This item is held in Loughborough University's Institutional Repository (https://dspace.Iboro.ac.uk/) and was harvested from the British Library's EThOS service (http://www.ethos.bl.uk/). It is made available under the following Creative Commons Licence conditions.

Attribution-NonCommercial-NoDerius 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:
BY:
Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes,

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Leqal Code (the full license).
Disclaimer 민

For the full text of this licence, please go to:

# RECOGNITION OF OFF-LINE HANDWRITTEN CURSIVE TEXT 

by

## Ibrahim Sulaiman Ibrahim Abuhaiba

## A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of the degree of

## Doctor of Philosophy

Department of Electronic and Electrical Engineering of the Loughborough University of Technology

[^0]
## ACKNOWLEDGEMENT

I first thank God Almighty for giving me the strength and patience to go through with this research and for the knowledge He has bestowed upon me for fulfilling its requirements.

I would like to express my sincere thanks, appreciation, and special acknowledgements to my supervisors Dr. M. J. J. Holt and Dr. S. Datta for their advise through the entire phases of the research. They were very kind in granting me from their time and knowledge.

Last and not least, I would like to thank my sincere wife, Suhad, for her patience and encouragement.


إلى أل، واثى وزو.ــتي

## PAGE

## MISSING

## IN

ORIGINAL

## CONTENTS

LIST OF ALGORITHMS ..... xi
ABSTRACT ..... xii
1 Introduction ..... 1
1.1. PROBLEM DEFINITION. ..... 1
1.2. GENERAL MOTIVATIONS ..... 2
1.3. CHARACTERISTICS OF ARABIC HANDWRITING ..... 3
1.4. MAIN PROBLEMS TO BE SOLVED IN A TEXT RECOGNITION SYSTEM ..... 6
1.5. STATE OF THE ART OF ARABIC TEXT RECOGNITION ..... 8
1.6. CURRENT OBJECTIVES ..... 13
1.7. SYSTEMS OVERVIEW ..... 18
Part One
Preprocessing
OVERVIEW ..... 24
2 Straight Line Approximation ..... 27
2.1. STRAIGHT LINE APPROXIMATION OF DIGITAL CURVES ..... 28
2.1.1. Definitions ..... 28
2.1.2. Example ..... 32
2.1.3. Causes of Failure. ..... 34
2.2. ENHANCED STRAIGHT LINE APPROXIMATION ..... 36
2.2.1. Definitions ..... 38
2.2.2. Examples. ..... 40
2.2.3. Results and Causes of Failure. ..... 43
2.3. LOOP REDUCTION ..... 47
2.3.1. Definitions ..... 47
2.3.2. Example ..... 48
3 Enforcement of Temporal Information ..... 51
3.1. DETERMINATION OF START VERTEX: HEURISTICS ..... 52
3.1.1. Examples ..... 54
3.1.2. Causes of Failure ..... 57
3.2. DETERMINATION OF END VERTEX: HEURISTICS ..... 58
3.2.1. Examples ..... 59
3.2.2. Causes of Failure ..... 62
3.3. DETERMINATION OF START AND END VERTICES: THEOREMS ..... 63
3.3.1. Definitions ..... 64
3.3.2 $\quad$ Theorem 1 ..... 64
3.3.3. Theorem 2 ..... 65
3.4. ENFORCEMENT OF TEMPORAL INFORMATION ..... 66
3.4.1. Solution of the Chinese Postman's Problem ..... 67
3.4.2. Algorithm to Enforce Temporal Information ..... 70
3.4.3. Example ..... 73
3.4.4 Testing ..... 75
3.4.4.1. Data Acquisition ..... 75
3.4.4.2 Results ..... 77
3.4.5. Causes of Failure ..... 82
Part Two
Isolated Arabic Character Recognition System (IACR)
overview ..... 87
4 Tempo-Structural Representation ..... 93
4.1. . STROKE SEGMENTATION ..... 93
4.2. : INHERENT PROPERTIES OF PRIMITIVE SEQUENCES ..... 95
4.3. PRIMITIVE RELATIONSHIPS ..... 96
4.4. GLOBAL FEATURES OF STROKE ..... 97
4.4.1. Selected Global features ..... 97
4.5. EXAMPLE ..... 98
5 Stroke Recognition ..... 101
5.1. BACKGROUND ..... 101
5.1.1. Toward a Sequential Machine Model of Handwriting. ..... 102
5.1.2. Concepts Based on Fuzzy Set Theory ..... 104
5.1.3. Fuzzy numbers ..... 106
5.2. THE FUZZY SEQUENTIAL MACHINE ..... 108
5.2.1. Definition of Fuzzy Sequential Machine ..... 109
5.2.2. Example ..... 111
5.3. RECOGNITION ALGORITHM. ..... 113
5.3.1. Definitions ..... 113
5.3.2. Example ..... 118
6 Stroke Learning ..... 123
6.1. LEARNING ..... 124
6.2. GENERATION OF FUZZY SEQUENTIAL MACHINES ..... 126
6.2.1. Example ..... 130
6.3. MERGING OF FUZZY SEQUENTIAL MACHINES ..... 135
6.3.1. Definitions ..... 135
6.3.2. Algorithm to Merge Two Fuzzy Sequential Machines ..... 137
6.3.3. Example ..... 139
6.4. MODIFICATION OF A FUZZY SEQUENTIAL MACHINE ..... 144
6.4.1. Example ..... 146
7 Experimentation ..... 149
7.1. CHOOSING THE CHARACTER SET ..... 149
7.2. DATA ACQUISITION ..... 150
7.3. LEARNING ..... 152
7.4. TESTING ..... 152
7.5. REJECTION AND ERROR ANALYSIS ..... 155
Part Three
Cursive Arabic Script Recognition System (CASR)
OVERVIEW ..... 160
8 Stroke Segmentation ..... 169
8.1. ALGORITHM FOR STROKE SEGMENTATION ..... 169
8.1.1. Definitions ..... 170
8.1.2. Example ..... 173
8.2. TOKEN FEATURES ..... 174
9 Token Recognition ..... 178
9.1. TOKEN FUZZY SEQUENTIAL MACHINE ..... 179
9.1.1. Definition of Token Fuzzy Sequential Machine ..... 180
9.1.2. Example ..... 181
9.2. RECOGNITION ALGORITHM. ..... 182
9.2.1. Definitions ..... 182
9.2.2. Algorithm Description ..... 183
9.2.3. Example ..... 185
10 Token Learning ..... 189
10.1. LEARNING ALGORITHM ..... 190
10.2. GENERATION OF FUZZY SEQUENTIAL MACHINES ..... 192
10.2.1. Example ..... 195
10.3. MERGING OF FUZZY SEQUENTIAL MACHINES ..... 199
10.3.1. Example ..... 199
10.4. MODIFICATION OF A FUZZY SEQUENTIAL MACHINE ..... 203
10.4.1. Example ..... 205
11 Learning of Token Strings. ..... 207
11.1. FROM TOKENS TO TOKEN STRINGS ..... 207
11.1.1. Definitions ..... 208
11.1.2. Features of Token Strings ..... 215
11.2. LEARNING OF TOKEN STRINGS ..... 216
11.3. EXAMPLE ..... 218
12 Line Extraction and Stroke Ordering. ..... 223
12.1. BACKGROUND ..... 224
12.2. SEPARATING MAIN AND SECONDARY STROKES ..... 225
12.2.1. Definitions ..... 225
12.2.2. Example ..... 227
12.3. LINE EXTRACTION AND STROKE ORDERING ..... 228
13 Word Formation ..... 238
13.1. CSH INTERPRETATIONS OF MAIN STROKES ..... 240
13.1.1. Definitions ..... 240
13.1.2. Example ..... 244
13.2. CHARACTER FORMATION ..... 248
13.2.1. Definitions ..... 250
13.2.2. Problem Formulation ..... 250
13.2.3. Example ..... 255
13.3. MANIPULATION OF REDUNDANT SECONDARY STROKES ..... 258
13.3.1. Manipulation of Vertical Bar Secondary Stroke ..... 258
13.3.2. Extraction of Isolated $R a$ and Isolated Zain Characters ..... 259
13.3.3. Extraction of Miscellaneous Characters ..... 260
14 Experimentation ..... 262
14.1. DATA ACQUISITION ..... 262
14.2. LEARNING ..... 263
14.3. TESTING ..... 264
14.4. REJECTION ANALYSIS ..... 269
14.5. ERROR ANALYSIS ..... 269
15 Conclusions ..... 274
15.1. PREPROCESSING ..... 274
15.2. IACR SYSTEM ..... 277
15.3. CASR SYSTEM. ..... 278
15.4. HOW THIS WORK CONTRIBUTED TO THE FIELD ..... 280
15.5. SUGGESTIONS FOR FURTHER WORK ..... 282
References ..... 285
Publications from the Thesis ..... 292
Appendices
A Learning Data Set of the IACR System: ..... 293
B Testing Data Set of the IACR System ..... 305
C . Learning Data Set of the CASR System ..... 327
D Testing Data Set of the CASR System ..... 341

## LIST OF ALGORITHMS

| Algorithm | Use to | Page |
| :---: | :---: | :---: |
| 2.1 | obtain a direct straight line approximation of a stroke | 31 |
| 2.2 | obtain an enhanced direct straight line approximation of a stroke | 39 |
| 2.3 | obtain a reduced graph of a stroke | 48 |
| 3.1 | find the start vertex of a stroke | 52 |
| 3.2 | find the end vertex of a stroke | 58 |
| 3.4 | enforce temporal information on an off-line handwritten Arabic stroke | 70 |
| 4.1 | segment a stroke into a sequence of primitives | 94 |
| 5.3 | recognize a stroke | 115 |
| 6.1 | learn a stroke | 125 |
| 6.2 | generate a fuzzy sequential machine from a stroke | 128 |
| 6.3 | merge two fuzzy sequential machines | 137 |
| 6.4 | learn a stroke by modifying a fuzzy sequential machine | 145 |
| 8.1 | segment a cursive stroke into tokens | 172 |
| 9.2 | recognize a token | 183 |
| 10.1 | learn a multi-vertex token | 191 |
| 10.2 | generate a fuzzy sequential machine from a multi-vertex token | 193 |
| 10.4 | learn a token by modifying a fuzzy sequential machine | 204 |
| 11.2 | learn logical token strings in a stroke | 216 |
| 12.2 | find main strokes and secondary stroke candidates | 226 |
| 12.3 | extract lines, order main strokes, and present secondary stroke candidates to main strokes | 235 |
| 13.1 | enumerate all possible common shape interpretations of a main stroke | 240 |
| 13.2 | combine common shapes with secondary strokes to form characters | 254 |

# RECOGNITION OF OFF-LINE HANDWRITTEN CURSIVE TEXT 

 byIbrahim Sulaiman Ibrahim Abuhaiba
Keywords-Handwritten Cursive Text, Off-Line Recognition, Straight Line Approximation, Temporal Information


#### Abstract

The author presents novel algorithms to design unconstrained handwriting recognition systems organized in three parts:

In Part One, novel algorithms are presented for processing of Arabic text prior to recognition. Algorithms are described to convert a thinned image of a stroke to a straight line approximation. Novel heuristic algorithms and novel theorems are presented to determine start and end vertices of an off-line image of a stroke. A straight line approximation of an off-line stroke is converted to a one-dimensional representation by a novel algorithm which aims to recover the original sequence of writing. The resulting ordering of the stroke segments is a suitable preprocessed representation for subsequent handwriting recognition algorithms as it helps to segment the stroke. The algorithm was tested against one data set of isolated handwritten characters and another data set of cursive handwriting, each provided by 20 subjects, and has been $91.9 \%$ and $91.8 \%$ successful for these two data sets, respectively.

In Part Two, an entirely novel fuzzy set-sequential machine character recognition system is presented. Fuzzy sequential machines are defined to work as recognizers of handwritten strokes. An algorithm to obtain a deterministic fuzzy sequential machine from a stroke representation, that is capable of recognizing that stroke and its variants, is presented. An algorithm is developed to merge two fuzzy machines into one machine. The learning algorithm is a combination of many described algorithms. The system was tested against isolated handwritten characters provided by 20 subjects resulting in $95.8 \%$ recognition rate which is encouraging and shows that the system is highly flexible in dealing with shape and size variations.

In Part Three, also an entirely novel text recognition system, capable of recognizing off-line handwritten Arabic cursive text having a high variability is presented. This system is an extension of the above recognition system. Tokens are extracted from a onedimensional representation of a stroke. Fuzzy sequential machines are defined to work as recognizers of tokens. It is shown how to obtain a deterministic fuzzy sequential machine from a token representation that is capable of recognizing that token and its variants. An algorithm for token learning is presented. The tokens of a stroke are re-combined to meaningful strings of tokens. Algorithms to recognize and learn token strings are described. The recognition stage uses algorithms of the learning stage. The process of extracting the best set of basic shapes which represent the best set of token strings that constitute an unknown stroke is described. A method is developed to extract lines from pages of handwritten text, arrange main strokes of extracted lines in the same order as they were written, and present secondary strokes to main strokes. Presented secondary strokes are combined with basic shapes to obtain the final characters by formulating and solving assignment problems for this purpose. Some secondary strokes which remain unassigned are individually manipulated. The system was tested against the handwritings of 20 subjects yielding overall subword and character recognition rates of $55.4 \%$ and $51.1 \%$, respectively.


## Introduction



## OVERVIEW

In this introductory chapter, the problem of Arabic handwriting recognition is defined. The general motivations behind this work are revealed. The characteristics of Arabic cursive script are presented. Main problems of a text recognition system are explained. State of the art of Arabic text recognition is discussed. Our current objectives are clearly stated. Finally, overviews of the systems to be developed are presented.

### 1.1. PROBLEM DEFINITION

Machine recognition of Arabic handwriting is the long-term problem which we address in this research. It is the problem of transforming Arabic handwritten text from a two-dimensional spatial writing form into a symbolic representation. The recognition of Arabic text, the character set of which and similar other character sets are used by more than $30 \%$ of world population and serve in writing of many widespread languages such as Arabic, Farsi and Urdu [1], is a great challenge.

For writing systems other than Arabic, handwriting can be divided into two categories: text of isolated characters and cursive scripts. In Practice, it is difficult to draw a clear distinction between them. A combination of these two forms can be seen frequently. However, Arabic script, is always cursive whether it is handwritten or printed.

Handwriting recognition is a most challenging problem, especially when off-line cursive script recognition is addressed [2-37]. A main problem in off-line recognition is
the loss of dynamic information, chiefly the disappearance of natural segmentation. The lifeless drawing has no speed, no direction; pen up and pen down places cannot be situated. It is practically impossible to restore the real trajectory of the pen. On the other hand, in on-line recognition, [28, 38-42], all such information is available which makes it easier than off-line recognition. Off-line recognition is more difficult than on-line recognition not only because of the loss of dynamic information, but also because the scanning process brings additional noise on the remaining information. However, one can assume that the recognition remains possible, since a man is able to do so.

### 1.2. GENERAL MOTIVATIONS

(a) Economical: Developing input devices capable of capturing data at its source has been one of the primary objectives of the data processing industry. Development of efficient text recognition systems is an effort in this direction. The reading ability of such systems not only provides solutions to data entry problems but also it gives a very flexible and economic means to use computers to solve various real-life problems that might not be solved otherwise. Recent advances in office automation and rising data entry cost, which, according to some estimates, is increasing by 14 percent annually [43], have created more than ever needs for efficient and reliable automatic reading systems to enable communicating with machines in a person's natural modalities such as handwriting.
(b) Seeking for Better Performance: Since the performance of a reading machine heavily depends on the recognition process, for the past several decades, development of a good character recognition technique for handwritten and printed isolated characters and cursive scripts of European languages (English, French, German, etc.) and other languages (Arabic, Farsi, Chinese, etc.) has been a challenging research issue. Numerous interdisciplinary approaches to devise a reliable character recognition technique have been reported [4-15, 20-42]. These techniques differ from one another on their feature extraction and classification schemes depending upon character recognition applications and complexity of input character images. It has been observed that the high variability in handwriting makes it difficult to apply decision theoretic approaches because often real-life data do not
hold assumptions of decision theoretic approaches. For the same and other similar reasons, fuzzy set theoretic approaches have been suggested [2, 3, 16, 17]. Recently, artificial neural networks have been applied in pattern recognition [18, 19].

Therefore, it is clear that the problem of character and cursive script recognition remains an open problem for the researchers to devise solutions. The ultimate goal of such research is to obtain recognition systems with performance comparable to that of human.
(c) Lack of Studies on Arabic Text Recognition: In contrast to the advance, both in theory and practice, in the recognition of other systems of characters, such as Latin and Chinese [9-15, 20-37, 40-42], relatively few studies have been devoted to Arabic character recognition [2-9, 38, 39]. Arabic text recognition is not a direct implementation of the recognition techniques used for other languages. The reason is simply Arabic text characteristics are different than those of texts of other languages. The characteristics of Arabic handwriting are presented in the next section.

### 1.3. CHARACTERISTICS OF ARABIC HANDWRITING

Since handwritten Arabic script is the domain of the proposed system, the reader is first acquainted with some of its characteristics which are different from those of other scripts [2, 3, 44-46]. Many of these characteristics impede the automatic recognition of Arabic script, see Figure 1.1:
(a) Arabic script is cursive and is written from right to left. Several characters are cursive by themselves; they consist of loops and curves (For a clear definition of a loop, see Definition 2.3.1(a)).
(b) An Arabic word consists of one or more cursive subwords, each comprising one or more characters. The discontinuities between subwords result from subwords ending in characters which are not connectable from the left side with the succeeding character. Some Arabic characters appear only at the end of a subword.
(c) The cursive nature of Arabic script is the main problem in Arabic text recognition and makes it difficult to segment a subword directly into characters.

Figure 1.1. Characteristics of Arabic handwriting; the letters a-1 refer to characteristics (a-1) of Section 1.3.
(d) Generally, an Arabic subword is written as a single main stroke, where the pen is not lifted until the stroke is complete. There are also secondary strokes in which the pen has to be lifted up to complete the writing of a subword. Accordingly, any Arabic subword has exactly one main stroke and zero or more secondary strokes.
(e) Usually, a secondary stroke does not touch the main stroke. If this happens inadvertently then, in the algorithms of the following chapters, it will be considered as a part of the main stroke.
(f) Some Arabic characters have the same shape; however, they are distinguished from each other by the addition of secondary strokes, e.g., dots, in different positions relative to the main stroke. Sometimes, the ambiguity of the position of these secondary strokes in handwriting brings out many different readings for one word.
(g) An Arabic character can have different shapes depending on its position in the word (beginning, middle, end, or isolated). This increases the number of fundamental shapes to more than twice the number of characters which complicates the recognition of Arabic text.
(h) Some Arabic characters contain loops, but no more than two loops may be adjacent (share a common link); for a clear definition of a loop, see Definition 2.3.1(a). There is no Arabic stroke ending with two adjacent loops.
(i) Arabic characters vary in size, particularly in width, even within the same font of typeprinted text which is an impeding property in automatic recognition of Arabic text.
(j) Certain Arabic characters may overlap with neighbouring ones. The degree of overlap varies according to the typeface and the typewriter design or the handwriting style. The overlap adds to the difficulty in segmenting characters.
(k) Some Arabic characters use special marks to modify the character accent, such as Hamza and Madda, which are positioned at a certain distance from the character.
(l) The Arabic writing system uses another type of special characters for short vowels, which are called diacritics. When diacritics are used they appear above or below the characters and they are drawn as isolated entities: Although different diacritics on the same set of characters could lead to different words, an Arabic reader is trained to deduce the meaning of undiacriticized text. Thus, diacritics will be overlooked in
this research.
(m) Other characteristics of Arabic handwriting will be mentioned in proper places through the next chapters since they require some terms to be understood first. Such terms are not yet defined.
1.4. MAIN PROBLEMS TO BE SOLVED IN A TEXT RECOGNITION SYSTEM

The main components of a text recognition system are explained below.
(a) Image Acquisition: In this context, it is the process of scanning a document to represent its image as a gray-scale array. Since text is usually printed as dark points on light background (or vice versa) binary images are generally sufficient for text reading. Recent advances in scanning technologies make it possible to scan documents with high resolutions which reach 1200 dots per inch (dpi). For the purpose of handwriting recognition, a resolution of 300 dpi is commonly used.
(b) Preprocessing: The information obtained during acquisition is redundant and noisy. Thus, the acquired image undergoes some preprocessing operations such as thresholding, smoothing, skew detection and correction, gap filling, normalization, thinning, connected component analysis, etc. These are very important operations which highly contribute in determining the level of success of the recognition system. There are plenty of algorithms addressing such low level operations [4757, 70].
(c) Segmentation: This is an ambiguous process which may be counted, or part of it, as a preprocessing step. However, we list this problem separately due to its importance and the difficulty encountered when dealing with it especially in cursive text. Segmentation includes line segmentation, word segmentation, and character segmentation.
(d) Representation: This is the problem of choosing the data structure that both abstractly represents the character or word and is the source of information for the analysis mechanisms necessary for recognition. The accuracy and efficiency of recognition systems rely directly upon the degree of the success of chosen shape representations. In deriving a shape representation, several factors must be
considered: (1) Can most patterns be represented?, (2) Can the representations be easily implemented?, and (3) What is the sensitivity of this representation to noise and other degradations? For example, in character recognition, is this representation very sensitive to small variations in contour images?

Many specific representations have been created for special applications, for example, the straight line approximation of digitized curves [3, 45, 46, 58-65], Fourier descriptors [32, 33, 42], Walsh functions [19], and structural representations [34] are widely used in character recognition. It is worth mentioning that the selected representation determines the characteristic signs of the classification stage.
(e) Classification: This component reads the underlying text by recognizing extracted shape representations as members of known character classes. Text recognition approaches are characterized by their adopted representation. Three main approaches used in the classification stage are:

1. Statistical Approach: $[6,8,10-12,21,30,32,33,38-40,42]$, where the features extracted from the pattern are arranged in a list of ordered values. The feature space is partitioned into a number of classes (equal to the number of patterns to be classified). Distance measures are formulated to quantify the degree of dissimilarity between a pattern and the classes. An unknown pattern is assigned the class which satisfies a minimum distance criterion in the feature space.
2. Structural Approach: $\quad[2-5,7,9,13-17,21-23,26,28,34-37,40]$, It is claimed that the principle underlying the statistical approach is really only appropriate for the recognition of printed characters. The variation of shape of handwritten characters is so large that it is difficult to rely on statistical features to classify them. Thus, a so-called structure analysis method has been applied to handwritten character recognition. In this method, there is no mathematical principle. Rather it is still an open problem. Since a structure can be broken into parts, it can be described by the features of these parts and by the relationships between these parts. Then the problem is how to choose features and relationships between them so that the description gives each
character clear identification.
3. Hybrid Statistical and Structural Approaches: [12, 20, 21, 24, 27, 40], In general, it is difficult to draw a clear boundary to separate the classes in the space of the previous two recognition approaches since a pattern usually has both statistical features and structural attributes. Thus, hybrids of the statistical and structural approaches were tried.

Learning: A system is learned so that the application of a set of input textual materials produces the desired performance. Learning approaches are of same type as those of the classification component, i.e., if a statistical approach is followed in the classification stage then also a statistical approach is used in the learning stage.

### 1.5. STATE OF THE ART OF ARABIC TEXT RECOGNITION

We could have access only to ten studies, [2-9, 38, 39], addressing Arabic text recognition. Of course, there are other references in the field which are not available to us, however, they are rare. These studies are explained below in a chronological order.

Since high variability is expected even in printed characters, due to the large number of font styles and other reasons, Nouh et al. [4], in 1980, suggested a standard Arabic character set, in order to facilitate computer processing of Arabic characters. Isolated characters are simulated and described by suitably chosen components (radicals). The simulated Arabic alphabet is classified utilizing a sequential tree search technique and certain correlation measurements. The disadvantage of the proposed system is the assumption that the incoming characters are generated according to specified standard rules putting strict constrains on font style design.

In 1981, Parhami and Taraghi [8] presented a technique for the automatic recognition of printed Farsi text. The technique is applicable, with little or no modification, to printed Arabic text (it has an alphabet similar to Farsi). Recognition of handwritten Farsi text is not addressed. The authors state that this kind of problem is beyond present day capabilities, since even human readers experience considerable difficulties in this respect. The most important parts of the system are: (1) the isolation of symbols within each subword and (2) the recognition. After symbols are separated according to specified rules, the recognition procedure, which is based on certain
geometric properties of the Farsi symbols such as relative width and the existence of concavities and loops, is applied. Twenty geometric features are used for each symbol to be recognized. Practical application of the technique to Farsi newspaper headlines has been $100 \%$ successful, as reported by the authors. However, smaller type fonts will result in less than perfect recognition. The system is heavily font dependent and the segmentation process is expected to give wrong results in some cases.

Almuallim and Yamaguchi [7], in 1987, proposed a structural recognition method of Arabic handwritten words. Since it is difficult to separate a cursive word directly into characters, words are first segmented into strokes. These strokes are then classified using their geometrical and topological properties. The relative position of the classified strokes are examined, and the strokes are combined in several steps into a string of characters that represents the recognized word. A maximum recognition rate of $91 \%$ was achieved. The system failure, in most of the cases, was due to wrong segmentation of words.

Ramsis et al. [6], in 1988, adopted a statistical approach for Arabic typewritten character recognition. This approach uses accumulative invariant moments to build the feature space. This measure is invariant with respect to size, translation and rotation. The calculations of the moments were implemented accumulatively in what is called accumulative moment invariants. A total of up to seven moments is needed to reach a practical recognition rate which requires heavy computations. In this approach, characters are segmented after they are recognized. Two character segmentation models are proposed and tested. These models seem to be sensitive to font variations. The system is limited to the recognition of typewritten fonts. Moreover, it is font-dependent andsensitive to small variations in input patterns. No figures are reported regarding the system recognition rate and efficiency.

In 1989, El-Wakil and Shoukry [38], proposed a system for recognition of on-line isolated handwritten Arabic characters. Features which are found to be independent of the writer style are represented as a list of integer values, while those which are subject to more variations are represented using a Freeman-like chain code. This mixing of the representation combined with a hierarchical organization of the characters aimed at reducing the recognition time. The system was tested on a small data set of 58 characters (seven samples per character, for a total of $7 \times 58=406$ samples) and the maximum
recognition rate was $93 \%$.
In 1989, also, Amin and Mari [9] presented a structural probabilistic approach to recognize Arabic printed text. The system is based on character recognition and word recognition. Character recognition includes segmentation of words into characters and identification of characters. Word recognition is based on Viterbi algorithm and can handle some identification errors. The system was tested on just few words and no figures are reported about its performance.

In 1990, Al-Emami and Usher [39] presented an on-line system to recognize handwritten Arabic words. Words are segmented into primitives which are usually smaller than characters. The system is taught by being fed by the specifications of the primitives of each character. In the recognition process, the parameters of each primitive are found and special rules are applied to select the combination of primitives which best matches the features of the learned characters. The method requires manual adjustment of some parameters. The system was tested against 170 words written by 11 different subjects for a total of 540 characters. The reported recognition rate was $100 \%$.

Abuhaiba [3], in 1991, presented an off-line character recognition system for handwritten Arabic characters. In this system, the character is converted to a tree structure suitable for recognition. A set of fuzzy constrained character graph models (FCCGM's), which tolerate large variability in writing, is designed. These models are graphs, with fuzzily labelled arcs, used as prototypes for the characters. Rules are applied in sequence to match a character tree to an FCCGM. Although the system proved to be powerful in tolerance to variable writing, speed and recognition rate, it was restricted to isolated characters. Details of the system can be found in [2].

Zahour et al. [5], 1992, presented a method for automatic recognition of off-line Arabic cursive handwritten words based on a syntactic description of words. The features of a word are extracted and ordered to perform a tree description of the script with two primitive classes: branches and loops. In this description, the loops are characterized by their classes and the branches by their marked curvature, their relationship, and if they are in clockwise or counterclockwise direction. Some geometrical attributes are applied to the primitives which are combined to basic forms. A character is then described by a sequence of the basic forms. The reported recognition rate of the system is $86 \%$.

A comparison between the studies mentioned above is shown in Table 1.1 from which the following figures and conclusions can be drawn:
(a) The interest in Arabic text recognition started late, early in the eighties, although for other languages in Germany and U.S.A., the first concepts of machine character recognition appeared early in 1929 [72].
(b) Only two studies, [5, 7], addressed the problem of off-line cursive handwritten recognition.
(c) Eight studies, [2-9], addressed off-line recognition which agrees with the need to devote more effort on automatic reading of static documents since they are the main source of the bulky textual material.
(d) Two studies on isolated handwritten characters, [2, 3], used fuzzy concepts which is preferred to us since the high variability in handwriting, especially in the cursive type, makes it difficult to only apply decision theoretic approaches.
(e) None of the studies addressing handwritten text addressed the line segmentation problem. They were only starting from the image of either a single isolated character or word. In practice, this is not sufficient since we want to read whole pages of text.
(f) Three studies on cursive printed text, [6, 8, 9], were trying to directly segment the word into characters. Actually, this very error prone especially in handwriting.
(g) Three studies, $[5,7,39]$, on cursive handwriting segmented words into primitives which are pieces of line drawing usually smaller than characters. These primitives are recombined in a certain manner to form characters. In our research, we follow a similar strategy due to the reason mentioned in ( f ).
From this survey, it becomes clear that the problem of off-line handwritten cursive Arabic script recognition remains an open problem for the researchers to devise solutions. It requires advanced segmentation techniques, involving the interaction of segmentation and recognition. For efficiency, it is desirable for the recognizer to be free of constraints on primitive number. The ultimate goal is to obtain recognition systems with performance comparable with that of human. The performance of a recognition system remains unknown until it is complete and tested against real data, which is also a good reason for researchers to develop and test new methods in this domain.
Table 1.1. Comparison between ten studies on Arabic text recognition.
Statist.: Statistical approach, Struct.: Structural approach.

| Ref. No. | Year | Text style |  |  |  | Acquisition mode |  | Segmentation |  |  |  | Classification |  | Fuzzy |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Printed | Hand. | Isolated | Cursive | Off-line | On-line | Line | Word | Char | Primitive | Statist. | Struct. |  |
| 4 | 1980. | $\checkmark$. |  | $\checkmark$ |  | $\checkmark$ |  |  |  |  |  |  | $\checkmark$ |  |
| 8 | 1981 | $\checkmark$ | . |  | $\checkmark$ | $\checkmark$ |  | $\checkmark$ | $\checkmark$ | $\checkmark$ |  | $\checkmark$ |  |  |
| 7 | 1987 |  | $\checkmark$ |  | $\checkmark$ | $\checkmark$ |  |  |  |  | $\checkmark$ |  | $\checkmark$ |  |
| 6 | 1988 | $\checkmark$ |  |  | $\checkmark$ | $\checkmark$ |  | $\checkmark$ | $\checkmark$ | $\checkmark$ |  | $\checkmark$ |  |  |
| 38 | 1989 | . | $\checkmark$ | $\checkmark$ |  |  | $\checkmark$ |  |  |  |  | $\checkmark$ |  |  |
| 9 | 1989 | $\checkmark$ |  |  | $\checkmark$ | $\checkmark$ |  | $\checkmark$ | $\checkmark$ | $\checkmark$ |  |  | $\checkmark$ |  |
| 39 | 1990 |  | $\checkmark$ |  | $\checkmark$ |  | $\checkmark$ |  |  |  | $\checkmark$ | $\checkmark$ |  |  |
| 3 | 1991 | $\checkmark$ | $\checkmark$ | $\checkmark$ |  | $\checkmark$ |  |  |  |  |  |  | $\checkmark$ | $\checkmark$ |
| 5 | 1992 |  | $\checkmark$ |  | $\checkmark$ | $\checkmark$ |  |  |  |  | $\checkmark$ |  | $\checkmark$ |  |
| 2 | 1994 |  | $\checkmark$ | $\checkmark$ |  | $\checkmark$ |  |  |  |  |  |  | $\checkmark$ | $\checkmark$ |

### 1.6. CURRENT OBJECTIVES

This research aims at designing an off-line cursive Arabic handwriting recognition system. This is achieved in two steps:
(a) First, we design an Isolated Arabic Character Recognition system (IACR) for recognizing off-line isolated handwritten Arabic strokes.
(b) After having the above system operational, its methods are extended and new other methods are formulated to obtain a Cursive Arabic Script Recognition system (CASR) to recognize off-line handwritten cursive Arabic script having high variability.

Although, the components of these systems are designed to work with Arabic text, other writing systems can benefit from our new methods as well.

The strong cursive nature of Arabic text lends itself better to a structural or hybrid approach. Thus, the structural approach is followed to design both systems. The simplified data flow diagram of Figure 1.2 depicts the basic components of both systems which shows that they incorporate the basic components of any document recognition system. In this research, completely new methods will be developed, to design the representation, classification, and learning components:
(a) Representation: New methods will be developed to obtain the following representations:

1. Straight Line Approximation of Strokes: This is an intermediate representation used in both systems: the Isolated Arabic Character Recognition system and the Cursive Arabic Script Recognition system.

Why? Many specific representations have been created for special applications. Straight line approximation is often an efficient representation for expressing the structural relationship of the major components of an object. One advantage of the straight line approximation of digitized curves is to reduce memory space required for storing the essential structural information present in the characters. It also simplifies the data structures required in processing the character and reduces the required processing time. Many algorithms have been formulated to solve the problem of straight line approximation of digitized curves [3, 45, 46, 58 - 65]. Straight line


Figure 1.2. Simplified data flow diagram of the IACR and CASR systems.
approximations are usually obtained by finding the skeleton of a digital image, using a thinning, distance transform, etc., operation, locating dominant points (end, bifurcation, high curvature, and inflection points), and finally connecting these points by straight line segments. In text and signature recognition, thinning is the widely-used operation to obtain skeletons [47-56, 66]. However, the thinning process has an undesirable side effect of producing artifacts, i.e., artificially created patterns that do not conform with the original patterns, e.g., spurious bifurcation points [67]. Because artifacts adversely affect the accurate extraction of end and bifurcation points from the skeleton, resulting in incorrect representation of a pattern, it may be better to handle such artifacts a priori.

Thus, in this research, thinned images are converted to straight line approximations, consisting of vertex coordinates connected by straight line segments. The likelihood of spurious tails is reduced and, spurious bifurcation points, which are unavoidable when thinning algorithms are used, are removed, and the actual bifurcation points are recovered. The obtained straight line approximations preserve the structural information of the original patterns. The suggested method does not resort to distortable geometrical properties.
2. Temporal Information: This helps to segment strokes in both systems. Why? Many recognition systems have gained more sophistication in recent years by exploiting the dynamic information and drawing from research on understanding the handwriting process and on the analysis of character shape deformation [38-41]. The dynamic information consists of the number of the strokes, the order of the strokes, and the direction of the writing of each stroke. The availability of dynamic information often permits the use of more accurate recognition algorithms. Since, textual documents are usually stored as off-line / static images, many techniques have been suggested to recover such dynamic information from these images and exploit this information to recognize the textual material [67-69].
Off-line data have been converted by line thinning to sequences of points similar to on-line data (but without the timing information), achieving reasonable recognition accuracy [31, 71]. Recently, a heuristic-rule-based tracing algorithm has been presented which transforms the skeleton of an offline image of a signature into a particular sequence of strokes, simulating the original writing sequence [67]. Although robust recognition and verification are claimed, the system uses a relatively complicated set of rules, that mainly operate at pixel level. In [68], the recovery of some temporal clues from static images of handwritten text is addressed. Of these clues are the position of the stroke segment, classification of end points, junction interpretations, stroke segment curvature and distances between stroke endpoints. Although the temporal clues recovered in [68] appear useful, the technique operates on a
multi-level, high-resolution digital image, and must inevitably incur considerable memory and processing overheads.

Thus, after recognizing the benefits of such dynamic information and due to drawbacks of some previous methods, we will try to recover the original sequence of writing off-line Arabic strokes as an aid to their straight line approximations to segment strokes.
3. Stroke Segmentation: Here methods are introduced to accept a straight line approximation and temporal information of a stroke and produce small basic units constituting the stroke as the final representation.
Why? The cursive nature makes it difficult to segment a subword directly into characters. Rather, a subword is segmented into basic units which are usually smaller than characters.
(b) Classification: As mentioned above, strokes will be segmented into small basic units in both systems. In the CASR system, these basic units are combined into strings of basic units. These strings are hypothesized into characters to obtain the target word. Also, we adopt a new method for line segmentation which tolerates large variations in handwriting. Our approach; is constructed using concepts from sequential machine, fuzzy set, and graph theories.
Why?

1. The strategies for cursive script recognition can be classified into three broad categories [10]:
i. In the first category, the word is segmented into several characters and the character recognition techniques are applied to each segment. This method depends heavily on the accuracy of the segmentation points found. However, such accurate segmentation technique is not yet available, and may need to combine the interaction of word segmentation and character recognition. Thus, this strategy will not be used in our research.
ii. In the second category, the whole words are recognized without doing any kind of formal segmentation. This strategy may be useful when the word vocabulary is very limited. Our current and long term goals go far
beyond this since we are looking for an unconstrained cursive handwriting recognition system in all respects.
iii. The third category is a compromise solution between the above two schemes. It does a loose segmentation to find a number of potential segmentation points in the pre-segmentation procedure. The final segmentation and the word length are determined later in the recognition stage by the help of a lexicon which is the current more efficient trend in cursive handwriting recognition.

Thus, in our system for handwritten cursive recognition, a similar strategy is followed, but without using a lexicon. A cursive word is recognized through a hierarchical analysis by proceeding from segmented small basic units, to strings of basic units, to hypothesized characters, and then to the final string of characters.
2. The methods used by researchers to segment lines, words and characters were primarily developed for printed text in which a horizontal baseline usually exists. This enabled them to use simple horizontal and vertical projections or Hough transform methods in segmentation. This means that, in case of printed text, segmentation can be usually performed as an independent process before classification. The situation is different in handwritten cursive text due to the following reasons:
i. A horizontal baseline does not exist in unconstrained handwriting.
ii. There is a change in the slant even in a single line of handwritten text.
iii. Secondary strokes (e.g., dots and dashes) are not carefully plotted, in handwriting, with respect to main strokes.

Thus, the above reasons make it natural to us to view line and character segmentation processes as part of the classification process and to seek for other general methods for initial segmentation. Initial. segmentation is already discussed in (a)3, above.
(c) Learning: The consequences of this problem are similar to those of the classification problem. Thus, the learning methods and the classification methods will be developed to form one couple. The important thing here is that, as you
notice in the data flow diagram of Figure 1.2, the learning stage comes after the classification stage! Actually this arrangement is preferred to us because our philosophy lies in: "What is this i? If you know then you earn, otherwise come to learn." This means that trying to recognize comes first. If the system fails then it is taught.

### 1.7. SYSTEMS OVERVIEW

A data flow diagram of the Isolated Arabic Character Recognition system (IACR) is shown in Figure 1.3. My new contribution is represented by the filled processes / rounded rectangles, i.e., Straight Line Approximation, Enforcement of Temporal Information, Stroke Segmentation, Stroke Recognition, and Stroke Learning. The data flow diagram of the IACR system consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten stroke is captured using a scanner.
(b) Low Level Preprocessing: which includes:

1. Smoothing: where the acquired binary image of the stroke is smoothed.
2. Thinning: where the smoothed binary image of the stroke is thinned.
(c) Straight Line Approximation, Chapter 2: which accepts a smoothed thinned binary image of the stroke and produces two representations of the stroke. The first representation is a direct straight line approximation and the other is called a reduced graph which is also a straight line approximation with loops represented as vertices.
(d) Enforcement of Temporal Information, Chapter 3: Here temporal information of the stroke are extracted from its straight line approximations.
(e) Stroke Segmentation, Chapter 4: The reduced graph and temporal information are used to segment the stroke into small units, called primitives. The reduced graph and segmented primitives are necessary inputs for the subsequent two processes, i.e., Stroke Recognition and Stroke Learning.
(f) Stroke Recognition, Chapter 5: which receives a fuzzy sequential machine, which is a representation of the learned strokes, a reduced graph, and primitives of the stroke. It outputs recognition results indicating whether the stroke belongs to a


Figure 1.3. Data flow diagram of the IACR system.
certain class or it could not be recognized. If a stroke could not be recognized, then its acceptance information, represented as a tree data structure, is fed to the Stroke Learning process.
(g) Stroke Learning, Chapter 6: This process gets as inputs the fuzzy sequential machine which was used in recognition but failed to recognize the stroke, a reduced
graph of the stroke, primitives of the stroke, and the acceptance tree which is passed by the Stroke Recognition process. It outputs a new fuzzy sequential machine which can recognize the input stroke and strokes of the old machine and variants of these strokes.
Experimental results and performance of the IACR system are reported in Chapter 7.
Figure 1.4 shows a simplified data flow diagram, of the Cursive Arabic Script Recognition system (CASR). Again, my new contribution is represented by the filled processes / rounded rectangles in the figure, i.e., Straight Line Approximation, Enforcement of Temporal Information, Stroke Segmentation, Token Recognition, Token Learning, Learning of Token Strings, Line Extraction and Stroke Ordering, and Word Formation. The data flow diagram of the CASR system consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten page of cursive Arabic script is captured using a scanner.
(b) Low Level Preprocessing: which includes:

1. Smoothing: The acquired binary image of the page is smoothed.
2. Stroke Extraction: Here single component strokes are extracted, where each stroke is represented as a smoothed binary image.
3. Thinning: The smoothed binary images of the strokes are thinned.
(c) Straight Line Approximation, Chapter 2: which accepts smoothed thinned binary images of the strokes and produces two representations for each stroke; a direct straight line approximation and reduced graph.
(d) Enforcement of Temporal Information, Chapter 3: Here temporal information of the stroke are extracted from the reduced graphs of the strokes.
(e) Stroke Segmentation, Chapter 8: A cursive stroke is segmented into small parts, called tokens. Tokens are logical units which are usually larger and more suitable for cursive script than the primitives of the IACR system.
(f) Token Recognition, Chapter 9: For every input token, this process finds whether it belongs to a certain class or it could not be recognized. If a token could not be recognized, then its acceptance information, represented as a tree data structure, is fed to the Token Learning process.

(g) Token Learning, Chapter 10: This process gets as an input the acceptance tree which is passed by the Token Recognition process, in addition to other inputs. It outputs a new token fuzzy sequential machine which can recognize the input token and tokens of the old machine and variants of these tokens.
(h) Learning of Token Strings, Chapter 11: where tokens are recombined into meaningful sets of tokens; logical token strings. Logical token strings are associated with possible interpretations and their fuzzy features.
(i) Line Extraction and Stroke Ordering: which includes:
4. Separating Main and Secondary strokes, Chapter 12: where strokes which can represent secondary strokes are marked. Remaining strokes are main strokes.
5. Extracting Lines and Ordering Strokes, Chapter 12: where lines are extracted and their constituent main strokes are ordered from right to left. Secondary stroke candidates are presented to main strokes.
(j) Word Formation: which includes:
6. Common Shape Interpretations of Main Strokes, Chapter 13: where all possible basic shape interpretations of main strokes are enumerated and represented in a tree data structure. We call this tree Enumeration and Requirement Tree (ERT), in which information about secondary strokes required to associate basic shapes to form characters is included.
7. Character Formation, Chapter 13: where ERT's are combined with presented candidate secondary strokes to form characters. Assignment problems are formulated and solved for this purpose. The solution which exhibits the minimum cost is selected. This process results in some redundant secondary strokes which cannot be combined with basic shapes to form characters:
8. Manipulating Redundant Secondary Strokes, Chapter 13: Redundant secondary strokes are manipulated to form some other characters which are inserted in their proper places within lines. The final result is a list of ordered lines of ordered lists of words.
Experimental results and performance of the CASR system are reported in Chapter 14.

## Part One

Preprocessing


Figure I.1. Preprocessing operations in the IACR system.

## OVERVIEW

In this part, preprocessing operations are addressed. A data flow diagram of preprocessing operations in the Isolated Arabic Character Recognition system (IACR) is shown in Figure I.1. This data flow diagram consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten stroke is captured using a scanner.
(b) Smoothing: where the acquired binary image of the stroke is smoothed.
(c) Thinning: where the smoothed binary image of the stroke is thinned.
(d) Straight Line Approximation, Chapter 2: which accepts a smoothed thinned binary image of the stroke and produces two representations for the stroke. The
first representation is a direct straight line approximation and the other is called a reduced graph which is also a straight line approximation with loops represented as vertices.
(e) Enforcement of Temporal Information, Chapter 3: Here temporal information of the stroke are extracted from its straight line approximations. Temporal information helps to segment strokes.

Figure I. 2 shows a data flow diagram of preprocessing operations in the Cursive Arabic Script Recognition system (CASR). This data flow diagram consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten page of cursive Arabic script is captured using a scanner.
(b) Smoothing: The acquired binary image of the page is smoothed.
(c) Stroke Extraction: Here single component strokes are extracted, where each stroke is represented as a smoothed binary image.
(d) Thinning: The smoothed binary images of the strokes are thinned.
(e) Straight Line Approximation, Chapter 2: which accepts smoothed thinned binary images of the strokes and produces two representations for each stroke. The first representation is a direct straight line approximation and the other is a reduced graph.
(f) Enforcement of Temporal Information, Chapter 3: Here temporal information of the strokes are extracted from their straight line approximations.

Notice that in Figures I. I and I.2, our new contribution is represented by the filled processes / rounded rectangles, i.e., Straight Line Approximation and Enforcement of Temporal Information.

Although stroke segmentation in both the IACR and CASR systems is an extra preprocessing step, it is not addressed in this part. Instead, it is postponed to their corresponding parts; Two and Three. This is due to the difference in complexity of the problem in these systems which implies different ways of stroke segmentation.

There are some points which we make clear. Firstly, the images which appear in the following chapters are off-line images which are captured using a scanner. Secondly, for the second process, smoothing, in the IACR and CASR systems, a suitable smoothing


Figure I.2. Preprocessing operations in the CASR system.
algorithm can be found in [57]. Finally, Whenever thinning is mentioned, it is achieved by using the "Safe Point Thinning Algorithm," or SPTA [66]. The reason for selecting SPTA is not necessarily that the algorithm gives the best results, but mainly because it provides a good compromise between quality and speed. Other thinning algorithms, which may produce better results, can be found in references [48-55].

## Straight Line

## Approximation



## OVERVIEW

In this chapter, methods are developed to convert a smoothed thinned binary image of a stroke into a straight line approximation, consisting of vertex coordinates connected by straight line segments. A two-subprocess data flow diagram of the process is shown in Figure 2.1:
(a) Direct Straight Line Approximation, Sections 2.2, 2.3: The input to this module is a smoothed thinned binary image of a stroke. The output is a direct straight line approximation, since it may contain loops as they are without any special manipulation. The method, Section 2.2, incorporates heuristics which ensure a unique centre for each intersection vertex, and to reduce the likelihood of spurious tails. However, there are some cases in which these straight line approximations have unavoidable spurious artifacts introduced by the thinning process. These artifacts do not correspond to true segments in the original image. Thus, an alternative enhanced method, Section 2.3, is described. It uses the distance transform of thinned binary images to identify spurious bifurcation points, remove them, and recover the original ones. The obtained straight line approximations preserve the structural information of the original patterns. The method does not resort to distortable geometrical properties.
(b) Loop Reduction, Section 2.4: This subprocess accepts a direct straight line approximation, produced by the previous subprocess, and outputs a loopless


Figure 2.1. Data flow diagram of the Straight Line Approximation process.
straight line approximation (reduced graph), i.e., it does not have loops, as an auxiliary compact representation of the stroke. Here, loops are replaced by vertices with features.

### 2.1. STRAIGHT LINE APPROXIMATION OF DIGITAL CURVES

In this section an algorithm is described to convert a thinned image of a stroke into a straight line approximation, consisting of vertex coordinates connected by straight line segments. The algorithm incorporates heuristics which ensure a unique centre for each intersection vertex, and to reduce the likelihood of spurious tails.

### 2.1.1. Definitions

Refer to Figure 2.2 while reading the following definitions.
(a) A stroke is the line art drawn from pen down to pen up, e.g., the stroke shown in Figure 2.2(a).
(b) A straight line approximation of a stroke is represented as a non-directed graph, $\mathrm{G}=(\mathrm{V}, \mathrm{L})$, consisting of a set of vertices, V , and a set of links, L. Each vertex $\mathrm{v} \in$ V is represented by its $\mathrm{x} \& \mathrm{y}$ coordinates and each link in L is associated with exactly two vertices in V. Figure 2.2(b) is a straight line approximation / non-


Figure 2.2. Explaining Definitions 2.1.1.(a-j): (a) handwritten stroke, and (b) straight line approximation of the stroke.
directed graph, G, of the stroke of Figure 2.2(a). The set of vertices is $V=\{i, i=$ $1,2, \ldots, 18\}$, and the set of links, $L$, is clear from the figure.
(c) A terminal vertex is a vertex having no more than one associated link, e.g., vertices 10 and 18.
(d) An intersection vertex is a vertex having three or more associated links in G, e.g., vertices 1 and 9.
(e) Two vertices are said adjacent if they are associated with at least one link, e.g., vertices 1 and 2 are adjacent.
(f) A path is a sequence of vertices such that each two consecutive vertices, in the sequence, are adjacent, e.g., the sequence of vertices $1,2,3,4,5,6,7,1$.
(g) An elementary path is a path which does not use the same vertex more than once; e.g., the path which consists of vertices 9,10 .
(h) A tail is an elementary path with the following properties

1. One end vertex of the path is a terminal vertex, $u$,
2. The other end vertex of the path is an intersection vertex, $v$, and
3. None of the intermediate vertices, in the path between $u$ and $v$ is an


Figure 2.3. Thinned binary image to explain Definitions 2.1.1(k-q).
intersection vertex.
For example, in the elementary path $9,11,12,13,14,15,16,17,18$, vertex 18 is a terminal vertex, vertex 9 is an intersection vertex, and none of the vertices 11 to 17 is an intersection vertex. Thus, this constitutes a tail.
(i) The length of a tail is the sum of the lengths of the links constituting it.
(j) A spurious tail is a tail the length of which is less than a certain threshold.

The following definitions apply to thinned images which can be produced using any thinning algorithm (or any other method) such that lines of one-point width are obtained.
For easy understanding of these definitions, refer to Figure 2.3.
(k) The 8-neighbourhood of a black point, p, is a $3 \times 3$ window centred at p, e.g., the 8neighbourhood of black point, $f$, is the $3 \times 3$ window defined by the points $b, c, d, e$, $f, g, h, i$, and $j$.
(l) Two black points are adjacent if one belongs to the 8-neighbourhood of the other, e.g., points a and d.
(m) An end point, $E P$, is a black point that is adjacent to only one other black point, e.g., point a.
(n) A bifurcation point, $B P$, is a black point that is adjacent to at least three other black points, e.g., points $e, f, h$, and $i$.
(o) A dominant point is an EP or BP. In the literature, a dominant / critical point is an end, bifurcation, high-curvature, or inflection point $[63,70]$. According to the author's experimentation for the purpose of text recognition, it is found that the most important dominant points are EP's and BP's, with straight line segments between them, which are sufficient to preserve the structural properties of the
textual shape.
(p) A cluster of dominants is a set of dominant points, each of which is adjacent to at least one point in the cluster and is not adjacent to any dominant point that is outside the cluster under consideration, e.g., the set of dominant points $e, f, h$, and $i$.
(q) The mean point of a cluster of dominants, containing n dominant points, equals the sum of these points divided by their number, $n$.
A formal description of the straight line approximation of a stroke follows.

## Algorithm 2.1

Use: To obtain direct straight line approximation/s for input stroke/s
Input: Smoothed thinned binary image/s of stroke/s
Output: Direct straight line approximation/s of stroke/s

## Procedure:

Step 1. In the thinned image of the stroke, the set of dominant points, $S$, is found.

Step 2. For the set $S$, the set of clusters of dominants is found. The mean point of each cluster is found. Each cluster of dominants will correspond to a vertex in the final straight line approximation, $G$, with $x \& y$ coordinates equal to those of the mean point of the cluster. Thus, initially the number of vertices in $G$ equals the number of clusters.

Step 3: The dominant points which are adjacent only to dominant points are deleted from S.

Step 4. The remaining dominant points of each cluster of dominants are connected, by straight line segments, to the mean point of that cluster. These remaining dominant points and new segments are added to $G$.

Step 5. Each path of consecutive points connecting two points $s_{1}, s_{2} \in S$, such that none of the intermediate points between $\mathrm{s}_{1}$ and $\mathrm{s}_{2}$ is a bifurcation point, is divided into a number of straight line segments. The length of each segment is proportional to the line thickness of the stroke. New vertices are produced at the ends of these segments. The generated vertices and segments are added to $G$.

Step 6. Spurious tails, in G, are deleted. The threshold for the length of a spurious tail is proportional to line thickness.

Step 7. Segments, in G, which are approximately collinear are merged.

In Step 2 of the above algorithm, a temporary image is created with the same size as the original stroke image. In this image, only dominant points are retained while others are deleted. Each cluster of dominants is viewed as a connected component. Thus, the algorithm of [56] is used to extract the set of connected components (i.e., the set of clusters of dominants).

The generated segments in Step 5 can be of variable or equal length. If they are variable, then a segment can be grown point by point. A point is added to a segment if it is collinear with that segment to a certain extent. However, in our algorithm, the skeleton is initially reduced to equal length segments rather than variable length segments since it is more noise-immune to quantify the collinearity measure on a segment basis rather than on a point basis.

In Steps 5 and 6, the segment length and spurious tail length are determined as follows. Let $N_{1}$ be the number of points in the stroke before thinning. In fact, $N_{1}$ equals the area, in points, of the stroke. An approximate length of the line script of the stroke is the number of points, $N_{2}$, in the thinned image. An approximation of the line thickness is given by $w=N_{1} / N_{2}$. In Step 4, the length of each segment is set equal to $\left\lceil k_{1} \times w\right\rceil$, and in Step 5, the threshold for spurious tail determination is set equal to $k_{2} \times w$, where $k_{1}$ and $k_{2}$ are constants. Suitable range of $k_{1}$ and value of $k_{2}$, which were empirically found, are $1.0 \leq \mathrm{k}_{1} \leq 2.0$ and 0.5 , respectively.

### 2.1.2. Example

The smoothed thinned binary image of a stroke and the image after thinning are shown in Figures 2.4(a, b), respectively. To obtain a straight line approximation of this stroke, Algorithm 2.1 is applied as follows:

Step 1. In the thinned image of Figure 2.4(b), there are 2 end points, which are obvious, and 6 bifurcation points which are pointed to by arrows. Thus, S has initially 8 dominant points.

Step 2. In S, there exists 6 clusters of dominants: two of which represent 2 end vertices while the others represent 4 intersection vertices in the final G. Notice that since


Figure 2.4. (a) Smoothed image, (b) smoothed and thinned image with bifurcation points being pointed to by arrows, and (c) its straight line approximation, G .
each of the left-most five clusters consists of only one point, the mean point of each of these clusters equals its constituent point.

Step 3. Since each point in $S$ is adjacent to at least another point which is not dominant, no dominant point is deleted.

Step 4. For the right-most cluster, since no dominant point is deleted in Step 3, each of its three dominant points is connected by a straight line segment to the mean point of the cluster. The remaining points and new segments are added to $\mathbf{G}$.

Step 5. An approximate length of the line script of the stroke is found to be 4.33 points. The constant $k_{1}$ is set to 1.0 . Thus, the initial segment length is $\left\lceil k_{1} \times w\right\rceil=\lceil 1.0 \times$ $4.337=5$ points. Each path of consecutive points connecting two points $s_{1}, s_{2} \in S$, such that none of the intermediate points between $s_{1}$ and $s_{2}$ is a bifurcation point, is divided into a number of straight line segments. The length of each segment is 5 points. New vertices are produced at the ends of these segments. The generated vertices and segments are added to $G$.

Step 6. The constant $k_{2}$ is set to 0.5 . The threshold for spurious tail determination is $\mathrm{k}_{2} \times \mathrm{w}=0.5 \times 4.33=2.17$ points. According to this threshold, the stroke has no spurious tails.

Step 7. Finally, the segments which are approximately collinear are merged yielding the graph, G, shown in Figure 2.4(c).

### 2.1.3. Causes of Failure

There are some cases in which Algorithm 2.1 produces straight line approximations which have spurious artifacts introduced by the thinning process. These artifacts do not correspond to true segments in the original image, and are of two types:
(a) Elongation Artifact: When two line segments converge to a point with a small angle such as a cusp, the thinning algorithm generates an elongated segment at the end by merging the two line segments near the point. Figure 2.5 illustrates the image of two lines converging to a point with a small angle and the corresponding pattern that may be produced by a thinning algorithm. It can be observed that, the smaller the converging angle becomes, the more prominently the elongation artifact emerges. The cause of the elongation artifact lies in the fact that when two


Figure 2.5. Elongation artifact: (a) two lines converging to a point, and (b) a pattern that may be produced by a thinning algorithm.


Figure 2.6. Bifurcation artifact: (a) two crossing lines, and (b) a pattern that may be produced by a thinning algorithm.
unthinned line segments meet or cross, their junction is an area rather than a point. The geometry of the junction area depends on the thickness of the lines and the meeting or crossing angle. In Figure 2.5(a), when boundary points are stripped, by applying a thinning algorithm, it may happen that one right-most point possesses the properties of an EP so that it can not be deleted by a next stripping iteration. Similarly, two other EP's are generated at the left side of the stroke. Thus, the thinning algorithm ends by generating a BP. The segment between the BP and the right-most EP is an elongation artifact.
(b) Bifurcation Artifact: When two or more line segments cross each other at a junction point, a thinning algorithm may generate a bifurcation artifact for the same reason that it causes an elongation artifact. The simplest case of a bifurcation artifact occurs when only two lines intersect as shown in Figure 2.6(a). In this simple case, a bifurcation artifact can be considered as the overlapping of two elongation artifacts in opposite directions. As a result, two spurious bifurcation points, SBP's, depicted as A and B in Figure 2.6(b), are generated separated by an elongation segment. The length of the elongation segment is a function of the line thickness and crossing angle. SBP's are undesirable as they complicate the skeleton representation which
may degrade any further postprocessing operation, e.g., recognition. Hence, it is better to identify and remove them and recover the original intersection point to ensure a natural representation of a pattern.
In the algorithm of the following section, a bifurcation artifact can be identified and replaced with a single vertex. The replacement vertex is approximately equivalent to the original crossing point in the sense that the topological properties of the thinned shape are preserved. Also, the algorithm incorporates new measures for line thickness and spurious tail length determination.

### 2.2. ENHANCED STRAIGHT LINE APPROXIMATION

In [67], a heuristic was suggested to identify SBP's, remove them, and recover original ones. This heuristic depends on certain geometrical properties in the proximity of the spurious points. However, although geometrical properties are maintained in ideal images, they are rarely preserved in digital ones due to technical reasons such as resolution of scanning device, thinning algorithm, etc. Moreover, that heuristic deals with the simple case, i.e., only two intersecting lines since the output of the algorithm is a binary image in which a bifurcation point can not have more than four branches. In this section, a method is presented which identifies simple as well as complex spurious bifurcation points, removes them, and substitutes with one point that is approximately equivalent to the original crossing point in the sense that the topological properties of the thinned shape are preserved. This new method does not resort to distortable geometrical properties.

Figure 2.7 illustrates two intersecting straight lines of uniform thickness with an angle of intersection, $\alpha$. The line thickness is $w$ and the area of intersection is the diamond shown by dashed lines. When thinned, the skeleton for the two lines may look like the dotted lines with two spurious bifurcation points, B and D . Ideally, the lines must intersect at one point, $C$. The distance from $A$ to $B, d_{1}$, is the same as the distance from $D$ to $E$. Also, the distance from $B$ to $C, d_{2}$, equals the distance from $C$ to $D$. As long as $\alpha$ is not smaller than a certain threshold, the distance $\mathrm{d}_{1}$ is larger than $\mathrm{d}_{2}$. Let us define the domain of a bifurcation point, BP , as the circle which is centred at BP and whose radius equals the distance of BP to the nearest point on the boundary of the intersecting lines. For


Figure 2.7. Two intersecting lines: line thickness $=w$, angle of intersection $=\alpha$, the skeleton is dotted, the bifurcation area is the diamond surrounded with dashed lines, and the circles are the domain circles of bifurcation points B and D .
example, the domain of the bifurcation point, $B$, is a circle centred at $B$ with a radius $d_{1}$. Most of the skeleton points that lie inside the domain of B belong to the area of intersection, i.e., the diamond. The same applies to point $D$. As long as $d_{2} \leq d_{1}$, the domain circles of $B$ and $D$ intersect or touch each other. Thus, a criterion for two BP's to belong to the same crossing area is that their domain circles must intersect or at least touch each other which is the new idea for determining SBP's.

A bifurcation point was defined as a black point that is adjacent to at least three other black points which is different than the usually used definition of a bifurcation point. In the literature, a bifurcation point, BP, is defined as a black point whose crossing number is at least 3 where the crossing number is the number of white to black transitions in the 8-neighbours of the point [67]. We find that our definition of a BP is more flexible and useful as it is clear from Figure 2.8. The point configuration shown in this figure frequently results when thinning algorithms, that produce skeletons of one-point width,


Figure 2.8. A point configuration that may result after thinning.
are applied to binary images. If the crossing number is used, then neither of the four centred points is a BP. However, using our definition the four centred points are BP's. Ultimately, these four points must be merged into one BP as this is one of the advantages of the enhanced straight line approximation algorithm which is developed in this section. Before the straight line approximation algorithm is described, the following definitions are presented.

### 2.2.1. Definitions

(a) The distance of a black point from the image background is that which is determined by a suitable distance transform $[56,73]$. The chamfer $3 / 4$ distance transform is recommended since the maximum distance error does not exceed $8.1 \%$, as reported in [73].
(b) If A is a BP with distance d , then its domain square is a square centred at A with side $=2 \times d-1$. This is an approximation of the domain circle, defined earlier, for the sake of an easy implementation. The domain square of an EP is itself. For example, consider the thinned image of Figure 2.3. If we assume that the distance of point a, which is an EP, equals 2, then its domain square is that square centred at a with side $=2 \times 2-1=3$, i.e., it is a $3 \times 3$ window centred at a. Similarly, if the distance of points $e, f, h$, and $i$, which are BP's, is 2 , then their domain squares are the $3 \times 3$ windows centred at $e, f, h$, and $i$, respectively.
(c) Two domain squares are adjacent if they have at least two adjacent black points. A dominant area is a set of adjacent domain squares each of which is not adjacent to any domain square that lies outside the dominant area under consideration. In Figure
2.3, the domain squares of points $\mathrm{e}, \mathrm{f}, \mathrm{h}$, and i are adjacent to each other and they are not adjacent to any other domain square. Thus, they constitute a dominant area.
(d) A black point whose all black neighbours belong to a dominant area is an Inside-Dominant-Area Point, IDAP, e.g., points a, d, e, f, h and in Figure 2.3.

Now, a formal description of the enhanced straight line approximation algorithm follows.

Algorithm 2.2

Use: $\quad$ To obtain enhanced direct straight line approximation/s for input stroke/s
Input: Smoothed thinned binary image, I, of one-point width for each input stroke. We assume that the distance of $I$ is already calculated, using a suitable distance transform, as part of the thinning process.

Output: Direct straight line approximation/s of stroke/s

## Procedure:

Step 1. The majority of the points in $I$ have the distance $d_{\text {maj }}$. An estimated line thickness of the stroke is $w=2 \times d_{\text {maj }}-1$. A parameter of significance is $\beta=k \times w$, where $k$ is a constant. If the total number of points in $I$ is less than $n_{\text {dot }}=\lceil\beta\rceil$ then the skeleton is considered as a single dot stroke which is represented by a vertex located in the mean point of the skeleton's points and the algorithm is exited.

Step 2. The set, $S$, of dominant points in $I$ is found. Mark each black point, $p$, in $S$ along with black points of I that lie inside the domain square of $p$. If the distance of p is d then the side length of its domain square is $2 \times \mathrm{d}-1$. Identify IDAP's.

Step 3. A temporary image, $T$, is created with the same size as the original stroke image. In this image, only the points which were marked in Step 2 are retained while others are deleted. The set of dominant areas is found by extracting the set of connected components of T where there is a one-to-one correspondence between the dominant areas and connected components. Each dominant area will correspond to a vertex in the final straight line approximation with $x \& y$ coordinates equal to those of the mean point of the skeleton's points that lie inside that area. Thus, initially the number of vertices equals the number of dominant areas:

Step 4. Delete the IDAP's from T. The remaining points in T will be vertices in
the final straight line approximation. Each vertex found in Step 3 is connected to the points that are not IDAP's and lie inside the same dominant area.

Step 5. Each path in the skeleton image, $I$, connecting two points $p_{1}, p_{2} \in T$ is divided into a number of straight line segments, the length of each $\leq\lceil\beta\rceil$.

Step 6. Tails whose length is less than w/2 are considered spurious; hence they are deleted. If the remaining straight line approximation has no intersection points and its length is less than $l_{\text {dot }}=\beta$, then it is considered as a dot which is represented by the straight line approximation's centre of gravity and the algorithm is exited.

Step 7. Segments which are approximately collinear are merged. That is, if the angle, $\alpha$, between two segments is larger than a specified threshold, they are merged into one segment.

### 2.2.2. Examples

Figure 2.9(a) shows a smoothed binary image of a handwritten $\times$. The skeleton, I, is obtained using the SPTA thinning algorithm [66] as illustrated in Figure 2.9(b) with the distance, indicated using a chamfer 3/4 distance transform [73]; the distance is coded using hexadecimal numbers and is $f$ times the actual Euclidean distance where $f=3$ for a chamfer $3 / 4$ distance transform. Algorithm 2.2 is applied to obtain a straight line approximation for this stroke as follows:

Step 1. The majority of the points in $I$ have the distance $d_{m a j}=8 / 3$. The estimated line thickness is $w=2 \times 8 / 3-1=4.3$ points. The parameter of significance is $\beta=1.5 \times 4.3 \approx 6.5$ (A suitable value for $k$ was found to be 1.5 ). Since the total number of dots in the skeleton is not less than $\lceil\beta\rceil=\lceil 6.5\rceil=7$ points, the skeleton is not a dot stroke and the algorithm is continued.

