Template Based Recognition of On-Line Handwriting

Sternby, Jakob

2008

Preface

I am always doing that which I cannot do, in order that I may learn how to do it.

Pablo Picasso

List of Papers

[JS1] Jakob Sternby, Jonas Andersson, Jonas Morwing and Christer Friberg. On-line Arabic Handwriting Recognition With Templates. In *Proc. of the First International Conference on Frontiers in Handwriting Recognition* (2008), Accepted.

[JS2] Jakob Sternby. Graph Based Shape Modeling for On-line Character Recognition. (2008), Manuscript.

[JS3] Jakob Sternby. An Additive Single Character Recognition Method. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 417-422.

[JS4] Jakob Sternby. Prototype Selection Methods for On-line HWR. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 157-160.

[JS5] Jakob Sternby. Class Dependent Cluster Refinement. In *Proc. 18th International Conference on Pattern Recognition* (2006), pp. 833-836.

[JS6] Jakob Sternby. Frame Deformation Energy Matching of On-line Handwritten Characters. In *Proceedings of the 10th Iberoamerican Congress on Pattern Recognition* (2005), pp. 128-137.

[JS7] Jakob Sternby. Structurally Based Template Matching of On-line Handwritten Characters. In *Proc. of the British Machine Vision Conference 2005* (2005), pp. 250-259.

[JS8] Jakob Sternby and Christer Friberg. The Recognition Graph - Language Independent Adaptable On-line Cursive Script Recognition. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 14-18.

[JS9] Jakob Sternby and Anders Ericsson. Core Points - A Framework For Structural Parameterization. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 217-221.

[JS10] Jakob Sternby and Anders Holtsberg. Core Points for Segmentation and Recognition of On-line Cursive Script. In *Proceedings SSBA'05 Symposium on Image Analysis* (2005), pp. 37-40.

[JS11] Jakob Sternby. On-line Signature Verification by Explicit solution to the Point Correspondence Problem. In *First International Conference on Biometric Authentication* (2004), pp. 569-576.

Organization of the thesis

This thesis contains a sufficient set of strategies for implementing a template based on-line handwriting system for recognition of words or characters from arbitrary scripts. The outline of the thesis will follow a typical textbook approach where subproblems and the tools needed to solve them are presented in order of need. A generalized description of the handwriting recognition problem as well as an overview of state-of-the-art techniques starts off the thesis in Chapter 2. Preprocessing, normalization of input and segmentation strategies are covered in Chapter 3. The simplest recognition case of isolated single characters including the specific additivity concept for template distance functions are explained in Chapter 4. Extension of the additive template distance function with graphs to handle connected script as well as noisy input is found in Chapter 5. Many connected scripts have the characteristic that strokes belonging to the same character may not always be written together. A strategy for treating this property with the connected script recognition described in Chapter 5 is found in Chapter 6. This is followed by an explanation of how to add linguistic knowledge efficiently to the recognition process by utilizing a graph structured dictionary in Chapter 7. Some automatic techniques for improving the template modeling through specialized clustering algorithms are described in Chapter 8. Since the recognition system implements the strategies presented in Chapters 3-8, experiments have been gathered in the dedicated Chapter 9. The final chapter contains a brief summary and provides some suggestions on further developments to strategy presented in this thesis.

Acknowledgments

This work has been funded by the Swedish Research Council (diary nr. VR 2002-4249) and Zi Decuma and I am very thankful for the opportunity that was given to me. I am also grateful for several generous grants from the Lannér-Dahlgren foundation, the Kungliga Fysiografiska sällskapet as well as from Nordqvists foundation, enabling me to attend and present my work at several international conferences.

Obviously I would never have been able to write this thesis if it weren't for my inspiring supervisors, Kalle Åström and Rikard Berthilsson. They are two of the founders of Decuma AB, where I was just recently employed when this project started. They are notoriously enthusiastic and have provided lots of encouragement as well as valuable input, especially during the first critical years on this road.

The atmosphere at Decuma has been and still is truly stimulating, filled with people thirsty for knowledge. Although I feel indebted to everyone who has worked there during this period, the permanent staff consisting of Jonas Andersson, Jonas Morwing and Christer Friberg deserve special credit. Especially during the last year they have all largely contributed to the progress of the thesis through valuable discussions as well as hard work.

Sincere thanks also to the people in the vision group at Lund Institute of Technology. Particularly during my first years as a PhD student, the company of the other researchers in the vision group made the time spent reading and studying so much more interesting.

Friends, relatives and (more recently) in-laws have always been an important part of my life, and you all deserve credit. Many of you have motivated me by manifesting an unwavering belief in my work despite having but a clue of what I am doing - that is friendship. In particular I want to thank my parents, brother and sister and their spouses for being ever supportive during this period.

Finally, and most important throughout my work, has been the continued support of my beloved wife, Hanna. Our son Viktor has certainly also contributed to my work by constantly providing a fun and stimulating escape from the academic reality.

Jakob Sternby

Lund, April 2008

Notation

In general the jth element X of a set will be denoted by X_j. The aim has been to maintain a consistent notation throughout the thesis using the notation listed below. Where this is not the case the meaning of a certain notation should be possible to derive from the context.

X	A sample of online handwriting
$\mathcal{X}$	A penup reordering variation of X
X^j	The jth stroke of sample X
Λ_i	The ith segment of X
$\Lambda_i^{X^j}$	The ith segment of stroke j in X
p	A point $(x, y) \in \mathbb{R}^2$ in the input coordinate system
p_y	The vertical coordinate of a point p
p_x	The horizontal coordinate of a point p
$\mathbb{X}$	The sample space
$\mathbb{X}_k$	The sample space of class k
$\mathbb{F}$	The set of monotone increasing functions $f : \mathbb{Z} \to \mathbb{Z}$
$\mathfrak{f}$	A point in a feature space
$\mathfrak{F}^m$	Feature space of dimension m
$\mathcal{S}(X)$	The segmentation points of the sample X
A	A segment arctype
$\mathcal{M}$	A flexible model of a character

T	A static template of a character
$\mathcal{T}$	A template sequence
$\mathbb{T}$	Space of template sequences
$\mathbb{D}$	The set of all templates (database)
Φ	Alignment function
$\mathfrak{O}$	Segment connection $\in \{+, \cup\}$

Graph Notation

$\mathfrak{S}$	A segmentation graph
$\mathfrak{R}$	A recognition graph
d	A partial distance value as for an edge
D	A path distance
e	An edge in the segmentation graph
T_e	The template corresponding to edge e
N	A graph node
E	A set of edges
v	A path in the recognition graph
V	A set of paths
$B_{\mathfrak{S}}$	Segmentation graph beam width
B_I	Segmentation graph incomplete beam width
$B_{\mathfrak{R}}$	Recognition graph beam width
$B_{\mathfrak{D}}$	Diacritic search space beam
B_U	Diacritic unmatched edge beam width

Abbreviations

ASM	Active Shape Model
DP	Dynamic Programming
DTW	Dynamic Time Warping
CRF	Conditional Random Field
HMM	Hidden Markov Modeling
HWR	Handwriting Recognition
kNN	k-Nearest Neighbor
LVQ	Learning Vector Quantization
MAP	Maximum A Posteriori
ML	Maximum Likelihood
MLP	Multi Layer Perceptron
PCA	Principal Components Analysis
SCR	Single Character Recognition
SOM	Self-Organizing Map
SVM	Support Vector Machines
TDNN	Time Delay Neural Network
UCR	Unconstrained Handwriting Recognition
VQ	Vector Quantization

Contents

Preface **i**

List of Papers ii

Organization iii

Acknowledgements iv

Notation **v**

Abbreviations **vii**

Contents **ix**

1 Overview **1**

1.1 Contributions 3

2 Handwriting Recognition **7**

2.1 On-line Handwriting Recognition 8

2.2 Single Character Recognition Methods 11

2.3 Statistical Models 13

2.4 Template Based Methods 17

2.5 Syntactic 19
2.6 Implicit Modeling 19
2.7 Combination Methods 23

3 Preprocessing And Segmentation 25
3.1 Introduction 26
3.2 Segmentation 27
3.3 Preprocessing 43
3.4 Experiments 46

4 Additive Template Matching 47
4.1 The Frame Deformation Energy Model 48
4.2 Feature Space 49
4.3 Distance Function in $\mathfrak{F}^m$ 54
4.4 Segmented Template Matching 55

5 Connected Character Recognition With Graphs 59
5.1 Introduction 60
5.2 Connection Properties 61
5.3 Segmentation Graph 65
5.4 Noise Modeling 72
5.5 The Recognition Graph 75
5.6 Preprocessing Parameters 79

6 Delayed Strokes And Stroke Attachment 81
6.1 Introduction 82
6.2 Diacritic Modeling 83
6.3 Pen-up Attachment 85
6.4 Dynamic Treatment Of Diacritic Strokes 86

7 Application of Linguistic Knowledge 97
7.1 Introduction 98
7.2 Contextual Shape 100

7.3 Static Lexical Lookup . . . 103
7.4 Dynamic Lexical Lookup . . . 106
7.5 Conclusions . . . 106

8 Clustering And Automatic Template Modeling Techniques 107
8.1 Introduction . . . 108
8.2 Holistic Clustering . . . 109
8.3 Segmentation Definition Database . . . 110
8.4 Forced Recognition And Labeling . . . 110
8.5 Segmental clustering . . . 113
8.6 The Variation Graph . . . 114
8.7 Recognition with Variation Graphs . . . 117

9 Experiments 119
9.1 Introduction . . . 120
9.2 Datasets . . . 120
9.3 Single Character Recognition . . . 123
9.4 Unconstrained Character Recognition . . . 126

10 Conclusions and Future Prospects 139

Bibliography 143

Index 157

A UNIPEN Train-R01/V07-1a Allograph Content 159
A.1 Training Set Allograph Distribution . . . 159
A.2 Test Set Allograph Distribution . . . 162

CHAPTER 1

Overview

To know that we know what we know, and to know that we do not know what we do not know, that is true knowledge.
Nicolaus Copernicus

Software for recognition of handwriting has been available for several decades now and research on the subject has generated several different strategies for producing competitive recognition accuracies, especially in the case of isolated single characters. From a commercial perspective, the task of producing handwriting recognition that will be accepted and ultimately a preferential choice of input method for users of devices such as mobile phones is not simple. Some crucial aspects are that (1) recognition accuracy is a subjective matter highly dependent on the data sets used for experiments and (2) in terms of user satisfaction, the types of errors made by the system also affect the opinion on system performance. Typically, users of a system are more lenient in judgement of performance in terms of accuracy when they have more understanding of why certain characters are confused. Such an understanding also provides users with the necessary tools to adapt (even if the adaptation is subconscious) and use writing styles that generate less conflicts and thus a higher recognition accuracy. Ultimately such information could also be used to explicitly allow users to interact with the system and decide the types of shapes that should be associated to each output symbol. This paradigm contrasts somewhat to the methods presented during the past decade of research, where the implicitly modeled recognizers that can be automatically trained, tested and compared on a given data set have been favored.

Since the work in this thesis has been conducted in cooperation with the handwriting recognition company Zi Decuma (with funding from Zi Decuma and the Swedish Research Council [1]), the focus has been on improving technology for the type of recognition system described above, where the use of explicit modeling makes recognition limitations transparent. Other factors that are critically important to commercial systems are memory consumption and response time, as stated in [5]. Much through the highly competent implementations by coworkers at Zi Decuma, the experimental results of the system presented here are very competitive in this respect and clearly within the scope of current hardware limitations even for simpler mobile platforms.

The problem of recognizing samples of handwriting with arbitrary connections between constituent characters (*unconstrained handwriting*) adds considerable complexity in form of the segmentation problem. In other words a recognition system, not constrained to the isolated single character case, needs to be able to identify the location in the sample where one letter ends and another begins. In view of published results during the last decade, the most common technique for solving this problem has been to train multi-layer networks for partial character recognition and combine results by means of sequence oriented

[1] Vetenskapsrådet, `http://www.vr.se`

modeling such as Hidden Markov Models. The work in this thesis will recapitulate some of the simplest and most basic pattern recognition techniques and step-by-step show how these can be extended to tackle the difficulties inherent to the unconstrained recognition problem. Being a completely template based approach with a possibly transparent database, there is explicit information on the types of samples that can be correctly recognized by the system. Thus the system provides some intuition regarding the types of errors (confusions) that the system is prone to make.

Due to commercial requirements, the primary system reported in the experimental section of the thesis was developed for Arabic unconstrained recognition, but the underlying techniques are fully applicable to other scripts, and just require the production of another shape definition database. This versatility and intrinsic sequential property is common to HMMs and in many respects the work of this thesis can be viewed as a non-probabilistic exploration of sequence based recognition.

1.1 Contributions

Although the topic and the fundamental approach taken to recognition in this thesis is the same as in my licentiate thesis [113], there are important differences and none of the experiments conducted for that thesis have been replicated here. The main reason for this is that the complete recognition implementation has been reimplemented in this thesis. A preliminary version with all components presented here was implemented mainly in matlab and with my guidance this has been ported to ANSI c-code mainly by my co-workers at Zi Decuma. In the experimental section some of the experiments have been conducted in the matlab environment (single character experiments) and some with the C-implementation. Some of my previous publications did not fit the scope of the thesis and have not been included here [112, 117, 118]. From my point of view the most important contributions is the additive concept for template methods presented in [116], the diacritic handling partly included in [120] and the variation graph concept in [119]. A complete breakdown of the contributions for each chapter is given below. For all of the papers, the roles of included co-authors have been confined to (1) being discussion partners or (2) implementing a certain part of the experimental setup and they have not participated in the writing process. In all cases the contributions of these individuals have been specified below.

The contents of Chapter 3 mainly derive from papers [121], [115] and [114]. Segmented shape description, Dijkstra Curve Maximization and the arctype approximation based on Dynamic Programming are the main contributions

presented in this chapter. Of these, the segmented shape description originated from discussions about applying Active Shape to handwriting recognition, but the actual use of segmentation as alignment prior to PCA can be attributed to the author.

The main contributions of Chapter 4 are the features used in recognition, especially the relative features which induce the Markov characteristic in the sequential template matching, the noise handling, and the additive concept, all more or less derived from [116]. Of these, the original length ratio and connection angle features were initially proposed by the author while the relative positions are complementary suggestions by Jonas Andersson derived from the C-implementation of the core system from Matlab.

Chapters 5 and 7 originate in papers [123] and [122] and the results from these papers were also included in my licentiate thesis [113]. In particular the first of these papers [123], presenting a template based system with a single segmentation graph with equal conceptual contributions from the author and Anders Holtsberg later proved not to contain any remarkable findings compared to previous research, but failure to find prior art such as [50] led us to believe that a template based segmentation graph was a new contribution. Nevertheless further development led to the main contribution of the chapter, the dual graph structure presented in [122], where path expansion can be done with simultaneous path-wise adjustments and fast lexical validation. In this last paper, the system and ideas can be attributed to the author while the lexicon and dictionary functions were implemented by Christer Friberg. In the latest incarnation of the system the segmentation graph was implemented in C by Jonas Andersson and some of the ideas of various strategies for building the segmentation graphs not included in previous papers can be attributed to discussions with Andersson.

The dynamic handling of the combinatorial problem of diacritic association is the main contribution of Chapter 6. Although the core algorithms and ideas as presented in [120] are fathered by the author, many aspects of the implementation in C were based on fruitful discussions with Jonas Morwing who implemented the second graph structure, the recognition graph, in the C version of the system used in the Arabic experiments.

The variation graph including the rudimentary algorithms for generative and discriminative training is the contribution of Chapter 8 and it will also be published as [119].

The C implementation used in the Arabic experiments in Chapter 9 has to a large extent been implemented by Jonas Andersson, Jonas Morwing and Christer Friberg in addition to the author. The respective parts of the implemen-

tations of the first two have already been mentioned above. Christer Friberg did most of the work in designing the segment definition database (which corresponds to the *untrained* database in the experiments). In particular Jonas Andersson has put a large effort into memory and speed optimizations of the code, resulting in the commercially highly competitive figures presented for various parameter settings in the experiments.

CHAPTER 2

Handwriting Recognition

Without words, without writing and without books there would be no history, there could be no concept of humanity.

Hermann Hesse

Despite the appearance of type-writers and other types of key-based input systems, handwriting will surely maintain its position as the most versatile way of recording text. It is much easier to adapt to various kinds of recording devices and it is independent of the script - as opposed to fixed keyboards which only work for a particular set of characters. It is also a powerful complement to the spoken word. For languages based on ideographic scripts, it is for instance common to explain the meaning of a homophone by gesturing the shape of the character meant. The historical and cultural importance goes without saying. The indisputably weakest characteristic of handwriting which has driven the development of alternative methods for recording text, such as type-writers, is the *human factor*. Handwriting is so individually specific that it is even used in forensic analysis for identification. Somewhat like English is the lingua franca of the spoken word, normalized handwriting in form of the printed character is a common written language recognized by writers independently of their specific handwriting style. This lack of ambiguity makes buttons with printed characters indeed a very powerful way to enter text into machines. At the dawn of the computer area there were also no reliable devices for capturing handwriting digitally, thereby definitely not leaving handwriting as an option. Several years of intense research rendered more reliable ways to capture handwriting, but for the past decade the presence of handwriting as an input method for machines has been limited to high-end mobile phones and other comparatively marginal devices such as PDAs and later tablet PCs. Recently, various forms of handwritten input is receiving renewed attention along with the development of more complex user interfaces and larger screens. Such visual requirements is driving a rejuvenated search towards input alternatives that are free from space consuming hardware such as keyboards.

2.1 On-line Handwriting Recognition

The setting of the handwriting recognition problem depends greatly on the device capturing the handwriting. The touch-sensitive screens in mobile phones, PDAs and tablet PCs mentioned above can produce what is referred to as *on-line* information implying that it contains the information of the movement of the writing-tool. The case where the movement is unknown but the resulting pictographic information is at hand is called *off-line* information. In the handwriting recognition community *off-line* handwriting recognition particularly in the form of OCR (*Optical Character Recognition*) dates back to the days of the first optical scanners, whereas the on-line recognition research began first when the trace of the writing (the order in which the pen moved to produce the writing) was made available through early digitizing tablets. Off-line in-

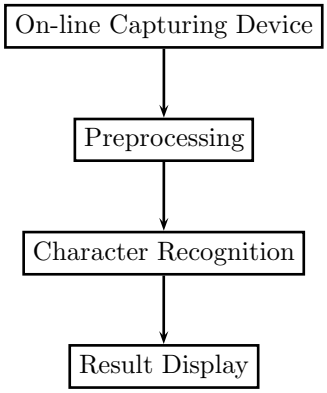


FIGURE 2.1: A schematic view of an on-line handwriting system

formation can always be produced easily from on-line information whereas the opposite is much more difficult. In this aspect a recognizer used for off-line recognition can be used for on-line recognition problems whereas the opposite is usually not true. From a mathematical point of view the off-line writing is a two-dimensional function of coordinate space $\mathbb{R}^2 \to \mathbb{R}$ producing one pixel value for every coordinate in the image, whereas on-line writing can be viewed as a one-dimensional curve $\mathbb{R} \to \mathbb{R}^2$ so that each parameter value corresponds to a coordinate. The thesis investigates the treatment of handwriting in the form of curves and consequently it is limited to *on-line* handwriting. A schematic view of an on-line handwriting system is seen in Figure 2.1.

2.1.1 On-line Parameter Space

As noted above, on-line handwriting can be seen as a one-dimensional structure since each point (x, y) on the piecewise continuous curve $\lambda(t)$ constituting the handwriting can be described by a single parameter t i.e. $\lambda : \mathbb{R} \to \mathbb{R}^2$. As mentioned above this can not be true in the off-line case since each pixel-value depends both on the vertical and horizontal position of the pixel. The parameter t above can be thought of as a time marker for the writing, where the first value corresponds to where writing was started (usually when the tip of the writing-tool meets the capturing surface) and the last value to where writing was stopped. This naturally introduces the next concept in on-line handwriting: *sampling*. Since the device capturing the handwriting is some

form of a machine it unavoidably converts the piecewise continuous handwriting curve λ into a finite set of discrete points X - it thereby *samples* the curve. In terms of the previously mentioned parameter t it can also be said that each such way of sampling the same curve is a parameterization of λ since for each corresponding $X = \{(x_s, y_s)\}, s = 1, \ldots, n$ there is a non-decreasing function $f : \mathbb{R} \to \mathbb{R}$ such that $\lambda(f(s)) = \lambda(t_s)$ simply because all points on any discrete sampling of a continuous curve also exists on the original curve.

Reparameterization Since each sample of on-line handwriting consists of a discrete set of coordinate pairs as seen above, it can also be viewed as a single point in a high dimensional space. This provides a natural way of comparing such objects by simply evaluating the distance between such points in the high dimensional space. The problem with this however is that the dimension depends on the number of sample points. For this reason it is common to reparameterize the curve into something more suitable for recognition purposes. Some of the most common methods are given below.

