | ۴ | ۶ |
| (c) | ۰ | ١ | ٢ | ٣ | ٤ | ٥ | ٦ | ٧ | ٨ | ٩ | | ٢ | | |

Figure 88: (a) Arabic digits (used mainly in Latin and English speaking countries), (b) Farsi digits (used mainly in Iran), (c) Indian (Hindi) digits (used in most Arab countries).

## 11.2 Creating a Comprehensive Database for Research on Farsi OCR

Standard databases are essential requirements of research and development in the field of off-line and on-line handwriting recognition. Recently, some databases were released for Arabic handwriting recognition such as CENPARMI Arabic Cheques [7] and IFN/ENIT

| | | | | | | |
|---|---|---|---|---|---|---|
| (a) | ۶۸۰۰۰ | ۴۸۹۹ | ۲۵۰۰ | ۲۷۸ | ۱۴۱ | ۵۴۴ |
| (b) | ٦٨٠٠٠ | ٤٨٩٩ | ٢٥٠٠ | ٢٧٨ | ١٤١ | ٥٤٤ |
| (c) | 68000 | 4899 | 2500 | 278 | 141 | 544 |

Figure 89: (a) Numeral strings in Farsi, [(b), and (c)] the same numeral strings written in Indian digits (used in most Arabic countries) and Arabic digits (used in Latin countries).

handwritten Arabic words [171]. The CENPARMI Arabic Cheques database consists of legal and courtesy amounts on Arabic bank cheques, and isolated handwritten digits. As mentioned in the previous section, there are similarities and dissimilarities between Indian digits (used in Arab countries) and Farsi digits (used in Iran); so CENPARMI's Indian digits databases can not be completely used for research on Farsi handwritten numeral strings recognition. Also, until recently, there was no publicly available Farsi handwritten database for researchers on Farsi OCR. Therefore, we decided to assemble such a database for handwritten Farsi script. Collecting and arranging samples for such a database was a huge effort which was done as a collaborative research work in CENPARMI. We presented our Farsi handwritten database at the International Workshop on Frontiers in Handwriting Recognition (IWFHR) in 2006. In the next paragraph, a summary of the main features of our Farsi database is presented (its details can be found in [21]).

Our comprehensive database consists of 6 different datasets and all its samples were written by 175 individual writers that were randomly chosen from different places, groups of ages, genders, and levels of education in Iran. These writes were randomly divided into three different sets of 105, 20, and 50 writers. The samples from the first set of writers (105 writers) were taken for training sets of all our datasets. The samples from the second set of writers (20 writers) were taken for verifying sets of all our datasets, and the samples from the third set of writers (50 writers) were taken for testing sets of all our datasets. These datasets include: isolated Farsi digits (18000 samples), isolated Farsi alphabet letters (11900 samples), Farsi numeral strings (7350 samples), Farsi dates (175 samples), Farsi legal amounts words (7875 samples), and Arabic digits (3500 samples of digits used in English/Latin language countries). Our database also provides both gray level and binary versions of all the images in all these datasets. Therefore, it can be used for research on both gray level and binary image processing/feature extraction techniques. Table 19, shows details of the distribution of samples in our Farsi database.

Table 19: Distribution of samples in our Farsi database among its 6 different datasets.

| Dataset | Classes / Fields | Total Samples | Training Set | Verifying Set | Testing Set |
|---|---|---|---|---|---|
| Isolated Digits | 10 Classes | 18000 | 11000 | 2000 | 5000 |
| Isolated Letters | 34 Classes | 11900 | 7140 | 1360 | 3400 |
| Numeral Strings | 42 Fields | 7350 | 4410 | 840 | 2100 |
| Dates | 1 Fields | 175 | 105 | 20 | 50 |
| Legal Amount Words | 45 Fields | 7875 | 4725 | 900 | 2250 |
| Isolated Digits (Arabic) | 10 Classes | 3500 | 2100 | 400 | 1000 |

## 11.3 Recognition of Isolated Numerals in Farsi

Like Latin languages (such as English), handwritten Farsi digit recognition is an interesting and challenging problem. However, compared to a Latin languages, little research has been conducted in this area. In the last ten years, some papers were published on Farsi/Arabic digit recognition, and they used different methods for feature extraction such as: geometric moment invariants, Zernike moments [168], shadow coding [169]; and different methods for classification such as: statistical, fuzzy, or neural approaches [168, 169]. To the best of our knowledge, profile features, and SVM classifiers have not been used in Farsi digit recognition. For the first time, we used profile features and SVM classifiers for the recognition of isolated Farsi digits in [23]. The details of profile features and SVM classifiers for Farsi digit recognition can be found in [21, 23]. For the experiments on Farsi/Arabic recognition, first we used the CENPARMI Indian digit database and then we used our newly developed Farsi database. Confusion matrix in Figure 91-(a) shows the results of our experiments on CENPARMI Indian digit database. The overall performance of our system on the test set of CENPARMI Indian digit database was 94.14% with zero rejection. In another experiment, more features were used to represent isolated digits: profiles from four directions; crossing counts; and projected histogram from two directions as shown in Figure 90, and we tested these features in our Farsi database. These results are shown in confusion matrix of Figure 91-(b). The overall recognition performance that we obtained in testing of our Farsi Database

was 97.32% with zero rejection. As seen in these confusion matrices (Figure 91), most of the errors are due to the mis-recognition of zeros with other digits, and this is because of the shape of zero after size normalization is mis-recognized with other digits such as '1' or '5' (or vice versa).

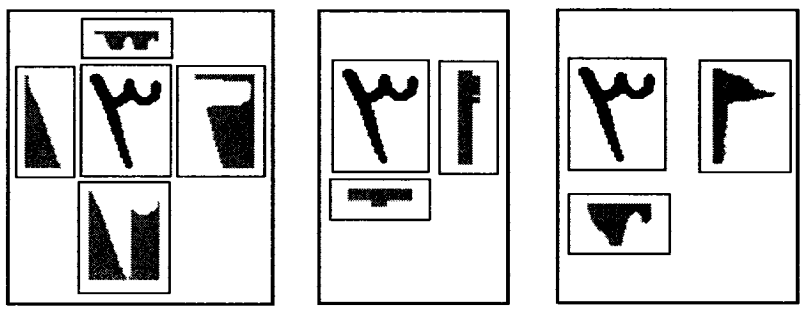



Figure 90: An example of feature extraction for a digit (3): (a) Profiles, (b) Crossing counts, (c) Projection histograms.

## 11.4 Segmentation of Numeral Strings in Farsi

For the first time, we considered the problem of separating touching digits in handwritten numeral strings in Farsi, and we presented our results in [24]. The method that we introduced in Chapter 5 was applied to touching digits of Farsi numeral strings. Figures 92, 93, and 94 show examples of feature extraction in the foreground and background and construction of segmentation path(s) for Farsi handwritten numeral strings.

For testing our segmentation algorithm on Farsi numerals, we collected a set of 100 touching pairs of Farsi digits from the images in our Farsi database. Results of our algorithm showed that our algorithm was able to correctly segment 75% of the samples in our testing set (100 touching pairs), for the rest of the samples (25%), the algorithm was not able to find the correct segmentation path.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1525 | 31 | 0 | 1 | 0 | 11 | 1 | 0 | 2 | 3 | 1574 |
| 1 | 37 | 264 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 304 |
| 2 | 6 | 0 | 213 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 225 |
| 3 | 2 | 0 | 4 | 134 | 2 | 0 | 1 | 1 | 0 | 0 | 144 |
| 4 | 4 | 0 | 6 | 0 | 123 | 0 | 0 | 0 | 0 | 0 | 133 |
| 5 | 29 | 0 | 0 | 0 | 0 | 233 | 0 | 1 | 0 | 0 | 263 |
| 6 | 0 | 2 | 0 | 0 | 0 | 0 | 107 | 0 | 0 | 2 | 111 |
| 7 | 4 | 1 | 0 | 3 | 0 | 0 | 0 | 101 | 0 | 0 | 109 |
| 8 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 93 | 0 | 98 |
| 9 | 6 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 64 | 74 |

(a) Confusion matrix on test set of CENPARMI Indian digit database

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 459 | 33 | 1 | 2 | 0 | 3 | 0 | 0 | 2 | 0 | 500 |
| 1 | 12 | 483 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 500 |
| 2 | 0 | 1 | 493 | 2 | 3 | 0 | 1 | 0 | 0 | 0 | 500 |
| 3 | 1 | 0 | 17 | 472 | 8 | 2 | 0 | 0 | 0 | 0 | 500 |
| 4 | 0 | 0 | 3 | 4 | 492 | 0 | 1 | 0 | 0 | 0 | 500 |
| 5 | 8 | 0 | 0 | 0 | 0 | 492 | 0 | 0 | 0 | 0 | 500 |
| 6 | 1 | 2 | 3 | 1 | 3 | 1 | 484 | 0 | 0 | 5 | 500 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 500 | 0 | 0 | 500 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 499 | 0 | 500 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 3 | 492 | 500 |

(b) Confusion matrix on test set of our Farsi database

Figure 91: Confusion matrices on the test set of CENPARMI Indian digit database (a), and on our Farsi isolated digit database (b).

## 11.5 Discussion

Comparison of the results that we obtained for segmentation and recognition of the numeral strings in Farsi with the results that we obtained in Chapter 8 for Numeral strings in English shows that the recognition rates for Farsi script are much lower than the results for English script. This shows that for segmentation and recognition of Farsi numeral strings, our algorithm must be adjusted and we need specialized features in order to segment and recognize Farsi numerals with high precision.