Step 2. This skeleton has 6 dominant points: two BP's and four EP's. The domain squares for the EP's are the points themselves. The distances of the upper and lower BP's $\operatorname{are} \mathrm{F} / 3=5$ and $\mathrm{E} / 3 \approx 5$, respectively, with domain squares each of side length $=2 \times 5$ $-1=9$ points. The black points which lie inside these six domain squares are marked.

Step 3. The points which are marked in Step 2 constitute the image T as shown in Figure 2.9(c). The domain squares of the two bifurcation points are adjacent constituting one dominant area. Thus, the image, T , has five connected components which


Figure 2.9. (a) Binary image of a stroke, (b) the skeleton, I, with indicated distance after applying a chamfer 3/4 distance transform, (cont.)
are represented by five vertices in the straight line approximation; one for each EP and the fifth for the dominant area.

Step 4. The points of the dominant area, except the four end points of that area, are IDAP's, hence they are deleted. The remaining four points of the dominant area are connected to the vertex that represents that area via straight line segments.

Step 5. Each path in the skeleton of Figure 2.9(b) connecting two points in T, after the removal of IDAP's, is divided into straight line segments, the length of each is $\leq$ $\lceil\beta\rceil=\lceil 6.5\rceil=7$ points. The obtained straight line approximation is shown in Figure 2.9(d).


Figure 2.9. (c) dominant areas, the initial $T$ image; chamfer $3 / 4$ distance is coded using hexadecimal numbers, (d) the straight line approximation, and (e) the straight line approximation after collinear segment merge.

Step 6. The threshold for spurious tail determination equals $\mathbf{w} / 2=2.15$ points. In Figure 2.9(d), there is no tail the length of which is less than 2.15 points. Thus, no parts are deleted from the straight line approximation. The length of this approximation is not less than $l_{\mathrm{dot}}=\beta=6.5$, hence, the stroke is not a dot and the approximation remains
as it is.
Step 7. Finally, after merging of collinear segments with angle greater than $165^{\circ}$, the final straight line approximation of Figure 2.9(e) is obtained which has only one intersection point as it should be.

Figure 2.10 shows the binary image and the skeleton of another stroke, Arabic Ha . The skeleton has bifurcation points with a point configuration similar to that of Figure 2.8. The algorithm successfully yielded the straight line approximation shown in Figure 2.10(c).

Figure 2.11(a) shows the image of four intersecting lines. Ideally, the lines must intersect at one point which can not be achieved by a thinning algorithm as it is clear from Figure $2.11(\mathrm{~b})$ which has many SBP's. Using Algorithm 2.2, the straight line approximation of Figure 2.11 (c) is obtained with only one intersection point; interesting!

### 2.2.3. Results and Causes of Failure

The algorithm was tested on the handwriting of two writers. The writers were given a list of strokes with one intersection point where every stroke represents one or two characters. The intersection point is associated with four branches which is the result of two crossing segments. The writers provided 133 samples. Spurious bifurcation points and tails could be detected and removed and the original bifurcation point could be recovered in $98.5 \%$ of the samples. In $1.5 \%$ of the samples, the causes of failure are:
(a) For intersecting lines, the angle of intersection, $\alpha$, is small which results in a long elongation segment between the spurious bifurcation points. Thus, the domain squares are not adjacent which ends with the bifurcation points as they are without being merged.
(b) The length of an actual spurious tail is not less than the threshold of spurious tail determination which is one half the line thickness of the stroke. These tails are thinning-elongation and blob artifacts. The algorithm can not be forced to remove such tails as they may conform with the original pattern and thus may be informative parts of it.

(a)

(b)

Figure 2.10. (a) Binary image, (b) the skeleton with a point configuration like the one shown in Figure 2.8, (cont.)

(c)

Figure 2.10. (c) the straight line approximation with one intersection point.

(a)

Figure 2.11. (a) Four intersecting lines, ideally they should intersect at one point, (cont.)


Figure 2.11. (b) the skeleton with several bifurcation points, and (c) the final straight line approximation with only one intersection point.


Figure 2.12. A graph, G, to explain Definitions 2.3.1.(a-c).

### 2.3. LOOP REDUCTION

It is easier to deal with a loop as a whole single entity whose position with respect to other components of a stroke is retained rather than retaining every segment of the loop. In the following, an algorithm will be developed to extract an auxiliary representation (reduced graph) of the stroke in which the loops are reduced, represented by vertices with features, and the loop segments are deleted.

### 2.3.1. Definitions

(a) A loop is a set of vertices and links in G such that

1. Given any vertex in the loop, there is a path from that vertex to itself which visits all the other vertices in the loop only once, and which traverses all the links in the loop only once.
2. No two vertices in the loop are connected by a path which crosses the interior of the loop.

For example, in Figure 2.12, the set of vertices 1, 2, 3, 4, 5, 6, 9, 1 and the set of vertices $1,9,6,7,8,1$ constitute two loops.
(b) Two loops, in G, are adjacent if they have at least one common link, e.g., the two loops mentioned in (a) have two common links: $(1,9)$ and $(9,6)$; hence, they are adjacent.
(c) A loop set is a set of loops such that each loop is adjacent to at least one other loop in the set and is not adjacent to any other loop outside the set, e.g., the two loops mentioned in (a) and (b) constitute one loop set. If a loop is not adjacent to any other loop, then it also constitutes a loop set.

In Arabic script, a loop set can not contain more than two loops. Thus, every loop set contains either a single loop or two adjacent loops. The algorithm which reduces loops
now follows.

## Algorithm 2.3

Use: To obtain reduced graph/s of stroke/s
Input: Direct straight line approximation/s of stroke/s
Output: Reduced graph/s of strokes

## Procedure:

Step 1. Find the set of all loops in the graph, G, of the stroke.
Step 2. Find all the loop sets in G.
Step 3. Replace each loop set with a single vertex (it will be called a loop set vertex). The features of a loop set vertex are:
(a) the $x \& y$ coordinates of the mean point of the vertices of the loop set,
(b) the number of the loops in the loop set,
(c) length of the loop set which equals the sum of lengths of the links constituting the loop set, and
(d) the minimum and maximum $\mathrm{x} \& \mathrm{y}$ coordinates of the vertices which constitute the loop set.

Step 4. In each loop set, delete all the links in the set, together with any vertices which do not have links outside the loop set.

Step 5. For every loop set vertex, introduce new links connecting the loop set vertex to any vertices remaining in the loop set (i.e., to those vertices in the set which have links outside the set).

Algorithm 2.3 yields a reduced graph, $\mathrm{G}^{\prime}$, which contains no loops. If the original graph $G$ has no loops, then $G$ and $G^{\prime}$ will be the same.

### 2.3.2. Example

Algorithm 2.3 is used to reduce the graph, G, of Figure 2.12 as follows:
Step 1. Two loops are found in the graph, $G$, which are represented by the sets of vertices $1,2,3,4,5,6,9,1$ and $1,9,6,7,8,1$.

Step 2. One loop set is found which consists of the two loops found in Step 1.


Figure 2.13. The reduced graph, $\mathrm{G}^{\prime}$, of the graph, G , shown in Figure 2.12.

Step 3. The loop set is replaced with a single loop set vertex, vertex 1 in Figure 2.23.

Step 4. The links of the loop set are deleted. All vertices in the loop set, except vertex 7, are deleted since they do not have links outside the loop set.

Step 5. Vertex 7 in Figure 2.12, which corresponds to vertex 2 in Figure 2.13, is connected to loop set vertex 1 . Notice that vertex 10 in Figure 2.12 corresponds to vertex 3 in Figure 2.13.

## SUMMARY

Algorithms were presented for processing Arabic text prior to recognition. First, an algorithm was described which converts a smoothed thinned image into a straight line approximation, consisting of vertex coordinates connected by straight line segments. The algorithm incorporates heuristics which ensure a unique centre for each intersection vertex, and to reduce the likelihood of spurious tails. This algorithm produces straight line approximations which have spurious artifacts introduced by the thinning process which complicate skeleton representation and may degrade further postprocessing operations. Thus, another algorithm was developed which uses the distance transform of thinned binary images to identify spurious bifurcation points which are unavoidable when thinning algorithms are used, remove them, and recover the original ones. The obtained straight line approximations preserve the structural information of the original pattern ensuring a natural representation of it: Unlike existing approaches to the same problem (e.g., [67]), this new method can deal with complex junctions where more than two lines cross, and does not resort to geometrical properties which are prone to distortion by scanning and quantization noise.

Finally, an algorithm is suggested in to obtain reduced straight line approximations (reduced graphs) in which loops are represented by vertices with features. In this
approximation, a loop is dealt with as a whole single entity whose position, with respect to other components of a stroke, is retained rather than retaining every segment of the loop. Reduced graphs are auxiliary representations which are used in subsequent processes together with direct straight approximations to extract other useful information.

## Enforcement

## of Temporal

## Information

## OVERVIEW

In this chapter, straight line approximations of off-line Arabic strokes are converted into one-dimensional representations by algorithms which attempt to recover the original sequence of writing. The data flow diagram of Figure 3.1 illustrates the process which consists of two steps:
(a) Determining Start and End Vertices: This step accepts as input straight line approximations of strokes, see Chapter 2, and outputs the start and end vertices of strokes. The start and end vertices are determined based on heuristics, Sections. 3.1 and 3.2, two new theorems, Section 3.3, and a minimum distance path criterion, Section 3.4.
(b) Solving a Minimum Distance Path Problem, Section 3.4: The inputsto this step are straight line approximations of strokes and the start and end vertices which are determined using Step (a). The output is the vertices of the input straight line approximation remumbered according to their order of appearance in the minimum distance path. The method implements the following heuristic rule: The minimum distance path that traverses the stroke's straight line approximation from the start vertex to the end vertex has its vertices ordered in the same way as they were generated when the stroke was written.


Figure 3.1. Data flow diagram of the Enforcement of Temporal Information process.

### 3.1. DETERMINATION OF START VERTEX: HEURISTICS

Owing to stylistic variations in writing, the determination of the start and end points of an off-line image of a stroke becomes a non-trivial matter. In this section and the next one, algorithms are developed for this purpose. The vertex chosen for the start or end point is either a terminal vertex or an intersection vertex in a loop. The algorithm for start vertex determination follows after the following definition.

## Definition

The rectangle of minimum area surrounding the graph, $G$, of a stroke is called the bounding rectangle.

|  | Algorithm 3.1 |
| :--- | :--- |
| Use: | To find the start vertex of every input stroke |
| Input: | For each input stroke: |
|  | 1. $\quad$ Direct straight line approximation, $G$ |
|  | 2. $\quad$ Reduced graph, $G^{\prime}$ |

Output: Start vertices of input strokes

## Procedure:

Step 1. Locate the set of terminal vertices in $\mathrm{G}^{\prime}$. The nearest terminal vertex, t , to the Upper-Right Corner, URC, of the bounding rectangle is found. If a terminal vertex, v , represents a loop set, then the distance used to select v is the distance from URC to the nearest of the vertices in $G$ that constitute the loop set.

## Step 2.

(a) If t does not represent a loop set, it is considered as the start vertex.
(b) If $t$ represents a single loop having no intersection vertex (i.e., the stroke consists of a single loop), then the nearest vertex to the Lower-Left Corner, LLC, of the bounding rectangle is the start vertex.
(c) If t represents a single loop containing an intersection vertex, v , then:

1. If v has only one link to a vertex outside the loop set, then v is the start vertex.
2. Otherwise, find the set of terminal vertices, $U$, in the original graph, $G$, such that for each $u \in U$ there is a path from $u$ to $v$ with the following property: none of the intermediate vertices in the path between $u$ and $v$ is an intersection vertex. In Arabic writing, the number of vertices in the set $U$ does not exceed two. If $U$ is empty, the stroke is discarded since the algorithm concludes that the structure of the stroke is not consistent with the characteristics of Arabic script, otherwise:
i. If $U$ has one vertex, $u$, then $u$ is the start vertex.
ii. If $U$ has two vertices and $v$ lies to the left of the loop set vertex, then the lower terminal vertex in U is chosen as the start vertex.
iii. If $U$ has two vertices and $v$ lies to the right of the loop set vertex, then the right-most terminal vertex in U is chosen as the start vertex.
(d) If $t$ represents a loop set of two adjacent loops then the uppermost intersection vertex, $v$, in $G$, which is part of the loop set, is the start vertex. In Arabic script, $v$ will have exactly three links in G.

(a)

(b)

Figure 3.2. (a) Bounding rectangle of a graph, G, with 1 being the start vertex, and (b) the reduced graph, $\mathrm{G}^{\prime}$, with vertices 1 and 2 representing a loop set of two adjacent loops and a loop set of a single loop, respectively.

### 3.1.1. Examples

The steps of the Algorithm 3.1 are illustrated by examples in Figures 3.2 to 3.7.
Step 1. Figure 3.2(a) shows the graph, G, of a stroke having two connected characters: $H a$ and $W a w$, from right to left. $H a$ has two adjacent loops, while Waw has one loop. The two loops of $H a$ have two common links; therefore they constitute a loop set of two adjacent loops. The third loop is not adjacent to any other loop; therefore it represents a second loop set consisting of one loop. The two loop sets are represented by vertices 1 and 2 in the reduced graph, $G^{\prime}$, Figure 3.2(b). The minimum bounding rectangle for the graph $G$ of Figure 3.2(a) is shown in dashed lines. In Figure 3.2(b), there are three terminal vertices: 1,2 , and 3 . Since 1 represents a loop set, its distance, $d$, to URC is that shown in Figure 3.2(a). It is clear that 1 is the nearest terminal vertex to URC.


Figure 3.3. A graph of a stroke without loops: vertex 1 is the start vertex.


Figure 3.4. Arabic numeral " $\circ$ ", or isolated $H a$ character: vertex 1 is a start vertex.

## Step 2.

(a) An example of this case is vertex 1 of Figure 3.3.
(b) This case occurs for Arabic numeral " 0 " and an isolated Ha character (both are just closed loops) as shown in Figure 3.4. For this special case, the start vertex can be any one of the vertices in the closed straight line approximation. However, in most of the handwritings the start vertex is the nearest one to LLC which is our choice in this algorithm. Thus, in Figure 3.4, vertex 1, whose distance, d, to LLC is minimum, is the start vertex.
(c) 1. This case is shown in Figure 3:5 with 1 as the start vertex.
2. In Figure 3.6(a), the loop when converted into a loop set vertex (vertex 4 in Figure 3.6(b)), becomes the nearest terminal vertex to URC. The set U consists only of vertex 1, which is chosen as the start vertex. In Figure 3.7, the intersection vertex, 2 , lies to the right of the loop set vertex in $\mathrm{G}^{\prime}$ (not shown). The set $U$ consists of two vertices, 1 and 3 . Since 1 is the right-most vertex, it is selected as the start vertex.


Figure 3.5. A stroke that starts with a loop. The start vertex is 1.

(a)

(b)

Figure 3.6. (a) A stroke that starts with a loop (1 is the start vertex), and (b) the reduced graph, $\mathrm{G}^{\prime}$.


Figure 3.7. A stroke with a loop closest to URC, and two terminal vertices in the set $\mathrm{U}, 1$ is the start vertex.


Figure 3.8. Vertices 1 and 2 are incorrectly selected as the start and end vertices, respectively. The converse is true.


Figure 3.9. The segment between vertices 1 and 2 is a spurious elongation artifact. Vertices 1 and 7 are incorrectly selected as the start and end vertices, respectively. The true start and end vertices are 3 and 5, respectively.
(d) An example of this case is vertex 1 of Figure 3.2(a).

### 3.1.2. Causes of Failure

Algorithm 3.1 can fail to identify the actual start vertex for the following reasons:
(a) There can be a terminal vertex, $t$, which is nearer than the actual start vertex to URC. In such a case, $t$ will be wrongly selected as the start vertex. In Figure 3.8, the actual start vertex is 2 although its distance, $\mathrm{d}_{2}$, to URC is larger than the distance, $d_{1}$, of vertex 1 . Thus, vertex 1 is incorrectly selected as the start vertex.
(b) A spurious elongation artifact may produce a terminal vertex with the same property as in the previous case. In Figure 3.9, the segment between 1 and 2 is a spurious artifact that results in vertex 1 being the nearest to URC. Thus 1 is incorrectly selected as the start vertex instead of 3 , the actual start vertex.
(c) A secondary stroke may touch a main stroke with the former having a terminal


Figure 3.10. The segment between vertices 1 and 2 is a secondary stroke that touches the main stroke. Thus, vertex 1 is incorrectly selected as the start vertex while the true start vertex is 3 .
vertex as in case (a). In Figure 3.10, the segment between vertices 1 and 2 is a secondary stroke that touches the main stroke. This causes vertex 1 to be incorrectly selected as the start vertex instead of vertex 3 , the true start vertex.

### 3.2. DETERMINATION OF END VERTEX: HEURISTICS

A formal description of an algorithm to determine the end vertex of a stroke now follows.

## Algorithm 3.2

Use: To find the end vertex of every input stroke
Input: For each input stroke:

1. Direct straight line approximation, $G$
2. Reduced graph, $\mathbf{G}^{\prime}$

Output: End vertices of input strokes

## Procedure:

Step 1. Locate the set of terminal vertices, T, in $\mathrm{G}^{\prime}$. Arrange these vertices in ascending order according to their Euclidean distance from LLC of the bounding rectangle. If a terminal vertex, $v$, represents a loop set, then the distance of $v$ is the distance from LLC to the nearest of the vertices in $G$ that constitute the loop set. Let $t$ be the first vertex in T, i.e., t is the nearest terminal vertex to LLC.

Step 2. Let s represent the start vertex that was found using Algorithm 3.1.
(a) If $t$ does not represent a loop set then:

1. If $G$ consists of a single vertex, $t$ is considered as the end vertex. This vertex is also selected as the start vertex in Algorithm 3.1, Step 2(a).
2. If $G$ consists of more than one vertex and $t \neq s$, then $t$ is considered as the end vertex.
3. If $G$ consists of more than one vertex and $t=s$, then let $t$ be the next vertex in T and repeat Step 2.
(b) If $t$ represents a single loop having no intersection vertex (i.e., the stroke consists of a single loop), then the nearest vertex to LLC is the end vertex. This vertex is also selected as the start vertex in Algorithm 3.1, Step 2(b).
(c) If $t$ represents a single loop containing an intersection vertex, $v$, then:
4. If $v$ has only one link to a vertex outside the loop set, then:
i. If $v \neq s$ then $v$ is the end vertex.
ii. If $v=s$, then let $t$ be the next vertex in $T$ and repeat Step 2.
5. Otherwise, find the set of terminal vertices, $U$, the same as in Step 2(c)2 of Algorithm 3.1. If $U$ is empty, the stroke is discarded since the algorithm concludes that the structure of the stroke is not consistent with the characteristics of Arabic script, otherwise, let $u$ be the nearest of the vertices in $U$ to LLC:
i. If $u \neq s$, then $u$ is the end vertex.
ii. If $u=s$ and $U$ has two vertices, then the other vertex in $U$ is the end vertex.
iii. If $u=s$ and $U$ has one vertex, then let $t$ be the next vertex in $T$ and repeat Step 2.
(d). If $t$ represents a loop set of two adjacent loops then let $t$ be the next vertex in $T$ and repeat Step 2 since an Arabic stroke can not end with two adjacent loops.

### 3.2.1. Examples

The steps of Algorithm 3.2 are illustrated by examples.
Step 1. In Figure 3.2, the set of terminal vertices, T, is $\{1,2,3\}$ which, when


Figure 3.11. The start and end vertices are 1 and 2, respectively.


Figure 3.12. The start and end vertices are 1 and 2, respectively.
arranged in ascending order according to their Euclidean distance from LLC of the bounding rectangle, becomes $\{3,2,1\}$. Thus, the nearest terminal vertex to LLC is vertex 3.

Step 2.
(a) 1. The case in which $G$ consists of a single vertex occurs in dot strokes. It is obvious that this single vertex works as a start and end vertex.
2. In Figure 3.7, $s=1, t=3 \neq s$. Thus, 3 is the end vertex.
3. In Figure 3.11, since $t=s=1$, the next vertex, 2, in $T$ is selected and Step 2 is repeated. This time, $t=2 \neq \mathrm{s}$, resulting in 2 being the end vertex.
(b) An example of this case is vertex 1 of Figure 3.4. This vertex is a start and end vertex.
(c) 1. i. The loop in Figure 3.12, when converted to a loop set vertex (not shown in the figure), becomes the nearest terminal vertex to LLC. Here, $s=1$, $v=2 \neq s$. Thus, 2 is the end vertex.
ii. In Figure 3.13, since $v=s=1$, the next vertex, 2, in $T$ is selected and


Figure 3.13. The start and end vertices are 1 and 2, respectively.


Figure 3.14. The start and end vertices are 1 and 3, respectively.


Figure 3.15. The start and end vertices are 1 and 3, respectively.

Step 2 is repeated. This time, $\mathrm{t}=2 \neq \mathrm{s}$, resulting in 2 being the end vertex.
2. i. In Figure 3.14, $s=1, u=3 \neq s$. Thus, 3 is the end vertex.
ii. In Figure 3.15, $u=s=1$ and $U$ has two vertices, 1 and 3. Thus, the other vertex, 3 , in $U$ is the end vertex.
iii. In Figure 3.16, $u=s=1$ and $U$ has one vertex. Thus, the next vertex in


Figure 3.16. The start and end vertices are 1 and 3, respectively.


Figure 3.17. The start and end vertices are 1 and 2, respectively.

T is selected and Step 2 is repeated. This time, t is a loop set vertex which represents the uppermost single loop. Here, $v=3 \neq s$ and has only one link to a vertex outside the loop set. Thus, 3 is the end vertex.
(d) If the loop set of Figure 3.17, which consists of two loops, is reduced to a loop set vertex it becomes the nearest terminal vertex to LLC. Since Arabic strokes can not end with two adjacent loops, another terminal vertex, 2, is selected and Step 2 is repeated. Since $s=1, t=2 \neq s, 2$ is the end vertex.

### 3.2.2. Causes of Failure

Algorithm 3.2 can fail to identify the actual end vertex for the following reasons:
(a) There can be a terminal vertex, $t$, which is nearer than the actual end vertex to LLC. In such a case, $t$ will be wrongly selected as the end vertex. In Figure 3.18, the actual end vertex is 2 although its distance to LLC is larger than that of vertex 1.


Figure 3.18. Vertex 1 is incorrectly selected as the end vertex. The true end vertex is 2.


Figure 3.19. The segment between vertices 1 and 2 is a spurious elongation artifact. Vertex 1 is incorrectly selected as the end vertex. The true end vertex is 3.

Thus, vertex 1 is incorrectly selected as the end vertex.
(b) A spurious elongation artifact may produce a terminal vertex with the same property as in the previous case. In Figure 3.19, the segment between 1 and 2 is a spurious artifact that results in vertex 1 being the nearest to LLC. Thus, 1 is incorrectly selected as the end vertex instead of 3 , the actual end vertex.
(c) It may happen that Algorithm 3.1 selects the true end vertex as a start vertex. Thus, Algorithm 3.2 is forced to select another incorrect vertex to work as an end vertex. This is clear from Figure 3.8 in which 1 is selected as the start vertex. Thus, the algorithm is incorrectly forced to select 2 as the end vertex.

The earlier algorithms for the determination of start and end vertices can be developed to be more sophisticated so that almost all possible conditions that may arise in Arabic handwriting can be dealt with successfully.

### 3.3. DETERMINATION OF START AND END VERTICES: THEOREMS

In this section, we present two theorems to determine start and vertices of an Arabic stroke which are based on our observations of Arabic handwriting. First, some terms are
defined in the following subsection.

### 3.3.1. Definitions

(a) SNSENR Property of Cursive Arabic Handwriting: In correct cursive Arabic writing, any segment having the start or end vertex as one of its vertices is not retraced. This property will be called the SNSENR property which is the abbreviation of: a Segment which is Near a Start or End vertex is Not Retraced.
(b) The degree of a vertex $\mathrm{v} \in \mathrm{V}$ is the number of links in L associated with v .
(c) A noiseless graph, $G$, is a graph that is written such that the SNSENR property is not violated, and has no spurious tails.
(d) A multi-vertex graph, $G$, is a graph that has more than one vertex.
(e) An open graph, G, is a graph that does not consist of loops only, i.e., it contains at least one link which is not one of the links of a loop in G.

### 3.3.2. Theorem 1

In a noiseless, multi-vertex, and open graph, G, of a stroke the start and end vertices have odd degrees.

Proof See Figure 3.20.
(a) When a stroke is to be drawn, first the pen is laid on paper to plot the start vertex, $\mathrm{v}_{0}$. Since G must be a multi-vertex graph, some vertex other than the start vertex must be plotted. Thus, the start vertex is left by drawing a segment, $\mathrm{s}_{1}$. Drawing $\mathrm{s}_{1}$ generates another vertex which makes the degree of $\mathrm{v}_{0}$ odd. If it is continued to plot anything without returning to $v_{0}$, then $v_{0}$ preserves its degree (i.e., it remains odd). Now, if $v_{0}$ is to be returned to then this must be done via another segment, say $\mathbf{s}_{\mathbf{m}}$, otherwise the SNSENR property will be violated. Returning to $\mathrm{v}_{\mathbf{0}}$ via $\mathrm{s}_{\mathrm{m}}$ makes its degree even. To preserve the SNSENR property and exclude the cases for which $G$ only consists of loops, $v_{0}$ has to be left again via a new segment, say $s_{m+1}$, which retains the previous odd degree of $v_{0}$. Thus, it turns out to be true that whenever $v_{0}$ is left it will have an odd degree. This completes the proof of the first half of the theorem that concerns the start vertex.
(b) If the vertex $v_{z}$ is intended to be the end vertex of a stroke, then the first time it is


Figure 3.20. A graph used to assist the proofs of Theorems 1 and 2.


#### Abstract

drawn it will have an odd degree since it has to be arrived at via a segment, say $\mathrm{s}_{\mathrm{n}}$. If the writing subsequently leaves $v_{z}$ via a segment, say $s_{n+1}$, then the degree of $v_{z}$ becomes even. But since $\mathrm{v}_{\mathbf{z}}$ has to be returned to by some segment, $\mathrm{s}_{\mathrm{q}}$, its degree again becomes odd. Thus, it is true that whenever $\mathrm{v}_{\mathrm{z}}$ is arrived at it has an odd degree and whenever it is departed from it has an even degree. Sooner or later, $\mathbf{v}_{\mathbf{z}}$ will be returned to and the writing process will terminate leaving $\mathrm{v}_{\mathbf{z}}$ with an odd degree. Of course, this also excludes the cases where G is a vertex. This completes the proof of the theorem.


### 3.3.3. Theorem 2

If any of the start and end vertices of a stroke's graph, G, is an intersection vertex then it must be one of the vertices of a loop in G. Alternatively, an intersection vertex that is not one of the vertices of a loop in $G$ can not be a start or an end vertex.

Proof According to the SNSENR property of Arabic handwriting, if a start or an end vertex has to be returned to then the only way to do that is via another segment other than the firstly plotted one. Remember that in cursive writing the pen is not lifted from start to end which means that returning to the start or end vertex creates a loop whose one of its vertices is the start or end vertex. This start or end vertex is already an intersection one if it has been previously visited or it becomes an intersection by being departed from if it is the first time to be returned to. Thus, if an intersection is either a start or end vertex, it must be in a loop and this completes the proof of the theorem.

The importance of Theorem 1 is that not all of the vertices of a stroke's graph, G,
are candidates for start or end vertices. The start-end pair of vertices exists in the set of those vertices with odd degree. Theorem 2 reduces the set of candidate start and end vertices to those of degree 1 , together with vertices of odd degree which exist in loops. For example, in Figure 3.20 the vertices from $v_{3}$ to $v_{6}$ can not be start or end vertices. Later, a method will be developed to determine the start-end pair of vertices.

### 3.4. ENFORCEMENT OF TEMPORAL INFORMATION

Any stroke has a start vertex, $v_{s}$, an end vertex, $v_{e}$, and a set of links some of which may be traced more than once at the time of writing. It can be easily observed that if the links of the stroke are traversed from $v_{\mathbf{z}}$ to $v_{c}$ then the traversing path reveals the fact that the writer usually tries to minimize the energy required to write the stroke. This energy can be directly minimized by minimizing the overall distance moved by the pen on the writing surface. Thus, the restoration of temporal information of off-line writing is based on the following heuristic rule, which is consistent with our observations of Arabic script: The minimum distance path that traverses the stroke's graph, G, starting at the start vertex and ending at the end vertex has its vertices ordered in the same way as they were generated when the stroke was written.

In graph theory, finding the minimum distance path described in the above rule is a variant of the Chinese postman's problem for the corresponding graph: If the links of the graph $G$ are weighted with positive costs, find a path which will traverse every link of $G$ at least once and for which the total cost of traversal (being the sum of $n_{j} c_{p}$ where $n_{j}$ is the number of times a link $l_{j}$ is traversed and $c_{j}$ is its cost), is minimum [74]. For a minimum distance solution, the cost of a link is equal to its length.

The author therefore proposes to enforce the temporal information by an algorithm which is a variant of a standard approach to the Chinese postman's problem, applied to the graph of the stroke. Before we proceed any further, we introduce the following definitions.

## Definitions

(a) An Eulerian path of $G$ is a path that traverses every link of $G$ once and only once.
(b) An Eulerian circuit of $G$ is an Eulerian path that starts and ends at the same vertex.

### 3.4.1. Solution of the Chinese Postman's Problem

A standard method [74] for solving the Chinese postman's problem is as follows. It can be shown that $G$ possesses an Eulerian circuit if and only if all its vertices have even degree. In the more general case, where $G$ contains a non-empty set $\mathrm{V}^{-}$of vertices of odd degree, an Eulerian circuit will not exist, and a path which traverses all the links of $G$ will inevitably traverse some links more than once. The problem then reduces to minimizing the additional cost of the links which are retraced.

It can also be shown that the number $\left|\mathrm{V}^{-}\right|$of odd-degree vertices in G is always even as follows. Of the vertices of V some vertices (say in the set $\mathrm{V}^{+}$) will have even degrees and some (in the set $\mathrm{V}^{-}=\mathrm{V}-\mathrm{V}^{+}$) will have odd degrees. Now, the sum of the degrees $d_{i}$ of all vertices $v_{i} \in V$ is equal to twice the number of links in $L$ (since each link adds unity to the degrees of its two end vertices), and is therefore an even number 2 m . Hence:

$$
\begin{equation*}
\sum_{v_{t} \in V^{V}} d_{i}=\sum_{v_{t} \in V^{+}} d_{i}+\sum_{v_{t} \in V^{-}} d_{t}=2 m \tag{3.1}
\end{equation*}
$$

and since $\sum_{v_{i} \in V^{+}} d_{i}$ is even, $\sum_{v_{i} \in V^{-}} d_{i}$ is also even, which means that since all $d_{i}$ in this last expression are odd the number $\left|\mathrm{V}^{-}\right|$of vertices of odd degree is even.

Let $M$ be a set of paths of $G\left(\mu_{i j}\right.$ say) between end vertices $v_{i}$ and $v_{j},\left(v_{i}, v_{j} \in V^{-}\right)$, so that no two paths have any end vertex the same, i.e., the paths are between disjoint pairs of vertices of $\mathrm{V}^{-}$and M constitutes a pairwise vertex matching. The number of paths $\mu_{\mathrm{ij}}$ in $M$ is $1 / 2\left|\mathrm{~V}^{-}\right|$, and since $\left|\mathrm{V}^{-}\right|$was shown to be always even, this number is always an integer, as of course, it should be. All the links forming a path $\mu_{\mathrm{ij}}$ are added into G as artificial links in parallel with the links of $G$ already there. (In the first instance this means that all links of $G$ forming $\mu_{\mathrm{ij}}$ are now doubled.) This is done for every path $\mu_{\mathrm{ij}} \in \mathrm{M}$ and the resulting graph is called $\mathrm{G}^{-}(\mathrm{M})$. Since some links of $G$ may appear in more than one path, $\mu_{\mathrm{ij}}$, some links of $\mathrm{G}^{-}(\mathrm{M})$ may (after all the paths $\mu_{\mathrm{ij}}$ have been added in) be in triplicate, quadruplicate, etc. All the vertices of $G^{-}(M)$ have even degree, and $G^{-}(M)$ therefore possesses an Eulerian circuit.

For the solution of the Chinese postman's problem, it is necessary to find that set of paths $\mathrm{M}^{*}$ (matching the vertices of odd degree) which produces the least additional cost
(or distance in the case of writing). The least distance path traversing G would then correspond to an Eulerian circuit of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$, whose length is equal to the sum of the lengths of the links of $G$ plus the sum of the lengths of the links in the paths of $\mathrm{M}^{*}$.

It should be noted that since a minimum matching $\mathrm{M}^{*}$ is used, no two paths $\mu_{\mathrm{ij}}$ and $\mu_{\mathrm{pq}}$ in such a matching (of, say, $\mathrm{v}_{\mathrm{i}}$ to $\mathrm{v}_{\mathrm{j}}$ and $\mathrm{v}_{\mathrm{p}}$ to $\mathrm{v}_{\mathrm{q}}$ ) can now have any link in common. This means that the graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ does not have more than two links in parallel between any two vertices, i.e., the optimal path never traverses any link of $G$ more than twice. This is consistent with Arabic writing since a stroke segment may be retraced only once (i.e., it is drawn twice).

An Eulerian circuit can be traced through $\mathbf{G}^{-}\left(\mathrm{M}^{*}\right)$ by selecting an arbitrary start vertex and adopting Fleury's Rule for finding an Eulerian path: Each time a link has been traversed, erase it. Never traverse a link if the removal of this link will divide the remaining graph into two connected components (excluding isolated vertices).

At any stage, in the construction of the Eulerian circuit of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$, there is often more than one choice for the next link to be traced which satisfies Fleury's Rule. Thus, there will generally be a large number of distinct Eulerian circuits of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$, all of which correspond to minimum-distance paths through $\mathbf{G}$.

Floyd's Algorithm for shortest paths between all pairs of vertices of a graph can be used to check whether the graph is divided into two components [74]. If the removal of a link produces at least two vertices of shortest path the length of which equals infinity, then this means that the graph will be divided into two connected components. In that case, the link is not removed and another vertex is searched for possible traversal.

As an illustration of the above method, consider the graph $G$ depicted in Figure 3.21(a). This graph has 12 links, all of which are assigned unit cost. Here, $\mathrm{V}^{\vee}$ consists of the four vertices of odd degree, numbered $1,2,3$ and 7 .

An arbitrary pairwise matching of the vertices in $\mathrm{V}^{-}$is $\mathrm{M}=\left\{\mu_{13}=1-2-3, \mu_{27}=2-5-\right.$ 6-7). When the paths in $M$ are added to the graph $G$, the graph $G^{-(M)}$ which has 17 links, is obtained, as shown in Figure 3.21(b). An Eulerian circuit of $\mathrm{G}^{-}(\mathrm{M})$, obtained by starting at vertex 1 and applying Fleury's Rule, is 1-2-3-4-8-7-6-9-10-7-6-5-2-3-6-5-2-1, and has a total cost of 17 .

The minimal distance matching of the vertices in $\mathrm{V}^{-}$is $\mathrm{M}^{*}=\left\{\mu_{12}=1-2, \mu_{37}=3-6-\right.$


Figure 3.21. Solution of the Chinese postman's problem: (a) the original graph, G, (b) the extended graph $\mathrm{G}^{-}(\mathrm{M})$, from an arbitrary matching $M$, and (c) the extended graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$, from the minimum cost matching $\mathrm{M}^{*}$.

7\}. The extended graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ has 15 links and is shown in Figure 3.21(c). An Eulerian circuit of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ is 1-2-3-4-8-7-6-9-10-7-6-3-6-5-2-1, and the corresponding path in G is therefore a solution of the Chinese postman's problem, having a total traversal cost of 15.

### 3.4.2. Algorithm to Enforce Temporal Information

For the purpose of enforcing temporal information in a graph extracted from an Arabic handwritten stroke, it is necessary to find an Eulerian path (rather than circuit), beginning at the start vertex and finishing at a distinct end vertex. If an Eulerian path exists it means that the graph can be drawn on paper by following this path and without lifting the pen from the paper. A path consistent with Arabic handwriting is found by the algorithm introduced below, which differs from the solution of the Chinese postman's problem in two important respects:
(a) The start and end vertices are removed from the set $\mathrm{V}^{-}$prior to constructing the graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$. Three options will be suggested to determine the start and end vertices. A start or end vertex has odd degree (either 1 or 3 ).
(b) Ambiguities in the application of Fleury's Rule are resolved by adopting heuristics consistent with the properties of Arabic handwriting.

The algorithm which follows is applicable in all cases, with the exception of one special case, where the stroke consists of a single closed loop. In that case, the loop is traced in a clockwise direction, beginning and ending at the start vertex.

## Algorithm 3.4

Use: To enforce temporal information on an off-line handwritten Arabic stroke
Input: For each input stroke:

1. Direct straight line approximation, $\mathbf{G}$
2. Reduced graph, $\mathbf{G}^{\prime}$

Output: Temporal information of stroke/s, i.e., an optimum Eulerian path traversing the graph, $G$, from the start vertex to the end vertex, for every input stroke

## Procedure:

Step 1. Determine the length of all links in $G$. Find the set, $\mathrm{V}^{-}$, of vertices in G ,
having odd degree. Using a shortest path algorithm [74], compute the $\left|\mathrm{V}^{-}\right|$by $\left|\mathrm{V}^{-}\right|$matrix $D=\left[d_{i j}\right]$ where $d_{i j}$ is the sum of the link lengths in the least-distance path from a vertex $v_{i}$ $\in \mathrm{V}^{-}$to another vertex $\mathrm{v}_{\mathrm{j}} \in \mathrm{V}^{-}$.

Step 2. In Arabic writing the first drawn line segment after the start vertex is not retraced. The same applies for the last drawn line segment just before the end vertex is reached (see SNSENR property). Thus, the start and end vertices are excluded from the matching, to be found, by removing them from the set $\mathrm{V}^{-}$.To determine the start and end vertices, three options are suggested:

## Option 1.

(a) Algorithms 3.1 and 3.2 are used to determine the start and end vertices, respectively.
(b) Exclude the start and end vertices from the set $\mathrm{V}^{-}$.
(c) Find that pairwise matching $\mathrm{M}^{*}$ of the vertices in $\mathrm{V}^{-}$which produces the least distance according to the distance matrix D . This can be done efficiently by the minimum matching algorithm described in reference [74].

## Option 2.

(a) Use Algorithm 3.1 to determine the start vertex.
(b) The end vertex is found as follows:

1. Remove the start vertex from the set $\mathrm{V}^{-}$.
2. Each remaining $v_{i} \in \mathbf{V}^{-}$is a candidate to be an end vertex. Find that pairwise matching $\mathrm{M}^{i}$ of the vertices in $\mathrm{V}^{-}$, excluding $\mathrm{v}_{\mathrm{i}}$, which produces the least distance according to the distance matrix $D$. This is repeated for each $v_{i} \in V^{-}$.
3. Determine which of the matchings $M^{i}$ has the least additional cost, and denote that matching by $\mathrm{M}^{*}$. The vertex $\mathrm{v}^{*} \in \mathrm{~V}^{-}$, which is excluded from $\mathrm{V}^{-}$to obtain $\mathrm{M}^{*}$, is considered as the end vertex.

Option 3. The start and end vertices are determined as follows:
(a) A pair of vertices $\mathbf{v}_{\mathbf{i}}, \mathbf{v}_{\mathbf{j}} \in \mathrm{V}^{-}, \mathrm{i} \neq \mathrm{j}$, such that Theorems 1 and 2 are satisfied is a candidate for a start-end pair. Find that pairwise matching $M^{i j}$ of the vertices in $\mathrm{V}^{-}$, excluding $v_{i}$ and $v_{j}$, which produces the least distance according to the distance matrix $D$. This is repeated for every pair of vertices $v_{i}, v_{j} \in V^{-}, i \neq j$.
(b) The pair of vertices $v_{i}{ }^{*}, v_{j}{ }^{*} \in V^{-}$, such that Theorems 1 and 2 are satisfied, which
when excluded a minimum matching $\mathrm{M}^{*}=\min \mathrm{M}^{j \mathrm{j}}$ is obtained, is considered as the start-end pair of vertices and hence $\mathrm{v}_{\mathrm{i}}{ }^{*}$ and $\mathrm{v}_{\mathrm{j}}{ }^{*}$ are removed from $\mathrm{V}^{-}$.
(c) The nearest vertex of $v_{i}^{*}$ and $v_{j}^{*}$ to URC is the start vertex and the other one is the end vertex.

Step 3. In the matching, $M^{*}$, if a vertex $v_{i}$ is matched to another vertex $v_{j}$, identify the least distance path $\mu_{i j}$ (from $v_{i}$ to $v_{j}$ ) corresponding to the distance $d_{i j}$ of Step 1. Insert artificial links in $G$ corresponding to links in $\mu_{\mathrm{ij}}$ and repeat for all other paths in the matching $\mathrm{M}^{*}$ to obtain the graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$.

Step 4. Obtain an Eulerian path of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$, and hence the corresponding optimum path traversing the original graph $G$, by beginning at the start vertex and applying Fleury's Rule in parallel with two additional rules:

Rule 1. If none of the links of $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ has been traversed (we are at the start vertex $v_{s}$ ), then the degree of $v_{s}$ is either 1 or 3 , and there are three possibilities:
(a) There is only one vertex to be visited after the start vertex provided that Fleury's Rule is satisfied, in which case the traversal proceeds to that vertex.
(b) The number of vertices which may be visited after the start vertex, provided that Fleury's Rule is satisfied, is two, say $v_{i}$ and $v_{j}$ (i.e., the start vertex is in a loop). Here there is also a third vertex, $\mathrm{v}_{\mathrm{b}}$ connected to $\mathrm{v}_{\mathrm{s}} . \mathrm{v}_{\mathrm{i}}$ will be the next vertex to be visited if the directional change between the two links $\left(v_{j}, v_{s}\right)$ and $\left(v_{v}, v_{k}\right)$ is less than the directional change between the two links $\left(v_{j}, v_{z}\right)$ and $\left(v_{k}, v_{k}\right)$, otherwise, $v_{j}$ will be the next vertex to be visited.
(c) The number of vertices which may be visited after the start vertex, provided that Fleury's Rule is satisfied, is three. The uppermost vertex of these three vertices is the next vertex to be visited.

The reason for which the number of the above cases is limited to three simply is: In Arabic script writing, the cases in which the number of vertices that can be visited; after the start vertex, is greater than one occur in those characters starting with loops. If the character starts with one loop, then this number equals two. For characters with two adjacent loops, it is three. Finally, there is no single character that may start with more than two adjacent loops; therefore only the above three cases are obtained.

Rule 2. If the traversal is not at its start (i.e., we have left $v_{\mathrm{s}}$ ), let $\mathrm{v}_{\mathrm{j}}$ be the current
visited vertex and $v_{i}$ be the previously visited vertex just before $\mathbf{v}_{\mathbf{j}}$. Let $\mathrm{V}^{\mathrm{n}}$ be the set of vertices which may be visited after $\mathbf{v}_{\mathbf{j}}$ such that Fleury's Rule is satisfied. The vertex $\mathbf{v}_{\mathbf{k}} \in$ $V^{n}$ that produces the minimum change between the directions of the two links $\left(v_{i}, v_{j}\right)$ and $\left(\mathrm{v}_{\mathrm{j}}, \mathrm{v}_{\mathrm{k}}\right)$ is selected as the next vertex to be visited.

The stroke's graph, G, and the temporal information extracted using the above algorithm constitute a one-dimensional representation of a stroke that was originally plotted in a plane.

The above algorithm can also be used to enforce temporal information on reduced graphs. However, since in a reduced graph there are no loops, a start or end vertex is always a terminal vertex.

### 3.4.3. Example

The various steps of Algorithm 3.4 are illustrated in the following detailed example. Figure 3.22(a) shows the image of a stroke with two loops, and Figure 3.22(b) shows a straight line approximation of the same stroke with initial numbering of vertices.

Step 1. The set of vertices of odd degree is $\mathrm{V}^{-}=\{1,2,3,4,5,6,8,9\}$. Since $\left|\mathrm{V}^{-}\right|$is 8 , a shortest path algorithm obtains an $8 \times 8$ matrix $\mathrm{D}=\left[\mathrm{d}_{\mathrm{ij}}\right]$, where $\mathrm{d}_{\mathrm{ij}}$ is the length of the least-distance path from a vertex $v_{i} \in V^{-}$to another vertex $v_{j} \in V^{-}$.

Step 2. Any of the three options of this step yields vertices 9 and 1 as start and end vertices, respectively. These two vertices are excluded from $\mathrm{V}^{-}$.

Step 3. The matching $M^{*}$ consists of: 2 to 6 with the shortest path 2-6, 3 to 8 with the shortest path $3-8$, and 4 to 5 with the shortest path 4-5. Artificial links are inserted in $G$ corresponding to the links of the paths 2-6, 3-8, and 4-5 to obtain the graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ which is shown in Figure 3.22(c), where artificial links are dotted lines. Notice that the graph $\mathrm{G}^{-}\left(\mathrm{M}^{*}\right)$ does not have more than two links in parallel between any two vertices. This is consistent with Arabic writing since a stroke segment may be retraced only once (i.e., it is drawn twice).

Step 4. An Eulerian path of this graph, which is an optimum path traversing the original graph G of Figure 3.22(b), is constructed, by applying Fleury's Rule taking into account the previously mentioned rules for Arabic handwriting, as follows. The vertices


Figure 3.22. (a) An Arabic stroke, (b) its graph, $G$, (c) $G^{-}\left(M^{*}\right)$, (d) the graph, $G$, after finding the optimum Eulerian path and renumbering of vertices, and (e) the reduced graph, $\mathrm{G}^{\prime}$, with the optimum Eulerian path shown.

20 and 24 are candidates to be visited after the start vertex， 9 ．Since the directional change between the two links $(24,9)$ and $(9,19)$ is less than that between the two links $(20,9)$ and $(9,19), 20$ is selected as the next vertex to be visited．When vertex 7 is reached， vertices 5 and 15 are candidates for the next visited vertex．Since the directional change achieved by going through the sequence $17,7,5$ is less than that for the sequence 17,7 ， 15 ，vertex 5 is selected as the next visited vertex．The minimum Euler path is 9－20－21－22－ 23－24－9－19－8－3－8－18－17－7－5－4－5－16－15－7－14－13－6－2－6－12－11－10－1．The vertices of the minimum path are renumbered according to the order by which they appear in the minimum path．Thus，the path becomes：1－2－3－4－5－6－1－7－8－9－8－10－11－12－13－14－13－15－ 16－12－17－18－19－20－19－21－22－23－24．The graph of the stroke after renumbering of vertices is shown in Figure 3．22（d）．

To clarify how reduced graphs are manipulated，the graph of Figure 3．22（b）is reduced to the graph shown in Figure 3．22（e）．The two loops are converted to loop set vertices， 1 and 9 in Figure 3．22（e）．The determined start and end vertices are 1 and 19， respectively．Traversing the reduced graph from start to end results in its vertices being numbered as shown in the figure．

## 3．4．4．Testing

Algorithm 3.4 was tested against two different data sets．The first data set represents samples of isolated handwritten characters provided by 20 subjects，see Appendix B， Figures B． 2 to B．21．The other data set represents samples of cursive Arabic handwriting provided by 20 subjects，see Appendix D，Figures D． 1 to D．20．A description of how these data sets were acquired is given in the following subsection．

## 3．4．4．1．Data Acquisition

For the data set of isolated handwritten characters，the character set under test is shown in Figure 3.23 which is，from right to left：Arabic numerals＂•＂，＂＂，＂r＂，＂r＂，＂ぇ＂， ＂○＂，＂マ＂，＂マ＂，＂А＂，and＂१＂，Arabic secondary strokes and special characters：Hamza， Madda，Shadda，slash，minus sign，right and left parenthesis，comma，and Damma．

Twenty subjects were asked to write one line of each character of the set under study．There was no restriction on pen type，ink type，or ink colour．Subjects were asked

## 

Figure 3.23. Set of stroke classes used to test Algorithm 3.4, from right to left: Arabic numerals zero to nine, Hamza, Madda, Shadda, slash, minus sign, right parenthesis, left parenthesis, comma, and Damma.
to avoid generating blobs as possible as they can since the algorithm was not designed to deal with such a phenomenon. Each subject was given one A4 size blank sheet on top of another guiding sheet, see Appendix B, Figure B.1, with $20 \times 20$ empty squares. The reasons for using squares are:
(a) With a sheet without squares as guidelines, the writing of a subject may deteriorates as he proceeds from one end of the line to the other end since he knows that the same character is to be written. Thus, he may not mind to move and re-adjust his wrist as the pen moves. Including such guiding squares helps to achieve such wrist re-adjustment, hence, produces more natural writing.
(b) Without squares, characters of the same line may touch each other which raises some kind of a segmentation problem which is not addressed in the current research. The squares of the guiding sheet eliminate this problem.
(c) If the same size of data is acquired by every subject then more accurate evaluation of the algorithm is obtained. This is achieved by giving each subject a blank sheet on top of another sheet with a fixed number of empty squares per sheet.
(d) If the same number of samples is acquired for each character then more accurate evaluation of the algorithm is obtained. This is achieved by giving each subject a black sheet on top of another sheet with a fixed number of empty squares per line. Reproductions of the images of isolated handwritten characters used to test Algorithm 3.4 are shown in Appendix B, Figures B. 2 to B.21. In these figures, notice that there is an extra character, the plus sign, in which the pen has to be lifted once during writing which produces two start vertices and two end vertices. Unfortunately, the current version of Algorithm does not deal with such cases. Thus, the samples of the plus sign were excluded from the test. They were only used in the testing stage of the IACR system, Part Two.

The data set of cursive handwriting which was used to test Algorithm 3.4 was not restricted to a limited list of words, i.e., an unlimited vocabulary was used. A subject can pick any book, story, journal, etc., and select the parts he wishes to write. Also, there was no restriction on pen type, ink type, or ink colour. The subjects were asked to fill an A4 size blank sheet of undiacriticized handwriting. Subjects were asked to use a common type of chirography of Arabic handwriting which is called Arreka chirography. Subjects who don't master Arreka chirography were asked to simply follow the rule: write every subword as single piece without lifting the pen except for secondary strokes (dots, dashes, etc.). They were asked to avoid generating blobs as possible as they can since the algorithm was not designed to deal with such a phenomenon. Unfortunately, most of the subjects were not conforming to these instructions. Mostly, a mixture of Arreka chirography and another common chirography in Arabic handwriting called Annaskh were used. In Annaskh chirography, the pen can be lifted more than once to write the main stroke of a subword. The subjects were asked to write from 10 to 15 lines per page which span the entire length of the A4 size sheet. Twenty unnormalized handwritten A4 size pages written by 20 subjects were collected. The whole data provided by the 20 subjects were used to test Algorithm 3.4 without discarding any proportion whether the subject follows Arreka chirography or not. Reproductions of the images used to test the algorithm are shown in Appendix D, Figures D. 1 to D.20.

Images of the two data sets were captured using an HP ScanJet scanner. The resolution used was 300 dots per inch in both the horizontal and vertical directions. The reason for selecting this value of resolution is based on our observation that undersampled pictures, e.g., less than 300 dpi, may create disconnected images for very thin strokes which produces multi-component straight line approximations for such strokes. This is not accepted by the algorithm since it does not have the capability to handle disconnected strokes.

### 3.4.4.2. Results

First, we have to mention the following:
(a) When writing a stroke, for most of Arabic writers, the generated vertices share a common vertex ordering which is a result of following common writing rules. Thus,
to determine when the actual temporal information was recovered we depend on these common rules and visual inspection of each processed sample without the need to know the writer's conditions.
(b) In the case of cursive handwriting, sometimes the pen has to be lifted, after finishing the main stroke, to write secondary strokes that complete the subword. Usually, a secondary stroke does not touch the main stroke. A secondary stroke which touches the main stroke will be treated as part of the main stroke.

The options in Algorithm 3.4 were used to generate three different methods. The first method implements Option 1 of Step 2 of the algorithm [44]. In the second and last methods, Options 2 and 3, of Step 2, are used, respectively. The key differences between the three methods are:
(a) In Methods 1 and 2, the start vertex is determined according to certain heuristic rules which depend on the geometrical properties of the stroke. In Method 3, the start vertex is determined according to a minimum-distance criterion.
(b) In Method 1, the end vertex is also determined according to certain heuristic rules which depend on the geometrical properties of the stroke. In Methods 2 and 3, the end vertex is determined according to a minimum-distance criterion.
The results of the three methods for the two data sets are detailed below.
(a) Results of the Isolated Character Data Set

As mentioned in Section 3.4.4.1, the plus sign was excluded from the test. Also, some other garbage samples were excluded. Thus, a total 7563 samples was used to test Algorithm 3.4. The overall success rates of restoration of temporal information for Methods 1,2 , and 3 were $90.9 \%, 90.7 \%$, and $91.9 \%$, respectively. Table 3.1 details the results of the three methods for the 20 subjects. In this table, notice that Methods 1 and 2 have the same success rate for the start vertex, as it should be, since they both use Algorithm 3.1 to determine the start vertex. From this table, it is clear that Method 3 exhibits the highest overall success rate for restoration of temporal information, $91.9 \%$. The reason for this is that in Method 3 the start and end vertices are concurrently determined according to a minimumdistance criterion. In other words, a candidate start-end pair of vertices is only selected as the final start-end pair of vertices if the corresponding path has the

Table 3.1. Comparison between the performance of three methods for restoring temporal information, where the data used to obtain this table are the isolated sample characters shown in Appendix B, Figures B. 2 to B. 21 .
S: Subject \#, Temp. info.: Temporal information, Tot: Total.

| S | No. of strokes | Success Rate, \% |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Method 1 |  |  | Method 2 |  |  | Method 3 |  |  |
|  |  | Start vertex | $\begin{gathered} \text { End } \\ \text { vertex } \end{gathered}$ | Temp. info. | Start vertex | End vertex | Temp. info. | Start vertex | End | Temp. info. |
| 1 | 380 | 89.2 | 90.0 | 88.2 | 89.2 | 88.2 | 86.8 | 90.5 | 88.2 | 88.2 |
| 2 | 380 | 93.9 | 94.5 | 92.6 | 93.9 | 91.6 | 91.1 | 94.5 | 91.6 | 90.8 |
| 3 | 380 | 91.6 | 93.2 | 90.8 | 91.6 | 91.6 | 90.5 | 93.2 | 91.6 | 91.1 |
| 4 | 380 | 92.9 | 93.4 | 91.8 | 92.9 | 93.4 | 92.1 | 93.4 | 93.4 | 92.6 |
| 5 | 379 | 92.9 | 93.4 | 92.6 | 92.9 | 93.4 | 92.3 | 93.4 | 93.4 | 93.1 |
| 6 | 380 | 93.4 | 94.7 | 92.6 | 93.4 | 94.5 | 92.9 | 94.5 | 94.5 | 94.2 |
| 7 | 380 | 89.5 | 90.5 | 88.4 | 89.5 | 90.0 | 88.7 | 90.5 | 90.0 | 89.5 |
| 8 | 373 | 90.9 | 93.8 | 89.5 | 90.9 | 93.8 | 89.8 | 90.9 | 93.8 | 89.8 |
| 9 | 379 | 94.2 | 94.5 | 93.7 | 94.2 | 94.5 | 93.9 | 94.2 | 94.5 | 93.4 |
| 10 | 375 | 94.1 | 94.1 | 93.3 | 94.1 | 94.7 | 94.1 | 94.4 | 94.7 | 94.1 |
| 11 | 377 | 90.5 | 90.7 | 89.9 | 90.5 | 90.7 | 89.9 | 90.7 | 90.7 | 90.2 |
| 12 | 376 | 94.9 | 94.9 | 94.7 | 94.9 | 94.9 | 94.7 | 94.9 | 94.9 | 94.7 |
| 13 | 375 | 93.9 | 94.1 | 93.6 | 93.9 | 93.9 | 93.3 | 94.1 | 93.9 | 93.3 |
| 14 | 379 | 92.1 | 94.7 | 91.3 | 92.1 | 93.7 | 91.6 | 94.7 | 94.7 | 94.5 |
| 15 | 380 | 91.3 | 94.7 | 90.5 | 91.3 | 93.7 | 90.5 | 94.7 | 94.7 | 94.2 |
| 16 | 379 | 89.4 | 92.9 | 88.7 | 89.4 | 89.7 | 88.7 | 92.6 | 92.1 | 90.8 |
| 17. | 379 | 90.2 | 93.4 | 89.2 | 90.2 | 93.1 | 89.4 | 93.1 | 93.1 | 92.3 |
| 18 | . 372 | 89.8 | 92.7 | 88.7 | 89.8 | 92.7 | 89.0 | 92.5 | 92.7 | 91.9 |
| 19 | 380 | 92.9 | 94.7 | 91.8 | 92.9 | 90.0 | 88.9 | 94.2 | 90.0 | 88.9 |
| 20 | 380 | 86.8 | 90.8 | 86.1 | 86.8 | 90.8 | 86.1 | 90.5 | 90.8 | 89.5 |
| Tot | 7563 | 91.7 | . 93.3 | 90.9 | 91.7 | 92.4 | 90.7 | 93.1 | 92.7 | 91.9 |

minimum length such that Fleury's rule and other additional rules are not violated.

## (b) Results of the Cursive Handwriting Data Set

Here, only the main strokes of the collected sample pages were used to test Algorithm 3.4 since a secondary stroke is usually a dot, dash, or Hamza character which were included in the test of the isolated characters, see (a) above. Thus, a total of 4272 main strokes was used to test Algorithm 3.4. The overall success rates of restoration of temporal information for Methods 1,2 , and 3 were $89.3 \%, 90.2 \%$, and $91.8 \%$, respectively. Table 3.2 details the results of the three methods for the 20 subjects. Notice that Methods 1 and 2 have the same success rate for the start vertex, as it should be, since they both use Algorithm 3.1 to determine the start vertex. Although Method 1 has high success rates for the determination of the start and end vertices, it has the lowest overall success rate of restoring the temporal information. The reason is that, in Method 1, the start and end vertices are determined independent of each other depending on geometric properties only. The result is that it happens more frequently that the start vertex, end vertex, or both are incorrectly determined yielding incorrect temporal information. In Method 2, finding the start vertex according to geometrical properties helps in the finding the end vertex, which matches that start vertex, according to a minimum-distance criterion. Thus, there is some kind of dependency of the determination of the end vertex on the determination of the start vertex with the observation that: if the start vertex is successfully determined, then the chance to successfully determine the end vertex increases. This dependency increases in Method 3 since both the start and end vertices are determined according to a minimum-distance criterion, i.e., they are determined in parallel, which yields the highest overall success rate of restoring the temporal information.

Due to its superiority, although it is slight, Method 3 will be used in the IACR and CASR systems to enforce temporal information.

Table 3.2. Comparison between the performance of three methods for restoring temporal information, where the data set used to obtain this table is the cursive handwritten cursive pages shown in Appendix D, Figures D. 2 to D.21.
S: Subject \#, Temp. info.: Temporal Information, Tot: Total.

| S | No. of main strokes | Success Rate, \% |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Method 1 |  |  | Method 2 |  |  | Method 3 |  |  |
|  |  | Start vertex | End vertex | Temp. info. | Start vertex | End vertex | Temp. | Start vertex | End vertex | Temp. info. |
| 1 | 187 | 94.1 | 96.3 | 89.8 | 94.1 | 96.8 | 90.4 | 95.2 | 98.4 | 92.0 |
| 2 | 219 | 96.8 | 94.5 | 88.1 | 96.8 | 95.0 | 89.0 | 95.9 | 95.0 | 91.8 |
| 3 | 184 | 97.8 | 98.4 | 92.4 | 97.8 | 98.9 | 92.4 | 97.8 | 98.4 | 91.8 |
| 4 | 227 | 93.4 | 92.5 | 89.9 | 93.4 | 96.5 | 90.3 | 94.7 | 95.6 | 91.6 |
| 5 | 192 | 96.4 | 95.3 | 88.0 | 96.4 | 95.8 | 87.5 | 96.9 | 96.4 | 89.1 |
| 6 | 198 | 92.9 | 96.5 | 89.4 | 92.9 | 97.5 | 90.9 | 96.5 | 96.0 | 91.9 |
| 7 | 256 | 93.8 | 93.4 | 90.2 | 93.8 | 94.5 | 91.0 | 95.3 | 95.7 | 93.4 |
| 8 | 160 | 94.4 | 91.3 | 88.8 | 94.4 | 92.5 | 90.0 | 96.3 | 95.6 | 93.1 |
| 9 | 226 | 93.8 | 94.7 | 88.9 | 93.8 | 95.6 | 91.2 | 93.8 | 94.7 | 90.7 |
| 10 | 219 | 96.3 | 95.0 | 88.1 | 96.3 | 94.5 | 87.7 | 97.7 | 95.0 | 89.5 |
| 11 | 206 | 96.1 | 96.1 | 89.8 | 96.1 | 94.7 | 91.3 | 97.1 | 96.1 | 93.7 |
| 12 | 208 | 93.8 | 96.2 | 88.9 | 93.8 | 97.1 | 90.9 | 95.7 | 96.6 | 90.9 |
| 13 | 215 | 94.0 | 94.9 | 90.7 | 94.0 | 96.3 | 90.7 | 94.9 | 96.3 | 90.2 |
| 14 | 197 | 95.4 | 97.0 | 89.3 | 95.4 | 96.4 | 90.9 | 94.9 | 95.4 | 90.9 |
| 15 | 190 | 94.7 | 90.5 | 88.4 | 94.7 | 92.6 | 89.5 | 95.3 | 94.2 | 92.1 |
| 16 | 215 | 93.5 | 96.7 | 89.8 | 93.5 | 97.2 | 89.3 | 95.3 | 96.7 | 90.7 |
| 17 | 235 | 93.6 | 94.5 | 89.4 | 93.6 | 95.7 | 91.5 | 95.7 | 97.9 | 94.5 |
| 18 | 233 | 94.8 | 94.4 | 88.4 | 94.8 | 94.8 | 90.1 | 96.6 | 94.4 | 91.8 |
| 19 | 249 | 94.0 | 93.6 | 91.2 | 94.0 | 94.4 | 91.6 | 96.8 | 97.2 | 94.4 |
| 20 | 256 | 93.0 | 92.6 | 86.3 | 93.0 | 93.8 | 87.9 | 94.5 | 95.7 | 91.0 |
| Tot | 4272 | 94.6 | 94.7 | 89.3 | 94.6 | 95.5 | 90.2 | 95.8 | 96.0 | 91.8 |



Figure 3.24. Vertex 1 is incorrectly selected as the start vertex. The true start vertex is 2 .

### 3.4.5. Causes of Failure

Examples where Algorithm 3.4 fails to enforce the correct temporal information are illustrated in Figures 3.8, 3.9, 3.24, and 3.25. The causes of failure of Algorithms 3.1 and 3.2 are also causes of failure of Algorithm 3.4. In addition, the failure can be due to the incorrect determination of the start and / or end vertices which is a result of a new way by which these vertices are determined:
(a) Although the start-end pair is determined correctly, of these two vertices the vertex which is nearer to URC is not the actual start vertex (in fact it is the end vertex while the other one is the start vertex). In Figure 3.8, the actual start vertex is 2 although its distance, $d_{2}$, to URC is larger than the distance, $d_{1}$, of vertex 1 . Thus, 1 is incorrectly selected as the start vertex.
(b) There can be a terminal vertex, $v$, other than the actual start vertex, that produces the minimum distance matching. In Figure 3.24, the algorithm selects vertex 1 as the start vertex although the actual one is vertex 2.
(c) A different vertex from the true end vertex may produce a matching with less cost, in which case the former will be selected as the end vertex. In Figure 3.9, the actual end vertex is 5. But since the distance from 4 to 5 is less than that from 6 to 7, it is less costly to retrace the segment from 4 to 5 than to retrace the segment from 6 to 7. The removal of vertex 7 from $\mathrm{V}^{-}$gives the minimum distance matching, and 7 is therefore wrongly selected as the end vertex.

(a)

(b)

Figure 3.25. (a) A stroke's image, and (b) its graph, G, with a spurious segment between vertices 1 and 2.

Moreover, when the start and end vertices are located correctly, the minimum distance criterion may still lead to incorrect temporal information. This is often a consequence of errors introduced at the thinning stage:
(d) A bifurcation artifact due to thinning, see Section 2.1.3, can introduce a spurious segment at a junction in a loop. This segment may change the order by which the links are traversed producing temporal information that is partially incorrect (the temporal information of the loop is reversed). For example, the straight line approximation of the stoke of Figure 3.25(a) may look like Figure 3.25(b), where the segment between 1 and 2 is a spurious bifurcation artifact. When vertex 2 is reached, Algorithm 3.4 finds that going to vertex 3 results in a smaller directional change than going to 4 . Thus, 3 is selected as the next vertex to be visited, although the correct choice is 4. In Section 2.2, a spurious segment could be identified and replaced with a single vertex. The replacement vertex is approximately equivalent to the original crossing point in the sense that the topological properties of the thinned shape are preserved. Thus, the temporal information of a loop is correctly restored in most of the cases. In some cases, a spurious segment can not be
identified due to small angle of intersection between lines, see Section 2.2.3. Thus, a bifurcation artifact still remains a cause of error in Algorithm 3.4. It is worth mentioning here that the problem of spurious segments does not appear in reduced graphs in which loops, represented as vertices, are treated as whole entities.
(e) Temporal information is lost when thinning a small loop that has become a blob due to blotting. Blotting occurs in both printed and handwritten text. The reasons for this phenomenon are excess ink, type of paper, the way of motion of pen head, overwriting, etc. For a human reader, these blobs can be easily recognized as loops from the context. However, such instances are difficult to deal with and cause problems in automatic recognition systems.