Definition 2.1.1. Arclength Parameterization
For a fixed number of n sample points and a curve $\gamma(t) \in \mathbb{R}^2, t \in [1, n]$ the arclength parameterization of γ has the property

$$\int_{t=j}^{t=j+b} \left| \frac{d\gamma}{dt} \right| dt = kb, k > 0, b \geq 0.$$

For this reason a discrete curve $X = (x_1, \ldots, x_n)$ is said to be parameterized by arclength if the points are uniformly distributed on the curve. The obvious weakness of the arclength parameterization strategy is that it fails to ensure correspondence between points of the same index in different samples of the same symbolic interpretation. A remedy to this problem is the versatile strategy of DTW as presented in Section 2.4.2.

For Chinese characters it is common to exploit the fact that each character often fits nicely into a square $\mathbb{S}_{[a,b,c,d]} = (x, y) \in \mathbb{R}^2 | x \in [a, b], y \in [c, d], a, b, c, d \in \mathbb{R}$. This way the character can be normalized to a fixed size and the obtained measure on relative distances can be used to perform polygonal approximation reparameterization as defined in Definition 2.1.2.

Definition 2.1.2. Polygonal Approximation Parameterization
For a fixed value η, the polygonal approximation parameterization of $\gamma \in \mathbb{S}_{[a,b,c,d]}$ has the property that there is a number n_η such that

$$\max_{i \in (2,\ldots,n_\eta)} \max_{t \in [i,i+1]} \frac{|(\gamma(t) - \gamma(i)) \cdot (\gamma(i+1) - \gamma(i))|}{\|\gamma(i+1) - \gamma(i)\|} < \eta.$$

2.2 Single Character Recognition Methods

There are various ways to classify handwriting recognition strategies and, in general, terminology suffers from inconsistent naming conventions. The naming of one system may correspond to a technique used to solve part of a problem in another thereby obfuscating the process of comparing different methods and identifying their respective merits and deficiencies. The categorization provided here is aimed at the pure shape matching techniques used within handwriting recognition systems and not the complete systems. Therefore the discussion in this section is limited to the SCR case.

In principle these methods can be divided into two main categories, also shown with subcategories in Figure 2.2.

- *Explicit Modeling* which uses a database lookup to identify the most probable recognition target.
- *Implicit Modeling* where the modeling is implicitly designed by the system through a training procedure.

From a statistical point of view all recognition methods strive at finding the optimal decision boundary in sample space. The differentiator between the two paradigms stated above can be said to be that the explicit modeling uses *a priori* information in the system design phase (i.e. when modeling the recognition targets). As stated earlier the merit of such a design is that the implementor of such a system has more control over the recognition procedure (i.e. recognition target support) but naturally it may require more work to attain the maximal *a posteriori* performance.

2.2.1 Modeling

The term explicit modeling for on-line pattern recognition here implies that the targets of recognition are defined through an observable model. In other words, each recognition target is defined through a set of properties often enabling a graphical view of each target. Recognition is thereby conducted by comparing (or *matching*) input to all such models. This in turn can be accomplished through similarity (proximity) measures, distance functions or probability calculations.

Depending on the method used for recognition models can be said to be *static* or *flexible*. Static models will from now on be called *templates* and these will be used directly through some form of a distance function to obtain a measure on how well the sample matches the template. Flexible models on the other

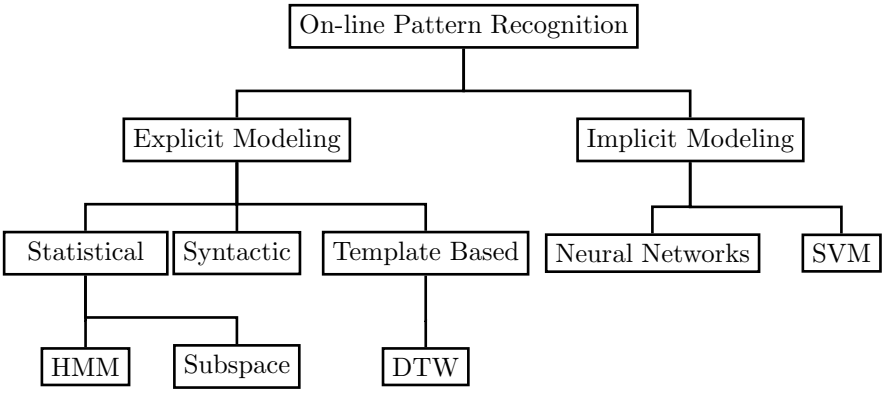


FIGURE 2.2: A classification tree of different methods used in on-line handwriting recognition

hand implicitly contain some variation and thus the comparison of sample will be made to the closest variation of the flexible model.

2.2.2 Clustering

A difficulty when comparing the explicit recognition methods is that the content of the set of models has a crucial bearing on recognition performance. Clustering algorithms are often employed to partition each target character class into submodels usually referred to as allographs. The number of such clusters, the choice of cluster representative (model) and the clustering method used to arrive at them all have bearing on the recognition results.

For the dynamic modeling methods such as those covered in Section 2.3 the clustering process is required in order to limit the shape variability modeled by a single model, since increased modeling complexity (handling larger variability) leads to reduced discriminatory power (in the form of larger parameter variance) [49]. For static template comparisons it is natural to use the template distance function also in the clustering stage, this may be more difficult for flexible models such as HMMs [64], although it is possible to use model probability in combination with the k-means algorithm [91].

For static template methods there are two fundamentally different approaches to the clustering problem, where typically the complexity (and thereby discriminatory power) of the discrimination function has determined the choice.

For naive matching functions such as simple Euclidean distance requiring large amounts of templates it is common to use template set editing schemes which aim at removing templates of little use to the discrimination process, e.g. templates which are surrounded by templates from the same class [28]. Optimally this will leave a template defined boundary around each class in sample space (also called the Voronoi tesselation [49]) and in this respect the determination of one-against-all support vectors (cf. Section 2.6.2) falls into this category. The other approach more often used in HWR under the assumption that only a few clusters will be used as templates is to use classical unsupervised classification such as the k-means algorithm [56] or some hierarchical agglomerative method [134]. Both of these methods are however generative and naturally the optimal number of clusters k or the dendrogram threshold value T can not be determined with certainty. In lack of the optimal value, many researchers resort to empirical determination of suitable values [8, 77] there are some methods that try to find a reasonable value automatically [91, 93]. It is also possible to update an initial clustering by modifying templates through LVQ [60] or by comparing surrounding classes directly with the recognition function [117, 118].

2.3 Statistical Models

The main benefit of using a statistical model for handwriting recognition is the possibility to describe the intrinsic variations in a compact manner through a set of parameters that can be inferred from a training set. A weakness of many of these models is that they are generative to their nature. In other words the modeling for each class is conducted without influence from neighboring classes unlike more genuine discriminative methods such as Neural Networks.

2.3.1 Markov Modeling

Hidden Markov Modeling (HMM) has been a popular approach to the handwriting recognition problem ever since its introduction in the late 80s. Prior to this it had been applied to speech recognition with pioneering work of L.R. Rabiner, author of several early papers in this field as well as the standard tutorial in [96]. A Hidden Markov Model $\mathcal{M}$ of a shape is defined as a sequence of hidden states $(\xi_1, \ldots, \xi_n)$ along with state transition probabilities $A = [a_{ij}]_{i,j=1}^{n}$ where a_{ij} denote the transition from state i to j, and probability density functions $(b_1, \ldots, b_n)$ such that

$$b_j(\mathfrak{f}) = p(\mathfrak{f}|\phi = \xi_j), j = 1, \ldots, n \tag{2.1}$$

corresponds to the probability of observing a feature set $\mathfrak{f}$ given that the hidden state ϕ is ξ_j. The recognition process of a sample X with HMMs is then

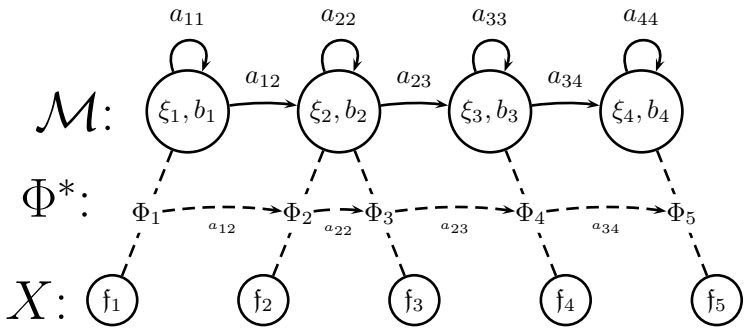


FIGURE 2.3: A graphic view of a linear Hidden Markov Model with 4 states. The most likely state sequence corresponding to the sample X with input sequence $(\mathfrak{f}_1, \ldots, \mathfrak{f}_5)$ is shown as Φ^*.

straightforward and and the class index $\mathcal{I}(X)$ of X is optimally determined according to

$$\mathcal{I}(X) = argmax_j\{p(X|\mathcal{M}_j)\}. \tag{2.2}$$

The solution to (2.2) can be obtained through the *forward-backward algorithm* [96]. However, it has been shown that $p(X|\mathcal{M}_j)$ is strongly correlated to $p(X, \Phi^*|\mathcal{M}_j)$, where Φ^* is the most likely state sequence of $\mathcal{M}_j$ given X [7]. Since the latter is effectively computed by the Viterbi algorithm [129] it is often preferred for time complexity reasons.

A common set of allowed transitions, often referred to as a linear HMM topology, are loop ($a_{ii} > 0$) and forward ($a_{i(i+1)} > 0$) transitions [61, 4, 7, 53, 68]. Topologies that prevent backward state connections are called Bakis or left-to-right models and due to the sequential nature of handwriting curves these are probably the only topologies used in HWR. There are fewer examples of topologies allowing state skipping ($a_{ij} > 0, j > i + 1$) and in these cases the skip is usually restricted to one or two states [28, 35, 107]. So in general for handwriting the transition matrix A has non-zero elements only in elements within a certain proximity (depending on the number of allowed skips) of the diagonal.

An important parameter when designing a recognition system based on HMM is the number of hidden states. Naturally a larger number of states allow a more exact description but such an increase in model complexity also require more training data in order to avoid overfitting [49]. Segmentation analysis has been proposed as one way to determine this number from training data [64].

Being a statistical model, each HMM $\mathcal{M}$ is dependent on training data. The parameter estimation of each such model (referred to as the *training*) is usu-

ally conducted with either the Baum-Welch algorithm (which is also called the EM-algorithm) or through Viterbi-training [7]. The Baum-Welch training is an iterative procedure based on the *maximum likelihood* (ML) criterion. Several researchers have recognized this as an intrinsic weakness of the HMM concept, as the correct training criterion from a discriminative perspective is the *maximum a posteriori* (MAP) criterion [14, 83]. Another weakness of first-order HMMs when applied to on-line HWR is the lack of a global perspective. Models which have similar hidden states but with slightly different transition probabilities are difficult to discriminate [4]. Hu et al. proposes an *augmented HMM system* to improve this situation by adding features that capture global characteristics of the shape [53].

In the beginning of the 90s HMM was one of the most exciting methods for HWR and some early papers reported that HMMs were able to produce better recognition results than DTW (cf. Section 2.4) for instance [11]. A comparison of state-of-the-art methods on a benchmark set of on-line digits, however, provides no support stating that HMM based systems would provide higher recognition accuracies than Neural Networks [97] or even modern variations of DTW [3, 8, 77]. Instead the merits of HMMs are their intrinsically sequential nature [28] rendering extensions to recognition of connected sequences of characters straightforward [4, 53, 108]. The generative nature of the HMM training (using the ML criterion) has inspired many to replace the shape recognition component by Neural Networks although maintaining the HMMs to combine partial shape recognition results [14, 22, 101, 55].

Conditional Random Fields (CRF) A recent interesting and well-cited development which reduces some of the known problems with HMMs are Conditional Random Fields [62]. CRFs are conditional models that model the conditional probability distribution of a label sequence given the observation sequence directly instead of modeling joint distribution of observation and label. There are still few publications of applying CRFs to the problem of HWR but some encouraging initial results are provided in [34] showing the increased discrimination power of modeling.

2.3.2 Subspace Based Recognition

Principal Component Analysis (PCA) is a popular method for describing shape variation. Basically this method sort the features of sample space according to prevalence through eigenvalue decomposition. By fixing the number of Eigenvalues N used for each character class k, the explicit models $\mathcal{M}_k$ can thus be defined as a subspace in the space $\mathbb{X}_k$ of all samples with class k

as

$$\mathcal{M}_k = u_1, \ldots, u_N \text{ and } \lambda_j u_j = C(\mathbb{X}, \mathbb{X})u_j, \tag{2.3}$$

where λ_j is the jth largest eigenvalue and u_j the corresponding eigenvector to the covariance matrix $C(\mathbb{X}, \mathbb{X})$. An intuitive way to perform recognition is then to find the best approximating model for a sample X as the model with the smallest orthogonal distance to the sample. It has however been observed that the orthogonal distance is insufficient for discriminating in sample space and one remedy has been to add a distance component to the outer bounds of the distribution of the samples in subspace [32].

Active Shape A method closely related to PCA for modeling shape variation is Active Shape as introduced by Cootes et al [51]. In Active Shape eigenvalue decomposition is performed on an estimate of the covariance matrix S_k of the samples $X \in \mathbb{X}_k$.

An Active Shape Model (ASM) $\mathcal{M} = (\mu, U)$ is then defined as the mean shape μ along with the subspace spanned by N eigenvectors $U = \{U_j\}_{j=1}^{N}$ of S_k and a set of constraints $\vec{\sigma}_0 = (\sigma_{10}, \ldots, \sigma_{N0})$ on the variations corresponding to how much the model can be extended in the direction of a certain base (eigenvector) of the subspace. Usually the constraints are given as a number of standard deviations. An intuitive way to use an Active Shape Model in recognition is then to find the closest variation to sample in the constrained subspace by searching for the parameters $\vec{b} = (b_1, \ldots, b_N)$ in the constrained volume of $\mathbb{R}^N$ bounded by $\vec{\sigma}_0$ realizing

$$\vec{b}^* = \underset{\vec{b}, |b| \leq \sigma_0}{\operatorname{argmin}} \|X - \mu - U\vec{b}\|. \tag{2.4}$$

Recognition can then be achieved simply by choosing the class j with model M_j fulfilling $\min_j d(\mathcal{M}_j, X) = \min_j \|X - \mu_j - U_j\vec{b}_j^*\|$. It is interesting to note that this distance function corresponds well to the modified PCA distance proposed in [32].

An important weakness of all PCA based methods is the alignment problem. As will be shown later in Section 3.2.4, eigenvalue decomposition on arbitrarily sampled characters often incorporate parameterization variations causing the subspace of eigenvectors to include distorsions of some salient features. The two applications of ASM found in the literature attacked this problem from two directions. Mitoma et al. aligned samples by DTW (cf. Section 2.4) prior to the Active Shape analysis for on-line digit recognition [77]. Sridhar et al. performed the same operations in opposite order, in other words, started to find the best

set of parameters and then used DTW as the distance function between sample and the resulting model in on-line Tamil recognition [110]. Shi et al. have used Active Shape Models together with an image distance transform to search for radicals in images (off-line) of Chinese handwritten characters [105].

Curve Space Invariants Another interesting method which is not based on PCA but shares strategy of working with curve space is the invariant technology proposed by Berthilsson et al. [15]. In very simplified terms, the idea behind this method is to define each allograph model as a curve space, invariant to a certain set of transformations (e.g. *positive similarity transformations*). As for the PCA-based methods the projection to the curve space of the model is calculated. Recognition is then enabled by a *proximity distance* μ evaluating how similar the two spaces are through the Hilbert-Schmidt norm

$$\mu = \|P_{\mathcal{M}} Q_X\|_{HS}, \tag{2.5}$$

where $P_{\mathcal{M}}$ and Q_X denote the orthogonal projections to the curve space of model $\mathcal{M}$ and sample space of X respectively.

Lately, this technology has also successfully been employed for the recognition of Cyrillic [31] and Arabic scripts [10].

2.4 Template Based Methods

Given previous explicit modeling methods it is here necessary to clarify the meaning of *Template Based* as used in this thesis. Unlike the flexible statistical models a template is a static representation of a character in sample space. Template based methods are very intuitive and simple in this sense since all that is required is a distance function in sample space $\mathbb{X}$ to construct a classifier. The problem with handwriting recognition however is that the sample space is a strange space which is not only of high dimension but which varies in dimensionality from sample to sample. Traditional template based matching methods either employ a distance function that can compare objects of different dimensions or try to fix the dimensionality of input in some way. Usually some combination of the two strategies are used in a template matching method. Another problem is that the high-dimensional points may be parameterized in different ways thereby failing to assert that the dimensions actually correspond to each other.

2.4.1 The Database

One of the major merits of the template based methods is the transparency of the recognition process. The templates themselves are actual entities in sample space and recognition errors can often be explained through insufficient template coverage. In fact there are even studies showing that this renders errors made by template based systems more intuitive than those made by implicit modeling methods such as Neural Networks [84]. Another merit is that the static nature of these systems enable dynamic modification of template content, making this approach popular in adaptive systems [36, 132].

2.4.2 Dynamic Programming

A common way for static template matching methods to incorporate flexibility is by using Dynamic Programming (DP). This technique allows for matching samples of different dimensions as well as dynamically determining alignment even between samples in the same space. Dynamic Programming appears in the on-line handwriting literature under several names such as *elastic matching* [124, 136], *string matching* [29] and Dynamic Time Warping (DTW) [3, 8, 77, 130]. Basically all of these methods share the algorithmic method of the Viterbi [129] algorithm which in turn uses the same strategy as shortest path algorithms such as the Dijkstra algorithm [33].

Dynamic Programming strategies differentiate the static template matching by introducing a point-wise distance function $d : \mathfrak{F} \times \mathfrak{F} \to \mathbb{R}$ so that each point p (or corresponding feature $\mathfrak{f}$) can be matched individually instead of statically assigning alignment by index in the parameterization. Vuori has investigated several such point distance functions in [130] but recently there seems to be some consensus that best discrimination is achieved with the simple Euclidean norm acting on the feature points $\mathfrak{f} = (x, y, \theta)$ consisting of the point coordinates $p = (x, y)$ complemented by the angle to the next point $\theta = \arg(p_{i+1} - p_i)$. Matching is then performed by dynamically finding the correspondence function $\Phi = (\phi_X, \phi_Z) : \{1, \ldots, N\} \to \{1, \ldots, |X|\} \times \{1, \ldots, |Z|\}$, also referred to as an alignment, between two samples $X = (p_1^X, \ldots, p_{|X|}^X)$ and $Z = (p_1^Z, \ldots, p_{|Z|}^Z)$ that minimizes the sum of the distances. This alignment is obtained implicitly by minimizing the distance function:

$$\min_{\Phi} D_\Phi(X, Z) = \frac{1}{C} \sum_{n=1}^{N} d(p^X_{\phi_X(n)}, p^Z_{\phi_Z(n)}), \tag{2.6}$$

where $\Phi(1) = (1, 1), \Phi(N) = (|X|, |Z|)$ and C a normalization constant usually equal to N. The *Sakoe-Chiba* transitions are defined as $\Upsilon = \{(1, 0), (0, 1), (1, 1)\}$

[8]. A DP function based on these transitions imply that $\Delta\Phi = \Phi(n+1) - \Phi(n) \in \Upsilon, n \in (1, \ldots, N-1)$.

Apart from the obvious time complexity issues of quadratic programming when running recognition on a very large number of classes as in [100], there are some issues caused by the dynamic programming. One problem is that the algorithm takes no account of the way characters differ thus favoring character shapes with a *mean* appearance such as straight strokes. A more powerful discrimination can therefore be obtained by limiting the use of DP to the alignment problem and using other methods specialized at discriminating between aligned samples [77].

2.5 Syntactic

The syntactic modeling concept relies on the philosophy that handwriting are observations of an ideal representation of characters and that recognition is performed with reference to such instances. In this respect the syntactic approach represents the case where the system designer has most control over the recognition process by manually designing syntactic rules for the interpretation results of given shapes. One of the large merits of such a system is thereby that only limited or no training data (depending on if properties of some rules are deduced from statistical analysis) is required for such methods. The human interaction in handwriting infallibly produces variations difficult to model with harsh rules. For this reason syntactic systems found in the literature employ some type of fuzzy-shape grammars to avoid harsh and premature exclusion of recognition candidates [5, 89].

Parizeau et al. have developed a system where each syntactic model is manually generated as a set of primitive curve segments with syntactic descriptors in the form of fuzzy rules. Recognition is then conducted by using rules associated with each primitive and each property to calculate the degree of membership of the sample X to these allograph parts [89]. Anquetil et al. use a similar process but use a decision function based on a sum-product so that the particular output of each rule is normalized by the total sum of the products of all membership functions for all features [5].

2.6 Implicit Modeling

In implicit modeling, class boundaries can be determined a posteriori but the actual properties of each recognition target is never explicitly defined (in the form of a tangible class representative as in the explicit modeling methods de-

scribed in above). A clear advantage of this modeling is the focus on discrimination when determining class boundaries instead of the generative modeling perspective common for explicit modeling methods such as HMM. Judging by the number of publications during the past decade these methods are by far the most popular to use for the on-line handwriting recognition problem. One of the most striking arguments for using such methods apart from the quantitative results in the form of high recognition accuracy, is the generality of these methods. In particular Neural Networks (Convolutional Nets) is an extremely versatile strategy for generic object recognition [63] and it is naturally very favorable if all that needs to be done for a recognition task is to find training data. Unfortunately this is also one of the major flaws of Neural Networks as they usually require vast amounts of data to ensure that the system avoids overfitting [106].