Unfortunately, our results for segmentation and recognition in Farsi are not comparable with any other method in Farsi, because most of the developed methods for Farsi have been mainly tested on small and private databases collected by the researchers, and those databases have

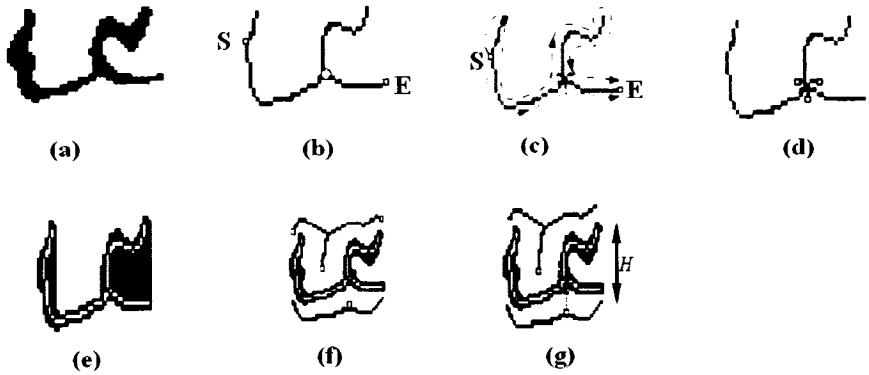



Figure 92: (a) Original image (4 touching 3), (b) Foreground skeleton; starting point and ending point are depicted by S and E, respectively, (c) From starting point (S), skeleton is traversed in two different directions (clockwise: dashed arrows, and counter-clockwise: dotted arrows) to the end point (E), (d) Mapping of intersection points on the outer contour by bisectors to form foreground-features, (e) Background region, (f) Background feature points, (g) Feature points on the background and foreground (from top and bottom) are matched, and assigned together to construct segmentation path(s) .

not always been available to others. We hope that by introducing new developed databases (such as our database) we can provide a standard database to the research community for comparison of algorithms in Farsi OCR.

## 11.6 Summary

This chapter briefly reviewed the evolution of Farsi (Persian) script and its historic connections to Arabic script. It discussed the similarities and dissimilarities between numeral strings in these two scripts from the OCR point of view. Creating a comprehensive database for research on Farsi recognition was described, and our new methods for recognition and segmentation of handwritten numeral strings in Farsi were tested. There are still many unsolved problems in Farsi and Arabic script recognition systems. The main problem is the lack of a standard database with a huge number of samples of words, letters, digits and numeral strings (touching and non touching cases) for research on Farsi recognition. Another important problem is the lack of online resources that show the results of previous

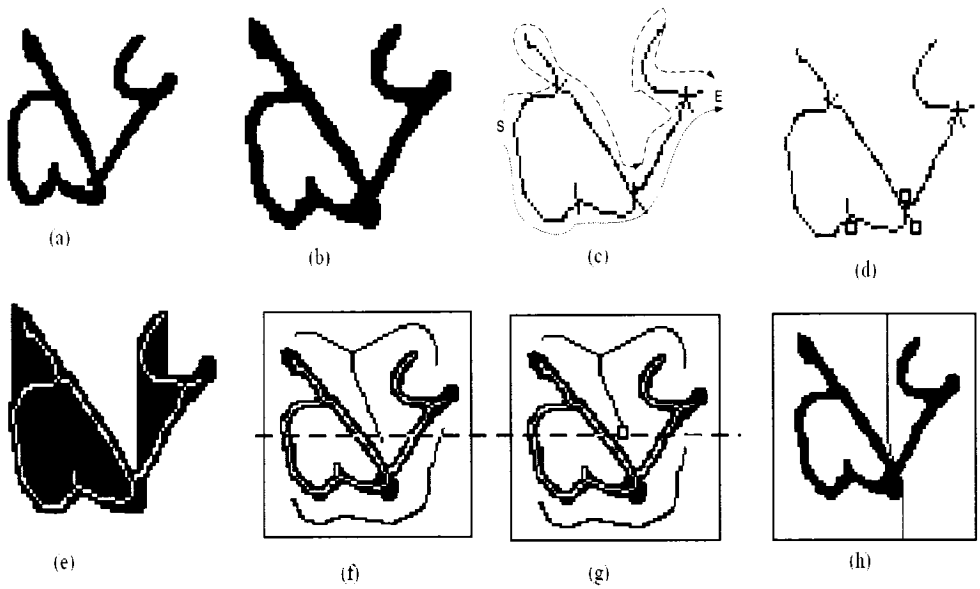
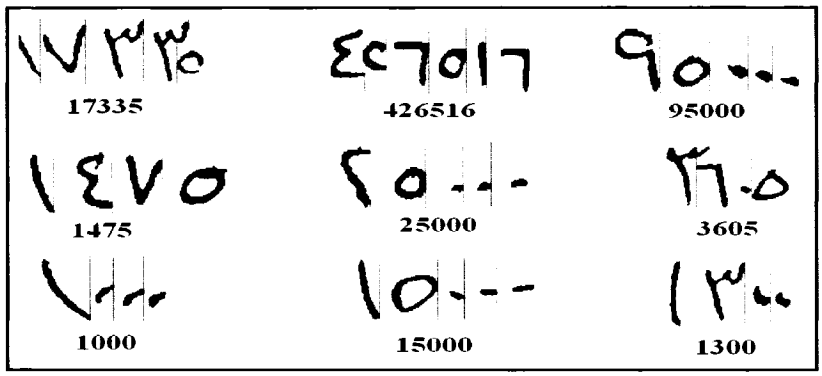
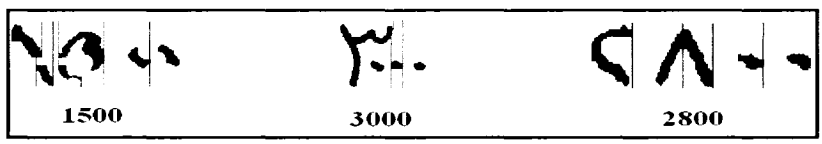



Figure 93: (a) Original image of touching digits 5 and 6, (b) Pre-processed image, (c) skeleton traversals from S to E in clockwise (dashed arrows), and counter-clockwise (dotted arrows), (d) Mapping of intersection points on the outer contour by bisectors to form foreground-features, (e) Background region, (f) Top/bottom background skeletons, (g) Top/bottom-background-skeletons after removing parts that are lower/higher than middle line, (h) Segmentation path for separating two digits.

research works on Farsi recognition to the international research community. We hope that our comprehensive database will become a popular database for Farsi script recognition and will be used for testing and comparison of systems in the field of Farsi handwritten document analysis and recognition research.




(a)




(b)

Figure 94: (a) Samples of numeral strings which our algorithm could find the correct segmentation paths, (b) Samples of numeral strings which our algorithm did not perform well for the over-segmentation of the strings. These numeral strings have been randomly selected from CENPARMI handwritten Arabic check database [7].

# Part III

# Final Conclusion

# Chapter 12

# Conclusion and Future Works

The main focus of this thesis was on the segmentation and recognition of unconstrained handwritten numeral strings. We formulated the segmentation and recognition of unconstrained handwritten numeral strings as an optimization problem, and we proposed novel solutions and algorithms for solving this challenging problem. In order to test our algorithms, we designed and implemented a modular system for the segmentation and recognition of handwritten numeral strings. Our system has several important modules such as: pre-processing, segmentation, feature extraction, classification/evaluation, and an evolutionary algorithm. In each module, we contributed some new algorithms (solutions).

For pre-processing of handwritten numeral strings, we introduced a novel slant correction algorithm. We showed that our algorithm is able to improve extracting of segmentation feature points, and as a result it is able to improve the separation of touching digits. For the first time, we also investigated the statistical characteristics of slant angles in handwritten numeral strings, and we showed some novel statistics. We also showed that slant angles (in handwritten numeral strings) have a near normal distribution.

Constructing segmentation paths is one of the main challenges in the segmentation of handwritten numeral strings. We proposed a novel segmentation method for the segmentation of touching digits and construction of segmentation paths. We introduced a novel algorithm which is called skeleton tracing, for finding feature points on the foreground and background skeletons. In our segmentation algorithm, feature points on the background and foreground

of the numeral strings are extracted, and based on a combination of these features, segmentation paths (segmentation hypotheses) are constructed. Our skeleton tracing algorithm can also be utilized for extracting structural features and exploring skeletons of any 2D objects. In order to handle over and under-segmentation in numerals, we introduced segmentation scores. These scores allow us to efficiently utilize contextual information from the string image. Segmentation scores help us to incorporate information about the shapes, relative sizes, or positions of connected components in the evaluation of the segmentation hypotheses. Our segmentation scores also facilitate and improve the detection of outliers (over/under-segmented components). These scores compensate for low outlier resistance of isolated digit classifiers, and improve their outlier rejection. We showed that by employing segmentation scores (as a source of contextual knowledge), even lower computational cost classifiers can reach a very good performance, comparable to the more complicated classifiers in handwritten numeral string recognition. These scores also help to avoid considerable computation, which is normally required to reject outlier patterns by ordinary digit classifiers.

In order to capture the great variability that exists in the shapes of handwritten digits (especially after their segmentation in numeral strings), we introduced an efficient incremental clustering algorithm that clusters the shapes and the styles of writing of handwritten digits. Our clustering algorithm is an unsupervised incremental learning algorithm that can improve stability and plasticity in handwritten recognition systems. Also, for the first time (to the best of our knowledge), we discussed and considered the problem of stability and plasticity in handwritten recognition systems, and we designed a system that shows stability and plasticity in its recognition. Our clustering algorithm is general and can be used in other similar domains such as data mining or bioinformatics.

We introduced a general framework based on evolutionary algorithms for finding the optimum segmentation and recognition in unconstrained handwritten numeral strings. We introduced a novel genetic representation and operator definition that can be applied for searching the optimal path in any Directed Acyclic Graph (DAG). Also, we proposed a novel hybridization (generalization) of the two popular evolutionary algorithms: Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO). We named this new model as Genetic Based Particle

Swarm Optimization (GBPSO). Our GBPSO model has important elements such as variable size population, birth and death operations, history; and it shows a higher performance and faster convergence compared to the ordinary evolutionary algorithms.

We studied segmentation and recognition of handwritten numeral strings in Farsi and Arabic languages. We also created a comprehensive standard database for research and evaluation of handwritten recognition systems in Farsi (Persian) language. This database is composed of six different sets of handwritten images including: isolated digits, isolated letters, numeral strings, legal amounts, dates, and Arabic isolated digits written by people who speak Farsi. Our Farsi database is publicly available to the research community, and it can function as a very helpful and necessary tool for the investigation and development of Farsi handwritten recognition systems. We also applied our segmentation and recognition algorithms to Farsi handwritten numeral strings. For the first time, we considered the problem of separation of touching digits in Farsi/Arabic handwritten numeral strings, and we provided some promising results on the segmentation of touching digits in Farsi/Arabic.