One trial was performed by the author to restore such lost loops. The idea depends on removing points whose distance, based on a distance transform, from the image boundary of the stroke exceeds a calculated threshold. This method was tried on the handwritings of two writers. They supplied 65 subwords containing 159 blobs. Loops were recovered successfully with a rate of $83.6 \%$. The main reason behind this low rate of success is that the line width and blob size vary even in the same subword which makes it difficult to recover all loops using a single threshold. It was also noticed that spurious holes may be introduced into the image especially in thick parts that are not blobs and at points of intersection that look like blobs. If these holes are small then they can be removed by finally smoothing the image. However, if the holes are big then it is difficult to remove them by a conventional smoothing process. It was thought that a more accurate method is needed to calculate the distance transform since the city-block distance transform was initially used. As reported in [73], the city-block distance transform is the worst approximation of the Euclidean distance; the maximal difference between the two may exceed $55 \%$. Thus a chamfer $3 / 4$ transformation was tried since the maximum distance error does not exceed $8.1 \%$, as reported in [73]. Nevertheless, almost the same results were obtained. The method was also tried on printed text. Unfortunately, the same results of handwritten text were obtained for the same reasons mentioned above. The generation of new spurious loops is the main problem that could not be eliminated. Spurious loops may degrade the performance of a recognition system. Thus, it is
concluded that there should be no preprocessing stage to recover lost loops due to blotting.

## SUMMARY

In this chapter, algorithms were presented to process Arabic text prior to recognition. In Sections 3.1 and 3.2, heuristic algorithms were presented to determine the start and end vertices of an off-line image of Arabic stroke. Two novel theorems, which restrict candidate start and end vertices, were presented in Section 3.3. Theorem 1 reduces the set of vertices which are candidates for start or end vertices to those vertices with odd degree. Moreover, Theorem 2 reduces the set of candidate start and end vertices to those of degree 1 , together with vertices of odd degree which exist in loops.

In Section 3.4, straight line approximations of off-line Arabic strokes were converted into one-dimensional representations by a novel algorithm, which aims to recover the original sequence of writing according to a minimum-distance criterion. The algorithm was developed from a standard solution of the Chinese postman's problem applied to the graph of the stroke. Options in this algorithm were used to generate three different methods to enforce temporal information. The results of Method 3, in which the determination of the start and end vertices is based on Theorems 1 and 2, were superior. Illustrative examples were presented to explain the algorithms. The main cases for which the algorithms could not produce proper results were mentioned.

The algorithms presented in this chapter can deal with many of the situations that may arise in Arabic handwriting, however, they can be developed so that more possible conditions, which might not be faced during testing, are dealt with. The author considers that the resulting ordering of the stroke segments is a suitable preprocessing method for subsequent Arabic handwriting recognition algorithms.

## Part Two

## Isolated

Arabic
Character

## Recognition

System<br>(IACR)

## OVERVIEW

In this part, an Isolated Arabic Character Recognition System (IACR) is presented. Figure II.I shows a data flow diagram, of the system. One important point here is that Stroke Learning comes after Stroke Recognition! Actually this arrangement is preferred to us because our philosophy, which was also highlighted in Chapter 1, lies in: "What is this ${ }_{\tau}$ ? If you know then you earn, othervise come to learn." This means that trying to recognize comes first. If the system fails then learning follows. Our new contribution is represented by the filled processes / rounded rectangles in Figure II.1, i.e., Straight Line Approximation, Enforcement of Temporal Information, Stroke Segmentation, Stroke Recognition, and Stroke Learning. Straight Line Approximation and Enforcement of Temporal Information were introduced in Chapters 2 and 3, respectively. The next three chapters address the three other processes: Stroke Segmentation, Stroke Recognition, and Stroke Learning. The data flow diagram of the IACR system consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten stroke is captured using a scanner..
(b) Smoothing: The acquired binary image of the stroke is smoothed. A suitable smoothing algorithm can be found in [57].
(c) Thinning: The smoothed binary image of the stroke is thinned using the "Safe Point Thinning Algorithm," or SPTA [66].
(d) Straight Line Approximation, Chapter 2: which accepts a smoothed thinned binary image of the stroke and produces two representations of the stroke: a direct straight line approximation and a loopless straight line approximation (reduced graph).
(e) Enforcement of Temporal Information, Chapter 3: Here temporal information of the stroke are extracted from the reduced graph of the stroke.
(f) Stroke Segmentation, Chapter 4: The reduced graph and temporal information are used to segment the stroke into small units, called primitives. The reduced graph and segmented primitives are necessary inputs for the subsequent two processes, i.e., Stroke Recognition and Stroke Learning.
(g) Stroke Recognition, Chapter 5: which receives a fuzzy sequential machine, which


Figure II.1. Data flow diagram of the IACR system.
is a representation of the learned strokes, a reduced graph, and primitives of the stroke. It outputs recognition results indicating whether the stroke belongs to a certain class or it could not be recognized. If a stroke could not be recognized, then its acceptance information, represented as a tree data structure, is fed to the Stroke Learning process.
(h) Stroke Learning, Chapter 6: This process gets as inputs the fuzzy sequential machine which was used in recognition but failed to recognize the stroke, a reduced graph of the stroke, primitives of the stroke, and the acceptance tree which is passed by the Stroke Recognition process. It outputs a new fuzzy sequential machine which can recognize the input stroke and strokes of the old machine and variants of these strokes.

A hierarchical structural chart, which corresponds to the data flow diagram of Figure II.1, is shown in Figure II.2. In the hierarchical structural chart, we would like to make the following points clear:
(a) Again, our new contribution is represented by the filled modules / rectangles, i.e., Straight Line Approximation, Enforcement of Temporal Information, Stroke Segmentation, Stroke Recognition, and Stroke Learning.
(b) The numbers near the head of an arrow indicate the type of the data flowing in the direction of the arrow. If there is a link with two arrows between a higher-level module, $A$, and a lower-level module, $B$, then module $A$ first passes to $B$ the data shown near the head of the arrow entering $B$, then $B$ returns to $A$ the data shown near the head of the arrow entering A. For example, the High Level Preprocessing sub-coordinator passes a thinned binary image of a stroke, data type 5, to the Straight Line Approximation module which returns to it a direct straight line approximation, data type 6, and a loopless straight line approximation (reduced graph), data type 7.
As shown in Figure II.2, the ISOLATED CHARACTER RECOGNITION SYSTEM (IACR) coordinator manages data exchange between the following modules / subcoordinators:
(a) Input: which returns to the IACR coordinator the data base of learned items, represented in as fuzzy sequential machine, and a binary image of the stroke to be

recognized or learned. Thus, the Input sub-coordinator calls the following two modules:

1. Input Machine: which reads the Stroke Fuzzy Sequential Machine.
2. Acquire Image: which gets a binary image of a handwritten stroke.
(b) Preprocessing: This sub-coordinator receives from the IACR coordinator a binary image of the stroke and returns to it the stroke as a sequence of small segmented units called primitives. It consists of two parts:
3. Low Level Preprocessing: which receives from the Preprocessing subcoordinator a binary image and returns to it a smoothed thinned binary image of the stroke. It calls two modules:
i. Smoothing: which receives a binary image and returns a smoothed binary image of the stroke.
ii. Thinning: which receives a smoothed binary image and returns a smoothed thinned binary image of the stroke.
4. High Level Preprocessing: which receives from the Preprocessing subcoordinator a smoothed thinned binary image and returns to it the primitives of the stroke. It calls three modules:
i. Straight Line Approximation, Chapter 2: which receives a smoothed thinned binary image and returns two representations of the stroke: a direct straight line approximation and a reduced graph.
ii. Enforcement of Temporal Information, Chapter3: which receives the two representations of the stroke and returns its temporal information.
iii. Stroke Segmentation, Chapter 4: which receives a reduced graph and temporal information and returns the segmented primitives of the stroke.
(c) Stroke Recognition, Chapter 5: This module receives from the IACR coordinator a fuzzy sequential machine which is used in recognition, a reduced graph, and the primitives of the stroke. It returns to the IACR coordinator recognition results indicating whether the stroke belongs to a certain class or it could not be recognized. If a stroke could not be recognized, then its acceptance information,
represented as a tree data structure, is returned to the IACR coordinator. This tree can be fed to the Learning module for learning.
(d) Stroke Learning, Chapter 6: This module receives from the IACR coordinator four data items: (1) the fuzzy sequential machine which was used in recognition but failed to recognize the stroke, (2) a reduced graph of the stroke, (3) primitives of the stroke, and (4) the acceptance tree which was returned by the Stroke Recognition module. It returns to the IACR coordinator a new fuzzy sequential machine which can recognize the input stroke and strokes of the old machine and variants of these strokes.
(e) Output: which receives from the IACR coordinator recognition results and a new fuzzy sequential machine. Thus, the Output sub-coordinator calls the following two modules:
5. Output Machine: which outputs the Stroke Fuzzy Sequential Machine which results from learning.
6. Output Results: which outputs results indicating whether the stroke belongs to a certain class or it could not be recognized.

Experimental results and performance of the $I A C R$ system are reported in Chapter 7.

## Tempo-Structural

## Representation



## OVERVIEW

In this chapter, we suggest a structural representation of a handwritten character. Not only the suggested representation is structural but it is also temporal since it carries temporal information. Hence, what so-called a tempo-structural representation of a stroke will be derived from its reduced graph, $G^{\prime}$, and its temporal information, which consists of:
(a) small segmented parts, called primitives, which are temporally ordered, where a primitive can be a vertex, a loop set vertex, or a straight line segment. Primitives have implicit relationships between them. Figure 4.1 shows a data flow diagram of the segmentation process.
(b) features of primitives,
(c) implicit and explicit relationships between primitives, and
(d) global features of the whole stroke.

Global features of the stroke and explicit relationships between primitives are extracted at the time of recognition or learning.

### 4.1. STROKE SEGMENTATION

In this section, an algorithm is presented to segment a stroke into ordered small parts, called primitives, which have features, where a primitive can be:
(a) a vertex representing either an isolated vertex or an intersection vertex,


Figure 4.1. Data flow diagram of the Stroke Segmentation process in the IACR system.
(b) a loop set vertex, or
(c) a link (i.e., straight line segment).

## Definitions

(a) A link, $\left(v_{\mathrm{i}}, v_{\mathrm{j}}\right)$, in the reduced graph, $\mathrm{G}^{\prime}$, of a stroke, is an original link if both $\mathrm{v}_{\mathrm{i}}$ and $\mathrm{v}_{\mathrm{j}}$ do not represent loop sets. Otherwise, $\left(\mathrm{v}_{\mathrm{i}}, \mathrm{v}_{\mathrm{j}}\right)$ is an artificial link.
(b) The total length, L , of a stroke is the sum of lengths of all original links in $\mathrm{G}^{\prime}$ plus the lengths of all loop sets in the stroke.
The description of the segmentation algorithm now follows.

## Algorithm 4.1

Use: To segment a stroke into a sequence of primitives
Input: 1. Reduced graph, $\mathrm{G}^{\prime}$, of the stroke
2. Temporal information of the stroke, i.e., the path, $\mu_{\mathrm{se}}$, between the start vertex $\mathrm{v}_{\mathrm{s}}$ and the end vertex $\mathrm{v}_{\mathrm{c}}$

Output: Sequence of primitives $\psi_{1}, \psi_{2}, \ldots, \psi_{m}$, where $m$ is the sequence length.

## Procedure:

Step 1. If the path $\mu_{\mathrm{se}}$ consists of a single vertex, $\mathrm{v}_{\mathrm{l}}$, vertex then:
(a) If the vertex represents a loop set, then it is represented by a loop set primitive $\psi$ $=O\left(v_{1}\right)$, where $v_{1}$ has the features of a loop set.
(b) Otherwise, it is represented by a vertex primitive $\psi=\star\left(v_{1}\right)$.

Step 2. Otherwise, Let $\left(v_{i}, v_{j}\right)$ be a link in the path $\mu_{s c}$ with $v_{i}$ visited first. The path $\mu_{s c}$ is scanned from $v_{\mathrm{s}}$ to $v_{\mathrm{c}}$ and a sequence of primitives, $\Psi_{1}, \Psi_{2}, \ldots$, which represent the stroke, is generated by applying the following steps, in order, to each link of the path:
(a) If $v_{i}$ is an intersection vertex, then a new primitive, $\psi=\star\left(v_{i}\right)$, is generated.
(b) If $v_{i}$ represents a loop set, then a new primitive, $\psi=O\left(v_{i}\right)$, is generated.
(c) If $\left(v_{i}, v_{j}\right)$ is an original link, then it is represented by the primitive $\psi=l\left(v_{i}, v_{j}\right)$.
(d) If $\left(v_{i}, v_{j}\right)$ is the last link in the path, then apply Steps (a) and (b) to $v_{j}$.

The primitives which are extracted by the above algorithm have the following features:
(a) Feature of direction, $\theta$, which applies only to link primitives, where $\theta$ is the direction angle of the link.
(b) Feature of length ratio, $1 \mathrm{r}=l / \mathrm{L}$, where $l$ is the primitive length and L is the total length of the stroke. For an isolated vertex primitive, $l r$ is set to 1 , since an isolated vertex constitutes the whole stroke's graph. For an intersection vertex primitive, lr is set to 0 , since the length of a vertex in a graph is ideally zero. The length of a link primitive equals the Euclidean distance between its two vertices. The length of a loop set primitive is the sum of lengths of links constituting the loop set.
(c) Feature of number of loops, $n$, which applies only to loop set primitives and specifies the number of loops in the loop set.

### 4.2. INHERENT PROPERTIES OF PRIMITIVE SEQUENCES

A sequence of primitives, $\Psi=\left\{\Psi_{1} ; \Psi_{2}, \ldots, \Psi_{m}\right\}$, generated using the above segmentation algorithm, has inherent properties. Some important properties, which will be used in recognition and learning, Chapters 5 and 6, are listed below.
(a) The sequence length $\mathrm{m} \geq 1$.
(b) If the sequence length, $m$, is greater than 1 , i.e. it does not represent a dot or loop set stroke, $\Psi_{1}$ or $\Psi_{m}$ can not be a vertex primitive, i.e., it can not start or end with a vertex primitive.
(c) If $m>2$ and $\Psi_{1}$ is a loop set, then $\Psi_{2}$ must be a vertex, i.e., for a sequence the
length of which is greater than 2 and starting with a loop set primitive, $\Psi_{1}, \Psi_{1}$ must be directly followed by a vertex primitive.
(d) If $m>2$ and $\Psi_{m}$ is a loop set, then $\Psi_{m-1}$ must be a vertex, i.e., for a sequence the length of which is greater than 2 and ending with a loop set primitive, $\Psi_{m}, \Psi_{m}$ must be directly preceded by a vertex primitive.
(e) If $m>4$ and $\Psi_{i}$ is a loop set, $i=3,4, \ldots$, or $m-2$, then $\Psi_{i-1}$ and $\Psi_{i+1}$ must be vertex primitives, i.e., a loop set in the middle of a sequence of primitives must preceded and followed by vertex primitives.
(f) If $m>2, \Psi_{i}$ is a vertex, $i=2,3, \ldots$, or $m-1$, and both $\Psi_{i-1}$ and $\Psi_{i+1}$ is not a loop set vertex, then the vertex which is represented by $\psi_{i}$ must appear more than once in the sequence, i.e., a vertex primitive that is not directly followed and preceded by a loop set primitive must appear more than once in the sequence.
(g) If $m>3$, then two adjacent primitives, $\psi_{i}^{\prime}$ and $\Psi_{i+1}^{\prime}$, can not both be a vertex primitive.

### 4.3. PRIMITIVE RELATIONSHIPS

There are two types of relationships between primitives:
(a) Explicit Relationships: which are defined between an intersection primitive and a loop set primitive which are connected by an artificial link. For each intersection primitive which is connected to the loop set primitive, a relationship is established by defining the following two features:

$$
\begin{align*}
& f_{1}=\left(x-x_{\min }\right) /\left(x_{\max }-x_{\min }\right),  \tag{4.1}\\
& f_{2}=\left(y-y_{\min }\right) /\left(y_{\text {max }}-y_{\text {min }}\right), \tag{4.2}
\end{align*}
$$

where $x \& y$ are the coordinates of the intersection primitive, $x_{\text {min }}, x_{\text {max }}, y_{\text {min }}$, and $y_{\text {max }}$ are the minimum and maximum $x \& y$ coordinates of the vertices constituting the loop set primitive, respectively.
(b) Implicit Relationships: which are of two types:

1. Implicit Positional Relationships: The existence of such relationships comes from the fact that the vertices in the path $\mu_{s c}$, are already ordered in a way that enforces temporal information. The same happens during primitive segmentation. For example, if a stroke is segmented into to link primitives, $\Psi_{1}$
and $\psi_{2}$, which have the features $\theta_{1}, \mathrm{lr}_{1}$, and $\theta_{2}, \mathrm{lr}_{2}$, respectively, then the position of $\Psi_{1}$ is completely defined with respect to the position of $\Psi_{2}$ since their directions, $\theta_{1}$ and $\theta_{2}$, are known and $\psi_{2}$ follows $\Psi_{1}$.
2. Implicit size relationships: For example, if two primitives of a stroke were $\psi_{1}=O\left(v_{1}\right)$ and $\psi_{2}=O\left(v_{2}\right)$, with length ratios $l r_{1}$ and $l r_{2}$, respectively, then the ratio $l r_{1} / r_{2}$ expresses the length of $\Psi_{1}$ to that of $\Psi_{2}$.

### 4.4. GLOBAL FEATURES OF STROKE

Feature selection is an important and delicate stage in pattern recognition. Features must be adequate to describe samples of patterns in the classes under study. At the same time, the number of features must be within limits in order to keep the computational requirements reasonable. In our case, handwritten character recognition, a small set of global features was developed and proved to be powerful, as will be shown in the next three chapters, in the distinction between the highly different characters.

## Definitions

(a) A vertex, in the path, $\mu_{s c}$, of the stroke, is a dominant vertex if it is either an intersection vertex, an end vertex that is not a loop set vertex, or a vertex connected to a loop set vertex.
(b) A dominant path, $\mu_{\mathrm{ij}}$, is a path such that its two end vertices, $\mathrm{v}_{\mathrm{i}}$ and $\mathrm{v}_{\mathrm{j}}$, are dominant vertices and none of the vertices of $\mu_{\mathrm{ij}}$, which lie between $v_{i}$ and $v_{j}$, is a dominant vertex.

### 4.4.1. Selected Global features

In the following, a detailed description of the global features of a stroke is presented. These features apply to strokes which consist of more than one vertex.
(a) Features of Dominant Paths (DP features)

For each dominant path, $\mu_{\mathrm{ij}}$ in the stroke, add the following feature, which is of DP type:

$$
\begin{equation*}
f=l\left(\mu_{i j}\right) / L \tag{4.3}
\end{equation*}
$$

where $l\left(\mu_{\mathrm{ij}}\right)$ is the length of the path and $L$ is the total length of the stroke.

Feature of Height to Height-plus-Width Ratio
Let H and W be the height and width of the bounding rectangle of the stroke, respectively. Define the feature:

$$
\begin{equation*}
f=H /(H+W) \tag{4.4}
\end{equation*}
$$

The motivation for setting the dominator to $\mathrm{H}+\mathrm{W}$ is to keep f always less than or equal to 1 .
(c) Features of Heights of Right and Left Ends

Some strokes have the same structure but only differ in the relative heights of both ends of the stroke. Thus, the heights of the right and left ends of a stroke are related to the height of the stroke by the following two features:

$$
\begin{align*}
& f_{1}=\left(y_{r}-y_{0}\right) / H  \tag{4.5}\\
& f_{2}=\left(y_{1}-y_{0}\right) / H \tag{4.6}
\end{align*}
$$

where $y_{r}$ and $y_{1}$ are the $y$ coordinates of the right and left ends of the stroke, respectively, and $\mathrm{y}_{0}$ and H are the minimum y coordinate and the height of the stroke, respectively.

## (d) Feature of Curvature

Some strokes look similar but differ in the degree of curvature. An appropriate measure of curvature is:

$$
\begin{equation*}
\mathrm{f}=(\mathrm{D} / \mathrm{L})^{a} \tag{4.7}
\end{equation*}
$$

where D is the Euclidean distance between the two ends of the stroke and L is the length of the stroke. A suitable value of the exponent, $\alpha$, was found to be 5 .

The last two features only apply to strokes that neither have intersections nor loop sets.

It is worth mentioning here that global features of a stroke and explicit relationships are calculated during the recognition or learning process, since these processes may suppress some parts, which are assumed to be spurious, of the original stroke producing a modified stroke with new global features and new explicit relationships.

### 4.5. EXAMPLE

Figure 4.2(a) shows the graph, G, of a stroke, Damma. After converting the loop to a vertex, the reduced graph, $\mathrm{G}^{\prime}$, of Figure 4.2(b) is obtained. By applying Algorithm


Figure 4.2. (a) A graph, G, of an Arabic stroke, Damma, and (b) its reduced graph, $\mathrm{G}^{\prime}$.
3.4 which enforces temporal information, the vertices are numbered as shown in Figure 4.2(b). Vertex 3 represents a loop set consisting of one loop. Vertices 2 and 4 are intersections, while vertices 1 and 6 are ends. The structure elements of the stroke are found as follows:
(a) Primitive Segmentation: The links $(2,3)$ and $(3,4)$ are not original links since vertex 3 represents a loop set vertex, i.e., it does not represent an original vertex in G. The stroke is segmented to the following primitives: $\Psi_{1}=l(1,2)$ with two features $\theta=207^{\circ}$ and $\operatorname{lr}=0.13, \Psi_{2}=\star(2)$ with one feature $\operatorname{lr}=0.0, \Psi_{3}=O(3)$ with two features $\mathrm{lr}=0.37$ and $\mathrm{n}=2, \psi_{4}=\star(4)$ with one feature $\mathrm{lr}=0.0, \Psi_{5}=l(4,5)$ with two features $\theta=263^{\circ}$ and $l r=0.05$, and $\Psi_{6}=l(5,6)$ with two features $\theta=$ $219^{\circ}$ and $\mathrm{Ir}=0.45$.

Explicit Relationships: Primitive $\Psi_{2}$ is an intersection that is connected to a loop set primitive, $\Psi_{3}$. This relationship is expressed by the two features: $f_{1}=(178$ $-155) /(178-155)=1.0$ and $f_{2}=(404-396) /(409-396)=0.62$. Similarly, another relationship is expressed between the two primitives $\Psi_{4}$ and $\Psi_{3}$ by setting the two features: $\mathrm{f}_{3}=(175-155) /(178-155)=0.87$ and $\mathrm{f}_{4}=(407-396) /(409$ $-396)=0.85$.
(b) Global Features of the Stroke: There are four dominant vertices: 1, 2, 4, and 6.

1. DP features: The dominant paths are: $\mu_{12}=1-2$ and $\mu_{46}=4-5-6$, where $l\left(\mu_{12}\right)=20, l\left(\mu_{46}\right)=76$. The loop set has a length $c=56$. The total length of the stroke is $L=152$. The first two global features are of DP type: $\mathrm{f}_{5}=l\left(\mu_{12}\right)$ $/ L=0.13$ and $f_{6}=l\left(\mu_{46}\right) / L=0.50$.
2. Feature of Height to Height-plus-Width Ratio: The height, H, and width, W , of the bounding rectangle of the stroke are 63 and 75 , respectively. The global feature of height to height-plus-width ratio is $f_{7}=63 /(63+75)=0.46$.
3. In this stroke, the features of heights of right and left ends and the feature of curvature are not defined since the stroke has intersection vertices or a loop.

## SUMMARY

In this chapter, a tempo-structural representation of a stroke is derived from its reduced graph and temporal information. The stroke is segmented into temporally ordered small parts constituting the stroke. These parts have relationships between them. Necessary features of the stroke and segmented parts are extracted. Some of these featüres are extracted during recognition or learning, however, they are mentioned here to show all the constituent parts of our suggested tempo-structural representation of a stroke.

## Stroke

## Recognition



## OVERVIEW

This chapter addresses the recognition component of the IACR system. Fuzzy sequential machines are defined to work as recognizers of handwritten Arabic strokes. A data flow diagram of the recognition process is shown in Figure 5.1, which consists of two subprocess:
(a) Construction of Acceptance Tree: This subprocess accepts as inputs the segmented sequience of primitives of the unknown stroke to be recognized, a reduced graph of the stroke, and a fuzzy sequential machine which is used in the recognition. The acceptance information of the primitives is stored in a tree data structure, which is output by this subprocess.
(b) Checks and Decisions: Here, the inputs are the same inputs of the previous subprocess in addition to the acceptance tree. Some calculations, based on the information in the acceptance tree, are performed to evaluate some features of the stroke. Finally, recognition results are output indicating whether the unknown stroke belongs to a certain class or it could not be recognized. If a stroke could not be recognized, then its acceptance tree is output which can be fed to a subsequent learning stage.

### 5.1. BACKGROUND

A formal method is needed to flexibly describe handwritten text. If such descriptors


Figure 5.1. Data flow diagram of the Stroke Recognition process in the IACR system.
are designed, then they may be used as acceptors (i.e recognizers) in automatic character / text recognition systems. In this chapter, it will be shown that, what we call, a fuzzy sequential machine approach is a powerful tool for isolated handwritten Arabic stroke recognition. Before we can proceed any further, a background leading to a sequential machine formulation of the problem and some concepts from fuzzy set theory have to be introduced first.

### 5.1.1. Toward a Sequential Machine Model of Handwriting

The process of writing an Arabic stroke using straight line segments starts by laying the pen tip at the writing surface; the starting state. Then, the pen is moved according to the stroke's requirements. Each time a new straight line segment is made, a new situation (state) is created. The direction of the next segment depends on the current and previous segments, i.e., the history. Finally, we end at some terminal state having the stroke written. Thus, it is believed that Arabic handwriting is better described by a certain type


Figure 5.2. $\quad$ Process of writing Arabic numeral " $r$ " which is $c$ in $q_{3}$.
of "flexible pen motion description scheme" [2,3]. For example, the process of writing Arabic numeral "r" using a series of straight line segments can be achieved as follows:
(a) Start at some initial state, $\mathrm{q}_{0}$, by laying the pen on paper.
(b) Leave $\mathrm{q}_{0}$ to a new state, $\mathrm{q}_{1}$, in which you move to the left some number of straight line segments.
(c) Now, turn down. This can be done gradually by leaving $\mathrm{q}_{1}$ to a series of new states, e.g., $\mathrm{q}_{2}$ and $\mathrm{q}_{3}$ in which we move left-down and down, respectively, or abruptly by leaving $\mathrm{q}_{1}$ to $\mathrm{q}_{3}$. Also, some number of straight line segments are created while in states $\mathrm{q}_{2}$ and $\mathrm{q}_{3}$ depending on the required final size.
The transition from state $\mathrm{q}_{0}$ to $\mathrm{q}_{1}$ can be labelled by the symbol "left" which dictates that such a transition requires the segment to be pointing to the left direction. While in state, $\mathrm{q}_{1}$, more left segments can be created without leaving $\mathrm{q}_{1}$. Thus, another transition from $\mathrm{q}_{1}$ to itself is added which is labelled with "left", also. The transitions between other states are similarly labelled. When the final size is reached at state, $q_{3}$, we have the target (output) stroke written. We do not expect any output from state $q_{0}$, since it is simply a starting state, or from $\mathrm{q}_{1}$ or $\mathrm{q}_{2}$ since they are transient states in which the intended stroke is still incomplete. The whole process is graphically displayed in Figure 5.2.

It is clear that we have obtained a system that can be characterized by a quintuple

$$
\mathrm{M}=(\mathrm{Q}, \Sigma, \mathrm{Z}, \zeta, \mathrm{~g})
$$

where $\quad Q=$ finite nonempty set of states: $q_{0}, q_{1}, q_{2}$, and $q_{3}$
$\Sigma=$ finite nonempty set of state entrance qualifiers: "left", "left-down", and "down"
$\zeta=$ next-state mapping function $\zeta: \mathrm{Q} \times \Sigma \rightarrow \mathrm{Q}: \zeta\left(\mathrm{q}_{0}\right.$, "left" $)=\mathrm{q}_{1}, \zeta\left(\mathrm{q}_{1}\right.$,


Figure 5.3. Three of many possibilities for writing Arabic numeral " r ".

$$
\begin{aligned}
& \text { "left" })=q_{1}, \zeta\left(q_{1}, \text { "left-down" }\right)=q_{2}, \zeta\left(q_{1}, " d o w n "\right)=q_{3}, \zeta\left(q_{2},\right. \text { "left- } \\
& \text { down" })=q_{2}, \zeta\left(q_{2}, \text { down" }\right)=q_{3}, \text { and } \zeta\left(q_{3}, \text { "down" }\right)=q_{3} . \\
\mathrm{Z}= & \text { finite nonempty set of output symbols: " } \mathrm{r} " \\
\mathrm{~g}= & \text { output mapping function } \mathrm{g}: \mathrm{Q} \rightarrow \mathrm{Z}: \mathrm{g}\left(\mathrm{q}_{0}\right)=\mathrm{g}\left(\mathrm{q}_{1}\right)=\mathrm{g}\left(\mathrm{q}_{2}\right)=\text { nothing, } \\
& \mathrm{g}\left(\mathrm{q}_{3}\right)="\ulcorner "
\end{aligned}
$$

Actually this agrees with the definition of one type of deterministic sequential machines, called Moore Machine [75, pages 237, 238]. By a deterministic sequential machine it is meant that for every combination of state-state entrance qualifier, $\left(q_{i}, \sigma_{i}\right)$, there is one and only one next state. In a similar way, other sequential machines can be derived to describe the writing process of other strokes. Thus, we conclude that the process of writing a stroke can be modelled as a sequential machine.

### 5.1.2. Concepts Based on Fuzzy Set Theory

Following the steps mentioned in the previous section, Arabic numeral "r" can be constructed with many possibilities, as shown in Figure 5.3. There is no need to define each direction precisely. For example, a segment to the left can be achieved by a number of straight line segments each pointing to the approximate direction: "left". We will call such directions fuzzy directions. Each of these segments can be associated by a truth value indicating how much the proposition "move to left" is true. So, labels of directions such as: right, left, up, and down can be viewed as fuzzy mapping of the closed interval [$\left.180^{\circ}, 180^{\circ}\right]$ to the closed interval [ 0,1$]$. When a segment (expressed as an angle) is tested against a fuzzy direction, the latter maps the segment to some value which is the degree to which the segment agrees with that direction.

In fuzzy set theory, there exists what so-called, fuzzy numbers [76]. It is a concept

Figure 5.4. A stroke, the structure of which is similar to Arabic numeral "r", however, it does not represent any valid Arabic stroke.
where numeric values are viewed as approximate rather than exact values. For example, if we say 1 , we do not mean exact 1 ; rather, it is a fuzzy 1 . Any arbitrary value, $u$, can be associated with a truth value telling to what degree $u$ can be considered as 1 . A fuzzy number is characterized by a possibility distribution which involves a small number of parameters which can be adjusted to fit the given distribution. Such possibility distributions are used to calculate the truth value of u. Accordingly, a fuzzy direction, e.g., $90^{\circ}$ upward, can be easily viewed as a fuzzy 90 , i.e., a fuzzy number, with a proper possibility distribution.

One more point is that in the above example it is not always true that once $q_{3}$ is entered the shape of " $r$ " is obtained. To be accepted, the resultant shape must pass some tests. In other words, the class of accepted Arabic numeral " r " has some features which must exist in any shape generated using the underlying sequential machine. These features are embedded in $\mathrm{q}_{3}$. This is necessary since two shapes may have the same structure but different global features. For example, although the stroke of Figure 5.4 has the same structure as the right-most stroke of Figure 5.3 (each consists of two link primitives, one is vertical and the other is horizontal), the former does not represent any Arabic stroke, while the latter represents Arabic numeral "r". Thus, feeding such strokes to a machine which does not have additional classifying features results in classifying them as belonging to the same class. By introducing features, a sequential machine resolves this ambiguity. For example, one feature might be that the height / width ratio of the shape must be equal to 1.0 . When we say 1.0 , it is not meant that it is exactly 1.0. Rather, we consider it as a fuzzy 1.0. Thus, we find ourselves once again talking about fuzzy numbers. It is concluded that the embedded features in such states, e.g. $q_{3}$, are fuzzy features which have possibility distributions similar to fuzzy numbers. Due, to its importance in the development of the IACR and CASR systems, possibility distributions of fuzzy numbers are explained in Section 5.1.3.

Thus, what we obtained is a sequential machine with fuzzily labelled transitions. This
machine can be used to generate many shapes of Arabic numeral " $\ulcorner$ ". Similarly, other machines can be obtained for other strokes.

Now, we want to invert the problem, i.e., given such a sequential machine, can we use it to decide whether a given stroke, represented as a sequence of straight line segments, belongs to the class for which that machine was designed? The answer is YES, but how? Simply, starting at state $q_{0}$, use the sequence of segments as an input to excite the states of the machine. If you could proceed from $q_{0}$ to a state, $q_{i}$, that produces an output, such as $\mathrm{q}_{3}$ in the previous example, such that all states' requirements are fulfilled then you can say that the stroke is of the same class as the output of $\mathrm{q}_{\mathrm{i}}$. An example of a state requirement is to have an input segment point to the fuzzy direction of that state.

### 5.1.3. Fuzzy numbers

In this research, we are interested in the following fuzzy number distributions [76]:
(a) s-numbers: As its name implies, the possibility distribution of an s-number has the shape of $s$. The equations defining an s-number, expressed as ( $p / \beta$ ), are:

$$
\begin{align*}
\pi_{x}(u) & =0, & & u \leq p-\beta \\
& =\left(2 / \beta^{2}\right)(u-p+\beta)^{2}, & & p-\beta \leq u \leq p-\beta / 2 \\
& =1-\left(2 / \beta^{2}\right)(u-p)^{2}, & & p-\beta / 2 \leq u \leq p \\
& =1, & & u \geq p
\end{align*}
$$

where $\beta$ (the bandwidth) is the length of the transition interval from $\pi_{x}=0$ to $\pi_{x}=$ 1 , and $p$ is the left peak-point (i.e. the right end point of the transition interval), as shown in Figure 5.5(a).
(b) z-numbers: A z-number is a mirror image of an s-number. Thus, the defining equations for a z-number which is expressed as ( $p \backslash \beta$ ), are:

$$
\begin{align*}
\pi_{x}(u) & =1, & & u \leq p \\
& =1-\left(2 / \beta^{2}\right)(u-p)^{2}, & & p \leq u \leq p+\beta / 2 \\
& =\left(2 / \beta^{2}\right)(u-p-\beta)^{2}, & & p+\beta / 2 \leq u \leq p+\beta \\
& =0, & & u \geq p+\beta
\end{align*}
$$

where $p$ is the right peak-point and $\beta$ is the bandwidth, as shown in Figure 5.5(b).
(c) $s / z$-numbers: An $s / z$ number has a flat-top possibility distribution, which may be regarded as the intersection of the possibility distribution of an s-number and a


Figure 5.5. $\quad$ Distributions of fuzzy numbers: (a) s-number, (b) z-number, and (c) s/znumber.
z-number, with the understanding that the left peak-point of the s-number lies to the left of the right peak-point of the z-number, see Figure 5.5(c). An $s / z$-number is represented as an ordered pair ( $p_{1} / \beta_{1} ; p_{2} \backslash \beta_{2}$ ).

We will regard fuzzy directions and fuzzy features as having $s / z$-number possibility distributions. For example, the fuzzy direction "left" has an $s / z$ possibility distribution defined in the interval $0^{\circ}$ to $360^{\circ}$. The four parameters of the distribution can be: $\mathrm{p}_{1}=165^{\circ}, \beta_{1}=30^{\circ}, \mathrm{p}_{2}=195^{\circ}$, and $\beta_{2}=30^{\circ}$. Therefore, $\pi_{\text {left }}$ $(\theta)=\left(165^{\circ} / 30^{\circ} ; 195^{\circ} \backslash 30^{\circ}\right)$. If a line segment, with a direction angle, $\theta$, is tested against this fuzzy direction, the value of $\pi_{\text {lef }}(\theta)$ gives the degree to which this segment is pointing to the left. An example on fuzzy features might be: the height to width ratio equals "fuzzy 1 ", where $\pi_{\text {fuzz } 1}(u)=(0.8 / 0.2 ; 1.2 \backslash 0.2)$.

### 5.2. THE FUZZY SEQUENTIAL MACHINE

In this section, the complete definition of a fuzzy sequential machine, which is capable of recognizing a set of handwritten strokes, will be given. To be able to define our machine, two questions have to be answered:
(a) Are there other types of state entrance qualifiers?

The fuzzy sequential machine, to be defined, works on a sequence of segmented primitives of the stroke. Recall that such a sequence can contain up to three types of primitives: vertex primitive, loop set primitive, and link primitive. So far, the only state entrance qualifiers which have been investigated are fuzzy directions, which act on link primitives. Thus, to generalize the model such that other strokes can be recognized, additional state entrance qualifiers must be defined. Thus, a state entrance qualifier can be one of the following:

1. "a vertex referred to as $\star$,
2. . a loop set which is referred to by $O(n)$ where $n$ is the number of loops in the loop set.
3. a fuzzy direction expressed as an $s / z$-number $\left(p_{1} / \beta_{1} ; p_{2} \backslash \dot{\beta}_{2}\right)$, where $p_{1}, \beta_{1}$, $p_{2}$, and $\beta_{2}$ are as defined earlier.

The reader should not mix between the notation used to describe a sequence of primitives of a stroke and the notation used to describe state entrance qualifiers. To

Table 5.1. Notations used to describe primitives of a stroke and state entrance qualifiers.

| Tempo-structural representation |  | State entrance qualifier |  |
| :--- | :---: | :--- | :---: |
| Primitive type | Notation | Qualifier type | Notation |
| vertex | $\star\left(v_{i}\right)$ | vertex | $\star$ |
| loop set | $O\left(v_{i}\right)$ | loop set | $O(n)$ |
| link | $l\left(v_{i}, v_{j}\right)$ | fuzzy direction | $\left(p_{1} / \beta_{1} ; p_{2} \backslash \beta_{2}\right)$ |

remove any confusion that might occur, see Table 5.1.
(b) What fuzzy features are needed for further discrimination between stroke classes?

The required fuzzy features are:

1. Fuzzy features which are of the same types as explicit relationships between primitives of a stroke, see Section 4.3.
2. Fuzzy features which are of the same types as global features of a stroke, see Section 4.4.

Although they have similar types, there is an intrinsic difference between global features of a given stroke and explicit relationships between primitives on one side and fuzzy features on the other side. Global features of a given stroke and explicit relationships have exact values, i.e., they are deterministic or they are not fuzzy. Fuzzy features are fuzzy numbers having $\mathrm{s} / \mathrm{z}$ possibility distributions.

Now, having the background material being laid, our complete definition of a fuzzy sequential machine can presented.

### 5.2.1. Definition of Fuzzy Sequential Machine

A deterministic Fuzzy sequential Machine which recognizes a set of stroke classes, where a stroke is supplied as a reduced graph and a sequence of primitives, is a system that is characterized by the 5 -tuple $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, where
(a) C is the set of stroke classes that are recognized by this machine.
(b) Q is a finite set of states $\mathrm{q}_{0}, \mathrm{q}_{1}, \ldots, \mathrm{q}_{0}$ being the starting state.
(c) $\Sigma$ is a finite nonempty set of state entrance qualifiers $\sigma_{1}, \sigma_{2}, \ldots$ Each state, $q_{i}$,
except $\mathrm{q}_{0}$, has one and only one entrance qualifier, $\sigma_{i}$.
(d) $\zeta$ is the next-state mapping function $\zeta: \mathbf{Q} \times \Sigma \rightarrow \mathbf{Q}$.
(e) $T \subset Q$ is a finite nonempty set of terminal states. A terminal state $q_{i} \in T$ recognizes the stroke classes $c_{i 1}, c_{i 2}, \ldots \in C$, each of which has $n_{i}$ fuzzy features of the same type, i.e., more than one stroke class can be recognized in one terminal state with the same number and types of fuzzy features. We mean by same types of fuzzy features that the first fuzzy feature of $c_{i 1}$ has the same type of the first fuzzy feature of $\mathrm{c}_{\mathrm{i} 2}, \mathrm{c}_{\mathrm{i} 3}, \ldots$.. The same applies to the remaining fuzzy features.

Of course, there is a recognition algorithm which explains how a stroke is handled by the fuzzy sequential machine to decide whether it belongs to a stroke class $\mathrm{c} \in \mathrm{C}$. The details of the recognition algorithm are presented in Section 5.3.

Based on the background material given in Section 5.1, the differences between this definition and that of a conventional sequential machine should be clear, however, we summarize these differences in the following two points:
(a) A state entrance qualifier, $\sigma$, of a fuzzy sequential machine can be a fuzzy direction allowing for a high degree of variability of accepted shapes. In conventional sequential machines, a state entrance qualifier is exact which is usually binary, i.e., it is either 1 or 0 . Exact qualifiers restrict the set of accepted shapes to those which exactly satisfy the qualifiers. Real-life patters have fuzzy rather than exact properties which makes handling them by systems with fuzzy capabilities (fuzzy sequential machines, in our case) more appropriate.
(b) A fuzzy sequential machine has terminal states in which fuzzy features of accepted shapes are inserted. An output class, $\mathrm{c} \in \mathrm{C}$, is not generated unless its features are satisfied. In conventional sequential machines, states do not contain features. Once a state is reached, outputs are generated which is inadequate to differentiate between patterns having the same structure.

A fuzzy sequential machine will be pictorially described by a transition diagram. The automatic generation of fuzzy sequential machines is a learning process which is described in the next chapter.


Figure 5.6. A deterministic fuzzy sequential machine which can recognize two stroke classes: Arabic Damma and Arabic numeral " r ".

### 5.2.2. Example

Figure 5.6 shows a transition diagram of a deterministic fuzzy sequential machine $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, where
(a) $\mathrm{C}=\{$ Arabic Damma, Arabic numeral " $\ulcorner$ "\},
(b) $\mathrm{Q}=\left\{\mathrm{q}_{\mathrm{i}}, \mathrm{i}=0,1, \ldots, 11\right\}$,
(c) $\Sigma=\left\{\sigma_{\mathfrak{p}} \mathrm{i}=1,2, \ldots, 11\right\}, \sigma_{1}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \cup 30^{\circ}\right), \sigma_{2}=\star, \sigma_{3}=O(1), \sigma_{4}=\star$, $\sigma_{5}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right), \sigma_{6}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right), \sigma_{7}=\left(195^{\circ} / 30^{\circ}\right.$; $\left.225^{\circ} \backslash 30^{\circ}\right), \sigma_{8}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right), \sigma_{9}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right), \sigma_{10}=$ $\left(285^{\circ} / 30^{\circ} ; 315^{\circ} \backslash 30^{\circ}\right)$, and $\sigma_{1 i}=\left(315^{\circ} / 30^{\circ} ; 345^{\circ} \backslash 30^{\circ}\right)$.
(d) $\zeta$ is the next-state mapping function $\zeta: Q \times \Sigma \rightarrow \mathbf{Q}$. This mapping can be easily read from the figure,
(e) $\mathrm{T}=\left\{\mathrm{q}_{7}, \mathrm{q}_{11}\right\}$, where

1. A Damma class is embedded in the terminal state $\mathrm{q}_{7}$. One stroke, belonging to this class is shown in Figure 5.7. Strokes belonging to a Damma class can have one loop set primitive connected to two intersection primitives. For the first intersection primitive, there are two fuzzy features: $F_{1}=(0.8 / 0.2 ; 1.0 \backslash$


Figure 5.7. (a) A graph, G, of an Arabic Damma stroke and (b) its reduced graph, $\mathrm{G}^{\prime}$.
$0.2)$ and $\mathrm{F}_{2}=(0.6 / 0.2 ; 0.8 \backslash 0.2)$ which relate the $\mathrm{x} \& \mathrm{y}$ coordinates of the intersection primitive to the $x \& y$ coordinates of centre of gravity of the loop set primitive, respectively. Similarly the other intersection primitive has the features $\mathrm{F}_{3}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$ and $\mathrm{F}_{4}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$. The next two fuzzy features are global and of DP type since a stroke belonging to Damma class has two dominant paths, see Section 4.4. These two fuzzy features are $F_{5}=(0.0 / 0.2 ; 0.2 \backslash 0.2)$ and $F_{6}=(0.4 / 0.2 ; 0.6 \backslash 0.2)$. There is a global fuzzy feature of height to height-plus-width ratio: $\mathrm{F}_{7}=(0.4 / 0.2 ; 0.6 \backslash 0.2)$. For Damma class, the global fuzzy features of heights of right and left ends and the fuzzy feature of curvature are not defined since a stroke belonging to this class has intersection or loop set primitives. Thus, we end up with seven fuzzy features associating this stroke class.
2. A class of Arabic numeral " $\left\ulcorner\right.$ " is embedded in the terminal state $\mathrm{q}_{11}$. Strokes belonging to this class are shown in Figure 5.3. This class has five fuzzy features. The first fuzzy feature is $F_{1}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$ which is of $D P$ type since this class has one dominant path. A fuzzy feature of height to height-plus-width ratio is $\mathrm{F}_{2}=(0.4 / 0.2 ; 0.6 \backslash 0.2)$. Two other fuzzy features are $\mathrm{F}_{3}=(0.0 / 0.0 ; 0.2 \backslash 0.2)$ and $\mathrm{F}_{4}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$, which relate the heights of the right and left ends to the total height of a stroke belonging to this class. The last fuzzy feature is $\mathrm{F}_{5}=(0.0 / 0.0 ; 0.2 \backslash 0.2)$ which is a measure of curvature of such strokes.

### 5.3. RECOGNITION ALGORITHM

A fuzzy sequential machine has a recognition algorithm, $R$, which details how the recognition process is achieved. The description of such algorithm is necessary which is a result of the existence of fuzzy state entrance qualifiers, i.e., fuzzy directions, and fuzzy features. In conventional sequential machine there is also a recognition algorithm, however, it is so simple: if the input exactly matches the state entrance qualifier, then enter that state.

To clarify what kind of jobs are handled by the recognition algorithm, R, we present the following scenario. Suppose that we have a stroke consisting of two link primitives, $\Psi_{1}$ and $\Psi_{2}$ which is to be recognized by the machine of Figure 5.6. Initially, we start from $\mathrm{q}_{0}$. Then, the first primitive, $\Psi_{1}$, is tested to see if it can satisfy the state entrance qualifier, $\sigma_{1}$, of $q_{1}$. If so, then a transition is made to $q_{1}$, otherwise, we can not proceed any further which means that the input stroke is rejected. Suppose that the transition could be achieved, so we are, now, in state $\mathrm{q}_{1}$. Next, the second primitive, $\psi_{2}$, is tested against six state entrance qualifiers: $\sigma_{1}, \sigma_{2}, \sigma_{8}, \sigma_{9}, \sigma_{10}$, and $\sigma_{11}$. Of course, a transition to $\mathrm{q}_{2}$ is impossible since $\psi_{2}$ is a link primitive and $\sigma_{2}$ is a loop set state entrance qualifier, i.e., their types are different. Regarding the other state entrance qualifiers, all of them are fuzzy directions. This means that $\Psi_{2}$, which is a link having a direction, can be tested against these qualifiers to decide where we can go. It may happen that a transition should be made to many states in parallel. Now, what will be the situation if the stroke has several primitives? Clearly, a huge keep-tracking job is required. So, who keeps track of all of this? It is the brain, i.e., the recognition algorithm, $R$, of the fuzzy sequential machine. Still, one more job is needed if a terminal state could be reached. It may happen also that many terminal states are reached. There, some features of the stroke have to be tested against the fuzzy features of the terminal states. Who performs this? Again, it is the recognition algorithm, $R$, of the fuzzy sequential machine.

### 5.3.1. Definitions

(a) Testing a primitive, $\psi$, to see whether it can be accepted by a state, $q$, is called state excitation. Thus, we say: $\psi$ excites $q$ for possible acceptance. Testing the sequence of primitives against states of a fuzzy sequential machine is called machine
excitation. Here, primitive acceptance by a state means that a transition can be made to that state.
(b) In the recognition algorithm, a primitive will be considered for the two situations: spurious and non-spurious. This is due to the fact that an input stroke may have some spurious (noisy) primitives, i.e., their deletion does not alter the stroke's class. Initially, such primitives are unknown. Thus, we enforce every primitive to be a spurious candidate. This way, noisy strokes are given the chance to be recognized as well as clean strokes.
(c) Any spurious primitive, $\Psi$, can be only accepted by a current state, q, with an acceptance degree $x(\Psi, q)=0$. For example in Figure 5.6, if we are currently at state $\mathrm{q}_{0}$ and the input is a spurious primitive, $\psi_{\mathrm{i}}$, then we can not leave $\mathrm{q}_{0}$ to a new state. Instead we enforce $q_{0}$ to accept $\psi_{i}$ with $\varkappa\left(\psi_{i}, q_{0}\right)=0$. This means that transitions between different states will be caused only by non-spurious primitives.
(d) A non-spurious primitive, $\Psi$, is accepted, by a state, $q$, the state entrance qualifier of which is $\sigma$, with an acceptance degree $\varkappa(\psi, q)$ if:

1. $\Psi$ is a vertex primitive and $\sigma$ is a vertex state entrance qualifier, where $\varkappa(\psi$, $q)=1.0$.
2. $\Psi$ is a loop set primitive having $n$ loops and $\sigma$ is a loop set state entrance qualifier having the same number of loops, n , where $x(\Psi, \mathrm{q})=1.0$.
3. $\Psi$ is a link primitive and $\sigma$ is a fuzzy direction state entrance qualifier, where $x(\Psi, q)=\pi_{0}(\theta), \pi_{o}$ is the possibility distribution that characterizes $\sigma$, and $\theta$ is the angle of the link primitive, $\psi$.
(e) Using a sequence of primitives, $\Psi$, to excite a machine, $F M$, starting from $q_{0}$, results in many possible paths of states accepting the primitives which requires huge amount of memory. These paths have many common states. Searching operations and other calculations are performed on these paths which requires fast processing. A tree is one data structure that can be fastly searched with reduced memory requirements. Thus, the information resulting from machine excitation will be saved in what we call an acceptance tree which has the following properties:
4. The root node of the acceptance tree, which is considered as level 0 of the tree, contains the 2-tuple ( $\mathrm{q}_{0}, 1.0$ ), where $\mathrm{q}_{0}$ is the starting state of FM and the
1.0 is the initial credit, $\omega$, assigned to the unknown stroke. This credit is an initial acceptance degree of the stroke.
5. The nodes of level $i$ of the acceptance tree, $i \geq 1$, contain acceptance information only about primitive $\psi_{i}$ of the unknown stroke. Thus, you expect that the maximum level of the tree will not exceed the number of primitives in the stroke.
6. Each time a primitive, $\Psi$, is accepted by a state, $q$, the stroke is punished by subtracting from its credit, $\omega$, a value, $\delta=\operatorname{lr} \times(1.00-x(\Psi, q))$, where $x(\psi$, $\mathrm{q})$ is determined according to Definitions 5.3.1( $\mathrm{c}, \mathrm{d}$ ) depending on whether the primitive is spurious or non-spurious. Notice that, $\delta$, is proportional to the length ratio, lr , of the primitive, $\psi$, and the degree of disagreement between $\Psi$ and the state entrance qualifier of q , which is a logical punishment rule.
7. Any node in the acceptance tree, other than the root node, contains the 3-tuple $\left(q_{j}, t, \omega\right)$, where $q_{j}$ is an accepting state in FM that was entered after being excited by a primitive, $\Psi_{i}, t \in\{S, N\}$, ( $S$ refers to spurious and $N$ refers to non-spurious), and $\omega$ is the remaining credit for the unknown stroke.

The detailed description of the recognition algorithm, R , which associates a deterministic fuzzy sequential machine, now follows.

## Algorithm 5.3

Use: To recognize a stroke
Input: 1. A fuzzy sequential machine $F M=(C, Q, \Sigma, \zeta, T)$
2. Reduced graph of the stroke
3. The sequence of primitives $\Psi=\left\{\Psi_{i} ; i=1,2, \ldots, m\right\}$ of the stroke

Output: 1. Recognition results: If the stroke is recognized, the output is the class, $c^{*}$, of the stroke and the overall acceptance degree, otherwise, a rejection is reported.
2. If the stroke could not be recognized, the acceptance tree is output to be passed to a subsequent learning process.

## Procedure:

Step 1. Construction of acceptance tree
(a) Create the root node, node 0 , which constitutes level 0 of the tree, with the 2-tuple $\left(q_{0}, 1.0\right)$,
(b) Create level $i$ of the tree which contains acceptance information of primitive, $\Psi_{i}, i$ $=1,2, \ldots, m$ as follows. For every node, $\mathrm{d}_{\mathrm{j}}$, that exists in level $\mathrm{i}-1$ and has the state q and credit $\omega$ as tuples:

1. First assume that $\Psi_{i}$ is a spurious primitive. Add a new node, $d_{k}$, that is a son of node $d_{j}$, where $d_{k}$ has the 3 -tuple ( $q, S, \omega-\delta$ ), and $\delta=l_{i}$, such that the remaining credit, $\omega-\delta$, is not less that a specified threshold, THR $_{1}$.
2. Next, assume that $\Psi_{i}$ is non-spurious. For every next state, $\zeta(q, \sigma)$, of $q$ that can accept $\Psi_{\dot{b}}$ add a new node, $d_{k}$, that is a son of node, $d_{j}$, where $d_{k}$ contains the 3-tuple $(\zeta(q, \sigma), N, \omega-\delta), \delta=1 r_{i} \times\left(1.00-\chi\left(\Psi_{i}, \zeta(q, \sigma)\right)\right)$, such that the remaining credit, $\omega-\delta$, is not less that a specified threshold, $\mathrm{THR}_{1}$.

For example, consider the machine of Figure 5.8 where the sate entrance qualifiers, of states $q_{1}$ to $q_{3}$, are fuzzy directions. Assume that a stroke consisting of a single link primitive, $\Psi_{1}$, is to be recognized using this machine. The above steps are applied as follows:
(a) Create the root node, node 0 , which contains the tuple ( $\mathrm{q}_{0}, 1.0$ ).
(b) To create level 1 of the tree which contains acceptance information of the first primitive, $\Psi_{1}$, i.e., $i=1$, we have to consider the nodes of level $1-1=0$. Level 0 has one node, the root node, thus:

1. First assume that $\Psi_{1}$ is a spurious primitive. Thus, a new node that is a son of the root node is added with the 3-tuple ( $\mathrm{q}_{0}, \mathrm{~S}, 1.00-\delta$ ), where $\delta=1 \mathrm{r}_{1}$.
2. Next, assume that $\psi_{1}$ is non-spurious. There are three next states of $q_{0}$ which can accept $\psi_{1}$. These next states are $\zeta\left(q_{0}, \sigma_{1}\right)=q_{1}, \zeta\left(q_{0}, \sigma_{2}\right)=q_{2}$, and $\zeta\left(q_{0}\right.$, $\left.\sigma_{3}\right)=q_{3}$. Thus, for every state of $q_{1}$ to $q_{3}$, a new node that is a son of node 0 is added. These nodes contain the 3-tuples $\left(q_{1}, N, 1-\delta_{1}\right),\left(q_{2}, N, 1-\delta_{2}\right)$, and $\left(q_{3}, N, 1-\delta_{3}\right)$, where $\delta_{1}=\operatorname{lr}_{1} \times\left(1.00-x\left(\Psi_{1}, q_{1}\right)\right), \delta_{2}=\operatorname{lr}_{1} \times\left(1.00-x\left(\Psi_{1}\right.\right.$, $\left.q_{2}\right)$, and $\delta_{3}=\operatorname{lr} \times\left(1.00-x\left(\Psi_{1}, q_{3}\right)\right)$, respectively. Of course, we assume that the remaining credit in every newly generated node is not less that a specified


Figure 5.8. Four state fuzzy sequential machine, where $\sigma_{1}, \sigma_{2}$, and $\sigma_{3}$ are fuzzy directions.
threshold, $\mathrm{THR}_{1}$.
Step 2. Checks and decisions
After constructing the acceptance tree, one of the following two cases occurs:
(a) The last level of the tree is less than the number of the primitives, m. This occurs if there is no state sequence that can accept the primitive sequence, $\Psi$, provided that the final credit is not less than $\mathrm{THR}_{1}$, in which case the stroke is unknown.
(b) The last level of the tree equals m . In this case, retain the set of leaf nodes, D , of level m . For each node $\mathrm{d} \in \mathrm{D}$, where the node d contains the remaining credit, $\omega_{1}$, of the stroke, do:

1. Trace the path from d to its father node in level 1. Extract those primitives which were counted non-spurious. Let these primitives be represented, from top to bottom of the tree, by the new sequence $\Psi^{\prime}=\left(\psi_{1}^{\prime}, \psi_{2}^{\prime}, \ldots, \psi^{\prime}{ }_{\mathrm{m}}\right), \mathrm{m}^{\prime}$ $\leq \mathrm{m}$, with $\Psi_{m^{\prime}}^{\prime}$ being accepted by some state, $q$.
2. Check if the sequence $\Psi^{\prime}$ is valid. A sequence of primitives is considered valid if the following two conditions are satisfied:
i. The inherent properties, mentioned in Section 4.2, are satisfied, and
ii. $q \in T$, i.e., $q$ is a terminal state, since the last primitive of a sequence should be accepted by a terminal state where stroke classes and their fuzzy features reside.
3. If the sequence $\Psi$ ' is invalid, then it is discarded. Otherwise, let $n$ be the number of the fuzzy features in the terminal state $q$ :
i. Compute the features of explicit relationships between the primitives of

## $\Psi '$, see Section 4.3.

ii. Compute the global features of the sequence $\Psi^{\prime}$, see Section 4.4. Append these to the features computed in (i). The total number of features must sum to the number, $n$, of fuzzy features in q , which is an inherent property of the recognition algorithm.
iii. For each stroke class, c , in q :

Find the acceptance degree, $\pi_{F}(f)$, of each feature, $f$, computed in (i) and (ii) above, in the corresponding fuzzy feature, $F$, of the class, $c$. If the minimum, $\omega_{2}$, of these acceptance degrees is less than a specified threshold, $\mathrm{THR}_{2}$, then the stroke can not be recognized as belonging to class $c$. Otherwise, the class, $c$, and the minimum acceptance degree are retained and a triple of the form $\left(\omega_{1}, \omega_{2}, c\right)$ is created.

If it was not possible to obtain at least one triple, $\left(\omega_{1}, \omega_{2}, c\right)$, then the stroke is unknown. Otherwise, the stroke is assigned the class, c , of the triple which has the maximum $\omega_{3}=\min \left(\omega_{1}, \omega_{2}\right)$, which is the overall acceptance degree of the stroke.

### 5.3.2. Example

A graph of Arabic numeral " r " and its reduced graph are shown in Figure 5.9. The loop in Figure 5.9(a) is spurious. This stroke is to be recognized by the fuzzy machine shown in Figure 5.6. The stroke is segmented into the following primitives: $\Psi_{1}=l(1,2)$, $\Psi_{2}=\star(2), \psi_{3}=O(3), \psi_{4}=\star(2)$, and $\Psi_{5}=l(2,4)$. The lengths of these primitives are: $l_{1}$ $=144, l_{2}=0, l_{3}=56, l_{4}=0$, and $l_{5}=195$. The total length of the stroke is $\mathrm{L}=\sum_{i=1}^{5} l_{\mathrm{i}}=$ 395. Length ratios are: $\mathrm{lr}_{1}=0.37, \mathrm{lr}_{2}=0.00, \mathrm{lr}_{3}=0.14, \mathrm{lr}_{4}=0.00$, and $\mathrm{lr}_{5}=0.49$. The two link primitives $\Psi_{1}$ and $\psi_{5}$ have angles $\theta_{1}=208^{\circ}$ and $\theta_{5}=329^{\circ}$, respectively. The loop set primitive $\Psi_{3}$ has one loop. In this example, $\mathrm{THR}_{1}$ and $\mathrm{THR}_{2}$ are set to 0.6 .

Step 1. Construction of acceptance tree, see Figure 5.10.
(a) Create the root of the tree, node 0 , with the 2-tuple $\left(q_{0,}, 1.00\right)$, i.e., we start from the starting state, $\mathrm{q}_{0}$, of the machine shown in Figure 5.6 with an initial credit $\omega=1.00$.


Figure 5.9. (a) A graph of Arabic numeral " $\ulcorner$ " with a spurious loop, and (b) it's reduced graph.


Figure 5.10. Acceptance tree for Example 5.3.2.
(b) Create level 1 of the tree which contains acceptance information about the first primitive, $\psi_{1}$. In level 1-1 $=0$, there is one node, 0 , which has $q_{0}$ as one tuple, thus:

1. First, the primitive $\psi_{1}$ is considered spurious. The value to be discounted from the credit of node 0 , is $\delta=\mathrm{Ir}_{1}=0.37$. The remaining credit will be $1.0-0.37$ $=0.63>\mathrm{THR}_{1}$. Thus, a new node, 1 , is created with the 3 -tuple ( $\mathrm{q}_{0}, \mathrm{~S}, 0.63$ ).
2. Now, consider $\Psi_{1}$ as non-spurious. There is one next state, $q_{1}$, of $q_{0}$ with a
fuzzy direction state entrance qualifier $\sigma_{1}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} 30^{\circ}\right)$. Thus, the link primitive, $\Psi_{1}$, can be accepted by state $q_{1}$ with an acceptance degree $x\left(\psi_{1}, q_{1}\right)=\pi_{01}\left(\theta_{1}=208^{\circ}\right)=1.00$. If a value $\delta=1 r_{1} \times\left(1.00-x\left(\psi_{1}, q_{1}\right)\right)=$ 0.0 is discounted from the credit of node 0 , then the remaining credit will be $1.0-0.0=1.0>$ THR $_{1}$. Thus, a new node, 2 , is created with the 3-tuple ( $\mathrm{q}_{1}$, $\mathrm{N}, 1.00$ ).

Create level 2 of the tree which contains acceptance information about the second primitive, $\Psi_{2}$. In level 2-1 $=1$, there are two nodes, 1 and 2. Thus:

First, consider node 1 , which has $q_{0}$ as one tuple:

1. Consider $\psi_{2}$ as spurious. The value to be discounted from the credit of node 1 is $\delta=\mathrm{r}_{2}=0.00$. The remaining credit will be $0.63-0.00=0.63>\mathrm{THR}_{1}$. Thus, a new node, 3 , is created with the 3 -tuple ( $\mathrm{q}_{0}, \mathrm{~S}, 0.63$ ).
2. Consider $\psi_{2}$ as non-spurious. Since $\psi_{2}$ is a vertex primitive and there is no next state of $q_{0}$ with a vertex state entrance qualifier, no more sons are added to node 1.

Second, consider node 2, which has $q_{1}$ as one tuple:

1. Consider $\psi_{2}$ as spurious. The value to be discounted from the credit of node 2 is $\delta=\mathrm{r}_{2}=0.00$. The remaining credit will be $1.00-0.00=1.00>\mathrm{THR}_{1}$. Thus, a new node, 4 , is created with the 3 -tuple ( $q_{1}, S, 1.00$ ).
2. Consider $\psi_{2}$ as non-spurious. For state $q_{1}$, there is only one next state, $q_{2}$, which has a vertex state entrance qualifier. Thus, $\Psi_{2}$, which is a vertex primitive, can be accepted by $\mathrm{q}_{2}$. The value to be discounted from the credit of node 2 is $\delta=1 \mathrm{r}_{2}=0.00$. The remaining credit will be $1.00-0.00=1.00>$ $\mathrm{THR}_{1}$. Thus, a new node, 5 , is created with the 3 -tuple ( $\mathrm{q}_{2}, \mathrm{~N}, 1.00$ ).
Similarly, levels 3 to 5 are created. Finally, the tree of Figure 5.10 is obtained with the nodes being referenced by the numbers underneath.
Step 2. Checks and decisions
Here, case (b) of Step 2 of Algorithm 5.3, applies:
(b) The tree has node, 13 , with its level equal to $5=\mathrm{m}$; the number of primitives. The remaining credit in node 13 is 0.65 :
3. By following the path from node 13 to node 2 and extracting the primitives
that were counted non-spurious, a new stroke sequence, $\Psi^{\prime}=\left\{\Psi_{1}^{\prime}, \Psi_{2}^{\prime}\right\}$ is obtained where $\Psi_{1}^{\prime}=\Psi_{1}$ and $\Psi_{2}^{\prime}=\Psi_{3}$. The primitive $\Psi_{2}^{\prime}$ is accepted by the state $\mathrm{q}_{10}$.
4. The sequence $\Psi$ ' fulfils the inherent properties mentioned in Section 4.2, however, $\mathrm{q}_{10} \notin \mathrm{~T}$, where T is the set of terminal states of the fuzzy machine of Figure 5.6. Thus, condition (ii) of the two conditions mentioned in Step 2(b)2 of Algorithm 5.3 is not satisfied which makes $\Psi '$ an invalid sequence. Hence, it is discarded.

Also, the tree has node, 14 , with its level equal to $5=\mathrm{m}$; the number of primitives. The remaining credit in node 14 is 0.86 :

1. By following the path from node 14 to node 2 and extracting the primitives that were counted non-spurious, a new stroke sequence, $\Psi^{\prime}=\left\{\Psi_{1}^{\prime}, \Psi_{2}^{\prime}\right\}$ is obtained where $\Psi_{1}^{\prime}=\Psi_{1}$ and $\Psi_{2}^{\prime}=\Psi_{5}$. The primitive $\Psi_{2}^{\prime}$ is accepted by the state $\mathrm{q}_{11}$.
2. The sequence $\Psi$ ' fulfils the inherent properties mentioned in Section 4.2 and $q_{11} \in T$, where $T$ is the set of terminal states of the fuzzy machine of Figure 5.6. Thus, the two conditions mentioned in Step 2(b)2 of Algorithm 5.3 are satisfied which makes $\Psi '$ a valid sequence.
3. i. There is no explicit relationship features for the sequence $\Psi^{\prime}$ since it does not have connected loop set and vertex primitives.
ii. Five global features are calculated for the sequence $\Psi ':$ DP type feature $f_{1}=1.00$, height to height-plus-width ratio feature $f_{2}=0.50$, two features to relate the heights of the right and left ends to the total height of the stroke $f_{3}=0.00, f_{4}=1.00$, respectively, and feature of curvature $f_{5}=0.03$. Thus, we end at a total number of features, 5 , which equals the number of fuzzy features in $\mathrm{q}_{11}$, as it should be.
iii. State $\mathrm{q}_{11}$ has one class, Arabic numeral " r " with five fuzzy features, $\mathrm{F}_{1}$ to $F_{5}$, refer to Example 5.2.3. The acceptance degrees of the features which were computed in (ii) in the corresponding fuzzy features are: $\pi_{\text {Fi }}\left(f_{i}\right)=1.00, i=1,2,3,4,5$. The minimum value of the acceptance of
features is $\underset{i=1}{s} \min ^{2}\left(\pi_{\mathrm{Fi}}\left(\mathrm{f}_{\mathrm{i}}\right)\right)=1.00>$ THR $_{2}$. Thus, the class " $r$ " and the value
1.00 are retained and the triple $(0.86,1.00, " r ")$ is created.