2.6.1 Neural Networks

Neural Network methods are among the most common methods used for pattern recognition of images. Their original principal idea is a simplified model of the human nerve system, where inputs pass through a system of nodes which, in analogy with the human perception system, often are referred to as neurons. From a statistical point of view a neural network corresponds to a nonlinear statistical model [49]. Due to the intrinsic facility in which the system can be trained for various recognition tasks it has also become one of the most popular methods for on-line character recognition and certainly the most popular for the off-line case [55, 74, 101, 104]. For the off-line case it has also been shown to produce the best recognition results on the MNIST benchmark database [106]. This research area has matured significantly in the past decade and there are several extensive and generic treatments of Neural Networks in the literature [17, 99, 49]. A neural network consists of nodes and connections. Each node has a weight and an activation function, usually the sigmoid $\sigma(v) = \frac{1}{1+e^{-v}}$. The activation of each node is calculated by feeding the weighted linear combination of the activations of connected nodes in the previous layer into the activation function. A basic system will have three layers: an input layer where each node is activated by a certain feature in the input sequence, a hidden layer and an output layer with one node for every symbol or every state of a symbol that should be recognized, see Figure 2.4.

Thus activation f_{hk} in a node with index k in the hidden layer of the network in Figure 2.4 for instance can be written as

$$f_{hk} = \sigma(w_{hk0} + \sum_{j \in E_k} w_{hkj} \mathfrak{f}_j), \tag{2.7}$$

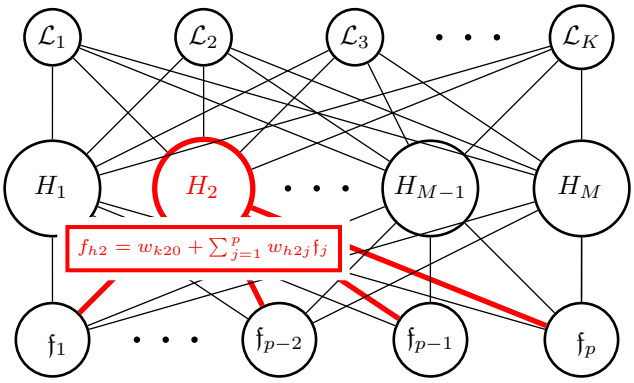


FIGURE 2.4: A graphic view of a typical three layer network with an input layer activated by p feature functions, a hidden layer with M nodes, and an output where activation in each of the K output nodes corresponds to a probability for a certain symbol being matched.

where E_k denote the indexes of the connected nodes in input layer and w_{hkj} is the weight of that particular feature connection.

Training of a neural networks thus corresponds to estimating the parameters w for every layer. This is normally conducted by a gradient descent method called back-propagation [55, 104] (cf. [49]).

TDNN The main neural approach applied to the on-line character recognition problem so far is the Time Delay Neural Network (TDNN) [22, 55, 101, 104] although there are some recent results also with more basic networks such as the multi-layer perceptron (MLP) [86]. The TDNN is a multi-layer feed-forward architecture that has been especially successful in learning to recognize time sequences such as those appearing in on-line HWR and speech recognition systems. Typically a TDNN has an input frame treating a time-slice of the input data feeding into one or more hidden layers that eventually terminates in an output layer with one output per symbol in the character set used for the cursive handwriting sample. Moving the input frame along the sample will thus generate a sequence of observations which can be interpreted and compared to a dictionary to retrieve plausible word recognition candidates. HMM is a popular choice of method to aid in this process [55, 101].

Kohonen Maps Another interesting type of network with desirable discrimination properties are Kohonen Self-Organizing Maps (SOM) [59], which in a way are network extensions of the learning rules of LVQ [60]. The Kohonen Self-Organizing Map can be seen as a matrix of detectors specialized at recognizing the input of some feature. It is trained by gradually making the detector $m_c(t)$ most similar to an input vector $x(t)$ even more similar by applying a learning rule of type:

$$m_c(t+1) = m_c(t) + \alpha(t)d(x(t), m_c(t)). \tag{2.8}$$

The learning rule is applied to all $x(t)$ in a neighborhood $N_c(t)$ of $m_c(t)$. $\alpha(t)$ is a monotonically decreasing sequence of learning factors. When training is completed the detectors will be organized spatially to reflect the topography of the training space.

Some experiments have been conducted using Kohonen maps directly as *databases* for on-line recognition [76, 78]. Essentially this corresponds to a template based nearest neighbor recognition, but where templates are adaptively trained so that the resulting template (node in the SOM) corresponds to a synthetic sample consisting of a linear combination of samples in the training set. SOM has previously been used in the feature extraction stage in order to implicitly define the set of features used in recognition [58]. It has also been implemented with the function of sharing the emission probabilities between HMMs (cf. $\{b_j\}$ in Section 2.3.1) and also to improve recognition accuracy through a simple additional feature mapping (nearest neighbor style as above) [43]. In another experiment self-organizing maps were used to identify inter-writer similarities [131].

2.6.2 SVM

A pattern recognition method that has received a lot of focus in late years is Support Vector Machines (SVM), introduced by Vapnik [126]. Support vector machines are originally a genuine two-class discriminator that approximates a boundary (separating hyperplane) between two classes by finding relevant boundary samples called support vectors. In its simplest form the search for the maximal margin hyperplane is conducted in linear space but often a kernel can be used to map linearly inseparable data into another favorable space. With such a kernel the SVM classifier with N support vectors $\{Y_j\}^N$ with labels $\{\mathcal{L}\}^N, \mathcal{L} \in [-1, 1]$ can be written as in (2.9):

$$f(X) = \text{sign}(\sum_{i=1}^{N} \alpha_i \mathcal{L}_i K(Y_i, X) + b), \tag{2.9}$$

where K is the kernel, α weights and b an offset. Training of the SVM classifier corresponds to finding the maximum separating hyperplane and thus the set $(Y_i, \mathcal{L}_i, \alpha_i)$ which involves solving of a quadratic programming problem with equality and inequality constraints. There are several publicly available packages for doing this, like libSVM.

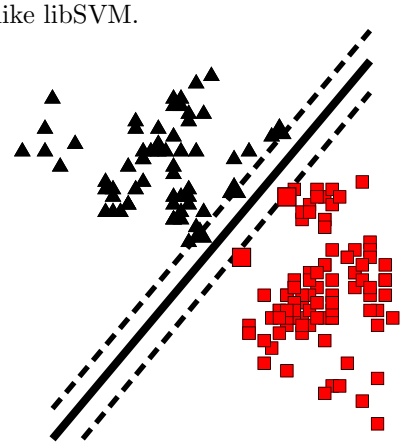


FIGURE 2.5: A graphic view of a two-class problem with separating hyperplane (solid line) and support vectors (larger markers on dashed line).

One of the major problems with applying support vector machines to recognition of on-line HWR is the intrinsic two-class nature of the SVM. A popular method for coping with this in on-line HWR is by voting schemes applied to multiple two-class discriminators [1]. Another strategy is to retain the genuine two-class application of SVM by applying these as a second stage classifier for solving common confusion errors made by a genuine multi-class method such as the template based methods described earlier [31, 46, 135]. In another experiment Bahlmann et al. use a conventional dynamic programming distance function (cf. Section 2.4.2) as a kernel, but somewhat surprisingly this approach does not seem to produce better results than a genuine multi-class DP-based classifier even in the two-class discrimination case [9].

2.7 Combination Methods

A logical continuation to the type of recognizer analysis included above is to try to exploit the differences in a constructive manner by combining different methods. Conceptually such an approach is very attractive and, to use a popular description, in an abstract way it corresponds to the situation of relying on a panel of *multiple experts* instead of being dependent on the decision by a sole

judge. In practice, it has also proven that this analogy can be extended further. Since it has been difficult to produce reliable ways to combine the actual confidence values of different classifiers, voting methods using the candidate lists (or rank) produced by the classifiers are often an effective combination strategy [52]. It has been used successfully to combine on-line and off-line systems in [69] and in a more comprehensive combination method comparison by Alpaydin et al. it produces the highest recognition accuracy for writer independent tests [2].

Another paradigm for combining methods is the *multi-stage* strategy which is more of a serial approach where the next classifier is consulted after treatment by the first. This can work in two ways, either by (1) only consulting the next method upon rejection by the first (cascading) or (2) by allowing each method to narrow the search as in a coarse-to-fine search. The latter is an important step in recognition of scripts with large character sets such as e.g. Asian scripts [66] and can also be used analogously for dynamic lexicon reduction in cursive word recognition for western scripts [23, 94, 104]. One type of fine search is to discriminate between top candidates from a first recognizer (also called a discrimination zoom) [31]. Another interesting approach is *boosting* (here AdaBoost) where a composite classifier is constructed by iteratively training classifiers to correct errors made by the current composition [42]. Although improvements have been made to simple Neural Networks [102] there are few published results on boosting applied to on-line handwriting recognition.

CHAPTER 3

Preprocessing And Segmentation

By failing to prepare, you are preparing to fail.
Benjamin Franklin

THIS CHAPTER TREATS various preprocessing and segmentation methods. Popular preprocessing methods include various forms of normalization in scale, position and variance as well as curve smoothing and other resampling schemes. The latter were especially important in the past when digitally sampled data contained a significant amount of noise [54, 90, 125]. For the system presented in this thesis, most conventional preprocessing steps can be incorporated into the recognition algorithms and therefore focus here lies on the segmentation techniques. Many segmentation techniques however often make the same assumptions on input data and thus they have similar flaws.

3.1 Introduction

Various recognition methods have a varying dependency upon the preprocessing step and this is yet another factor that adds to the difficulty in comparing recognition methods on a given data set. In general preprocessing techniques imply that some restrictions on the expected structure of the input data are imposed with the merit of reducing the required modeling complexity. Preprocessing and normalization is traditionally also intricately dependent on the script to be recognized [54]. For Asian scripts, for instance, the square shape of the characters can be exploited to reduce variance and fix scale, which in turn enables special resampling schemes. A common such method is the recursive polygonal approximation [66] (cf. Section 2.1.1) seen in Figure 3.3 . For most other scripts however the shape of the input depend on the written word and the constituent characters. For these methods it is common to normalize by some kind of helplines as seen in Figure 3.1, often inferred from the writing [54, 104] or assumed from the writing user interface [36].

This chapter will discuss some of the most common preprocessing methods and what requirements would be added to the recognition algorithms if they were to be skipped. There are quite a few strategies for segmenting on-line cursive script [6, 19, 25, 85, 88, 111] and since segmentation of input is a required step for the algorithms treated in this thesis some examples of implementations and modifications will also be covered. In principle however the standpoint of this thesis is that no segmentation method is flawless! Therefore it is crucial also to have strategies to cope with missing or extra segmentation points. The difficulty in producing reliable segmentation points are also a factor that has boosted the popularity of methods with implicit segmentation. Implicit segmentation can be realized through a sliding window of a time-delay neural network [21, 55, 101] or through a sequence of frames with Hidden Markov Models [68, 98, 108].

Segmentation can also be interpreted as a structurally dependent shape analy-

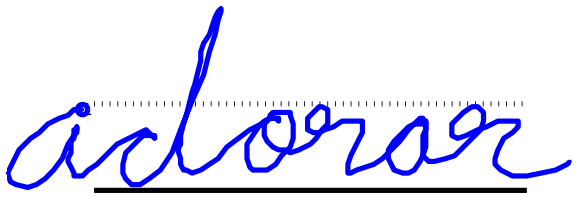


FIGURE 3.1: An on-line sample of the cursive word *adorar* segmented at y-extrema, plotted with estimated base- (solid) and helplines (dashed).

sis. It is well-known that the dimensionality of a curve can be reduced without loss of significant morphological information by encoding structural content (i.e. curvature) [75]. For this reason points of high curvature are also often the focus of segmentation algorithms as well as feature point extraction [6, 36, 47, 85] It will be shown that this fact can be exploited to tailor special features for segmented input.

3.2 Segmentation

The problem of segmentation can be seen as a type of unsupervised classification. Given an input sample X of a set of strokes $X^1, \ldots, X^n$ each stroke shall be analyzed and possibly divided into smaller parts, conventionally referred to as *segments*, according to certain characteristics. Extending the unsupervised classification analogy further, the segmentation problem can be described as the problem of clustering points on each stroke into an unknown number of compact clusters distinctly separated in time. Since the desired number of clusters is unknown an applicable such segmentation method is Hierarchical Agglomerative Clustering (cf. [56] and Section 2.2.2). In Hierarchical clustering clusters are determined through a threshold in the inter-class dendrogram. Using this

terminology, the segmentation problem can be described as the problem of finding a cluster distance function $d(c_i, c_j)$ between pairs of clusters c_i, c_j (sets of points) and a suitable threshold T. The set of clusters $C = c_1, \ldots, c_n$ can then by defined as the minimal set of subsets of $X = \{p_1, \ldots, p_{|X|}\}$ satisfying (3.1).

$$\max_{c_i, c_j \in C} d(c_i, c_j) < T. \tag{3.1}$$

The border between each such one dimensional cluster $c_i = \{p_{ij}\}_j$ will be exactly the set of segmentation points. An example of this description with a commonly used segmentation method (vertical extreme point segmentation) is given in Example 3.2.1.

Example 3.2.1. *Vertical Extreme Point Segmentation. Define a cluster distance function by*

$$d(c_i, c_j) = \begin{cases} \infty, & \text{if } |i - j| > 1 \\ 0, & \text{if } c_i = p_{i_1}, c_j = p_{j_1} \\ 0, & \text{if } \operatorname{sign}(p_{ik} - p_{jl}) = \kappa, \forall k, l, \end{cases}$$

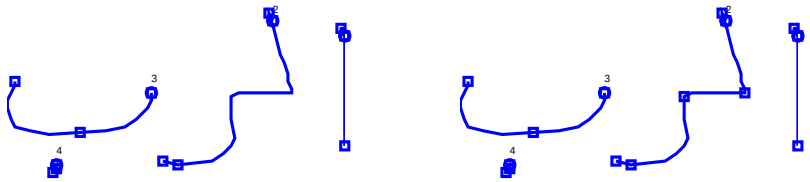
and define an arbitrary threshold $T \in \mathbb{R}^+$, then this corresponds to segmentation by local vertical extreme points. This function can be extended to handle other and more flexible versions of extreme point segmentation.

The notation $\mathcal{S}(X^i) = \{p_j\}$ will be used for the set of segmentation points of stroke X^i and thus $\mathcal{S}(X^i) \subset X^i$. Correspondingly $\{\Lambda_j\}$ will denote the segments in between which are one fewer than the segmentation points.

Definition 3.2.1. Two segmentations $\mathcal{S}(X), \mathcal{S}(Y)$ are said to be similar $\mathcal{S}(X) \sim \mathcal{S}(Y)$ if they have the same number of strokes and the same number of segments for each stroke.

3.2.1 Script Dependent Segmentation

It has been observed by several researchers in the past that local extreme points orthogonal to the writing direction can be used reliably for segmentation [36, 65] as seen in Figure 3.9a. This coarse approach will however generally miss a number of segmentation points even for natural variations in input. Furthermore it is flawed by the fact that it is dependent both on a reliable estimate of the writing direction (which can even vary within a handwritten word). Nevertheless this type of script dependent segmentation method can be used to evaluate other parts of a segmentation based system. Even very simple methods for complementing insufficient segmentation routines such as *augmentation*



(a) Y segmentation (b) Extra segmentation

FIGURE 3.2: An example of an Arabic word (الرب) where segmentation only in vertical extremes is insufficient. Heuristic rules as in (3.2) is a simple but sometimes effective way of adding more segmentation points. Segmentation points in both figures are marked with squares.

points in [85] or *splitting* points in [36] has been applied successfully in the past. Modifications to the extreme point approach mentioned above to produce a segmentation method for the Arabic script, also used in the recognition experiments of Section 9.4 is described below.

3.2.2 Segmentation Of Arabic Script

The starting point for the segmentation scheme is the same as described in Example 3.2.1 segmenting input at the extreme points in the vertical direction. The aim of the segmentation procedure is to divide input into segments containing at most the shape of one individual character. For Arabic cursive script there are several cases when the primitive strategy above is insufficient as seen in Figure 3.2a. For the system used in the experiments with Arabic cursive recognition presented in Chapter 9, a set of simple heuristic rules has been added to trigger additional segmentation points on a segment Λ consisting of n sample points $p_1, \ldots, p_n \in \mathbb{R}^2$. Heuristic rules are generally not very robust but segmentation is not the focus in this thesis and good recognition results are achieved despite this fact. Each rule consists of a set of rudimentary comparisons with ad hoc threshold values $\{T_j\}$ as in (3.2), and the result of the implemented rules on the input in Figure 3.2a is displayed in Figure 3.2b.

$$\text{For a segment } \Lambda = (p_1, \ldots, p_n), \text{ split in}$$
$$p_e, p_f \text{ if } \begin{cases} \frac{\Delta(p_i)_y}{\Delta(p_i)_x} > T_1, i = 1, \ldots, e \\ \frac{\Delta(p_i)_y}{\Delta(p_i)_x} < T_0, i = e+1, \ldots, f \\ \frac{\Delta(p_i)_y}{\Delta(p_i)_x} > T_2, i = f+1, \ldots, n. \end{cases} \tag{3.2}$$

Another general problem with the simple segmentation at vertical extreme points is that the placement of such points shows large variance in horizontal placement when they are put on smooth arcs such as the "bottom" of the letter *u*. To improve this situation an energy equation can be introduced for vertical minima points as in (3.3). This energy model will be evaluated for all occurrences of the segmentation points corresponding to local minima in y that are bordered by local maxima on both sides, i.e. points that are located in a visual "valley". The main task is to try to center these points in the middle of the valley (middle in terms of x w.r.t. the surrounding y-maxima points) and this is accomplished by modeling a *spring* pulling the point towards the center in x. Since the centering should be applied only for weak slants a counteracting force is modeled by a *gravity force* F_g pulling the point down towards the y-minima. Finally a third force component is added which is related to the curvature property of the y-minima point. The main aim of this force component is to maintain the location of strong feature points corresponding to very sharp turns in the y-direction and it is modeled as a spring with a spring constant proportional to the square of the curvature.

Two things are done to determine where a y-minima segmentation point should be moved:

1. Check if the y-minimum constitutes a point of local (w.r.t. scale) energy equilibrium. If this is true, the point remains in its original location.
2. If the y-minimum is is not a local energy equilibrium point then it is moved to the global energy equilibrium.

The energy $\Omega(p)$ of a given point $p = (p_x, p_y) \in \mathbb{R}^2$ on the curve segment $\Lambda = (p_1, \ldots, p_n)$ consisting of the discrete points between the surrounding y-maxima points (p_0, p_{n+1}) is given by:

$$\Omega(p) \propto R(p_y - p_y^*) + \phi(p_y - p_y^*)^2 + (p_x - \overline{x})^2, \tag{3.3}$$

where $p^* = (p_x^*, p_y^*) = \text{argmin}_{p \in \Lambda} p_y$ and $\overline{x} = (p_{0x} + p_{(n+1)x})/2$. Here R should be seen as the ratio of the gravity force pulling the point p toward p^* and the spring constant of the last term. This spring constant has thus been removed

from (3.3). The middle component is also conceptualized as a spring pulling the segmentation point towards p^*, where the value of the spring constant is related to $\phi = \kappa(p^*)/\pi$, where $\kappa : \mathbb{R}^2 \to \mathbb{R}$ is the curvature of p^* measured in radians.

The y-minimum point p^* is defined to be a ϵ-local minimum if the following statements are true:

$$\begin{aligned} \Omega(p^*_{\epsilon-}) &> \Omega(p^*) \\ \Omega(p^*_{\epsilon+}) &> \Omega(p^*), \end{aligned}$$

where $p^*_{\epsilon+}$ is the prior point on the curve at ϵ distance and $p^*_{\epsilon-}$ the corresponding subsequent point.

Consequently the rule defined above for placing the segmentation point p between two y-maxima segmentation points can be written as

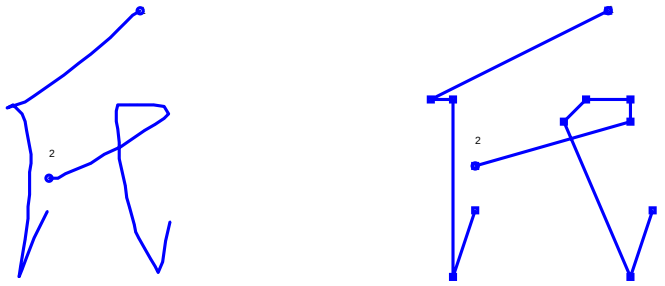
$$p = \begin{cases} p^*, & \text{if } p^* \text{ is a } \epsilon\text{-local minimum of } \Omega \\ \operatorname{argmin}_{p \in \Lambda} \Omega(p), & \text{the global minimum of } \Omega \end{cases}. \tag{3.4}$$

3.2.3 Generic Segmentation

Generally the type of heuristic rules applied in Section 3.2.2 will be unreliable as it is virtually impossible to foresee all input variations of handwritten script. The dependency on writing direction estimation also introduces limitations on the system as well as a new source of error. A purely shape dependent segmentation technique closer to the piece-wise polygonal approximation methods used for sampling Chinese characters in [80] would therefore be a more robust solution to the segmentation problem. The result of typical polygonal approximation for a Chinese character is shown in Figure 3.3. Such techniques have also been developed to retrieve important feature points in on-line signature verification [19, 47]. The rest of the segmentation based techniques presented in this thesis will work independently on the choice of segmentation method and consequently the quest for the ultimate segmentation method has been left for future research.