**Future Works**

Due to time constraints, we did not have the opportunity to address some issues in our research. Here, we outline some directions which we believe are worthy of future investigations:

- We think that it may be possible to improve our segmentation algorithm by utilizing all sources of global and contextual information that can be extracted from numeral string images.
- We think that it may be possible to find a mechanism to estimate the number of digits in the strings, so that this information could be used as another source of knowledge to improve the performance of the system.
- It will be very helpful to find an optimized feature set or a combination of different feature sets in order to improve the classification of segmented numerals.

- It will be very helpful to design a combination of the classifiers in order to improve the recognition accuracy and avoid mis-classifications of segmented or isolated digits. Also, it will be very helpful to use verification strategies for the segmented numerals.

- We think that the problem of segmentation and recognition of handwritten numeral strings can be considered as a multi-objective optimization problem, and in order to solve this problem, we must optimize different (conflicting) objectives for segmentation, recognition, .... Recently, multi-objective optimization models have received great attention in solving machine learning and pattern recognition problems.

- We think that our clustering algorithm can also be modeled as a multi-objective optimization problem in order to optimize the number of clusters, and at the same time to optimize the overall similarity (distance) among the samples, and the shapes (sizes) of the clusters.

- We think that our Genetic Based Particle Swarm Optimization (GBPSO) algorithm may also be improved by using more information from the previous generations (history of the swarm) in order to adjust the birth and mortality rates of future generations. Also, we think that our GBPSO model can be extended for solving multi-objective optimization problems.

- Our investigation on Farsi/Arabic script recognition showed that there are still many challenges and unsolved problems in these two scripts. New sophisticated algorithms for pre-processing, segmentation, and recognition of these scripts must be developed, and the results of these methods must be internationally available in order to avoid repetition in the research. We still think that the main problems are: lack of popular standard databases for development, comparison, and evaluation of handwritten recognition systems, and the lack of on-line resources for research on these two languages.

We hope the efforts in this thesis will improve or speed up the research in these challenging fields.

# Part IV

# Appendices

# Appendix A

## NSTRING SD19 and CENPARMI Databases

In this appendix, we explain the main features of the two main databases (NSTRING SD19 database and CENPARMI isolated digit database) that we have used for our experiments. In the next appendix (B), we will present our Farsi database.

### NSTRING SD19: an Standard Database for Research on Numeral Strings

NSTRING SD19 database is based on NIST Special Database 19 (SD19) (developed by the National Institute of Standards and Technology). This database was originally created to support the PhD. project of Alceu de Souza Britto Jr., who was a Ph.D student (2000) supervised by Prof. Robert Sabourin (ETS-Canada), Prof. Flavio Bortolozzi (PUCPR-Brazil), and co-supervised by Prof. Ching Y. Suen (CENPARMI-Canada), and Prof. Edouard Lethelier (PUCPR-Brazil) [172].
With the numeral strings extracted from the full-page forms available in the Special Database 19 (SD19) provided by NIST, Britto et al. created a numeric string database called NString SD19 useful for research on handwritten numeral string recognition. Such a database is valuable to train, validate and test recognizers that go beyond the isolated digit classification. In this appendix, we first present a brief description of NIST Special Database 19 (SD19) and then we present the structure and content of the NString SD19. A detailed description of the origin, publication history and organization of the NIST SD19 is available in [172].

### NIST Special Database 19 (SD19)

The SD19 is composed of 3669 full-page binary images of Handwritten Sample Forms (HSF), which are organized into eight series, denoted by hsf_0,1,2,3,4,6,7,8. A sample of an HSF

form is shown in Figure 95.

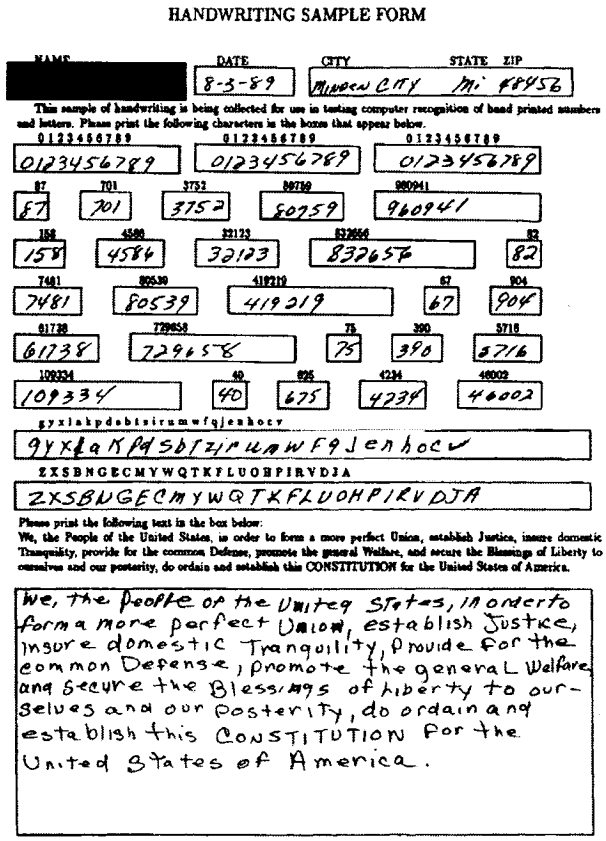



Figure 95: Handwriting Sample Form (HSF) page.

A total of 814,255 handwritten labeled characters (digits and letters) have been segmented from these forms and organized by class field and writer. These isolated characters as well as the full-page images can be found in the original SD19 compact disk. Since the numeral strings were not extracted, Britto et al. decided to fill in this gap [172]. Table 20 shows the distribution of HSF pages per series. From the example shown in Figure 95, we can see that an HSF page consists of 34 fields, 28 of which contain only numeric characters. One can also notice that the information pertaining to each field is given. The field descriptions are presented in Table 21.

Table 20: HSF series distribution

| Series | Number of pages |
|---|---|
| hsf_0 | 500 |
| hsf_1 | 500 |
| hsf_2 | 500 |
| hsf_3 | 600 |
| hsf_4 | 500 |
| hsf_6 | 499 |
| hsf_7 | 500 |
| hsf_8 | 70 |
| Total | 3669 |

Table 21: Handwriting sample forms fields

| Field | Description |
|---|---|
| fld_0 | Name |
| fld_1 | Date |
| fld_2 | City/State/ZIP |
| fld_3 ... fld_30 | Numeric Values |
| fld_31 | Lower case character box |
| fld_32 | Upper case character box |
| fld_33 | Free format text |

A total of 100 HSF templates were used to fill in the HSF pages. The number, size and location of the fields are the same in all template variations, however, they present different character strings.

### NString D19: Extracting Numeral Strings

The process used to extract the numeral strings from the HSF pages is based on two steps: 1) field extraction from the HSF page, 2) pre-processing for bounding box deskewing and removal. In the first step, we detect and extract the field boxes from a page, which are saved separately in new files named fyyyy_xx_w.tif, where yyyy and xx identify the writer and the template, and w corresponds to the field number from 3 to 30. To this end, a process based on vertical and horizontal projections is used to locate the coordinates of the upper left corner of the field fld_0. The coordinates of the other fields are calculated using the fld_0 coordinates as a reference. In order to extract the numeral strings from these

cut fields, the pre-processing first deskews the field bounding box. Afterwards, vertical and horizontal projections are used to locate the lines of the field bounding box in order to remove them. The resulting images, containing only the cleaned numeral string files, are saved as cdfyyyy_xx_w.tif (cd = cleaned and deskewed).

### NString SD19: Organization

The structure of the NString SD19 database is shown in Figure 96. The directory NString SD19 is divided into 11 subdirectories: one for each of the 8 NIST series (hsf_*), a subdirectory for the reference files (truerefs), a subdirectory for the isolated digit database (digits), and a subdirectory for the touching digit pairs (tdp). The HSF series subdirectory is divided into 3 sections as follows:

- pages: contains a copy of the original NIST HSF pages;
- fields: contains the numeric fields cut from each HSF page.
- numeral_strings: contains the numeral strings extracted from the field images.

The string image files are saved in the directory of images. In addition, several groups of file name lists are provided as follows:

*2_digit.list, 3_digit.list, 4_digit.list, 5_digit.list, 6_digit.list, and 10_digit.list natseg.list, touch.list, fragm.list, noise.list, other.list.*

The digits also directory is divided into 3 sections, which contains handwritten isolated digits extracted from hsf_0123, hsf_7 and hsf_4. Some samples of numeral strings from NString SD19 database are shown in Figure 97. More details about the extraction, pre-processing operations, and organization of this database can be found in [172].

### CENPARMI: an Standard Database for Research on Handwritten Isolated Digits

The CENPARMI handwritten isolated digit database has 4000 training samples (400 samples per digit) and 2000 testing samples (200 samples per digit). Compared to other similar

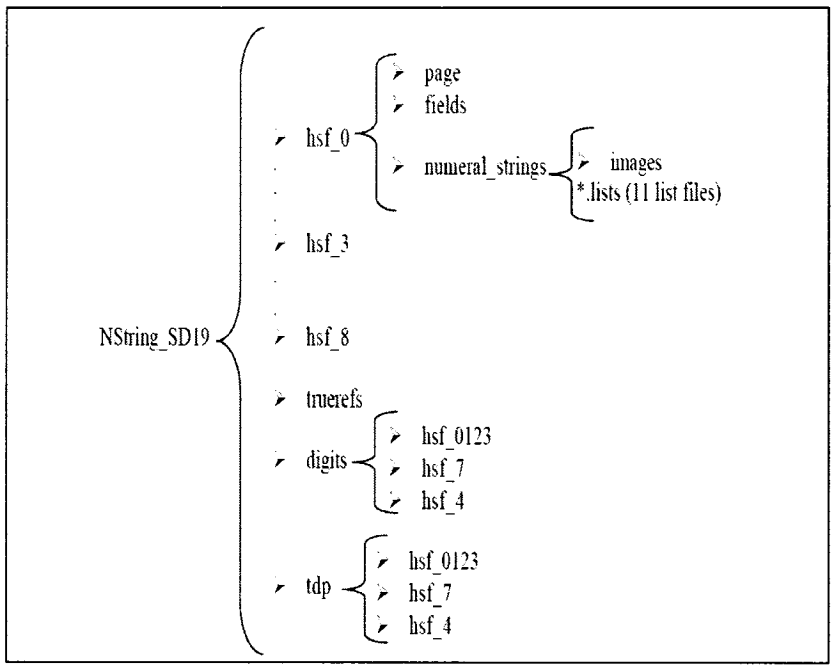



Figure 96: Nstring SD19 structure (note: hsf stands for handwritten sample form, tdp stands for touching digit pairs).

handwritten databases, samples in the CENPARMI database show more variations in their shapes and styles of writings. Figure 98 shows samples of digits in this database.