Since only one triple, $(0.86,1.00, " r ")$, could be obtained, the stroke is recognized as a member of the class " $r$ " with an overall acceptance degree of $0.86=\min (0.86$, 1.00).

## SUMMARY

In this chapter, The recognition component of the IACR system was described in detail. It was shown how a sequential machine model can be used to define recognizers of handwritten Arabic strokes. Flexibility is added to the system by building on concepts from fuzzy set theory to finally define fuzzy sequential machines. The recognition algorithm was formally presented and demonstrated by an example. The automatic generation of such fuzzy sequential machines will be addressed in the next chapter.

## Stroke

## Learning



## OVERVIEW

In this chapter the learning component of the IACR system is presented. A data flow diagram of the learning process is shown in Figure 6.1, which consists of two sides, where only one side is active at a time. The learning algorithm determines which side to activate, depending on the last level of the acceptance tree of an unrecognized stroke which is passed by the recognition algorithm, as follows:
(a) If the last level of the acceptance tree is less than the number of primitives in the stroke, then the right side is activated, which consists of the following subprocesses:

1. Generating Fuzzy Sequential Machine from the Input Stroke, Section 6.2: where the inputs are the sequence of primitives and reduced graph of the stroke. The output is a fuzzy sequential machine which can recognize the input stroke and variants of it.
2. Merging Fuzzy Sequential Machines, Section 6.3: which accepts as inputs the machine generated in (1) and the old machine which was used in recognition but failed to recognize the stroke. The result is a new machine which can recognize the input stroke and strokes of the old machine and variants of these strokes.
(b) If the last level of the acceptance tree equals the mumber of primitives in the stroke, then the left side of Figure 6.1 is activated, where some modifications are


Figure 6.1. Data flow diagram of the Stroke Learning process in the IACR system. Only one side of this diagram is active at a time. If the last level of the acceptance tree is less than the number of primitives in the unknown stroke, then the right side is activated, otherwise, the left side is activated.
introduced into the machine that was used in recognition, which are sufficient to learn the stroke, Section 6.4. The inputs in this case are the sequence of primitives, reduced graph of the stroke, and the acceptance tree. The output is a modified fuzzy sequential machine which can recognize the input stroke and strokes of the old machine and variants of them.

### 6.1. LEARNING

In Chapter 5, fuzzy sequential machines were introduced as a new model to recognize handwritten Arabic strokes. If the system is unable to recognized a stroke, then there should be somemeans to learn it. This chapter is concerned with the learning process of unrecognized strokes. If $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, \mathrm{~T}_{1}\right)$ is the machine that was used when recognition was tried but failed, then our goal is to modify or expand $\mathrm{FM}_{1}$ so that it
becomes capable of recognizing strokes belonging to the unknown stroke's class. Of course, the set $\mathrm{Q}_{1}$ is initially empty. $\mathrm{FM}_{1}$ possesses new states and classes when more strokes are learned. An algorithm to learn strokes is described below.

## Algorithm 6.1

Use: To learn a stroke
Input: 1. Sequence of primitives, $\Psi$, of the stroke to be learned
2. Reduced graph of the stroke to be learned
3. Acceptance tree of the stroke which is unrecognized by Algorithm 5.3
4. Fuzzy sequential machine, $\mathrm{FM}_{1}$, of strokes which are already learned

Output: New fuzzy sequential machine, FM, which can recognize the underlying stroke and previously learned strokes

## Procedure:

Depending on the last level of the acceptance tree of the unrecognized stroke, one of the following two cases occurs:
(a) The last level of the acceptance tree is less than the number of primitives, $m$ : This means that it was not possible for the whole sequence of primitives to terminate at some state such that the remaining credit of the stroke $\geq \mathrm{THR}_{1}$. In this case, simple modifications of some states are not adequate. We adopted the following solution:

1. Obtain a new fuzzy sequential machine, $\mathrm{FM}_{2}$, from the stroke's sequence of primitives, $\Psi$. This machine can recognize strokes belonging to the underlying stroke's class. Section 6.2 explains how to obtain a fuzzy sequential machine from a given input stroke.
2. Merge $\mathrm{FM}_{1}$ and $\mathrm{FM}_{2}$ to obtain the machine FM that is capable of recognizing the strokes belonging to the underlying stroke's class with an overall acceptance degree equal to 1.00 . The new machine still memorizes all previously learned strokes. Merging of sequential machines is explained in Section 6.3.
(b) The last level of the tree equals, $m$, the number of primitives in the sequence, $\Psi$. This case occurs:
3. if the stroke sequence could terminate at one or more terminal state such that
the remaining credit of the stroke did not fall below $\mathrm{THR}_{1}$, but could not be recognized due to severe deviation of the stroke's features from the fuzzy features of the classes embedded in any of these terminal states, i.e., the acceptance degree of the stroke's features is less than THR $_{2}$.
4. and / or the stroke sequence terminated in one or more states, NT, which are not terminal states and such that the remaining credit of the stroke is not less than THR $_{1}$.

In this case, there is no need to create a new fuzzy sequential machine as in (a). Only some modifications to $\mathrm{FM}_{1}$ are sufficient, which are:

1. Either add a new class, with its fuzzy features, in one terminal state which already exists and could be reached by the sequence of primitives, $\Psi$, or
2. Change one of the states in NT into a terminal state, i.e., append it to the set T , and insert in it the class of the stroke with its fuzzy features.

These modifications are explained in detail in Section 6.4.

### 6.2. GENERATION OF FUZZY SEQUENTIAL MACHINES

As shown in Figure 6.1, a necessary step in the learning process is to obtain a fuzzy sequential machine from an input stroke. The manual generation of this machine is cumbersome, since it may contain several states requiring proper state entrance qualifiers, next state mapping function, and insertion of stroke classes with their fuzzy features in suitable states. Thus, it is important to search for somemeans to automate the generation of fuzzy sequential machines. In this section, an algorithm is developed for this purpose which is based on the following points:
(a) For a vertex or loop set primitive, one state is generated which has a vertex or loop set state entrance qualifier. Thus, when the primitive excites this state, the later will accept it.
(b) For a link primitive, a state is generated which has an adjusted fuzzy direction state entrance qualifier such that the link is $100 \%$ accepted by that state.
(c) For each two consecutive link primitives, a sequence of states is generated, each having an adjusted fuzzy direction state entrance qualifier. Transitions can be made from any of these states to itself in addition to the other states following it in the


Figure 6.2. A fuzzy sequential machine which can be generated from a stroke consisting of two link primitives.
sequence. By adopting this, we add flexibility to the generated fuzzy sequential machine since the change from a given direction of a link primitive to another direction of another link primitive can be achieved either directly or gradually by passing via many intermediate states. For example, consider the machine of Figure 6.2 , in which $\sigma_{1}=\left(15^{\circ} / 30^{\circ} ; 45^{\circ} \backslash 30^{\circ}\right), \sigma_{2}=\left(45^{\circ} / 30^{\circ} ; 75^{\circ} 130^{\circ}\right), \sigma_{3}=\left(75^{\circ} / 30^{\circ}\right.$; $105^{\circ} 130^{\circ}$ ), and $\sigma_{4}=\left(105^{\circ} / 30^{\circ} ; 135^{\circ} 130^{\circ}\right)$. This kind of machine can be generated by a stroke having two link primitives, $\Psi_{1}$ and $\Psi_{2}$, which have the angles $25^{\circ}$ and $130^{\circ}$, respectively. Other similar strokes consisting of link primitives which can be accepted by $\mathrm{q}_{1}$ followed by link primitives which can be accepted by $\mathrm{q}_{4}$, can be recognized. Now, what happens for a stroke consisting of the four link primitives, $\psi_{1} \psi_{2}, \psi_{3}$, and $\psi_{4}$, which have the direction angles $25^{\circ}, 50^{\circ}, 90^{\circ}$, and $130^{\circ}$, respectively?. Clearly, We can not proceed from $q_{0}$ to $q_{1}$ to $q_{4}$, without passing through other intermediate states. The best sequence of states which can accept these primitives is $q_{0}, q_{1}, q_{2}, q_{3}, q_{4}$. From this example, we can see that although the stroke which is used to generate this machine requires only states $q_{0}, q_{1}$, and $q_{4}$, future variants of this stroke were taken into account by adding states $q_{2}$ and $q_{3}$. Thus, it becomes clear how much flexibility is added by adopting such a scheme in generating fuzzy sequential machines.
(d) In the final generated state, a class of the underlying stroke is added with fuzzy features which are obtained from the calculated stroke's features.
A formal description of the algorithm follows.

## Algorithm 6.2

Use: To generate a fuzzy sequential machine from a stroke
Input: 1. Sequence of primitives of the stroke to be learned $\Psi=\left\{\Psi_{j}, j=1,2,3\right.$, . . ., m $\}$
2. Reduced graph of the stroke to be learned

Output: Fuzzy sequential machine $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$ which can recognize strokes belonging to the class of the input stroke

## Procedure:

Step 1. Initialization
Let $\mathrm{i}=1$ and $\mathrm{j}=1$, where i and j are used as indices for the generated states and the primitives of the stroke, respectively.

Step 2. Creation of the starting state
(a) Create the starting state $\mathrm{q}_{0}$.
(b) If $\Psi_{1}$ is a vertex then go to Step 3, else if $\Psi_{1}$ is a loop set then go to Step 4, else go to Step 5.

Step 3. Creation of an accepting state for the vertex primitive $\Psi_{\mathbf{j}}$
(a) Create a new state, $q_{i}$, which has $\sigma_{i}=\star$ as a vertex state entrance qualifier. $A$ directed arc, which is labelled with $\sigma_{i}$, is added to point from state $q_{i-1}$ to state $q_{i}$, i.e., we set $\zeta\left(q_{i-1}, \sigma_{i}\right)=q_{i}$.
(b) If $\psi_{j}$ is the last primitive then go to Step 7, else: (a) increment $i$ and $j$; (b) if $\Psi_{j}$ is a loop set then go to Step 4, else go to Step 5.

Step 4. Creation of an accepting state for the loop set primitive $\Psi_{j}$
(a) Create a new state $q_{i}$ which has $\sigma_{i}=O(n)$ as a loop set state entrance qualifier, where n is the number of loops in the loop set primitive. A directed arc, which is labelled with $\sigma_{\dot{p}}$ is added to point from state $q_{i-1}$ to state $q_{i}$, i.e., we set $\zeta\left(q_{i-1}, \sigma_{i}\right)=$ $q_{i}$.
(b) If $\Psi_{\mathrm{j}}$ is the last primitive then go to Step 7, else increment i and j and go to Step 3. According to inherent properties of sequences of primitives, Section 4.2, notice that for a loop set primitive, $\Psi_{j}, \Psi_{j+1}$ must be a vertex primitive.
Step 5. Creation of an accepting state for the link primitive $\Psi_{j}$, which is not
preceded by a link primitive, i.e., either $\mathrm{j}=1$ or $\psi_{\mathrm{j}-1}$ is a vertex primitive. According to inherent properties of sequences of primitives, Section 4.2, notice that for a link primitive $\Psi_{j}, \Psi_{j-1}$ or $\psi_{j+1}$ can not be a loop set primitive.
(a) Fuzzify the direction angle, $\theta$, of $\Psi_{j}$ by transforming it into a fuzzy direction $\sigma_{i}=$ $\left(p_{1} / \beta_{1} ; p_{2} \backslash \beta_{2}\right)$ as follows:

1. Set both the left and right bandwidths, $\beta_{1}$ and $\beta_{2}$, to suitable values.
2. The range from $0^{\circ}$ to $360^{\circ}$ is divided into $360^{\circ} /$ core angular intervals. The first angular interval is centred at $0^{\circ}$. If $\theta$ lies in the angular interval the centre of which is $\Theta_{0}$, then set the left peak point $p_{1}=\Theta_{0}$ - core / 2 and the right peak point $p_{2}=\Theta_{0}+$ core $/ 2$.
(b) Create a new state $q_{i}$ which has the fuzzy direction, $\sigma_{i}$, as its state entrance qualifier. Two directed arcs, which are labelled with $\sigma_{i}$, are added to point from $q_{i-1}$ to $q_{i}$ and from $q_{i}$ to itself and , i.e., we set $\zeta\left(q_{i-1}, \sigma_{i}\right)=q_{i}$ and $\zeta\left(q_{i}, \sigma_{i}\right)=q_{i}$.
(c) If $\Psi_{j}$ is the last primitive then go to Step 7, else: if $\Psi_{j+1}$ is a vertex primitive then increment $i$ and $j$ and go to Step 3, else go to Step 6.
Step 6. Creation of an accepting state for the link primitive $\psi_{j+1}$ and a sequence of states which lie between state $q_{i}$ and the state which accepts $\psi_{j+1}$ to allow for gradual change of direction
Let $\theta_{\mathbf{k}}$ be the angle of the primitive, $\Psi_{j+1}$, which lies in the angular interval the centre of which is $\Theta_{k}$ :
(a) Consider the centres $\Theta_{1}, \Theta_{2}, \ldots, \Theta_{k}$, of the angular intervals which lie between $\Theta_{0}$ and $\Theta_{b}$ where $\Theta_{l}=\Theta_{0} \pm l \times \operatorname{core}, l=1, \ldots, k$. The plus sign is used if $\Theta_{0}<\Theta_{k}$, and vice versa.
(b) Create k new states $\mathrm{q}_{\mathrm{i}+1}, \mathrm{q}_{\mathrm{i}+2}, \ldots, \mathrm{q}_{\mathrm{i}+\mathrm{k}}$, where state $\mathrm{q}_{\mathrm{i}+1}$ has the fuzzy direction $\sigma_{\mathrm{i}+1}=$ $\left(p_{n} / \beta_{1} ; p_{n} \backslash \beta_{2}\right)$ as its state entrance qualifier, $p_{n}=\Theta_{l}-$ core $/ 2, p_{n}=\Theta_{l}+$ core $/ 2$, $\beta_{1}$ and $\beta_{2}$ are the left and right bandwidths.
(c) Add directed arcs from state $q_{a}$ to states $q_{a+1}, q_{a+2}, \ldots, q_{i+k}$, where $a=i, i+1, \ldots$, $\mathrm{i}+\mathrm{k}-1$. All arcs entering a state $\mathrm{q}_{\mathrm{i}+1}$ are labelled with the fuzzy direction state entrance qualifier $\sigma_{i+1}$. Thus, we get $\zeta\left(q_{v}, \sigma_{b}\right)=q_{b}, a=i, i+1, \ldots, i+k-1$, and $b$ $=a+1, \ldots, i+k$.
(d) For every state $\mathrm{q}_{\mathrm{i}+1}$ add a directed arc, which is labelled with $\sigma_{i+1}$, to point from that
state to itself, i.e., $\zeta\left(\mathrm{q}_{\mathrm{i}+b}, \sigma_{\mathrm{i}+1}\right)=\mathrm{q}_{\mathrm{i}+b} l=1,2, \ldots, \mathrm{k}$.
(e) Increment i by k and j by 1.
(f) If $\psi_{j}$ is the last primitive then go to Step 7, else if $\psi_{j+1}$ is a link primitive then let $\Theta_{0}$ $=\Theta_{k}$ and repeat Step 6, else increment i and j and go to Step 3.
In Steps 5 and 6, the three parameters: core, $\beta_{1}$, and $\beta_{2}$ were suitably found to be equal to $30^{\circ}$.

Step 7. Setup of a terminal state
The last created state, $\mathrm{q}_{\mathrm{i}}$, is considered as a terminal state. Thus, for the machine, FM, T $=\left\{q_{i}\right\}$. A class, $c_{\text {new, }}$ of the underlying stroke, with fuzzy features, is embedded in $q_{i}$, hence, $C=\left\{c_{\text {new }}\right\}$. The fuzzy features are calculated as follows:
(a) Find the explicit relationships between primitives of the sequence $\Psi$, see Section 4.3. Find the global features of the stroke, see Section 4.4, and append them to the features of explicit relationships.
(b) Fuzzify every feature, f, computed in (a), of the stroke by transforming it into an s/znumber as follows:

1. Set both the left and right bandwidths to $\boldsymbol{\beta}_{\mathrm{f}}$.
2. The range from 0.0 to 1.0 is divided into $n_{f}$ equal intervals. The left and right peak points, $p_{1}$ and $p_{2}$, are taken to be equal to the limits of the interval in which the value flies.

Suitable values of $\beta_{f}$ and $n_{f}$ were found to be 0.2 and 5 , respectively. For example if $\mathrm{f}=0.65$, then the corresponding fuzzy feature will be $\mathrm{F}=(0.6 / 0.2 ; 0.8 \backslash 0.2)$.

### 6.2.1. Example

Figures 6.3(a,b) show the graph, $G$, and the reduced graph, $\mathrm{G}^{\prime}$, of a stroke, Damma. The stroke consists of the following primitives: $\Psi_{1}=l(1,2), \Psi_{2}=\star(2), \Psi_{3}=$ $O(3), \psi_{4}=\star(4), \psi_{5}=l(4,5)$, and $\Psi_{6}=l(5,6)$. A fuzzy sequential machine, $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}\right.$, $Q_{1}, \Sigma_{1}, \zeta_{1}, T_{1}$ ), is to be obtained which can recognize this stroke and other strokes belonging to Damma class. For the sake of easy referencing, in the next sections where other machines are generated, we use two subscripts to refer to a state in the machine, $\mathrm{FM}_{1}$, e.g., $\mathrm{q}_{1 i}$ refers to state $\mathrm{q}_{\mathrm{i}}$ of machine $\mathrm{FM}_{1}$. See Figure 6.3(c) while reading the following steps.


Figure 6.3. (a) A graph, G, of an Arabic stroke, Damma, (b) its reduced graph, $\mathrm{G}^{\prime}$, and (c) a deterministic fuzzy sequential machine obtained from this stroke.

## Step 1. Initialization

Let $\mathrm{i}=1$ and $\mathrm{j}=1$.
Step 2. Creation of the starting state
(a) Create the starting state, $\mathrm{q}_{10}$.
(b) Since $\psi_{1}$ is a link primitive go to Step 5 .

Step 5. Creation of an accepting state for the link primitive $\Psi_{1}=l(1,2)$ with angle $\theta=207^{\circ}$ :
(a) Fuzzify the angle value $207^{\circ}$ by transforming it into a fuzzy direction $\sigma_{11}=\left(p_{1} / \beta_{1}\right.$; $p_{2} \backslash \beta_{2}$ ) as follows:

1. Set both the left and right bandwidths, $\beta_{1}$ and $\beta_{2}$, to $30^{\circ}$.
2. By dividing the range from $0^{\circ}$ to $360^{\circ}$ to $360^{\circ} /$ core $=360^{\circ} / 30^{\circ}=12$ angular intervals, we find that the angle $207^{\circ}$ lies in the angular interval the centre of which is $\Theta_{0}=210^{\circ}$. The left peak point is set to $p_{1}=210^{\circ}$ - core / $2=210^{\circ}-30^{\circ} / 2=195^{\circ}$ and the right peak point is set to $\mathrm{p}_{2}=210^{\circ}+$ core $/ 2=210^{\circ}+30^{\circ} / 2=225^{\circ}$. Finally, we get $\sigma_{11}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right)$.
(b) Create a new state $q_{11}$ which has the fuzzy direction $\sigma_{11}$ as its state entrance qualifier. Two directed arcs, which are labelled with $\sigma_{11}$, are added to point from $q_{10}$ to $q_{11}$ and from $q_{11}$ to itself, i.e., we set $\zeta_{1}\left(q_{10}, \sigma_{11}\right)=q_{11}$ and $\zeta_{1}\left(q_{11}, \sigma_{11}\right)=q_{11}$.
(c) Since there are more primitives and the next primitive, $\Psi_{2}$, is a vertex we increment i to 2 and j to 2 and go to Step 3.
Step 3." Creation of an accepting state for the vertex primitive $\psi_{2}$
(a) Create a new state $q_{12}$ which has $\sigma_{12}=\star$ as a vertex state entrance qualifier. A directed arc, which is labelled with $\sigma_{12}$, is added to point from state $q_{11}$ to state $q_{12}$, i.e., we set $\zeta_{1}\left(q_{11}, \sigma_{12}\right)=q_{12}$.
(b) Since there are more primitives: (1) increment ito 3 and j to 3 and (2) go to Step 4 since $\psi_{3}$ is a loop set primitive.

Step 4. Creation of an accepting state for the loop set primitive $\Psi_{3}$
(a) Create a new state $q_{13}$ which has $\sigma_{13}=O(1)$ as a loop set state entrance qualifier, where the 1 is the number of loops in the loop set primitive. A directed arc, which is labelled with $\sigma_{13}$, is added to point from state $q_{12}$ to state $q_{13}$, i.e., we set $\zeta_{1}\left(q_{12}\right.$, $\left.\sigma_{13}\right)=q_{13}$.
(b) Since there are more primitives, increment ito 4 and j to 4 and go to Step 3.

Step 3. Creation of an acceptance state for the vertex primitive $\Psi_{4}$
(a) For the vertex primitive, $\psi_{4}$, create a new state $q_{14}$ which has $\sigma_{14}=\star$ as a vertex state entrance qualifier. A directed arc, which is labelled with $\sigma_{14}$, is added to point from state $\mathrm{q}_{13}$ to state $\mathrm{q}_{14}$, i.e., we set $\zeta_{1}\left(\mathrm{q}_{13}, \sigma_{14}\right)=\mathrm{q}_{14}$.
(b) Since there are more primitives: (1) increment i to 5 and j to 5 and (2) go to Step 5 since $\psi_{5}$ is a link primitive.

Step 5. Creation of an accepting state for the link primitive $\psi_{s}=l(4,5)$ with angle $\Theta=2633^{\circ}$ :
(a) Fuzzify the angle value $263^{\circ}$ by tränsforming it into a fuzzy direction $\dot{\sigma}_{15}=\left(p_{1} / \beta_{1}\right.$; $\mathrm{p}_{2} \backslash \beta_{2}$ ) as follows:

1. Set both the left and right bandwidths, $\beta_{1}$ and $\beta_{2}$, to $30^{\circ}$.
2. By dividing the range from $0^{\circ}$ to $360^{\circ}$ to $360^{\circ} /$ core $=360^{\circ} / 30^{\circ}=12$ angular intervals, we find that the angle $263^{\circ}$ lies in the angular interval the centre of which is $\Theta_{0}=270^{\circ}$. The left peak point is set to $p_{1}=270^{\circ}$ - core /

$$
\begin{aligned}
& 2=270^{\circ}-30^{\circ} / 2=255^{\circ} \text { and the right peak point is set to } p_{2}=270^{\circ}+\text { core } \\
& / 2=270^{\circ}+30^{\circ} / 2=285^{\circ} . \text { Finally, we get } \sigma_{1 s}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right) .
\end{aligned}
$$

(b) Create a new state $q_{1 s}$ which has the fuzzy direction $\sigma_{1 s}$ as its state entrance qualifier. Two directed arcs, which are labelled with $\sigma_{15}$, are added to point from $q_{14}$ to $q_{1 s}$ and from $q_{1 s}$ to itself, i.e., we set $\zeta_{1}\left(q_{14}, \sigma_{15}\right)=q_{1 s}$ and $\zeta_{1}\left(q_{1 s} ; \sigma_{1 s}\right)=q_{1 s}$.
(c) Since the link primitive, $\Psi_{5}$, is directly followed is by another link primitive, $\Psi_{6}$, we go to Step 6.

Step 6. Creation of an accepting state for the link primitive $\Psi_{6}$ and a sequence of states which lie between state $\mathrm{q}_{1 s}$ and the state which accepts $\Psi_{6}$

The angle of $\Psi_{6}$ is $\theta_{k}=219^{\circ}$, which lies in the angular interval the centre of which is $\theta_{k}=$ $210^{\circ}$ :
(a) Consider the centres $\Theta_{1}=240^{\circ}$ and $\Theta_{2}=210^{\circ}$, i.e., $k=2$, of the angular intervals which lie between $\Theta_{0}=270^{\circ}$ and $\theta_{2}=210^{\circ}$.
(b) Create two (since $k=2$ ) new states, $\mathrm{q}_{16}$ and $\mathrm{q}_{17}$, where state $\mathrm{q}_{16}$ has the fuzzy direction $\sigma_{16}=\left(p_{11} / \beta_{1} ; p_{12} \backslash \beta_{2}\right)$ as its state entrance qualifier, and state $q_{17}$ has the fuzzy direction $\sigma_{17}=\left(p_{21} / \beta_{1} ; p_{22} \backslash \beta_{2}\right)$ as its state entrance qualifier, $p_{11}=\Theta_{1}$ - core $/ 2=240^{\circ}-30^{\circ} / 2=225^{\circ}, \mathrm{p}_{12}=\Theta_{1}+\operatorname{core} / 2=240^{\circ}+30^{\circ} / 2=255^{\circ}, \mathrm{p}_{21}=\Theta_{2}$ - core $/ 2=210^{\circ}-30^{\circ} / 2=195^{\circ}, \mathrm{p}_{22}=\Theta_{2}+$ core $/ 2=210^{\circ}+30^{\circ} / 2=225^{\circ}, \beta_{1}$ and $\beta_{2}$ are set to $30^{\circ}$. Thus, we get $\sigma_{16}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right)$ and $\sigma_{17}=$ $\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right)$.
(c) Add directed arcs to point from state $\mathrm{q}_{15}$ to states $\mathrm{q}_{16}$ and $\mathrm{q}_{17}$, and from state $\mathrm{q}_{16}$ to state $\mathrm{q}_{17}$. The arcs entering states $\mathrm{q}_{16}$ and $\mathrm{q}_{17}$ are labelled with the fuzzy direction state entrance qualifiers $\sigma_{16}$ and $\sigma_{17}$, respectively. Thus, we get $\zeta_{1}\left(q_{15}, \sigma_{16}\right)=q_{16}$, $\zeta_{1}\left(q_{15}, \sigma_{17}\right)=q_{17}$, and $\zeta_{1}\left(q_{16}, \sigma_{17}\right)=q_{17}$.
(d) For state $q_{16}$ add a directed arc, which is labelled with $\sigma_{16}$, to point from that state to itself. Similarly a directed arc, which is labelled with $\sigma_{17}$, is added to point from state $\mathrm{q}_{17}$ to itself. Thus, we get $\zeta_{1}\left(\mathrm{q}_{16}, \sigma_{16}\right)=\mathrm{q}_{16}$ and $\zeta_{1}\left(\mathrm{q}_{17}, \sigma_{17}\right)=\mathrm{q}_{17}$.
(e) Increment ito 7 and j to and j to 6.
(f) Since $\Psi_{6}$ is the last primitive go to Step 7.

Step 7. Setup of a terminal state
The last created state, $\mathrm{q}_{17}$, is considered as a terminal state. Thus, for the machine, $\mathrm{FM}_{1}$,
$T_{1}=\left\{q_{17}\right\}$. A class, Damma, which is the class to which the underlying stroke belongs, is embedded in $\mathrm{q}_{17}$ with its fuzzy features, hence, $\mathrm{C}_{1}=\{$ Damma $\}$. The fuzzy features are calculated as follows:
(a) Explicit relationships between primitives are found as follows, see Section 4.3: Primitive $\Psi_{2}$ is vertex that is connected to a loop set primitive, $\Psi_{3}$. This relationship is expressed by the two features: $f_{1}=1.0$ and $f_{2}=0.62$, which relate the $x \& y$ coordinates of the vertex primitive to the $x \& y$ coordinates of centre of gravity of the loop set primitive. Similarly, another relationship is expressed between the two primitives $\Psi_{4}$ and $\Psi_{3}$ by the two features: $f_{3}==0.87$ and $f_{4}=0.85$. The global features of the stroke are found and appended to the features of explicit relationships, see Section 4.4: There are two features of DP type since a stroke belonging to Damma class has two dominant paths. These two features are $f_{5}=$ 0.13 and $f_{6}=0.50$. There is one feature of height to height-plus-width ratio: $f_{7}=$ 0.46 . For this stroke, the features of heights of right and left ends and the feature of curvature are not defined since it has vertex or loop set primitives.
(b) The features which are calculated above are fuzziffied by transforming them into $\mathrm{s} / \mathrm{z}$ numbers. $\beta_{f}$ and $n_{f}$ are set to 0.2 and 5 , respectively. Thus, the following fuzzy features are obtained: two fuzzy features expressing an explicit relationship between a vertex primitive connected to a loop set primitive: $\mathrm{F}_{1}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$ and $F_{2}=(0.6 / 0.2 ; 0.8 \backslash 0.2)$, two other similar fuzzy features: $F_{3}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$ and $F_{4}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$, which express another explicit relationship between another vertex primitive connected to the same loop set primitive, two global fuzzy features which are of DP type: $\mathrm{F}_{5}=(0.0 / 0.2 ; 0.2 \backslash 0.2)$ and $\mathrm{F}_{6}=(0.4 / 0.2 ; 0.6 \backslash$ 0.2 ), and a global fuzzy feature of height to height-plus-width ratio: $F_{7}=(0.4 / 0.2$; $0.6 \backslash 0.2$ ),

Finally, a fuzzy sequential machine $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \mathrm{~L}_{1}, \zeta_{1}, \mathrm{~T}_{1}\right)$ is obtained where
(a) $\mathrm{C}_{1}=\{$ Damma $\}$,
(b) $\mathrm{Q}_{1}=\left\{\mathrm{q}_{1 \mathrm{i}}, \mathrm{i}=0,1, \ldots, 7\right\}$,
(c) $\Sigma_{1}=\left\{\sigma_{1 \mathrm{~b}} \mathrm{i}=1,2, \ldots, 7\right\}, \sigma_{11}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right), \sigma_{12}=\star, \sigma_{13}=O(1), \sigma_{14}$ $=\star, \sigma_{15}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right), \sigma_{16}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right)$, and $\sigma_{17}=$ $\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right)$,
(d) $\zeta_{1}$ is the next-state mapping function $\zeta_{1}: Q_{1} \times \Sigma_{1} \rightarrow Q_{1}$, where $\zeta_{1}\left(q_{10}, \sigma_{11}\right)=q_{11}$, $\zeta_{1}\left(q_{11}, \sigma_{11}\right)=q_{11}, \zeta_{1}\left(q_{11}, \sigma_{12}\right)=q_{12}, \zeta_{1}\left(q_{12}, \sigma_{13}\right)=q_{13}, \zeta_{1}\left(q_{13}, \sigma_{14}\right)=q_{14}, \zeta_{1}\left(q_{14}, \sigma_{15}\right)$ $=q_{15}, \zeta_{1}\left(q_{15}, \sigma_{15}\right)=q_{15}, \zeta_{1}\left(q_{15}, \sigma_{16}\right)=q_{16} \zeta_{1}\left(q_{15}, \sigma_{17}\right)=q_{17}, \zeta_{1}\left(q_{16}, \sigma_{17}\right)=q_{17}, \zeta_{1}\left(q_{16}\right.$, $\left.\sigma_{16}\right)=q_{16}$, and $\zeta_{1}\left(q_{17}, \sigma_{17}\right)=q_{17}$,
(e) $T_{1}=\left\{\mathrm{q}_{17}\right\}$, where the fuzzy features which are embedded in $\mathrm{q}_{17}$ were explained in Step 7 of this example.

### 6.3. MERGING OF FUZZY SEQUENTIAL MACHINES

As shown in Figure 6.1, another necessary step in the learning process is to merge two fuzzy sequential machines into one machine. Since there can be a large number of stroke classes, it is impractical to create an independent machine for each stroke with the fact that these machines may have many common states. Instead, a set of machines can be merged into one machine with lesser number of states, which saves in recognition time and memory requirements. This section presents an algorithm to merge two fuzzy sequential machines.

### 6.3.1. Definitions

(a) A fuzzy sequential machine, $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, which recognizes a set of stroke classes, is said to be nondeterministic if there exists at least one state $q_{i} \in Q$ and a state entrance qualifier $\sigma_{j} \in \Sigma$ such that $\zeta\left(q_{i}, \sigma_{j}\right)$ is multivalued. By a multivalued $\zeta\left(q_{i}, \sigma_{j}\right)$ it is meant that for the same state-state entrance qualifier combination, $\left(q_{j} \sigma_{j}\right)$, there is more than one next state. For example, in the machine of Figure 6.4, we have $\zeta\left(q_{1}, \sigma_{2}\right)=q_{2}$ and $\zeta\left(q_{1}, \sigma_{2}\right)=q_{3}$, i.e., for the same state entrance qualifier, $\sigma_{2}, q_{1}$ has two next states, $q_{2}$ and $q_{3}$. Hence, it is said that this machine is nondeterministic which is one kind of machines that we shall form as a necessary step in our approach to merge fuzzy sequential machines.
(b) The transition tree is one graphical way to represent a sequential machine [75, 77], which will be used in the merging algorithm as a step toward obtaining a deterministic machine from a nondeterministic machine. To illustrate how a transition tree is formed, consider the fuzzy sequential machine of Figure 6.4 and its transition tree shown in Figure 6.5 (The nodes in Figure 6.5 are referenced by the


Figure 6.4. A nondeterministic fuzzy sequential machine.


Figure 6.5. The transition tree of the fuzzy sequential machine of Figure 6.4.
nearby numbers). First, the root node, node 0 , which contains the starting state, $q_{0}$, is generated. There is one entrance qualifier, $\sigma_{1}$, which causes a transition from $\mathrm{q}_{0}$ to $q_{1}$. Thus a new node, node 1 , containing $q_{1}$, is added as a son of node 0 . The arc from node 0 to node 1 is labelled with the state entrance qualifier, $\sigma_{1}$. The root node, node 0 , constitutes level 0 of the tree. Now, consider node 1 , which contains $\mathrm{q}_{1}$. There are two state entrance qualifiers, $\sigma_{1}$ and $\sigma_{2}$, which cause transitions from
$\mathrm{q}_{1}$ to other states. $\sigma_{1}$ causes a transition from $\mathrm{q}_{1}$ to itself. Thus, a new node, node 2 , which contains $q_{1}$, is added as a son of node 1 . The arc between node 1 and node 2 is labelled with $\sigma_{1}, \sigma_{2}$ causes a transition from $q_{1}$ to either $q_{2}$ or $q_{3}$. Thus, a new node, node 3 , which contains both $\mathrm{q}_{2}$ and $\mathrm{q}_{3}$, is added as a son of node 1 . The arc between node 1 and node 3 is labelled with $\sigma_{2}$. Now, node 3 contains the set of states $\left\{q_{2}, q_{3}\right\}$. For these two states, there are two state entrance qualifiers, $\sigma_{2}$ and $\sigma_{3}$, which cause transitions to other states. $\sigma_{2}$ causes a transition from $q_{2}$ to itself or from $\mathrm{q}_{3}$ to itself. Thus, a new node, node 4 , containing both $\mathrm{q}_{2}$ and $\mathrm{q}_{3}$, is added as a son of node 3. The arc between node 3 and node 4 is labelled with $\sigma_{2} . \sigma_{3}$ causes a transition from $q_{3}$ to $q_{4}$. Thus, a new node, node 5 , containing $q_{4}$, is added as a son of node 3. The arc between node 3 and node 5 is labelled with $\sigma_{3}$. Node 5 contains $\mathrm{q}_{4}$. There is one state entrance qualifier, $\sigma_{3}$, which causes a transition from $\mathrm{q}_{4}$ to itself. Thus, a new node, node 6 , containing $\mathrm{q}_{4}$, is added as a son of node 5 , which completes the transition tree. Notice that a state combination does not appear in more than two levels of the transition tree.

### 6.3.2. Algorithm to Merge Two Fuzzy Sequential Machines

The merging algorithm consists of the following three basic steps:

1. Form one nondeterministic machine, see Definition 6.3.1(a), from the two machines to be merged. The ambiguity in the formed machine, i.e., having a multivalued $\zeta\left(q_{i}\right.$, $\sigma_{\mathrm{j}}$ ), is removed by the following two more steps.
2. Construct the transition tree, see Definition 6.3.1(b), of the nondeterministic machine formed in (1).
3. The transition diagram of a deterministic machine is obtained from the transition tree. The final machine recognizes the strokes of the two original machines. The detailed algorithm now follows.

## Algorithm 6.3

Use: $\quad$ To merge two fuzzy sequential machines
Input: Two deterministic fuzzy sequential machines to be merged: $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \boldsymbol{L}_{1}\right.$, . $\left.\zeta_{1}, \mathrm{~T}_{1}\right)$ and $\mathrm{FM}_{2}=\left(\mathrm{C}_{2}, \mathrm{Q}_{2}, \mathrm{~L}_{2}, \zeta_{2}, \mathrm{~T}_{2}\right)$

Output: Deterministic fuzzy sequential machine $\mathrm{FM}=(\mathbf{C}, \mathbf{Q}, \boldsymbol{\Sigma}, \zeta, \mathrm{T})$, which recognizes the strokes of $\mathrm{FM}_{1}$ and $\mathrm{FM}_{2}$

## Procedure:

Step 1. Forming of a nondeterministic fuzzy sequential machine
Let $Q_{10}$ and $Q_{20}$ be the sets of next states of $q_{10}$ and $q_{20}$, respectively. Form a nondeterministic machine from $\mathrm{FM}_{1}$ and $\mathrm{FM}_{2}$ as follows:
(a) Remove the starting states $\mathrm{q}_{10}$ and $\mathrm{q}_{20}$, and the arcs emitting from them.
(b) Create a common starting state, $\mathrm{q}_{0}$.
(c) For each state $\mathrm{q}_{1 \mathrm{i}} \in \mathrm{Q}_{10}$ add a directed arc to point from $\mathrm{q}_{0}$ to that state. This arc is labelled with the state entrance qualifier of $\mathrm{q}_{\mathrm{li}}$ Similarly, for each state $\mathrm{q}_{2 j} \in \mathrm{Q}_{20}$, add a directed arc to point from $\mathrm{q}_{0}$ to that state. This arc is labelled with the state entrance qualifier of $q_{2 j}$. The result is that it generally happens that more than one arc emitting from $\mathrm{q}_{0}$ will have the same input symbol, $\sigma$, which makes $\zeta\left(\mathrm{q}_{0}, \sigma\right)$ a multivalued function. Accordingly, what is obtained is a nondeterministic fuzzy sequential machine $\mathrm{FM}_{3}=\left(\mathrm{C}_{3}, \mathrm{Q}_{3}, \Sigma_{3}, \zeta_{3}, T_{3}\right)$, where $\mathrm{C}_{3}=\mathrm{C}_{1} \cup \mathrm{C}_{2}, \mathrm{Q}_{3}=\left\{\mathrm{q}_{0}\right\} \cup \mathrm{Q}_{1}$ $\cup Q_{2}-\left\{q_{10}, q_{20}\right\}, \Sigma_{3}=\Sigma_{1} \cup \Sigma_{2}, \zeta_{3}$ is the next-state mapping function $\zeta_{3}: Q_{3} \times \Sigma_{3}$ $-Q_{3}$, and $T_{3}=T_{1} \cup T_{2}$.
Step 2. Constructing the transition tree for the machine $\mathrm{FM}_{3}$
(a) Generate the root node of the tree which has the state $\mathrm{q}_{0}$.
(b) For each generated node, d, starting from the root node, let $S=\left\{q_{1 i} q_{l j} \ldots, q_{i 2}, q_{2 b}\right.$ $\ldots\}$ be the set of states that was inserted in node $d$. For each input $\sigma_{1} \in \Sigma_{3}$, find the set of next states, $S_{i}$, of the set $S$. Insert $S_{i}$ in a new node that is a son of node $d$. This step, (b), is repeated for each newly generated node such that any state combination does not appear in more than two levels of the tree. The link from a node, $\mathrm{d}_{\mathrm{p}}$ to another node, $\mathrm{d}_{\mathrm{j}}$, is labelled with the state entrance qualifier that causes the transition from the state combination of the former node to that of the later.
Step 3. Obtain the transition diagram which represents a deterministic machine, FM, from the transition tree:
(a) Create the starting state $\mathrm{q}_{0}$, of the transition diagram, which corresponds to the root node of the transition tree.
(b) Scan the transition tree, level by level, starting from level 1 so that a single state is
created for each combination of states that exists in a node, d , as follows:

1. If the node, $d$, has a state combination for which no state was created, then create a new state, $q_{i}$. Otherwise, a state, $q_{i}$, that corresponds to the state combination of $d$ already exists.
2. Add an arc to point from the state that corresponds to the father node of $d$ to $q_{i}$. This arc is labelled with the state entrance qualifier which caused the transition from the father node to the node d.
3. If there is a terminal state $q_{1 j}$ or $q_{2 x}$ in the state combination of the node $d$ then copy the stroke classes along with their fuzzy features which are embedded in $q_{1 j}$ or $q_{2 k}$ to the new state $q_{i}$, in which case $q_{i}$ is identified as a terminal state.

At the end of this step, a transition diagram is obtained which represents a deterministic fuzzy sequential machine, FM, which is capable of recognizing both sets of strokes which were recognized by $\mathrm{FM}_{1}$ or $\mathrm{FM}_{2}$.

### 6.3.3. Example

Figure 6.6(a) shows a graph of Arabic numeral "r". This graph consists of two primitives: $\Psi_{1}=l(1,2)$ and $\Psi_{2}=l(2,3)$. The machine, $\mathrm{FM}_{1}$, which is shown in Figure 6.3(c), is unable to recognize this stroke since the last level of the acceptance tree is less than 2, the number of primitives. Thus, to learn this stroke:

- A deterministic machine, $\mathrm{FM}_{2}=\left(\mathrm{C}_{2}, \mathrm{Q}_{2}, \Sigma_{2}, \zeta_{2}, T_{2}\right)$, is created for the stroke, where $C_{2}=\{$ Arabic numeral "r" $\}, Q_{2}=\left\{q_{2 i}, i=0,1,2,3,4,5\right\}, \Sigma_{2}=\left\{\sigma_{2}, i=1,2,3,4\right.$, $5\}, \zeta_{2}$ is the next-state mapping function $\zeta_{2}: Q_{2} \times \Sigma_{2} \rightarrow Q_{2}$, which can be read from Figure 6.6(b), and $T_{2}=\left\{q_{2 s}\right\}$. The state entrance qualifiers are: $\sigma_{21}=\left(195^{\circ} / 30^{\circ}\right.$; $\left.225^{\circ} \backslash 30^{\circ}\right), \sigma_{22}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right), \sigma_{23}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right), \sigma_{24}=$ $\left(285^{\circ} / 30^{\circ} ; 315^{\circ} \backslash 30^{\circ}\right)$, and $\sigma_{25}=\left(315^{\circ} / 30^{\circ} ; 345^{\circ} \backslash 30^{\circ}\right)$. In the terminal state, $\mathrm{q}_{25}$, five fuzzy features are embedded. The first feature is of DP type: $\mathrm{F}_{1}=(0.8 / 0.2 ; 1.0$ \0.2). The fuzzy feature of height to height-plus-width ratio is $\mathrm{F}_{2}=(0.4 / 0.2 ; 0.6$ \0.2). There are two fuzzy features which relate the heights of the right and left ends to the total height of the stroke. These fuzzy features are $F_{3}=(0.0 / 0.0 ; 0.2$ $\backslash 0.2)$ and $F_{4}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$. The last feature is $F_{5}=(0.0 / 0.0 ; 0.2 \backslash 0.2)$ which is a measure of the stroke's curvature.

(a)

(b)

Figure 6.6 (a) A graph of Arabic numeral " $\ulcorner$ ", and (b) the corresponding fuzzy sequential machine.

- The machine, $\mathrm{FM}_{1}$, is merged with the machine, $\mathrm{FM}_{2}$, described above. The merging process is detailed below.

Step 1. Forming of a nondeterministic fuzzy sequential machine
The sets of next states of $q_{10}$ and $q_{20}$ are $Q_{10}=\left\{q_{11}\right\}$ and $Q_{20}=\left\{q_{21}\right\}$, respectively:
(a) Delete the starting states $\mathrm{q}_{10}$ and $\mathrm{q}_{20}$ and the arcs emitting from them
(b) Create a common starting state, $\mathrm{q}_{0}$
(c) Since $q_{11} \in Q_{10}$, an arc, which is labelled with $\sigma_{11}$, is added to point from $q_{0}$ to $q_{11}$. Similarly, another arc, labelled with $\sigma_{21}$, is added to point from $q_{0}$ to $q_{21}$. Note that $\sigma_{11}=\sigma_{21}=\sigma$ which makes $\zeta\left(q_{0}, \sigma\right)$ a multivalued function. The result is the transition diagram shown in Figure 6.7 which represents a nondeterministic machine, $\mathrm{FM}_{3}$.


Figure 6.7. A nondeterministic machine obtained from the machines of Figures 6.3(c) and 6.6(b).

Step 2. Constructing the transition tree for the machine $\mathrm{FM}_{3}$
While reading this step refer to Figure 6.8, which displays the whole transition tree. The nodes are referenced by the nearby numbers.
(a) Generate the root node, node 0 , of the tree which has the state $q_{0}$.
(b) For node 0 , which has $q_{0}$, the state entrance qualifier, $\sigma_{11}=\sigma_{21}$, causes a transition from $q_{0}$ to the set of next states $S=\left\{q_{11}, q_{21}\right\}$. Thus, a new node, node 1 , which is a son of node 0 with a state combination equal to $S$ is, added. The arc from node 0 to node 1 is labelled with $\sigma_{11}=\sigma_{21}$. Node 1 , in turn, has $\left\{q_{11}, q_{21}\right\},\left\{q_{22}\right\},\left\{q_{23}\right\}$, $\left\{q_{24}\right\},\left\{q_{2 s}\right\}$, and $\left\{q_{12}\right\}$ as the sets of next states for the state entrance qualifiers $\sigma_{11}$ $=\sigma_{21}, \sigma_{22}, \sigma_{23}, \sigma_{24}, \sigma_{25}$, and $\sigma_{12}$, respectively. Thus a new node is created for each set of states, nodes 2 to 7 . The arcs from node 1 to nodes $2,3, \ldots, 7$, are labelled with $\sigma_{11}=\sigma_{21}, \sigma_{23} \sigma_{23}, \sigma_{24}, \sigma_{23}$, and $\sigma_{12}$, respectively. This process is repeated for newly generated nodes provided that a state combination does not appear in more than two levels of the tree. The final transition tree is shown in Figure 6.8.


Figure 6.8. The transition tree for the machine of Figure 6.7.

Step 3. To obtain the transition diagram from the transition tree:
(a) Create the starting state, $\mathrm{q}_{0}$, of the transition diagram which corresponds to the root node of the transition tree.
(b) Scanning the tree starting from level 1 , first we face node 1:

1. Node 1 contains the state combination $\left\{q_{11}, q_{21}\right\}$ for which no state was created. Thus, a new state, $q_{1}$, is generated.
2. An arc is added to point from state $q_{0}$, which corresponds to node 0 , the father node of node 1 , to state $q_{1}$ which corresponds to node 1 . This arc is labelled with $\sigma_{1}=\sigma_{11}=\sigma_{21}$.


Figure 6.9. A deterministic fuzzy sequential machine obtained by merging the machines of Figures 6.3(c) and 6.6(b).

Next, we face node 2:

1. Node 2 contains the state combination $\left\{q_{11}, q_{21}\right\}$ for which a state, $q_{1}$, was created.
2. An arc is added to point from state $q_{1}$, which corresponds to node 1 , the father node of node 2 , to state $q_{1}$ which also corresponds to node 2 . This arc is labelled with $\sigma_{1}=\sigma_{11}=\sigma_{21}$.
This process is repeated for the other nodes which results in 11 unique states, excluding $q_{0}$. Regarding node 6 , it contains a terminal state, $q_{25}$. Thus, the stroke class, Arabic numeral " $\left\ulcorner\right.$ ", and its fuzzy features, which are embedded in $q_{25}$, are copied to state, $q_{11}$ of Figure 6.9, which corresponds to node 6. Hence, $q_{11}$ of Figure 6.9 becomes a terminal state. Node 23 , which contains the terminal state $q_{17}$, is handled similarly. The completed transition diagram is shown in Figure 6.9 which represents a deterministic machine $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, where $\mathrm{C}=\{$ Damma, Arabic numeral " r " $\}, Q=\left\{q_{i} i=0,1, \ldots, 11\right\}, \Sigma=\left\{\sigma_{i}, i=1,2, \ldots, 11\right\}, \zeta$ is the next-state mapping function $\zeta: Q \times \Sigma-\mathrm{Q}$, which can be read from the figure, and $T=\left\{q_{7}, q_{11}\right\}$. In the terminal state $q_{7}$, Damma strokes are recognized with the same set of fuzzy features that were embedded in state $q_{17}$ of machine $\mathrm{FM}_{1}$. Also, the
terminal state $q_{11}$ recognizes Arabic numeral "r" with those fuzzy features which were embedded in state $\mathrm{q}_{25}$ of machine $\mathrm{FM}_{2}$. The state entrance qualifiers are as follows: $\sigma_{1}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right), \sigma_{2}=\star, \sigma_{3}=O(1), \sigma_{4}=\star, \sigma_{5}=\left(255^{\circ} / 30^{\circ}\right.$; $\left.285^{\circ} \backslash 30^{\circ}\right), \sigma_{6}=\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right), \sigma_{7}=\left(195^{\circ} / 30^{\circ} ; 225^{\circ} \backslash 30^{\circ}\right), \sigma_{8}=$ $\left(225^{\circ} / 30^{\circ} ; 255^{\circ} \backslash 30^{\circ}\right), \sigma_{9}=\left(255^{\circ} / 30^{\circ} ; 285^{\circ} \backslash 30^{\circ}\right), \sigma_{10}=\left(285^{\circ} / 30^{\circ} ; 315^{\circ} \backslash 30^{\circ}\right)$, and $\sigma_{11}=\left(315^{\circ} / 30^{\circ} ; 345^{\circ} \backslash 30^{\circ}\right)$.

### 6.4. MODIFICATION OF A FUZZY SEQUENTIAL MACHINE

If $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, T_{1}\right)$ is the machine that was used when recognition was tried but failed, then there are cases in which an unrecognized stroke can be learned by incorporating some modifications in $\mathrm{FM}_{1}$ so that it becomes capable of recognizing strokes belonging to the unknown stroke's class. These cases occur if the last level of the stroke's acceptance tree equals, $m$, the number of primitives in the sequence, $\Psi$, as follows:
(a) The stroke sequence could terminate at one or more terminal states such that the remaining credit of the stroke did not fall below $\mathrm{THR}_{1}$, but could not be recognized due to severe deviation of the stroke's features from the fuzzy features of the classes embedded in any of these terminal states, i.e., the acceptance degree of the stroke's features is less than $\mathrm{THR}_{2}$. Thus, in the following algorithm, a possibility will be investigated to add a new class, with its fuzzy features, in one of these terminal states.
(b) And / or the stroke sequence terminated in one or more states, NT, which are not terminal states and such that the remaining credit of the stroke is not less than $\mathrm{THR}_{1}$. Here, there are two cases:

1. Some states $\mathrm{NT}_{s} \subseteq$ NT were reached by considering some primitives to be spurious.
2. The remaining states $\mathrm{NT}_{\mathrm{n}}=\mathrm{NT}-\mathrm{NT}_{4}$ were reached by considering all primitives to be non-spurious. We prefer the states in $\mathrm{NT}_{\mathrm{n}}$ on those in $\mathrm{NT}_{v}$, although in the later the remaining credit of the stroke may exceed that in the former. The rationale behind this preference is that learning a whole stroke is safer than considering some primitives spurious and learning just part of it.

Thus, in the following algorithm, a possibility can be searched to change one of the states in $\mathrm{NT}_{\mathrm{n}}$ into a terminal state, i.e., to append it to the set T , and to insert in it the class of the stroke with its fuzzy features.
Formal description of the algorithm which learns a stroke by modifying a fuzzy sequential machine follows.

## Algorithm 6.4

Use: To learn a stroke by modifying a fuzzy sequential machine
Input: 1. Sequence of primitives, $\Psi$, of the stroke to be learned
2. Reduced graph of the stroke to be learned
3. Acceptance tree of the stroke which is unrecognized by Algorithm 5.3
4. Fuzzy sequential machine, $\mathrm{FM}_{1}$, of strokes which are already learned

Output: New fuzzy sequential machine, FM, which can recognize the underlying stroke and previously learned strokes

## Procedure:

Step 1. Finding the path, in the acceptance tree, which has the maximum remaining credit
(a) Retain the set of leaf nodes, D , of level m . For each node, $\mathrm{d} \in \mathrm{D}$, where the node d contains the remaining credit, $\omega$, of the stroke, do:

1. Trace the path from $d$ to its father node in level 1 . Extract those primitives which were counted non-spurious. Let these primitives be represented, from top to bottom of the tree, by the new sequence $\Psi '$, the last primitive of which is accepted by some state, $q$.
2. Check if the sequence $\Psi^{\prime}$ is valid. A sequence of primitives is considered valid if the following two conditions are satisfied:
i. the inherent properties, mentioned in Section 4.2, are satisfied,
ii. if $q \notin T_{1}$, then the sequence length $m^{\prime}$ must be equal to the original sequence length, $m$.
(b) Retain the sequence, $\Psi^{\prime}$, which has the maximum credit, $\omega$, and denote it $\Psi^{*}$.

Step 2. Feature extraction of the stroke
(a) Find the explicit relationships between primitives of $\Psi^{*}$, see Section 4.3. Find the
global features of the stroke the sequence of which is $\Psi^{*}$, see Section 4.4, and append them to the explicit relationships.
(b) Fuzzify every feature, $\mathrm{f}_{\mathrm{i}}$, found in (a), of the stroke by transforming it into an $\mathrm{s} / \mathrm{z}$ number as follows:

1. Set both the left and right bandwidths to $\beta_{f}$.
2. The range from 0.0 to 1.0 is divided into $n$ equal intervals. The left and right peak points, $p_{1}$ and $p_{2}$, are taken to be equal to the limits of the interval in which the value $f_{i}$ lies.

Suitable values of $\beta_{\mathrm{f}}$ and n were found to be 0.2 and 5 , respectively.
Step 3. State modification
Let $\mathrm{q}^{*}$ be the state which accepted the last primitive of $\Psi^{*}$ :
(a) If $q^{*} \in T_{1}$, where $T_{1}$ is the set of terminal states of $\mathrm{FM}_{1}$, then add a new stroke class, $\mathrm{c}_{\text {nows }}$ in state $\mathrm{q}^{*}$ with the fuzzy features computed in Step 2. A new machine, $\mathrm{FM}=$ $(C, Q, \Sigma, \zeta, T)$, is obtained where $C=\left\{C_{\text {new }}\right\} \cup C_{1}, Q=Q_{1}, \Sigma=\Sigma_{1}, \zeta=\zeta_{1}$, and $T$ $=\mathrm{T}_{1}$.
(b) If $q^{*} \notin$ to $T_{1}$, then add $q^{*}$ to the set of terminal states. Insert a stroke class, $c_{\text {new }}$, in $q^{*}$ with the fuzzy features computed in Step 2. A new machine, $F M=(C, Q, \Sigma, \zeta$, $T$ ), is obtained which is similar to $\mathrm{FM}_{1}$ with the exception that $\mathrm{C}=\left\{\mathrm{c}_{\text {new }}\right\} \cup \mathrm{C}_{1}$ and $T=\left\{q^{*}\right\} \cup T_{1}$.

### 6.4.1. Example

Figure 6.10(a) shows a stroke of Arabic numeral "r". It has two components: $\Psi_{1}=$ $l(1,2)$ and $\Psi_{2}=l(2,3)$. The machine, $\mathrm{FM}_{1}$, of Figure 6.9 is used to recognize this stroke. The recognition algorithm produced the acceptance tree of Figure 6.10(b). Nodes 3, 4, and 5 have levels which are equal to 2 , the number of primitives. These nodes contain the states $\mathrm{q}_{8}, \mathrm{q}_{s}$, and $\mathrm{q}_{10}$, respectively. However, none of these states is a terminal one. Thus the stroke is unknown. Since, the last level of the acceptance tree is 2 which equals the number of primitives, $m$, the stroke can be learned by incorporating some modifications to $\mathrm{FM}_{1}$ as follows:


Figure 6.10. (a) A graph of Arabic numeral " $\ulcorner$ ", and (b) its acceptance tree.

Step 1. Finding the path, in the acceptance tree, which has the maximum remaining credit
(a) We retain the set of leaf nodes, $D$, of level 2. Here $D=\{$ node 3, node 4, node 5$\}$. First we trace the path from node 3 to its father node, node 2 , in level 1 . A sequence $\Psi_{1}^{\prime}=\left\{\Psi_{1}, \Psi_{2}\right\}$ is obtained which satisfies the inherent properties of sequences of primitives and its last primitive, $\Psi_{2}$, is accepted by $q_{8}$ which is not a terminal state of the machine under consideration; hence, it is a valid sequence. Similarly, starting from nodes 4 and 5 , two other valid sequences, $\Psi^{\prime}{ }_{2}=\left\{\Psi_{1}, \Psi_{2}\right\}$ and $\Psi^{\prime}{ }_{3}=\left\{\Psi_{1}, \Psi_{2}\right\}$, are obtained.
(b) The sequence $\Psi^{\prime}{ }_{2}$ has the maximum remaining credit, 1.00 ; hence, it is retained.

Step 2. Feature extraction of the stroke
(a) Notice that $\Psi{ }^{\prime}{ }_{2}$ does not have features of explicit relationships between primitives since it does not contain connected loop and vertex primitives. Five global features of the sequence $\Psi^{\prime}{ }_{2}$ are found: a DP feature $f_{1}=1.0$, a height to height-plus-width ratio feature $f_{2}=0.70$, features of heights of right and left ends relative to the height of the stroke $f_{3}=0.00$ and $f_{4}=1.0$, and the feature of curvature $f_{5}=0.46$.
(b) The above features are fuzziffied to obtain the following fuzzy features $\mathrm{F}_{1}=(0.80$ $/ 0.20 ; 1.00 \backslash 0.20), \mathrm{F}_{2}=(0.60 / 0.20 ; 0.80 \backslash 0.20), \mathrm{F}_{3}=(0.00 / 0.00 ; 0.20 \backslash 0.20)$, $\mathrm{F}_{4}=(0.80 / 0.20 ; 1.00 \backslash 0.20)$; and $\mathrm{F}_{5}=(0.40 / 0.20 ; 0.60 \backslash 0.20)$, where $\mathrm{F}_{1}$ is of a DP type, $F_{2}$ is the height to height-plus-width ratio fuzzy feature, $F_{3}$ and $F_{4}$ are the
fuzzy features of heights of right and left ends relative to the height of the stroke, and $F_{5}$ is the curvature fuzzy feature.

## Step 3. State modification

The last primitive, $\Psi_{2}$, of $\Psi_{2}^{\prime}$ is accepted by state $q_{9}$, which is not a terminal state in the machine $\mathrm{FM}_{1}$. Thus, $\mathrm{q}_{\mathrm{g}}$ is added to the set of terminal states and an Arabic numeral $" r "$ class is inserted in $q_{9}$ with the five fuzzy features computed in Step 2, above. Finally, a new machine $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, is obtained which is similar to $\mathrm{FM}_{1}$ with the exception that $C=\{$ Arabic numeral "r" $\} \cup C_{1}$ and $T=\left\{q_{q}\right\} \cup T_{1}$. Notice that $C=C_{1}$ since $\mathrm{C}_{1}$ already contains the class of Arabic numeral " r ".

## SUMMARY

In this chapter, the learning component of the IACR system was presented. The learning algorithm determines how to learn an unrecognized stroke depending on the last level of the acceptance tree of an unrecognized stroke, which is passed by the recognition algorithm, as follows:
(a) If the last level of the acceptance tree is less than the number of primitives in the stroke, then a fuzzy sequential machine is generated from the input stroke. This machine can recognize the input stroke and variants of it. The generated machine and the old machine, which was used in recognition but failed to recognize the stroke, are merged into a new single machine. The new machine can recognize the input stroke and strokes of the old machine and variants of these strokes.
(b) If the last level of the acceptance tree equals the number of primitives in the stroke, then some modifications are introduced into the machine which was used in recognition, as this is sufficient to learn the stroke. The output is a modified fuzzy sequential machine which can recognize the input stroke and strokes of the old machine and variants of them.

## Experimentation



## OVERVIEW

In this chapter, experimental results of the IACR system are reported. An explanation of how the character set under study was chosen is presented. A description of how the data of the learning and testing stages were acquired is given. The learning stage is described, For the testing stage, the performance of the system in terms of recognition, rejection, and error rates, and speed, is presented. Causes of rejection and error are analyzed.

### 7.1. CHOOSING THE CHARACTER SET

The character set under study is shown in Figure 7.1 which is, from right to left:
 strokes and special characters: Hamza, Madda, Shadda, slash, minus sign, plus sign, right and left parenthesis, comma, and Damma. This character set was selected as specified below:
(a) The IACR system is designed to recognise single-stroke characters, i.e., each character consists of a single connected component, which applies to every character in this set.
(b) Samples of these characters produce a relatively small number of primitives compared to some other characters. This results in smaller acceptance trees and a smaller stroke fuzzy sequential machine. Consequently, faster recognition is

$$
96()+-/ \omega \sim \varepsilon 9 \wedge \vee \neg 0 \& \mu\ulcorner 1 .
$$

Figure 7.1. The set of stroke classes used in the IACR system, from right to left: Arabic numerals zero to nine, Hamza, Madda, Shadda, slash, minus sign, plus sign, right parenthesis, left parenthesis, comma, and Damma.
obtained and less memory is needed.
There are other characters for which (a) and (b), above, also hold true. However, only the shown characters are used since at this stage we are testing the efficiency of this new approach of character recognition although we recognize that a wider character set gives a more accurate assessment of the system.

### 7.2. DATA ACQUISITION

In both the learning and testing stages, the subjects were asked to write one line of each character of the set under study. There was no restriction on pen type, ink type, or ink colour. Subjects were asked to avoid generating blobs as possible as they can since the IACR system was not designed to deal with such a phenomenon. More details about how data were acquired for the learning and testing stages is given below:
(a) Learning: Initially, 6061 unnormalized handwritten strokes written by five subjects were collected. Each subject was given two A4 size blank sheets each on top of another guiding sheet, see Appendix A, Figure A.1, with twenty black horizontal lines which work as guidelines when the subject writes the characters to prevent characters of adjacent lines from touching each other. Reproductions of the images used in the learning stage are shown in Appendix A, Figures A. 2 to A. 11.
(b) Testing: It is worth mentioning here that a different set of 20 subjects other than the subjects of the learning stage provided the data set of the testing stage. Initially, 8000 unnormalized handwritten strokes written by these 20 subjects were collected. Each subject was given one A4 size blank sheet on top of another guiding sheet, see Appendix B, Figure B.1, with $20 \times 20$ empty squares. The reasons for using squares instead of just horizontal lines are:

1. With a sheet without squares as guidelines, the writing of a subject may
deteriorates as he proceeds from one end of the line to the other end since he knows that the same character is to be written. Thus, he may not mind to move and re-adjust his wrist as the pen moves. Including such guiding squares helps to achieve such wrist re-adjustment, hence, produces more natural writing.
2. In the data set of the learning stage, characters of the same line may touch each other which raises some kind of a segmentation problem which is not addressed in the current research. The squares of the guiding sheet, which is used in the testing stage, eliminate this problem.
3. In the learning stage, the size of the acquired data varies with subjects. If the same size of data is acquired by every subject then more accurate evaluation of the system is obtained. This is achieved, in the testing stage, by giving each subject a blank sheet on top of another sheet with a fixed number of empty squares per sheet.
4. In the learning stage, the number of samples varies for each character. If the same number of samples is acquired for each character then more accurate evaluation of the system is obtained. Again, this is achieved, in the testing stage, by giving each subject a black sheet on top of another sheet with a fixed number of empty squares per line.
Reproductions of the images used in the testing stage are shown in Appendix B,
Figures B. 2 to B. 21 .
Images of the data sets of the learning and testing stages were captured using an HP ScanJet scanner. The resolution used was 300 dots per inch in both the horizontal and vertical directions. The reason for selecting this value of resolution is based on our observation that under-sampled pictures, e.g., less than 300 dpi , may create disconnected images for very thin strokes which produces multi-component straight line approximations for such strokes. This is not accepted by the IACR system since it does not have the capability to handle disconnected strokes.

Table 7.1. Number of collected, discarded, and used handwritten samples for the learning stage of the IACR system.

| Subject No. | No. of <br> Collected <br> Samples | No. of Discarded Samples |  | No. of Used Samples |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | Absolute | Percentage | Absolute | Percentage |  |
| 1 | 1194 | 35 | 2.9 | 1159 | 97.1 |
| 2 | 1127 | 26 | 2.3 | 1101 | 97.7 |
| 3 | 1035 | 18 | 1.7 | 1017 | 98.3 |
| 4 | 1295 | 44 | 3.4 | 1251 | 96.6 |
| 5 | 1410 | 48 | 3.4 | 1362 | 96.6 |
| Total | 6061 | 171 | 2.8 | 5890 | 97.2 |

### 7.3. LEARNING

In the data set of the learning stage, see Figures A. 2 to A.11, notice that it in the case of comma and Damma characters samples with lost loops, i.e., blobs, were generated which, when thinned, may produce distorted straight line approximations. Thus, such samples and some other garbage samples were discarded, i.e., they were not learned. This is achieved by watching the image of each character and using human judgement to decide whether to learn it or not. Table 7.1 shows the total number of characters provided by each subject and the percentage of discarded and used characters in the learning stage. According to this table, the IACR system was trained on $97.2 \%$ of the total number of strokes, i.e., 5890 strokes.

In the learning algorithms, the core and left and right bandwidths of the fuzzy direction state entrance qualifiers were set to $30^{\circ}$. The thresholds $\mathrm{THR}_{1}$ and $\mathrm{THR}_{2}$ were both set to 0.90 . The learning stage produced a fuzzy sequential machine of 20 stroke classes, as it should be, 2705 states, and 8640 arcs.