3.2.4 Segmented Shape Analysis

In addition to being a necessary component in recognition of connected script, segmentation also unleashes new possibilities for shape analysis. The nature of



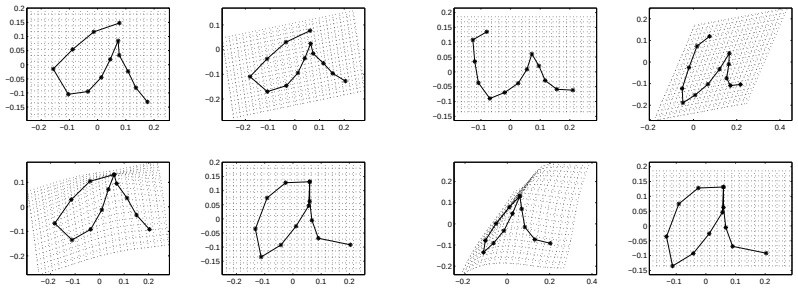
(a) Original sampling (b) Polygonal Approximation

FIGURE 3.3: An example of segmentation/resampling method for chinese characters that utilizes the intrinsic square property of chinese characters.

handwriting is such that variations of samples portraying the same set of symbols are highly non-linear and irregular. In terms of the sampling discussion in Section 2.1.1 it is clear that a given index of a point in one sample will not correspond to the same index in another sample. This is called the *alignment* problem. As previously explained, successful recognition methods such as the popular strategies covered in Section 2.2 all include some form of implicit remedy for this discrepancy. Wakahara et al. attack the problem by introducing local approximations into a global transformation [136]. Segmentation techniques add another layer of structurally motivated alignment as defined by the segmentation function in (3.1).

The aim of a segmentation is to identify points in an input that correspond to a certain characteristic such as a transition between two letters. In some respect segmentation points should therefore be consistent in this functional respect for samples portraying natural variations of the same shape. In this section it will be shown that it can make sense to separate variations in the structure of the segmentation points to the variations of the curves in between the segmentation points. For this reason the set of segmentation points will often be referred to as the *frame* of a handwritten sample or a template.

Figure 3.4 depicts the thin-plate spline transformation of inter-class and intra-class segmentation points. After the thin-plate spline has been applied, the curve shape differences between the corresponding segments are much less complex. This gives some visual support for the notion that the non-linear components of handwritten character variations could be treated separately by some distance measure on the corresponding frame transformation. Although

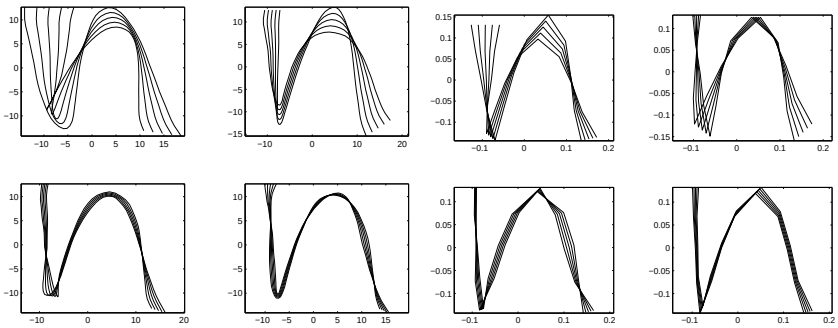


(a) An in-class example of bending an a to another.

(b) An inter-class example of bending a u to an a.

FIGURE 3.4: Bending the frame while leaving the parameterization fixed. The four figures in both cases display the original sample, the affine approximation to the target sample, the thin plate spline of the frame w.r.t the target frame and finally the target sample.

successfully applied to the field of HWR before [12], thin-plate splines are probably not a good alternative for modeling this energy since folding frequently occurs [38], which can be seen in the lower left plot in Figure 3.4b.



(a) Original parameterization.

(b) Segmentation based parameterization.

FIGURE 3.5: The first four components of principal component analysis of one segmented model of the letter n. Each plot in the figures above display $\mu \pm n\sigma U_j, n = 1, 2$, where μ is the mean shape and U_j the jth PC eigenvector and σ the standard deviation.

Using Segmentation For Reparameterization

As noted in Section 2.1.1 the terms sampling and parameterization are often used interchangeably when discussing on-line handwriting. Given samples that are samples of a given set of connected smooth curve segments it is therefore not certain that a parameter, i.e. the discrete point index, corresponds to a point on the same curve part. This in turn causes strange artefacts when analyzing variance of a set of samples portraying the same curve. In particular the popular shape variance analysis method of PCA may deform the discontinuous curve connections when applied to samples of just slightly varying such parametric alignment. This may cause the effect seen in Figure 3.5 where the first PC-component contains rounded features in samples of 'n' never observed in data. Since segmentation techniques in general try to extract such points corresponding to discontinuities in otherwise smooth curves, keeping segmentation points in correspondence during alignment should with this hypothesis guarantee the preservation of these important structural features. This partial forced alignment technique is here referred to as *segmentation based reparameterization*.

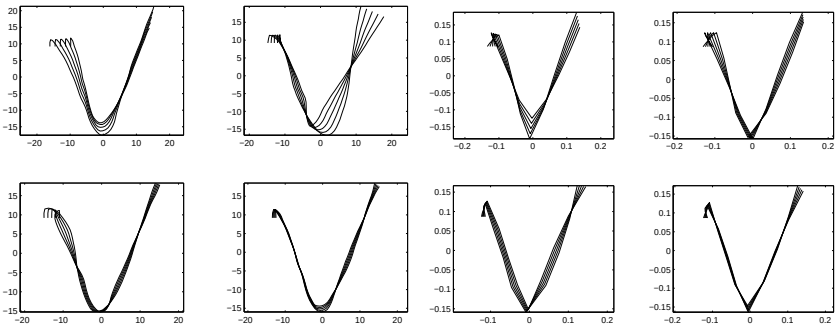
The results of the first modes of PCA with this parameterization compared to arclength parameterization are shown in Figures 3.5 and 3.6. It clearly shows that the reparameterization presented here aligns samples in a way that improves the point to point correspondences and better preserves structural features such as discontinuities in smooth curve segments.

Parameterization Of Segments

As for unsegmented handwritten input, the most basic approach for sampling (or reparameterizing) a segment of a curve is to sample by arclength as described in Definition 2.1.1. Recall that the weakness of this method is that segments may require a varied number of points in order to be described correctly under the constraint of minimizing the number of points. This basic method will be referred to as the *Segment Arclength* (SA) method.

Aiming at enabling an upper bound for the required number of points on each segment, methods that try to approximate the segment by a few number of points have also been investigated below. In this case the segment arclength method is not recommendable since crucial shape information such as curvature may be randomly lost due to the placement of the interval of points.

Choosing the n points on a piece of a curve that *best* approximates it is an interesting problem that has been thoroughly studied in the field of discrete geometry [45, 127]. There it is common to refer to the best approximation in



(a) Original parameterization. (b) Segmentation based parameterization.

FIGURE 3.6: The first four components of principal component analysis of one allograph of *v*.Each plot in the figures above display $\mu \pm n\sigma U_j, n = 1, 2$, where μ is the mean shape and U_j the jth PC eigenvector and σ the standard deviation.

terms of the uniform metric i.e. the n points on the (discrete) curve $X = \{p_i\}_{i=0}^{m}$ realizing (3.5). Let $\mathbb{F}$ be the set of monotone increasing functions $f : \mathbb{Z} \to \mathbb{Z}$ such that $f(0) = 0$ and $f(n) = m$ for $f \in \mathbb{F}$.

$$Q = \min_{f \in \mathbb{F}} \max_{\substack{f(j-1) \le i \le f(j) \\ 1 \le j \le n}} d_{PL}(p_i, L_{p_{f(j-1)}, p_{f(j)}}), \tag{3.5}$$

where $d_{PL}(p_i, L_{p_{f(j-1)}, p_{f(j)}}$ denotes the Point-to-Line distance from the point p_i to the line segment $L_{p_{f(j-1)}, p_{f(j)}}$ defined by the points $(p_{f(j-1)}, p_{f(j)})$. In accordance with the literature in discrete geometry the problem of minimizing Q in (3.5) will be referred to as the min-ϵ problem.

The algorithms developed for solving (3.5) found in publications are generally focused on speeding up the process of recursively finding the largest distances d_{PL}. The following section presents a fundamentally different approach for n-point approximation of an m-point polygon. Instead of (3.5) the problem of finding the maximum value of the segment length function is considered.

Definition 3.2.2. The *linear segment length* function of order n of a curve segment $\gamma(t), t \in [0, 1]$ is the function $\Gamma_\gamma^n : \mathbb{Z}^n \to \mathbb{R}$ such that

$$\Gamma_\gamma^n(a) = d(\gamma(0), \gamma(a_1)) + d(\gamma(a_n), \gamma(1)) + \sum_{j=2}^{n} d(\gamma(a_j), \gamma(a_{j-1})), \tag{3.6}$$

for some metric d, and $a = (a_1, \ldots, a_n)$ such that $0 < a_1 < \ldots < a_n < 1$.

With a Euclidean metric the subset that maximizes the linear segment function is:

$$(p_{f(1)}, \ldots, p_{f(n)}) = \underset{f \in \mathbb{F}}{\operatorname{argmax}} \sum_{i=1}^{n} \|p_{f(j)} - p_{f(j-1)}\| \\ = \underset{f \in \mathbb{F}}{\operatorname{argmax}} \Gamma_X^n(f(1), \ldots, f(n)). \quad (3.7)$$

The method of finding $(p_{i_1}, \ldots, p_{i_n})$ on a m-polygon according to (3.7) will be referred to as the *Dijkstra Curve Maximization* (DCM) method since the set can be found by means of a modified version of the Dijkstras shortest path algorithm as seen in Algorithm 1.

Algorithm 1 Dijkstra Curve Maximization

```
Find (p_{i_1}, ..., p_{i_n}) of (3.7).
% Calculate distances
Set D_{i,k} = 0, i = 0, ..., n, k = 1, ..., m
for k = 2, ..., m − n + 1 do
    for i = 1, ..., min(m, n) do
        D_{i,k} = min_{l=i−1,...,k−1} D_{i−1,l}
        P_{i,k} = argmin_{l=i−1,...,k−1} D_{i−1,l}
    end for
end for
% Trace path backward
t = m
for l = n, ..., 1 do
    i_l = P_{n,t}
    t = P_{n−1,i_l}
end for
```

First some properties of the simplest types of curve segments with $n = 1$ will be shown for the continuous case. The following discussion is restricted to curves in $\mathbb{R}^2$ since this work is focused on handwritten characters. All curve segments are also assumed to be parameterized by arclength with significantly more points. The curve has been rotated so that it starts in the origin and and ends in $\gamma(1) = (x(1), 0)$. It is also assumed that no curve is a straight line. For the first property the case where the x-values are monotone-increasing are considered. In this case the notation $\gamma = (a, f(a)), a \in [0, x(1)]$ will be used.

Lemma 3.2.1. *Let $\gamma(t) = (t, f(t))$. If f is a constant function in the interval $a \in (0, x(1))$ such that $|f(a)| > 0, a \in (0, x(1))$ then $\Gamma^1_\gamma(a)(0, x(1))$ obtains it maximum at $a = x(1)/2$.*

Proof. Let $f(a) = \delta, a \in (0, x(1))$. Then

$$\Gamma^1_\gamma(a)(0, x(1)) = \Gamma(a) = \sqrt{a^2 + \delta^2} + \sqrt{(x(1) - a)^2 + \delta^2}.$$

This gives

$$\frac{d\Gamma}{da}(a) = \frac{a}{\sqrt{a^2 + \delta^2}} + \frac{a - x(1)}{\sqrt{(x(1) - a)^2 + \delta^2}}.$$

Then clearly $\frac{d\Gamma}{da}\big|_{a=x(1)/2} = 0$ and a sign study reveals that this is a global maximum. □

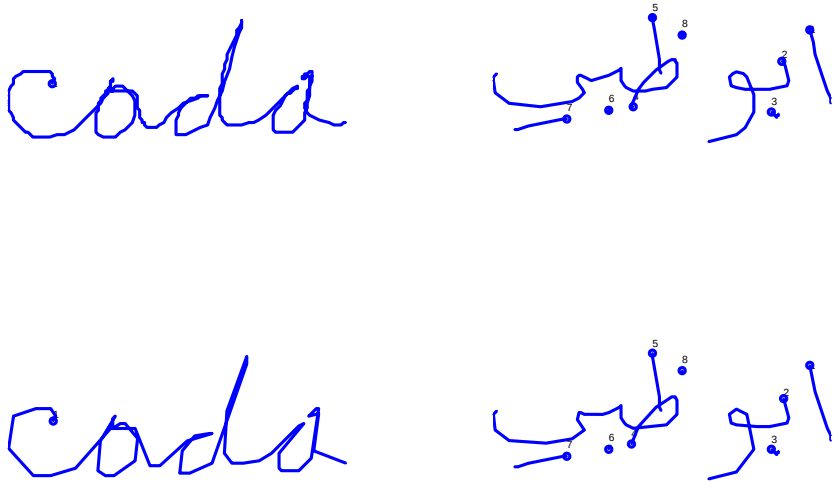
It is easy to deduce that the solution to the min-ϵ problem is the same as the solution to the longest path problem given in Lemma 3.2.1. This shows that these curve approximations are similar under some circumstances. One equally easily realizes that there are many cases when they differ. One interesting example are the respective solutions of the min-ϵ approach and the DCM to picking one point on a sinus curve on the interval $[0, 2\pi]$. Here the DCM has two optimal solutions lying close to the respective extreme points, whereas the min-ϵ approach will choose the middle point. In particular one easily observes that their behavior differ when the number n is less than the number of ϵ-local extreme points $p = \operatorname{argmin}_{\substack{p \in \gamma(t) \\ t_1 - \epsilon \leq t \leq t_1 + \epsilon}} p_y$ on the curve. The DCM gets many solutions in this case, all aiming at choosing one of the prominent features of the curve whereas the min-ϵ solution gives the mean path. Examples of the extracted sample points with DCM on some connected character sequences are shown in Figure 3.7. Apparently the DCM provides a nice and smooth curve.

Especially for handwritten characters where both the number n and m are comparatively small, the execution of DCM will not require a significant amount of processing power. Elementary calculations lead to the following statement:

Theorem 3.2.1. *The DCM algorithm for finding the set of n-points realizing $\max_{(p_1,\ldots,p_n)\subset(1,\ldots,m)} \Gamma^n_X(x_{p_1}, \ldots, x_{p_n})$ terminates after*

$$(n-2)(m-n+1) + \frac{(m-n+1)(m-n+2)}{2}$$

distance calculations.



(a) Spanish word *coda* (b) Arabic word ابوظبي

FIGURE 3.7: Two cursive words with the original sampling above and the structurally reparameterized words below. Here the DCM technique with $n = 3$ of Section 3.2.4 is used to parameterize each segment. The segmentation points are the local vertical extremes of the curve.

Proof. Denote the number of intermittent points by $n^* = n - 2$. Let k be the indexes of the point x_k and consider the required number of distance calculations to points with indexes larger than k. For the case when $k < n^* + 1$, only distances up to index $m - (n^* - (k - 1))$ need to be computed since n^* are required and at most $k - 1$ can be used up to index k. This implies $m-(n^*-(k-1))-k = m-n^*-1$ distance calculations based on point x_k. In total this contributes with $(n^*)(m-n^*-1)$ distance calculations. When $k \geq n^*+1$ all distances to subsequent points i.e. $m-k$ needs to be computed. The number of such calculations is an arithmetic sum $1+\ldots+m-(n^*+1) = \frac{(m-n^*-1)(m-n^*)}{2}$. Replacing n^* by $n - 2$ gives the sought time complexity. □

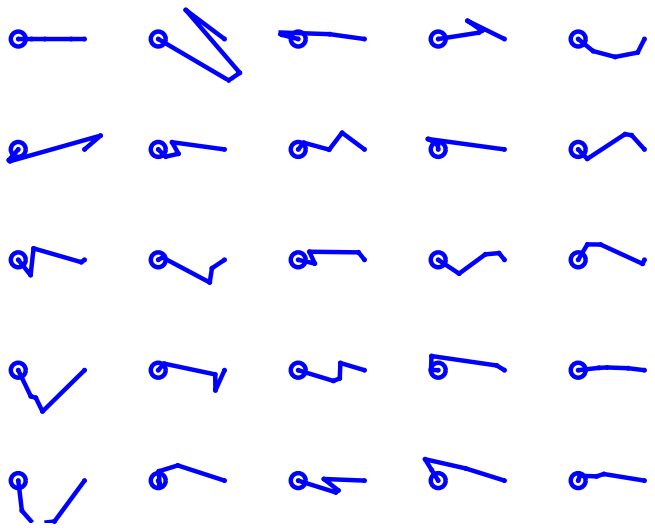


FIGURE 3.8: An example of a selection of segment arc types used to normalize shape of segments in template database and sample to speed up distance calculations.

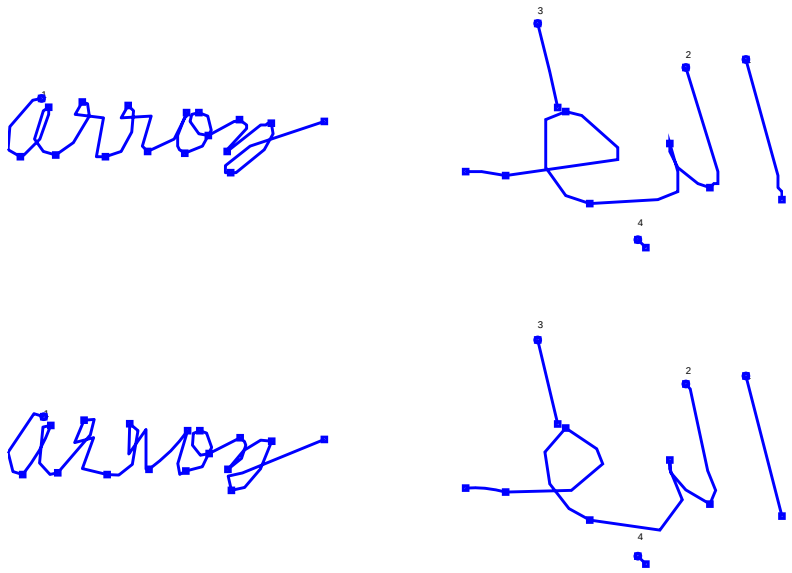
Discretization Of Curve Types

Another interesting property of segmented handwriting is that the variability in segment shape is significantly less than that of complete characters. Since in general, shape comparisons are computationally expensive, this has inspired to the possibility of discretizing the curve shape space, i.e. limiting the number of possible curve shapes to a fixed number of normalized *arctypes* as defined in Definition 3.2.3.

Definition 3.2.3. An arctype A will here be defined as any curve segment $\mathrm{A}(t) \in \mathbb{R}^2, t \in [0,1]$ such that $\mathrm{A}(0) = (0,0), \mathrm{A}(1) = (1,0)$.

In order find the best approximating arctypes for a given curve, a distance function for segment shape is needed. The alternative used for the segmented shape comparisons in this thesis is presented in Section 3.2.4 below. Examples of arctypes can be seen in Figure 3.8 and some samples forced into this space of discrete segment shapes for Arabic and cursive alphabetic script can be seen in Figure 3.9. Note how well the word *arroz* written by a Spanish native can be reconstructed using a limited set of 100 arctypes, generated from Arabic script samples, in Figure 3.9a.

This type of discretization of segment shape space has similarities with the type of structural reconstruction and analysis of on-line cursive script performed in the early 90's [87]. Parizeau et al. developed a programming language in which input was analyzed an reconstructed using segmentation points and fixed shapes in between [87, 88].



(a) Spanish word *arroz* (b) Arabic word البط

FIGURE 3.9: An example of arctype discretized cursive word samples. The top figures show the original samples and the bottom the samples restored with a limited set of arctypes. Both approximated from an arctype database generated from Arabic script with 105 arctypes.

Segmented Shape Distance

The curve distance function presented here is called DCM-DTW since it is a DTW (cf. Section 2.4.2) influenced distance function developed to discriminate well between curves parameterized according to the DCM method presented in Section 3.2. Points placed with DCM are spaced unevenly on the curve as this method focuses on retaining the shape information and not on providing

a smooth parameterization. For this reason a dynamic programming method matching two such point configurations need to allow Point-to-curve matching in addition to traditional Point-to-point matching. Furthermore as the number of points are few compared to traditional arclength parameterizations, the directional vector used in many implementations [3, 77], is no longer a stable feature. Instead DCM-DTW makes use of the intermittent angle θ_p of a point p defined as

$$\theta_p = \arg(p_{\epsilon+} - p) - \arg(p - p_{\epsilon-}), \theta_p \mod 2\pi \in [0, 2\pi), \tag{3.8}$$

where $p_{\epsilon+}, p_{\epsilon-}$ denote the next and the previous points at a distance ϵ on the curve and $\arg(p)$ is the angle of the vector p relative to the horizontal axis.