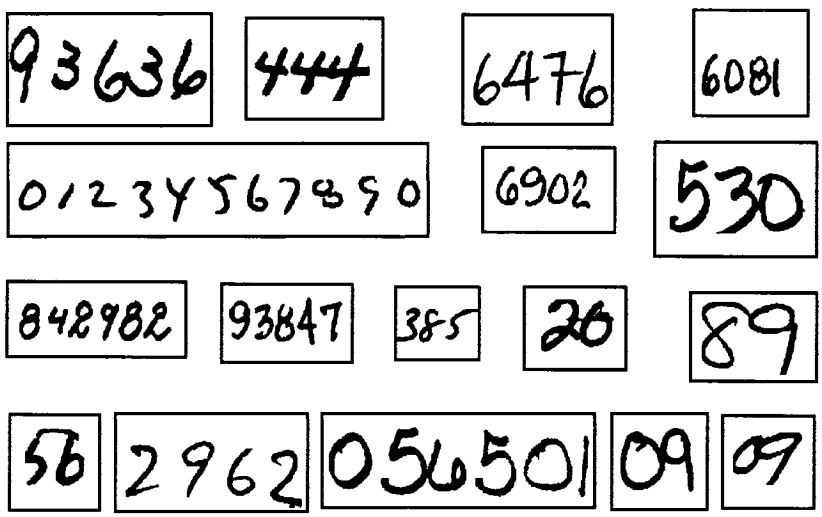


Figure 97: Some samples of numeral strings in NString SD19 database.

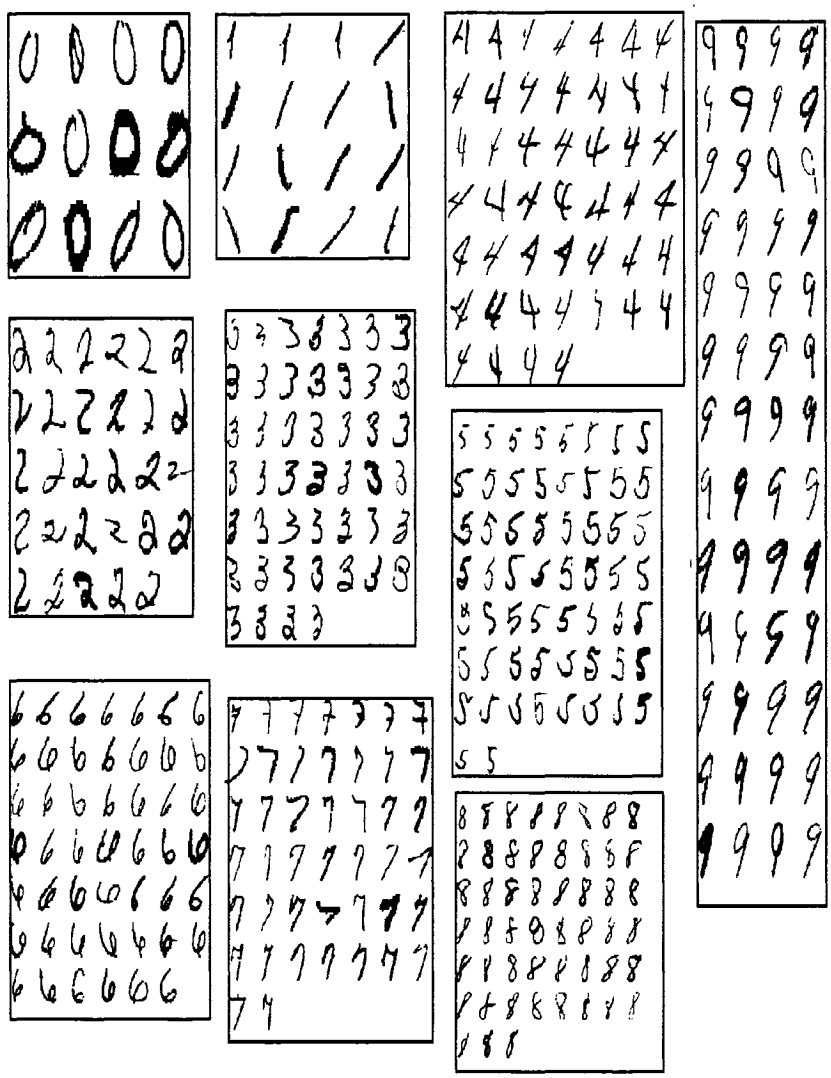


Figure 98: Samples of CENPARMI handwritten isolated digit database that shows the variations of digits in this database.

# Appendix B:

## Our Farsi (Persian) Database

Our comprehensive Farsi (Persian) database includes a set of handwritten datasets that can be used for off-line handwritten recognition research in Farsi [21, 22]. Our database consists of 6 different datasets and these datasets were written by individuals (writers) that were randomly chosen from different: places, groups of ages, genders, and levels of education in Iran. In total, 175 writers (individuals) were chosen to fill out our forms like those shown in Figures 99 and 100. The data in these forms were collected during the years 2005-2006, and then they were processed and organized as a comprehensive database in the Center for Pattern Recognition and Machine Intelligence (CENPARMI) at Concordia University. The writes of the forms were randomly divided into three different sets of 105, 20, and 50 writers. The samples from the first set of writers (105 writers) were taken for training sets of all our datasets. The samples from the second set of writers (20 writers) were taken for verifying sets of all our datasets, and the samples from the third set of writers (50 writers) were taken for testing sets of all our datasets. These datasets include: isolated Farsi digits (18000 samples), isolated Farsi alphabet letters (11900 samples), Farsi numeral strings (7350 samples), Farsi dates (175 samples), Farsi legal amounts words (7875 samples), and Arabic digits (3500 samples of digits used in English/Latin language countries). Our database also provides both gray level and binary versions of all the images in all these datasets. So, it can be used for research on both gray level or binary feature extraction or recognition methods. In this appendix, we show sample forms that were used for collecting handwritten samples, and also some samples from our final Farsi database (in gray level and binary formats). More details about the creation (collecting samples, extraction, pre-processing,...) of our Farsi database can be found in our papers: [21] and [22].

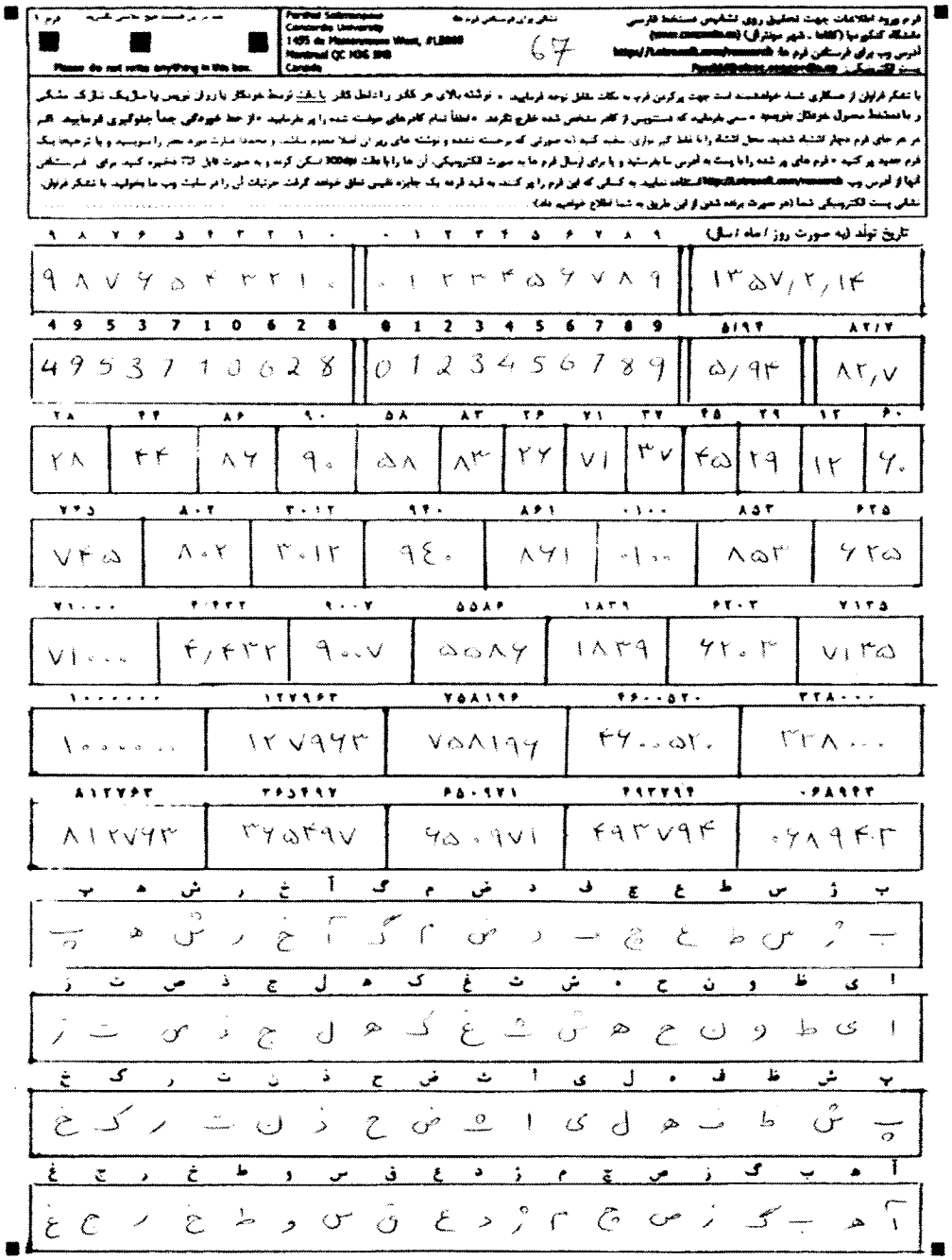


Figure 99: Sample form (From no. 1) that has been used for collecting handwritten data. The fields in this form contain numeral strings, dates, Arabic digits, and Farsi alphabet letters.