### 7.4. TESTING

In the data set of the testing stage, see Figures B. 2 to B.21, notice that some samples are garbage which were discarded, i.e., they were not used in the testing stage. This is achieved by watching the image of each character and using human judgement to decide whether to test it or not. Table 7.2 shows the total number of characters provided

Table 7.2. Number of collected, discarded, and used handwritten samples for the testing stage of the IACR system.

| Subject No. | No. of Collected Samples | No. of Discarded Samples |  | No. of Used Samples ... |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Absolute | Percentage | Absolute | Percentage |
| 1 | 400 | 0 | 0.00 | 400 | 100 |
| 2 | 400 | 0 | 0.00 | 400 | 100 |
| 3 | 400 | 0 | 0.00 | 400 | 100 |
| 4 | 400 | 0 | 0.00 | 400 | 100 |
| 5 | 400 | 1 | 0.25 | 399 | 99.75 |
| 6 | 400 | 0 | 0.00 | 400 | 100 |
| 7 | 400 | 0 | 0.00 | 400 | 100 |
| 8 | 400 | 7 | 1.75 | 393 | 98.25 |
| 9 | 400 | 1 | 0.25 | 399 | 99.75 |
| 10 | 400 | 5 | 1.25 | 395 | 98.75 |
| 11 | 400 | 3 | 0.75 | 397 | 99.25 |
| 12 | 400 | 4 | 1.00 | 396 | 99.00 |
| 13 | 400 | 5 | 1.25 | 395 | 98.75 |
| 14 | 400 | 1 | 0.25 | 399 | 99.75 |
| 15 | 400 | 0 | 0.00 | 400 | 100 |
| 16 | 400 | 1 | 0.25 | 399 | 99.75 . |
| 17 | 400 | 1 | 0.25 | 399 | 99.75 |
| 18 | 400 | 8 | 2.00 | 392 | 98.00 |
| 19 | 400 | 0 | 0.00 | 400 | 100 |
| 20 | 400 | 0 | 0.00 | 400 | 100 |
| Total | 8000 | 37 | 0.46 | 7963 | 99.54 |

by each subject and the percentage of discarded and used characters in the testing stage. According to this table, $99.54 \%$ of the total number of strokes, i.e., 7963 strokes, were used in the testing stage.

The testing stage was performed on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM. The overall recognition, rejection, and error rates were $95.8 \%, 1.5 \%$, and $2.7 \%$, respectively. Thus, the system reliability is $95.8 /(100-$

Table 7.3. Performance of the IACR system.
These results were obtained by running the algorithms on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM.

| Subject \# | No. of Strokes | Recognition Rate \% | Rejection Rate \% | Error Rate \% | Reliability \% |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 400 | 97.8 | 1.2 | 1.0 | 99.0 |
| 2 | 400 | 97.0 | 1.3 | 1.7 | 98.3 |
| 3 | 400 | 97.3 | 1.2 | 1.5 | 98.5 |
| 4 | 400 | 97.5 | 1.0 | 1.5 | 98.5 |
| 5 | 399 | 96.5 | 1.5 | 2.0 | 98.0 |
| 6 | 400 | 97.5 | 1.5 | 1.0 | 99.0 |
| 7 | 400 | 95.0 | 1.8 | 3.2 | 96.7 |
| 8 | 393 | 93.9 | 1.8 | 4.3 | 95.6 |
| 9 | 399 | 93.7 | 2.0 | 4.3 | 95.6 |
| 10 | 395 | 95.7 | 1.3 | 3.0 | 97.0 |
| 11 | 397 | 95.7 | 1.0 | 3.3 | 96.7 |
| 12 | 396 | 92.9 | 1.8 | 5.3 | 94.6 |
| 13 | 395 | 93.2 | 1.8 | 5.0 | 94.9 |
| 14 | 399 | 94.0 | 1.5 | 4.5 | 95.4 |
| 15 | 400 | 97.5 | 1.2 | 1.3 | 98.7 |
| 16 | 399 | 95.2 | 1.5 | 3.3 | 96.6 |
| 17 | 399 | 97.0 | 1.0 | 2.0 | $98.0^{\circ}$ |
| 18 | 392 | 96.4 | 1.3 | 2.3 | 97.7 |
| 19 | 400 | 97.5 | 1.5 | 1.0 | 99.0 |
| 20 | 400 | 95.3 | 2.0 | 2.7 | 97.2 |
| Total | 7963 | 95.8 | 1.5 | 2.7 | 97.3 |

$1.5) \times 100=97.3 \%$.
Table 7.3 shows the performance rates for the 20 subjects of the testing stage. In this table it is clear that, irrespective of the subject, the recognition rate always exceeds $90.0 \%$, and the rejection and error rates don't exceed $2.0 \%$ and 5.3 , respectively.

The performance of the system puts the IACR system in a competing position with
other systems, see for example $[14,36]$ the recognition rates of which were $95.9 \%$ and $94.5 \%$, respectively. The IACR system's rejection rate which is $1.5 \%$ is very low as compared to the rejection rates of $31.22 \%$ and $22.48 \%$ of statistical and structural recognition schemes, respectively, of [78]. Its estimated error rate, which is $2.7 \%$, is larger than the error rate $2.02 \%$ of a template matching recognition technique that uses normalization and forced decision, applied to a larger set of data ( 600 samples per class) reported in [79].

These results are encouraging especially in the recognition of handwritten characters and offer much potential within the field of automatic off-line character recognition. The system is highly flexible in dealing with shape and size variations. Also, it will be more developed and extended, in the next chapters, to deal with cursive handwriting.

Table 7.4 displays the time requirement of the system for the 20 subjects of the testing stage. Preprocessing time includes smoothing, thinning, straight line approximation, and enforcement of temporal information. The average time required to preprocess a stroke is about 0.84 seconds which is very small compared to the average time of 27.94 seconds required by the recognition algorithm. In other words, the recognition algorithm hogs $97.1 \%$ of the overall recognition time from smoothing to classification. The reason behind this is that the recognition algorithm depends on building tree data structures which are associated with time-consuming calculations and path following processes. However; the recognition time can be reduced by using faster machines.

### 7.5. REJECTION AND ERROR ANALYSIS

There is one main reason for rejection which is a new unlearned style of the stroke is introduced to the system. Table 7.5 details rejection occurrence in the IACR system for the data set of the testing stage. It is noticed that the " + " stroke is the major source of rejection. The rejection rate can be simply reduced by learning more styles of handwritten strokes.

The cases of error occur in similar pairs of characters. Examples of these pairs are:
(a) Flattened ")" similar to "1", Figure 7.2(a),
(b) Flattened "(" similar to "1", Figure 7.2(b),

Table 7.4. Speed of the IACR system.
These results were obtained by running the algorithms on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM.

| Subject \# | Time (sec.) / stroke |  |  |
| :---: | :---: | :---: | :---: |
|  | Preprocessing. | Recognition | Total |
| 1 | 0.90 | 28.40 | 29.30 |
| 2 | - 0.85 | 27.35 | 28.20 |
| 3 | 0.91 | 30.07 | 30.98 |
| 4 | 0.87 | 29.62 | 30.49 |
| 5 | 0.84 | 25.18 | 26.02 |
| 6 | 0.88 | 28.43 | 29.31 |
| 7 | 0.75 | 26.21 | 26.96 |
| 8 | 0.82 | 27.54 | 28.36 |
| 9 | 0.79 | 26.83 | 27.62 |
| 10 | 0.92 | 30.15 | 31.07 |
| 11 | 0.72 | 27.46 | 28.18 |
| . 12 | 0.87 | 29.01 | 29.88 |
| 13 | 0.80 | 26.66 | 27.46 |
| 14 | 0.91 | 28.37 | 29.28 |
| 15 | 0.82 | 28.92 | 29.74 |
| 16 | 0.78 | 26.59 | 27.37 |
| 17 | 0.90 | 29.60 | 30.50 |
| 18 | 0.86 | 28.14 | 29.00 |
| 19 | 0.88 | 27.93 | 28.81 |
| 20 | 0.79 | 26.40 | 27.19 |
| Average | 0.84 | 27.94 | 28.78 |

(c) Gradually curved "r" similar to "(", or excessively bent "(" similar to "r", Figure 7.2(c),
(d) "१" with bent end similar to "s", or "s" with short bent end similar to " $\downarrow$ ", Figure 7.2(d), and
(e) Inclined "-" similar to "/", Figure 7.2(e).

Table 7.5. Distribution of rejection in the IACR system.

| Cause of rejection | Percentage |  |  |
| :--- | :---: | :---: | :---: |
| New style of numeral " $r$ " | 7.5 |  |  |
| New style of numeral " $\varepsilon$ " | 6.8 |  |  |
| New style of numeral "॰" | 1.9 |  |  |
| New style of numeral " 4 " | 8.2 |  |  |
| New style of Hamza | 15.3 |  |  |
| New style of Madda | 7.4 |  |  |
| New style of Shadda | 12.6 |  |  |
| New style of plus sign | 28.2 |  |  |
| New style of comma | 6.6 |  |  |
| New style of Damma | 5.5 |  |  |
| Total |  |  | 100 |



(d)
(e)

Figure 7.2. Causes of error: (a) flattened ")" similar to numeral "।", (b) flattened "(" similar to numeral " ", (c) gradually curved numeral "r" similar to "(" or excessively bent "(" similar to numeral "r", (d) numeral " 4 " with bent end similar to "ر" or ", with short bent end similar to numeral " $\uparrow$ ", and (e) inclined "-" similar to " $/$ ".

Table 7.6 shows the distribution of error among the above sources of error. In these cases, even the human may confront difficulty in recognizing the character. However, the number of errors can be reduced by:
(a) Reducing the core and left and right bandwidths of the fuzzy direction state entrance qualifiers. A more suitable value of these three parameters is $22.5^{\circ}$ instead of $30^{\circ}$ which was used in the learning stage. However, reducing the core and bandwidths

Table 7.6. Distribution of error in the IACR system.

| Cause of error | Percentage |
| :---: | :---: |
| Flattened ")" similar to numeral ", " | 10.8 |
| Flattened "(\% similar to numeral ", " | 20.5 |
| Gradually curved numeral "r" similar to "(", or excessively bent "(" similar to numeral "r" | 41.5 |
| Numeral "4" with bent end similar to "ง", or "ง" with short bent end similar to numeral "4" | 19.8 |
| Inclined "-" similar to "/" | 7.4 |
| Total | 100 |

implies more memory requirement and lower speed of the system.
(b) Increasing the thresholds $\mathrm{THR}_{1}$ and $\mathrm{THR}_{2}$. Suggested values are 0.95 for both instead of 0.90 which was used in the learning stage, and
(c) Using context.

## SUMMARY

A set of 20 stroke classes was used in learning and testing stages. The system was trained on 5890 unnormalized handwritten strokes written by five subjects. The learning stage produced a fuzzy sequential machine of 2705 states and 8640 arcs. A total of 7963 unnormalized handwritten strokes, written by 20 subjects other than the subjects of the learning stage, was used in the testing stage. The recognition, rejection, and error rates were $95.8 \%, 1.5 \%$, and $2.7 \%$, respectively.

# Part Three 

## Cursive

Arabic

## Script

## Recognition

System
(CASR)

In this part, a Cursive Arabic Script Recognition System (CASR) is presented. Figure III.I shows a data flow diagram, of the system. Our new contribution is represented by the filled processes / rounded rectangles in the figure, i.e., Straight Line Approximation, Enforcement of Temporal Information, Stroke Segmentation, Token Recognition, Token Learning, Learning of Token Strings, Separating Main and Secondary strokes, Extracting Lines and Ordering Strokes, Common SHape (CSH) Interpretations of Main Strokes, Character Formation, and Manipulating Redundant Secondary strokes. Straight Line Approximation and Enforcement of Temporal Information were introduced in Chapters 2 and 3, respectively. The next six chapters address the remaining nine processes. The data flow diagram of the CASR system consists of the following processes:
(a) Image Acquisition: where an off-line binary image of a handwritten page of cursive Arabic script is captured using a scanner.
(b) Smoothing: The acquired binary image of the page is smoothed. A suitable smoothing algorithm can be found in [57].
(c) Stroke Extraction: Here single component strokes are extracted, where each stroke is represented as a smoothed binary image.
(d) Thinning: $\quad$ The smoothed binary images of the strokes are thinned using the "Safe Point Thinning Algorithm," or SPTA [66].
(e) Straight Line Approximation, Chapter 2: which accepts smoothed thinned binary images of the strokes and produces two representations for each stroke. The first representation is a direct straight line approximation and the other is called a reduced graph which is also a straight line approximation with loops represented as vertices.
(f) Enforcement of Temporal Information, Chapter 3: Here temporal information of the strokes are extracted from their straight line approximations.
(g) Stroke Segmentation, Chapter 8: A cursive stroke is segmented into small parts, called tokens. Tokens are logical units which are usually larger and more suitable for cursive script than the primitives of the IACR system.
(h) Token Recognition, Chapter 9: For every input token, it finds whether it belongs

to a certain class or it could not be recognized. If a token could not be recognized, then its acceptance information, represented as a tree data structure, can be fed to the Token Learning process.
(i) Token Learning, Chapter 10: This process gets as an input the acceptance tree which is passed by the Token Recognition process, in addition to other inputs. It outputs a new token fuzzy sequential machine which can recognize the input token and tokens of the old machine and variants of these tokens.
(j) Learning of Token Strings, Chapter 11: where tokens are recombined into meaningful sets of tokens; logical token strings. Logical token strings are associated with possible interpretation and their fuzzy features.
(k) Separating Main and Secondary Strokes, Chapter 12: where strokes which can represent secondary strokes are marked. Remaining strokes are main strokes.
(l) Extracting Lines and Ordering Strokes, Chapter 12: where lines are extracted and their constituent main strokes are ordered from right to left. Secondary stroke candidates are presented to main strokes.
(m) CSH Interpretations of Main Strokes, Chapter 13: Here, all possible CSH interpretations of main strokes are enumerated and represented in a tree data structure. We call this tree Enumeration and Requirement Tree (ERT), in which information about secondary strokes required to associate CSH's to form characters is included.
(n) Character Formation, Chapter 13: where ERT's are combined with presented candidate secondary strokes to form characters. Assignment problems are formulated and solved to for this purpose. The solution which exhibits the minimum cost is selected. This process results in some redundant secondary strokes which can not be combined with CSH's to form characters.
(o) Manipulating Redundant Secondary Strokes, Chapter13: Rechmdant secondary strokes are manipulated to form some other characters which are inserted in their proper places within lines. The final result is a list of ordered lines of ordered lists of words.
A hierarchical structural chart, which corresponds to the data flow diagram of
Figure III.1, is shown in Figure III.2. Again, our new contribution is represented by the


2. Logical token strings

8: Direct straight line approximations of strokes
9: Reduced graphs of strokes
11: Tokens of strokes
12: Classes of tokens
14: List of secondary strokes
15. List of main strokes

16: Ordered lines, Ordered main strokes
17: Secondary stroke presentation
18: CSH enumeration and requirement trees
19: Words of characters, Redundant secondary strokes
20: Ordered lines of ordered lists of words

Figure III.2. Hierarchical structural chart of the CASR system.
filled modules / rectangles. As shown in this chart, the CURSIVE ARABIC SCRIPT RECOGNITION SYSTEM (CASR) coordinator manages data exchange between the following modules / sub-coordinators:
(a) Input: which returns to the CASR coordinator the token fuzzy sequential machine, learned logical token strings, and binary image of the page to be recognized or learned. Thus, the Input sub-coordinator calls the following two modules / sub-coordinators:

1. Input Data Bases: It calls the following modules:
i. Input Machine: which reads the Token Fuzzy Sequential Machine, and
ii. Input Data Base: which reads Data Base of Logical Token Strings.
2. Acquire Image: which gets a binary image of a handwritten page of cursive Arabic script.
(b) Preprocessing: This sub-coordinator receives from the CASR coordinator a binary image of the page and returns to it, for each stroke in the page, a direct
straight line approximation, reduced graph, and segmented tokens. It consists of two parts:
3. Low Level Preprocessing: It receives from the Preprocessing subcoordinator a binary image of the page and returns to it smoothed thinned binary images of the strokes. It calls three modules:
i. Smoothing: which receives a binary image of the page and returns a smoothed binary image of the whole page.
ii. Stroke Extraction: which receives a smoothed binary image of the page and returns smoothed binary images of single components (strokes) in the page.
iii. Thinning: which receives smoothed binary images of the strokes and returns their smoothed thinned binary images.
4. High Level Preprocessing: It receives from the Preprocessing subcoordinator smoothed thinned binary images of the strokes and returns to it a direct straight line approximation, reduced graph, and tokens for each stroke. It calls three modules:
i. Straight Line Approximation, Chapter 2: which receives smoothed thinned binary images of the strokes and returns a direct straight line approximation and reduced graph for each stroke.
ii. Enforcement of Temporal Information, Chapter 3: which receives direct straight line approximations and reduced graphs of the strokes and returns temporal information of each stroke.
iii. Stroke Segmentation, Chapter 8: which receives reduced graphs and temporal information of the strokes and returns the segmented tokens for each stroke.
(c) Recognition: This sub-coordinator receives from the CASR coordinator a token fuzzy sequential machine, learned logical token strings, direct straight line approximations, reduced graphs, and tokens of the strokes. It returns to the CASR coordinator classes of tokens, acceptance trees of unrecognized tokens, and ordered lines of ordered lists of words. The Recognition sub-coordinator consists of the following:
5. Token Recognition, Chapter 9: It receives from the Recognition subcoordinator a token fuzzy sequential machine, reduced graphs, and tokens of the strokes. It returns classes of tokens and acceptance trees of unrecognized tokens.
6. Line Extraction and Word Formation: It receives from the Recognition coordinator learned logical token strings, direct straight line approximations, reduced graphs, tokens of the strokes, and classes of tokens. It returns ordered lines of ordered lists of words. The Line Extraction and Word Formation sub-coordinator consists of the following sub-coordinators:
i. Line Extraction and Stroke Ordering, Chapter 12: which receives learned logical token strings, direct straight line approximations, reduced graphs, tokens of the strokes, and classes of tokens. It returns a list of main strokes, ordered lines, ordered main strokes, secondary stroke presentation, in which secondary strokes are presented to main strokes. It calls the following modules:

- Separating Main and Secondary Strokes: It receives learned logical token strings, reduced graphs, tokens of the strokes, and classes of tokens. It returns list of secondary strokes and list of main strokes.
- Extracting Lines and Ordering Strokes: It receives direct straight line approximations of the strokes, list of secondary strokes, and list of main strokes. It returns ordered lines, ordered main strokes, and secondary stroke presentation.
ii. Word Formation, Chapter 13: which receives learned logical token strings, reduced graphs, tokens of strokes, classes of tokens, list of main strokes, ordered lines, ordered main strokes, and secondary stroke presentation. It returns ordered lines of ordered lists of words. It calls the following modules:
- CSH Interpretations of Main Strokes: It receives learned logical token strings, reduced graphs, tokens of the strokes, classes of tokens, and list of main strokes. It returns CSH
enumerations and requirement trees.
- Character Formation: It receives reduced graphs of strokes, secondary stroke presentation, and CSH enumeration and requirement trees. It returns words of characters and redundant secondary strokes which could be combined with CSH's to form meaningful characters.
- Manipulating Redundant Secondary Strokes: It receives ordered lines, ordered main strokes, words of characters, and redundant secondary strokes. It returns ordered lines of ordered lists of words.
(d) Learning: This sub-coordinator receives a token fuzzy sequential machine, learned logical token strings, reduced graphs, tokens of strokes, classes of tokens, and acceptance trees of unrecognized tokens. It returns a new token fuzzy sequential machine which can recognize the input tokens and tokens of the old machine and variants of them, and a new data base of logical token strings. It calls the following modules:

1. Token Learning, Chapter 10: It receives a token fuzzy sequential machine which was used in recognition but failed to recognize the tokens, reduced graphs, tokens of strokes, and acceptance trees of unrecognized tokens. It returns a new fuzzy sequential machine which can recognize the input tokens and tokens of the old machine and variants of them.
2. Learning of Token Strings, Chapter 11: It receives learned logical token strings, reduced graphs, tokens of strokes, and classes of tokens. It returns a new data base of logical token strings.
(e) Output: which receives ordered lines of ordered lists of words, a new token fuzzy sequential machine, and new data base of logical token strings. Thus; the Output sub-coordinator consists of the following:
3. Output Results: It outputs the recognized text of the page in the form of ordered lines of ordered lists of words.
4. Output Data Bases: It consists of two modules:
i. Output Machine: which outputs the new token fuzzy sequential
machine which resulted from token learning
ii. Output Data Base: which outputs the new data base of logical token strings.
Experimental results and performance of the CASR system are reported in Chapter 14.

## Stroke

## Segmentation



## OVERVIEW

In this chapter, a cursive stroke is segmented into small parts, called tokens, Section 8.1. Tokens are logical units usually larger and more suitable for cursive script than the primitives of the IACR system. Tokens have features which are explained in Section 8.2. The segmentation process, which is shown in the data flow diagram of Figure 8.1, consists of the following two subprocesses:

1. Marking Token Boundaries: which accepts as inputs the reduced graphs and temporal information of strokes. Here, some links are marked and used as delimiters between tokens.
2. Token Extracting: This subprocess also accepts as inputs the reduced graphs and temporal information of strokes in addition to the marking information produced by the previous subprocess. For each stroke, the output is a sequence of temporally ordered tokens represented as sequences of vertices.

### 8.1. ALGORITHM FOR STROKE SEGMENTATION

The cursive nature of Arabic text makes it difficult to segment a subword directly into characters. Rather, a subword is segmented into small parts, called tokens, which are usually smaller than characters. A token is usually larger than a primitive. The rationale behind this is that the strokes in cursive handwriting are longer than the strokes in the case of isolated characters. Factoring a cursive stroke into primitives yields a big number of


Figure 8.1. Data flow diagram of the Stroke Segmentation process in the CASR system.
such primitives which raises technical problems in the recognition process. Thus, it becomes necessary to find out larger building units; these are what we call tokens.

What we will do here is to segment a cursive stroke into a sequence of temporally ordered tokens which have features.

### 8.1.1. Definitions

(a) Let $\mu_{s e}$ be the path between the start vertex $v_{s}$ and the end vertex $v_{e}$ determined by the solution of the Chinese postman's problem for the reduced graph, $\mathrm{G}^{\prime}$, of the stroke. Each two consecutive vertices, $\mathrm{v}_{\mathrm{i}}$ and $\mathrm{v}_{\mathrm{j}}$, in $\mu_{\mathrm{sc}}$ constitute a link ( $\mathrm{v}_{\mathrm{i}}, \mathrm{v}_{\mathrm{j}}$ ).
(b) The first occurrence of a link, $\left(\mathrm{v}_{\mathrm{i}}, \mathrm{v}_{\mathrm{j}}\right)$, in $\mu_{\mathrm{se}}$ is called the basic trace of the link. If there is another occurrence, $\left(\mathrm{v}_{\mathrm{j}}, \mathrm{v}_{\mathrm{i}}\right)$, of the link, then it is called a retrace of the link (According to Algorithm 3.4, a link can have only one retrace). Figure 8.2(a) shows the image of one Arabic word which consists of six strokes; one main stroke, the largest one, and five secondary strokes. Most of the illustrations of this chapter and the next five chapters are based on this word. A straight line approximation, G , of the main stroke is shown in Figure 8.2(b) which is obtained using Algorithm 2.2.


Figure 8.2. (a) An Arabic word consisting of one main stroke and five secondary strokes, (b) a graph, $G$, of the main stroke, and (c) the reduced graph, $\mathrm{G}^{\prime}$, of the main stroke and the path $\mu_{s 0}$.

The graph, G, contains one loop set consisting of a single loop. The reduced graph, $\mathrm{G}^{\prime}$, is shown in Figure 8.2(c) with the loop set being represented by vertex 16 . The numbering of the vertices in $G^{\prime}$ follows the path $\mu_{x c}$ which is obtained using Algorithm 3.4, where $s$ and $e$, vertices 1 and 24, are the start and end vertices of writing, respectively. The path $\mu_{80}$ is 1-2-3-4-5-4-6-7-8-9-10-11-12-13-12-14-15-

16-15-17-18-17-19-20-21-22-23-24. The links (5, 4), (13, 12), (16, 15), and (18, 17) are retraces, while the remaining links are basic traces. According to Definition 4.1(a), all the links are original except link $(15,16)$ and its retrace since vertex 16 represents a loop set.
(c) A token is either:

1. a single isolated vertex, $\mathrm{v}_{1}$, representing a dot, referred to as $\Psi=\star\left(\mathrm{v}_{1}\right)$,
2. a single vertex, $v_{i}$, representing a loop set, referred to as $\Psi=O\left(v_{i}\right)$, e.g., vertex 16 in Figure 8.2(c), or
3. a sequence of consecutive vertices which connect original links such that each of the start and end vertices of the sequence is either a terminal vertex or an intersection vertex. If the token consists of the sequence of vertices $\mathrm{v}_{\mathbf{1}}, \mathrm{v}_{\mathbf{2}}, \ldots$ ., $\mathrm{v}_{\mathrm{n}}$ then it is represented as $\Psi=\left(\mathrm{v}_{1}, \mathrm{v}_{2}, \ldots, \mathrm{v}_{\mathrm{n}}\right)$ or $\Psi=\left(\psi_{1}, \psi_{2}, \ldots, \psi_{\mathrm{n}-1}\right)$, where $n$ is the number of vertices in the token and $\Psi_{i}$ is the link $\left(v_{i}, v_{i+1}\right)$. For example in Figure 8.2(c), the sequence of vertices 1,2,3, 4 represents one token and the sequence of vertices $5,4,6,7,8,9,10,11,12$ represents another token.
Now, a formal description of the segmentation algorithm follows.

## Algorithm 8.1

Use: To segment a cursive stroke into tokens
Input: 1. Reduced graph, $G^{\prime}$, of the stroke
2. Temporal information of the stroke, i.e., the path, $\mu_{s o}$, between the start vertex, $v_{s}$, and the end vertex, $v_{0}$

Output: Sequence of tokens $\Psi_{1}, \Psi_{2}, \ldots, \Psi_{m}$, where $m$ is the sequence length.

## Procedure:

Step 1. Marking token boundaries
(a) Mark the basic traces of links that have retraces.
(b) If there is an artificial link $\left(v_{i}, v_{j}\right)$ followed by another artificial link such that they have a common loop set vertex, then the link ( $\left.v_{i}, v_{j}\right)$ is marked. Marked links are considered as delimiters between tokens.

Step 2. Token extracting
(a) If the path $\mu_{\mathrm{sc}}$ consists of a single vertex, $\mathrm{v}_{1}$, then:

If the vertex represents a loop set, then it is represented by a loop set token $\Psi=O\left(v_{1}\right)$, where $v_{1}$ has the features of a loop set. Otherwise, it is represented by a vertex token $\Psi=\star\left(v_{1}\right)$. Stop.
(b) Otherwise, scan the path $\mu_{s e}$ starting from $v_{s}$. For every unmarked link $\left(v_{i}, v_{j}\right)$ do:

1. If either $\mathbf{v}_{\mathbf{i}}$ or $\mathbf{v}_{\mathbf{j}}$ is a vertex, $\mathbf{v}$, that represents a loop set then generate a new token $\Psi=O(v)$, where $v$ has the features of a loop sets. Mark the link $\left(v_{i}, v_{j}\right)$. Otherwise,
2. If no token was generated or the link $\left(v_{i}, v_{j}\right)$ is directly preceded by a marked link, then generate a new token $\Psi=\left(v_{i}, v_{j}\right)$. Otherwise, if the link $\left(v_{i}, v_{j}\right)$ is directly preceded by an unmarked link, then add $v_{j}$ to the last generated token.

### 8.1.2. Example

Algorithm 8.1 is used to segment the stroke shown in Figure 8.2(a), the reduced graph of which is shown in Figure 8.2(c), as follows:

Step 1. Marking token boundaries
This step results in marking the following links:
(a) $(4,5),(12,13),(15,16)$, and $(17,18)$ since they are basic traces of links that have retraces, and
(b) the link $(15,16)$ since it is an artificial link followed by another artificial link, (16, 15), with a common loop set vertex, 16 . Notice that link $(15,16)$ is already marked in (a) above, which does not harm. The usefulness of the case of two consecutive artificial links is made clearer in Figure 8.3. In the reduced graph of Figure 8.3(b), vertex 3 represents a loop set. The path for this graph is $\mu_{\infty 0}=1,2,3,4,5$. None of the links of this path has a retrace. Thus, Step 1(a) of Algorithm 8.1 does not mark any link of the path. The artificial link $(2,3)$ is followed by another artificial link (3, 4) with a common loop set vertex, 3. Thus, Step 1(b) results in marking link (2, 3).

Step 2. Token extracting
Since the path $\mu_{10}$ consists of more than one vertex, case (b) of this step applies:
(b) By scanning the path from vertex 1 , the first unmarked link, (1, 2), is encountered.


Figure 8.3. (a) The graph, G, of a stroke, and (b) its reduced graph $\mathrm{G}^{\prime}$; vertex 3 is a loop set.

Since neither vertex 1 nor vertex 2 represents a loop set, Step 2(b)2 is applied. Since no previous tokens were generated, a new token $\Psi_{1}=(1,2)$ is generated. The next unmarked link is $(2,3)$ which is preceded by an unmarked link, $(1,2)$. Thus, vertex 3 is added to $\Psi_{1}$. By the same way $\Psi_{1}$ grows to (1,2,3,4). When the unmarked link $(5,4)$ is reached, a new token is generated since that link is preceded by a marked link, $(4,5)$. When the unmarked link $(16,15)$ is reached, a new token is generated which has a loop set vertex, 16 , and that link is marked. At the end of the algorithm, the following tokens are obtained: $\Psi_{1}=(1,2,3,4), \Psi_{2}=(5,4,6,7,8$, $9,10,11,12), \Psi_{3}=(13,12,14,15), \Psi_{4}=(16), \Psi_{5}=(15,17)$, and $\Psi_{6}=(18,17$, $19,20,21,22,23,24)$. Figure 8.4 shows the extracted tokens from $\mathrm{G}^{\prime}$.

### 8.2. TOKEN FEATURES

Tokens have features which depend on whether they are single vertex or multivertex tokens as follows:
(a) For single vertex tokens there are two cases:

1. If the token represents an isolated dot then it has its $\mathrm{x} \& \mathrm{y}$ coordinates as features.
2. Otherwise, it represents a loop set with the features: $n_{\text {loops }}, x_{\text {cen }}, y_{\text {cen }}, x_{\text {min }}, x_{\text {max }}$, $y_{\text {min }} y_{\text {max }}$ and $c$, where $n_{\text {bops }}$ is the number of the loops in the loop set, $x_{\text {cen }}$ and $y_{c o n}$ are the $x \& y$ coordinates of the centre of gravity of the loop set, $x_{\min }, x_{\max }$, $y_{\text {min }}$ and $y_{\text {max }}$ are the limits of the bounding rectangle which surrounds the loop


Figure 8.4. Tokens extracted from the stroke of Figure 8.2(b): (a) $\Psi_{1}$, (b) $\Psi_{2}$, (c) $\Psi_{3},(\mathrm{~d}) \Psi_{4},(\mathrm{e}) \Psi_{5}$, and (f) $\Psi_{6}$.
set, and c is the sum of lengths of links constituting the loop set.
(b) For multi-vertex tokens, two features are used. To describe these features, first we define the bounding rectangle of a token as the minimum horizontal rectangle which encloses all the original vertices, in G, which represent the token. Figure 8.5 shows a token surrounded with its bounding rectangle. The Upper Right Corner (URC) of the bounding rectangle, and the parameters $H, W, d_{v}, d_{c}$, and $D$ are indicated, where $H$ and $W$ are the height and width of the bounding rectangle, respectively, $d_{1}$ and $d_{c}$ are the distances of the start and end vertices of the token to URC, respectively, and D is the diameter of the bounding rectangle. Now, the features of multi-vertex tokens are as follows:

1. Features of Start and End Vertices: Some tokens have similar shapes but only differ in the distance of both ends of the token to the Upper Right Corner (URC) of the bounding rectangle of the token, as shown in Figure 8.6. Thus, the following two features are defined:

$$
\begin{equation*}
f_{3}=d_{3} / D \tag{8.1}
\end{equation*}
$$

and

$$
\begin{equation*}
f_{c}=d_{c} / D \tag{8.2}
\end{equation*}
$$

where $d_{2}$ and $d_{c}$ are the distances of the start and end vertices of the token to


Figure 8.5. A token, its bounding rectangle, and other relevant parameters.


Figure 8.6. Two tokens having similar structures but different features of start and end vertices.

URC, respectively, and $D$ is the diameter of the bounding rectangle. Dividing by D normalizes the features so that they do not exceed unity.
2. Feature of Height to Height-plus-Width Ratio: This feature is similar to the feature of height-plus-width ratio which was defined for strokes in Section 4.4. Figure 8.7 shows two different tokens which have similar structures. To distinguish between these tokens, let H and W be the height and width of the bounding rectangle of the token, respectively, and define the feature:

$$
\begin{equation*}
\mathrm{f}_{\mathrm{HW}}=\mathrm{H} /(\mathrm{H}+\mathrm{W}) \tag{8.3}
\end{equation*}
$$

The motivation for setting the dominator to $\mathrm{H}+\mathrm{W}$ is to keep $\mathrm{f}_{\mathrm{HW}}$ always less


Figure 8.7. Two tokens with similar structures but different height to height-pluswidth ratios.
than or equal to 1.
For example, in Example 8.1.2, the first token, $\Psi_{1}$, has the features $f_{2}=0.02, f_{0}=$ 0.98 , and $\mathrm{f}_{\mathrm{Hw}}=0.64$.

## SUMMARY

In this chapter, an algorithm was presented to segment a cursive stroke into small parts, called tokens. Tokens are logical units usually larger and more suitable for cursive script than the primitives of the IACR system. Tokens were discriminated by adopting suitable features.

## Token

## Recognition



## OVERVIEW

This chapter addresses the token recognition component of the CASR system. Fuzzy sequential machines are defined to work as recognizers of segmented tokens of strokes. A data flow diagram of the token recognition process is shown in Figure 9.1, which consists of one filter and three subprocess:
(a) Isolating Single Vertex Tokens: which accepts the segmented tokens and reduced graphs of their strokes. Single vertex tokens are separated and output, without any further processing. Multi-vertex tokens and reduced graphs of their strokes are fed to the next three subprocesses.
(b) Calculating Link Features: where link features are calculated.
(c) Constructing Acceptance Trees: This subprocess accepts as inputs multi-vertex tokens to be recognized, reduced graphs of their strokes, and a token fuzzy sequential machine which is used in the recognition. The acceptance information of tokens is stored in tree data structures (one tree for each token), which is output by this subprocess.
(d) Checks and Decisions: Here, the inputs are multi-vertex tokens to be recognized, reduced graphs of their strokes, a token fuzzy sequential machine, and the acceptance trees which are output by the previous subprocess. Some calculations, based on the information in the acceptance trees, are performed to evaluate some features of the tokens. Recognition results are output indicating


Figure 9.1. Data flow diagram of the Token Recognition process.
whether an unknown token belongs to a certain token class. If a token could not be recognized, then its acceptance tree is output which can be fed to a subsequent token learning stage.

### 9.1. TOKEN FUZZY SEQUENTIAL MACHINE

In Part Two, fuzzy sequential machines were defined and proved to be powerful in modelling and recognizing isolated handwritten strokes. In this part, we will continue to use such machines to recognize multi-vertex tokens. The reader can refer to Section 5.1
for background material about this new approach.
Since single vertex tokens, i.e., dot and loop set tokens are simple units, they are recognized directly as soon as they are segmented. Thus, a token fuzzy sequential machine will recognize only multi-vertex tokens.

Multi-vertex tokens only contain links, i.e., they do not contain intersection or loop set vertices. Consequently, a state entrance qualifier can be only a fuzzy direction expressed as an $s / z$-number $\left(p_{1} / \beta_{1} ; p_{2} \backslash \beta_{2}\right)$, where $p_{1}, \beta_{1}, p_{2}$, and $\beta_{2}$ are as defined in Chapter 5. The features which are needed for further discrimination between tokens are of the same type as the features mentioned in Section 8.2, i.e., they are fuzzy features of start and end vertices and fuzzy features of height to height-plus-width ratio. Fuzzy features are fuzzy numbers having $\mathrm{s} / \mathrm{z}$ possibility distributions.

### 9.1.1. Definition of Token Fuzzy Sequential Machine

A deterministic Fuzzy sequential Machine which recognizes a set of multi-vertex tokens is a system that is characterized by the 5 -tuple $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, \mathrm{T})$, where
(a) C is the set of multi-vertex token classes that are recognized by this machine,
(b) Q is a finite set of states $\mathrm{q}_{0}, \mathrm{q}_{1}, \ldots, \mathrm{q}_{0}$ being the starting state,
(c) $\Sigma$ is a finite nonempty set of state entrance qualifiers $\sigma_{1}, \sigma_{2}, \ldots$ Each state, $q_{i}$, except $q_{0}$ which does not have an entrance qualifier, has one and only one entrance qualifier, $\sigma_{i}$. A state entrance qualifier is a fuzzy direction that has an $s / z$-number possibility distribution.
(d) $\zeta$ is the next-state mapping function $\zeta: \mathbf{Q} \times \Sigma \rightarrow \mathrm{Q}$,
(e) $T \subset Q$ is a finite nonempty set of terminal states. A terminal state $q_{i} \in T$ recognizes the token classes $c_{i 1}, c_{i 2}, \ldots \in C$, each of which has $n_{i}$ fuzzy features of the same type, i.e., more than one token class can be recognized in one terminal state with the same number and types of fuzzy features. We mean by same types of fuzzy features that the first fuzzy feature of $c_{11}$ has the same type of the first fuzzy feature of $c_{i 2}, c_{13}$, . . .. The same applies to the remaining fuzzy features. These fuzzy features are fuzzy numbers having $s / z$ possibility distributions.

There is a recognition algorithm which explains how a multi-vertex token is handled by a fuzzy sequential machine to decide whether it belongs to a token class $\mathbf{c} \in \mathrm{C}$. The

(a)

(b)

Figure 9.2. (a) A deterministic fuzzy sequential machine, FM, and (b) a multi-vertex token that can be recognized by this machine.
details of the recognition algorithm are presented in Section 9.2.

### 9.1.2. Example

Figure 9.2(a), shows a transition diagram of a deterministic fuzzy sequential machine $F M=(C, Q, \Sigma, \zeta, T)$, where
(a) $\mathrm{C}=\{$ token no. 1 $\}$,
(b) $Q=\left\{q_{i}, i=0,1, \ldots, 6\right\}$,
(c) $\mathrm{\Sigma}=\left\{\sigma_{\mathrm{i}}, \mathrm{i}=1,2, \ldots, 6\right\}, \sigma_{1}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right), \sigma_{2}=\left(236.25^{\circ}\right.$ $\left./ 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right), \sigma_{3}=\left(213.75^{\circ} / 22.5^{\circ} ; 236.25^{\circ} \backslash 22.5^{\circ}\right), \sigma_{4}=\left(236.25^{\circ}\right.$ $\left./ 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right), \sigma_{5}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right)$, and $\sigma_{6}=$ ( $281.25^{\circ} / 22.5^{\circ} ; 303.75^{\circ} \backslash 22.5^{\circ}$ ),
(d) $\zeta$ is the next-state mapping function, $\zeta: \mathbf{Q} \times \mathbf{\Sigma} \rightarrow \mathrm{Q}$, which can be easily read from the figure,
(e) $T=\left\{q_{6}\right\}$, where token class no. 1 is embedded in $q_{6}$. One token, belonging to this class is shown in Figure 9.2(b). This class is associated with fuzzy features of start
and end vertices, $\mathrm{F}_{\mathrm{s}}=(0.0 / 0.0 ; 0.2 \backslash 0.2)$ and $\mathrm{F}_{\mathrm{e}}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$, and the fuzzy feature of height to height-plus-width ration, $\mathrm{F}_{\mathrm{Hw}}=(0.6 / 0.2 ; 0.8 \backslash 0.2)$.

### 9.2. RECOGNITION ALGORITHM

In this section, the use of FM's to recognize multi-vertex tokens is detailed. The recognition algorithm differs from Algorithm 5.3, which was designed to recognize whole strokes, in the following:
(a) In Algorithm 5.3, a primitive was considered for the two situations: spurious and non-spurious. In one experiment performed on 2085 handwritten Arabic strokes written by four subjects, it was found that spurious tails occurred in $2.16 \%$ ( 45 out of 2085) of the graphs obtained using Algorithm 2.2. In the graphs with tails, there was $97.8 \%$ ( 44 out of 45 ) of the graphs with one single tail and $2.2 \%$ with two tails. None of the graphs have two adjacent tails, i.e., tails which have a common vertex. Spurious single loops were generated in 2 out of 2085 graphs. Based on this useful statistical experiment, it will be assumed henceforth that the graphs are free of spurious tails or loops. Thus, in the algorithm of this section, each link will be assumed non-spurious which reduces the size of the acceptance tree without greatly affecting the system performance.
(b) The existence of intersection vertices and loops and letting a primitive to be tested for the two situations mentioned above imposed conditions, in Step 2(b)2 of Algorithm 5.3, to test the validity of a sequence of primitives. Since multi-vertex tokens do not contain intersection vertices nor loops and their links are assumed non-spurious, these conditions are dropped from the algorithm of this section.

### 9.2.1. Definitions

(a) The total length, L , of a multi-vertex token is the summation of the lengths of its individual links.
(b) The acceptance degree of a link, $\psi$, of a token in a state, q, is $\psi(\Psi, \mathrm{q})=\pi_{0}(\theta)$, where $\pi_{0}$ is the possibility distribution that characterizes the fuzzy direction state entrance qualifier, $\sigma$, of the state, $q$, and $\theta$ is the angle of the link, $\psi$.
(c) The same as in Algorithm 5.3, the information resulting from machine excitation will
be saved in what we call an acceptance tree which has the following properties:

1. The root node of the acceptance tree, which is considered as level 0 of the tree, contains the 2-tuple ( $\mathrm{q}_{0}, 1.0$ ), where $\mathrm{q}_{0}$ is the starting state of FM and the 1.0 is the initial credit, $\omega$, assigned to the token. This credit is an initial acceptance degree of the token.
2. The nodes of level $i$ of the acceptance tree, $i \geq 1$, contain acceptance information only about link $\Psi_{i}$ of the token. Thus, you expect that the maximum level of the tree will not exceed the number of links in the token.
3. Each time a link, $\Psi$, is accepted by a state, q , the token is punished by subtracting from its credit, $\omega$, a value, $\delta=\operatorname{lr} \times(1.00-x(\Psi, q))$, where $x(\psi$, $q)$ is determined according to Definition 9.2.1(b), above, Ir is the ratio of the length of link, $\Psi$, to the total length of the token. Notice that, $\delta$, is proportional to the length ratio, lr , of the link, $\Psi$, and the degree of disagreement between $\psi$ and the state entrance qualifier of q , which is a logical punishment rule.
4. Any node in the acceptance tree, other than the root node, contains the 2-tuple $\left(q_{j}, \omega\right)$, where $q_{j}$ is an accepting state in FM that was entered after being excited by a link, $\Psi_{\mathrm{i}}$, and $\omega$ is the remaining credit for the token.

### 9.2.2. Algorithm Description

The detailed description of the algorithm which is used to recognize tokens now follows. For multi-vertex tokens, the best k token classes which are candidates to represent the unknown token are found using this algorithm. This is useful in the case of cursive handwritten strokes since a stroke usually consists of many tokens and it is not always true that the first best candidate class of every token of the stroke yields the best interpretation of the whole stroke.

Algorithm 9.2
Use: To recognize tokens
Input: 1. A token fuzzy sequential machine $F M=(C, Q, \Sigma, \zeta, T)$
2. Tokens to be recognized where each token is represented as a sequence

$$
\Psi=\left\{\Psi_{i}, i=1,2, \ldots, m-1\right\} \text { of } m-1 \text { links }
$$

3. Reduced graph of strokes from which the tokens were segmented

Output: If a token is recognized, the output is the best k classes which can represent the token, each with the corresponding overall acceptance degree, otherwise, the token is rejected and its acceptance tree is output to be passed to a subsequent token learning process.

## Procedure:

For each input token, if the token consists of a single vertex, i.e., it represents either a dot or a loop set, then it is considered as fully recognized with no extra processing, otherwise, the following steps are performed.

## Step 1. Calculating link features

(a) Find the length, $l_{\mathrm{j}}$ of link, $\Psi_{\mathrm{i}}$, which equals the Euclidean distance between the two vertices of the link. Find the total length, $L$, of the token.
(b) The length ratio, $\mathrm{Ir}_{\mathrm{i}}=l_{i} / \mathrm{L}$, and angle, $\theta_{i}$, of each link are found.

## Step 2. Constructing the acceptance tree

(a) Create the root node, node 0 , which constitutes level 0 of the tree, with the 2-tuple ( $\mathrm{q}_{0}, 1.0$ ),
(b) Create level $i$ of the tree which contains acceptance information of link, $\Psi_{i}, i=1,2$, $\ldots, \mathrm{m}-1$ as follows. For every node, $\mathrm{d}_{\mathrm{j}}$, that exists in level $\mathrm{i}-1$ and has the state $q$ and credit $\omega$ as tuples:

For every next state, $\zeta(q, \sigma)$, of $q$ that can accept $\Psi_{i}$, add a new node, $d_{k}$, that is a son of node, $d_{j}$, where $d_{k}$ contains the 2-tuple $(\zeta(q, \sigma), \omega-\delta), \delta=1 r_{i} \times$ $\left(1.00-x\left(\psi_{\dot{p}} \zeta(q, \sigma)\right)\right)$, such that the remaining credit, $\omega-\delta$, is not less than a specified threshold, THR $_{1}$.

## Step 3. Checks and decisions

After constructing the acceptance tree, one of the following two cases occurs:
(a) The last level of the tree is less than the number of the links, $\mathrm{m}-1$. This occurs if there is no state sequence that can accept the link sequence, $\Psi$, provided that the final credit is not less than $\mathrm{THR}_{1}$, in which case the token is unknown.
(b) The last level of the tree equals $m-1$. In this case retain the set of leaf nodes, $D$, the levels of which are equal to $m-1$. For each node $d \in D$ having the 2-tuple ( $q$,
$\omega_{1}$ ) such that $q \in T$, i.e., $q$ is a terminal state, do:

1. Compute the features, $f_{s}, f_{c}$, and $f_{H W}$, of the token, $\Psi$, see Section 8.2.
2. For each token class, c , in q :

Find the acceptance degree, $\pi_{F}(f)$, of each feature, $f \in\left\{f_{s}, f_{e}, f_{H W}\right\}$, of the token in the corresponding fuzzy feature, $F$, of the class, $c$. The overall acceptance degree, $\omega_{2}$, by which a class accepts a token is the summation of the acceptance degrees of the features of the token in the corresponding fuzzy features of the class, c , divided by the number of the features, $n_{f}=3$. If $\omega_{2}$ is less than a specified threshold, $\mathrm{THR}_{2}$, then the token is not accepted by class $c$. Otherwise, a triple of the form ( $\omega_{1}$, $\left.\omega_{2}, c\right)$ is created.

If it was not possible to obtain at least one triple, $\left(\omega_{1}, \omega_{2}, c\right)$, then the token is unknown, otherwise, the token is assigned the k classes of the k triples which have the maximum $\omega_{3}=\min \left(\omega_{1}, \omega_{2}\right)$ degree. The overall acceptance degree, $\omega_{3}$, of each class, c , of the assigned k classes, is also retained.

### 9.2.3. Example

The token of Figure 9.3(a) is to be recognized by the token fuzzy sequential machine, FM, shown in Figure 9.2(a). Since the token is a multi-vertex token, the following steps are performed:

Step 1. Calculating link features
(a) The lengths of the token's links are: $l_{1}=11.2, l_{2}=19.3, l_{3}=15.6$, and $l_{4}=20.0$. The total length of the token is $L=\sum_{i=1}^{4} l_{i}=66.1$.
(b) Length ratios are: $1 r_{1}=0.17,1 r_{2}=0.29,1 r_{3}=0.24$, and $l r_{4}=0.30$. Link angles are $\theta_{1}$ $=260^{\circ}, \theta_{2}=201^{\circ}, \theta_{3}=230^{\circ}$, and $\theta_{4}=270^{\circ}$.

Step 2. Constructing the acceptance tree, see Figure 9.3(b)
(a) Create the root of the tree, node 0 , with the 2 -tuple ( $\mathrm{q}_{0}, 1.00$ ), i.e., we start from the starting state, $q_{0}$, of the machine shown in Figure 9.2(a) with an initial credit $\omega=$ 1.00 .


Figure 9.3. (a) A token to be recognized by the machine, FM, of Figure 9.2(a), and (b) the acceptance tree obtained from this token and FM.
(b) 1. Create level 1 of the tree which contains acceptance information about the first link, $\psi_{1}$. In level $1-1=0$, there is one node, 0 , which has $q_{0}$ as one tuple. There is one next state, $q_{1}$, of $q_{0}$ with a fuzzy direction state entrance qualifier $\dot{\sigma}_{1}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right)$. Thus, the link, $\Psi_{1}$, is accepted by state $\mathrm{q}_{1}$ with an acceptance degree $x\left(\Psi_{1}, \mathrm{q}_{1}\right)=\pi_{01}\left(\theta_{1}=260^{\circ}\right)=1.00$. If a value $\delta$ $=1 r_{1} \times\left(1.00-x\left(\Psi_{1}, q_{1}\right)\right)=0.0$ is discounted from the credit of node 0 , then the remaining credit will be $(1.0-0.0=1.0)>\left(\mathrm{THR}_{1}=0.6\right)$. Thus, a new
node, 1 , is created with the 2-tuple ( $q_{1}, 1.00$ ).
2. Create level 2 of the tree which contains acceptance information about the second link, $\psi_{2}$. In level 2-1=1, there is one node, 1 , which has $q_{1}$ as one tuple. $q_{1}$ has three next states: $q_{1}, q_{2}$, and $q_{3}$. Next state $q_{1}$ has the fuzzy direction state entrance qualifier $\sigma_{1}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \cup 22.5^{\circ}\right)$. The link, $\Psi_{2}$ is accepted by state $q_{1}$ with an acceptance degree $x\left(\Psi_{2}, q_{1}\right)=\pi_{01}\left(\theta_{2}\right.$ $\left.=201^{\circ}\right)=0.00$. If a value, $\delta=r_{2} \times\left(1.00-x\left(\Psi_{2}, q_{1}\right)\right)=0.29$, is discounted from the credit of node 1 , then the remaining credit will be $0.71>\mathrm{THR}_{1}$. Thus, a new node, 2 , is created with the 2-tuple ( $q_{1}, 0.71$ ). In a similar manner, nodes 3 and 4 are created to correspond to next states $q_{2}$, and $q_{3}$.
3. Similarly, levels 3 and 4 of the tree are created. Finally, the tree of Figure 9.3(b) is obtained with the nodes being referenced by the numbers underneath.

Step 3. Checks and decisions
Since the last level of the acceptance tree is 4 which equals the number of the links in the token, case (b) applies:
(b) Here, we retain only nodes 15 and 18 since they contain a terminal state, $\mathrm{q}_{6}$ :

1. The features of the token are: $\mathrm{f}_{2}=0.00, \mathrm{f}_{\mathrm{c}}=1.00$, and $\mathrm{f}_{\mathrm{HW}}=0.62$.
2. State $q_{6}$ has one token class, token class no. 1 , three fuzzy features, $\mathrm{F}_{\mathrm{p}}, \mathrm{F}_{\mathrm{e}}$, and $\mathrm{F}_{\mathrm{Hw}}$; refer to Example 9.1.2. The acceptance degrees of the features of the token in the corresponding fuzzy features are: $\pi_{\mathrm{Fi}}\left(\mathrm{f}_{\mathrm{i}}\right)=1.00, \mathrm{i}=\mathrm{s}, \mathrm{e}, \mathrm{HW}$. The overall acceptance degree of features is $\left(\pi_{\mathrm{Fs}}\left(\mathrm{f}_{\mathrm{a}}\right)+\pi_{\mathrm{Fe}}\left(\mathrm{f}_{\mathrm{e}}\right)+\pi_{\mathrm{FHW}}\left(\mathrm{f}_{\mathrm{HW}}\right)\right) / 3=$ $1.00>\left(\mathrm{THR}_{2}=0.60\right)$. Thus, the token is accepted as belonging to token class no. 1.

Two triples are obtained, ( $0.67,1.00$, token no. 1 ) and ( $0.64,1.00$, token no. 1 ), corresponding to nodes 15 and 18 , respectively. Letting $k=1$, and since $\min (0.67$, $1.00)>\min (0.64,100)$, the token is assigned token class no. 1 with an overall acceptance degree $\omega_{3}=0.67$.

## SUMMARY

This chapter addressed the token recognition component of the CASR system. Fuzzy sequential machines were defined to work as recognizers of segmented tokens of
strokes. A formal description of the recognition algorithm was presented. The algorithm differs from that developed for stroke recognition in Chapter 5 due to reasons which were mentioned. The use of fuzzy machines to recognize tokens was clarified through an example

## Token

## Learning



## OVERVIEW

In this chapter, the token learning component of the CASR system is presented. A data flow diagram of the token learning process is shown in Figure 10.1, which consists of two sides, where only one side is active at a time. The learning algorithm determines which side to activate, depending on the last level of the acceptance tree of an unrecognized token which is passed by the token recognition process, as follows:
(a) If the last level of the acceptance tree is less than the number of links in the token, then the right side is activated, which consists of the following subprocesses:

1. Generating a Token Fuzzy Sequential Machine, Section 10.2: where the inputs are the sequence of links of the token and reduced graph of the corresponding stroke. The output is a fuzzy sequential machine which can recognize the input token and variants of it.
2. Merging Token Fuzzy Sequential Machines, Section 10.3: which accepts as inputs the machine generated in (1) and the old machine which was used in recognition but failed to recognize the token. The result is a new machine which can recognize the input token and tokens of the old machine and variants of these tokens.
(b) If the last level of the acceptance tree equals the number of links in the token, then the left side of Figure 10.1 is activated, where some modifications are introduced


Figure 10.1. Data flow diagram of the Token Learning process in the CASR system. If the last level of the acceptance tree is less than the number of links in the token, then the right side is activated. Otherwise, the last level of the acceptance tree equals the number of links in the token in which case the left side is activated.
into the machine that was used in recognition. These modifications are sufficient to learn the token, Section 10.4. The inputs in this case are the links of the token, reduced graph of the corresponding stroke, and the acceptance tree. The output is a modified fuzzy sequential machine which can recognize the input token and tokens of the old machine and variants of them.

### 10.1. LEARNING ALGORITHM

This chapter is concerned with the learning process of unrecognized multi-vertex tokens. Single vertex tokens are considered fully recognized once they are segmented. If $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, \mathrm{~T}_{1}\right)$ is the machine that was used when recognition was tried but failed, then our goal is to modify or expand $\mathrm{FM}_{1}$ so that it becomes capable of recognizing tokens belonging to the unknown token's class. Of course, the set $Q_{1}$ is initially empty. $\mathrm{FM}_{1}$ possesses new states and classes when more tokens are learned. An algorithm to
learn tokens is described below.

Algorithm 10.1

Use: $\quad$ To learn a multi-vertex token
Input: 1. Sequence of links, $\Psi$, of the token to be learned
2. Reduced graph of the stroke from which the token, $\Psi$, was segmented
3. Acceptance tree of the token which is unrecognized by Algorithm 9.2
4. Fuzzy sequential machine, $\mathrm{FM}_{1}$, of multi-vertex tokens which are already learned

Output: New fuzzy sequential machine, FM, which can recognize the underlying token and previously learned tokens

## Procedure:

Depending on the last level of the acceptance tree of the unrecognized token, one of the following two cases occurs:
(a) The last level of the acceptance tree is less than the number of links of the token, $m$
-1 . This means that it was not possible for the whole sequence of links to terminate at some state such that the remaining credit of the token $\geq \mathrm{THR}_{1}$. In this case, simple modifications of some states are not adequate. We adopted the following solution:

1. Obtain a new fuzzy sequential machine, $\mathrm{FM}_{2}$, from the token's sequence of links, $\Psi$. This machine can recognize tokens belonging to the underlying token's class. Section 10.2 explains how to obtain a fuzzy sequential machine from a given input token.
2. Merge $\mathrm{FM}_{1}$ and $\mathrm{FM}_{2}$ to obtain the machine FM that is capable of recognizing tokens belonging to the underlying token's class with an overall acceptance degree equal to 1.00 . The new machine still memorizes all previously learned tokens. Merging of sequential machines is addressed in Section 10.3.
(b) The last level of the acceptance tree equals, $m-1$, the number of links in the sequence, $\Psi$. This case occurs:
3. if the link sequence could terminate at one or more terminal state such that the remaining credit of the stroke did not fall below $\mathrm{THR}_{1}$, but could not be
recognized due to severe deviation of the token's features from the fuzzy features of the classes embedded in any of these terminal states, i.e., the acceptance degree of the token's features is less than THR $_{2}$.
4. and / or the link sequence terminated at one or more states, NT, which are not terminal states and such that the remaining credit of the token is not less than $\mathrm{THR}_{1}$.

In this case, there is no need to create a new fuzzy sequential machine as in (a). Only some modifications to $\mathrm{FM}_{1}$ are sufficient, which are:

1. Either add a new class, with its fuzzy features, in one terminal state which already exists and could be reached by the sequence of links, $\Psi$, or
2. Change one of the states in NT into a terminal state, i.e., append it to the set T, and insert in it the class of the token with its fuzzy features.

These modifications are explained in detail in Section 10.4.

### 10.2. GENERATION OF FUZZY SEQUENTIAL MACHINES

As shown in Figure 10.1, a necessary step in the learning process is to obtain a fuzzy sequential machine from an input token. The manual generation of this machine is cumbersome, since it may contain several states requiring proper state entrance qualifiers, next state mapping function, and insertion of token classes with their fuzzy features in suitable states. Thus, it is important to search for somemeans to automate the generation of fuzzy sequential machines. In the following, an algorithm is described to obtain a fuzzy sequential machine from a multi-vertex token. The difference between this algorithm and Algorithm 6.2 , which obtains a fuzzy sequential machine from a stroke, is that in the later, a primitive can be: (1) a vertex representing an isolated vertex or an intersection vertex, (2) a loop set, or (3) a link. Multi-vertex tokens, which are the input to the current algorithm, consist of links; i.e., they have no intersection nor loop set vertices. The current algorithm is based on the following points:
(a) For a link, a state is generated which has an adjusted fuzzy direction state entrance qualifier such that the link is $100 \%$ accepted by that state.
(b) For each two consecutive links, a sequence of states is generated, each having an adjusted fuzzy direction state entrance qualifier. Transitions can be made from any
of these states to itself in addition to the other states following it in the sequence. By adopting this, we add flexibility to the generated fuzzy sequential machine since the change from a given direction of a link to another direction of another link can be achieved either directly or gradually by passing via many intermediate states. This allows future variants of the token, which was used to generate the machine, to be also recognized by the same machine.
(c) In the final generated state, a class of the underlying token is added with fuzzy features which are obtained from the calculated token's features. A formal description of the algorithm follows.

## Algorithm 10.2

Use: To generate a fuzzy sequential machine from a multi-vertex token
Input: 1. Sequence of links of the token to be learned $\Psi=\left\{\Psi_{j}, j=1,2,3, \ldots, m\right.$ - 1 \}
2. Reduced graph of the stroke from which the token, $\Psi$, was segmented

Output: Fuzzy sequential machine $F M=(C, Q, \Sigma, \zeta, T)$ which can recognize tokens belonging to the class of the input token

## Procedure:

Step 1. Initialization
Let $\mathrm{i}=1$ and $\mathrm{j}=1$, where i and j are used as indices for the generated states and the links of the token, respectively.

Step 2. Creation of the starting state
Create the starting state, $\mathrm{q}_{0}$.
Step 3. Creation of an accepting state for the first link $\Psi_{1}$
(a) Fuzzify the direction angle, $\theta$, of $\Psi_{1}$ by transforming it into a fuzzy direction $\sigma_{1}=$ $\left(p_{1} / \beta_{1} ; p_{2} \backslash \beta_{2}\right)$ as follows:

1. Set both the left and right bandwidths, $\beta_{1}$ and $\beta_{2}$, to suitable values.
2. The range from $0^{\circ}$ to $360^{\circ}$ is divided into $360^{\circ}$ / core angular intervals. The first angular interval is centred at $0^{\circ}$. If $\theta$ lies in the angular interval the centre of which is $\Theta_{0}$, then set the left peak point to $p_{1}=\Theta_{0}-$ core $/ 2$ and the right peak point to $p_{2}=\Theta_{0}+$ core $/ 2$.
(b) Create a new state $q_{1}$ which has the fuzzy direction, $\sigma_{1}$, as its state entrance qualifier. Two directed arcs, which are labelled with $\sigma_{1}$, are added to point from $q_{0}$ to $q_{1}$ and from $q_{1}$ to itself and, i.e., we set $\zeta\left(q_{0}, \sigma_{1}\right)=q_{1}$ and $\zeta\left(q_{1}, \sigma_{1}\right)=q_{1}$.
(c) If $\psi_{1}$ is the only link then go to Step 5 .

Step 4. Creation of an accepting state for the link $\psi_{j+1}$ and a sequence of states which lie between state $q_{i}$ and the state which accepts $\psi_{j+1}$ to allow for gradual change of direction

Let $\theta_{\mathbf{k}}$ be the angle of the link, $\psi_{j+1}$, which lies in the angular interval the centre of which is $\Theta_{k}$ :
(a) Consider the centres $\Theta_{1}, \Theta_{2}, \ldots, \Theta_{k}$, of the angular intervals which lie between $\Theta_{0}$ and $\Theta_{\mathfrak{b}}$ where $\Theta_{l}=\Theta_{0} \pm l \times$ core, $l=1, \ldots, k$. The plus sign is used if $\Theta_{0}<\theta_{k}$, and vice versa.
(b) Create k new states $\mathrm{q}_{\mathrm{i}+1}, \mathrm{q}_{\mathrm{i}+2}, \ldots, \mathrm{q}_{i+\mathrm{b}}$, where state $\mathrm{q}_{\mathrm{i}+\mathrm{l}}$ has the fuzzy direction $\sigma_{i+l}=$ $\left(p_{n} / \beta_{1} ; p_{n} \backslash \beta_{2}\right)$ as its state entrance qualifier, $p_{n}=\theta_{1}-$ core $/ 2, p_{n}=\theta_{l}+$ core $/ 2$, $\beta_{1}$ and $\beta_{2}$ are the left and right bandwidths.
(c) Add directed arcs from state $q_{2}$ to states $q_{a+1}, q_{a+21}, \ldots, q_{i+b}$, where $a=i, i+1, \ldots$, $\mathrm{i}+\mathrm{k}-1$. All arcs entering a state $\mathrm{q}_{\mathrm{i}+1}$ are labelled with the fuzzy direction state entrance qualifier $\sigma_{i+1}$. Thus, we get $\zeta\left(q_{v}, \sigma_{b}\right)=q_{b}, a=i, i+1, \ldots, i+k-1$, and $b$ $=a+1, \ldots, i+k$.
(d) For every state $\mathrm{q}_{\mathrm{i}+1}$ add a directed arc, which is labelled with $\sigma_{i+p}$ to point from that state to itself, i.e., $\zeta\left(\mathrm{q}_{\mathrm{i}+\mathrm{b}} \sigma_{\mathrm{i}+1}\right)=\mathrm{q}_{\mathrm{i}+\mathrm{b}} l=1,2, \ldots, \mathrm{k}$.
(e) Increment i by k and j by 1 .
(f) If $\Psi_{j}$ is not the last link then let $\Theta_{0}=\Theta_{k}$ and repeat Step 4.

Step 5. Setup of a terminal state
The last created state, $q_{\mathrm{i}}$, is considered as a terminal state. Thus, for the machine, FM, T $=\left\{q_{i}\right\}$. A class, $c_{\text {new }}$ of the underlying token, with fuzzy features, is embedded in $q_{i}$, hence, $\mathrm{C}=\left\{\mathrm{c}_{\mathrm{new}}\right\}$. The fuzzy features are calculated as follows:
(a) Find the features of the token, $\Psi$, see Section 8.2.
(b) Fuzzify every feature, f, computed in (a), by transforming it into an $\mathrm{s} / \mathrm{z}$-number as follows:

1. Set both the left and right bandwidths to $\beta_{\mathrm{f}}$.
2. The range from 0.0 to 1.0 is divided into $n_{f}$ equal intervals. The left and right peak points, $p_{1}$ and $p_{2}$, are taken to be equal to the limits of the interval in which the value $f$ lies.

Suitable values of $\beta_{f}$ and $n_{f}$ were found to be 0.2 and 5 , respectively.

In Steps 3 and 4 of the above algorithm, the three parameters: core, $\beta_{1}$, and $\beta_{2}$ were suitably found to be equal to $22.5^{\circ}$.

### 10.2.1. Example

Figure 10.2(a) shows one token $\Psi=\left(\Psi_{1}, \Psi_{2}, \Psi_{3}\right)$, where $\Psi_{1}=l(1,2), \Psi_{2}=l(2,3)$, and $\Psi_{3}=l(3,4)$. A fuzzy sequential machine, $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, \mathrm{~T}_{1}\right)$, is to be obtained which can recognize this token. For the sake of easy referencing, in the next sections where other machines are generated, we use two subscripts to refer to a state in the machine, $\mathrm{FM}_{1}$, e.g., $\mathrm{q}_{1 \mathrm{i}}$ refers to state $\mathrm{q}_{\mathrm{i}}$ of machine $\mathrm{FM}_{1}$. See Figure 10.2(b) while reading the following steps.