To accomplish the desired flexible Point-to-Curve matching, the closest point on each line segment of the opposing curve is calculated for each point. Let

$$L_j(t) = tp_{j-1} + (1-t)p_j, t \in [0, 1]$$

denote the line segment between points p_{j-1}, p_j on curve $P = \{p_k\}$. Let

$$t_{q_i}^{L_j} = \underset{t\in[0,1]}{\operatorname{argmin}} \|L_j(t) - q_i\| \tag{3.9}$$

for a line segment $L_j(t)$ on P and a point q_i on a compared curve Q. With this notation the *pseudo* points $x_{j,r}$ on the curve P w.r.t. the curve Q, both with n points are defined as $x_{j,r}^{P,Q} = L_j(t_{q_r}^{L_j})$.

A basic distance function between points or pseudo points $p, q \in \mathbb{R}^2$ is introduced as

$$\mathfrak{g}(p, q) = \|p - q\| + k_\theta|\theta_p - \theta_q|, \tag{3.10}$$

where k_θ is a normalization constant for balancing angle distance with coordinate distance. Notice that the angle values from (3.8) can be interpreted as a *signed* curvature as seen in the example below.

Example 3.2.2. *Given two curves $\gamma_1(x), \gamma_2(x)$ along the x-axis such that $\gamma_1(0.5) = 1, \gamma_1(x) = 0, x \neq 0.5$ and $\gamma_2(0.5) = -1, \gamma_2(x) = 0, x \neq 0.5$, then $|\theta_{\gamma_1(0.5)} - \theta_{\gamma_2(0.5)}|$ will attain the maximal value of 2π.*

A dynamic programming distance function based on this premise has an alignment function $\Phi(k) = (\phi_p(k), \phi_q(k))$ and transitions $(1, 0)$, $(1, \xi)$, $(\xi, 1)$, $(0, 1)$, $(1, 1)$ where the novel $(1, \xi)$ and $(\xi, 1)$ transitions mark the transition to or from the *pseudo* points in P or Q respectively. The alignment function has the requirements that $\Phi(1) = (1, 1), \Phi(m) = (n, n)$. When adding transitions to the alignment functions the addition of two ξ is defined by $\xi_1 + \xi_2 = 1$.

The alignment state $(\phi_p(r), \phi_q(r)) = (k+\xi, j)$ is defined as $(p_{\phi_p(r)}, q_{\phi_q(r)}) = (p_{k+\xi}, q_j) = (x^{P,Q}_{k+1,j}, q_j)$. The distance function d_{DCM} can now be formulated as

$$d_{DCM}(P,Q) = \min_{\Phi} \sum_{i=1}^{m} \mathfrak{g}(p_{\phi_p(i)}, q_{\phi_q(i)}), \tag{3.11}$$

where only the transitions $(0,1), (\xi,1)$ are valid from states $(\phi_p(r), \phi_q(r)) = (k+\xi, j)$, $k, j = 1, \ldots, n$ and similarly $(1,0), (0,\xi)$ are valid from states

$$(\phi_p(r), \phi_q(r)) = (k, j+\xi), k, j = 1, \ldots, n.$$

Notice also that the definition of the addition of the ξ transition to a ξ state implies that only $\xi = 0.8$ will be allowed for transition $(\xi, 1)$ from a state $(k+0.2, j)$ for instance.

As pointed out in [77] conventional DP-algorithms for matching handwritten characters suffer from over-fitting the template to the sample, and to improve the situation the simple distance function $\mathfrak{g}$ has been updated with a weight function $f : \mathbb{R} \to \mathbb{R}$ which also considers the context in which the points differ. The over-fitting problem for conventional DTW arises from the fact that samples of classes with very curved strokes such as the digit '3' differ much more than classes with straight strokes such as the digit '1'.

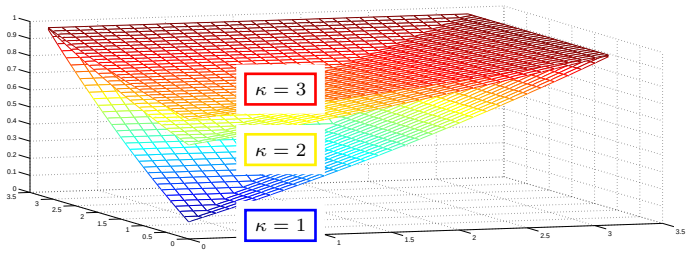


FIGURE 3.10: A plot of the relief function f_1 with $(a = -0.2, b = 1.1, c = 1)$ defined in (3.13) for the possible input values. The plot shows the appearance for $\kappa \in (1, 2, 3)$.

In this thesis the central feature distance function $\mathfrak{g}$ in (3.10) has been modified slightly by introducing continuous *relief* functions f_1, f_2 designed to cope with this problem. This is done by diminishing large feature differences between samples that have a similar feature profile as seen in the definition of $\mathfrak{g}$ in (3.12). A graphic view of these functions is included in Figure 3.10.

Define the height value p_h as the distance from the point p to the base line of the segment $P = \{p_j\}_{j=1}^{n}$, which in turn correspond to the line through p_1, p_n.

$$\begin{aligned}\mathfrak{g}(p,q) = f_1(\theta_p,\theta_q)\cdot|\theta_p-\theta_q|^2+\\ f_2(\theta_p,\theta_q,p_h,q_h)\cdot\|p-q\|^2.\end{aligned} \tag{3.12}$$

$$f_1(\theta_p,\theta_q) = \begin{cases} a\alpha^2/\kappa + b\alpha/\kappa + c, & \theta_p,\theta_q \le \pi \\ a\beta^2/\kappa + b\beta/\kappa + c, & \theta_p,\theta_q \ge \pi \\ c, & \text{otherwise}\end{cases}, \tag{3.13}$$

where $\alpha = (\pi - \max(\theta_p,\theta_q))/\pi)$ and $\beta = (\min(\theta_p,\theta_q) - \pi)/\pi$. The constant κ can be used to alter the appearance of the relief function whereas c is the default angle feature weight value.

$$f_2(\theta_p,\theta_q,p_h,q_h) = \begin{cases} (a\alpha^2 + b\alpha)\kappa(p_h,q_h) + c, & \theta_p,\theta_q \le \pi, y_1, y_2 \ge T \\ (a\beta^2 + b\beta)\kappa(p_h,q_h) + c, & \theta_p,\theta_q \ge \pi, p_h, q_h \ge T \\ c, & \text{otherwise}\end{cases}, \tag{3.14}$$

where

$$\kappa(p_h,q_h) = \frac{\min((\min(p_h,q_h) - T,\lambda)}{\lambda\omega}$$

with T being a threshold parameter for p_h, q_h. The length of between first and last point is λ and ω a weight parameter for this function, which in Figure 3.10 corresponds to κ.

The complete algorithm between two segments $P = \{p_j\}_{j=1}^{n}$ and $Q = \{q_j\}_{j=1}^{n}$ can now be formulated as in Algorithm 2.

3.3 Preprocessing

This section provides a brief overview of some common preprocessing techniques for on-line handwriting recognition.

3.3.1 Smoothing

In early versions of hardware with the aim of capturing electronic ink, the trace of the pen movement was often inexact causing several obstacles obstructing the recognition process. Often the curve sampling of the handwriting samples

Algorithm 2 DCM-DTW

for $i, j := 1, \ldots, n$ **do**
  if $i < n$ **then**

$$d(i_\xi, j) := \mathfrak{g}(x_{i,j}^{P,Q}, q_j) + \min \begin{cases} d(i, j-1) \\ d(i_\xi, j-1) \\ d(i, (j-1)_\xi) \end{cases}$$

  end if
  if $j < n$ **then**

$$d(i, j_\xi) := \mathfrak{g}(p_i, x_{j,i}^{Q,P}) + \min \begin{cases} d(i-1, j) \\ d(i-1, j_\xi) \\ d((i-1)_\xi, j) \end{cases}$$

  end if

$$d(i,j) := \min \begin{cases} d(i-1,j) + \mathfrak{g}(p_i, q_j) \\ d(i-1, (j-1)_\xi) + 2\mathfrak{g}(p_i, q_j) \\ d(i, (j-1)_\xi) + \mathfrak{g}(p_i, q_j) \\ d(i, j-1) + \mathfrak{g}(p_i, q_j) \\ d((i-1)_\xi, j-1) + 2\mathfrak{g}(p_i, q_j) \\ d((i-1)_\xi, j) + \mathfrak{g}(p_i, q_j) \\ d(i-1, j-1) + 2\mathfrak{g}(p_i, q_j) \end{cases}$$

end for
$d_{DCM}(P, Q) := d(n, n)/2n$

would be very jagged curves [47]. For isolated single character recognition the arclength resampling progress has the implicit side effect of a low-pass filter. Even though the hardware for capturing online handwriting is much more reliant nowadays smoothing techniques could still be applicable for handwriting under extremely shaky conditions. Most segmentation methods would benefit from smoothing under such circumstances as they are sensitive to discontinuities in the curve. The DCM parameterization itself is also sensitive to high frequency noise in that it always maximizes the path length. Garutto et al. attacks the problem by extracting salient feature points in multiple scales [111].

3.3.2 Helpline Estimation

With segmentation methods such as the one described in Section 3.2 a reliable estimate of the writing direction is absolutely crucial since this is the basis for the extracted set of segmentation points. An intuitive user interface can somewhat remove the dependency of reliable helpline estimation by guiding the

writer to write in a certain direction or even by explicitly inserting helplines [36]. Often the estimated helplines may also serve as input to further scale and slant normalization as described in Sections 3.3.3-3.3.4.

3.3.3 Scale Normalization

For template matching strategies but also for feature generators feeding statistical recognition methods, scaling of input is often an important part of the preprocessing stage. Features dependent on coordinate values such as horizontal and vertical displacement between certain feature points are intuitive and powerful features included in many recognition systems [36, 50, 85]. For horizontally written scripts, the scale normalization is often approximated by scaling to certain inferred help-lines [20, 36, 55, 90, 101]. LeCun et al. report achieving better recognition results for normalization based on global helpline estimation than character level normalization [13].

It is also possible to make a recognition system invariant to global scale differences in input by simply assuring that all features are scale-invariant. Examples of some such features will be provided in Section 4.2.

3.3.4 Slant Correction

For handwriting, especially horizontally written scripts, a typical writer dependent consistent shape perturbation is the grade of slanting of the writing. This may differ not only between writers but also depending on the writing style. Especially for cursive handwriting recognition, it is common to infer the slant of the input sample to limit the between-writer variability and thus facilitate modeling by removing the skew correction from the recognition stage [47, 109]. Severe slanting may also cause inconsistent placement of segmentation points for some segmentation methods such as the naive segmentation method in Section 3.2. Slant correction can therefore be beneficial to recognition methods based on such segmentation strategies.

The great merit of deslanting techniques in template based systems for handwriting recognition are that they may remove some of the global template variations and thus reduce the need for some modeling. On the other hand these techniques generally also make assumptions based on the mean slant of all segments in a sample and may thereby cause some misinterpretations. In particular for connected sequences of characters a global deslanting process will fail to capture variations in slant within a word.

3.4 Experiments

In this section some quantitative results for the DCM segment shape approximation technique described in Section 3.2.4 will be given. It is difficult to evaluate parts that will be components in a larger context, such as the segmental shape comparisons made here, but to get some results, the single segment digits i.e. all samples X such that $|\mathcal{S}(X)| = 2$ were extracted from the train and test sets of the UNIPEN/1a dataset (cf. Section 9.2.2). These were then used in a recognition experiment such that all samples in the training set were used as templates. The comparison is made to the standard mass center normalized DTW (cf. Section 2.4.2). The top-n results denote that the correct class was among the n best. Although this is a very limited test, it does show that Algorithm 2 works and that it provides competitive results in a simple recognition setting.

Method	Top-1	Top-2
DCM-DTW	**98.93%**	100%
DTW (MC)	97.87%	99.79%

TABLE 3.1: Recognition results on the single segment samples of the UNIPEN/1a data set (includes 469 single segment test digits of '1','2','3','5', '7' and '9')

CHAPTER 4

Additive Template Matching

I have had my results for a long time, but I do not yet know how I am to arrive at them.

Carl Friedrich Gauss

TEMPLATE MATCHING SCHEMES for on-line handwriting recognition are essentially the techniques for comparing objects in the form of discretely sampled curves. In recent years template matching methods have mainly been applied to the problem of recognizing single characters implying that each such curve object corresponds to a single symbol, but some previous work on applications to template sequences can be found in [36, 50]. This chapter presents a template matching scheme especially designed to work with segmented input by applying similar object recognition methods to the set of curve parts resulting from the segmentation strategy. In subsequent chapters it will be shown that this design creates a suitable framework for easy extension of single character recognition to connected character recognition. In this chapter however, focus is on the task of comparing two segmented objects, i.e. the search for a suitable distance function applicable to segmented input. It will also be shown that this particular design enables exploitation of some other beneficial traits of segmented input.

4.1 The Frame Deformation Energy Model

Inspired by the possible consistency of certain key feature points in the set of samples portraying a given handwritten character shape, the starting point for the search of a suitable distance function will be based on the segmentation points of a sample. To simplify the discussion somewhat, the targets of recognition in this chapter are limited to single characters, i.e. input portraying one isolated character. The subset of points of a template (i.e. model) corresponding to the structurally significant segmentation points will here be referred to as the *frame* of the character template. Given two samples X, Y with similar segmentations $\mathcal{S}(X) \sim \mathcal{S}(Y)$, i.e. same number of segments and points per segment it is possible to define a transformation $\mathrm{T} : X \to Y$. In particular this transformation can be written as

$$\mathrm{T}(X) = \mathrm{T}_{\mathrm{frame}}(X) + \sum_{i}^{|\mathcal{S}(X)|} \Gamma_{\mathcal{S}_i}(\mathrm{T}_{\mathrm{frame}}(\Lambda_i^X) - \Lambda_i^Y), \tag{4.1}$$

where $\mathrm{T}_{\mathrm{frame}}$ has the property that $\mathrm{T}_{\mathrm{frame}}(\mathcal{S}(X)) = \mathcal{S}(Y)$ and each function $\Gamma_{\mathcal{S}_i}$ takes the points on the transformed segments $\mathrm{T}_{\mathrm{frame}}(\Lambda^X)$ in the frame to the corresponding points on Λ^Y in the target sample. This splitting creates an intuitive scenario for application of a classic *coarse-to-fine* recognition strategy. The coarse part of the recognition process can be to evaluate the magnitude of $\mathrm{T}_{\mathrm{frame}}$ and the fine part to compare the residuals $\mathrm{T}_{\mathrm{frame}}(X) - Y$.

In this thesis the magnitude of $\mathrm{T}_{\mathrm{frame}}$ will be assessed by introducing an analogy

with a mechanical framework of stretchable coils and springs. Each segment in a frame (consisting of two segmentation points) can be exactly transformed into a segment in a second frame by translating, rotating and scaling. Translation of a segment is not relevant unless there are intermediate lifts of the pen in the handwriting and for now this can be ignored. Scaling and rotating a two-point segment fits well into such an analogy since it can be modeled mechanically by a spring and a coil. After performing this transformation on the first pair of segments, the first two segmentation points will coincide. This removes the need for further translation and the third segmentation point will coincide after application of suitable rotation and scaling. This way the whole frame transformation process can be seen as twisting the coils and springs of each segment of the frame successively. The mechanical analogy is fairly popular and has for instance been used for describing a word normalization process in the past [13]. Another way of illustrating the magnitude of such a transformation is to plot a deformation grid as for the thin-plate spline transformation in Figure 3.4.

4.2 Feature Space

Although the choice of distance function for a template matching problem is of critical importance for maximizing recognition accuracy, even very simple distance functions can produce great results if operating in a feature space suitable for the matching problem. Many researchers have stressed the importance of the feature space in the past [85]. Lots of time has been devoted to the design of features aimed at treating a specific discrimination problem in handwriting. Apart from the dominant features also used in single character recognition, namely vertical and horizontal positioning along with local direction [8, 29, 77] the most common features include Fourier coefficients [36], curvature [70, 55, 85, 101], speed [70, 101] and the hat feature [35, 70, 55, 101] aimed at detecting diacritical marks. Jaeger et al. and Liwicki et al. have defined even more features but in an attempt to optimize the selection Liwicki found that a subset of only five features was enough to produce competitive results [55, 70]. One way to include the strengths of off-line systems (the insensitivity to coordinate sequences) into an on-line system is to add off-line features such as context maps [55].

In view of the two-step strategy of the Frame Deformation Energy concept described in Section 4.1 the features used in this thesis can also be divided into two categories:

- Frame features corresponding to the magnitude of the stress on ”coils” and ”springs” on the frame when bending.

- Segmental features corresponding to the difference between two segments with common start and end (i.e. with frame deformation removed)

To simplify notation a point of m features in feature space $\mathfrak{F}^m$ will be denoted by $\mathfrak{f}$ and the jth feature by $\mathfrak{f}_j$.

4.2.1 Frame Features

Since the modeling concept for the frame deformations are based on the mutual relations of length and angles between subsequent segments in the frame, the two natural features corresponding to this are of course relative length and angle of the segments. This feature is not new as is, relative length has been used in a system based on HMM in the past [35], but then without the type of normalization described in Section 4.2.1. A complete list of frame features used within this thesis can be found in Table 4.1. An interesting comparison is Duneau et al. who use the segment angle and length as features, enabled through a scale normalization enforced by a restrictive user interface [36]. One of the key features of the frame features in this thesis is the correlation between subsequent segments through the relative features. This concept of relating subsequent segments through relative shape is an intuitive way to extend a single character recognition function to connected script [79].

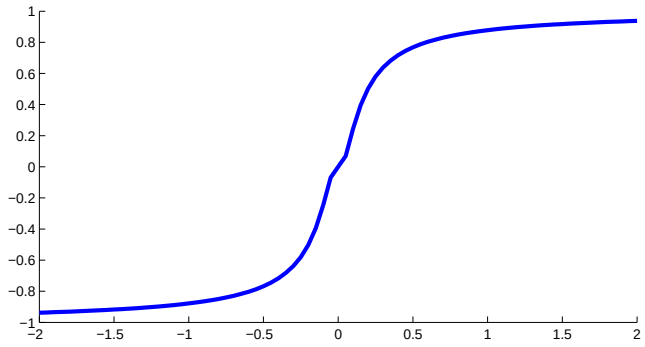


FIGURE 4.1: A plot of the normalization function $\mathcal{H}(r)$ for logarithmic ratio values $\log(r)$ in the range (-2, 2).

Frame Features	
Name	Description
Angle, ϕ	The angle of the segment between p_i, p_{i+1} as calculated in the input coordinate system $p_i = (p_{ix}, p_{iy})$ i.e. $\arctan(\frac{p_{(i+1)y}-p_{iy}}{p_{(i+1)x}-p_{ix}})$.
Relative Length, λ	The length of a segment (p_i, p_{i+1}) in relation to the length of a subsequent segment (p_{i+1}, p_{i+2}). Calculated through a normalizing function $\mathcal{H}(\frac{\|p_{i+1}-p_i\|}{\|p_{i+2}-p_{i+1}\|})$ as described in Section 4.2.1.
Relative Horizontal Position, $\mathcal{R}_x$	The relative displacement of the mean horizontal value compared to previous segment, as in $\frac{\bar{\Lambda}_i^x - \bar{\Lambda}_{i+1}^x}{N_X}$, where $\bar{\Lambda}_i^x$ is the mean horizontal value for segment i and N_x is a normalization constant.
Relative Vertical Position, $\mathcal{R}_y$	The relative displacement of the mean vertical value compared to previous segment, as in $\frac{\bar{\Lambda}_i^y - \bar{\Lambda}_{i+1}^y}{N_Y}$, where $\bar{\Lambda}_i^y$ is the mean vertical value for segment i and N_y is a normalization constant.

TABLE 4.1: A table of the frame features used within this thesis.

Ratio Normalization

Some features e.g. angular values are intrinsically limited by their periodic nature. Weights can be used to balance magnitude of different features but this is an insufficient remedy for balancing bounded and unbounded features. To handle this problem a normalization function $\mathcal{H} : \mathbb{R} \rightarrow [a, b]$ that takes unbounded values to a bounded interval has been introduced. For the features covered in this thesis this applies to the relative length features λ. The aim of the function design is that it should be tolerant to small differences and reach a maximum value comparable to periodic features such as the angular values. The values for ratios in the interval (0.01, 100) can be seen on a logarithmic scale in Figure 4.1.

$$\mathcal{H}(r) = \begin{cases} (2\arctan(\frac{3\ln(r)}{2\pi}))^3, & \text{if } |(2\arctan(\frac{3\ln(r)}{2\pi}))^3| < 1 \\ \text{sign}(\ln(r)), & \text{otherwise.} \end{cases} \tag{4.2}$$

4.2.2 Segmental Features

The primary objective of this set of features is to discriminate between handwriting samples of different classes that display similar shape variations in frame. A typical example of this are the differences between samples of the letters U and V as seen in Figure 4.2. The complete set of segmental features used within this thesis can be found in Table 4.2.