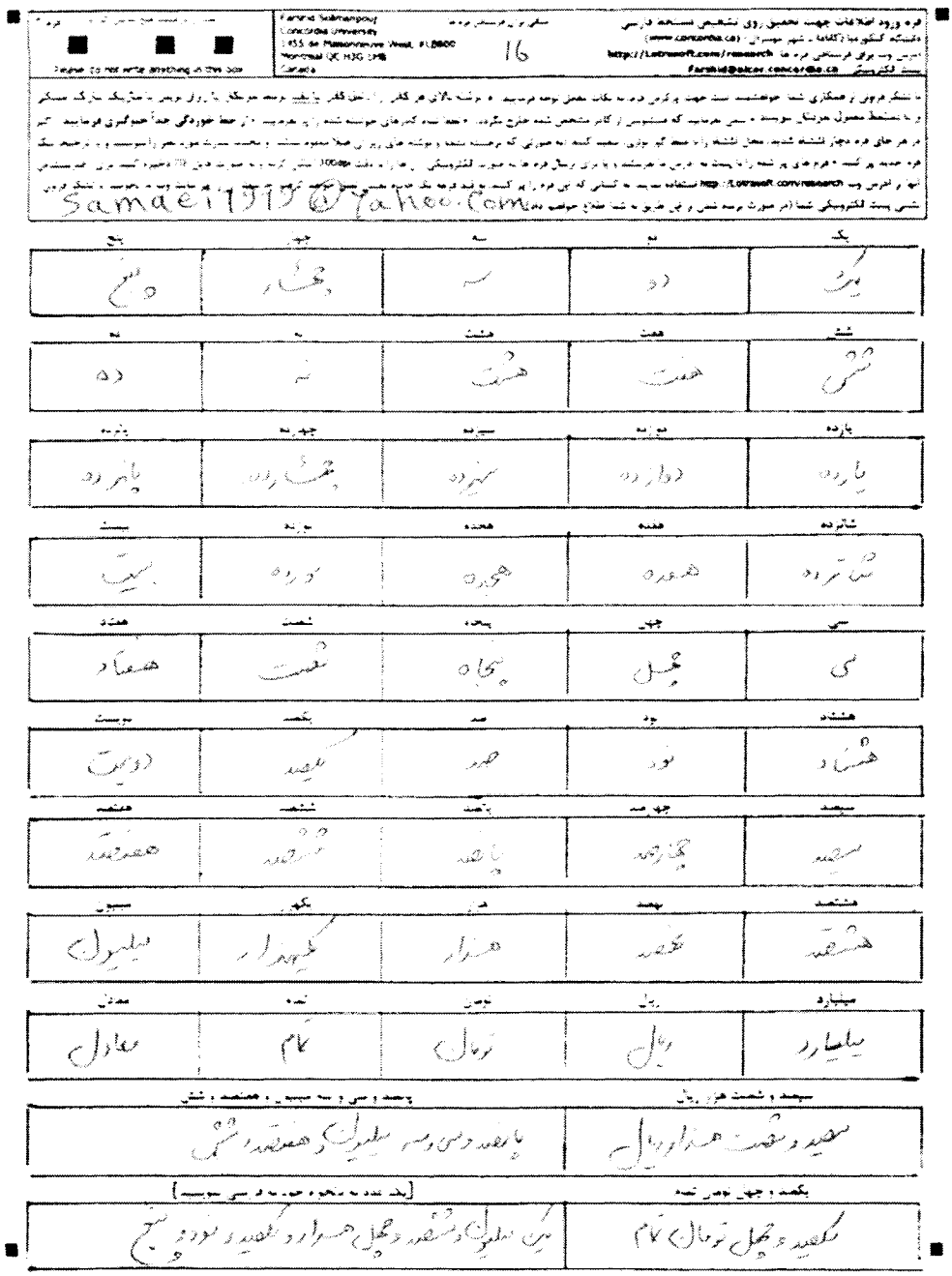


Figure 100: Sample form (From no. 2) that has been used for collecting handwritten data. The fields in this form contain legal amount words, and legal amount values.

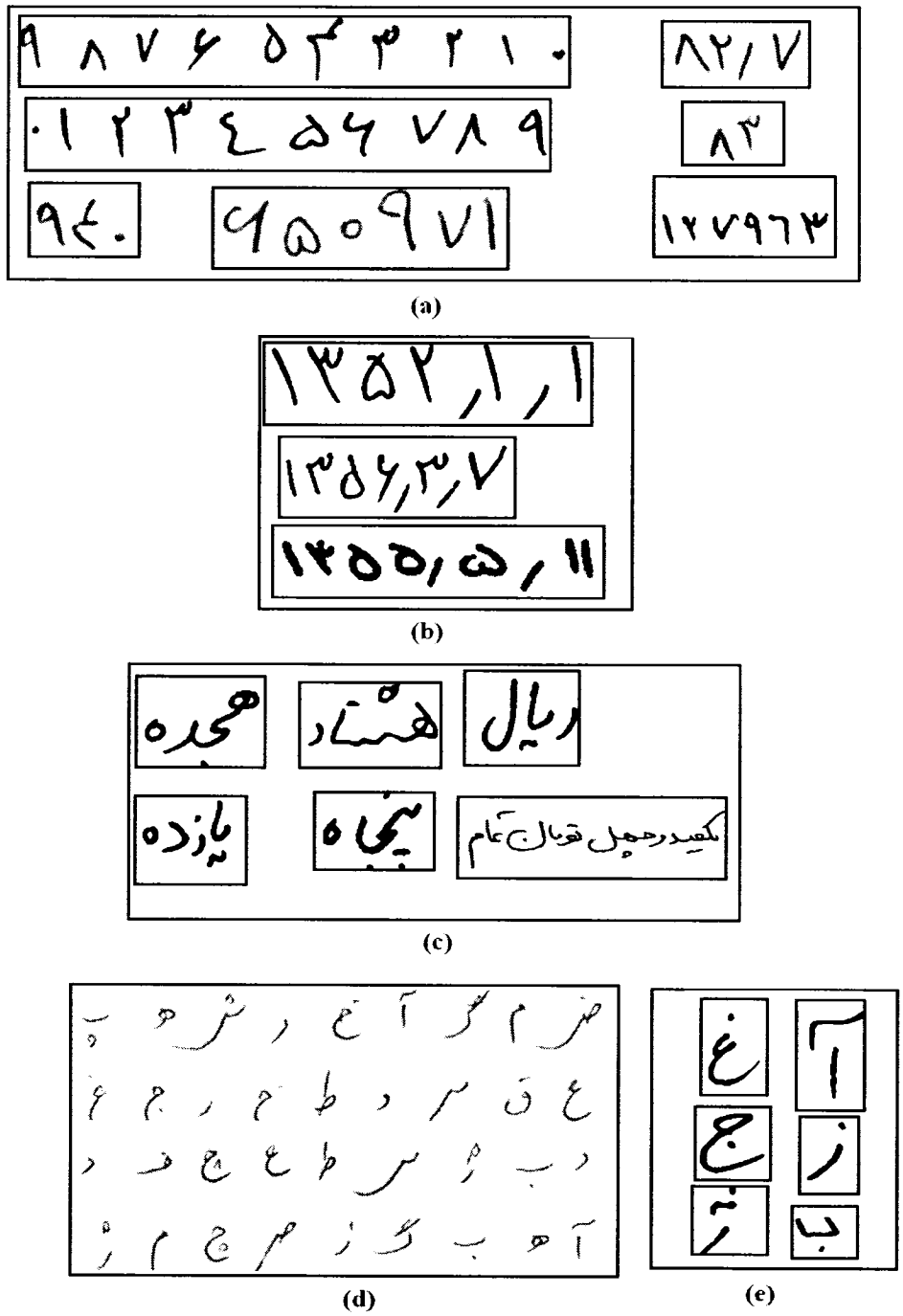



Figure 101: Samples from our Farsi handwritten database, (a) Farsi Numeral strings, (b) Farsi dates, (c) Legal amount words (for Farsi checks), (d) and (e) Farsi letters (gray level and binary formats).

# Appendix C:

## Required Formulas

Here, we present the formulas that are required in order to find $\alpha$, $\beta$ , and $\theta$. These formulas are used for computations of these parameters in Chapter 10, Section 10.4.1.

$$b(t) = \alpha_1 \cos(\omega t + \theta_1) + \beta_1$$

$$m(t) = \alpha_2 \cos(\omega t + \theta_2) + \beta_2$$

$$r(t) = b(t) - m(t)$$

$$r(t) = \alpha \cos(\omega t + \theta) + \beta$$

$$\alpha = \sqrt{(\lambda_1^2 + \lambda_2^2)}$$

$$\theta = \arctan(\frac{\lambda_2}{\lambda_1})$$

$$\beta = \beta_1 - \beta_2$$

Where

$$\lambda_1 = \alpha_1 \cos(\theta_1) - \alpha_2 \cos(\theta_2)$$

$$\lambda_2 = \alpha_1 \sin(\theta_1) - \alpha_2 \sin(\theta_2)$$

# Part V

# Bibliography

# Bibliography


[1] Z. Lu, Z. Chi, W. Siu, and P. Shi. A Background-Thinning-Based Approach, for Separating and Recognizing Connected Handwriting Digit Strings. *Pattern Recognition*, 32(6):921–933, 1999.

[2] Y. K. Chen and J. F. Wang. Segmentation of Single or Multiple Touching Handwritten Numeral String Using Background and Foreground Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1304–1317, 2000.

[3] A. D. S. Britto JR., R. Sabourin, E. Lethelier, F. Bortolozzi, and C. Y. Suen. Improvement in Handwritten Numeral String Recognition by Slant Correction and Contextual Information. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 323–332, Amsterdam, September 2000.