Step 1. Initialization
Let $\mathrm{i}=1$ and $\mathrm{j}=1$.
Step 2. Creation of the starting state
Create the starting state, $\mathrm{q}_{10}$.
Step 3. Creation of an accepting state for the first link $\psi_{1}=l(1,2)$ with angle $\theta$ $=274^{\circ}$
(a) Fuzzify the angle value $274^{\circ}$ by transforming it into a fuzzy direction $\sigma_{11}=\left(p_{1} / \beta_{1}\right.$; $\mathrm{p}_{2} \backslash \beta_{2}$ ) as follows:

1. Set both the left and right bandwidths, $\beta_{1}$ and $\beta_{2}$, to $22.5^{\circ}$.
2. By dividing the range from $0^{\circ}$ to $360^{\circ}$ into $360^{\circ} / 22.5^{\circ}=16$ angular intervals, we find that the angle $274^{\circ}$ lies in the angular interval the centre of which is $\Theta_{0}=270^{\circ}$. The left peak point is set to $p_{1}=270^{\circ}-22.5^{\circ} / 2=$ $258.75^{\circ}$ and the right peak point is set to $p_{2}=270^{\circ}+22.5^{\circ} / 2=281.25^{\circ}$. Finally, we get $\sigma_{11}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right)$.
(b) Create a new state $q_{11}$ which has the fuzzy direction $\sigma_{11}$ as its state entrance qualifier. Two directed arcs, which are labelled with $\sigma_{11}$, are added to point from $\mathrm{q}_{10}$


Figure 10.2. (a) A multi-vertex token, and (b) a deterministic fuzzy sequential machine, $\mathrm{FM}_{1}$, obtained from this token.
to $q_{11}$ and from $q_{11}$ to itself, i.e., we set $\zeta_{1}\left(q_{10}, \sigma_{11}\right)=q_{11}$ and $\zeta_{1}\left(q_{11}, \sigma_{11}\right)=q_{11}$. (c) Since there are more links we go to Step 4.

Step 4. Creation of an accepting state for the link $\Psi_{2}$ and a sequence of states which lie between state $\mathrm{q}_{11}$ and the state which accepts $\Psi_{2}$
The angle of $\Psi_{2}$ is $\theta_{k}=222.0^{\circ}$, which lies in the angular interval the centre of which is $\Theta_{k}=225^{\circ}$ :
(a) Consider the centres $\Theta_{1}=247.5^{\circ}$ and $\Theta_{2}=225.0^{\circ}$, i.e., $k=2$, of the angular intervals which lie between $\Theta_{0}=270.0^{\circ}$ and $\Theta_{2}=225.0^{\circ}$.
(b) Create two (since $k=2$ ) new states $q_{12}$ and $q_{13}$, where state $q_{12}$ has the fuzzy direction $\sigma_{12}=\left(p_{11} / \beta_{1} ; p_{12} \backslash \beta_{2}\right)$ as its state entrance qualifier, and state $q_{13}$ has the fuzzy direction $\sigma_{13}=\left(p_{21} / \beta_{1} ; p_{22} / \beta_{2}\right)$ as its state entrance qualifier, where $p_{11}=\Theta_{1}$ - core $/ 2=247.5^{\circ}-22.5^{\circ} / 2=236.25^{\circ}, p_{12}=\Theta_{1}+$ core $/ 2=247.5^{\circ}+22.5^{\circ} /$ $2=258.75^{\circ}, p_{21}=\Theta_{2}-$ core $/ 2=225.0^{\circ}-22.5^{\circ} / 2=213.75^{\circ}, p_{22}=\Theta_{2}+$ core $/$
$2=225.0^{\circ}+22.5^{\circ} / 2=236.25^{\circ}, \beta_{1}$ and $\beta_{2}$ are set to $22.5^{\circ}$. Thus, we get $\sigma_{12}=$ $\left(236.25^{\circ} / 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right)$ and $\sigma_{13}=\left(213.75^{\circ} / 22.5^{\circ} ; 236.25^{\circ} \backslash 22.5^{\circ}\right)$.
(c) Add directed arcs to point from state $q_{11}$ to states $q_{12}$ and $q_{13}$, and from state $q_{12}$ to state $q_{13}$. The arcs entering states $\dot{q}_{12}$ and $q_{13}$ are labelled with the fuzzy direction state entrance qualifiers $\sigma_{12}$ and $\sigma_{13}$, respectively. Thus, we get $\zeta_{1}\left(q_{11}, \sigma_{12}\right)=q_{12}$, $\zeta_{1}\left(q_{11}, \sigma_{13}\right)=q_{13}$, and $\zeta_{1}\left(q_{12}, \sigma_{13}\right)=q_{13}$.
(d) For state $q_{12}$ add a directed arc, which is labelled with $\sigma_{12}$, to point from that state to itself. Similarly a directed arc, which is labelled with $\sigma_{13}$, is added to point from state $q_{13}$ to itself. Thus, we get $\zeta_{1}\left(q_{12}, \sigma_{12}\right)=q_{12}$ and $\zeta_{1}\left(q_{13}, \sigma_{13}\right)=q_{13}$.
(e) Incrementito 3 and j to 2 .
(f) Since $\psi_{\mathrm{j}} \mathrm{j}=2$, is not the last link we let $\Theta_{0}=\Theta_{\mathrm{k}}=225.0^{\circ}$ and repeat Step 4.

Step 4. Creation of an accepting state for the link $\psi_{3}$ and a sequence of states which lie between state $q_{13}$ and the state which accepts $\psi_{3}$
The angle of $\psi_{3}$ is $\theta_{k}=285.0^{\circ}$, which lies in the angular interval the centre of which is $\Theta_{k}=292.5^{\circ}$ :
(a) Consider the centres $\Theta_{1}=247.5^{\circ}$ and $\Theta_{2}=270.0^{\circ}, \Theta_{3}=292.5^{\circ}$, i.e., $\mathrm{k}=3$, of the angular intervals which lie between $\theta_{0}=225.0^{\circ}$ and $\theta_{3}=292.5^{\circ}$.
(b) Create three (since $k=3$ ) new states $q_{14}, q_{15}$, and $q_{16}$, which have the fuzzy directions $\sigma_{14}=\left(p_{11} / \beta_{1} ; p_{12} / \beta_{2}\right), \sigma_{15}=\left(p_{21} / \beta_{1} ; p_{22} \backslash \beta_{2}\right)$, and $\sigma_{16}=\left(p_{31} / \beta_{1} ; p_{32} \backslash \beta_{2}\right)$ as their state entrance qualifiers, respectively, where $p_{11}=\theta_{1}$ - core $/ 2=247.5^{\circ}-$ $22.5^{\circ} / 2=236.25^{\circ}, \mathrm{p}_{12}=\Theta_{1}+$ core $/ 2=247.5^{\circ}+22.5^{\circ} / 2=258.75^{\circ}, \mathrm{p}_{21}=\theta_{2}$ - core $/ 2=270.0^{\circ}-22.5^{\circ} / 2=258.75^{\circ}, \mathrm{p}_{22}=\Theta_{2}+$ core $/ 2=270.0^{\circ}+22.5^{\circ} /$ $2=281.25^{\circ}, p_{31}=\Theta_{3}-$ core $/ 2=292.5^{\circ}-22.5^{\circ} / 2=281.25^{\circ}, p_{32}=\theta_{3}+$ core $/$ $2=292.5^{\circ}+22.5^{\circ} / 2=303.75^{\circ}, \beta_{1}$ and $\beta_{2}$ are set to $22.5^{\circ}$. Thus, we get $\sigma_{14}=$ $\left(236.25^{\circ} / 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right), \sigma_{15}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right)$, and $\sigma_{16}=\left(281.25^{\circ} / 22.5^{\circ} ; 303.75^{\circ} \backslash 22.5^{\circ}\right)$.
(c) Add directed arcs to point from state $q_{13}$ to states $q_{14}, q_{15}$, and $q_{16}$, from state $q_{14}$ to states $q_{15}$ and $q_{16}$ and from state $q_{15}$ to state $q_{16}$. The arcs entering states $q_{14}, q_{15}$, and $q_{16}$ are labelled with the fuzzy direction state entrance qualifiers $\sigma_{14}, \sigma_{15}$, and $\sigma_{16}$, respectively. Thus, we get $\zeta_{1}\left(q_{13}, \sigma_{14}\right)=q_{14}, \zeta_{1}\left(q_{13}, \sigma_{15}\right)=q_{15}, \zeta_{1}\left(q_{13}, \sigma_{16}\right)=q_{16}$, $\zeta_{1}\left(q_{14}, \sigma_{15}\right)=q_{15}, \zeta_{1}\left(q_{14}, \sigma_{16}\right)=q_{16}, \zeta_{1}\left(q_{15}, \sigma_{16}\right)=q_{16}$.
(d) For state $\mathrm{q}_{14}$ add a directed arc, which is labelled with $\sigma_{14}$, to point from that state to itself. Similar arcs, which are labelled with $\sigma_{1 s}$ and $\sigma_{16}$, are added for states $q_{1 s}$ and $q_{16}$, respectively. Thus, we get $\zeta_{1}\left(q_{14}, \sigma_{14}\right)=q_{14}, \zeta_{1}\left(q_{15}, \sigma_{15}\right)=q_{15}$, and $\zeta_{1}\left(q_{16}\right.$, $\left.\sigma_{16}\right)=q_{16}$.
(e) Increment ito 6 and j to 3.
(f) Since $\psi_{\mathrm{j}}, \mathrm{j}=3$, is the last link we go to Step 5.

Step 5. Setup of a terminal state
The last created state, $\mathrm{q}_{16}$, is considered as a terminal state. Thus, for the machine, $\mathrm{FM}_{1}$, $T_{1}=\left\{q_{16}\right\}$. A class, token class no. 1 , of the underlying token is embedded in $q_{16}$ with its fuzzy features, hence, $\mathrm{C}_{1}=\{$ token class no. 1$\}$. The fuzzy features are calculated as follows:
(a) The features of the token are: $\mathrm{f}_{\mathrm{s}}=0.02, \mathrm{f}_{\mathrm{e}}=0.98$, and $\mathrm{f}_{\mathrm{HW}}=0.64$.
(b) The features which are calculated above are fuzziffied by transforming them into $\mathrm{s} / \mathrm{z}-$ numbers. $\beta_{\mathrm{f}}$ and $\mathrm{n}_{\mathrm{f}}$ are set to 0.2 and 5 , respectively. Thus, the following fuzzy features are obtained: $\mathrm{F}_{8}=(0.0 / 0.0 ; 0.2 \backslash 0.2), \mathrm{F}_{\mathrm{e}}=(0.8 / 0.2 ; 1.0 \backslash 0.2)$, and $\mathrm{F}_{\mathrm{Hw}}$ $=(0.6 / 0.2 ; 0.8 \backslash 0.2)$.
Finally, a fuzzy sequential machine $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, T_{1}\right)$ is obtained where
(a) $\mathrm{C}_{1}=\{$ token class no. 1\},
(b) $\mathrm{Q}_{1}=\left\{q_{1}, i=0,1, \ldots, 6\right\}$,
(c) $\Sigma_{1}=\left\{\sigma_{1 \mathrm{p}} \mathrm{i}=1,2, \ldots, 6\right\}, \sigma_{11}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right), \sigma_{12}=\left(236.25^{\circ}\right.$ $\left./ 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right), \sigma_{13}=\left(213.75^{\circ} / 22.5^{\circ} ; 236.25^{\circ} \backslash 22.5^{\circ}\right),{ }_{14}=\left(236.25^{\circ}\right.$ $\left./ 22.5^{\circ} ; 258.75^{\circ} \backslash 22.5^{\circ}\right), \sigma_{15}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} \backslash 22.5^{\circ}\right)$, and $\sigma_{16}=$ ( $281.25^{\circ} / 22.5^{\circ} ; 303.75^{\circ} \backslash 22.5^{\circ}$ ),
(d) $\zeta_{1}$ is the next-state mapping function $\zeta_{1}: \mathrm{Q}_{1} \times \Sigma_{1} \rightarrow \mathrm{Q}_{1}$, where $\zeta_{1}\left(\mathrm{q}_{10}, \sigma_{11}\right)=\mathrm{q}_{11}$, $\zeta_{1}\left(q_{11}, \sigma_{11}\right)=q_{11}, \zeta_{1}\left(q_{11}, \sigma_{12}\right)=q_{12}, \zeta_{1}\left(q_{11}, \sigma_{13}\right)=q_{13}, \zeta_{1}\left(q_{12}, \sigma_{13}\right)=q_{13}, \zeta_{1}\left(q_{12}, \sigma_{12}\right)$ $=q_{12} \zeta_{1}\left(q_{13}, \sigma_{13}\right)=q_{13}, \zeta_{1}\left(q_{13}, \sigma_{14}\right)=q_{14}, \zeta_{1}\left(q_{13}, \sigma_{15}\right)=q_{15}, \zeta_{1}\left(q_{13}, \sigma_{16}\right)=q_{16,} \zeta_{1}\left(q_{14}\right.$, $\left.\sigma_{15}\right)=q_{15}, \zeta_{1}\left(q_{14}, \sigma_{16}\right)=q_{16}, \zeta_{1}\left(q_{15}, \sigma_{16}\right)=q_{16}, \zeta_{1}\left(q_{14}, \sigma_{14}\right)=q_{14}, \zeta_{1}\left(q_{15}, \sigma_{15}\right)=q_{15}$, and $\zeta_{1}\left(q_{16}, \sigma_{16}\right)=q_{16}$,
(e) $T_{1}=\left\{q_{16}\right\}$, where the fuzzy features which are embedded in $q_{16}$ were explained in Step 5 of this example.


Figure 10.3. (a) A multi-vertex token, and (b) a deterministic fuzzy sequential machine, $\mathrm{FM}_{2}$, obtained from this token.

### 10.3. MERGING OF FUZZY SEQUENTIAL MACHINES

Algorithm 6.3, which was used to merge FM's which recognize strokes, is also used to merge FM's that recognize multi-vertex tokens. Here, there are some points to highlight:
(a) Algorithm 6.3 deals with strokes. The current section deals with multi-vertex tokens.
(b) In Section 6.3, a machine, FM, recognizes reduced graphs which are, in general, different than the original graphs when dealing with strokes. However, in the current section, a machine FM, recognizes original graphs of multi-vertex tokens which are the same as their reduced graphs.

### 10.3.1. Example

Consider the token, $\Psi$, which is shown in Figure 10.3(a), and the machine, $\mathrm{FM}_{1}$, whose transition diagram is shown in Figure 10.2(b). $\mathrm{FM}_{1}$ could not recognize this token since the last level of the recognition tree is less than 3 , the number of the links in $\Psi$. Thus, to learn this token:
(a) A deterministic fuzzy sequential machine, $\mathrm{FM}_{2}=\left(\mathrm{C}_{2} \mathrm{Q}_{2}, \mathrm{~L}_{2} \zeta_{2}, \mathrm{~T}_{2}\right)$, Figure 10.3(b),
is created for the token, where $\mathrm{C}_{2}=\left\{\right.$ token class no. 2\}, $\mathrm{Q}_{2}=\left\{\mathrm{q}_{2 ;}, \mathrm{i}=0,1, \ldots, 7\right\}$, $\Sigma_{2}=\left\{\sigma_{2 i}, i=1,2, \ldots, 7\right\}, \zeta_{2}$ is the next-state mapping function $\zeta_{2}: Q_{2} \times$ $\Sigma_{2} \rightarrow Q_{2}$ which can be read from Figure 10.3(b), and $T_{2}=\left\{\mathrm{q}_{27}\right\}$. The state entrance qualifiers are: $\sigma_{21}=\left(258.75^{\circ} / 22.5^{\circ} ; 281.25^{\circ} / 22.5^{\circ}\right), \sigma_{22}=\left(236.25^{\circ} / 22.5^{\circ}\right.$; $\left.258.75^{\circ} 122.5^{\circ}\right), \sigma_{23}=\left(213.75^{\circ} / 22.5^{\circ} ; 236.25^{\circ} 22.5^{\circ}\right), \sigma_{24}=\left(191.25^{\circ} / 22.5^{\circ}\right.$; $\left.213.75^{\circ} 122.5^{\circ}\right), \sigma_{25}=\left(168.75^{\circ} / 22.5^{\circ} ; 191.25^{\circ} 122.5^{\circ}\right), \sigma_{26}=\left(146.25^{\circ} / 22.5^{\circ} ;\right.$ $\left.168.75^{\circ} \mathrm{V2.5} 5^{\circ}\right)$, and $\sigma_{27}=\left(123.75^{\circ} / 22.5^{\circ} ; 146.25^{\circ} / 22.5^{\circ}\right)$. In the terminal state, $\mathrm{q}_{27}$, three fuzzy features are embedded: $\mathrm{F}_{\mathrm{t}}=(0.0 / 0.0 ; 0.2 \backslash 0.2), \mathrm{F}_{\mathrm{e}}=(0.8 / 0.2 ; 1.0$ 10.2), and $\mathrm{F}_{\mathrm{HW}}=(0.2 / 0.2 ; 0.4 \backslash 0.2)$.
(b) The machine $\mathrm{FM}_{1}$ is merged with the machine $\mathrm{FM}_{2}$, described above. The merging process is detailed below.
Step 1. Forming of a nondeterministic fuzzy sequential machine
The sets of next states of $\mathrm{q}_{10}$ and $\mathrm{q}_{20}$ are $\mathrm{Q}_{10}=\left\{\mathrm{q}_{11}\right\}$ and $\mathrm{Q}_{20}=\left\{\mathrm{q}_{21}\right\}$, respectively:
(a) Delete the starting states $\mathrm{q}_{10}$ and $\mathrm{q}_{20}$ and the arcs emitting from them.
(b) Create a common starting state, $\mathrm{q}_{0}$.
(c) Since $\mathrm{q}_{11} \in \mathrm{Q}_{10}$, an arc, which is labelled with $\sigma_{11}$, is added to point from $\mathrm{q}_{0}$ to $\mathrm{q}_{11}$. Similarly, another arc, labelled with $\sigma_{21}$, is added to point from $q_{0}$ to $q_{21}$. Notice that $\sigma_{11}=\sigma_{21}=\sigma$ which makes $\zeta\left(q_{0}, \sigma\right)$ a multivalued function. The result is the transition diagram shown in Figure 10.4 which represents a nondeterministic machine, $\mathrm{FM}_{3}$.
Step 2. Constructing the transition tree for the machine $\mathrm{FM}_{3}$
While reading this step refer to Figure 10.5, which displays the whole transition tree.
The nodes are referenced by the nearby numbers.
(a) Generate the root node, node 0 , of the tree which has the state $\mathrm{q}_{0}$.
(b) For node 0 , which has $q_{0}$, the state entrance qualifier, $\sigma_{11}=\sigma_{21}$, causes a transition from $q_{0}$ to the set of next states $S=\left\{q_{11}, q_{21}\right\}$. Thus, a new node, node 1 , which is a son of node 0 with a state combination equal to $S$, is added. The arc from node 0 to node 1 is labelled with $\sigma_{11}=\sigma_{21}$. Node 1 , in turn, has $\left\{q_{11}, q_{21}\right\},\left\{q_{12}, q_{22}\right\},\left\{q_{13}\right.$, $\left.q_{23}\right\}$, and $\left\{q_{24}\right\}$ as the sets of next states for the state entrance qualifiers $\sigma_{11}=\sigma_{21}$, $\sigma_{12}=\sigma_{22}, \sigma_{13}=\sigma_{23}$, and $\sigma_{24}$, respectively. Thus a new node is created for each set of states, nodes 2 to 5 . The arcs from node 1 to nodes $2,3,4,5$, are labelled with


Figure 10.4. A nondeterministic fuzzy sequential machine, $\mathrm{FM}_{3}$, obtained from the machines of Figures 10.2(b) and 10.3(b).


Figure 10.5. The transition tree of the machine, $\mathrm{FM}_{3}$, of Figure 10.4 .
$\sigma_{11}=\sigma_{21}, \sigma_{12}=\sigma_{22}, \sigma_{13}=\sigma_{23}$, and $\sigma_{24}$, respectively. This process is repeated for newly generated nodes provided that a state combination does not appear in more than two levels of the tree. The final transition tree is shown in Figure 10.5.

Step 3. Obtain the transition diagram which represents a deterministic machine, FM, from the transition tree:
(a) Create the starting state, $\mathrm{q}_{0}$, of the transition diagram which corresponds to the root node of the transition tree.
(b) Scanning the tree starting from level 1, first we face node 1:

1. Node 1 contains the state combination $\left\{q_{11}, q_{21}\right\}$ for which no state was created. Thus, a new state, $q_{1}$, is generated.
2. An arc is added to point from state $q_{0}$, which corresponds to the node 0 , the father node of node 1 , to state $q_{1}$ which corresponds to node 1 . This arc is labelled with $\sigma_{1}=\sigma_{11}=\sigma_{21}$.

Next, we face node 2:

1. Node 2 contains the state combination $\left\{\mathrm{q}_{11}, \mathrm{q}_{21}\right\}$ for which a state, $\mathrm{q}_{1}$, was created.
2. An arc is added to point from state $q_{1}$, which corresponds to the node 1 , the father node of node 2 , to state $q_{1}$ which also corresponds to node 2 . This arc is labelled with $\sigma_{1}=\sigma_{11}=\sigma_{21}$.
This process is repeated for the other nodes which results in 10 unique states, excluding $\mathrm{q}_{0}$. Regarding node 12 , it contains a terminal state, $\mathrm{q}_{16}$. Thus, token class no. 1 and its fuzzy features, which are embedded in $q_{16}$, are copied to state, $q_{6}$ of Figure 10.6, which corresponds to node 12. Of course, $q_{6}$ of Figure 10.6 becomes a terminal state. Node 17 , which contains the terminal state $\mathrm{q}_{27}$, is handled similarly. The completed transition diagram is shown in Figure 10.6 which represents a deterministic fuzzy sequential machine $F M=(C, Q, \Sigma, \zeta, T)$, where $C=\{$ token class no. 1, token class no. 2$\}, Q=\left\{q_{i j} i=0,1, \ldots, 10\right\}, \Sigma=\left\{\sigma_{i}, i=1,2, \ldots, 10\right\}$, $\zeta$ is the next-state mapping function $\zeta: Q \times \Sigma \rightarrow \mathrm{Q}$, which can be read from the figure, and $T=\left\{q_{6}, q_{10}\right\}$. In the terminal state $q_{6}$, tokens belonging to token class no. 1 are recognized with the same set of fuzzy features that were embedded in state $\mathrm{q}_{16}$ of machine $\mathrm{FM}_{1}$. Also, the terminal state $\mathrm{q}_{10}$ recognizes tokens belonging to token


Figure 10.6. A deterministic fuzzy sequential machine, FM, obtained by merging the machines $\mathrm{FM}_{1}$ and $\mathrm{FM}_{2}$ of Figures 10.2 and 10.3, respectively.
class no. 2 with those fuzzy features which were embedded in state $q_{27}$ of machine $\mathrm{FM}_{2}$. The state entrance qualifiers are as follows: $\sigma_{1}=\left(258.75^{\circ} / 22.5^{\circ}\right.$; $\left.281.25^{\circ} \cup 22.5^{\circ}\right), \sigma_{2}=\left(236.25^{\circ} / 22.5^{\circ} ; 258.75^{\circ} \cup 22.5^{\circ}\right), \sigma_{3}=\left(213.75^{\circ} / 22.5^{\circ} ;\right.$ $\left.236.25^{\circ} \vee 22.5^{\circ}\right), \sigma_{4}=\left(236.25^{\circ} / 22.5^{\circ} ; 258.75^{\circ} \vee 22.5^{\circ}\right), \sigma_{5}=\left(258.75^{\circ} / 22.5^{\circ} ;\right.$ $281.25^{\circ}\left\llcorner 22.5^{\circ}\right), \sigma_{6}=\left(281.25^{\circ} / 22.5^{\circ} ; 303.75^{\circ} \vee 22.5^{\circ}\right), \sigma_{7}=\left(191.25^{\circ} / 22.5^{\circ}\right.$; $\left.213.75^{\circ} \vee 22.5^{\circ}\right), \sigma_{8}=\left(168.75^{\circ} / 22.5^{\circ} ; 191.25^{\circ} \mathrm{V} 22.5^{\circ}\right), \sigma_{9}=\left(146.25^{\circ} / 22.5^{\circ} ;\right.$ $\left.168.75^{\circ} \cup 22.5^{\circ}\right)$, and $\sigma_{10}=\left(123.75^{\circ} / 22.5^{\circ} ; 146.25^{\circ} \cup 22.5^{\circ}\right)$.

### 10.4. MODIFICATION OF A FUZZY SEQUENTIAL MACHINE

Let $\mathrm{FM}_{1}=\left(\mathrm{C}_{1}, \mathrm{Q}_{1}, \Sigma_{1}, \zeta_{1}, \mathrm{~T}_{1}\right)$ be the machine that was used when recognition was tried but failed. There are cases in which an unrecognized token can be learned by incorporating some modifications in $\mathrm{FM}_{1}$ so that it becomes capable of recognizing tokens belonging to the unknown token's class. These cases occur if the last level of the token's acceptance tree equals $m-1$, the number of links in the sequence, $\Psi$, as follows:
(a) The token's link sequence could terminate at one or more terminal states such that the remaining credit of the stroke did not fall below $\mathrm{THR}_{1}$, but could not be
recognized due to severe deviation of the token's features from the fuzzy features of the classes embedded in any of these terminal states, i.e., the acceptance degree of the token's features is less than $\mathrm{THR}_{2}$. Thus, in the following algorithm, a possibility can be investigated to add a new class, with its fuzzy features, in one of these terminal states.
(b) And / or the token's link sequence terminated in one or more states, NT, which are not terminal states and such that the remaining credit of the token is not less than $\mathrm{THR}_{1}$. In the following algorithm, a possibility can be searched to change one of such states into a terminal state, i.e., to append it to the set T , and to insert in it the class of the token with its fuzzy features.

A formal description of the algorithm which learns a token by modifying a token fuzzy sequential machine follows.

## Algorithm 10.4

Use: To learn a token by modifying a token fuzzy sequential machine
Input: 1. Sequence of links, $\Psi$, of the token to be learned
2. Reduced graph of the stroke from which the token, $\Psi$, was segmented
3. Acceptance tree of the token which is unrecognized by Algorithm 9.2
4. Token fuzzy sequential machine, $\mathrm{FM}_{1}$, of tokens which are already learned

Output: New fuzzy sequential machine, FM, which can recognize the underlying token and previously learned tokens

## Procedure:

Step 1. Calculation and fuzzification of the features of the token
(a) Find the features of the token $\Psi$, see Section 8.2.
(b) Fuzzify every feature found in (a) by transforming it into an $\mathrm{s} / 2$-number as follows:

1. Set both the left and right bandwidths to $\beta_{\mathrm{f}}$.
2. The range from 0.0 to 1.0 is divided into $n$ equal intervals. The left and right peak points, $p_{1}$ and $p_{2}$, are taken to be equal to the limits of the interval in which the value of the feature lies.

Suitable values of $\beta_{\mathrm{f}}$ and n were found to be 0.2 and 5 , respectively.


Figure 10.7. (a) A token to be learned by machine, FM, of Figure 10.6, and (b) its acceptance tree.

## Step 2. State modification

Find the leaf node, $d$, with the 2 -tuple ( $q, \omega$ ), the level of which equals $m-1$, and such that $d$ has the maximum remaining credit, $\omega$.
(a) If $q \in T_{1}$, where $T_{1}$ is the set of terminal states of $\mathrm{FM}_{1}$, then add a new token class, $\mathrm{c}_{\text {new }}$, in state q with the fuzzy features computed in Step 1. A new machine, $\mathrm{FM}=$ $(C, Q, \Sigma, \zeta, T)$, is obtained where $C=\left\{C_{\text {new }}\right\} \cup C_{1}, Q=Q_{1}, \Sigma=\Sigma_{1}, \zeta=\zeta_{1}$, and $T$ $=\mathrm{T}_{1}$.
(b) If $\mathrm{q} \notin \mathrm{T}_{1}$, then add q to the set of terminal states. Insert a token class, $\mathrm{c}_{\text {new }}$, in q with the fuzzy features computed in Step 1. A new machine, $\mathrm{FM}=(\mathrm{C}, \mathrm{Q}, \Sigma, \zeta, T)$, is obtained which is similar to $\mathrm{FM}_{1}$ with the exception that $\mathrm{C}=\left\{\mathrm{c}_{\text {new }}\right\} \cup \mathrm{C}_{1}$ and $\mathrm{T}=$ $\{q\} \cup T_{1}$.

### 10.4.1. Example

Figure 10.7(a) shows a token consisting of two links. When trying to recognize this token using the machine, FM, of Figure 10.6, the acceptance tree of Figure 10.7(b) is obtained revealing that the token is unrecognized since the link sequence terminated at node 2 which has $\mathrm{q}_{3} \notin \mathrm{~T}$ with a remaining credit $=1.0>\left(\mathrm{THR}_{1}=0.6\right)$. Thus, to learn this token, modifications are incorporated into machine FM as follows:

Step 1. Calculation and fuzzification of the features of the token
(a) The calculated features of the token are: $f_{8}=0.0, f_{c}=1.0$, and $f_{\mathrm{HW}}=0.61$.
(b) By taking $\beta_{\mathrm{f}}=0.2$ and $\mathrm{n}=5$, the above features are fuzziffied to obtain the following fuzzy features: $\mathrm{F}_{\mathrm{t}}=(0.0 / 0.0 ; 0.2 \backslash 0.2), \mathrm{F}_{\mathrm{e}}=(0.8 / 0.2 ; 1.0 \backslash 0.0)$, and $\mathrm{F}_{\mathrm{Hw}}=(0.6 / 0.2 ; 0.8 \backslash 0.2)$.
Step 2. State modification
Node 2 is the only node the level of which equals 2 , the number of token links. This node contains state $\mathrm{q}_{3}$ which is not a terminal one. Thus, case (b) of this step applies. State $\mathrm{q}_{3}$ is added to the set of terminal states. A new a token class, token class no. 3 , is inserted in $q_{3}$ with the fuzzy features computed in Step 1 . A new machine, $F M=(C, Q, \Sigma, \zeta, T)$, is obtained which is similar to the original FM with the exception that $\mathrm{C}=\{$ token class no. 1 , token class no. 2 , token class no. 3$\}$, and $T=\left\{q_{3}, q_{6}, q_{10}\right\}$.

## SUMMARY

In this chapter, the token learning process of the CASR system was presented. The learning algorithm determines how to learn an unrecognized multi-vertex token depending on the last level of its acceptance tree, which is passed by the token recognition process, as follows:
(a) If the last level of the acceptance tree is less than the number of links in the token, then a fuzzy sequential machine is generated from the input token. This machine can recognize the input token and variants of it. The generated machine and the old machine, which was used in recognition but failed to recognize the token, are merged into one machine which can recognize the input token and tokens of the old machine and variants of these tokens.
(b) If the last level of the acceptance tree equals the number of links in the token, then some modifications are introduced into the machine which was used in recognition, which is sufficient to learn the token. The output is a modified fuzzy sequential machine which can recognize the input token and tokens of the old machine and variants of them.

## Learning of Token Strings



## OVERVIEW

In this chapter, tokens are recombined into meaningful sets of tokens; logical token strings. An algorithm to learn logical token strings is presented. The data flow diagram of the process of learning of token strings, Figure 11.1, consists of the following subprocesses:
(a) Manual Token Combining: which accepts as inputs the segmented tokens of strokes, classes assigned to the tokens using Algorithm 9.2, and reduced graphs of strokes. Here, the user interactively deals with a program which displays a stroke on the screen so that he can group tokens into logical token strings.
(b) Appending of New Logical Token Strings: It accepts the inputs of the above subprocess, logical token strings, and a data base of already learned logical token strings. Here, logical token strings which do not have similar strings in the data base are appended to the data base. Appended logical token strings are associated with possible interpretations and their fuzzy features.

### 11.1. FROM TOKENS TO TOKEN STRINGS

In Chapters 8 and 9, it was shown how to segment a stroke into tokens and recognize the segmented tokens. Sometimes an individual token, alone, does not constitute the main body of an integral number of characters. Thus, we re-combine tokens


Figure 11.1. Data flow diagram of the process of Learning of Token Strings in the CASR system.
into strings to obtain meaningful token combinations which can represent the main body of a whole number of characters. This chapter concerns such token combining.

### 11.1.1. Definitions

(a) An Arabic subword which consists of a single connected component is a main stroke. If a subword consists of more than one connected component then the first written component is a main stroke and the others are secondary strokes. Figure 11.2(a) shows one Arabic word which consists of one subword. This subword consists of six strokes; the largest one is the main stroke and the others are secondary strokes.
(b) The set $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$ refers to the tokens into which the reduced graph of a stroke is segment using Algorithm 8.1, where N is the number of tokens. A straight line approximation, $G$, of the main stroke of the subword, which is shown in Figure 11.2(a), is shown in Figure 11.2(b). The graph, G, contains one loop set consisting


Figure 11.2. (a) An Arabic word consisting of one main stroke and five secondary strokes, (b) a graph, $G$, of the main stroke, and (c) the reduced graph, $\mathrm{G}^{\prime}$, of the main stroke and the path $\mu_{s c}$.
of a single loop. The reduced graph, $\mathrm{G}^{\prime}$, of G is shown in Figure 11.2(c) with the loop set being represented by vertex 16 . The reduced graph is segmented into the following tokens, see Figure 11.3: $\Psi_{1}=(1,2,3,4), \Psi_{2}=(5,4,6,7,8,9,10,11$, 12), $\Psi_{3}=(13,12,14,15), \Psi_{4}=(16), \Psi_{5}=(15,17)$, and $\Psi_{6}=(18,17,19,20,21$, $22,23,24$ ).


Figure 11.3. $\quad$ Tokens of the main stroke of Figure 11.2: (a), $\Psi_{1}$, (b) $\Psi_{2}$, (c) $\Psi_{3}$, (d) $\Psi_{4}$, (e) $\Psi_{5}$, and (f) $\Psi_{6}$.


Figure 11.4. A common shape which can represent a secondary stroke or an isolated Hamza character.
(c) A Common SHape, (CSH), is a stroke or part of it which can represent:

1. a secondary stroke, or
2. many different characters depending on the type, number, and position of secondary strokes assigned to it.

Figure 11.4 shows a common shäpe which can represent a secondary stroke or a character (Hamza) without any additional secondary strokes. Figure 11.5(a) shows a common shape which can represent six different characters depending on the type, number and position of assigned secondary strokes. These characters are: $B a, T a$, Tha, Noon, Ya, Hamza, see Figure 11.5(b-g).

Sometimes, a single isolated token does not represent any common shape, .e.g., $\Psi_{1}$


Figure 11.5. (a) A common chape which can represent six different characters depending on assigned secondary strokes, (b) Ba character, (c) Ta character, (d) Tha character, (e) Noon character, (f) Ya character, and (g) Hamza character.


Figure 11.6. A stroke which can be segmented into three tokens representing one common shape.
of Figure 11.3(a). The same token can represent a whole common shape if combined with another token, e.g., if $\Psi_{1}$ is combined with $\Psi_{2}$, then $\Psi_{1}$ represents a common shape. A common shape can be just a part of a token, e.g., in token $\Psi_{2}$, the links between vertices 5 and 7, see Figure 11.2(c), represent a whole common shape. More than one token may combine to form a single common shape, e.g., the graph of Figure 11.6, if segmented, will produce three tokens, where the three, all together, represent a single common shape.
(d) A common shape can appear

1. isolated,
2. at the start, middle, or end of a stroke, or
3. any combination of the above cases, e.g., it can appear isolated in one stroke, at the start of a second stroke, and at the end of a third stroke. This does not mean that every common shape can appear in any position of a stroke.
For example, Figure 11.7(a) shows a graph which can be segmented into three tokens. The three combined tokens form one common shape (Ayn) which can

(a)

(b)

Figure 11.7. (a) A three-token stroke representing one common shape which can appear in the middle of a stroke, and (b) an example stroke in which the common shape of (a) appears in the middle of a stroke.

(a)

(c)

(d)

Figure 11.8. (a) A stroke representing one common shape which can appear isolated, (b) the common shape of (a) at the start, (c) the common shape of (a) in the middle, and
(d) the common shape of $(a)$ at the end of a stroke.
appear in the middle of a stroke, as shown in Figure 11.7(b). Figure 11.8(a) shows another graph consisting of one token which can represent one common shape. This common shape can appear isolated, at the start, in the middle, or at the end of a stroke as shown in Figures 11.8(a - d).
(e) A secondary stroke is an isolated common shape. For example, each of the two dots in Figure 11.2(a) produces a single isolated vertex graph, i.e., a single vertex token, which represents an isolated common shape that can be a dot secondary stroke. This does not mean that every isolated common shape is a secondary stroke.
(f) A set of secondary strokes assigned to a common shape is called a secondary stroke combination, ssc. Figure 11.9 shows common Arabic secondary strokes. Figure 11.10 shows some valid Arabic secondary stroke combinations with examples.
(g) Two tokens $\Psi_{i}$ and $\Psi_{j}$ are adjacent if they have a common vertex, e.g., $\Psi_{1}$ and $\Psi_{2}$,

```
.- ^| | N
```

Figure 11.9. Common Arabic secondary strokes, from left to right: Dot, Two Connected Dots, Three Connected Dots, Vertical Bar, Hamza, and Madda.


Figure 11.10. Some valid Arabic secondary stroke combinations.


Figure 11.11. A token string, ts: the links between vertices 1 to 4 constitute $\mathrm{CSH}_{1}$, the links between vertices 5 to 7 constitute $\mathrm{CSH}_{2}$, and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$.

Figure 11.3(a, b), are adjacent since vertex 4, see Figure 11.2(c), is common between them.
(h) A token string, ts, is a set of tokens such that they constitute a graph consisting of one component.
(i) A logical token string is a token string, ts, such that:

1. ts may have only one token which represents a loop set,
2. whether, alone or combined with secondary strokes, the string can represent at least one set of CSH's. Let $P=\left\{p_{1}, p_{2}, \ldots\right\} \neq \varnothing$ represent these sets, and
3. For every possible set of CSH's $p_{i} \in P$, the string, ts, can not be divided into two strings, $\mathrm{ts}_{1}$ and $\mathrm{ts}_{2}$ with sets of CSH's, $\mathrm{p}_{\mathrm{il}}$ and $\mathrm{p}_{\mathrm{i} 2}$, respectively, such that $\mathrm{p}_{\mathrm{i} 1} \cup \mathrm{p}_{\mathrm{i} 2}=\mathrm{p}_{\mathrm{i}}$.

For example, in Figure 11.3, the set of tokens ts $=\left\{\Psi_{1}, \Psi_{2}\right\}$ constitute a single component graph since the tokens $\Psi_{1}$ and $\Psi_{2}$ are adjacent, i.e., they have a common vertex, vertex 4, hence, ts is a token string. None of the tokens $\Psi_{1}$ or $\Psi_{2}$ is a loop set. The string ts represents one set, $\mathrm{p}_{11}$, of three CSH's denoted by $\mathbf{C S H}_{1}, \mathbf{C S H}_{2}$, and $\mathrm{CSH}_{3}$, which are shown, in Figure 11.11 where the links between vertices 1 to 4 constitute $\mathrm{CSH}_{1}$, the links between vertices 5 to 7 constitute $\mathrm{CSH}_{2}$, and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$. Also, ts can not be divided into two strings with $p_{1}$ and $p_{2}$ being their corresponding sets of CSH's such that $p_{1} \cup p_{2}=$ $\mathrm{p}_{11}$. Thus, ts is a logical token string.
(j) The length of a dot token is zero and the length of a loop set token is equal to the sum of lengths of the links constituting the loop set.


Figure 11.12. Strokes consisting of the same set of tokens and different characters: (a) Waw character, and (b) Meem character.
(k) The length of a token string is the sum of the lengths of the individual tokens constituting the string.

### 11.1.2. Features of Token Strings

A string of tokens has its own features to distinguish it from other strings of tokens.
In the CASR system, the following features are used:
(a) Token Codes: These are the identification codes of the classes of the tokens which constitute the string. Some token strings have the same list of token codes, however they differ in other features.
(b) Token / String Length Ratios, $\gamma_{i} \mathrm{i}=1,2, \ldots, \mathrm{n}$, where $\gamma_{i}$ is the ratio of the length of token $\Psi_{i}$ to the total length of the string.
(c) Intersection Vertices: This feature applies to token strings which have more than one token and a token representing a loop set. In this case, every intersection vertex that is a connection point between the loop set and another token in the string is described by two features:
and

$$
\begin{align*}
& \mathrm{xf}=\left(\mathrm{x}-\mathrm{x}_{\min }\right) /\left(\mathrm{x}_{\max }-\mathrm{x}_{\min }\right)  \tag{11.1}\\
& \mathrm{yf}=\left(\mathrm{y}-\mathrm{y}_{\min }\right) /\left(\mathrm{y}_{\max }-\mathrm{y}_{\min }\right) \tag{11.2}
\end{align*}
$$

where $x$ and $y$ are the $x \& y$ coordinates of the intersection vertex and $x_{\text {min }}, x_{\text {max }}, y_{\text {min }}$, and $y_{\max }$ are the boundary coordinates of the rectangle which encloses the loop set. This is important for strings which have loop sets and have the same token codes. For example, both graphs of Figure 11.12(a, b) represents a single token string consisting of the same set of tokens: a loop set token and a multi-vertex token. They only differ in the position of the intersection point relative to the loop set. The
feature of intersection vertices is used to distinguish between such token strings.

### 11.2. LEARNING OF TOKEN STRINGS

An important component in the CASR system is to extract feasible combinations of token strings, i.e., logical strings. To know whether a token string is logical there should be some kind of a data base in which logical token strings along with their meaning stored. The underlying token string is compared with stored logical strings to find whether there is a match. Such a data base must be generated first, which is the process that we call learning of token strings. An algorithm for this purpose follows.

## Algorithm 11.2

Use: To learn logical token strings in a stroke
Input: 1. For every input stroke:
i. The set, $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$, of segmented tokens of the stroke
ii. Token codes, i.e., classes assigned to the tokens $\left\{\Psi_{1}, \Psi_{2}, \ldots\right.$, $\left.\Psi_{\mathrm{N}}\right\}$ using Algorithm 9.2
iii. Reduced graph of the stroke
2. Data base, TS, of already learned logical token strings

Output: New data base of learned logical token strings

## Procedure:

Step 1. Manual token combining
The tokens of the stroke are combined into logical token strings. This is a manual process since the human judgement is indispensable to decide whether a token string can represent a whole number of CSH's. To facilitate this task, an interactive program was developed to display a stroke on the screen and allow the user to group tokens into logical strings.

Step 2. Appending of new logical token strings
In this step, we have to mention first that the features of token / string ratios and intersection vertices are stored in the data base, TS, as fuzzy numbers as it will be clear in Step 2(b) of this algorithm.

For every possible logical token string, ts, generated in Step 1:
(a) A test is performed to make sure that it is new, i.e., it does not exist in the data base,

TS, as follows:

1. Find the features of ts, see Section 11.1.2.
2. Verify that there is no token string ts* $\in$ TS such that
i. ts and ts* have the same list of token codes,
ii. The overall acceptance degree, $a_{\gamma}$, of the features of token / string ratios is not less than a specified threshold $\mathrm{THR}_{\mathbf{7}}$, where

$$
\begin{equation*}
a_{\gamma}=\frac{1}{n} \Sigma \pi_{r_{1}}\left(\gamma_{i}\right), i=1,2, \ldots, n \tag{11.3}
\end{equation*}
$$

where n is the number of tokens in ts and $\pi_{r_{1}}\left(\gamma_{1}\right)$ is the acceptance degree of the deterministic feature $\boldsymbol{\gamma}_{\mathrm{i}}$ of ts in the corresponding fuzzy feature $\Gamma_{i}$ of $\mathrm{ts}^{*}$.
iii. The overall acceptance degree, $a_{x y}$ of the features of intersection vertices is not less than $\mathrm{THR}_{\mathrm{xy}}$, where

$$
\begin{equation*}
a_{x y}=\frac{1}{2 \times n_{x y}} \Sigma\left(\pi_{x_{i}}\left(x f_{i}\right)+\pi_{r_{i}}\left(y f_{i}\right), i=1,2, \ldots, n_{x y}\right. \tag{11.4}
\end{equation*}
$$

where $n_{x y}$ is the number of intersection vertices, which exist in the loop set, if any, in ts and $\pi_{x_{i}}\left(x f_{i}\right)$ and $\pi_{r_{i}}\left(y f_{i}\right)$ are the acceptance degrees of the features $\mathrm{xf}_{\mathrm{i}}$ and $\mathrm{yf}_{\mathrm{i}}$ of ts in the corresponding fuzzy features $\mathrm{X}_{1}$ and $Y_{i}$ of ts*.

If there is no token string, $\mathrm{ts}^{*} \in \mathrm{TS}$, which fulfils the above conditions, then ts is new.
(b) If ts is new then:

1. Transform the features of token / string ratios and features of intersection vertices, which exist in a loop.set, of ts to fuzzy features, i.e., s/z-numbers, as follows:
i. Both the left and right bandwidths are set to $\beta_{t}$.
ii. The range from 0.0 to 1.0 is divided into $n_{s s}$ equal intervals. The left and right peak points are taken to be equal to the limits of the interval in which the value of the feature lies.

In general, the features of token / string length ratios have values of $\beta_{s}$ and $n_{w s}$
which differ from those of features of intersection vertices.
2. Associate with ts a set of possible interpretations where each interpretation is a set of CSH's. For each common shape, the following information is stored:
i. The positions in which the common shape can appear, i.e., start, middle, end, isolated, or any combination of these,
ii. The secondary strokes it can represent, if any, and
iii. The characters it can represent, where for each character the following information is stored:

- the character class, and
- the type and position of the secondary stroke combination that is required to be assigned to the common shape to form the character.

3. Append ts with its fuzzy features and interpretations to the data base TS.

### 11.3. EXAMPLE

The logical token strings of the stroke of Figure 11.2(b), the tokens of which are shown in Figure 11.3, are learned by Algorithm 11.2 as follows:

Step 1. Manual token combining
Following Definition 11.1.1(i), an Arabic reader can deduce eight logical token strings as shown in Figure 11.13, where $\mathrm{ts}_{1}=\left\{\Psi_{1}, \Psi_{2}\right\}, \mathrm{ts}_{2}=\left\{\Psi_{2}\right\}, \mathrm{ts}_{3}=\left\{\Psi_{3}\right\}, \mathrm{ts}_{4}=\left\{\Psi_{4}\right\}, \mathrm{ts}_{5}=$ $\left\{\Psi_{4}, \Psi_{5}\right\}, \mathrm{ts}_{6}=\left\{\Psi_{5}\right\}, \mathrm{ts}_{7}=\left\{\Psi_{4}, \Psi_{5}, \Psi_{6}\right\}$, and $\mathrm{ts}_{8}=\left\{\Psi_{6}\right\}$.

Step 2. Appending of new logical token strings
Each of the eight logical token strings which were formed in Step 1 is tested to see whether it is new. If it is new then it is appended to the data base, TS. For the first string, $\mathrm{ts}_{1}$, this step is applied as follows:
(a) Perform a test to find whether $\mathrm{ts}_{1}$ is new:

1. The features of token / string ratios of $\mathrm{ts}_{1}$ are $\gamma_{1}=0.28, \gamma_{2}=0.72$. There is no features of intersection vertices since $\mathrm{ts}_{1}$ does not contain a loop set token connected to a multi-vertex token.
2. If it is assumed that the set of learned logical token strings, TS, is initially empty, then $\mathrm{ts}_{1}$ is considered new.

(a)

(c)

(e)

(b)

(d)
(f)

(h)

Figure 11.13. Logical token strings of the stroke of Figure 11.2(b): (a) $t s_{1}$; the links between vertices 1 to 4 constitute $\mathrm{CSH}_{1}$, the links between vertices 5 to 7 constitute $\mathrm{CSH}_{2}$ and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$, (b) ts $\mathrm{s}_{2}$ : the links between vertices 5 to 7 constitute $\mathrm{CSH}_{1}$, and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$, (c) $\mathrm{ts}_{3}: \mathrm{CSH}_{1}$ or $\mathrm{CSH}_{4}$, (d) $\mathrm{ts}_{4}: \mathrm{CSH}_{5}$, (e) $\mathrm{ts}_{5}: \mathrm{CSH}_{6}$ or $\mathrm{CSH}_{7}$, (f) $\mathrm{ts}_{6}: \mathrm{CSH}_{8}$, (g) $\mathrm{ts}_{7}: \mathrm{CSH}_{9}$, and (h) $\mathrm{ts}_{8}$ : $\mathrm{CSH}_{10}$.
(b) Since $\mathrm{ts}_{1}$ is new it is added to TS äs follows:

1. The features of token / string ratios are fuzziffied by transforming them to $\mathrm{s} / \mathrm{z}$ numbers as follows:
i. The left and right bandwidths are set to $\beta_{\mathrm{t}}=0.25$.
ii. The range from 0.0 to 1.0 is divided into $n_{u}=4$ equal intervals, i.e. the interval length is 0.25 .

Finally, we obtain the fuzzy features $\Gamma_{1}=(0.25 / 0.25 ; 0.50 \backslash 0.25)$ and $\Gamma_{2}=$ ( $0.50 / 0.25 ; 0.75 \backslash 0.25$ ) which correspond to $\gamma_{1}=0.28$ and $\gamma_{2}=0.72$, respectively.
2. The set of possible interpretations of $\mathrm{ts}_{1}$ has only one interpretation, $\mathrm{p}_{11}$, where $\mathrm{p}_{11}$ consists of three CSH's denoted by $\mathrm{CSH}_{1}, \mathrm{CSH}_{2}$, and $\mathrm{CSH}_{3}$, which are shown, in Figure 11.13(a). The links between vertices 1 to 4 constitute $\mathrm{CSH}_{1}$, the links between vertices 5 to 7 constitute $\mathrm{CSH}_{2}$, and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$. String $\mathrm{ts}_{1}$ can not be divided into two strings with $p_{1}$ and $p_{2}$ being their corresponding sets of CSH's such that $p_{1} \cup p_{2}=p_{11}$. For the right-most common shape, $\mathrm{CSH}_{1}$, the following information is stored:
i. $\quad \mathrm{CSH}_{1}$ can appear at the start or in the middle of a stroke.
ii. $\quad \mathrm{CSH}_{1}$ can not represent any secondary stroke.
iii. $\quad \mathrm{CSH}_{1}$ can be read as different characters depending on the number and position of assigned secondary strokes as follows:

- If there is a single dot below, then it represents $B a$.
- If there are two isolated dots or a horizontal dash above, then it represents Ta.
- If there is a set of three isolated dots, a single dot which is located above a horizontal dash, or three connected dots (similar to caret) located above, then it represents Tha character.
- If there is a single dot above, then it represents Noon.
- If there are two isolated dots or a horizontal dash below, then it represents Ya character.
- If there is Hamza above, then it represents Hamza.

For the second common shape, $\mathrm{CSH}_{2}$, the following information is stored:
i. $\mathrm{CSH}_{2}$ can appear only in the middle of a stroke.
ii. $\mathrm{CSH}_{2}$ can not represent any secondary stroke.
iii. $\mathrm{CSH}_{2}$ has one reading, Meem character, without any secondary stroke.

For the last common shape, $\mathrm{CSH}_{3}$, the following information is stored:
i. $\quad \mathrm{CSH}_{3}$ can appear at the start or in the middle of a stroke.
ii. $\quad \mathrm{CSH}_{3}$ can not represent any secondary stroke.
iii. $\mathrm{CSH}_{3}$ has different readings as follows:

- If it is not associated with any secondary stroke, then it is Ha.
- If it has one dot above, then it represents Kha character.
- If it has one dot below, then it represents Jeem character.

3. Append $\mathrm{ts}_{1}$ with its fuzzy features and interpretations to the data base TS.

The other seven logical token strings, $\mathrm{ts}_{2}$ to $\mathrm{ts}_{8}$, are manipulated similarly. Finally, we get the data base, TS, with eight strings:

1. $\mathrm{ts}_{1}=\left\{\Psi_{1}, \Psi_{2}\right\}$, which was described above in detail,
2. $\mathrm{ts}_{2}=\left\{\Psi_{2}\right\}$ having one interpretation $\mathrm{p}_{21}=\left\{\mathrm{CSH}_{1}, \mathrm{CSH}_{3}\right\}$, where $\mathrm{CSH}_{1}$ and $\mathrm{CSH}_{3}$ are as before,
3. $\mathrm{ts}_{3}=\left\{\Psi_{3}\right\}$ having two interpretations: $\mathrm{p}_{31}=\left\{\mathrm{CSH}_{4}\right\}$ and $\mathrm{p}_{32}=\left\{\mathrm{CSH}_{1}\right\}$, where $\mathrm{CSH}_{1}$ is as before. For $\mathrm{CSH}_{4}$,
i. It appears either isolated or at the end of a stroke.
ii. It does not represent any secondary stroke.
iii. It can be read as follows:

- If it is without secondary strokes, then it is $R a$ character.
- If it has one dot above, then it is Zain character.

4. $t_{s_{4}}=\left\{\Psi_{4}\right\}$ having one interpretation $p_{41}=\left\{\mathrm{CSH}_{5}\right\}$. For $\mathrm{CSH}_{5}$,
i. It appears isolated.
ii. It can represent a secondary stroke, Sokoon.
iii. It can be read as follows:

- If it is without any secondary stroke, then it is either an isolated Arabic character Ha or Arabic numeral "o". Whether it is Ha or Arabic numeral "0" is a contextual problem which is out of scope of this research.
- If it has two dots or a horizontal dash above, then it is closed Ta.

5. $\quad \mathrm{ts}_{5}=\left\{\Psi_{4}, \Psi_{5}\right\}$ having two interpretations $\mathrm{p}_{51}=\left\{\mathrm{CSH}_{6}\right\}$ and $p_{52}=\left\{\mathrm{CSH}_{7}\right\}$. For $\mathrm{CSH}_{6}$,
i. It appears at the start or in the middle of a stroke.
ii. It does not represent any secondary stroke.
iii. It can be read as follows:

- If it has one dot above, then it is Pha character.
- If it has two dots or a horizontal dash above, then it is Qaf character.

For $\mathrm{CSH}_{7}$,
i. . It appears isolated, at the start, in the middle, or at the end of a stroke.
ii. It does not represent any secondary stroke.
iii. It can be read as follows:

- If it has one vertical bar above, then it is TTa character.
- If it has one vertical bar above and one dot to the right of the bar, then it is $Z a$ character.

6. $\mathrm{ts}_{6}=\left\{\Psi_{5}\right\}$ having one interpretation $\mathrm{p}_{61}=\left\{\mathrm{CSH}_{8}\right\}$. For where $\mathrm{CSH}_{8}$,
i. It appears isolated,
ii. It can represent Fatha, Kasra, or Two Connected Dots secondary stroke.
iii. It can be read as a minus sign character without any secondary stroke.
7. . $\mathrm{ts}_{7}=\left\{\Psi_{4}, \Psi_{5}, \Psi_{6}\right\}$ having one interpretation $\mathrm{p}_{71}=\left\{\mathrm{CSH}_{9}\right\}$. For $\mathrm{CSH}_{9}$,
i. It appears isolated or at the end of a stroke.
ii. It can not represent any secondary stroke.
iii. It can be read as different characters:

- If it is without secondary strokes, then it is Sad.
- If it has a dot above, then it is Dad.

8. $\mathrm{ts}_{8}=\left\{\Psi_{6}\right\}$ having one interpretation $\mathrm{p}_{81}=\left\{\mathrm{CSH}_{10}\right\}$. For $\mathrm{CSH}_{10}$,
i. It appears isolated or at the end of a stroke.
ii. It can not represent any secondary stroke.
iii. It can be read as Noon character provided that there is a dot above.

## SUMMARY

In this chapter, tokens were recombined into meaningful sets of tokens; logical token strings. An algorithm to learn logical token strings was presented. It consists of two steps. First, the user interactively deals with a program which displays a stroke on the screen so that he can group tokens into logical token strings. Second, formed logical token strings which do not have similar strings in the data base of learned logical token strings are appended to the data base. Appended logical token strings are associated with possible interpretations and their fuzzy features.

# Line Extraction and Stroke 



## Ordering

## OVERVIEW

In this chapter, a method will be developed to extract lines from binary images of pages of handwritten Arabic text and order their constituent strokes. A data flow diagram of the process is shown in Figure 12.1, which consists of the following subprocesses:
(a) Separating Main and Secondary Strokes: where strokes which can represent secondary strokes are marked. Remaining strokes are main strokes.
(b) Constructing ak-Connected Graph with Artificial Vertices: Here main strokes represent the vertices of a $k$-connected graph in addition to additional auxiliary artificial vertices.
(c) Finding the Shortest Spanning Tree of the $k$-Connected Graph: which results in the vertices representing main strokes of each line to be connected by one path. The paths corresponding to adjacent lines are connected by links between artificial vertices.
(d) Removing Artificial Vertices: which produces a multi-component graph where each component corresponds to one line.
(e) Extracting Lines and Ordering Main Strokes: where the uppermost path corresponds to the first line, the next lower path to the second line, and son on. The vertices of each path are arranged from right to left according to their order of appearance in the path. The right-most vertex corresponds to the first main stroke


Figure 12.1. Data flow diagram of the Line Extraction and Stroke Ordering process.
in the line, and son on.
(f) Presenting Secondary Strokes to Main Strokes: where every secondary stroke is presented to the nearest main stroke.

### 12.1. BACKGROUND

Part of off-line text recognition is to extract lines and to arrange the words in the same order as they were written. In printed text, for each line a horizontal baseline can be
calculated and used to extract lines. Then, words are extracted by raster-scanning the extracted lines. Although the process may be trivial in the case of printed text, the situation does not hold true in handwriting. In handwriting, a very high variability is expected in all respect. The causes could be the writing habit, style, education, mood, health and other conditions of the writer. These factors make it difficult to calculate a horizontal baseline for each line of text and to use the standard projection methods to extract lines $[6,8,9]$. In mixed text / graphics images, Hough transform is used to detect sets of connected components that lie along a given straight line $[80,81]$. However, severe difficulties are encountered in applying Hough transform since many parameters have to be computed in advance. The determination of these parameters assumes available knowledge about the string characteristics in the image which is difficult to obtain since the text strings themselves are unknown. Moreover, Hough transform assumes that the centroids of characters constituting a text string are collinear to a certain degree of accuracy. This can not be fulfilled even in printed text since printed characters vary in their size, height, etc.

In this chapter, a method will be developed to extract lines from binary images of handwritten Arabic text and order their constituent strokes. The method depends on identifying main and secondary strokes, which is the subject of the next section.

### 12.2. SEPARATING MAIN AND SECONDARY STROKES

In Arabic handwriting, the main strokes of any given line are aligned to some line whether straight or curved. Secondary strokes are written above or below this line. Main strokes are considered as the basic building blocks of a line on which secondary strokes are hooked. Thus, extraction of lines depends on first separating main strokes from secondary strokes, which is the purpose of the algorithm of this section.

### 12.2.1. Definitions

(a) The set $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$ refers to unknown tokens which are segmented from an unknown stroke using Algorithm 8.1, where N is the number of tokens. This is the same as Definition 11.1.1(b) which is repeated here for the reader convenience.
(b) codes is the set of token codes (classes) each of which is a candidate to represent
a token $\Psi_{i}, i=1,2, \ldots, N$. These codes are found using Algorithm 9.2.
(c) CODES is the set of all possible vectors code ${ }_{\text {vec }}=\left(\operatorname{code}_{1}, \operatorname{code}_{2}, \ldots\right.$, code $\left._{N}\right)$, code $_{i}$ $\in$ codes $_{i}$.

For example, for a stroke consisting of three tokens, we may find $\operatorname{codes}_{1}=\{2,4\}$, $\operatorname{codes}_{2}=\{1,2,3\}$, and $\operatorname{codes}_{3}=\{5\}$. The set, CODES, contains $2 \times 3 \times 1=6$ vectors as follows: $(2,1,5),(4,1,5),(2,2,5),(4,2,5),(2,3,5)$, and $(4,3,5)$.

An algorithm to separate main strokes from secondary stroke candidates now follows.

## Algorithm 12.2

Use: To find main strokes and secondary stroke candidates
Input: 1. For each stroke:
i. The set, $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$, of segmented tokens of the stroke
ii. Token codes, i.e., classes assigned to the tokens $\left\{\Psi_{1}, \Psi_{2}, \ldots\right.$, $\left.\Psi_{N}\right\}$ using Algorithm 9.2
iii. Reduced graph of the stroke
2. Data base, TS, of already learned logical token strings

Output: Either the stroke is marked as a secondary stroke candidate or it is left unmarked, i.e., it is a main stroke. If it is a secondary stroke candidate, then the sets SS and CHAR contain the secondary strokes and characters which the stroke can represent, respectively.

## Procedure:

Step 1. Recognizing the whole stroke as a single token string
Create the set CODES, see Definitions 12.2.1. Consider the whole stroke as a single unknown token string, ts $=\left(\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right)$. For every vector code ${ }_{\text {vec }} \in$ CODES, use the token codes which exist in vector code ${ }_{\text {vec }}$ and correspond to the tokens of ts to find the token string ts* $\in$ TS which accepts ts with the maximum possible degree, $a \geq \mathrm{THR}_{a}$, as follows:
(a) Find the features of ts, see Section 11.1.2.
(b) Find that token string, ts* $\in$ TS, such that

1. ts and ts* have the same list of token codes,
2. The acceptance degree $a=\min \left(a_{y}, a_{x y}\right)$ of $t$ is the maximum value that can be obtained, where

$$
\begin{equation*}
a_{\gamma}=\frac{1}{N} \Sigma \pi_{r_{i}}\left(\gamma_{i}\right), i=1,2, \ldots, N \tag{12.1}
\end{equation*}
$$

is the overall acceptance degree of the features of token / string ratios, $N$ is the number of tokens in ts and $\Pi_{\Gamma_{1}}\left(Y_{1}\right)$ is the acceptance degree of the deterministic feature $\gamma_{i}$ of ts in the corresponding fuzzy feature $\Gamma_{i}$ of ts*, and

$$
\begin{equation*}
a_{x y}=\frac{1}{2 \times n_{x y}} \Sigma\left(\pi_{x_{i}}\left(x f_{i}\right)+\pi_{r_{i}}\left(y f_{i}\right), i=1,2, \ldots, n_{x y}\right. \tag{12.2}
\end{equation*}
$$

is the overall acceptance degree of the features of intersection vertices, $\mathrm{n}_{\mathrm{xy}}$ is the number of intersection vertices, which exist in the loop set, if any, in ts and $\pi_{x_{i}}\left(x f_{i}\right)$ and $\pi_{r_{i}}\left(y f_{i}\right)$ are the acceptance degrees of the features $x f_{i}$ and $y f_{i}$ of ts in the corresponding fuzzy features $X_{i}$ and $Y_{i}$ of $t s^{*}$.
Step 2. Creating list of main strokes and list of secondary stroke candidates If no token string, ts*, could be found in Step 1, then the stroke is a main stroke. Otherwise, consider every interpretation, $p$, of $t s^{*}$. If $p$ consists only of one common shape, CSH, which can be isolated and can work, alone, as a secondary stroke, then
(a) Add the stroke to the list of secondary stroke candidates.
(b) Add the secondary stroke which CSH can represent to the set SS, where SS is the set of all secondary strokes which the stroke can represent.
(c) Add the characters, if any, which CSH can represent to the set CHAR which is the set of all characters which the stroke can represent. The set, CHAR, is used in the next chapter.

Otherwise, add the stroke to the list of main strokes.

### 12.2.2. Example

The largest stroke of Figure 11.2(a) is segmented to produce the set of tokens $\left\{\Psi_{i}\right.$ $i=1,2, \ldots, 6\}$, which are shown in Figures 11.3. Using Algorithm 9.2, the sets of token codes are $\operatorname{codes}_{1}=\{1\}, \operatorname{codes}_{2}=\{2\}, \operatorname{codes}_{3}=\{3\}, \operatorname{codes}_{4}\{4\}, \operatorname{codes}_{5}=\{5\}$, and $\operatorname{codes}_{6}$
$=\{6\}$. The set CODES consists of one vector element code ${ }_{v e c}=(1,2,3,4,5,6)$. Algorithm 12.2 is applied to this stroke as follows.

Step 1. Recognizing the whole stroke as a single token string
For the token string $\mathrm{ts}=\left\{\Psi_{\mathrm{i}}, \mathrm{i}=1,2, \therefore, 6\right\}$ and the vector $\operatorname{code}_{\text {vec }}=(1,2,3,4,5,6)$, there is no token string, ts* $\in$ TS, which fulfils the conditions which are listed in Step 1 of Algorithm 12.2.

Step 2. Creating list of main strokes and list of secondary stroke candidates Since no token string, ts*, could be found in Step 1, the stroke is a main stroke.

Algorithm 12.2 is also applied to the other five strokes of Figure 11.2(a). The result is that these five strokes are counted as secondary strokes candidates with their sets of secondary strokes being as follows: $\mathrm{SS}_{1}=\mathrm{SS}_{2}=\{$ Dot $\}, \mathrm{SS}_{3}=\mathrm{SS}_{4}=\{$ Damma $\}$, and $\mathrm{SS}_{5}$ $=\{$ Two connected dots $\}$. The set of characters which these secondary stroke candidates represent are as follows: $\mathrm{CHAR}_{1}=\mathrm{CHAR}_{2}=$ NULL, $\mathrm{CHAR}_{3}=\mathrm{CHAR}_{4}=\{$ Waw $\}$, and $\mathrm{CHAR}_{5}=\{$ Minus sign $\}$.

### 12.3. LINE EXTRACTION AND STROKE ORDERING

Figure 12.2 shows one page of handwritten Arabic text paragraph after being smoothed. In this figure, the lines are not exactly horizontal which is different than the situation in printed text. Figure 12.3 is the horizontal projection of the image of Figure 12.2. It is clear that the maxima of the projection are difficult to be found which complicates the process of extracting the exact number of lines and the line contents. Thus, the standard projection methods can not be used to extract lines.