Segmental Features	
Name	Description
Arctype, A	The curve shape of the segment from Definition 3.2.3 (cf. Section 3.2.4).
Connection Angle, θ	The local angle between subsequent curve types at the connection point as seen in Figure 4.2.

TABLE 4.2: A table of the segmental features used within this thesis

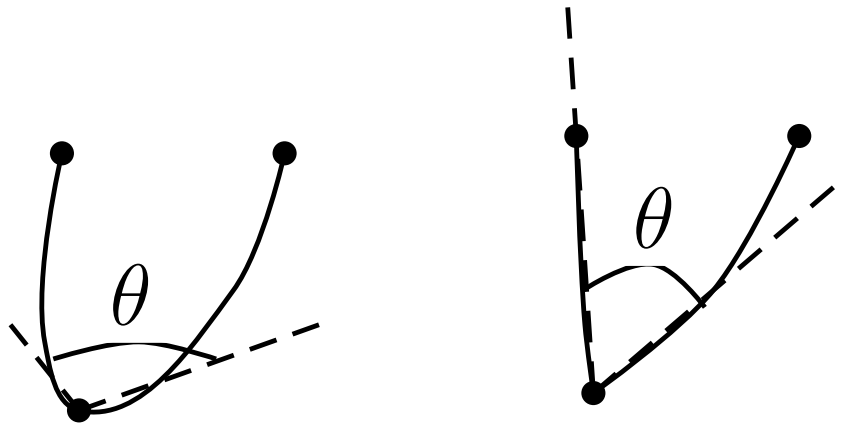


FIGURE 4.2: Two handwritten characters u and v with identical frame features. The connection angle θ is calculated as the angle between local subsequent arctypes and this is still significantly different.

Pen-up Modeling

A pen-up movement is actually the act of *non-writing* and thereby corresponds to movement which is not recorded by the sampling device. But although there is no information on exactly how the writer has moved the pen in between two pen-down movements, the actual points where the pen-tip has entered or exited the sampling device are significant. The difference between a pair of such pen-up and pen-down movements correspond to a translation and will here be referred to as a *pen-up segment*. For consistency the same modelling as for the actual writing has been applied to these segments with the exception of non-relevant features such as the Arctype in Table 4.2.

4.2.3 Virtual Reference Segments

Since the features of size and position as described in Section 4.2 are relative they will be undefined for the first segment in a match making it impossible to discriminate between shapes with common appearance but different size or position. To enable the inclusion of such properties it is possible to add virtual segments to the template matching both before and after the sample to be recognized as seen in Figure 4.3 if there is such information available in the user interface. With the aid of such segments it is possible to include size and position in the features of the first segment indirectly. In particular this design enables easy application of helpline information when available, without making the system helpline dependent.

FIGURE 4.3: A cursive word sample plotted with virtual reference segments in solid bars before and after the words. The dotted lines mark the pen-up segments in the sample.

4.3 Distance Function in $\mathfrak{F}^m$

Despite the attractive appearance of thin-plate deformed grids and some successful applications of thin-plate splines to the character recognition problem in the past [12], it is not suitable to use this model for frame deformation distance calculations. The main problem is that common variations in handwritten patterns involves points on the extension of a line being distributed on either side of the line causing folding of the thin-plate spline.

Instead a more straightforward implementation of the Frame Deformation Energy concept described in Section 4.1 will be used to define a distance function in $\mathfrak{F}^m$. If each bending operation on the frame could be done independently of the others then the energy required would be a simple sum of the components. It is therefore natural to introduce a linear distance function acting in this space. More importantly, as seen in the tables of features in Tables 4.1 and 4.2, several of the features themselves depend on their segmental context. Keeping the distance function linear in features also enable separation of context-dependent (relational) and context independent features. Since the context-dependent features are relevant whenever there is context, i.e. the current segment is followed by one or more other segments, this distance component is called connective distance d_C. Similarly the context independent part will be referred to as segmental distance d_S. A linear function in feature space $\mathfrak{F}^m$ can be written

$$d(X,Y) = \sum_{j}^{p} w_p d_{\mathfrak{f}}(\mathfrak{f}_j^X, \mathfrak{f}_j^Y) = \sum_{j\in \mathrm{Rel}} w_j d_C(\mathfrak{f}_j^X, \mathfrak{f}_j^Y) + \sum_{k\in \mathrm{Seg}} w_k d_S(\mathfrak{f}_k^X, \mathfrak{f}_k^Y), w_j \geq 0. \quad (4.3)$$

The greatest problem with this distance function is the question of optimality and the difficulty in optimizing the function with respect to recognition feature weights w_k. This discussion will be temporarily disregarded and it is just noted that even simple ad-hoc values can produce great results given well-balanced features. For now it is assumed that the feature values themselves have been balanced and that the weights are set to unit size, $w_j = 1$.

Viewing the variables $\mathfrak{f}^X, \mathfrak{f}^Y$ of samples X, Y as sequences of points in feature space $\mathfrak{F}^m$ it follows that (4.3) actually correspond to a weighted norm $(\mathfrak{f}^X - \mathfrak{f}^Y)^T W(\mathfrak{f}^X - \mathfrak{f}^Y)$ when the functions d_C, d_S also are sum of squares. Observe that this is a template independent normalization of feature space and not related to weighting through feature distribution analysis as performed in [50].

4.4 Segmented Template Matching

With a distance function acting in the segmented feature space $\mathfrak{F}^m$ as defined in Section 4.3 the process of segmented template matching is now straightforward. Given a template T and a sample X with the same number of segments, the distance between these two can be calculated simply as

$$\mathfrak{G}(T,X) = \sum_{j}^{|\mathcal{S}(T)|-1} d(\Lambda_j^T, \Lambda_j^X), \tag{4.4}$$

where $d(\Lambda_j^T, \Lambda_j^X)$ is the distance function from (4.3), and Λ^T, Λ^X the segments of T, X respectively.

4.4.1 Additivity

Enabling *fair* comparison of an input sample to various sequences of templates imposes requirements on the distance function. By *fair* is here meant that the distance value is independent on the number and position of connection points in the template sequence, and only on the combined shape. Some new terminology is introduced below to simplify the discussion and to show that the template distance function in (4.4) indeed has this property. An interesting comparison is the intrinsically sequential Hidden Markov Modeling technique, which given that segmental matching probabilities are unconditionally independent, can utilize the Viterbi algorithm to find the modeling sequences that maximizes the probability of generating input [108] (cf. Section 2.3.1).

Segmentations of two different sets of equal number of n strokes (X, Y) are said to be *similar*, $\mathcal{S}(X) \sim \mathcal{S}(Y)$ if $|\mathcal{S}(X^j)| = |\mathcal{S}(Y^j)|, j = 1, \ldots, n$. With respect to the pen-up modeling described in Section 4.2.2 this means that the similarity of two sequences will be valid if and only if the pen-up segments are in the same place.

To clarify the process of comparing sequences of templates, two operations in sample space $\mathbb{X}$ are introduced. The additive operator $+$ is used for adding two separate samples P, Q by letting the first point of the second sample constitute the starting point of a new stroke. The concatenation operator $\cup$ will denote the connection of two samples P, Q by attachment of the first point of the second segment to the last point of the first. The difference between these operations are illustrated in Figure 4.4. As seen in the figure, the $+$ operation differs to the $\cup$ operation by the introduction of a pen-up segment as described in Section 4.2.2. Let the partitioning $\mathcal{P}(X) = (\mathfrak{O}, I)$ of a sample X be defined as an operation $\mathfrak{O} \in \{+, \cup\}$ and an index I such that

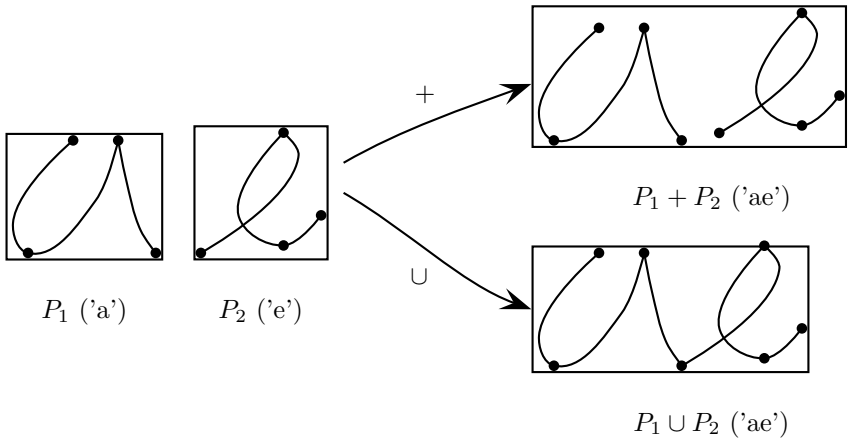


FIGURE 4.4: The difference between addition + and concatenation ∪ of two samples P_1 and P_2 with the resulting samples and their class labels to the right.

$\mathfrak{O}(\{\Lambda_j^X\}_{j=1}^{I-1}, \{\Lambda_j^X\}_{j=I}^{|\mathcal{S}(X)|-1}) = X.$

Definition 4.4.1. A distance function $\mathfrak{G} : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is *additive* if for any two samples $X, Y \in \mathbb{X}$ such that $\mathcal{S}(X) \sim \mathcal{S}(Y)$ and any partitioning $\mathcal{P} = (\mathfrak{O}, I)$ the difference

$$\mathfrak{G}(X,Y) - \mathfrak{G}(\{\Lambda_j^X\}_{j=1}^{I-1}, \{\Lambda_j^Y\}_{j=1}^{I-1}) - \mathfrak{G}(\{\Lambda_j^X\}_{j=I}^{|\mathcal{S}(X)|-1}, \{\Lambda_j^Y\}_{j=I}^{|\mathcal{S}(Y)|-1}) = \alpha(\mathcal{P}, X, Y) \quad (4.5)$$

does not depend on the partial matches $\mathfrak{G}(\{\Lambda_j^X\}_{j=1}^{I-1}, \{\Lambda_j^Y\}_{j=1}^{I-1})$ or $\mathfrak{G}(\{\Lambda_j^X\}_{j=I}^{|\mathcal{S}(X)|-1}, \{\Lambda_j^Y\}_{j=I}^{|\mathcal{S}(Y)|-1})$.

Example 4.4.1. *The conventional DTW distance $\mathfrak{G}_{DTW}$ with mass center normalization is a typical example of a distance function which is not additive according to Definition 4.4.1. Since alignment is performed it is very likely that the complete match $\mathfrak{G}_{DTW}(X,Y)$ would differ in alignment on the subpart $\{p_j^X\}_{j=1}^{I-1} \subset X$ from the alignment obtained with $\mathfrak{G}_{DTW}(\{p_j^X\}_{j=1}^{I-1}, \{p_j^Y\}_{j=1}^{I-1})$. Thus $\mathfrak{G}_{DTW}(X,Y)$ can not be calculated from the partial matches without knowing the alignment of the partial matches. This implies that α depends on $\mathfrak{G}_{DTW}(\{p_j^X\}_{j=1}^{I-1}, \{p_j^Y\}_{j=1}^{I-1})$.*

The following theorem follows immediately from this Definition 4.4.1 but is included for completeness:

Theorem 4.4.1. *The template distance function in (4.4) is additive according to Definition 4.4.1.*

Proof. Let X, Y be two samples such that $\mathcal{S}(Y) \sim \mathcal{S}(X)$ and let $\mathcal{P} = (\cup, I)$, then

$$\mathfrak{G}(Y,X) = \sum^{|\mathcal{S}(X)|-1} d(\Lambda_j^X, \Lambda_j^Y) =$$

$$\sum^{|\mathcal{S}(X)|-1} [\sum_{k\in \mathrm{Rel}} w_k d_C(\mathfrak{f}_{jk}^X, \mathfrak{f}_{jk}^Y) + \sum_{l\in \mathrm{Seg}} w_l d_S(f_{jl}^X, \mathfrak{f}_{jl}^Y)] =$$

$$\sum_j^{I-1} d(\Lambda_j^X, \Lambda_j^Y) + \sum_{i=I}^{I+1} [\sum_{k\in \mathrm{Rel}} w_k d_C(\mathfrak{f}_{ik}^X, \mathfrak{f}_{ik}^Y) + \sum_{l\in \mathrm{Seg}} w_l d_S(\mathfrak{f}_{il}^X, \mathfrak{f}_{il}^Y)] +$$

$$\sum_{i=I+1}^{|\mathcal{S}(X)|-1} d(\Lambda_i^X, \Lambda_i^Y) = \mathfrak{G}(\{\Lambda_j^Y\}_{j=1}^{I-1}, \{\Lambda_j^X\}_{j=1}^{I-1}) + \mathfrak{G}(\{\Lambda_j^Y\}_{j=I}^{|\mathcal{S}(Y)|}, \{\Lambda_j^X\}_{j=I}^{|\mathcal{S}(X)|}) +$$

$$\sum_{i=I}^{I+1} \sum_{k\in \mathrm{Rel}} w_k d_C(\mathfrak{f}_{ik}^X, \mathfrak{f}_{ik}^Y) + \sum_{l\in \mathrm{Seg}} w_l d_S(\mathfrak{f}_{Il}^X, \mathfrak{f}_{Il}^Y).$$

Evidently α in (4.5) is independent of $\mathfrak{G}(\{\Lambda_j^Y\}_{j=1}^{I-1}, \{\Lambda_j^X\}_{j=1}^{I-1})$ since it can be defined as $\sum_{i=I}^{I+1} \sum_{k\in \mathrm{Rel}} w_k d_C(\mathfrak{f}_{ik}^X, \mathfrak{f}_{ik}^Y) + \sum_{l\in \mathrm{Seg}} w_l d_S(\mathfrak{f}_{Il}^X, \mathfrak{f}_{Il}^Y)$. Similar operations can be done to show the same thing for $\mathcal{P} = (+, I)$. □

CHAPTER 5

Connected Character Recognition With Graphs

If I were to awaken after having slept for a thousand years, my first question would be: Has the Riemann hypothesis been proven?
David Hilbert

THIS CHAPTER TREATS the tools needed to extend the template matching scheme presented in Chapter 4 to additive recognition of arbitrary sequences of connected or non-connected characters. By using an additive distance function such as the one presented in Section 4.4 for the segmental template matching, graph techniques become powerful tools for evaluating partial template matches.

5.1 Introduction

The by far most common strategies for recognizing on-line cursive script in the literature is to conduct recognition on subparts of the input using a neural network [55, 82, 101, 104] or network of HMMs [68, 107]. The main motivation behind the popular Neural Network based approach is that it avoids the reliance on explicit segmentation points [104]. In these methods the Neural Network produces a detection score for each letter and each time frame, thus generating what is often referred to as a *detection matrix* in time and letter (See Section 2.6.1). Classification with Neural Networks in each frame is a fairly quick operation and it seems that a large part of the time complexity here derives from lexical post-processing. Most of these methods base the final recognition hypothesis on searching through the detection matrix for given dictionary words and will thus not produce any results without dictionary [82] although it is possible to retrieve non-dictionary results simply as the most probable path through the detection matrix [101]. Limiting the problem to the search for the best recognition hypothesis from a dictionary enables other holistic feature approaches for dictionary reductions which can produce good results efficiently but may be less robust for large variations in input [36, 94].

One of the major merits of a template based system compared to these conventional methods is that template based systems can produce better results when training data is scarce (lesser risk for overtraining). Furthermore the additivity of the template matching distance in Chapter 4 seems to make it easier to separate pure shape matching from linguistic processing. There are very few examples of template based systems for on-line cursive script in the literature and it seems that this type of *segment and recognize* approaches were more or less abandoned in the mid-90's [50, 90]. In one of the first accounts of on-line cursive script recognition, however, Tappert employs a template based system [124]. Tappert does not make use of segmentation points and instead dynamically matches input to complete sequences of possible templates. The three-dimensional lattice fed to dynamic programming however is computationally costly and for practical use required limitations in form of letter transitions as well as segmentation points. Segmentation graphs have also been used for

structurally defined templates but the actual pattern matching ranking criteria used differs greatly from conventional distance functions acting in feature space [89]. The previous work on template based matching with graphs which seems to bear most resemblance to the strategy presented in this thesis are Ford et al. [50], Duneau et al. [36] and Oh [85]. Ford et al. use statistically derived segmental matching scores. Duneau et al. apply a template matching scheme but use a dictionary driven search for word hypothesises. Oh applies Fischer discriminant analysis at the letter recognition stage and builds word hypothesises by evaluating a lattice structure with properties similar to that of the detection matrix used in the classical neural network approach [101]. Graphs have also been used for recognition of off-line cursive script [26, 39, 44, 71] and many aspects such as the dictionary structure and several graph search algorithms can be used for both settings.

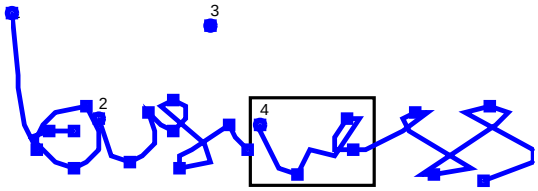
An important property of the methods based on implicit segmentation is the possibility to produce confidence scores (albeit low) even for severely degraded samples of characters. The template matching scheme in the most basic form presented in this chapter requires each template to have a similar segmentation to the part it is matched against. In other words for each template T there has to be a subset of segments $X^T = \{\Lambda_j^X\}_{j=a}^{a+|\mathcal{S}(T)|-2}$ in $X = \{\Lambda_j^X\}_{j=1}^{|\mathcal{S}(X)|-1}$ such that $\mathcal{S}(T) \sim \mathcal{S}(X^T)$. This is equivalent to finding the sequence of concatenated templates in the database that approximates the input best. In turn this means that input where some of the constituent characters have degraded so that they are illegible out of context, such as the samples seen in Figure 5.1, are also not recognizable by the system.

The remainder of the chapter is organized as follows: First the concept of connection properties for templates will be discussed in Section 5.2. This is the fundamental concept needed for defining the space of available template sequences implicitly defined through allowing recurrent connections in the template database. The next section presents the segmentation graph and finally Section 5.5 presents the secondary graph structure producing the actual recognition hypothesis.

5.2 Connection Properties

It is well-known that the shapes of characters in a connected character sequence are affected by surrounding characters [137]. In this section it will be shown how such context dependent shape information can be introduced into a template based recognition strategy through a template connectivity function. These restrictions serve two purposes, (1) limitation of the possible set of combinations reduces time complexity and (2) avoiding sample input to be

(a) Missing letter shapes for *ng* in frame.



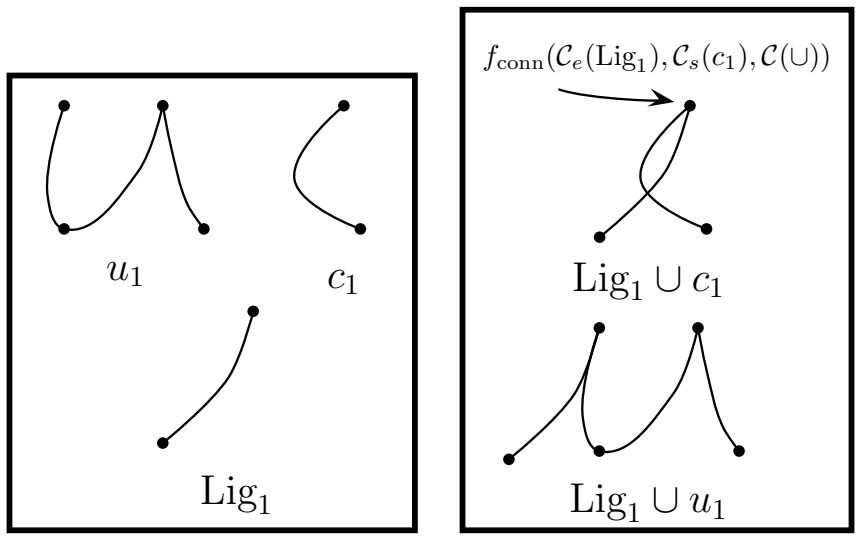
(b) Segmentation error missing points for *n*.

FIGURE 5.1: Two examples of degenerate cursive writing with letter shapes generally not included in a segmentation based template database.

compared to a template sequence corresponding to a non-existent handwritten pattern (which could possibly be similar to an existing pattern for another set of templates as seen in Figure 5.2b). Prohibiting connections of characters that would correspond to an invalid shape by imposing constraints in modeling connections has been evaluated for networks of HMM for Korean script in the past [107]. As stated there, optimally the pen-down connection between characters, here called *ligatures*[1], should optimally be modeled individually for each pair of characters. A reason for this is that some ligatures suitable for some character connections may cause an unobserved shape when combined with other characters as in Figure 5.2, where 'c' in combination with a ligature turns into the shape of the letter 'e'. By introducing restrictions on connections it is possible to selectively choose the type of ligatures that should connect to a certain character. The tradeoff of such manual control over the recognition system similar to the strategy of syntactical recognition is the extra effort put into template design [87].