[4] R. Eberhart and J. Kennedy. A New Optimizer Using Particles Swarm Theory. In *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, October 1995.

[5] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, November/December 1995.

[6] K. V. Koumousis and C. P. Katsaras. A Saw-Tooth Genetic Algorithm Combining the Effects of Variable Population Size and Reinitialization to Enhance Performance. *IEEE Transactions on Evolutionary Computation*, 10(1), 2006.

[7] Y. Al-Ohali, M. Cheriet, and C. Y. Suen. Databases for Recognition of Handwritten Arabic Cheques. *Pattern Recognition*, 36(1):111–121, 2003.

[8] C. L. Liu, H. Sako, and H. Fujisawa. Effects of Classifier Structure and Training Regimes on Integrated Segmentation and Recognition of Handwritten Numerals Strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1395–1407, November 2004.

[9] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Automatic Segmentation of Handwritten Numerical Strings: A Recognition and Verification Strategy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1438–1454, November 2002.

[10] U. Pal, A. Belaid, and Ch. Choisy. Touching Numeral Segmentation Using Water Reservoir Concept. *Pattern Recognition Letters*, 24(1-3):261–272, 2003.

[11] J. Punnoose. An Improved Segmentation Module for Identification of Handwritten Numerals. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, October 1999.

[12] X. Wang, V. Govindaraju, and S. Srihari. Holistic Recognition of Touching Digits. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 295–303, Taejon, Korea, August 1998.

[13] N. W. Strathy, C. Y. Suen, and A. Krzyzak. Segmentation of Handwritten Digits Using Contour Features. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 577–580, Tsukuba City, Japan, October 1993.

[14] K. K. Kim, J. H. Kim, and C. Y. Suen. Segmentation-Based Recognition of Handwritten Touching Pairs of Digits Using Structural Features. *Pattern Recognition Letters*, 23(1-3):13–24, 2002.

[15] J. Sadri, C. Y. Suen, and T. D. Bui. Statistical Characteristics of Slant Angles in Handwritten Numeral Strings and their Effects on Segmentation. *Submitted to the International Journal on Document Analysis and Recognition (IJDAR)*, 2006.

[16] J. Sadri, C. Y. Suen, and T. D. Bui. Automatic Segmentation of Unconstrained Handwritten Numeral Strings. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 317–322, Tokyo, Japan, October 2004.

[17] J. Sadri, C. Y. Suen, and T. D. Bui. A Genetic Framework Using Contextual Knowledge for Segmentation and Recognition of Handwritten Numeral Strings. *Pattern Recognition*, 40(3):898–919, 2007.

[18] J. Sadri, C. Y. Suen, and T. D. Bui. New Approach for Segmentation and Recognition of Handwritten Numeral Strings. In *Proceedings of Document Recognition and Retrieval XII (Part of SPIE-IS/T Electronic Imaging 2005)*, volume SPIE-5767, pages 92–100, San Jose, CA, USA, 2005.

[19] J. Sadri, C. Y. Suen, and T. D. Bui. A New Clustering Method for Improving Plasticity and Stability in Handwritten Character Recognition Systems. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 2, pages 1130–1133, Hong Kong, August 2006.

[20] J. Sadri and C. Y. Suen. A Genetic Binary Particle Swarm Optimization Model. In *IEEE Congress on Evolutionary Computation (IEEE CEC 2006)*, pages 656–663, Vancouver, B.C., Canada, July 2006.

[21] F. Solimanpour, J. Sadri, and C. Y. Suen. Standard Databases for Recognition of Handwritten Digits, Numerical Strings, Legal Amounts, Letters and Dates in Farsi Language. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 3–7, La Baule, France, October 2006.

[22] J. Sadri, S. Izadi, F. Solimanpour, C. Y. Suen, and T. D. Bui. State-of-the-Art in Farsi Script Recognition. In *Press, Proceedings of the International Symposium on Signal Processing and its Applications (ISSPA 2007)*, Sharjah, U.A.E., February 2007.

[23] J. Sadri, C. Y. Suen, and T. D. Bui. Application of support vector machines for recognition of handwritten arabic/persian digits. In *Proceedings of Iranian Conference on Machine Vision and Image Processing & Applications (MVIP)*, volume 1, pages 300–307, Tehran-Iran, February 2003.

[24] J. Sadri, C. Y. Suen, and T. D. Bui. Segmentation of Handwritten Numeral Strings in Farsi and English Languages. In *Proceedings of Iranian Conference on Machine Vision and Image Processing & Applications (MVIP)*, volume 1, pages 305–311, Tehran-Iran, February 2005.

[25] C. Y. Suen, S. Izadi, J. Sadri, and F. Solimanpour. Farsi Script Recognition-A Survey. In *Proceedings of International Summit on Arabic and Chinese Handwriting (SACH)*, pages 101–110, University of Maryland, College Park, MD, USA, September 2006.

[26] H. Fujisawa, Y. Nakano, and K. Michino. Segmentation Methods for Character Recognition: From Segmentation to Document Structure Analysis. *Proceedings of the IEEE*, 80(7):1079–1092, 1992.

[27] R. Fenrich. Segmentation of Automatically Located Handwritten Words. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 33–44, Chateau de Bonas, France, September 1991.

[28] D. Yu and H. Yan. Separation of Single-Touching Handwritten Numeral Strings Based on Structural Features. *Pattern Recognition*, 31(12):1835–1847, December 1998.

[29] D. Yu and H. Yan. Separation of Touching Handwritten Multi-Numeral Strings Based on Morphological Structural Features. *Pattern Recognition*, 34(3):587–599, March 2001.

[30] G. Congedo, G. Dimaura, S. Impedovo, and G. Pirlo. Segmentation of Numeric String. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 1038–1041, Montreal, Canada, August 1995.

[31] M. Shridhar and A. Badrelin. Recognition of Isolated and Simply Connected Handwritten Numerals. *Pattern Recognition*, 19(1):1–12, 1986.

[32] M. Blumenstein and S. Verma. A Neural Based Segmentation and Recognition Techniques for Handwritten Words. In *IEEE International Conference on Neural Networks*, volume 3, pages 1738–1742, Anchorage, Alaska, USA, 1998.

[33] G. Martin. Centered-Object Integrated Segmentation and Recognition of Overlapping Hand Printed Characters. *Neural Computation*. 5(3):419–429, 1993.

[34] G. Martin, R. Mosfeq, and J. Pittman. Integrated Segmentation and Recognition Through Exhaustive Scan or Learned Saccadic Jumps. *Pattern Recognition and Artificial Intelligence*, 7(4):831–847, 1993.

[35] L. S. Oliveira, E. Lethelier, F. Bortolozzi, and R. Sabourin. A New Approach to Segment Handwritten Digits. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 577–582, Amesterdam, September 2000.

[36] Z. Shi and V. Govindaraju. Segmentation and Recognition of Connected Handwritten Numeral Strings. In *Progress in Handwritten Recognition*, pages 515–518. World Scientific, Hackensack, NJ, USA, 1996.

[37] H. Nishida and S. Mori. A Model-Based Split-and-Merge Method for Character String Recognition. In P. S. P. Wang, editor, *Document Image Analysis*, pages 209–226. World Scientific, Hackensack, NJ, USA, 1994.

[38] O. Matan and C. J. C. Burges. Recognition of Overlapping Hand-Printed Characters by Centered-Objects Integrated Segmentation and Recognition. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, pages 504–511, Seattle, USA, 1991.

[39] S. Choi and I. Oh. A Segmentation-free Recognition of Two Touching Numerals Using Neural Networks. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 253–256, Bangalore, India, September 1999.

[40] X. Ye, M. Cheriet, and C. Y. Suen. Strcombo: Combination of String Recognizers. *Pattern Recognition Letters*, 23:381–394, 2002.

[41] R. G. Casey and E. Lecolinet. A Survey of Methods and Strategies in Character Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, July 1996.

[42] O. D. Trier, A. K. Jain, and T. Taxt. Feature Extraction Methods for Character Recognition: A Survey. *Pattern Recognition*, 29(4):641–662, 1996.

[43] C. L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten Digit Recognition: Benchmarking of State-of-the-Art Techniques. *Pattern Recognition*, 36(10):2271–2285, 2003.

[44] J. Hu and Y. Yan. Structural Primitive Extraction and Coding for Handwritten Numeral Recognition. *Pattern Recognition*, 31:493–509, 1998.

[45] S. W. Lee. Multilayer Cluster Neural Network for Totally Unconstrained Handwritten Numeral Recognition. *Neural Networks*, 8:783–792, 1995.

[46] H. Nishida. Curve Description Based on Directional Features and Quasiconvexity/Concavity. *Pattern Recognition*, 28(7):1045–1051, 1995.

[47] J. J. Zou and H. Yan. Extracting Strokes from Static Line Images Based on Selective Searching. *Pattern Recognition*, 32(6):935–946, 1999.

[48] C. Y. Suen, C. Nadal, R. Legault, T. A. Mai, and L. Lam. Computer Recognition of Unconstrained Handwritten Numerals. *Proceedings of IEEE*, 80:1162–1180, 1992.

[49] I.S. Oh and C. Y. Suen. Distance Features for Neural Network-Bbased Recognition of Handwritten Characters. *International Journal on Document Analysis and Recognition (IJDAR)*, 1(2):73–88, 1998.

[50] L. E. S. Oliveira. *Automatic Recognition of Handwritten Numerical Strings.* PhD thesis, Ecole de Tecnologie Superieure Universite du Quebec, Montreal, Canada, July 2003.

[51] T. Hirano, Y. Okada, and F. Yoda. Structural Character Recognition Using Simulated Annealing. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 507–510, Ulm, Germany, August 1997.

[52] R. R. Bailey and M. Srinath. Orthogonal Moment Features for Use with Parametric and Non-Parametric Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):389–399, 1996.

[53] M. K. Hu. Visual Pattern Recognition by Moment Invariant. *IEEE Transaction on Information Theory*, 8(2):179–187, 1962.

[54] M. Shridhar and A. Badreldin. High Accuracy Character Recognition Algorithm Using Fourier and Topological Descriptors. *Pattern Recognition*, 17(5):515–524, 1984.