Usually, a page of cursive text (e.g., Arabic, English, . . ., etc.) consists of lines with a minimum distance between lines, $D_{L}$. A line consists of words written from one end to its other end (from right to left in Arabic and from left to right in English). A word consists of one or more subwords (main strokes). A subword may have secondary strokes which are written close to the subword. In this section, it is assumed that a secondary stroke is one of those shown in Figure 11.9. There can be a maximum interstroke distance between strokes belonging to one line which is equal to $D_{s}$. Almost, the distance $D_{L}$ is larger than $\mathrm{D}_{\mathrm{s}}$. One may assume that for each line there can be a minimum distance path that extends from one end to the other end of the line and spans all the main strokes that


Figure 12.2 Smoothed image of handwritten Arabic text paragraph.
belong to that line. Such a path of each line has no intersections with those of the neighbouring lines. In fact, this kind of path is a shortest spanning tree that spans the main strokes of the underlying line. A minimum spanning tree that spans the main strokes in the whole page consists mainly of several paths; one for each line. These paths are connected by interline links that have minimum distance.

It is assumed that strokes are represented by direct straight line approximations, obtained using Algorithm 2.2. The main strokes of a graph will be viewed as vertices of a $k$-connected graph, $\Gamma$. The degree of the graph $\Gamma$ depends on the main stroke density in a line and the line density in the page, i.e., the number of main strokes per line and the number of lines per page. Each vertex in $\Gamma$ is connected to the nearest $k$ vertices. To find the cost matrix of $\Gamma$, the minimum and maximum $x \& y$ coordinates, $x m i n, x m a x, y m i n$, and ymax, of the direct straight line approximation of each main stroke are found. Two distances are defined between any two direct straight line approximations, $G_{i}$ and $G_{j}$, representing two main strokes. These are the $x$ and $y$ distances. The $x$ distance, $d_{x}$, is calculated as follows: a horizontal line is artificially drawn between the minimum and


Figure 12.3 Horizontal projection of the image of Figure 12.2.
maximum x coordinates of the first direct straight line approximation. A similar horizontal line is artificially drawn for the second direct straight line approximation. If these lines overlap then $d_{x}=0$, otherwise if the line of $G_{i}$ lies to the right of that of $G_{j}$, then $d_{x}=x \min _{i}$ - max $_{j}$ If the line of $G_{i}$ lies to the left of $G_{j}$ then $d_{x}=x \min _{j}-$ xmax $_{i}$. The $y$ distance, $d_{y}$, is calculated similarly with the artificial lines being vertical. The Euclidean distance between $G_{i}$ and $G_{j}$ is

$$
\begin{equation*}
d=\left(d_{x}^{2}+d_{y}^{2}\right)^{0.5} \tag{12.3}
\end{equation*}
$$

After calculating the distance between every pair of vertices / direct straight line approximations, a vertex is only connected to the nearest k vertices thus forming the k connected graph, $\Gamma$. This results in reducing the time needed to compute the shortest spanning tree.

Figure 12.4 shows the main and secondary strokes of Figure 12.2 represented as large and small squares, respectively. The figure shows also the shortest spanning tree (SST), calculated using the algorithm of Prim [74], considering only the main strokes and taking the graph degree $\mathrm{k}=8$. The main strokes of each of the first three lines are spanned


Figure 12.4. Shortest spanning tree using Equation 12.3.
by a path that does not intersect with the paths of the neighbouring lines. These paths are connected by minimum distance links. For the forth line, it was not possible to obtain a path that does not intersect with that of the third line. The reason is that, as it is clear from the figure, there are two consecutive main strokes in the fourth line whose distance in larger than the distance between the right-most one of these strokes and another stroke belonging to the third line. However, this can be remedied by forcing the links to be horizontal as possible as we can. This can be achieved by letting the distance between any two direct straight line approximations be proportional to the angle of that link. The more vertical the link is the more the cost. One way to fulfill this requirement is to calculate a factor, f, where

$$
\begin{equation*}
\mathrm{f}=1.0+\gamma \times \tan ^{-1}\left(\mathrm{~d}_{\mathrm{y}} / \mathrm{d}_{\mathrm{x}}\right) /(\pi / 2) \tag{12.4}
\end{equation*}
$$

where $\mathrm{d}_{\mathrm{y}}$ and $\mathrm{d}_{\mathrm{x}}$ are the y and x distances between the two direct straight line approximations, respectively, and calculated as before. The constant, $\gamma$, is selected such that the distance between any two consecutive main strokes belonging to one line becomes smaller than the distance between any of these strokes and any other stroke belonging to another line. In practice, a value of $\gamma=10$ worked properly in almost all the cases. Then,


Figure 12.5. Shortest spanning tree using Equation 12.5.
the distance becomes:

$$
\begin{equation*}
\mathrm{d}=\mathrm{f} \times\left(\mathrm{d}_{\mathrm{x}}^{2}+\mathrm{d}_{\mathrm{y}}^{2}\right)^{0.5} \tag{12.5}
\end{equation*}
$$

According to this new formula for distance calculation, horizontal links are not changed while right-angled links get the maximum punishment. Figure 12.5 shows the new spanning tree calculated using the new distance formula. One non-intersecting path is obtained for each line. Every pair of two adjacent paths are connected by a minimum distance link which tends be horizontal and looks illogical. However, remember that vertical distances are punished which makes them tend to be horizontal; hence longer in appearance.

In Figure 12.5, the links connecting the paths may be located any where; the only restriction is that they must have the minimum distance. If the degree of both ends of an interpath link is three then it is easily located and removed. However, this is not guaranteed since an interpath link may connect the two ends of two paths which makes these ends have a degree of two which is equal to the degrees of other vertices of the paths. If this happens, a difficulty is confronted in removing such links. Thus, we would like these links to be formed in positions which are known to us in order to be able to


Figure 12.6. Shortest spanning tree after the addition of artificial vertices.
remove them. One way to achieve this is to add to the graph, $\Gamma$, a set, S , of n artificial vertices. The vertices in S are eqi-spaced and lie on a vertical line which is located to the right-most edge of the page. The number, $n$, of such vertices is selected to be larger than the maximum number of lines that can be accommodated in one page. For example, if the page is of A4 size, then $n$ may be set to 100 . In this way, the right-most main stroke of every line is guaranteed to have a horizontal link connecting it to one of the artificial vertices. The distance between the artificial vertices is not punished and is calculated using Equation 12.3. This results in the paths being connected via links between artificial vertices. This idea is shown incorporated in Figure 12.6. The set S has 20 eqi-spaced artificial vertices. The paths connecting main strokes are connected via links that connect artificial vertices which are removed from the shortest spanning tree to obtain a multicomponent graph $\Gamma^{\prime}$. Each component corresponds to one line. Lines are extracted by finding end vertices of the graph $\Gamma^{\prime}$, i.e., vertices with degree equal to one. If a vertex $v_{1}$ has a degree equal to one then a path is followed starting from $v_{1}$ until some other end vertex, $\mathrm{v}_{2}$, is reached. The vertices of the path from $\mathrm{v}_{1}$ to $\mathrm{v}_{2}$ constitute the main strokes of one line. The line starts at the right-most vertex of $\left\{\mathrm{v}_{1}, \mathrm{v}_{2}\right\}$ and the order of writing of


Figure 12.7. The paths after the removal of artificial vertices and assignment of secondary strokes to main strokes.
main strokes is the same as they appear in the path from the start vertex to the end vertex. The uppermost path corresponds to the uppermost line in the page then the next lower one and so forth.

What remains is to present the secondary strokes to main strokes. It is well known that secondary strokes are written close to main strokes. Thus, a secondary stroke is presented to the nearest main stroke. If the strokes are represented by direct straight line approximations, then the distance between a secondary stroke $s$ and a main stroke $m$ is defined to be the distance between the pair of vertices, $v_{s}$ and $v_{m}$ where $v_{s}$ and $v_{m}$ are vertices in the direct straight line approximations of the secondary and main strokes, respectively, such that this distance is minimum.

Figure 12.7 shows the extracted lines after the removal of artificial vertices and presenting of secondary strokes to main strokes. A comparison between this figure and that of Figure 12.2 shows that the lines, order of main strokes in the lines, and secondary to main stroke presentation could be correctly found. Figure 12.8 shows each main stroke surrounded with the presented secondary strokes by rectangles. The contents of each


Figure 12.8 Sets of main-secondary strokes surrounded by rectangles.
rectangle can be the input to subsequent recognition algorithms. Whether a secondary stroke truly belongs to a main stroke or not is determined by algorithms of the next chapter. In most of the cases the presentation of this preprocessing stage is correct. The resultant algorithm to extract lines and order strokes of a page is formulated below.

## Algorithm 12.3

Use: $\quad$ To extract lines, order main strokes, and present secondary stroke candidates to main strokes

Input: 1. Direct straight line approximations of strokes of one page
2. List of main strokes
3. List of secondary strokes

Output: 1. Ordered lines
2. Ordered main strokes
3. Secondary stroke presentation

## Procedure:

Step 1. Construct a k-connected graph, $\Gamma$, as follows:
(a) The main strokes are represented as vertices in $\Gamma$.
(b) Add to $\Gamma$ a set, S , of n artificial vertices. The vertices in S are eqi-spaced and lie on a vertical line which is located to the right-most edge of the page. The number, $n$, of such vertices is selected to be larger than the maximum number of lines that can be accommodated in one page.
(c) The cost matrix of the graph, $\Gamma$, is calculated as follows:
i. For each two vertices, $v_{i}$ and $v_{j}$, if at least one vertex is not artificial, then the distance is $\mathrm{d}=\mathrm{f} \times\left(\mathrm{d}_{\mathrm{x}}^{2}+\mathrm{d}_{\mathrm{y}}^{2}\right)^{0.5}$.
ii. The distance between the artificial vertices are calculated using the original formula $d=\left(d_{x}^{2}+d_{y}^{2}\right)^{0.5}$.
(d) Each vertex in $\Gamma$ is connected to the nearest k vertices based on the cost matrix.

Step 2. Find the shortest spanning tree (SST) for the graph, $\Gamma$.
Step 3. Remove the links which connect artificial vertices from SST to obtain a multicomponent graph, $\Gamma^{\prime}$, where each component corresponds to one line.
Step 4. Lines are extracted by finding end vertices of the graph $\Gamma^{\prime}$, i.e., vertices with degree equal to one. If a vertex $v_{1}$ has a degree equal to one then a path is followed starting from $v_{1}$ until some other end vertex, $v_{2}$, is reached. The vertices of the path from $\mathrm{v}_{1}$ to $\mathrm{V}_{2}$ constitute the main strokes of one line. The line starts at the right-most vertex of $\left\{\mathrm{v}_{1}, \mathrm{v}_{2}\right\}$ and the order of writing of main strokes is the same as they appear in the path from the start vertex to the end vertex. The uppermost path corresponds to the uppermost line in the page then the next lower one and so forth.

Step 5. Every secondary stroke is presented to the nearest main stroke.

The algorithm assumes that text lines are approximately justified at the right hand margin. If one line were indented, it could form a link to the line above or below. For example, in Figure 12.9(a), $G_{1}$ is way to the left of $G_{2}$, where $G_{1}$ and $G_{2}$ represent the right-most main strokes of two consecutive lines. According to Equation 12.5, a link between $G_{1}$ and $G_{3}$ is formed since the distance between them is shorter than the distance between $G_{1}$ and any of the artificial vertices at the right margin. Similarly, if the right hand


Figure 12.9. Cases where Algorithm 12.3 could fail: (a) indented line, and (b) skewed right hand margin.
margin were skewed, as it would be if the paper was not straight in the scanner, the method could fail for the same reason, see Figure 12.9(b).

## SUMMARY

In this chapter, a method was developed to extract lines from binary images of pages of handwritten Arabic text. It depends on first identifying main and secondary strokes. Then, lines are extracted and main strokes of extracted lines are arranged in the same order as they were written. Finally, the secondary strokes are presented to main strokes. At the end, an ordered list of main strokes each with the corresponding number of presented secondary strokes is obtained.

## Word Formation



## OVERVIEW

This chapter is concerned with the last component, word formation, of the recognition process in the CASR system, where strokes are interpreted as sets of characters. A data flow diagram of this process is shown in Figure 13.1, which consists of the following stages:

- CSH INTERPRETATIONS OF MAIN STROKES, Section 13.1: where all possible CSH interpretations of main strokes are enumerated and represented in a tree data structure. This stage consists of the following subprocesses:
(a) Finding CSH Enumeration Sets: where the best sets of logical token strings, with their corresponding CSH sets, which represent the main strokes are found.
(b) Generating Enumeration and Requirement Trees: The logical token strings and CSH sets which are output by the previous subprocess are combined to form what we call Emmeration and Requirement Trees (ERT's), in which information about secondary strokes required to associate CSH's is included.
- CHARACTER FORMATION, Section 13.2: where ERT's are combined with candidate secondary strokes to form characters. It consists of the following subprocesses:
(a) Solving of Assignment Problems: where assignment problems are


Figure 13.1. Data flow diagram of the Word Formation process of the CASR system.
formulated and solved to assign secondary strokes to form characters.
(b) Selecting the Minimum Cost Solution: Of the solutions obtained in (a), the solution which exhibits the minimum cost is selected. The words which correspond to this solution are output in addition to some redundant secondary strokes which could not be assigned to CSH's.

- MANIPULATING REDUNDANT SECONDARY STROKES, Section 13.3:

Redundant secondary strokes are manipulated to form some other characters which
are inserted in their proper places within lines. The final result is a list of ordered lines where a line is an ordered list of words.

### 13.1. CSH INTERPRETATIONS OF MAIN STROKES

After learning of logical token strings, it becomes possible to extract the sets of common shapes, where each set is a valid interpretation of the whole main stroke. In this section, an algorithm will be developed to achieve this goal.

### 13.1.1. Definitions

(a) The set $\left\{\Psi_{1}, \Psi_{2} \ldots, \Psi_{N}\right\}$ refers to unknown tokens which are segmented from an unknown stroke using Algorithm 8.1, where N is the number of tokens. This is the same as Definition 11.1.1(b) which is repeated here for the reader convenience.
(b) codes $_{\mathrm{i}}$ is the set of token identification codes each of which is a candidate to represent a token $\Psi_{i} \mathrm{i}=1,2, \ldots, N$. These codes are found using Algorithm 9.2.
(c) CODES is the set of all possible vectors code ${ }_{v e c}=\left(\operatorname{code}_{1}, \operatorname{code}_{2}, \ldots\right.$, code $\left._{N}\right)$, code $_{i}$ E codes ${ }_{i}$.
(d) CUTS represents the set of combinations of unknown token strings cuts $=$ (uts ${ }_{1}$, $u^{2} s_{2}, \ldots$, uts $_{q}$ ) such that $u t_{i} \cap u t s_{j}=\varnothing, i \neq j, \cup u t_{i}=\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$. For example, for a stroke consisting of three tokens, the set CUTS contains the following combinations:

1. cuts $_{1}=\left(\right.$ uts $_{1}$, uts $_{2}$, uts $\left._{3}\right)$, where uts ${ }_{1}=\left\{\Psi_{1}\right\}$, uts $_{2}=\left\{\Psi_{2}\right\}$, and uts ${ }_{3}=\left\{\Psi_{3}\right\}$,
2. cuts $_{2}=\left(\text { uts }_{1}, \text { uts }\right)_{2}$, where uts $s_{1}=\left\{\Psi_{1}, \Psi_{2}\right\}$, and uts ${ }_{2}=\left\{\Psi_{3}\right\}$,
3. cuts $_{3}=\left(\right.$ uts $_{1}$, uts $\left._{2}\right)$, where uts ${ }_{1}=\left\{\Psi_{1}\right\}$, and uts ${ }_{2}=\left\{\Psi_{2}, \Psi_{3}\right\}$, and
4. cuts $_{4}=\left(\right.$ uts $\left._{1}\right)$, where uts $=\left\{\Psi_{1}, \Psi_{2}, \Psi_{3}\right\}$.

Now, a formal description of an algorithm to find CSH interpretations of main strokes is presented.

## Algorithm 13.1

Use: To enumerate all possible CSH interpretations of every input main stroke
Input: 1. Tokens of strokes, i.e., the set, $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$, of segmented tokens for each stroke
2. Classes of token, i.e., classes assigned to the tokens $\left\{\Psi_{1}, \Psi_{2}, \ldots, \Psi_{N}\right\}$ using Algorithm 9.2
3. Reduced graphs of strokes
4. List of main strokes
5. Data base, TS, of already learned logical token strings

Output: All possible CSH interpretations (Enumeration and Requirement Trees, ERT's) of every main stroke

## Procedure

For every input main stroke, the following steps are performed:

## Step 1. Finding CSH enumeration sets

Prepare the sets CODES and CUTS for the input stroke.
For every vector code $_{\text {vec }} \in \operatorname{CODES}$ do
For every cuts $=\left(\right.$ uts $_{1}$, uts $_{2}, \ldots$, uts $\left._{\mathbf{q}}\right) \in$ CUTS do
(a) For every uts ${ }_{i}=\left(\Psi_{j}, \Psi_{j+1}, \ldots, \Psi_{k}\right), i=1,2, \ldots, q$, do

Use the token codes which exist in vector code vec and correspond to the tokens of $\mathrm{uts}_{\mathrm{i}}$ to find the token string $\mathrm{ts}_{\mathrm{i}}{ }^{*} \in \mathrm{TS}$ which accepts $u_{t} s_{i}$ with the maximum possible degree, $a_{i} \geq$ THR $_{\mathbf{a}}$ :

1. Find the features of uts $_{\mathrm{i}}$, see Section 11.1.2.
2. Find that token string, $\mathrm{ts}_{\mathrm{i}}{ }^{*} \in \mathrm{TS}$, such that
i. utsi ${ }_{i}$ and $t s_{i}$ * have the same list of token codes,
ii. The acceptance degree $a_{i}=\min \left(a_{y}, a_{x y}\right)$ of $u t s_{i}$ is the maximum degree that can be obtained, where

$$
\begin{equation*}
a_{\gamma}=\frac{1}{N} \sum \pi_{r_{i}}\left(\gamma_{i}\right), i=1,2, \ldots, N \tag{13.1}
\end{equation*}
$$

is the overall acceptance degree of the features of token / string ratios, $N$ is the number of tokens in uts ${ }_{i}$ and $\pi_{r_{1}}\left(Y_{1}\right)$ is the acceptance degree of the deterministic feature $\gamma_{i}$ of uts $_{\mathbf{i}}$ in the corresponding fuzzy feature $\Gamma_{i}$ of $\mathrm{ts}_{i}{ }^{*}$, and

$$
\begin{equation*}
a_{x y}=\frac{1}{2 \times n_{x y}} \Sigma\left(\pi_{x_{i}}\left(x f_{i}\right)+\pi_{y_{i}}\left(y f_{i}\right), j=1,2, \ldots, n_{x y}\right. \tag{13.2}
\end{equation*}
$$

is the overall acceptance degree of the features of intersection vertices, $n_{\mathrm{xy}}$ is the number of intersection vertices, which exist in the loop set, if any, in uts and $\pi_{x_{l}}\left(x f_{i}\right)$ and $\pi_{r_{1}}(y)$ are the acceptance degrees of the features $\mathrm{xf}_{\mathrm{i}}$ and $\mathrm{yf}_{\mathrm{i}}$ of uts in the corresponding fuzzy features $\mathrm{X}_{\mathrm{i}}$ and $Y_{i}$ of $t s i s^{*}$.
(b) The acceptance degree, $a$, of cuts equals to the minimum of the $a_{i}$ degrees, $i=1,2, \ldots, q$.

If $\mathrm{a} \geq \mathrm{TRH}_{\boldsymbol{p}}$ then, depending on the number of token strings, q , in cuts, there are two cases:

1. $\mathrm{q}=1$, i.e., cuts $=\left(\right.$ uts $\left._{1}\right)$. Consider every interpretation, p, of $\mathrm{ts}_{1}{ }^{*}$ :
i. If p consists only of one CSH which can be isolated, then retain $p$ as a valid interpretation of $u$ uts $_{1}$.
ii. Otherwise, if p consists of n common shapes $\mathrm{CSH}_{1}, \mathrm{CSH}_{2}$, . . ., $\mathrm{CSH}_{n}$, such that $\mathrm{CSH}_{1}$ can appear at the start of a stroke, all the common shapes $\mathrm{CSH}_{2}$ to $\mathrm{CSH}_{n-1}$ can appear in the middle of a stroke, and $\mathrm{CSH}_{n}$ can appear at end of a stroke, then retain $p$ as a valid interpretation of uts.
2. $q>1$.
i. Consider every interpretation, p , of $\mathrm{ts}_{1}{ }^{*}$. If p consists of n common shapes $\mathrm{CSH}_{1}, \mathrm{CSH}_{2}, \ldots, \mathrm{CSH}_{n}$, such that $\mathrm{CSH}_{1}$ can appear at the start of a stroke and all the common shapes $\mathrm{CSH}_{2}$ to $\mathrm{CSH}_{n}$ can appear in the middle of a stroke, then retain $p$ as a valid interpretation of uts.
ii. Consider every interpretation, $p$, of $\mathrm{ts}_{\mathrm{i}}{ }^{*}, \mathrm{i}=2,3, \ldots, \mathrm{q}-1$. If $p$ consists of $n$ common shapes $\mathrm{CSH}_{1}, \mathrm{CSH}_{2}, \ldots, \mathrm{CSH}_{n}$, such that all the common shapes $\mathrm{CSH}_{1}$ to $\mathrm{CSH}_{n}$ can appear in the middle of a stroke, then retain $p$ as a valid interpretation of uts.
iii. Consider every interpretation, $p$, of $t s_{q}{ }^{*}$. If $p$ consists of $n$
common shapes $\mathrm{CSH}_{1}, \mathrm{CSH}_{2}, \ldots, \mathrm{CSH}_{n}$, such that all the common shapes $\mathrm{CSH}_{1}$ to $\mathrm{CSH}_{\mathrm{n}-1}$ can appear in the middle of a stroke and $\mathrm{CSH}_{\mathrm{n}}$ can appear at the end of a stroke, then retain $p$ as à valid interpretation of uts ${ }_{q}$.
(c) Retain the set of sets of interpretations $S P=\left\{P_{1}, P_{2}, \ldots, P_{q}\right\}$, where $P_{i}$ $=\left\{p_{i 1}, p_{i,}, \ldots\right\} \neq 0$, is the set of valid interpretations of uts ${ }_{j}$ and $p_{i j}=$ $\left\{\mathrm{CSH}_{1}, \mathrm{CSH}_{2}, \ldots\right\} \neq \varnothing$, is a set of common shapes.
The k combinations, cuts $_{1}{ }^{*}$, cuts $_{2}{ }^{*}, \ldots$, cuts ${ }_{2}{ }^{*}$, and their corresponding sets $\mathrm{SP}_{1}, \mathrm{SP}_{2}$, , . ., $\mathrm{SP}_{\mathbf{b}}$, respectively, which have the maximum k acceptance degrees are retained. If at least one set, $S P=\left\{P_{1}, P_{2}, \ldots, P_{q}\right\}$, could not be found, such that none of the sets $P_{i}, i$ $=1,2, \ldots, q$, is empty, then the input main stroke is unknown.

Step 2. Generating a CSH Enumeration and Requirement Tree (ERT) for every set, $\mathrm{SP}_{\mathrm{i}}$
The sets, $\mathbf{S P}_{i}$ 's, which are generated in Step 1, contain data about the possible interpretations of a main stroke. Other important data is added to these sets which specify for every interpretation what additional information is required to accompany that interpretation to produce one complete set of characters representing the main stroke. This additional data includes the characters which a CSH can represent, the required type and position of the secondary stroke combination. This expansion of interpretations is stored in what we call an Enumeration and Requirement Tree (ERT). The rationale behind selecting a tree data structure is that saving such detailed information requires huge amount of memory. These interpretations have many common information. Using a tree is one way to reduce information duplication, hence, reduce the memory requirements. The information which reside in the nodes of every path from a leaf node to the root node of the tree represents a complete interpretation, with its requirements, of the main stroke. To generate an ERT of an $\mathrm{SP}_{\mathrm{i}}$ first we need to define the following two recursive functions:
ERT_Expand_1(current_node, j) \{
For every possible interpretation $p_{p_{k}} \in P_{j}, k=1,2, \ldots$, of token string uts ${ }_{j}$, call the function ERT_Expand_2(current_node, $\mathbf{j}, \mathrm{k}, 1$ ).

```
ERT_Expand_2(current_node, j, k,l) {
    For every possible character, char, which CSH
    (a) Add a son to the node, current_node, which contains the following
                information:
            1. the character class, char,
            2. a secondary stroke combination that is required to accompany \(\mathrm{CSH}_{\text {}}\) to form that character,
3. The required position of the secondary stroke combination relative to \(\mathrm{CSH}_{b}\) and
4. The token string number, j .
(b) If \(l\) is less than the number of CSH's in \(\mathrm{p}_{\mathrm{j} k}\) then call the function ERT_Expand_2(son of current_node, \(\mathrm{j}, \mathrm{k}, \mathrm{l}+\mathrm{l}\) ).
Otherwise, if \(j\) is less than the number of token strings in \(\mathrm{SP}_{\mathrm{i}}\) then call the function ERT_Expand_1(son of current_node, \(\mathrm{j}+1\) )
    }
}
To generate an ERT for an \(\mathrm{SP}_{\mathrm{i}}\) :
(a) Create the root node of the ERT. The root node does not contain any special information; it is just a root.
(b) Call the function ERT_Expand_1(root node of ERT, 1).
```


### 13.1.2. Example

The main stroke of Figure 13.2(b) is segmented to produce the set of tokens $\left\{\Psi_{i}\right.$ $\mathrm{i}=1,2, \ldots, 6\}$, which were shown in Figure 11.3. Using Algorithm 9.2, the sets of token identification codes are codes $_{1}=\{1\}$, codes $_{2}=\{2\}, \operatorname{codes}_{3}=\{3\}, \operatorname{codes}_{4}\{4\}, \operatorname{codes}_{5}=$ $\{5\}$, and $\operatorname{codes}_{6}=\{6\}$. To enumerate all possible sets of CSH's, with their requirements, of this main stroke, Algorithm 13.1 is applied as follows:

Step1. Finding CSH enumeration sets
The set CODES consists of one vector element code $e_{w e c}=(1,2,3,4,5,6)$. The set CUTS has many combinations of unknown token strings one of which is cuts $=\left(\text { uts }_{1}, \text { uts }_{2} \text {, uts }\right)_{3}$ ) where uts ${ }_{1}=\left\{\Psi_{1}, \Psi_{2}\right\}$, uts $s_{2}=\left\{\Psi_{3}\right\}$, and uts $s_{3}=\left\{\Psi_{4}, \Psi_{5}, \Psi_{6}\right\}$.


- (a)

(b)

(c)

Figure 13.2. (a) An Arabic word consisting of one main stroke and five secondary strokes, (b) a graph, $G$, of the main stroke, and (c) the reduced graph, $\mathrm{G}^{\prime}$, of the main stroke and the path $\mu_{8}$.

For the vector code $_{v a c}=(1,2,3,4,5,6)$ and the set cuts $=\left(\right.$ uts $_{1}$, uts $s_{2}$, uts $\left._{3}\right)$ :
(a) The codes of uts $i=1,2,3$, are extracted from code $e_{\text {vec }}$. Based on the logical token strings which were learned in Example 11.3, these token strings are recognized as $u t s_{1}=t s_{1}, u t s_{2}=t s_{3}$, and $u t s_{3}=t s_{7}$, each of which has an acceptance degree $a_{i}=1.0$ $>\left(\mathrm{THR}_{\mathbf{i}}=0.6\right), \mathrm{i}=1,2,3$, where $t \mathrm{~s}_{1}, \mathrm{ts}_{3}, \mathrm{ts} \in \mathrm{TS}$, see Figure 13.3.

(a)

(b)

(c)

Figure 13.3. Logical token strings of the main stroke of Figure 13.2(b): (a) $t s_{1}$ : the links between vertices 1 to 4 constitute $\mathrm{CSH}_{1}$, the links between vertices 5 to 7 constitute $\mathrm{CSH}_{2}$, and the links between vertices 7 to 12 constitute $\mathrm{CSH}_{3}$, (b) $\mathrm{ts}_{3}: \mathrm{CSH}_{1}$, and (c) $\mathrm{ts}_{7}$ : $\mathrm{CSH}_{9}$.
(b) The acceptance degree of cuts is $a=\min \left(a_{1}, a_{2}, a_{3}\right)=1.0>\operatorname{THR}_{2}=0.6$. Since cuts consists of more than one token string:

1. $u t s_{1}=t s_{1}$ has one interpretation $\mathrm{p}_{11}=\left\{\mathrm{CSH}_{1}, \mathrm{CSH}_{2}, \mathrm{CSH}_{3}\right\}$, where $\mathrm{CSH}_{1}$ can appear at the start of a stroke, and both $\mathrm{CSH}_{2}$ and $\mathrm{CSH}_{3}$ can appear in the middle of a stroke.
2. $u t s_{2}=t s_{3}$ has one interpretation $p_{31}=\left\{\mathrm{CSH}_{1}\right\}$ where $\mathrm{CSH}_{1}$ can appear in the middle of a stroke.
3. $u t s_{3}=t s_{7}$ has an interpretation $\mathrm{p}_{71}=\left\{\mathrm{CSH}_{9}\right\}$, where $\mathrm{CSH}_{9}$ can appear at the end of a stroke.

The interpretations $p_{11}, p_{31}$, and $p_{71}$ are retained as valid interpretations of $u t s_{1}$, uts $s_{2}$, and $u t_{3}$, respectively.
(c) The set of sets of interpretations is $S P=\left\{P_{1}, P_{2}, P_{3}\right\}$, where $P_{1}=\left\{p_{11}\right\}, P_{2}=\left\{p_{31}\right\}$,


Figure 13.4. An Enumeration and Requirement Tree (ERT) for Example 13.1.2.
and $P_{3}=\left\{p_{71}\right\}$.
Many combinations of token strings can be extracted from the stroke. For simplicity, it is assumed that there is only one set, cuts, which has an acceptance degree $\mathrm{a}=1.0>$ ( $\mathrm{THR}_{\boldsymbol{a}}=0.6$ ).

Step 2. Generating a CSH Enumeration and Requirement Tree (ERT) for SP
By applying this step to SP, an ERT is obtained, part of which is shown in Figure 13.4. Each node of the ERT has four tuples, except the root node. The first tuple is the character class which is abbreviated by one character. The abbreviations of character classes which are used in this example are shown in Table 13.1. The next tuple is the required secondary stroke combination. The symbols shown in Table 13.2 are used to denote these combinations. The third tuple is the required position of the secondary stroke combination relative to the CSH, where A and B refer to "above" and "below", respectively. The last tuple is the token string number to which the CSH belongs.

Table 13.1. Abbreviations of character classes used in Example 13.1.2.

| Character Class | Abbreviation |
| :--- | :---: |
| $B a$ | B |
| Ha | H |
| Hamza | Z |
| Meem | M |
| Sad | S |
| Ta | T |
| Ya | Y |

Table 13.2. Symbols of secondary stroke combinations.

| Secondary Stroke Combination | Symbol |
| :--- | :---: |
| Null | $\lambda$ |
| Single Dot | . |
| Two Isolated Dots | $\ldots$ |
| Horizontal Dash | . |
| Hamza | z |
| Damma | , |

### 13.2. CHARACTER FORMATION

The process of combining the secondary strokes, which are presented to a main stroke using Algorithm 12.3, with the CSH's of that main stroke to form characters is a sophisticated problem which we will try to solve in this section.

Figure 13.5, shows one Arabic word, the same word of Figure 13.2(a), which consists of one main stroke and five secondary strokes. Although not all the secondary strokes are positioned accurately with respect to their corresponding CSH's, an Arabic human reader can easily divide the main stroke into five CSH's, shown numbered in the figure, and correctly spell the word to the characters, from right to left, Ta, Meem, Ha, Ya, and Sad. In this spelling, the first CSH is assigned the two-dots secondary strokes to yield the Ta character, the second and the third CSH's do not need any secondary stroke


Figure 13.5. The word of Figure 13.2(a), enlarged; the numbers refer to CSH's.
yielding Meem and Ha characters, respectively, the forth CSH is assigned the horizontal dash (two connected dots) secondary stroke to yield Ya character, and the fifth CSH does not need any secondary stroke yielding Sad character, however, it is assigned the two Damma secondary strokes (similar to comma), which are used as diacritics. This assignment is based on the following rules, which work together:

1. Proximity Rule: A secondary stroke most probably belongs to the nearest CSH.
2. Position Rule: The same combination of secondary strokes can be located in different positions relative to a CSH yielding different characters. Therefore, the position of a secondary stroke combination plays an important rule in determining the exact intended character.
3. Semantics Rule: A set of secondary strokes, which lie in certain positions relative to a CSH , are assigned to that CSH such that they form a valid Arabic character.

The above three rules are applied by a human in parallel to read handwriting; there may exist other rules, also. From Figure 13.5, it is clear that we can not only depend on a single rule neglecting the other rules. If only Rule 1 is applied, then the third CSH is assigned the dash secondary stroke, since the later lies completely below the former, yielding an undefined Arabic character. This kind of problem can be resolved by applying Rule 3 in which case the horizontal dash is assigned to the fourth CSH yielding the
meaningful character, Ya. If Rule 2 is neglected, then the two dots and the first CSH will have two readings: $T a$ and $Y a$, an ambiguous situation.

In this section, an algorithm will be developed to automate the process of combining secondary strokes with CSH's to form characters. The above three rules will be used in this process.

### 13.2.1. Definitions

(a) A nondirected graph $G=(V, L)$ is a set of vertices $V=\left\{v_{1}, v_{2}, \ldots\right\}$, and a set of links $L=\left\{l_{1}, l_{2} \ldots\right\}$ having no orientation and joining all or some of these vertices.
(b) A nondirected graph $\mathrm{G}=(\mathrm{V}, \mathrm{L})$ is said to be bipartite, if the set V can be partitioned into two subsets $\mathrm{V}^{\mathbf{a}}$ and $\mathrm{V}^{+}$so that all links have one terminal vertex in $\mathrm{V}^{a}$ and the other in $\mathrm{V}^{\mathrm{b}}$.
(c) A bipartite graph $G=\left(V^{a} \cup V^{b}, L\right)$ is said to be complete if for every two vertices $v_{i} \in V^{\mathbf{r}}$ and $v_{j} \in V^{b}$ there exists a link in $G$. If $\left|V^{d}\right|$, the number of vertices in set $V^{d}$, is $r$ and $\left|V^{\dagger}\right|$ is $s$, then the complete nondirected bipartite graph $G=\left(V^{a} \cup V^{b}, L\right)$ is denoted by $\mathrm{K}_{\mathrm{t}}$.
(d) A perfect matching in G is a matching where every vertex is matched to some other vertex.

For example, Figure 13.6(a) shows a nondirected graph, G, with the set of vertices $V=\{1,2,3,4,5,6,7,8\}$ and 15 links. The set $V$ is partitioned into two subsets $V^{a}=\{1$, $2,3)$ and $V^{b}=\{4,5,6,7,8\}$. Every link in the graph has one terminal vertex in $V^{r}$ and the other in $V^{b}$. Hence, the graph is bipartite. Since for every two vertices $v_{i} \in V^{a}$ and $v_{j}$ $\in \mathrm{V}^{\mathrm{b}}$ there exists a link in G , the bipartite graph, G , is complete. In $\mathrm{G},\left|\mathrm{V}^{\mathrm{d}}\right|=3$ and $\left|\mathrm{V}^{\dagger}\right|$ $=5$; hence, G is denoted by $\mathrm{K}_{3}$. There is no perfect matching in G since it is not possible to match every vertex to some other vertex. A complete bipartite graph, $\mathrm{K}_{3,3}$, which has a perfect matching is shown in Figure 13.6(b).

### 13.2.2. Problem Formulation

In Step 2 of Algorithm 13.1, an ERT was generated for every $\mathrm{SP}_{\mathrm{i}}$ of a main stroke. Consider the set, $S S=\left\{S_{1},{S S_{2}}_{2} \ldots, S_{4}\right\}$, where $S_{i}$ is the set of secondary strokes which a secondary stroke candidate, $G_{i}$, may represent, and $s$ is the number of secondary


Figure 13.6. Graph definitions: (a) a complete bipartite graph, $\mathrm{K}_{3,5}$, and (b) a complete bipartite graph, $\mathrm{K}_{3,3}$, which has a perfect matching.
stroke candidates presented to a main stroke. Secondary stroke candidates are found using Algorithm 12.2 and are presented to main strokes using Algorithm 12.3. The set SSV is the set of all possible combinations of vectors $\operatorname{ssv}=\left(\mathrm{ss}_{1}, \mathrm{ss}_{2}, \ldots, \mathrm{ss}_{\mathrm{s}}\right)$, such that $\mathrm{ss}_{\mathrm{i}} \in \mathrm{SS}_{\mathrm{i}}$. For example, if $\mathrm{s}=3$, and $\mathrm{SS}_{1}=\{D o t\}, \mathrm{SS}_{2}=\{$ Two connected dots, Fatha $\}, \mathrm{SS}_{3}=$ \{Damma\}, then the set SSV contains the vectors (Dot, Two connected dots, Damma) and (Dot, Fatha, Damma). For every path from a leaf node of the ERT to the node just before the root node:
(a) Let current_node point to that leaf node. Initialize the index variables $\mathrm{k}=0$, and $l$ $=0$.
(b) While current_node $\neq$ root node of ERT do \{

## Increment $l$.

Let $n$ be equal to the number of secondary strokes which exist in the secondary stroke combination which is required to form the character, char, that is stored in current_node.
Let the n sets of coordinates $\left(\mathrm{xmin}_{\mathrm{k}+1}, \mathrm{xmax}_{\mathrm{k}+1}, \mathrm{ymin}_{\mathrm{k}+1}, \mathrm{ymax}_{\mathrm{k}+1}\right),\left(\mathrm{xmin} \mathrm{m}_{\mathrm{k}+2}\right.$, $\left.\operatorname{xmax}_{k+2}, \mathrm{ymin}_{\mathrm{k}+2}, \mathrm{ymax}_{\mathrm{k}+2}\right), \ldots$, be equal to the boundary coordinates of the rectangle that encloses the vertices of the token string whose index, j , in stored
in current_node.
Let char ${ }_{k+1}$, char $_{k+2}, \ldots$, char $r_{k+a}$ be equal to character class, char, which is stored in current_node.
Let sec_type ${ }_{k+1}$, sec_type $\mathrm{e}_{\mathrm{k}+2}, \ldots$, sec_type $\mathrm{e}_{\mathrm{k}+\mathrm{a}}$ be equal to the classes of the secondary strokes which comprise the required secondary stroke combination. For example, if the secondary stroke combination consists of the two secondary strokes: Dot and Two connected dots, i.e., $\mathrm{n}=2$, then we have sec_type $\mathrm{g}_{\mathrm{k}+1}=$ Dot and sec_type $\mathrm{k}_{\mathrm{k}+2}=T$ wo connected dots.
Let pos $s_{k+1}$, pos $_{k+2} \ldots$. . pos $s_{k+1}$ be equal to the position of the secondary stroke combination which is required by char and is stored in current_node.
Let index ${ }_{k+1}$, index $x_{t+2}, \ldots$, index ${ }_{k+a}$ be equal to $l$.
Increment k by n .
Move current_node to point to the father of current_node.
\}
(c) Let char ${ }_{k+1}$, char $_{k+2}, \ldots$, char $_{k+1}$ be equal to the NULL character, $\Lambda$. The corresponding CSH of a NULL character is called a NULL CSH.
(d) Every $s s v=\left(s s_{1}, s s_{2}, \ldots, s s_{1}\right) \in S S V$ is extended to obtain another vector ssve $=$ ( $\mathrm{ss}_{1}, \mathrm{ss}_{2} \ldots, \mathrm{ss}_{p} \mathrm{ss}_{+1+1} \ldots$, , $s_{p+1}$ ), where $\mathrm{ss}_{\mathrm{t}}$ i is equal to the NULL secondary stroke, $i=1,2, \ldots, k$. The set of extended vectors is SSVE.
Let char $r_{1}$, char $r_{2}, \ldots$, char $r_{k+1}$ be represented by the set of vertices $V^{d}=\left\{v_{1}{ }^{2}, v_{2}^{2}, \ldots\right.$ ., $\left.v_{k+1}^{d}\right\}$, where $v_{i}^{d}$ corresponds to char $r_{i}$ Also, let the elements of the vector ssve $=\left(s_{1}, s_{2}\right.$, $\left.\ldots, s_{s}, s_{s+1}, \ldots, s s_{t+1}\right)$ be represented by the set of vertices $\mathrm{V}^{b}=\left\{\mathrm{v}_{1}^{\mathrm{b}}, \mathrm{v}_{2}^{\mathrm{b}}, \ldots, \mathrm{v}_{\mathrm{b}+\mathrm{k}}\right\}$, where $v_{i}^{b}$ corresponds to ss. Consider the complete graph $K_{t+, h+t}=G=\left(V^{a} \cup V^{b}, L\right)$, where $V^{v}$ and $V^{d}$ are independent sets of vertices and links $l_{i j}=\left(v_{i}^{2}, v_{j}^{b}\right) \in L$ have $v_{i}^{2} \in V^{d}$, $v_{j}^{b} \in \mathrm{~V}^{\mathrm{b}}$ and cost $\mathrm{c}_{\mathrm{i}}$.

The new idea of finding the best way of combining secondary strokes with CSH's to form characters depends on finding a perfect matching of $G$ with minimum cost. This is well known in the literature as the Assignment Problem (AP) and is often discussed in connection with complete bipartite graphs. The algorithm of [74], referred as the Hungarian Method, will be used to solve this assignment problem.

The details of the Hungarian Method are not important for the development of a
solution to our problem. Interested readers may refer to [74]. There are two key points in using the Hungarian Method to solve our AP:
(a) The first point lies in finding the cost matrix C which is an $(\mathrm{k}+\mathrm{s}) \times(\mathrm{k}+\mathrm{s})$ matrix whose rows correspond to the vertices of $V^{2}$ and whose columns correspond to the vertices of $\mathrm{V}^{\downarrow}$. The initial entries, $\mathrm{c}_{\mathrm{i} j}$, are the costs of links $\left(\mathrm{v}_{\mathrm{i}}, \mathrm{v}_{\mathrm{j}}\right), \mathrm{v}_{\mathrm{i}} \in \mathrm{V}^{\mathrm{d}}, \mathrm{v}_{\mathrm{j}} \in \mathrm{V}^{\mathrm{b}}$, which are found as follows:

1. If $\mathrm{v}_{\mathrm{i}}$ represents a NULL character, then $\mathrm{c}_{\mathrm{j}}$ equals a large number, $\mathrm{N}_{\text {Lage }}$, which is selected to guarantee that
i. the secondary strokes are not easily assigned to NULL characters to give other characters the chance to hook suitable secondary strokes, and
ii. a NULL secondary stroke is not easily assigned to a NULL character. This assures that there will be an adequate number of NULL secondary strokes to be assigned to CSH's that do not require secondary strokes, i.e., require NULL secondary strokes.
2. Else, if sec_type ${ }_{i}$ does not represent a NULL secondary stroke and $\mathbf{v}_{\mathbf{j}}$ represents a NULL secondary stroke, then $\mathrm{c}_{\mathrm{ij}}$ equals $\mathrm{N}_{\text {Laro }}$. In this way, the CSH's which require secondary strokes, which can not be properly provided, are assigned NULL secondary strokes, i.e., they are left without gaining the required secondary strokes. The cost of this unavailability of a proper secondary stroke is to let $c_{i j}=N_{\text {Large }}$.
3. Else, if sec_type is not of the same type as the secondary stroke which is represented by $\mathrm{v}_{\mathrm{j}}$, then $\mathrm{c}_{\mathrm{j}}$ is set to infinity. This is clear since a CSH can not be assigned a secondary stroke which is not required by that CSH.
4. Else, if both sec_type, and the secondary stroke which is represented by $\mathrm{v}_{\mathrm{j}}$ are NULL secondary strokes, then $\mathrm{c}_{\mathrm{i}}$ is set to zero. This allows characters which do not require secondary strokes to be formed without any cost:
5. Else, depending on the position, pos ${ }_{j}$ of the required secondary stroke, the type, sec_type ${ }_{\mathrm{i}}$ of the required secondary stroke, the coordinates (xmin ${ }_{i}$, xmax $_{\mathrm{i}}, \mathrm{ymin}_{\mathrm{i}}$, max $_{\mathrm{i}}$ ), and the $\mathrm{x} \& \mathrm{y}$ coordinates of the mean point of the secondary stroke which is represented by $v_{j}, c_{j}$ equals either some finite value « $N_{\text {Lagce }}$, which is a function of the earlier coordinates, or equals infinity if $v_{j}$
does not lie in the required position, pos ${ }_{i}$.
(b) Second, the elements of the cost matrix which are added to find the cost of an assignment are:
6. The elements which correspond to non-NULL secondary strokes that are assigned to non-NULL CSH's,
7. The elements which correspond to NULL secondary strokes that are assigned to non-NULL CSH's, i.e., the elements which correspond to non-NULL CSH's requiring some secondary strokes but could not hook them,
8. The elements which correspond to non-NULL secondary strokes that are assigned to NULL CSH's. This is necessary since an existing spurious secondary stroke must add to the cost of the minimum matching.
Now, a formal description of the algorithm to form characters follows.

Algorithm 13.2
Use: To combine CSH's with secondary strokes to form characters
Input: 1. Reduced graphs of strokes
2. Secondary stroke presentation
3. CSH enumeration and requirement trees

Output: 1. Words of characters
2. Redundant secondary strokes, i.e., strokes which were presented to main strokes using Algorithm 12.3 but could not share in interpreting main strokes

## Procedure:

Step 1. Solving of all possible assignment problems
For every ERT of a main stroke
For every leaf node in the ERT
(a) Prepare the set SSVE.
(b) For every ssve $\epsilon$ SSVE

1. formulate the corresponding AP problem.
2. Use the Hungarian Method to solve this assignment problem.

Step 2. Selecting the minimum cost solution
(a) Of all the assignment problems, generated and solved in Step 1, retain the variables, $V^{2}, V^{b}$, char $_{i}$, index $_{i}, i=1,2, \ldots, k$, and the cost matrix of the AP which yields the minimum cost. The recognized characters of the stroke are extracted from the char and index arrays as follows: Every set char ${ }_{i}$, char $_{i+1}, \ldots$, char $_{i+j}$, such that index ${ }_{i}=$ index $_{i+1}=\ldots$ index $_{i+j}$, corresponds to one recognized character, char ${ }_{i}$, in the main stroke. Since char ${ }_{1}$ corresponds to the left-most character of the main stroke, the recognized characters are reversed in order.
(b) Secondary strokes which are assigned to NULL characters in the minimum solution are redundant secondary strokes which could not contribute in interpreting main strokes. These strokes are freed again, i.e., they are drawn off from the main strokes to which they were presented. Such secondary strokes require special manipulation to obtain their proper interpretations, which is the subject of Section 13.3.

### 13.2.3. Example

In Example 12.2.2, five strokes of the word which is shown in Figure 13.2(a) were marked as secondary stroke candidates with their sets of secondary strokes being as follows: $\mathrm{SS}_{1}=\mathrm{SS}_{2}=\{$ Dot $\}, \mathrm{SS}_{3}=\mathrm{SS}_{4}=\{$ Damma $\}$, and $\mathrm{SS}_{5}=\{$ Two connected dots, Fatha, Kasra\}. The sixth unmarked stroke, whose graph is shown in Figure 13.2(b), is a main stroke. Algorithm 13.2 is used to combine secondary strokes with CSH's as follows:

Step 1. Solving of all possible assignment problems For simplicity, we consider only one ERT which was obtained in Example 13.1.2 and part of which is shown in Figure 13.4. Also, we will consider only the leaf node of the ERT which has a double-line rectangle:
(a) Prepare the set SSVE

It is assumed that the five secondary stroke candidates are presented to the main stroke using Algorithm 12.3. One possible combination of secondary stroke candidates is ssv = (Dot, Dot, Damma, Damma, Two connected dots $)$. Consider the path of the ERT tree, shown in Figure 13.4, whose nodes are double-line rectangles.

From the leaf node of the path, the following information is established:

1. The set of coordinates $\left(x \min _{1}, x_{\max }^{1}, y \min _{1}, y \max _{1}\right)$ is set equal to the boundary coordinates of token string uts $_{3}$.
2. char ${ }_{1}$ is set equal to character class, Sad.
3. sec_type ${ }_{1}$ is set equal to $\lambda$ since no secondary stroke is required.
4. $\operatorname{pos}_{1}$ is left blank since $S a d$ does not require any secondary stroke.
5. index $x_{1}$ is set equal to 1 .

Similarly, from the next node, we set the following variables: $\left(\mathrm{xmin}_{2}, \mathrm{xmax}, y \mathrm{ymin}_{2}\right.$, ymax $_{2}$ ) $=$ boundary coordinates of token string uts ${ }_{2}$, char $_{2}=Y a$, sec_type $_{2}=T w o$ connected dots, $\operatorname{pos}_{2}=$ Below, and index $x_{2}=2$. From the third node: $\left(\mathrm{xmin}_{3}, \mathrm{xmax}_{3}\right.$, $\left.\mathrm{ymin}_{3}, \mathrm{ymax}_{3}\right)=$ boundary coordinates of token string uts ${ }_{1}$, char $_{3}=H a$, sec_type $=$ $\lambda$, pos $_{3}=$ Blank, and index $3=3$. From the forth node: $\left(\mathrm{xmin}_{4}, \mathrm{xmax}_{4}, \mathrm{ymin}_{4}, \mathrm{ymax}_{4}\right)$ $=$ boundary coordinates of token string uts ${ }_{1}$, char $_{4}=$ Meem, sec_type $4=\lambda$, pos $_{4}=$ Blank, and index $x_{4}=4$. Finally, from the fifth node: $\left(x \min _{5}, x^{2} \max _{5}, y \min _{5}, y \max \right)=$ $\left(\mathrm{xmin}_{6}, \mathrm{xmax}_{6}, \mathrm{ymin}_{6}, \mathrm{ymax}_{6}\right)=$ boundary coordinates of token string uts ${ }_{1}$, char $_{5}=$ char $_{6}=T a$, sec_type $_{5}=$ sec_type $_{6}=D o t$, pos $_{5}=$ pos $_{6}=$ Above, and index ${ }_{5}=$ index $_{6}$ $=5$. In this example, the final value of $k$ and the value of $s$ are 6 and 5 , respectively. Now, set char ${ }_{7}=$ char $_{8}=\ldots=$ char $_{6+5}=\Lambda$; the NULL character. The vector ssv is extended to ssve $=($ Dot, Dot, Damma, Damma, Two connected dots, $\lambda, \lambda, \lambda, \lambda$, $\lambda, \lambda)$.
(b) For the vector ssv

1. Formulate the corresponding AP problem:

The set $\left\{\right.$ char $\left._{\mathrm{j}} \mathrm{i}=1,2, \ldots, 11\right\}$ is represented by the set of vertices $V^{a}=\left\{v_{i}^{2}\right.$, $i=1,2, \ldots, 11\}$. The elements of the vector ssve are represented by the set of vertices $\mathrm{V}^{b}=\left\{\mathrm{v}_{\mathrm{i}}^{b}, \mathrm{i}=1,2, \ldots, 11\right\}$. A complete graph, $\mathrm{K}_{11,11}=\mathrm{G}=\left(\mathrm{V}^{a} \cup\right.$ $\left.V^{b}, L\right)$, is formed, where links $t_{i j}=\left(v_{i}^{a}, v_{j}^{b}\right) \in L$ have $v_{i}^{a} \in V^{a}, v_{j}^{b} \in V^{b}$. The cost matrix is constructed as described in Section 13.2.2 and is shown in Figure 13.7. The rows and columns of the matrix correspond to the elements of the arrays char and ssve, respectively. Characters and secondary strokes are referred to by the abbreviations of Table 13.1 and symbols of Table 13.2, respectively.


Figure 13.7. Cost matrix of the assignment problem of Example 13.2.3.
2. Use the Hungarian Method to solve this assignment problem:

This assignment problem is solved using the Hungarian Method which yields the solution shown in Figure 13.7. The total cost of the optimal solution, the elements of which are surrounded with double squares, is

$$
\begin{align*}
C_{\text {opt }} & =c_{25}+c_{51}+c_{62}+c_{16}+c_{37}+c_{48}+c_{73}+c_{84} \\
& =a+b+c+0+0+0+Z+Z \\
& =a+b+c+2 \times Z \tag{13.3}
\end{align*}
$$

where $Z$ is a very large number compared to $a, b$, and $c$.
Step 2. Selecting the minimum cost solution
(a) For illustration purposes, it is assumed that the assignment problem which was solved in Step 1 is the one which yields the minimum cost. The recognized characters of the stroke are extracted from the char and index arrays as follows:

1. The first character is char $=S a d$, where index ${ }_{1}=1$.
2. $\quad$ The second character is $\operatorname{char}_{2}=Y a$, where index $=2$.
3. $\quad$ The third character is char $_{3}=H a$, where index ${ }_{3}=3$.
4. The forth character is $\mathrm{char}_{4}=$ Meem, where index ${ }_{4}=4$.
5. $\quad$ The last character is char $_{5}=$ char $_{6}=T a$, where index ${ }_{5}=$ index $_{6}=5$.

Since char ${ }_{1}$ is the left-most character, the extracted characters are reversed in order to obtain the characters: Ta, Meem, Ha, Ya, and Sad. A human Arabic reader can easily find that these are the true characters which comprise the Arabic word shown in Figure 13.2(a).
(b) The third and forth secondary strokes are redundant since they are assigned to NULL CSH's. Thus, they are drawn off the main stroke for further interpretation.

### 13.3. MANIPULATION OF REDUNDANT SECONDARY STROKES

In this section, redundant secondary strokes are classified and put in the proper location of the line in which they exist. Let $G_{a}$ be a redundant secondary stroke which is to be classified and SS is the set of possible interpretations of $\mathrm{G}_{\mathbf{a}}$. Let $l$ be the line number of the main stroke to which $G_{a}$ was presented.

### 13.3.1. Manipulation of Vertical Bar Secondary Stroke

If one of the interpretations of $G_{\Delta}$, in $S S$, is a vertical bar, then $G_{a}$ can be classified as many characters depending on the closest secondary stroke, $G_{b}$, to $G_{a}$ :
(a) If there is no other secondary stroke, $G_{b}$ close to $G_{a}$ then $G_{a}$ is classified as Alif and is added to the list of main strokes of line $l$.
(b) Else, if $G_{b}$ is redundant, then

1. If one of the interpretations of $G_{b}$ is Hamza, then: if $G_{b}$ is above $G_{a}$ then a Hamza_above_Alif character is added to the list of main strokes of line $l$. If $\mathrm{G}_{\mathrm{b}}$ lies below $\mathrm{G}_{\mathrm{a}}$ then a Hamza_below_Alif character is added to the list of main strokes of line $l$. Otherwise, two new main strokes representing the characters Alif and Hamza, respectively, are added to the list of main strokes of line $l . G_{b}$ is removed from the set of redundant strokes.
2. Else, if one of the interpretations of $G_{b}$ is $M a d d a$, then: If $G_{b}$ lies above $G_{a}$ then a Madda above_Alif character is added to the list of main strokes of line
l. Otherwise, two new main strokes representing the characters Alif and an isolated Noon (Madda and isolated Noon have the same shape) are added to the list of main strokes of line $l . \mathrm{G}_{\mathrm{b}}$ is removed from the set of redundant strokes.
3. If neither of the interpretations of $\mathrm{G}_{\mathrm{b}}$ is Hamza nor it is Madda, then an Alif character is added to the list of main strokes of line $l$.
(c) If $\mathrm{G}_{\mathrm{b}}$ is assigned, i.e., it is not redundant, then an Alif character is added to the list of main strokes of line $l$.

In all the three cases above, $(\mathrm{a}, \mathrm{b}, \mathrm{c}), \mathrm{G}_{\mathrm{a}}$ is removed from the set of redundant strokes.

### 13.3.2. Extraction of Isolated Ra and Isolated Zain Characters

$R a$ and the main stroke of Zain (a Dot above $R a$ ) may be initially classified as secondary strokes due to the similarities between them and some secondary strokes like slash and Fatha. Thus, a redundant secondary stroke, $G_{y}$, whose one of its interpretations char $\in$ CHAR is Ra, where the set CHAR is determined by Algorithm 12.2, is classified as follows:
(a) If $\mathrm{G}_{\mathrm{a}}$ has no close secondary stroke neighbours or the closest secondary stroke neighbour is assigned, then a Ra character is added to the list of main strokes of line $l$.
(b) If $G_{a}$ has a closest redundant secondary stroke neighbour, $G_{b}$, then

1. If $G_{b}$ is $a$ Dot then: if $G_{b}$ is above $G_{a}$ then a Zain character is added to the list of main strokes of line $l$, else, two new main strokes representing the characters $R a$ and isolated Dot are added to the list of main strokes of line $l$. $\mathrm{G}_{\mathrm{b}}$ is removed from the set of redundant strokes.
2. If $\mathrm{G}_{\mathrm{b}}$ is not a Dot, then a $R a$ character is added to the list of main strokes of line $l$.

In the two cases above, $(\mathrm{a}, \mathrm{b}), \mathrm{G}_{\mathbf{a}}$ is removed from the set of redundant strokes.

### 13.3.3. Extraction of Miscellaneous Characters

A remaining redundant secondary stroke, $G_{\mathbf{2}}$, is classified as follows. A character is added to the list of main strokes of line $l_{2}$ if there is one interpretation, $p$, of $G_{a}$ such that p is
(a) a Dot, in which case a Dot character is added,
(b) Two Connected Dots, in which case a minus sign character is added,
(c) Three Connected Dots, in which case Arabic numeral " $\wedge$ " is added,
(d) a Madda, in which case an isolated Noon character is added,
(e) a Hamza, in which case a Hamza character is added,
(f) an inclined slash, in which case a division sign character is added,
(g) a minus sign, in which case a minus sign character is added, or
(h) a Damma, where there are three cases:

1. If the Damma lies above the stroke to which it was presented, then it is true Damma in which case it is neglected since it is only used to change a character accent.
2. If the Damma lies to the right or left of the stroke, then a Waw character is added.
3. If the Damma lies below the stroke, then it is neglected since, most probably, it belongs to another stroke in a following line.
In the cases (a-h), above, $\mathrm{G}_{\mathbf{a}}$ is removed from the set of redundant strokes. Any remaining redundant secondary stroke is assumed not to carry important information; thus it is neglected and no output is produced to represent that stroke. However, these algorithms can be further developed to deal with such redundant secondary strokes.

After the above miscellaneous secondary strokes are added to their proper lines, the main strokes of each line are arranged on a right-to-left flow basis such that the right-most main stroke appears first.

In Example 13.2.3, the third and forth secondary strokes were two Damma's which were redundant. Since these secondary strokes lie above the main stroke, they are interpreted as true Damma's.

## SUMMARY

This chapter addressed the last component of the recognition process of the CASR system, where strokes are interpreted as sets of characters. This component consists of three stages. First, all possible CSH interpretations of main strokes are enumerated and represented in tree data structures, called Enumeration and Requirement Trees (ERT's). Second, ERT's are combined with secondary stroke candidates to form characters by solving assignment problems. Finally, redundant secondary strokes, which are filtered by the second stage, are manipulated to form some other characters which are inserted in their proper places within lines. The final result is a list of ordered lines where each line is an ordered list of words.

## Experimentation



## OVERVIEW

In this chapter, experimental results of the CASR system are reported. A description of how the data of the learning and testing stages were acquired is given. The learning stage is described. For the testing stage, the performance of the system in terms of recognition, rejection, and error rates, and speed, is reported. Causes of rejection and error are analyzed.

### 14.1. DATA ACQUISITION

The data sets which were used in the learning and testing stages of the CASR system were not restricted to a limited list of words, i.e., an unlimited vocabulary was used. There was no restriction on the content, i.e., a subject can pick any book, story, journal, etc., and select the parts he wishes to write. Also, there was no restriction on pen type, ink type, or ink colour. In both the learning and testing stages, the subjects were asked to fill an A4 size blank sheet of undiacriticized handwriting. Subjects were asked to use a common type of chirography of Arabic handwriting which is called Arreka chirography. Subjects who don't master Arreka chirography were asked to simply follow the rule: write every subword as single piece without lifting the pen except for secondary strokes (dots, dashes, etc.). They were asked to avoid generating blobs as possible as they can since the CASR system was not designed to deal with such a phenomenon. Unfortunately, most of the subjects were not conforming to these instructions. Mostly, a mixture of Arreka
chirography and another common chirography in Arabic handwriting called Annaskh were used. In Annaskh chirography, the pen can be lifted more than once to write the main stroke of a subword. The whole data provided by the subjects of the learning and testing stages were used without discarding any proportion whether the subject follows Arreka chirography or not. More details about how data were acquired for the learning and testing stages is given below:
(a) Learning: Here, there was no restriction neither on the number of lines per page nor on the number of words per line. Thirteen unnormalized handwritten A4 size pages written by 13 subjects, one page per subject, were collected and used in the learning stage. Reproductions of the images of the learning stage are shown in Appendix C, Figures C. 1 to C. 13 .
(b) Testing: Another set of subjects other than the subjects of learning stage was used in the testing stage. It was noticed that in the data set of the learning stage some subjects may write very nearby lines which may result in line overlap. This raises a problem in the line segmentation algorithm, Algorithm 12.3, which cannot deal with such a phenomenon. Thus, in the testing stage the subjects were asked to write from 10 to 15 lines per page which span the entire length of the A4 size sheet. Twenty unnormalized handwritten A4 size pages written by 20 subjects were collected and used in the testing stage. Reproductions of the images used in the testing stage are shown in Appendix D, Figures D. 1 to D. 20.
Images of the data sets of the learning and testing stages were captured using an HP ScanJet scanner. The resolution used was 300 dots per inch in both the horizontal and vertical directions.. The reason for selecting this value of resolution is based on our observation than under-sampled pictures, e.g., less than 300 dpi., may create disconnected images for very thin strokes which produces multi-component straight line approximations for such strokes. This is not accepted by the CASR system since it does not have the capability to handle disconnected strokes.

### 14.2. LEARNING

Samples of cursive handwriting of 13 subjects were used in the learning stage, see Appendix C, Figures C. 1 to C.13. In the algorithms of token learning, the core and left
and right bandwidths of the fuzzy direction state entrance qualifiers were set to $22.5^{\circ}$. A total of 11,083 tokens were segmented and learned with the thresholds $\mathrm{THR}_{1}$ and $\mathrm{THR}_{2}$ both being set to 0.90 . A token fuzzy sequential machine was obtained which is capable of recognizing the segmented tokens. $\mathbf{2 , 9 2 2}$ logical token strings were learned with the thresholds THR, and THR $_{x y}$ both being set to 0.90 . Eighty two CSH's shared in formulating token strings' interpretations.

### 14.3. TESTING

Samples of cursive handwriting of 20 subjects were used in the testing stage, see Appendix D, Figures D. 1 to D.20. The testing stage was performed on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM. Here, we define the following measures to evaluate the performance of the CASR system:
(a) Subword based measures: where four parameters are defined:

1. Subword recognition rate: It is the percentage of subwords which were fully recognized,
2. Subword rejection rate: It is the percentage of subwords which were rejected,
3. Subword error rate: It is the percentage of subwords which have at least one erroneous character, and
4. Subword reliability rate: which equals subword recognition rate / ( $100-$ subword rejection rate) $\times 100$.
(b) Character based measures: where four parameters are defined:
5. Character recognition rate: It is the percentage of characters which were correctly recognized,
6. Character rejection rate: It is the percentage of characters which were rejected due to rejection of their corresponding subwords,
7. Character error rate: It is the percentage of characters which were substituted by some other erroneous characters, and
8. Character reliability rate: which equals character recognition rate / (100character rejection rate) $\times 100$.

The subword and character reliability rates are important factors in evaluating the
performance of the system. The higher these factors are the more reliable is the system. A system which rejects suspectable characters is more reliable than a system which, anyhow, assigns labels to such characters. Thus, a higher reliability means a lower error rate.

Based on the above measures, the overall system performance is as follows: subword recognition, rejection, error, reliability $=55.4 \%, 17.6 \%, 27.0 \%, 67.2 \%$, respectively, and character recognition, rejection, error, reliability $=51.1 \%, 29.3 \%$, $19.6 \%, 72.3 \%$, respectively. Table 14.1 shows detailed performance rates for the 20 subjects. From this table, it is clear that these rates reflect the level of quality of handwriting for each subject and the degree to which a subject was conforming to the mentioned instructions. For example, subjects no. 1, 11, 17, and 19 provided neat handwriting and almost followed the given instructions which results in relatively high subword and character recognition rates.

It is noticed, in Table 14.1, that the subword recognition rate is higher than the character recognition rate. This may seem surprising since it is not normally the case is text recognition systems as each character error also causes an error in the word in which it occurs. However; we try to remove this surprise by explaining how the figures of Table 14.1 were obtained. Detailed analysis data about one typical line consisting of 21 subwords and taken from a page used in the testing stage, is shown Table 14.2. Tables 14.1 and 14.2 are obtained according to the previously-defined performance measures. From Table 14.2 we notice the following:

1. Rejection and error usually occur in subwords which consist of more than one character.
2. A rejected subword, consisting of n characters, adds only one to the total number of rejected subwords. However, it adds n characters to the total number of rejected characters.