In order to control whether two templates may connect to each other using one of the operations $+, \cup$ defined in Section 4.4.1 a set of connection properties $\mathcal{C}_s, \mathcal{C}_e$ can be assigned to each template. Since connection may occur both before and after each template separate connection properties will be given to

[1] In other work often also called a *letter-join*[50]



(a) Database (b) Implicit template sequences

FIGURE 5.2: An example of connective properties for templates in the database. Figure 5.2b shows some template sequence shapes implicitly defined by allowing connections between all templates.

the starting point $\mathcal{C}_s(T)$ and the end point $\mathcal{C}_e(T)$ of each template T.

Connectivity may also be dependent on the connective operation, denoted by $\mathfrak{O} \in \{+, \cup\}$ used to connect the templates. The case of the adding $(+)$ operation adds an intermittent pen-up segment unlike the concatenation $(\cup)$. The connectivity properties of the operation are written as $\mathcal{C}(\mathfrak{O})$ in correspondence with earlier notation. For a more compact representation the functional join $\uplus$ dependent on the operation is introduced as an alternative segmental breakdown of a sample X. With this representation X can be written as $X = \uplus_j(\Lambda_j^X, \mathfrak{O}_j)$, where $\{\Lambda_j^X\}$ is the segment sequence corresponding to $\mathcal{S}(X)$. With this notation the Connectivity Function for templates in a database can be defined as in Definition 5.2.1.

Definition 5.2.1. Let $\mathcal{C}_s(T), \mathcal{C}_e(T)$ denote the sets of connective properties of the template T at the start and endpoint respectively. Then the binary non-commuting connective function between two templates T_1, T_2 and a connective operation $\mathfrak{O}$ is defined as:

$$f_{\text{conn}}(T_1, T_2, \mathfrak{O}) = \begin{cases} 1, & \text{if } \mathcal{C}_e(T_1) \cap \mathcal{C}_s(T_2) \cap \mathcal{C}(\mathfrak{O}) \neq \emptyset \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Example 5.2.1. *The connectivity property is best illustrated by an example. Based on the templates seen in Figure 5.2, create and assign the connection properties listed in Table 5.1. From Definition 5.2.1 it is now clear that* $f_{conn}(T_1, T_2, \mathfrak{O}) = 1$ *for the following triplets:* $(c_1, Lig_1, \cup)$, $(u_1, Lig_1, \cup)$, $(Lig_1, u_1, \cup)$, $(c_1, u_1, +)$, $(u_1, c_1, +)$, $(c_1, c_1, +)$, $(u_1, u_1, +)$.

Set	Properties
$\mathcal{C}_s(\text{Lig}_1)$	C_L_1
$\mathcal{C}_e(\text{Lig}_1)$	C_L_2
$\mathcal{C}_s(c_1)$	C_L_3, C_U_1
$\mathcal{C}_e(c_1)$	C_L_1, C_U_1
$\mathcal{C}_s(u_1)$	C_L_2, C_L_3, C_U_1
$\mathcal{C}_e(u_1)$	C_L_1, C_U_1
$\mathcal{C}(+)$	C_U_1
$\mathcal{C}(\cup)$	C_L_1, C_L_2, C_L_3

TABLE 5.1: An example of a set of connective properties for the templates seen in Figure 5.2.

5.2.1 Ligaturing

The pen-down movement between two letters in a cursive word is often referred to as a ligature [108]. In this section the same word will also be used for pen-up connections corresponding to the + operation between two individual letters. The reason for sharing terminology is that with the modeling of pen-up movements as described in Section 4.2.2 the functional part of these non-symbolic shape parts are very similar. During the recognition phase they will however be treated quite differently.

Pen-down ligatures will be modeled explicitly just as other shapes portraying symbols and the connection properties backward and forward governs the template sequences they can exist in. Pen-down ligatures have shape and will thus also contribute to the shape part of the approximation distance for the best word hypothesis.

Pen-up ligatures correspond to the modeling of the lift of the pen between two templates. These ligatures will not be modeled explicitly but instead calculated dynamically through the templates before and after the lift of the pen.

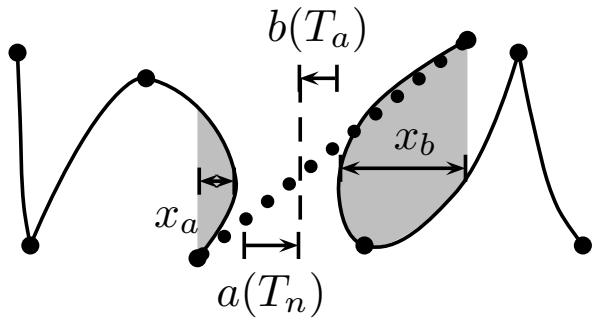


FIGURE 5.3: An example of dynamically calculated pen-up ligature (dotted line) between a template T_n of the character 'n' and a template T_a representing an 'a' according to the template properties in (5.2).

By defining the properties `after_separation` $a(T)$,`before_separation` $b(T)$ for a template T the dynamic pen-up ligature between two templates T_1, T_2 is calculated as

$$\vec{l} = (a(T_1) + x_a(T_1) + b(T_2) + x_b(T_2), y_b(T_2) - y_a(T_1)) \in \mathbb{R}^2, \tag{5.2}$$

x_a, x_b are relative x-values denoting the offsets for the reference points from where the dynamic pen-up calculation should be done. The variables y_a, y_b denote the absolute y-values of the reference points used for calculating the dynamic pen-up. A graphic view of the calculation between two templates of n and a is shown in Figure 5.3.

5.3 Segmentation Graph

This section will show how to use the segmental distance function in (4.3) to build a graph containing the template matching information. The segmentation graph for a template based system can be characterized by Definition 5.3.1.

Definition 5.3.1. A *Segmentation Graph* $\mathfrak{S}_{\mathbb{D}}(X) = (N, E)$ for a sample X with segmentation points $\mathcal{S}(X) = \{p_k\}$ constructed by a database $\mathbb{D}$, is a graph such that each node $n \in N$ corresponds to a segmentation point p_k and each edge $e^{i\rightarrow j} \in E$ to the properties of matching a template in $\mathbb{D}$ between segmentation points p_i and p_j.

For simplicity any edge ending in point p_j will be denoted by $e^{\rightarrow j}$ and with $e^{i\rightarrow}$ is implied any edge starting in point p_i. When talking about recognition between two given nodes in the segmentation graph, different recognition

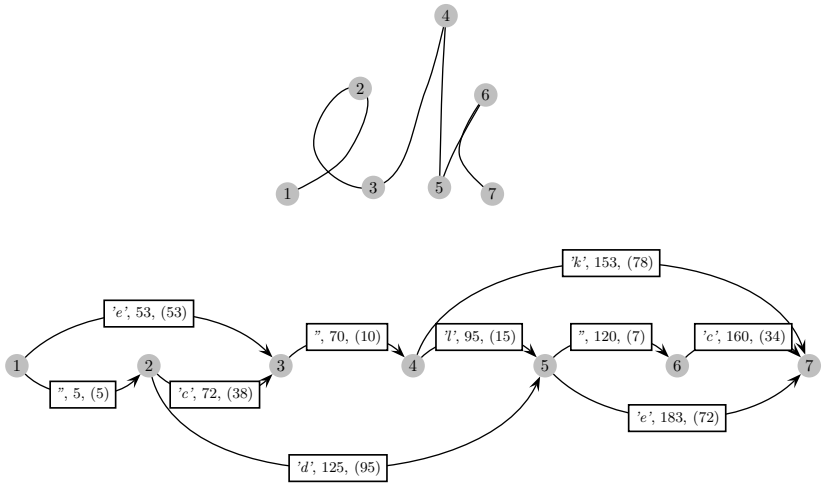


FIGURE 5.4: An example of constructing a segmentation graph for the Swedish word *'ek'*. The edge values are the best cumulative distance for that edge, given edges to start node. The template distance for the isolated part of input is included within parenthesis.

candidates can use different number of edges. The set of edges in node N_i corresponding to segmentation point p_i will be denoted by E_i. Any such sequence of edges will here be called a *path*. For notational convenience alignment between a sample X and a template sequence $\mathcal{T} = (T_1, \ldots, T_Q)$ is introduced as

$$\Phi(X, \mathcal{T}) = (\Phi_{k_1}(X, T_1), \ldots, \Phi_{k_Q}(X, T_Q)), \tag{5.3}$$

where $\Phi_k(X, T_q) = \{p_{\phi_{T_q}(j)}\}_{j=1}^{|\mathcal{S}(T_q)|}$ so that $\phi_{T_q}(j)$ gives the index of the segmentation point in $\mathcal{S}(X)$ that corresponds to segmentation point j in template T_q and k denotes the index of the first segmentation point in X that corresponds to the first segmentation point in T_q. Thus $\phi_{T_q}(|\mathcal{S}(T_q)|) = \phi_{T_{q+1}}(1)$ when $\mathfrak{O}_{\phi_{T_q}(|\mathcal{S}(T_q)|)} = \cup$, but differ when the + connects T_q, T_{q+1} and $k = \phi_{T_q}(1)$.

Example 5.3.1. *Figure 5.5 shows an example of a template sequence $\mathcal{T}$ similar to a sample X ($\mathcal{S}(X) \sim \mathcal{S}(\uplus_j T_j, \mathfrak{O}_j)$). In this example the alignment function $\Phi(X, \mathcal{T})$ becomes $(\Phi_1(X, T_e), \Phi_3(X, T_{Lig}), \Phi_4(X, T_k), \Phi_8(X, T_a))$ and for instance $\Phi_4(X, T_k) = \{p_i\}_{i=4}^{7} \subset X$.*

Definition 5.3.2. The complete graph $\mathfrak{S}^*_{\mathbb{D}}(X)$ of a sample X with segmentation $\mathcal{S}(X) = \{p_k\}_{k=1}^{|\mathcal{S}(X)|}$ and database $\mathbb{D}$ is the segmentation graph s.t. for every

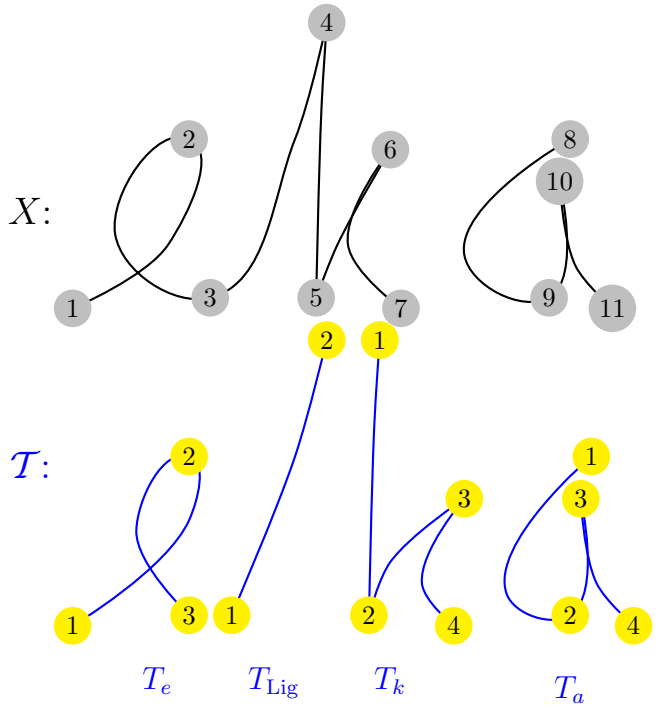


FIGURE 5.5: An example of an input sequence X and a template sequence $\mathcal{T} = (T_e, T_{\text{Lig}}, T_k, T_a)$.

template $T \in \mathbb{D}$, $\exists e^{i \to i+|\mathcal{S}(T)|-1}$, $\forall i$ such that $\mathcal{S}(T) \sim \mathcal{S}(\Phi_i(X, T))$.

Especially when introducing a limiting connectivity function as specified in Definition 5.2.1 another question arises, and that is the existence of a sequence of templates with a segmentation *matching* the input sequence. A template can only be matched to a segmentation *similar* part of input and this in conjunction with connectivity restrictions could potentially cause the situation that no sequence of templates connected with the $+, \cup$ operations is similar to input. To describe this situation the term *segmentability* of an input sample by a database $\mathbb{D}$ is introduced in Definition 5.3.3.

Definition 5.3.3. A sample X is *segmentable* by a database $\mathbb{D}$ if there is a sequence of m templates and connection operations $\{T_i\}^m, \{\mathfrak{O}_i\}^{m-1}$ s.t. $f_{\text{conn}}(T_i, T_{i+1}, \mathfrak{O}_i) = 1, i = 1, \ldots, m-1$ and $\mathcal{S}(\uplus_i^{m-1}(T_i, T_{i+1}, \mathfrak{O}_i)) \sim \mathcal{S}(X)$.

The complete graph as defined in Def. 5.3.2 will contain paths for all template sequences that matches every segment in input. In terms of computational

complexity as well as memory usage, it is normally however not a good idea to work with the complete graph. Various strategies that aim at calculating only a limited and effective set of edges are discussed in the next section.

5.3.1 Building A Segmentation Graph

When building an acyclic graph for handwriting recognition it is very common to do this with the graph search algorithm to be applied in mind. The Viterbi algorithm used for HMM and DTW can best be viewed as a graph search algorithm but it is so intricately connected to the method used to analyze the graph that the graph terminology seldom appears in those cases [129]. Since the Viterbi algorithm is very much a specialization of graph search for a sequence with probabilistic edge length values [40] the more general shortest path algorithm by Dijkstra is seen as the starting point for the graphs discussed in this chapter [33]. In the dual graph strategy proposed in this chapter the purpose of the segmentation graph is not to produce the best complete paths of templates corresponding to the input sample. Instead the objective of this first graph is to produce a graph with the most relevant templates matching to each part of input. Since a complete path may contain a varied number of edges, the edge distance in itself is inadequate for determining how well a certain edge matches to a certain part of input. Instead, the complete best path distance including the last edge will be used when comparing the qualification of edges.

There are two main strategies to consider when adding edges to the segmentation graph. Edges can be added either

segment-by-segment so that all matches against one segment in the input sample are completed before matching the next segment,

or it can be

template-by-template so that all starting segments of all valid templates are matched from each segmentation point.

Both of the strategies are presented in Algorithms 3 and 5. Although it will be shown later that the segment-by-segment strategy has certain other favorable traits one distinct difference can be made here. The segment-by-segment strategy is by definition harsher at segment level and thus risks eliminating paths upon a large matching distance on a single segment whereas the template-by-template strategy is less sensitive to peaks in segmental distance. On the other hand the template-by-template strategy can be more sensitive when the global match of the template is less accurate. Naturally the construction of the

complete graph in Def. 5.3.2 is completely independent of which of these two strategies are used since all valid edges are kept.

Additivity Requirement

The Dijkstra shortest path algorithm is a very simple but effective algorithm to dynamically calculate the shortest path through a graph [33]. This algorithm however only works if the path distance is independent on the number of nodes used and only dependent on the edge distances - but this is exactly the additivity concept as in Definition 4.4.1.

Tappert [124] also mentions additivity as a concept but in that case additivity refers to the point matching which in his case corresponds to DTW as solved with the Viterbi algorithm. Unlike template matching the additivity of the logarithm of conditional probabilities has driven the design of that algorithm [40]. Without this property recognition results would potentially be dependent on the length of the individual templates used in the complete template sequence matched.

Edge Connection Distance

The template distance function defined in Section 4.4 does not contain the features connecting *forward* from a given segmentation point and therefore connection calculations treating these features need to be treated when finding the total distance for a sequence of edges. These connection calculations are precisely the function α specified in Definition 4.4.1. In other words, in order to calculate a complete distance value for the best path backwards when adding a new edge, the best connection to edges in the starting point of the current match need to be calculated. Including the edge connection operation $\mathfrak{O}^i$ for connection two edges in point i, the edge connection distance g for edges $e^{\rightarrow i}, e^{i\rightarrow}$ can be written as in (5.4).

$$g(T_{e^{\rightarrow i}}, T_{e^{i\rightarrow}}, \mathfrak{O}^i, X) = \begin{cases} \sum_{\text{Rel}} d_C(\mathfrak{f}^{(T_{e^{\rightarrow i}},+)}, \mathfrak{f}_i^X) + \\ \qquad d_C(\mathfrak{f}^{(+,T_{e^{i\rightarrow}})}, \mathfrak{f}_{i+1}^X) & , \quad \text{if } \mathfrak{O}^i = + \\ \sum_{\text{Rel}} d_C(\mathfrak{f}^{(T_{e^{\rightarrow i}},T_{e^{i\rightarrow}})}, \mathfrak{f}_i^X), & \quad \text{otherwise} \end{cases} \tag{5.4}$$

where $\mathfrak{f}^{(T_{e^{\rightarrow i}},T_{e^{\rightarrow i}})}$ is a feature value between the templates corresponding to edges $e^{\rightarrow i}, e^{i\rightarrow}$ and $\mathfrak{f}^{(T_{e^{\rightarrow i}},+)}$ is a feature value corresponding to connection between template and the pen-up segment induced by the + operation. With

this function at hand it is now possible to formulate the segmentation graph construction algorithm as in Algorithm 3.

Algorithm 3 Segmentation Graph $\mathfrak{S}_{\mathbb{D}}(X)$, *Template by template*

```
% Initialize
E_k = ∅, k = 1, ..., |S(X)|
for k = 1, ..., max_{T∈D} |S(T)| do
    for T ∈ D, |S(T)| = k do
        if f_conn(∅, T, +) = 1 then
            Set d(e_T^{1→k}) = G(T, Φ_1(T, X))
            E_k = E_k ∪ e_T^{1→k}
        end if
    end for
end for
% Loop segmentation points
for i = 2, ..., |S(X)| − 1 do
    % Loop database templates
    for k = 1, ..., |D| do
        if E_i^k = {e ∈ E_i | f_conn(T_e, T_k, O_i) = 1} ≠ ∅ then
            Set ď = G(T_k, Φ_i(T_k, X))
            Set d(e_{T_k}^{i→(i+|S(T_k)|−1)}) = ď + min_{e∈E_k^i} d(e) + g(T_e, T_k, O^i, X)
            % Add to edges in node N_{(i+|S(T_k)|−1)}
            E_{(i+|S(T_k)|−1)} = E_{(i+|S(T_k)|−1)} ∪ e_{T_k}^{i→(i+|S(T_k)|−1)}
            % Apply beam width
            if |E_{(i+|S(T_k)|−1)}| > B_S then
                E_{(i+|S(T_k)|−1)} = E_{(i+|S(T_k)|−1)} − argmax_{e∈E_{(i+|S(T_k)|−1)}} d(e)
            end if
        end if
    end for
end for
```

Since Algorithm 3 is the well-known Dijkstra algorithm applied to graph building with an additive template distance function, it follows immediately by construction [33]. An example of constructing the segmentation graph from a simple database can be seen in Figure 5.4. Let $\mathbb{T}_{\mathbb{D}}(X) = \{\mathcal{T} | \mathcal{S}(T) \sim \mathcal{S}(X)\}$ be the set of segmentable sequences $\mathcal{T} = \uplus_j(T_i, \mathfrak{O}_i), T_i \in \mathbb{D}$ of X. Let the sequence with the smallest cumulative distance be defined as in

$$\mathcal{T}^* = \underset{\mathcal{T} \in \mathbb{T}_{\mathbb{D}}(X)}{\operatorname{argmin}} \sum_{i}^{|\mathcal{T}|} \mathfrak{G}(T_i, \Phi_{\phi_{T_i}(1)}(T_i, X)) + \sum_{i=1}^{|\mathcal{T}|-1} g(\mathcal{T}_i, \mathcal{T}_{i+1}, \mathfrak{O}_i, X). \tag{5.5}$$

Algorithm 4 Finding best path $v^{1\rightarrow|\mathcal{S}(X)|}$ in $\mathfrak{S}^*_{\mathbb{D}}$

1: Set $n = N_{|\mathcal{S}(X)|}$
2: Set $v = \{\}$
3: **while** $n > 1$ **do**
4: Set $e^{i\rightarrow n} = \text{argmin}_{e\in E_n}\, d(e)$
5: $v = v \cup e^{i\rightarrow n}$
6: $n = i$
7: **end while**

Theorem 5.3.1. *Relax the beam constraint on line 21 in Algorithm 3. Then Algorithm 4 can be used to find $\mathcal{T}^*$ in (5.5) by tracing $\mathfrak{S}_{\mathbb{D}}(X)$ produced by Algorithm 3 from node $N_{|\mathcal{S}(X)|}$.*

Proof. Clearly the best path (template sequence) needs to end in $N_{|\mathcal{S}(X)|}$ or it would not be a sequence that segments X. Then the theorem follows from the cumulative distance property of $d(e)$ in Line 17 of Algorithm 3. □

5.3.2 Limiting the Graph Size

As seen in Algorithm 3 the computations required for the construction of the segmentation graph are bounded by the number of database template $|\mathbb{D}|$ and the number of segmentation points $|\mathcal{S}(X)|$. The actual time complexity is however also dependent on the connectivity function f_{conn} and on the positions of the pen-up segments corresponding to the + operation in template space, since these factors have impact on the number of template sequences that are *similar* to the segmentation of input. In this respect a large number of unnecessary computations have already been avoided in the construction phase. The problem with estimating the bounds of a general time complexity is thus that the number of sequences that can be expanded from any given segmentation point will depend tremendously on input. It is however equally clear that an exhaustive search among all possible combinations of all templates is an extremely time consuming task.