[55] S. Pittner and S. V. Kamarthi. Feature Extraction from Wavelet Coeficients for Pattern Recognition Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(21):83–88, 1999.

[56] S. E. N. Correia and J. M. Carvalho. Optimizing the Recognition Rates of Unconstrained Handwritten Numerals Using Biorthogonal Spline Wavelets. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 2, pages 251–254, Barcelona, Spain, September 2000.

[57] T. M. Bruel. A System for the On-Line Recognition of Handwritten Text. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 2, pages 129–133, Jerusalem, October 1994.

[58] L. Heutte, T. Paquet, J. V. Moreau, Y. Lecourtier, and C. Olivier. A Structural/Statistical Feature Based Vector for Handwritten Character Recognition. *Pattern Recognition Letters*, 19(7):629–641, 1998.

[59] R. Plamondon and S. N. Srihari. On-line and Off-line Handwriting Recognition: Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000.

[60] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.

[61] T. Hastie and P. Y. Simard. Metrics and Models for Handwritten Character Recognition. *Statistical Science*, 13(1):54–65, 1998.

[62] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge, UK, 2000.

[63] B. T. Mittchell and A. M. Gillies. A Model-Based Computer Vision System for Recognizing Handwritten ZIP Codes. *Machine Vision and Applications*, 2(4):231–243, 1989.

[64] M. Shi, Y. Fujisawa, T. Wakbayashi, and F. Kimura. Handwritten Numeral Recognition Using Gradient and Curvature of Gray Scale Image. *Pattern Recognition*, 35(10):2051–2059, 2002.

[65] A. Krzyzak, W. Dai, and C. Y. Suen. Unconstrained Handwritten Character Recognition Using Modified Backpropagation Model. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 155–166, Montreal, Canada, April 1990.

[66] G. Y. Chen, T. D. Bui, and A. Krzyzak. Contour-Based Handwritten Numeral Recognition Using Multiwavelets and Neural Networks. *Pattern Recognition*, 36(7):1597–1604, 2003.

[67] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley and Sons, Inc., Wiley-Interscience, New York, NY, USA, 2000.

[68] A. F. R. Rahman and M. C. Fairhurst. An Evaluation of Multi-Expert Configuration for the Recognition of Handwritten Numerals. *Pattern Recognition*, 31(9):1255–1273, 1998.

[69] J. Kittler, M. Hatef, R. Duin, and J. Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

[70] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[71] D. Guillevic and C. Y. Suen. Cursive Script Recognition Applied to the Processing of Bank Cheques. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 11–14, Montreal, Canada, August 1995.

[72] L. Mic and J. Oncina. Comparison of Fast Nearest Neighbour Classifier for Handwritten Character Recogniton. *Pattern Recognition Letters*, 19(3-4):351–356, 1999.

[73] K. W. Cheung, D. Y. Yeung, and R. T. Chin. A Bayesian Framework for Deformable Pattern Recognition with Application to Handwritten Character Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1382–1388, 1998.

[74] J. Schurmann. *Pattern Classification - A Unified View of Statistical and Neural Approaches.* JohnWiley and Sons Inc., Wiley Interscience, New York, NY, USA, 1996.

[75] Wikipedia, The Free Online Encyclopedia, http://en.wikipedia.org/wiki/, 2006.

[76] A. S. Britto, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Recognition of Handwritten Numeral Strings Using a Two-Stage HMM-Based Method. *International Journal on Document Analysis and Recognition (IJDAR)*, 5(2-3):102–117, 2003.

[77] A. Britto-Jr., R. Sabourin, F. Bortolozzi, and C. Y. Suen. A String Lenght Predictor to Control the Level Building of HMMs for Handwritten Numeral Recognition. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 4, pages 31–34, Quebec City, Canada, August 2002.

[78] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Application in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[79] U. V. Marti and H. Bunke. *Using a Statistical Language Model to Improve the Performance of an HMM-Based Cursive Handwriting Recognition Systems.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2001.

[80] B. Zhang, M. Fu, H. Yan, and M. A. Jabri. Handwritten Digit Recognition by Adaptive-Subspace Self-Organizing Map (ASSOM). *IEEE Transactions on Neural Networks*, 10(4):939–945, 1999.

[81] Z. Chi, Q. Wang, and W. C. Siu. Hierarchical Content Classification and Script Determination for Automatic Document Image Processing. *Pattern Recognition*, 36(11):2483–2500, November 2003.

[82] Y. LeCun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. *Handwritten Digit Recognition with a Back-Propagation Network.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[83] A. Amin, H. B. Al-Sadoun, and S. Fischer. Hand-Printed Arabic Character Recognition System Using an Artificial Network. *Pattern Recognition*, 29(4):663–675, 1996.

[84] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 2003.

[85] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard W. Hubbard, and L. D. Jacket. Backpropagation Applied to Handwritten ZIP Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[86] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, USA, 1995.

[87] L. N. Teow and J.F. Loe. Robust Vision-Based Features and Classification Schemes for Off-line Handwritten Digit Recognition. *Pattern Recognition*, 35(11):2355–2364, 2002.

[88] H. Byun and S. W. Lee. Applications of Support Vector Machines for Pattern Recognition. In *Proceedings of the International Workshop on Pattern Recognition with Support Vector Machine*, pages 213–236, Niagara Falls, Canada, August 2002.

[89] N. E. Ayat, M. Cheriet, and C. Y. Suen. Optimization of the SVM Kernels Using an Empirical Error Minimization Scheme. In *Proceedings of the International Workshop*

*on Pattern Recognition with Support Vector Machine*, pages 354–369, Niagara Falls, Canada, August 2002.

[90] J. Dong, A. Krzyzak, and C. Y. Suen. Fast SVM Training Algorithm With Decomposition on Very Large Training Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):603–618, 2005.

[91] P. D. Gader, J. M. Keller, and J. Cai. A Fuzzy Logic System for Detection and Recognition of Street Number Fields on Handwritten Postal Addresses. *IEEE Transactions on Fuzzy Systems*, 3(1):83–95, 1995.

[92] P. D. Gader, J. M. Keller, R. Krishnapuram, J. H. Chiang, and M. A. Mohamed. Neural and Fuzzy Methods in Handwriting Recognition. *Computer*, 30(2):79–86, February 1997.

[93] B. Lazzerini and F. Marcelloni. A Linguistic Fuzzy Recognizer of Off-line Handwrriten Characters. *Pattern Recognition Letters*, 21(4):319–327, 2000.

[94] C. Y. Suen, R. Legault, C. Nadal, M. Cheriet, and L. Lam. Building a New Generation of Handwriting Recognition Systems. *Pattern Recognition Letters*, 14(4):303–315, 1993.

[95] A. Krogh and J. Vedelsby. Neural Networks Ensembles, Cross Validation, and Active Learning. In G.Tesauro et al., editor, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, Cambridge, MA, USA, 1995.

[96] L. Hansen and O. Salomon. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[97] L. Xu, A. Krzyzak, and C. Y. Suen. Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):418–435, 1992.

[98] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptative Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991.

[99] P. D. Gader, M. A. Mohamed, and J. M. Keller. Fusion of Handwritten Word Classifiers. *Pattern Recognition Letters*, 17(6):577–584, 1996.

[100] R. P. W. Duin. The Combining Classifier: To Train or Not to Train? In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 2, pages 765–770, Quebec City, Canada, August 2002.

[101] Y. S. Huang and C. Y. Suen. An Optimal Method of Combining Multiple Classifiers for Unconstrained Handwritten Numeral Recognition. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 11–20, Buffalo, NY, USA, May 1993.

[102] Y. S. Huang and C. Y. Suen. A Method of Combining Experts for the Recognition of Unconstrained Handwritten Numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94, 1995.

[103] D. Decoste and B. Scholkopf. Training Invariant Support Vector Machines. *Machine Learning*, 46(1-3):160–190, 2002.

[104] P. Zhang. *Reliable Recognition of Handwritten Digits Using A Cascade Ensemble Classifier System and Hybrid Features.* PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, April 2006.

[105] J. Zhou, A. Krzyzak, and C. Y. Suen. Verification-A Method of Enhancing the Recognizers of Isolated and Touching Handwritten Numerals. *Pattern Recognition*, 35(5):1179–1189, 2002.

[106] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Impacts of Verification on A Numeral String Recognition System. *Pattern Recognition Letters*, 24(7):1023–1031, July 2003.

[107] E. Lecolinet and O. Baret. Cursive Word Recognition: Methods and Strategies. In *Fundamentals in Handwriting Recognition*, pages 235–263. Springer-Verlag Inc., New York, NY, USA, 1994.

[108] S. Madhvanath and V. Govindaraju. The Role of Holistic Paradigms in Handwritten Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):149–164, 2001.

[109] T. K. Ho, J. J. Hull, and S. N. Srihari. A Word Shape Analysis Approach to Lexicon Based Word Recognition. *Pattern Recognition Letters*, 13:821–826, 1992.

[110] J. Zhou and C. Y. Suen. Unconstrained Numeral Pair Recognition Using Enhanced Error Correcting Output Coding: A Holistic Approach. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 484–488, Seoul, Korea, August-September 2005.

[111] J. Zhou, Q. Gan, A. Krzyzak, and C. Y. Suen. Recognition of Hhandwritten Numerals by Quantun Neural Networks with Fuzzy Features. *International Journal on Document Analysis and Recognition (IJDAR)*, 2(1):30–36, 1999.

[112] J. Park, V. Govindaraju, and S. N. Srihari. OCR in a Hierarchical Feature Space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4), 1998.

[113] T. M. Ha and H. Bunke. Off-line Handwritten Numeral Recognition by Perturbation Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):535–539, 1997.

[114] C. L. Liu and M. Nakagawa. Handwritten Numeral Rrecognition Using Neural Networks: Iimproving the Accuracy by Discriminative Training. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 257–260, Bangalore, India, September 1999.

[115] G. Mayraz and G. E. Hinton. Recognizing Handwritten Digits Uusing Hierarchical Pproducts of Experts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):189–197, 2002.

[116] F. Lauer, C. Y. Suen, and Gerard Bloch. A Trainable Feature Extractor for Handwritten Digit Recognition. *Pattern Recognition*, 40:1816–1824, 2007.