The above factors interact to yield the figures of Table 14.1.
Table 14.3 displays the time requirement of the system for the 20 subjects. Preprocessing time includes smoothing, stroke extraction, thinning, straight line approximation, enforcement of temporal information, and stroke segmentation. Recognition time includes token recognition, separating main and secondary strokes,

Table 14.1. Performance of the CASR system.
These results were obtained by running the algorithms on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM. Sub: subject \#, NL: number of lines, NSW: number of subwords, NC: number of characters, Rec: recognition rate, Rej: rejection rate, Err: error rate, Rel: reliability.

| Sub | NL | NSW | NC | Subword |  |  |  | Character |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | $\begin{aligned} & \text { Rec } \\ & \% \end{aligned}$ | Rej $\%$ | $\begin{aligned} & \text { Err } \\ & \% \end{aligned}$ | Rel $\%$ | Rec $\%$ | $\begin{gathered} \operatorname{Rej} \\ \% \end{gathered}$ | Err | Rel $\%$ |
| 1 | 11 | 200 | 400 | 62.0 | 16.0 | 22.0 | 73.8 | 56.2 | 24.3 | 19.5 | 74.2 |
| 2 | 12 | 220 | 461 | 55.0 | 18.2 | 26.8 | 67.2 | 50.1 | 29.5 | 20.4 | 71.1 |
| 3 | 10 | 187 | 381 | 57.2 | 16.0 | 26.8 | 68.1 | 53.0 | 27.6 | 19.4 | 73.2 |
| 4 | 10 | 231 | 434 | 57.1 | 18.6 | 24.3 | 70.1 | 53.5 | 27.9 | 18.6 | 74.2 |
| 5 | 10 | 195 | 420 | 52.3 | 17.9 | 29.8 | 63.7 | 47.6 | 31.9 | 20.5 | 69.9 |
| 6 | 10 | 203 | 420 | 53.2 | 17.7 | 29.1 | 64.6 | 50.0 | 31.0 | 19.0 | 72.5 |
| 7 | 13 | 264 | 531 | 54.9 | 16.7 | 28.4 | 65.9 | 50.8 | 29.6 | 19.6 | 72.2 |
| 8 | 12 | 166 | 343 | 56.0 | 15.7 | 28.3 | 66.4 | 53.1 | 27.7 | 19.2 | 73.4 |
| 9 | 11 | 228 | 454 | 52.2 | 19.7 | 28.1 | 65.0 | 48.0 | 32.2 | 19.8 | 70.8 |
| 10 | 11 | 221 | 447 | 54.3 | 19.0 | 26.7 | 67.0 | 51.0 | 28.6 | 20.4 | 71.4 |
| 11 | 11 | 210 | 449 | 65.2 | 11.9 | 22.9 | 74.0 | 61.0 | 22.3 | 16.7 | 78.5 |
| 12 | 12 | 213 | 457 | 50.7 | 19.7 | 29.6 | 63.1 | 46.8 | 32.0 | 21.2 | 68.8 |
| 13 | 12 | 217 | 450 | 49.3 | 21.2 | 29.5 | 62.6 | 44.9 | 32.0 | 23.1 | 66.0 |
| 14 | 13 | 203 | 411 | 50.7 | 18.7 | 30.6 | 62.4 | 45.0 | 33.8 | 21.2 | 68.0 |
| 15 | 11 | 196 | 419 | 53.1 | 19.9 | 27.0 | 66.3 | 50.1 | 30.3 | 19.6 | 71.9 |
| 16 | 12 | 222 | 465 | 52.3 | 19.8 | 27.9 | 65.2 | 47.7 | 31.0 | 21.3 | 69.1 |
| 17 | 12 | 239 | 494 | 64.8 | 13.0 | 22.2 | 74.5 | 60.1 | 23.1 | 16.8 | 78.2 |
| 18 | 12 | 238 | 522 | 51.3 | 18.5 | 30.2 | 62.9 | 46.9 | 32.8 | 20.3 | 69.8 |
| 19 | 13 | 253 | 503 | 67.2 | 13.0 | 19.8 | 77.2 | 63.8 | 23.5 | 12.7 | 83.4 |
| 20 | 14 | 267 | 559 | 48.0 | 21.3 | 30.7 | 61.0 | 44.0 | 33.6 | 22.4 | 66.3 |
| Total | 232 | 4373 | 9020 | 55.4 | 17.6 | 27.0 | 67.2 | 51.1 | 29.3 | 19.6 | 72.3 |

Table 14.2. Detailed performance data of a typical line of Arabic text.

| Subword No. | No. of characters | Subword |  |  | Characters |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | fully recognized | rejected | with error | recognized | rejected | erroneous |
| 1 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 2 | 3 | $\checkmark$ |  |  | 3 | 0 | 0 |
| 3 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 4 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 5 | 2 |  | $\checkmark$ |  | 0 | 2 | 0 |
| 6 | 4 |  |  | $\checkmark$ | 2 | 0 | 2 |
| 7 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 8 | 3 |  |  | $\checkmark$ | 2 | 0 | 1 |
| 9 | 2 |  |  | $\checkmark$ | 1 | 0 | 1 |
| 10 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 11 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 12 | 3 |  | $\checkmark$ |  | 0 | 3 | 0 |
| 13 | 3 | $\checkmark$ |  |  | 3 | 0 | 0 |
| 14 | 2 | $\checkmark$ |  |  | 2 | 0 | 0 |
| 15 | 2 |  | $\checkmark$ |  | 0 | 2 | 0 |
| 16 | 5 |  | $\checkmark$ |  | 0 | 5 | 0 |
| 17 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 18 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 19 | 3 |  |  | $\checkmark$ | 1 | 0 | 2 |
| 20 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| 21 | 1 | $\checkmark$ |  |  | 1 | 0 | 0 |
| Total | 42 | 13 | 4. | 4 | 24 | 12 | 6 |
| Percentage, \% |  | 62 | 19 | 19 | 57.1 | 28.6 | 14.3 |
| Reliability, \% |  | 76.5 |  |  | 80 |  |  |

Table 14.3. Speed of the CASR system.
These results were obtained by running the algorithms on a 486DX IBM PC compatible microcomputer, with 50 MHz clock and 16 MB RAM. Prep: preprocessing time, Recog: recognition time, A: average time.

| $*$ <br> Subject | Time (min.) /page |  |  | Time (min.)/subword |  |  | Time (min.)/character |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Prep | Recog | Total | Prep | Recog | Total | Prep | Recog | Total |
| 1 | 11.25 | 335.33 | 346.58 | 0.05 | 1.68 | 1.73 | 0.03 | 0.84 | 0.87 |
| 2 | 11.50 | 556.48 | 567.98 | 0.05 | 2.53 | 2.58 | 0.02 | 1.21 | 1.23 |
| 3 | 9.34 | 522.60 | 531.94 | 0.05 | 2.79 | 2.84 | 0.02 | 1.37 | 1.39 |
| 4 | 11.56 | 715.10 | 726.66 | 0.05 | 3.10 | 3.15 | 0.03 | 1.64 | 1.67 |
| 5 | 11.71 | 905.80 | 917.51 | 0.06 | 4.65 | 4.71 | 0.03 | 2.15 | 2.18 |
| 6 | 12.78 | 886.11 | 898.89 | 0.06 | 4.37 | 4.43 | 0.03 | 2.11 | 2.14 |
| 7 | 14.84 | 925.64 | 940.48 | 0.05 | 3.51 | 3.56 | 0.03 | 1.74 | 1.77 |
| 8 | 10.16 | 809.08 | 819.24 | 0.06 | 4.87 | 4.93 | 0.03 | 2.36 | 2.39 |
| 9 | 13.88 | 871.96 | 885.84 | 0.06 | 3.82 | 3.88 | 0.03 | 1.92 | 1.95 |
| 10 | 13.46 | 685.52 | 698.98 | 0.06 | 3.10 | 3.16 | 0.03 | 1.53 | 1.56 |
| 11 | 12.69 | 624.80 | 637.49 | 0.06 | 2.98 | 3.04 | 0.03 | 1.39 | 1.42 |
| 12 | 10.75 | 607.92 | 618.67 | 0.05 | 2.85 | 2.90 | 0.02 | 1.33 | 1.35 |
| 13 | 12.99 | 633.81 | 646.80 | 0.06 | 2.92 | 2.98 | 0.03 | 1.41 | 1.44 |
| 14 | 12.38 | 638.45 | 650.83 | 0.06 | 3.15 | 3.21 | 0.03 | 1.55 | 1.58 |
| 15 | 11.96 | 673.24 | 685.20 | 0.06 | 3.43 | 3.49 | 0.03 | 1.61 | 1.64 |
| 16 | 13.02 | 967.62 | 980.64 | 0.06 | 4.36 | 4.42 | 0.03 | 2.08 | 2.11 |
| 17 | 14.53 | 915.27 | 929.80 | 0.06 | 3.83 | 3.89 | 0.03 | 1.85 | 1.88 |
| 18 | 14.20 | 648.36 | 662.56 | 0.06 | 2.72 | 2.78 | 0.03 | 1.24 | 1.27 |
| 19 | 15.06 | 924.45 | 939.51 | 0.06 | 3.65 | 3.71 | 0.03 | 1.84 | 1.87 |
| 20 | 13.45 | 763.12 | 776.57 | 0.05 | 2.86 | 2.91 | 0.02 | 1.37 | 1.39 |
| A | 12.58 | 730.53 | 743.11 | 0.06 | 3.34 | 3.40 | 0.03 | 1.62 | 1.65 |
|  |  |  |  |  |  |  |  |  |  |

extracting lines and ordering strokes, CSH interpretations of main strokes, character formation, and manipulating redundant secondary strokes. The average times required to preprocess a page, subword, and character are $12.58 \mathrm{~min} ., 0.06 \mathrm{~min}$., and 0.03 min ., respectively. These times are very small compared to the average times required by the
recognition stage, which are 730.53 min . / page, 3.34 min . / subword, and 1.62 min . / character. In other words, the recognition algorithms occupied more than $98 \%$ of the overall page, subword, or character processing time from smoothing to final classification. The reason behind this is that the recognition algorithms are based on a combinatorial approach requiring long computations. However, the recognition time can be reduced by using faster machines.

### 14.4. REJECTION ANALYSIS

The high rejection rate was expected since a very high variability exists in handwriting, even in the handwriting of a single subject. The rejection rate can be reduced by learning more styles of handwriting. The reasons for rejection are:
(a) In $96.1 \%$ of the rejection cases, all the tokens of the stroke are recognized. However, the stroke contains an unknown token string which is a result of the following reasons:

1. An unknown combination of token identification codes is obtained.
2. The overall acceptance degree, $a_{y}$, of the features of token / string ratios is less than the specified threshold, THR ${ }_{8}$.
3. The overall acceptance degree, $a_{x y}$, of the features of intersection vertices is less than the specified threshold, $\mathrm{THR}_{\mathrm{xy}}$
(b) In $3.9 \%$ of the rejection cases, at least one token in the stroke is unknown. These strokes are rejected by the CASR system.

### 14.5. ERROR ANALYSIS

The main reasons for error cases are:
(a) A token string can have more than one valid interpretation of characters. This happens in $26.6 \%$ of the erroneously recognized characters. For example, the subword in Figure 14.1, has two interpretations. Although, a human reader can interpret it as Lam, Ta, and Ra characters, the CASR system may recognize it as a single Noon character considering the dash as a redundant secondary stroke. This kind of error is difficult to control in handwriting
(b) The tokens of a single character can be erroneously divided into strings representing

## ?

Figure 14.1. A subword which has two interpretations: Lam, Ta, and Ra, or Noon and redundant secondary stroke.


Figure 14.2. One shape of the main stroke of Seen and Sheen which consists of three tokens extending between points $1 \& 2,3 \& 4$, and $5 \& 6$, respectively.
more than one CSH which happens in $26.0 \%$ of the error cases. This error frequently happens with the Seen and Sheen characters, the main stroke of which consists of three tokens as shown in Figure 14.2. The CASR system may interpret each token, combined with suitable secondary strokes in its neighbourhood, as $B a$, Ta, Tha, Noon, or Ya. This is due to insufficient number of samples, introduced to the system in the learning stage with a main stroke as in Figure 14.2, of Seen and Sheen. Thus, more learning is needed. Also, any of the three tokens of the stroke of Figure 14.2 can be interpreted as Lam if its height starts to be larger than its width. This can be greatly reduced by introducing height / width relations between the individual tokens constituting a token string and the token string itself which is absent in the current version of the CASR system.
Sometimes, a shape, like that of Figure 14.3, which really consists of a single Pha character, can be interpreted as two characters: Pha and Alif if the left vertical termination, between points 1 and 2 , is relatively long. Although, this kind of error is very difficult to control in handwriting, it may be reduced by using context.
(c) The CSH's of some different characters look similar which comprises $20.9 \%$ of the


Figure 14.3. A main stroke of Pha character.



Figure 14.4. Examples of pairs of similar CSH's, the vertical pairs correspond to the CSH's of Ba \& Lam, Ba \& Ha, Ra \& Dal, Dal \& right parenthesis, Hamza \& Kaf, and Meem \& Ha, where the first CSH of each pair is the upper one.
error cases. Examples of pairs of similar CSH's are: Ba and Lam, Ba and Ha, Ra and Dal, Dal and right parenthesis, Hamza and Kaf, and Meem and Ha, see Figure 14.4. This kind of error can be reduced by using context.
(d) In $14.0 \%$ of error cases, there can be more than one valid assignment of secondary strokes to CSH's when the characters are formed. This is clear from the subword of Figure 14.5 which has two valid readings depending on the assignment of secondary strokes. The main stroke of this subword consists of two CSH's: the part between points 1 and 2 and the part between points 2 and 3 constitute the first and the second CSH's, respectively. If the two dots are assigned to the first CSH, then the subword consists of $T a$ and $R a$. If the right-most dot is assigned to the first CSH and the other dot is assigned to the second CSH, then the subword consists of Noon and Zain. An error of this kind can be alleviated by using context which selects the interpretation that gives a meaning. However, sometimes more than one interpretation may have a meaning which shows that this is a hard problem in automatic recognition systems.
(e) Inaccurate placement of secondary strokes yields wrong characters in $4.8 \%$ of the error cases. Figure 14.6 shows one word consisting of one main stroke and four


Figure 14.5. A subword the main stroke of which consists of two CSH's between points $1 \& 2$ and $2 \& 3$, respectively.


Figure 14.6. A word the main stroke of which consists of three CSH's which extend between points $1 \& 2,2 \& 3$, and $3 \& 4$, respectively.
secondary strokes; Dots. Although the dots are inaccurately positioned, an Arabic human reader can easily read the word as consisting of the characters: Noon, Qaf, and Pha. However, the CASR system reads this word as: Ta, Pha, and Phà. This is the result of assigning the right-most two dots to the right-most CSH yielding Ta, the next dot to the second CSH yielding Pha, and the left-most dot to the left-most CSH yielding Pha. Inaccurate positioning of secondary strokes relative to CSH's happens frequently in handwriting and is difficult to control. One means to partially solve this problem is to use context.
(f) In $6.2 \%$ of the error cases, characters with lost loops due to blotting were misrecognized which is the result of loss of temporal information, see Section 3.4.5. Unfortunately, the CASR system was not designed to deal with such a phenomenon, i.e., lost loops due to blotting. However, this kind of error may be reduced by working directly on the bitmap image of the stroke without being thinned.
(g) A very short secondary stroke representing two connected dots can be reduced to

Figure 14.7 A character with a short dash representing two dots. The dash may be reduced to a single dot by the straight line approximation algorithm.
a single dot by the straight line approximation algorithm, Algorithm 2.2. This results in a different character. For example, the character shown in Figure 14.7, which is $T a$, is interpreted as Noon if the short dash is reduced to a single dot. This happens in $1.5 \%$ of the error cases which we consider a minor cause of error that may be reduced by controlling the threshold of the length of a single dot and using context.

It is clear that context should play a significant role in reducing the error rate. The automation of such use of context is another point of research that deserves study, but it is out of scope of this research.

## SUMMARY

The system was trained on cursive handwritings of 13 subjects comprising 11,083 tokens and 2,922 logical token strings. The handwriting of 20 subjects other than the subjects of the learning stage were used in the testing stage. Subword and character recognition rates were $55.4 \%$ and $51.1 \%$, respectively. Although these rates are modest, the system performance can be enhanced by learning more styles, including more useful features, and using context.

The author has presented a new theoretical basis and concepts to design handwritten cursive script recognition systems which he considers as a broad foundation on which other similar studies can be established.

## Conclusions.



## OVERVIEW

In this chapter, the three parts of the work are summarized. Quantitative summary of the results is presented. Critical evaluation and concluding remarks are provided. An assessment of how the work has contributed to the field of off-line recognition of handwritten cursive text is presented. Finally, some points where further work can be done are highlighted.

### 15.1. PREPROCESSING

In Part One, novel algorithms were presented for processing of Arabic text prior to recognition. Algorithms were described to convert a thinned image of a stroke to a straight line approximation. These algorithms incorporate heuristics which ensure a unique centre for each intersection vertex, and to reduce the likelihood of spurious tails. Algorithm 2.2 can identify spurious bifurcation points which are unavoidable when thinning algorithms are used, remove them, and recover the original ones. Unlike existing approaches to the same problem, this new method can deal with complex junctions where more than two lines cross, and does not resort to geometrical properties which are prone to distortion by scanning and quantization noise. The obtained straight line approximations preserve the structural information of the original pattern ensuring a natural representation of it .

Novel heuristic algorithms and novel theorems, Theorems 1 and 2, were presented
to restrict and determine start and end vertices of an off-line image of a stroke. A straight line approximation of an off-line stroke is converted to a one-dimensional representation by a novel algorithm which aims to recover the original sequence of writing. Options were used to generate three different methods to enforce temporal information. The algorithm was tested against one data set of isolated handwritten characters and another data set of cursive handwriting, each provided by 20 subjects, see Appendices B and D. The results of Method 3, in which the determination of the start and end vertices is based on Theorems 1 and 2, were superior. This method has been $91.9 \%$ and $91.8 \%$ successful for the se two data sets, respectively. The resulting ordering of the stroke segments was a suit.2ble preprocessed representation for subsequent handwriting recognition algorithms as it helped to segment the stroke.

The earlier algorithms can deal with many of the situations that may arise in Arabic handwriting, however, they can be developed more so that almost all possible conditions are dealt with.

## Critique

In the algorithms of Part One, thinning was a necessary preprocessing step. However, there were cases in which these algorithms failed due to spurious artifacts introduced to the straight line approximations by the thinning process. These artifacts do not correspond to true segments in the original image and result in complex representations of strokes. Another drawback of thinning can be identified by the stroke of Figure 15.1. Figure 15.1(a) shows one handwritten Arabic subword which contains three loops; all appearing as blobs. This image was thinned to obtain that of Figure 15.1(b). For a human, these blobs can be easily recognized as loops from the context. However, they are difficult to deal with and cause problems in automatic recognition systems which accept thinned images. For example, a recognition system that accepts thinned images encounters difficulty in recognizing the thinned image of Figure 15.1(b). Thus, thinning results in the loss of very informative pieces of the image. For these reasons, we believe that researchers in the field of automatic text recognition must be aware of the consequences of drawbacks of the thinning process.

Also, in Algorithm 3.4, temporal information was lost when thinning a small loop that has become a blob due to blotting. As mentioned in Section 3.4.5, one trial was


Figure 15.1. (a) A binary image of a handwritten Arabic stroke with three loops appearing as blobs, and (b) its skeleton.
performed by the author to restore such lost loops. However, the generation of new spurious loops was the main problem which could not be eliminated. Spurious loops may degrade the performance of a recognition system. Thus, it is concluded that there should be no preprocessing stage to recover lost loops due to blotting.

### 15.2. IACR SYSTEM

In Part Two, an entirely novel fuzzy set-sequential machine character recognition system was presented. A stroke representation was developed which proved to be useful for the development of subsequent algorithms. Fuzzy sequential machines were defined to work as recognizers of handwritten strokes. An algorithm to obtain a deterministic fuzzy sequential machine from a stroke representation, that is capable of recognizing that stroke and its variants, was presented. An algorithm was developed to merge two fuzzy machines into one machine. The learning algorithm was a combination of many described algorithms.

A set of 20 stroke classes was used in the learning and testing stages. The system was trained on 5890 unnormalized handwritten strokes written by five subjects, see Appendix A. The learning stage produced a fuzzy sequential machine of 2705 states and 8640 arcs. A total of 7963 unnormalized handwritten strokes, written by 20 subjects other than the subjects of the learning stage, see Appendix B, was used in the testing stage. The recognition, rejection, and error rates were $95.8 \%, 1.5 \%$, and $2.7 \%$, respectively. These results are encouraging and offer much potential within the field of automatic off-line character recognition. The system is highly flexible in dealing with shape and size variations.

## Critique

Although a $95.8 \%$ recognition rate could be achieved, the IACR system is still lagging other recent systems which use traditional techniques. Smith et al. [11], in 1994, showed that systems built on a simple statistical technique and a large training database can be automatically optimized to produce classification accuracies of $99 \%$ in the domain of handwritten digits. The error rate is cut by more than half for every tenfold increase in the size of the training set from 10 to 100,000 examples. Three distance metrics for the standard Nearest Neighbour classification system were investigated: a simple Hamming distance metric, a pixel distance metric, and a metric based on the extraction of penstroke features.

Thus we suggest that researchers might better spend their time to use such wellmatured approaches in isolated Arabic character recognition systems instead of spending a lot of time seeking for new methods the performance of which remains unknown until
they are tried. The use of methods such as statistical ones eliminates the need for regeneration of time information.

### 15.3. CASR SYSTEM

In Part Three, an entirely novel text recognition system, capable of recognizing offline handwritten Arabic cursive text having a high variability was presented. This system was an extension of the IACR system. Tokens were extracted from a one-dimensional representation of a stroke. Fuzzy sequential machines were defined to work as recognizers of tokens. It was shown how to obtain a deterministic fuzzy sequential machine from a token representation that is capable of recognizing that token and its variants. An algorithm for token learning was presented. The tokens of a stroke were re-combined to meaningful strings of tokens. Algorithms to recognize and learn token strings were described. The recognition stage used algorithms of the learning stage. The process of extracting the best set of basic shapes which represent the best set of token strings that constitute an unknown stroke was described. A method was developed to extract lines from pages of handwritten text, arrange main strokes of extracted lines in the same order as they were written, and present secondary strokes to main strokes. Presented secondary strokes are combined with basic shapes to obtain the final characters by formulating and solving assignment problems for this purpose. Some secondary strokes which remain unassigned are individually manipulated.

Samples of cursive handwriting of 13 subjects, see Appendix C, were used in the learning stage. A total of 11,083 tokens were extracted and learned. A fuzzy sequential machine was obtained which is capable of recognizing the extracted tokens. A total of 2,922 token strings were learned. By inspecting these token strings, 82 raw characters were obtained.

The system was tested against the handwritings of 20 subjects other than the subjects of learning stage, see Appendix D, yielding overall subword and character recognition rates of $55.4 \%$ and 51.1 , respectively. The system performance can be enhanced by learning more styles, including more useful features, and making use of the context. Although these rates of the CASR system are modest and do not take the system to the level of commercial practicability, the author could present new theoretical basis
and concepts to design handwritten cursive script recognition systems which he considers as a broad foundation on which other similar studies can be established. Some other general concluding remarks are listed below:
(a) The strong cursive nature of handwritten Arabic text lends itself better to a structural or hybrid approach. The high variability in handwriting makes it difficult to apply decision theoretic approaches because often real-life data do not hold assumptions of decision theoretic approaches. For these reasons, the author considers that his fuzzy set-sequential and graph theoretic approach is a big step in the right way in addressing this hard problem.
(b) The problem of off-line handwritten cursive Arabic script recognition remains an open problem for the researchers to devise solutions. It requires advanced segmentation techniques, involving the interaction of segmentation and recognition. For efficiency, it is desirable for the recognizer to be free of constraints on primitive number. The ultimate goal is to obtain recognition systems with performance comparable with that of human. The performance of a recognition system remains unknown until it is complete and tested against real data, which is also a good reason for researchers to develop and test new methods in this domain.
(c) Off-line recognition of handwritten Arabic text is a great challenge. It is not a direct implementation of the recognition techniques used for other handwriting systems. The reason is simply it has different characteristics. Some impeding characteristics are listed below:

1. Arabic script is cursive which makes it difficult to segment a subword directly into characters.
2. Some Arabic characters have the same shape; however, they are distinguished from each other by the addition of secondary strokes, e.g., dots, in different positions relative to the main stroke. Sometimes, the ambiguity of the position of these secondary strokes in handwriting brings out many different readings for one word.
3. An Arabic character can have different shapes depending on its position in the word (beginning, middle, end, or isolated). This increases the number of fundamental shapes to more than twice the number of characters which
complicates the recognition of Arabic text.
4. Arabic characters vary in size, particularly in width, even within the same writing style.
5. Certain Arabic characters may overlap with neighbouring ones. The degree of overlap varies according to the handwriting style. The overlap adds to the difficulty in segmenting characters.
(d) In general, off-line recognition of unconstrained handwriting is a very hard especially on whole language vocabulary, because the segmentation of words into letters or small basic units induces many possible combinations.

### 15.4. HOW THIS WORK CONTRIBUTED TO THE FIELD

The author presented new contributions to the field of off-line recognition of handwritten cursive text in the following components:
(a) Representation: New methods were developed to obtain the following representations:

1. Straight Line Approximation of Strokes: This is an intermediate representation used in both systems: the Isolated Arabic Character Recognition system and the Cursive Arabic Script Recognition system. In this representation, the likelihood of spurious tails is reduced and, spurious bifurcation points, which are unavoidable when thinning algorithms are used, are removed, and the actual bifurcation points are recovered. The obtained straight line approximations preserve the structural information of the original patterns. The suggested method does not resort to distortable geometrical properties.
2. Temporal Information: Recognizing the benefits of the availability of dynamic information of handwriting and due to drawbacks of some previous methods [67-69], the author presented novel theorems and algorithms to recover the original sequence of off-line Arabic handwriting. This was used to aid in segmenting strokes in both systems.
3. Stroke Segmentation: Here methods were introduced to accept a straight line approximation and temporal information of a stroke and produce
small basic units constituting the stroke as the final representation. These basic units are temporally ordered which is a new contribution by itself.
(b) Classification: As mentioned above, strokes were segmented into small basic units in both systems. In the CASR system, these basic units were combined into strings of basic units. These strings were hypothesized into characters to obtain the target word. Also, we adopted a novel method for line segmentation which tolerates large variations in handwriting. Our novel approach, in both the IACR and CASR systems, is constructed using concepts from sequential machine, fuzzy set, and graph theories.
The methods used by researchers to segment lines, words and characters were primarily developed for printed text in which a horizontal baseline usually exists. This enabled them to use simple horizontal and vertical projections or Hough transform methods in segmentation. This means that, in case of printed text, segmentation can be usually performed as an independent process before classification. The situation is different in handwritten cursive text due to the following reasons:
4. A horizontal baseline does not exist in unconstrained handwriting.
5. There is a change in the slant even in a single line of handwritten text.
6. Secondary strokes (e.g., dots and dashes) are not carefully plotted, in handwriting, with respect to main strokes (main bodies of characters or words).
The author's new contribution here is that he viewed line and character segmentation processes as part of the classification process and presented other general methods for initial segmentation. Initial segmentation is already mentioned in (a)3, above.
(c) Learning: The learning and classification methods were developed to form one couple. The important thing here is that, as you notice in the data flow diagram of Figure 1.2, the learning stage comes after the classification stage! Actually this arrangement is preferred to us because our philosophy lies in: "What is this 2 ? If you know then you earn, otherwise come to learn." This means that trying to recognize comes first. If the system fails then it is taught.

### 15.5. SUGGESTIONS FOR FURTHER WORK

The following are points where further work can be done:
(a) Use of Contextual Information: Factors like letter sequences, word dependencies, sentence structure, style and subject matter, as well as comprehension, knowledge, inference, guessing, prediction, and imagination all take place very naturally during the process of human reading. These processes take place extremely effectively and efficiently in the human brain because they are the results of many years of trial, learning, and correction [82].

Many investigations on the process of human reading and comprehension, and the effects of contextual information have been made by linguists and psychologists [83 - 85]. Since there are thousands of writing styles in the world, it may not be feasible to build a vision system which is capable of recognizing all of them by shapes alone. The best solution seems to be making machines more intelligent which is a major step towards successful artificial intelligence in the area of natural language understanding and processing [82]. In order to do so, the use of contextual information is indispensable. Indeed, text processing, combined with techniques dealing with automatic correction of deletion, and substitution, is a subject of great interest [82, 86-93]. The effective and efficient use of contextual information in handwriting recognition systems remains an open area that deserves further research.
(b) Page Layout Analysis: There are two analyses necessary for reading documents: one is character recognition (OCR), and the other is page layout analysis. Although text is an important part of a document, it is also essential to know where the text resides in a page. Page format establishes meaning to regions of text. For instance, when searching for a paper by author name, only the author block of the title page needs to be examined. Names in the body of the text and in the references have different connotations. Therefore, determining the location of text by page layout analysis is an essential complement to OCR. Hence, studies for handwritten documents similar to those of references [94, 95], which concern printed documents, are needed.
(c) Segmentation of Mixed Text / Graphics Images: A digitized image consisting of
a mixture of text and graphics should be segmented in order to represent more efficiently both areas of text and graphics. The segmentation must be independent of changes in text writing style and size, and of string orientation. Although some works have been published in this field [69, 80], more general algorithms are still needed.
(d) In spite of our support and encouragement for researchers especially in Arabic cursive handwritten script recognition to go on, we also present an advice, as we believe, for them. The advice is to concentrate on recognition of printed script instead of handwritten script. Reasons for this are:

1. Many of the problems which exist in handwriting are absent in the case of printed script. Some of these problems are:
i. Handwritten script has a very high variability even for the same writer.
ii. It is very difficult for a writer to stick to a single style of writing even in a single page. Thus, a handwritten page is often a mixture of more than one writing style.
iii. Direction of flow of writing often changes from line to line and even in the same line.
iv. Secondary strokes are rarely positioned in the exact position relative to main strokes. This leads to ambiguities in automatic recognition systems.
v. Usually blotting occurs in handwriting especially at instances when the pen touches the paper and in loops which is an additional noise source.
vi. It is difficult to maintain a constant pen pressure on the writing surface. This often leads to some fading (lightly written) or broken parts of strokes' images.
2. Most of Arabic knowledge is typeprinted but not handwritten, e.g. books, journals, periodicals. A method, e.g., an automatic text recognition system, is needed to transform this knowledge to an electronic format.
3. Even the problem of printed Arabic script still is not completely solved.
4. There is no known efficient method to solve the hard problem of segmenting a subword into characters. Thus, researchers are encouraged to double their efforts to solve this problem.
5. Vital data, such as n-gram statistical data, is missing to researchers in Arabic text recognition. Generating this data is a project by itself. Also, an Arabic document image database helps very much in building document analysis and recognition systems. It can also serve as a benchmark for evaluating the performance of recognition systems for printed Arabic text.

## References

[1]. M. Ghuwar and W. Skarbek, "Detection of baseline for Arabic text lines using Hough type transforms", 7th Portuguese Conference on Pattern Recognition, Aveiro, Portugal, pp. 531-533, March 1995.
[2] I.S.I. Abuhaiba, S.A. Mahmoud, and R.J. Green, "Recognition of Handwritten Cursive Arabic Characters", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 16, No. 6, pp. 664-672, 1994.
[3] I.S.I. Abuhaiba, Use of Fuzzy Set Theory in Pattern Recognition with Application to Arabic Characters, MPhil Thesis, University of Bradford, 1991.
[4] A. Nouh, A. Sultan, and R. Tolba, "An approach for Arabic characters recognition", J. Eng. Sci., Univ. Riyadh, Vol. 6 (2) pp. 185-191, 1980.
[5] A. Zahour, B. Taconet, and A. Faure, "Machine recognition of Arabic cursive writing", in From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon (eds.), Amsterdam, Elsevier Science Publishers B.V., 1992, pp. 289-296.
[6] R. Ramsis, S.S. El-Dabi, and A. Kamel, "Arabic character recognition system," IBM Kuwait Scientific Centre, report No. KSC027, Jan. 1988.
[7] H. Almuallim, and S. Yamaguchi, "A method for recognition of Arabic cursive handwriting", IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-9 No. 5, pp. 715-722, Sep. 1987.
[8] B. Parhami and M. Taraghi, "Automatic recognition of printed Farsi texts", Pattern Recognition, Vol. 14, Nos 1-6, pp. 395-403, 1981.
[9] Adnan Amin and J.F. Mari, "Machine recognition and Correction of Printed Text," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 5, pp. 13001306, 1989.
[10] Mou-Yen Chen, A. Kundu, and J. Zhou, "Off-line handwritten word recognition using a hidden Markov model type stochastic network", IEEE Trans. Pattern. Anal. Mach. Intell., Vol. PAMI-16, No. 5, pp. 481-496, 1994.
[11] S.J. Smith, M.O. Bourgoin, K. Sims, and L. Voorhees, "Handwritten Character Classification Using Nearest Neighbor in Large Databases", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 16, No. 9, pp. 915-919, 1994.
[12] T. Wakahara, "Shape Matching Using LAT and its Application to Handwritten Numeral Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 16, No. 6, pp. 618-629, 1994.
[13] J. Rocha and T. Pavlidis, "A Shape Analysis Model with Applications to a Character Recognition System", IEEE Transactions on Pattern Analysis and Machine

Intelligence (PAMI), Vol. 16, No. 4, pp. 393-404, 1994.
[14] H. Nishida and S. Mori, "An Algebraic Approach to Automatic Construction of Structural Models", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 15, No. 12, pp. 1298-1311, 1993.
[15] G. Wolberg, "A syntactic omni-font character recognition system," Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Miami Beach, Fla., USA, pp. 168-173, May 1986.
[16] W.J.M. Kickert and H. Koppelaar, "Application of fuzzy set theory to syntactic pattern recognition of handwritten capitals," IEEE Trans. Syst. Man Cybernet., SMC-6, No. 2, pp. 148-151, Feb. 1976.
[17] P. Siy and C.S. Chen, "Fuzzy logic for handwritten numeral character recognition," IEEE Trans. Syst. Man Cybernet., Vol. 4, No. 6, pp. 570-575, Nov. 1974.
[18] Y. LeCun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard, "Handwritten digit recognition: Applications of neural network chips and automatic learning", IEEE Communications Magazine, Vol. 27, No. 11, pp. 41-46, Nov. 1989.
[19] A. Rajavelu, M.T. Musavi, and M.V. Shirvaikar, "A neural network approach to character recognition", Neural Networks, Vol. 2, pp. 387-393, 1989.
[20] R.M.K. Sinha, B. Prasada, G.F. Houle, and M. Sabourin, "Hybrid Contextual Text Recognition with String Matching", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 15, No. 9, pp. 915-925, 1993.
[21] L. Xu, A. Krzyżak, and C.Y. Suen, "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition", IEEE Trans. Syst. Man Cybernet., SMC-22, No. 3, pp. 418-435, 1992.
[22] F. Ali and T. Pavlidis, "Syntactic Recognition of Handwritten Numerals," IEEE Trans. Syst. Man Cybern., Vol. SMC-7, pp. 537-541, 1977.
[23] M. Shridhar and A. Badreldin, "A High Accuracy Syntactic Recognition Algorithm for Handwritten Numerals," IEEE Trans. Syst. Man Cybern., Vol. SMC-15, pp. 152-158, 1982.
[24] B. Duerr, et al, "A combination of Statistical and Syntactical Pattern Recognition Applied to Classification of Unconstrained Handwritten Numerals," Pattern Recognition, Vol. 12, pp. 189-199, 1980.
[25] R.M. Bozinovic and S.N. Srihari, "Off-line cursive script word recognition", IEEE Trans. Pattern. Anal. Mach. Intell., Vol. PAMI-11, No. 1, pp. 68-83, 1989.
[26] B. Taconet, A. Zahour, and A. Faure, "A new global off-line recognition method for handwritten words", in From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon (eds.), Amsterdam, Elsevier Science Publishers B.V., 1992, pp. 327-338.
[27] J.J. Hull, T.K. Ho, J. Favata, V. Govindaraju, and S.N. Srihari, "Combination of segmentation-based and wholistic handwritten word recognition algorithms", in

From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon (eds.), Amsterdam, Elsevier Science Publishers B.V., 1992, pp. 261-272.
[28] J. Camillerapp, G. Lorette, G. Menier, H. Oulhadj, and J.C. Pettier, "Off-line and on-line methods for cursive handwriting recognition", in From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon (eds.), Amsterdam, Elsevier Science Publishers B.V., 1992, pp. 273-287.
[29] C.C. Tappert, "Cursive Script Recognition by Elastic Matching", IBM J. Res. Develop., Vol. 26, No. 6, pp. 765-771, 1982.
[30] M.K. Brown, Cursive word recognition, PH.D. dissertation, The university of Michigan, Ann Arbor, MI, University Microfilms International, 1981.
[31] Y. Fujimoto, S. Kadota, S. Hayashi, M. Yamamoto, S. Yajima and M. Yasuda, "Recognition of Handprinted characters by nonlinear elastic-matching", Proc. 3rd Int. Joint Conf. Pattern Recognition, pp. 113-118, Nov. 1976.
[32] E. Persoon and King-Sun Fu, "Shape Discrimination Using Fourier Descriptors", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-7, No. 3, pp. 170179, 1977.
[33] M.T.Y. Lai and C.Y. Suen, "Automatic Recognition of Characters by Fourier Descriptors and Boundary Line Encodings", Pattern Recognition, Vol. 14, Nos. 1-6, pp. 383-393, 1981.
[34] L.G. Shapiro," A Structural Model of Shape", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 2, pp. 111-126, 1980.
[35] A.B. Wang, K.C. Fan, and J.S. Huang, "Recognition of handwritten Chinese characters by modified relaxation methods", Image and Vision Computing, Vol. 12, No. 8, pp. 509-522, 1994.
[36] F.H. Cheng, W.H. Hsu, and M.Y. Chen, "Recognition of handwritten Chinese characters by modified Hough transform techniques", IEEE Trans. Pattern Anal. Machine Intell., Vol.11, No. 4, pp. 429-439, April 1989.
[37] C.H. Leung, Y.S. Cheung, and Y.L. Wong, "A knowledge-based stroke-matching method for Chinese character recognition", IEEE Trans. Syst. Man Cybernet., SMC-17, No. 6, pp. 993-1003, 1987.
[38] M.S. El-Wakil and A. Shoukry, "On-Line Recognition of Handwritten Isolated Arabic Characters", Pattern Recognition, Vol. 22, No. 2, pp. 97-105, 1989.
[39] S. Al-Emami and M. Usher, "On-Line Recognition of Handwritten Arabic Characters", IEEE Trans. Pattern Anal. Mach. Intell., Vol. 12, No. 7, pp. 704-710, 1990.
[40] C.C. Tappert, C.Y. Suen, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition", IEEE Trans. Pattern Anal. Mach. Intell., Vol. 12, No. 8, pp. 787-808, 1990.
[41] E. Mandler, R. Oed and W. Doster, "Experiments in on-line script recognition", in Proc. 4th Scandinavian Conf. Image Anal., pp. 75-86, June 1985.
[42] S. Impedovo, B. Marangelli, and M. Fanelli, "A Fourier Descriptor Set for Recognizing Nonstylized Numerals", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-8, No. 8, pp. 640-645, 1978.
[43] F. Schantz, The History of OCR Optical Character Recognition, Recognition Technologies Users Association (RTUA), USA, 1982.
[44] I.S.I. Abuhaiba and P. Ahmed, "Restoration of Temporal Information in Off-Line Arabic Handwriting", Pattern Recognition, Vol. 26, No. 7, pp. 1009-1017, 1993.
[45] I.S.I. Abuhaiba, S.A. Mahmoud, and R.J. Green, "Cluster Number Estimation and Skeleton Refining Algorithms for Arabic Characters", The Arabian Journal for Science and Engineering (AJSE), Vol. 16, No. 4B, pp. 519-530, 1991.
[46] S.A. Mahmoud, I.S.I. Abuhaiba, and R.J. Green, "Skeletonization of Arabic Characters using Clustering Based Skeletonization Algorithm (CBSA)", Pattern Recognition, Vol. 24, No. 5, pp. 453-464, 1991.
[47] T. Pavlidis, "Algorithms for graphics and image processing", Rockville MD: Computer Science Press, pp. 195-214, 1982.
[48] L. Lam, S.-W. Lee, and C.Y. Suen, "Thinning Methodologies - A Comprehensive Survey, " IEEE, PAMI, Vol. 14, No. 9, pp. 869-885, 1992.
[49] B.K. Jang and R.T. Chin, "Analysis of Thinning Algorithms Using Mathematical Morphology," IEEE, PAMI, Vol. 12, No. 6, pp. 541-551, 1990.
[50] E.R. Davies and A.P.N. Plummer, "Thinning algorithms: a critique and a new methodology", Pattern Recognition, Vol. 14, Nos. 1-6, pp. 53-63 1981.
[51] R.W. Smith, "Computer processing of line images: a survey," Pattern Recognition, Vol. 20, No. 1, pp. 7-15, 1987.
[52] E.S. Deutsh, "Thinning algorithms on rectangular, hexagonal and triangular arrays", Communs. Ass. Comput. Mach., Vol. 15, No. 9, pp. 827-837, 1972.
[53] R. Stefanelli and A. Rosenfeld, "Some parallel thinning algorithms for digital pictures", J. Ass. Comput. Mach., Vol. 18, No. 2, pp. 255-264, Apr. 1971.
[54] H. Tamura, "A comparison of line thinning algorithms from digital geometry viewpoint", Poc. 4th Int. Jt. Conf. on Pattern Recognition, Kyoto, Japan, pp. 715-719, IEEE 1978.
[55] D. Kalles and D.T. Morris, "A novel fast and reliable thinning algorithm", Image and Vision Computing, Vol. 11, No. 9, pp. 588-603, 1993.
[56] A. Rosenfeld and J. Pfaltz, "Sequential operations in digital picture processing," J. Ass. Comput. Mach., Vol. 13, pp. 471-494, 1966.
[57] L. O'Gorman, "Image and document processing techniques for the RightPages electronic library system", in Proc. 11th Int. Conf. Patt. Recogn. (ICPR), The Hague, The Netherlands, pp. 260-263, Aug. 1992.
[58] Y.M. Sharaiha and N. Christofides, "An Optimal Algorithm for the Straight Segment Approximation of Digital Arcs", CVGIP: Graphical Models and Image

Processing, Vol. 55, No. 5, pp. 397-407, 1993.
[59] E. Saund, "Identifying Salient Circular Arcs on Curves", CVGIP: Image Understanding, Vol. 58, No. 3, pp. 327-337, 1993.
[60] W.M. Tsang, P.C. Yuen, and F.K. Lam, "Detection of dominant points on an object boundary: a discontinuity approach", Image and Vision Computing, Vol. 12, No. 9, pp. 547-557, 1994.
[61] L. Wang and T. Pavlidis, "Detection of Curved and Straight Segments from Gray Scale Topography", CVGIP: Image Understanding, Vol. 58, No. 3, pp. 352-365, 1993.
[62] W-Y. Wu and M-J.J. Wang, "Detecting the Dominant Points by the CurvatureBased Polygonal Approximation", CVGIP: Graphical Models and Image Processing, Vol. 55, No. 2, pp. 79-88, 1993.
[63] C.-H. Teh and R.T. Chin, "On the Detection of Dominant Points on Digital Curves," IEEE, PAMI, Vol. 11, No. 8, pp. 859-872, 1989.
[64] C.-S. Fahn, J.-F. Wang, and J.-Y. Lee, "An Adaptive Reduction Procedure for the Piecewise Linear Approximation of Digitized Curves," IEEE, PAMI, Vol. 11, No. 9, pp. 967-973, 1989.
[65] U. Montanari, "A note on the minimal length polygonal approximation to a digitized contour", Communs. Ass. Comput. Mach., Vol. 13, No. 1, pp. 41-47, 1970.
[66] N.J. Naccache and R. Shinghal, "SPTA: A proposed algorithm for thinning binary patterss," IEEE Trans. Syst. Man Cybernet., SMC-14, No. 3, pp. 409-418, May/Jun 1984.
[67] S. Lee and J.C. Pan, "Offline Tracing and Representation of Signatures," IEEE Trans. Syst. Man Cybernet., SMC-22, No. 4, pp. 755-771, 1992.
[68] D.S. Doermann and A. Rosenfeld, "Temporal Clues in Handwriting", Proc. 11th International Conference on Pattern Recognition, Vol. II, pp. 317-320, 1992.
[69] V. Govindaraju and S.N. Srihari, "Separating Handwritten Text from Interfering Strokes", in From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon, eds., Elsevier Science Publishers B.V., pp. 17-28, 1992.
[70] A. Rosenfeld and E. Johnston, "Angle Detection on Digital Curves," IEEE Trans. Computers, pp. 875-878, 1973.
[71] S.A. Guberman and V.V. Rozentsveig, "Algorithm for the recognition of handwritten text", Avtomatika i Telemekanika, Vol. 5, pp. 122-129, May 1976.
[72] S. Mori, C.Y. Suen, and K. Yamamoto, "Historical Review of OCR Research and Development", Proceedings of the IEEE, Vol. 80, No. 7, pp. 1029-1058, 1992.
[73] G. Borgefors, "Distance Transformations in Digital Images," Computer Vision, Graphics, and Image Processing, 34, pp. 344-371, 1986.
[74] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press Inc., 1977.
[75] Samuel C. Lee, Modern Switching Theory and Digital Design, New Jersey, Prentice-Hall, 1978.
[76] A. Kandel, Fuzzy mathematical techniques with applications, Addison-Wesley, 1986.
[77] Zvi Kohavi, Switching and Finite Automata Theory, New York, McGraw-Hill, 1978.
[78] P. Ahmed and C. Y. Suen, "Computer Recognition of Totally Unconstrained Handwritten Zip Codes," International Journal of Pattern Recognition and Artificial Intelligence, Vol. 1 (1), pp. 1-15, 1987.
[79] P. Gader et. al., "Recognition of Handwritten Digits Using Template and Model Matching," Pattern Recognition, Vol. 24 (5), pp. 421-431, 1991.
[80] L.A. Fletcher and R. Kasturi, "A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images," IEEE, PAMI, Vol. 10, No. 6, pp. 910-918, 1988.
[81] R. Kasturi, S.T. Bow, W. El-Masri, J. Shah, J.R. Gattiker, and U.B. Mokate, "A System for Interpretation of Line Drawings," IEEE, PAMI, Vol. 12, No. 10, pp. 978-992, 1990.
[82] C.Y. Suen, "n-Gram Statistics for Natural Language Understanding and Text Processing", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 1, No. 2, pp. 164-172, 1979.
[83] R. Attar, Y. Choueka, N. Dershowitz, and A.S. Fraenkel, "KEDMA-Linguistic tools for retrieval systems", J. Ass. Comput. Mach., Vol. 25, pp. 52-66, 1978.
[84] L. Baker and J.L. Santa, "Context, integration, and retrieval", Memory and Cognition, Vol. 5, pp. 308-314, 1977.
[85] C. Conrad, "Context effects in sentence comprehension: A study of the subjective lexicon", Memory and Cognition, Vol. 2, pp. 130-138, 1974.
[86] L.J. Evett, C.J. Wells, F.G. Keenan, T. Rose, and R.J. Whitrow, "Using linguistic information to aid handwriting recognition", in From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J.C. Simon (eds.), Amsterdam, Elsevier Science Publishers B.V., 1992, pp. 339-348.
[87] G.T. Toussaint, "The Use of Context in Pattern Recognition", Pattern Recognition, Vol. 10, pp. 189-204, 1978.
[88] A.R. Hanson, E.M. Riseman, and E. Fisher, "Context in Word Recognition", Pattern Recognition, Vol. 8, pp. 34-45, 1976.
[89] M. Shimura, "Recognizing Machines with Parametric and Nonparametric Learning Methods Using Contextual Information", Pattern Recognition, Vol. 5, pp. 149-168, 1973.
[90] R. Bozinovic and S. Srihari, "A String Correction Algorithm for Cursive Script Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 4, No. 6, pp. 655-663, 1982.
[91] J.J. Hull and S. Srihari, "Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 4, No. 5, pp. 520-530, 1982.
[92] R. Shinghal and G.T. Toussaint, "Experiments in Text Recognition with the Modified Viterbi Algorithm", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 1, No. 2, pp. 184-193, 1979.
[93] R. Shinghal, D. Rosenberg, and G.T. Toussaint, "A Simplified Heuristic Version of a Recursive Bayes Algorithm for Using Context in Text Recognition", IEEE Trans. Syst. Man Cybernet., SMC-8, No. 5, pp. 412-414, 1978.
[94] L. O'Gorman, "The Document Spectrum for Page Layout Analysis", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 15, No. 11, pp. 1162-1173, 1993.
[95] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan, "Syntactic Segmentation and Labelling of Digitized Pages from Technical Journals", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 15, No. 7, pp. 737-747, 1993.

# Publications from the Thesis 

[1] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta, "Processing of Off-Line Handwritten Text: Polygonal Approximation and Enforcement of Temporal Information", CVGIP: Graphical Models and Image Processing, Vol. 56, No. 4, pp. 324-335, 1994.
[2] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta, "Straight line approximation and 1D representation of off-line handwritten text", Image and Vision Computing, Vol. 12, No. 10, pp. 649-659, 1994.
[3] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta, "Restoration of Temporal Information from Static Images of Handwritten Arabic Script", IEEE SICSPCS '95: IEEE Singapore International Conference on Signal Processing, Circuits and Systems '95, pp. 304-307, July 1995, Singapore.
[4] I. S. I. Abuhaiba, S. Datta, and M. J. J. Holt, "Line Extraction and Stroke Ordering of Text Pages", ICDAR'95: Third International Conference on Document Analysis and Recognition, pp. 390-393, August 1995, Montreal, Canada.
[5] I. S. I. Abuhaiba, S. Datta, and M. J. J. Holt, "Processing of Text Documents: Straight Line Approximation and Lost Loop Recovery", ICDAR'95: Third International Conference on Document Analysis and Recognition, pp. 1157-1160, August 1995, Montreal, Canada.
[6] I. S. I. Abuhaiba, S. Datta, and M. J. J. Holt, "Fuzzy State Machines to Recognize Totally Unconstrained Handwritten Strokes", Image and Vision Computing, Vol. 13, No. 10, pp. 755-769, 1995.
[7] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta, "Processing of Binary Images of Handwritten Text Documents", Pattern Recognition, Vol. 29, No. 7, 1996.

## Appendix A

## Learning Data Set of the IACR System

$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$

Figure A.1. Guiding sheet used to collect learning data set of the IACR system. Strokes are handwritten on another blank sheet which is attached on top of this sheet.

 rrerrrrrrrrrrrrrrrrrrrru
 0000000000000000000000

 へへヘヘヘ ヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘ＾ 99999999999999999999999999 SGS SGG GSG GSS SGS GGSSGS G GS SGG \＆ ～～～～ウ～～～～～～～～～～～～～～～～～～


) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) )
 66666666666666666666666666


Figure A．2．Page No． 1 of learning data which were provided by Subject No． 1 and used in the learning stage of the IACR system．
 esesecesesereseseceseseseseses ヶr urrurrrurrrurrurrrrurruru
 0000000000000000000000000000
 vvvvvvvvvvvvvvvvvvvv vvvvvvvv
 9999999999999999999999999


 ノイノノノノノノノノイノノノノノノノノノノノノノノノ
 ）））））））））））））））））））））））））））））））
 6666666666666666666666666666


Figure A．3．Page No． 2 of learning data which were provided by Subject No． 1 and used in the learning stage of the IACR system．
lllllllll111111111111111111111111111 esereseseseseceseceseceseseb rrrrrrrrurrrrurrurrrrurrrru
 00000000000000000000000


 99999999999999999999999999
 ~~~~~~~~~~~~~~~~~~~ ~~



Figure A.4. Page No. 1 of learning data which were provided by Subject No. 2 and used in the learning stage of the IACR system.

いいいいいいいいいいいいいいいいい
 ケrr rrrrrrrrrrrrrrrrrr
 0000000000000000000000000 า7า7าา7า7า7า7า7าาาา7าフาフาา

 99999999999999999999999999
 ルー～ル～～～～～～～～～～～～～～～～







Figure A．5．Page No． 2 of leaming data which were provided by Subject No． 2 and used in the learning stage of the IACR system．
 cecesesescesesesesecececece
 $\varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \Sigma \varepsilon \varepsilon \Sigma \Sigma \Sigma \Sigma \varepsilon \Sigma \varepsilon \varepsilon \Sigma \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon$ 00000000000000000000000000
 VVVVVVVVVVVVVVVVVVVVVVVVv
 999999999999999999999999 sSSSSSSS SSSSSSSSSSSS S s \&

 $+t+t+++t+++t+t++t++t++t+t$ ) ) ) ) ) ) ) ) ) ( ) ) ) ) ) ) ) ) ( ) ) ) ( $(C(1)$ 6666666666666666666666666 و و

Figure A.6. Page No. 1 of learning data which were provided by Subject No. 3 and used in the learning stage of the IACR system.
V11111111111111111111111

 000000000000000000000000
 $v \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee v \vee v$
 999999994999999999499999999 SSSSSSSSSSSSSSSSSSGSSS SS\&\& ~~~~~~~~~~~~~~~~~~~~~~~~
 ///////////|/1///|////////////ノ $t+t+t+t+t+t+t+t+t+t+t+t+$
 ( 666666666666666666666666666


Figure A.7. Page No. 2 of learning data which were provided by Subject No. 3 and used in the learning stage of the IACR system.
 cecesecercecesecececerececececesect

 $00000000000000000000000000^{0}$



 sssssss sssessssssusssssssss \& ~~~~~~~~~~~~~~~~~~~~~~~~

 +t+t+t+t+t+t+t+t+t+t+t+t+

 666666666666666666666666666664666


Figure A.8. Page No. 1 of learning data which were provided by Subject No. 4 and used in the learning stage of the IACR system.

11111111111111111111111111111111


 0000000000000000000000000000000000


 9999999999999999999999999999999




+     +         +             +                 +                     +                         +                             +                                 +                                     +                                         +                                             +                                                 +                                                     + ++ + + + + + + + + + + + +

 666666666666666666666666666666666


Figure A.9. Page No. 2 of leaming data which were provided by Subject No. 4 and used in the learning stage of the IACR system.

1111111111111111111111111 cocesecocecosecescescececos rrrrrurrrrrrrurrrururrrrrur
 000000000000000000000000000000


 9999999999999994999999999999999
 ~~~~~~~~~~NNN~N~~~~~~N~~N~



```
    -----------------------------
+++++++++++++++++++++++++++++++++
```




```
666666666666666666666666666666666666
g
```

Figure A.10. Page No. 1 of learning data which were provided by Subject No. 5 and used in the learning stage of the IACR system.

## PAGE

## MISSING

## IN

ORIGINAL

## Appendix B

## Testing Data Set of the IACR System

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure B.1. Guiding sheet used to collect testing data set of the IACR system. Strokes are handwritten on another blank sheet which is attached on top of this sheet.

11111111111111111111


 $\begin{array}{llllllllllllllllllll}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{array}$



 $G \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \&$ ~~~~~~~~~~~~~~~~~~~~~~~~~



Figure B.2. Testing data which were provided by Subject No. 1 and used in the testing stage of the IACR system.

11111111111111111111
 $\mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu \mu$ と $\mathcal{1} \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma \Sigma$ 000000000000000000000 フᄀ77777777777777777777
$\vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee$. $\wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge$ 499999949999999999999
 $\sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim$ ש $\omega$


Figure B.3. Testing data which were provided by Subject No. 2 and used in the testing stage of the IACR system.
$\begin{array}{llllllllllllllllllll}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{array}$





$$
7777777777777777777777
$$





 $\omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega \omega$

$$
\begin{aligned}
& \begin{array}{llllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9
\end{array}
\end{aligned}
$$

Figure B.4. Testing data which were provided by Subject No. 3 and used in the testing stage of the IACR system.












Figure B.5. Testing data which were provided by Subject No. 4 and used in the testing stage of the IACR system.













$$
\begin{aligned}
& t+t+t+t+t+t+t+t+t+t+t+t+t
\end{aligned}
$$

Figure B.6. Testing data which were provided by Subject No. 5 and used in the testing stage of the IACR system.

11111111111111111111111111
SS S LS LS S SS S SS SS SS SS S






 \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \&




Figure B.7. Testing data which were provided by Subject No. 6 and used in the testing stage of the IACR system.

11111111111111111111






 aq a a aq qq q aq aq a a a a a a a aq





Figure B.8. Testing data which were provided by Subject No. 7 and used in the testing stage of the IACR system.

11111111111111111111111







 $4 \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \&$
 Jj

$$
\begin{aligned}
& \begin{array}{llllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9
\end{array}
\end{aligned}
$$

Figure B.9. Testing data which were provided by Subject No. 8 and used in the testing stage of the IACR system.
1111111.1111111111111111


 0000000000000000000000
フ．フᄀ フ フ フ フ フ フ フ フ フ フ フ フ フ フ フ フ フ
$\vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee$
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 $\sim \sim \sim \dot{\sim} \sim \sim r \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim$


$$
\begin{aligned}
& \text { ノノノノノノノノーノノノノノノノノノノノノノ } \\
& t++t+t+t++t+t+t+t+t+t \\
& \text { ) ( ) , ( ) , , () , ) , ) , ) , , ) , }
\end{aligned}
$$

$$
\begin{aligned}
& \begin{array}{llllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9
\end{array}
\end{aligned}
$$

Figure B．10．Testing data which were provided by Subject No． 9 and used in the testing stage of the IACR system．

11111111111111111111


$\varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \varepsilon$ 00000000000000000000

 ヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘヘ 99999999949949999999 4 \＆ 4 \＆ ～～～～～～～～～～～～～～～～～～～～～～
 ノノノ／／／／／／／／／／／／／／／／ノ $t+t+t+t+t++t++t++t+t$
 C C C C C C C C C L L L C C 66666666666666666666 99999999999999999999

Figure B．11．Testing data which were provided by Subject No． 10 and used in the testing stage of the IACR system．

11111111111111111111111












$$
\begin{aligned}
& t+t+t+t++++t+t+t+t+t
\end{aligned}
$$

$$
\begin{aligned}
& \angle C \angle C \angle C \angle C \angle C 16 \\
& 66666666666666666 \quad 6 \quad 6 \quad 6 \quad 6 \quad 6 \quad 6
\end{aligned}
$$

Figure B.12. Testing data which were provided by Subject No. 11 and used in the testing stage of the IACR system.

111111111111111111111111












$$
\begin{aligned}
& \text { - - - - - - - - - - - - - - - - - }
\end{aligned}
$$

$$
\begin{aligned}
& \text { 人, , , , , , , , , , , , , , , , , , , } \\
& \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \\
& \begin{array}{lllllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & G & G & 0 & 6 & 6 & 6
\end{array}
\end{aligned}
$$

Figure B.13. Testing data which were provided by Subject No. 12 and used in the testing stage of the IACR system.

11111111111111111111
$\subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset \subset$

 $\begin{array}{llllllllllllllllllll}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{array}$



 s \& \& \& \& \& \& \& \& \& \& \& \& \& \& \&




Figure B.14. Testing data which were provided by Subject No. 13 and used in the testing stage of the IACR system.



 9999 q 9 9 9999999999999999

 س w w w w w w w w w w w w w w w w


Figure B.15. Testing data which were provided by Subject No. 14 and used in the testing stage of the IACR system.

1111111111111111111







 s 5 S 4 \& 4 y 4 \& 4 y 4 4 4 y 4 y 4 y 4




Figure B.16. Testing data which were provided by Subject No. 15 and used in the testing stage of the IACR system.

111111111111111111111111








 ~~~ ~~~~~~~~~~~~~~~~~~~~



Figure B.17. Testing data which were provided by Subject No. 16 and used in the testing stage of the IACR system.

11111111111111111111













Figure B.18. Testing data which were provided by Subject No. 17 and used in the testing stage of the IACR system.

11111111111111111111


 000000000000000000000
 $\vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee$ $\wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge$ 99999999999999999999 s \＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆\＆ ～～N～～N～～～～～～N～～～～～N～N～


$$
\begin{aligned}
& \text { ノノノノノノノノノノノノノノノノノノノノノノノ } \\
& t+t+t+t+t+t+t+t+t+t+
\end{aligned}
$$

$$
\begin{aligned}
& 66666666666666666666
\end{aligned}
$$

Figure B．19．Testing data which were provided by Subject No． 18 and used in the testing stage of the IACR system．

11111111111111111111




$\vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \vee \dot{v} \vee \vee \vee \vee$
$\wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge$
99999999999999999999


$\omega \omega$

$$
\begin{aligned}
& \text { ノ ノ / ノ ノ / ノ / ノ ノ ノ ノ ノ ノ ノ ノ ノ ノ ノ ノ } \\
& +++++t++++_{+}^{+}+t+t+t+t+
\end{aligned}
$$

$$
\begin{aligned}
& \begin{array}{lllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{array} 6 \\
& 9,9,9,9,9,9,9,9
\end{aligned}
$$

Figure B．20．Testing data which were provided by Subject No． 19 and used in the testing stage of the IACR system．

11111111111111111111






 a 9 a a 9 a a a a 9 a a 9 a a a a 9 a a $4 \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \& \&$



$$
\begin{aligned}
& \begin{array}{lllllllllllllllllll}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{array}
\end{aligned}
$$

Figure B.21. Testing data which were provided by Subject No. 20 and used in the testing stage of the IACR system.

## Appendix C

Learning Data Set of the CASR System

（18） ســ صمر


－لا
á ه وَ和
 فتَ原 ，安 ～ا
尾

 الـ


Figure C．1．Learning data which were provided by Subject No． 1 and used in the leaming stage of the CASR system．
 أولذنم

المرححـتَ 6 و اللذ لe







! !
 ذذ وتّمر

وْل




 اله النّانِّ
 اننع كثيرُ الل الدتیِ~ مـوiا

Figure C.2. Learning data which were provided by Subject No. 2 and used in the learning stage of the CASR system.





لـدـِّبَ النتّلمبيه هذه زوُ توْر وقتًا كنٌ , انِّنا ا,

 بِا من اهرافف غي مكال من مبالغ



Figure C.3. Learning data which were provided by Subject No. 3 and used in the learning stage of the CASR system.









;
ف

 و gl وعهِ





Figure C.4. Learning data which were provided by Subject No. 4 and used in the learning stage of the CASR system.










和
 st الدها
 سي


位
 مذ'




Figure C.5. Learning data which were provided by Subject No. 5 and used in the learning stage of the CASR system.


- الـط




 للصناعی gU

.




Figure C.6. Learning data which were provided by Subject No. 6 and used in the learning stage of the CASR system.




 للعطاء و من هـنزه الاصناونا د هن منا
 icier هوتاُوْ حا ( ll




Figure C.7. Learning data which were provided by Subject No. 7 and used in the learning stage of the CASR system.

 عمل بروسا و اكها




 sc النزّاع الدنی




 الستجاري



Figure C.8. Learning data which were provided by Subject No. 8 and used in the learning stage of the CASR system.

 اذا > خل كـك
 انى تَتِلِّل



 النكئِ



 wm




Figure C.9. Learning data which were provided by Subject No. 9 and used in the learning stage of the CASR system.
 و اشكبرى أهـو كل


 -


 هـ





 و تلرك توا





Figure C.10. Learning data which were provided by Subject No. 10 and used in the learning stage of the CASR system.










 =









Figure C.11. Learning data which were provided by Subject No. 11 and used in the learning stage of the CASR system.

؛

















 القلفـد了 $\int$





Figure C.12. Learning data which were provided by Subject No. 12 and used in the learning stage of the CASR system.


 U1 PO 0 0
踏 in）


 و غ
垪垪g Ur
 ＂

Figure C．13．Learning data which were provided by Subject No． 13 and used in the learning stage of the CASR system．

## Appendix D

## Testing Data Set of the CASR System






- منسـوبَ




 النُردوس البســـتا~ اكِامع كر ما

Figure D.1. Testing data which were provided by Subject No. 1 and used in the testing stage of the CASR system.



بــ

 ن

 الدتنو ,
 Ff

Figure D.2. Testing data which were provided by Subject No. 2 and used in the testing stage of the CASR system.
 6 Ci



 vo أُموا به نهو منا مرَ عقَلِيَ وثورة



Figure D.3. Testing data which were provided by Subject No. 3 and used in the testing stage of the CASR system.








رقا يُ الحناهم وذلتn



Figure D.4. Testing data which were provided by Subject No. 4 and used in the testing stage of the CASR system.




ser الهِ|


程




Figure D.S. Testing data which were provided by Subject No. 5 and used in the testing stage of the CASR system.

ت




النَائى

$$
\begin{aligned}
& \text {. }
\end{aligned}
$$

urns

Figure D.6. Testing data which were provided by Subject No. 6 and used in the testing stage of the CASR system.



 اُما راذ


 نِّتْ保




Figure D.7. Testing data which were provided by Subject No. 7 and used in the testing stage of the CASR system.



الصァراs با

الحت
نـر

اء




Figure D.8. Testing data which were provided by Subject No. 8 and used in the testing stage of the CASR system.






ور
على





Figure D.9. Testing data which were provided by Subject No. 9 and used in the testing stage of the CASR system.












Figure D.10. Testing data which were provided by Subject No. 10 and used in the testing stage of the CASR system.

آّ







 بلج Clan

Figure D.11. Testing data which were provided by Subject No. 11 and used in the testing stage of the CASR system.
del, Font الد,
 و التجَهِ次 -~少

 بَ و
 coL

Figure D.12. Testing data which were provided by Subject No. 12 and used in the testing stage of the CASR system.

く伍 أن ك



 نتَمr ar
 تَدر 1 لַِ الأنــا ،

Figure D.13. Testing data which were provided by Subject No. 13 and used in the testing stage of the CASR system.

العقع



نتَه



 J

 , J ا

Figure D.14. Testing data which were provided by Subject No. 14 and used in the testing stage of the CASR system.
 إمكاسِت

 gl,
oft
 و و و ( لنَّ in


Figure D.15. Testing data which were provided by Subject No. 15 and used in the testing stage of the CASR system.
 الا


位
Ch ك


角的 ل ل

Figure D．16．Testing data which were provided by Subject No． 16 and used in the testing stage of the CASR system．

 الـمَّةَوالأورا

 اجكِلهَ والُ شبا سالمعفوفة على جوانب اللُرقوعلى







Figure D.17. Testing data which were provided by Subject No. 17 and used in the
testing stage of the CASR system testing stage of the CASR system.

 ينير نی


 انضم لا كِس للنظام الد بالـ• دَ النو النمِر سرالا

 كاáa

Figure D.18. Testing data which were provided by Subject No. 18 and used in the testing stage of the CASR system.














Figure D.19. Testing data which were provided by Subject No. 19 and used in the testing stage of the CASR system.

لِ الد نــ Li, ~ الضمها
 i اللوْ لضون , عیر برامّ الد
 ت ( , ,


Figure D.20. Testing data which were provided by Subject No. 20 and used in the testing stage of the CASR system.


[^0]:    (C) by Ibrahim Sulaiman Ibrahim Abuhaiba, 1996