[117] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best Practices for Convolution Neural Networks Applied to Document Analysis. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 958–962, Edinburgh, Scotland, August 2003.

[118] M. Cheriet, Y. S. Huang, and C. Y. Suen. Background Region Based Algorithm for the Segmentation of Connected Digits. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, pages 619–622, Hague, Netherlands, September 1992.

[119] Z. Shi, S. N. Srihari, Y-C. Shin, and V. Ramanaprasad. A System for Segmentation and Recognition of Totally Unconstrained Handwritten Numeral Strings. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 455–458, Ulm, Germany, August 1997.

[120] R. Alhajj and A. Elnagar. Multiagents to Separating Handwritten Connected Digits. *IEEE Transactions on Man and Cybernetics, Part A*, 35(5):593–602, September 2005.

[121] E. Lethelier, M. Leroux, and M. Gilloux. An Automatic Reading System for Handwritten Nnumeral Amounts on French Checks. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 92–97, Montreal. Canada, August 1995.

[122] C. Y. Suen, K. Liu, and N. W. Strathy. Sorting and Recognizing Cheques and Financial Documents. In *Proceedings of the 3rd IAPR Workshop on Document Analysis Systems*, pages 1–18, Nagano, Japan, November 1998.

[123] G. Kaufmann and H. Bunke. Automated Reading of Cheque Amounts. *Pattern Analysis and Applications*, 3(2):132–141, 2000.

[124] D. E. Goldberg. *Genetic Algorithms in Search, Optimization , and Machine Learning.* Addison-Wesley Longman Inc., Boston, MA, USA, 1989.

[125] J. R. Koza. *Genetic Programming - On The Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, USA, 1992.

[126] P. Slavik and V. Govindaraju. Equivalence of Different Methods for Slant and Skew Correction in Word Recognition Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):323–326, March 2001.

[127] S. N. Srihari, S. H. Cha, H. Arora, and S. Lee. Individuality of Handwriting. *Journal of Forensic Science*, 47(4):1–17, July 2002.

[128] R. M. Bozinovic and S. N. Srihari. Off-Line Cursive Script Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68–83, January 1989.

[129] S. Uchida, E. Taira, and H. Sakoe. Nonuniform Slant Correction Using Dynamic Programming. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 434–438, Seattle, USA, September 2001.

[130] T. Yamaguchi, Y. Nakano, M. Maruyama, H. Miyao, and T. Hananoi. Digit Classification on Signboards for Telephone Number Recognition. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 359–363, Edinburgh, Scotland, August 2003.

[131] Y. Ding, F. Kimura, Y. Miyake, and M. Shridhar. Accuracy Improvement of Slant Estimation for Handwritten Words. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 4, pages 527–530, Barcelona, Spain, September 2000.

[132] G. Kim and V. Govindaraju. A Lexicon Driven Approach to Handwritten Word Recognition for Real-Time Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):366–379, April 1997.

[133] D. Guillevic and C. Y. Suen. Cursive Script Recognition: A Sentence Level Recognition Scheme. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 216–223, Taipei, Taiwan, December 1994.

[134] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. A Slant Removal Algorithm. *Pattern Recognition*, 33(7):1261–1262, July 2000.

[135] F. Kimura, M. Shridhar, and Z. Chen. Improvements of a Lexicon Directed Algorithm for Recognition of Unconstrained Handwritten Words. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 18–22, Tsukuba City, Japan, October 1993.

[136] H. S. M. Coxeter and S. L. Greitzer. *Geometry Revisited.* Mathematical Association of America, Washington DC, 1967.

[137] S. Shapiro. *How to Test Normality and Other Distribution Assumptions.* WI- American Society for Quality, Milwaukee, 1990.

[138] M. Hart and R. Hart. *Statistical Process Control for Health Care.* Duxbury Press, Pacific Grove, CA, USA, 2002.

[139] R. C. Gonzales and P. Wintz. *Digital Image Processing.* Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2001.

[140] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press, Rockville, MD, USA, 1982.

[141] A. G. Ghuneim. Contour Tracing. Technical report, Department of Computer Science, McGill University, Montreal, Canada, http://www.cs.mcgill.ca/aghnei/index.html, 2000.

[142] T. Y. Zhang and C. Y. Suen. A Fast Parallel Algorithm for Thinning Digital Patterns. *Communication ACM*, 27(3):236–239, 1984.

[143] T. Pavlidis. A Thining Algorithm for Discrite Binary Images. *Computer Graphics and Image Processing*, 13:142–157, 1980.

[144] L. Huang, G. Wan, and C. Liu. An Improved Parallel Thinning Algorithm. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 780–783, Edinburgh, Scotland, August 2003.

[145] J. T. Favata. Offline General Handwritten Word Recognition Using an Approximate BEAM Matching Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):1009–1021, 2001.

[146] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA ; McGraw-Hill, New York, NY, 2001.

[147] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Nashua, NH, USA, 2nd edition, 2000.

[148] M. Y. Chen, A. Kundu, and J. Zhou. Off-line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):481–496, 1994.

[149] J. H. Kim, K. K. Kim, C. P. Nadal, and C. Y. Suen. A Methodology of Combining HMM and MLP Classifiers for Cursive Word Recognition. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 2, pages 319–322, Barcelona, Spain, September 2000.

[150] L. S. Oliveira and R. Sabourin. Support Vector Machines for Handwritten Numerical String Recognition. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 39–44, Tokyo, Japan, October 2004.

[151] C. L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten Digit Recognition: Investigation of Normalization and Feature Extraction Techniques. *Pattern Recognition*, 37:265–279, 2004.

[152] G. Wahba, X. Lin, F. Gao, D. Xiang, R. Klein, and B. Klein. The Bias-Variance Trade-Off and the Randomized GACV. In *Proceedings of the 13th Conference on Neural Information Processing Systems (NIPS)*, pages 8–31, Vancouver, Canada, December 2001.

[153] Z. Chi, M. Suters, and H. Yan. Separation of Single and Double Touching Handwritten Numeral Strings. *Optical Engineering*, 34:1159–1165, 1995.

[154] C. L. Liu, H. Sako, and H. Fujisawa. Integrated Segmentation and Recognition of Handwritten Numerals: Comparison of Classification Algorithms. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 303–308, Niagara-on-the-Lake, Canada, August 2002.

[155] Y. Prudent and A. Ennaji. A K Nearest Classifier Design. *Electronic Letters on Computer Vision and Image Analysis*, 5(2):58–71, 2005.

[156] L. P. Cordella, C. De Stefano, A. Della Cioppa, and A. Marcelli. A New Evolutionary Learning Model for Handwritten Character Prototyping. In *Proceedings of International Conference on Image Analysis and Processing (ICIAP)*, pages 830–835, Venice, Italy, September 1999.

[157] G. A. Carpenter and S. Grossberg. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *IEEE Computer*, 21(3):77–88, March 1988.

[158] Y. Prudent and A. Ennaji. A New Learning Algorithm for Incrementalself-Organizing Maps. In *Proceedings of European Symposium on Artificial Neural Networks (ESANN)*, pages 7–12, Bruges, Belgium, April 2005.

[159] D. Rogers and T. Tanimoto. A Computer Program for Classifying Plants. *Science*, 3434(132):1115–1118, 1960.

[160] R. Xu and D. Wunsch II. Survey of Clustering Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):645–678, May 2005.

[161] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 2001.

[162] J. Lee, S. Lee, S. Chang, and B.-H. Ahn. A Comparison of GA and PSO for Excess Return Evaluation in Stock Markets. volume 3562, pages 221–230. Lecture Notes in Computer Science (LNCS), Springer Berlin/Heidelberg, 2005.

[163] A. A. A. Esmin, G. Lambert-Torres, and G. B. Alvarenga. Hybrid Evolutionary Algorithm Based on PSO and GA Mutation. page 57, Aukland, New Zealand, December 2006.

[164] R. Poli, W. B. Langdon, and O. Holland. Extending Particle Swarm Optimisation via Genetic Programming. In *European Conference on Genetic Programming (EuroGP)*, pages 291–300, Lausanne, Switzerland, March-April 2005.

[165] J. Kennedy and R. Eberhart. A Discrete Binary Version of the Particle Swarm Optimization Algorithm. In *Proceedings of IEEE International Conference On Systems, Man and Cybernetics (SMC)*, pages 4104–4109, Orlando, Florida, USA, October 1997.

[166] J. Roughgarden. *Theory of Population Genetics and Evolutionary Ecology: An Introduction.* Reprinted by Macmillan Publishing Company, New York , NY, 1987; and Prentice-Hall, Upper Saddle River, New Jersey, 1996.

[167] J. Song and J. Yu. *Population System Control.* China Academic Bublishers, Beijing, Springer-Verlag, Berlin, 1988.

[168] M. H. Shirali-Shareza, K. Faez, and A. Khotanzad. Recognition of Handwritten Farsi Numerals by Zernike Moments Features and a Set of Class Specific Neural Network Classifiers. In *Proceedings of The International Conference on Signal Processing Applications, and Technology (ICSPAT)*, volume 2, pages 998–1003, Dallas, Texas, USA, October 1994.

[169] M. H. Shirali-Shareza, K. Faez, and A. Khotanzad. Recognition of Handwritten Arabic/Persian Numerals by Shadow Coding and an Edited Probabilistic Neural Network. In *Proceedings of The International Conference on Image Processing (ICIP)*, volume 3, Washington DC, USA, October 1995.

[170] H. Soltanzadeh and M. Rahmati. Recognition of Persian Handwritten Digits Using Image Profiles of Multiple Orientations. *Pattern Recognition Letters*, 25(14):1569–1576, October 2004.

[171] M. Pechwitz and V. Mrgner. Baseline Estimation for Arabic Handwritten Words. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Niagara-on-the-Lake, Canada, August 2002.

[172] A. de S. Britto Jr. and C. P. Nadal. Numeral String Special Database 19 (NString SD19) Version 1.0. Technical report, Ecole de Technologie Superieure (ETS, Universite du Quebec, Montreal, Canada), Centre for Pattern Recognition and Machine Intelligence (CENPARMI, Concordia University, Montreal, Canada), Pontifcia Universidade Catlica do Paran (PUC-PR), Brazil, June 2000.