From this perspective, putting a constraint $B_{\mathfrak{S}}$ on the number of sequences that can be expanded from a given point is thus desirable. This procedure is very common in both offline and online cursive handwriting recognition as well as speech recognition and usually goes by the name BEAM search [39]. With this limitation the number of connective distance calculations in every node would be bounded by this value (instead of $\mathbb{D}$) as seen on in Alg. 3.21. The normal edge distance calculations are bounded by $|\mathbb{D}|$ from every node

and the total number of distance calculations during construction will thus be bounded by $\mathcal{O}(|\mathbb{D}|B_{\mathfrak{S}}(|\mathcal{S}(X)|-1))$. So in order to be computationally efficient the strategy is predetermined to deal with various ways of limiting $B_{\mathfrak{S}}$. In practice, however, limiting $B_{\mathfrak{S}}$ also puts constraints on the templates that connect via f_{conn} and thus the actual average number of calculations will be significantly smaller than $|\mathbb{D}|$. The effect of varying $B_{\mathfrak{S}}$ for time complexity, memory and recognition accuracy can be seen in Table 9.9.

It is here important to stress that Algorithm 3 only works with the complete graph $\mathfrak{S}^*_{\mathbb{D}}(X)$ and the reason for this is simply the Markovian characteristic of its construction. In other words, the last edge in the best path *to* a given node is not necessarily the edge used by the best path from the same node. Therefore limiting $B_{\mathfrak{S}}$ will invalidate Theorem 5.3.1 and it can no longer be asserted that the best path through $\mathfrak{S}_{\mathbb{D}}(X)$ will correspond to the sequence of templates best approximating the input. In practice, however, this does not invalidate the use of the algorithms, but it will limit performance since sequences corresponding to the best approximation to input are likely not to be available in $\mathfrak{S}_{\mathbb{D}}(X)$ with $B_{\mathfrak{S}}$ for some samples.

5.4 Noise Modeling

As opposed to many other template matching techniques, segmented template matching requires a more explicit treatment of possible *noise* in input. By noise here, is meant, either involuntary parts of strokes such as those often appearing at the start of a stroke if the writer slips with the pen, or other segmentation artifacts. Some such artifacts, usually at the beginning and end of strokes caused by immature hardware, have previously been the focus of preprocessing techniques [47]. Generally noise will be treated as parts of shape that don't match particularly well to anything. In the recognition process this can be handled in a number of ways:

- Introducing a noise template, possibly with special matching characteristics
- Introducing a fixed distance value for a noise segment
- Using heuristics to introduce a dynamic template, which depends on input, that models some known type of noise

Noise templates The largest problem encountered when trying to model noise by a fixed template is naturally that noise can have the appearance of *anything* that is dissimilar to a real shape. This strategy is therefore very

Algorithm 5 Segmentation Graph $\mathfrak{S}_{\mathbb{D}}(X)$, *Segment by segment*

1: % Initialize
2: $E_k, I_k = \emptyset, k = 2, \ldots, |\mathcal{S}(X)|$
3: $E_1^k = \{\emptyset\}, k = 1, \ldots, |\mathbb{D}|$
4: % Loop segmentation points
5: **for** $i = 1, \ldots, |\mathcal{S}(X)| - 1$ **do**
6: % Loop database templates
7: **for** $k = 1, \ldots, |\mathbb{D}|$ **do**
8: **if** $E_i^k = \{e \in E_i | f_{\text{conn}}(T_e, T_k, \mathfrak{O}_i) = 1\} \neq \emptyset$ **then**
9: Set $\check{d} = \mathfrak{G}(T_k^1, \Phi_i(T_k^1, X))$
10: Set $d(e_{T_k^1}^{i \to (i+1)}) = \check{d} + \min_{e \in E_k^i} d(e) + g(T_e, T_k^1, \mathfrak{O}^i, X)$
11: % Add to edges in node N_{i+1}
12: **if** $|\mathcal{S}(X)| > 1$ **then**
13: $I_{(i+1)} = I_{(i+1)} \cup e_{T_k^1}^{i \to (i+1)}$
14: % Apply incomplete beam width
15: **if** $|I_{(i+1)}| > B_I$ **then**
16: $I_{(i+1)} = I_{(i+1)} - \text{argmax}_{e \in I_{(i+1)}} d(e)$
17: **end if**
18: **else**
19: Add edge $e_{T_k^1}^{i \to (i+1)}$ as in Alg. 3.19
20: **end if**
21: **end if**
22: **end for**
23: % Continue incomplete matches
24: **for** $e_{T^j} \in I_i$ **do**
25: Set $\check{d} = \mathfrak{G}(T^{(j+1)}, \Phi_i(T^{(j+1)}, X))$
26: Set $d(e_{T^{(j+1)}}^{i \to (i+1)}) = \check{d} + d(e_{T^j}) + g(T^j, T_k^{(j+1)}, \mathfrak{O}^i, X)$
27: **if** $j + 1 < |\mathcal{S}(T)|$ **then**
28: $I_{(i+1)} = I_{(i+1)} \cup e_{T_k^{(j+1)}}^{i \to (i+1)}$
29: Apply beam as in Line 14
30: **else**
31: $E_{(i+1)} = E_{(i+1)} \cup e_T^{(i+1-|\mathcal{S}(T)|) \to (i+1)}$
32: Apply beam as in Alg. 3.19
33: **end if**
34: **end for**
35: **end for**

unlikely to succeed well. Since the aim of the noise distance is to enable stepping

past segmentation points without matching templates from the database using a fixed distance is probably a more suitable solution.

Fixed noise distance A weakness of the fixed distance strategy is that the acceptable distance values for noise depend on how well modeled other templates in the database are. When the average distance values become smaller as variation in database modeling increases this also implies that the average value of a non-match decreases, thus affecting the matching properties of noise with a fixed distance. The intrinsic robustness against various forms of noise has been one of the key arguments for using network based methods without explicit segmentation. Since the sliding window principle of such methods corresponds to a more global template match these are less sensitive to small noise. In one realization, the detection of *nil* letters in the detection matrix could also correspond to matching noise [101].

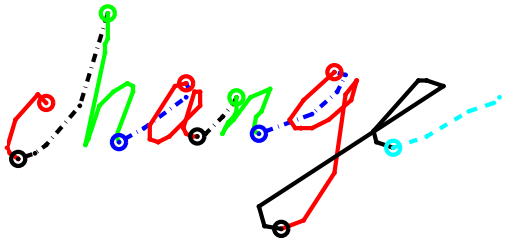
Dynamic noise modeling This has been the choice of method for the implementations covered in this thesis and can be interpreted as a combination of the previous two strategies. In this method noise is modeled by some observed heuristics, but instead of using a fixed template, the template adapts to input in some predetermined way. This will imply that those particular observations in input will match fairly well to input whereas other types of noise not fitting into the heuristic model will have large matching distance. To balance these two possibilities and assuring that noise is indeed a database element that does not match particularly well to anything, the matching distance can be truncated as in (5.6).

$$\mathfrak{G}(T_{\text{noise}}, X) = \begin{cases} d_{\min}, & \text{if } \mathfrak{G}(T_{\text{noise}}, X) < d_{\min}, \\ \mathfrak{G}(T_{\text{noise}}, X), & \text{if } d_{\min} \leq \mathfrak{G}(T_{\text{noise}}, X) \leq d_{\max}, \\ d_{\max}, & \text{otherwise.} \end{cases} \tag{5.6}$$

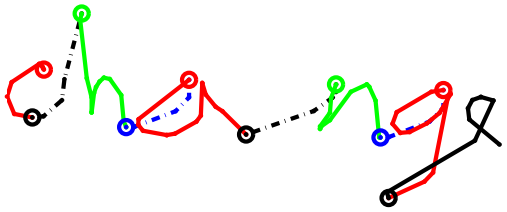
When matching noise to the segmentation graph there is also another consideration, that of adding noise as separate edges or incorporating noise segments in the database templates. The choice of strategy is here mostly dependent on the edge addition strategy. For a template-by-template strategy it is natural to add noise as separate edges connecting to everything, i.e.

$$f_{\text{conn}}(T_{\text{noise}}, T, \mathfrak{O}) = f_{\text{conn}}(T, T_{\text{noise}}, \mathfrak{O}) = 1, \forall T \in \mathbb{D}, \mathfrak{O} \in \{+, \cup\} \tag{5.7}$$

since one would otherwise have to evaluate a very large number of noise combinations with each template. For the segment-by-segment strategy, however, noise can be added either as a new edge or to an existing edge in the set of incomplete edges in each node.



(a) Sample



(b) Approximating template sequence

FIGURE 5.6: An example of the recognition of a sample of the word *change*. The aligned parts of the input sample corresponding to the best matching template sequence shown concatenated in Figure 5.6b is shown in Figure 5.6a with matching colors.

5.5 The Recognition Graph

The aim of the segmentation graph is to quickly conduct all relevant shape comparisons and store this in a compact format. The segmentation graph also allows easy access to the best path backward from any given node. Unfortunately however, this is not the case for the second or n-best results. Simon [71] has studied the application of path algebras for fast retrieval of such results from a segmentation graph, but the distance values in that case are multiplicative (i.e. probabilities) instead of additive as in the template matching case

and rely on that they are independent of each other. Unfortunately there is no simple way to retrieve such candidates from the complete graph $\mathfrak{S}^*_{\mathbb{D}}(X)$, and to make sure that elements are returned in order of distance, only lossless exhaustive search methods can be applied. Not only may the variation from the best sequence of matching templates alter the segmentation (and thus not correspond to just changing one edge), but the connection distance $g(e^{\rightarrow i}, e^{i\rightarrow})$ is just the *best* connection distance and thus needs to be recalculated for every path except the best.

The pragmatic approach taken in this thesis to attack the problem has been named the *dual graph* approach since it involves a second graph - *the Recognition Graph*. In practice this means that beam searching techniques will be applied in two levels - first to prune shape matching results and then to expand partial results to complete word hypothesis. This second graph produces the actual recognition hypothesis and thereby correspond to the actual path expansion in other work. In this chapter this will basically amount to summing the edge distance values, but further merits of this structure will be shown in subsequent chapters. In other work, averaging has been proposed as a procedure for summing sequential distance values [85]. In this implementation this has not been considered to be a good option since it fundamentally changes the additivity principles.

Definition 5.5.1. A *Recognition Graph* $\mathfrak{R}_{\mathbb{D}}(\mathfrak{S}, X) = (N, V)$ for a segmentation graph $\mathfrak{S}$ of a sample X with segmentation points $\{p_{ij}\}$ constructed by a database $\mathbb{D}$, is a graph such that each node n corresponds to a segmentation point p_k and each edge $v^{1\rightarrow j}$ to the properties of matching a sequence of templates in $\mathbb{D}$ between segmentation points from the beginning to j.

The recognition graph uses both the matching information in a segmentation graph as well as the corresponding sample and database to produce a set of symbols that, in view of the distance function used, are the most similar to the input sample.

5.5.1 Recalculated Pen-up Movements

When pen-up connections are calculated in the segmentation graph only the best edge can be considered and naturally only the last segment can therefore be taken into account. Since the aim of the matching process is to find the best approximating sequence of templates in the database the match of noise should in a modeling respect be disregarded. For this reason, when matching noise, one noise $\rightarrow$ noise segment connection distance should be replaced by the correct template $\rightarrow$ noise segment. This is done by remodeling a pen-up

translation between strokes in the input sample so that the starting point of the compared pen-up movement corresponds to the lift of the pen between two actual template matching edges as shown in Figure 5.7. Depending on the segmentation graph such recalculations may or may not be possible due to the Markovian characteristic (i.e. there may be a multitude of preceding noise edges), but the modeling of complete paths always enables such recalculations in the recognition graph.

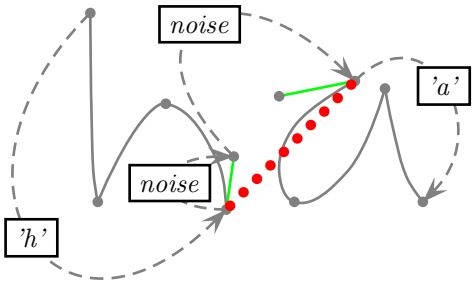


FIGURE 5.7: An example of pen-up recalculation (dotted line) in sample due to matched noise edges.

5.5.2 Building A Recognition Graph

The major computations required when adding the edges in the segmentation graph to obtain path candidates for the recognition graph are those conducted for the connection distance. The beauty of the additivity concept explored in this thesis is how the complete path hypothesis propagation in its initial form basically consists of expanding the compact segmentation graph into a Trie structure [41] with nodes sorted by their complete distance as seen in Figure 5.8. Ad hoc summation procedures for ranking complete paths as common for most other methods is thereby avoided [89].

5.5.3 Connection Distance Calculation

For each new edge from the segmentation graph added to the recognition graph a new pen-up movement or concatenation distance calculation is conducted. The connection distance of (5.4) calculated for the segmentation graph only accounts for the connection corresponding to the best path backward for the given edge. To get the real connection value when adding another arbitrary edge from the segmentation graph to a path in the recognition graph this

connection distance must therefore often be recalculated. Furthermore it is possible to incorporate better pen-up modeling information since the complete path from the first node is fixed for the elements in the recognition graph. For this reason the path distance D_v for a set of edges $e_1^{1\rightarrow j_2}, \ldots, e_n^{j_n \rightarrow j_{n+1}}$ can be recursively defined as

$$D_v(v^{1\rightarrow j_k}) = D_v(v^{1\rightarrow j_{k-1}}) + d(e^{j_{k-1}\rightarrow j_k}) + \hat{g}(T_{e^{j_{k-2}\rightarrow j_{k-1}}}, T_{e^{j_{k-1}\rightarrow j_k}}, \mathfrak{O}^{j_{k-1}}, X), D_v(v^{1\rightarrow j_1}) = d(e^{1\rightarrow j_1}), \quad (5.8)$$

where $\hat{g}$ is the distance in (5.4) but with corrected path pen-up modeling as described in Section 5.5.1.

Algorithm 6 Creation of the Recognition Graph $\mathfrak{R}_{\mathbb{D}}(X)$

```
% Initialize
Set V_i = {}, i = 1, ..., |S(X)|
for k = 1, ..., S(X) do
    Set V_k = E_k ⋂ {e^{1→}}
end for
% Expand Trie structure
for k = 1, ..., S(X), r = 1, ..., |E_k| do
    Let i_k be the start node of edge e_r ∈ E_k
    for q = 1, ..., |V_{i_k}| do
        Set v = (v_q^{1→i_k}, e_r^{i_k→k})
        Set D_v(v) according to (5.8)
        V_k = V_k ∪ v
        % Apply Beam width
        if |V_k| > B_ℜ then
            V_k = V_k − argmax_{v∈V_k} D_v(v)
        end if
    end for
end for
```

5.5.4 Time Complexity Considerations

The time complexity for building a recognition graph depends more than anything on two factors: 1) the number of edges in each node of the segmentation graph, 2) the number of paths in each node of the recognition graph as limited by the recognition graph beam width $B_{\mathfrak{R}}$. For the segment-by-segment strategy, naturally also the incomplete beam width B_I has impact on the time complexity.

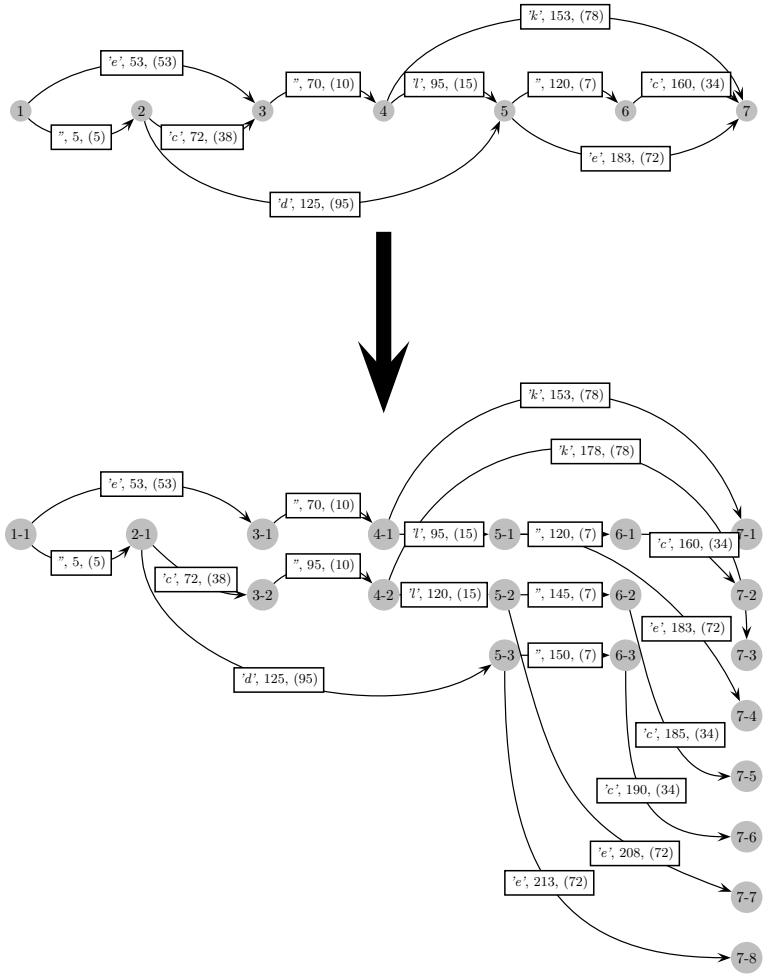


FIGURE 5.8: A graphic view of expanding the matching information in the segmentation graph as in Figure 5.4 into the Trie structure of the recognition graph.

5.6 Preprocessing Parameters

Another potentially powerful property of template matching with explicit segmentation is the ease with which global shape parameters can be localized. With global shape parameters is here implied reference line position, scaling and other global features often normalized at a preprocessing stage [47]. Se-

quential template matching opens up the possibilities to assess these parameters locally and adding a suitable distance component to $g, \hat{g}$ for modeling local changes in these global features. The relative length ratio in Table 4.1, is such a feature as differences in length only has a one-segment memory and thus the scaling cost of a template equals to the scaling of the first segment of the template.

CHAPTER 6

Delayed Strokes And Stroke Attachment

It is not worth an intelligent man's time to be in the majority. By definition, there are already enough people to do that.
G. H. Hardy

In Chapter 5 it is shown how the template distance function defined in Chapter 4 can be used to create a recognition system for connected character input. However, this strategy makes an oversimplifying and sometimes incorrect assumption in that all coordinates belonging to a character are written together. This is certainly not true for western cursive writing nor Arabic, scripts which both contain diacritic marks commonly written after all other letters have been entered.

6.1 Introduction

An obvious problem that on-line recognition methods encounter when applied to connected characters is the presence of *delayed* strokes. There are various strategies for coping with this problem and most solutions are closely related to the type of recognition method used. Recognition of Chinese characters is often compared to the task of recognizing cursive words for other alphabets [54]. In this field stroke-order invariance is a thoroughly studied phenomenon which is handled considerably well with Dynamic Programming algorithms [136]. For the recognition of on-line cursive script the problem has not received as much attention and diacritical removal with simple heuristics seems to be the most common approach although some strategies use explicit stroke reordering and insert probable diacritic strokes at a spatially decided point in time [16, 107, 124]. Often the removed information is used to boost confidence in the final hypothesis evaluation in combination with a dictionary look-up [50, 90, 104]. In recent work a more frequent approach is to convert the removed stroke into a `hat`-feature added to the features of the spatially underlying strokes as introduced by Guyon et al. [55, 68, 101]. Since this problem is particular to on-line methods treating the sequence of points chronologically rather than space-oriented, it is possible that the merits of adding off-line technology may be somewhat owed to the intrinsic stroke-order invariance of the off-line recognition setting. Nevertheless some off-line methods also try to find an explicit association of diacritics with base shapes [30, 37]. From a problem description standpoint the delayed strokes introduce a third complexity layer to recognition of connected characters by adding the ambiguity of each entered stroke can be a modifier of already entered characters. The possible confusion cases will now be:

- Segment subpart misinterpretation - this corresponds to recognition errors in conventional isolated character recognition, see Chapter 4.
- Segmentation errors - this corresponds to an error in recognized segmentation. The implied letter borders in the recognition results do not cor-

respond to the correct set of letter borders in input, see Chapter 5.

- Diacritic errors.

The last set of new *Diacritic errors* and a proposed algorithm for dealing with such is the focus of this chapter. The proposed algorithm avoids the problem of previously proposed systems with heuristically determined diacritic removal and insertion schemes and instead match diacritic parts of characters in the same way that other templates are treated. Previous work in trying to include diacritics dynamically into the recognition process involves successive association of delayed diacritics to letters earlier in the hypothesis through a best-first paradigm [103]. The algorithm proposed in this chapter will enable a complete beam search of the combinatorial possibilities to associate multiple diacritics with one hypothesis as well as introduce a way to account for unmatched information in a *fair* manner.

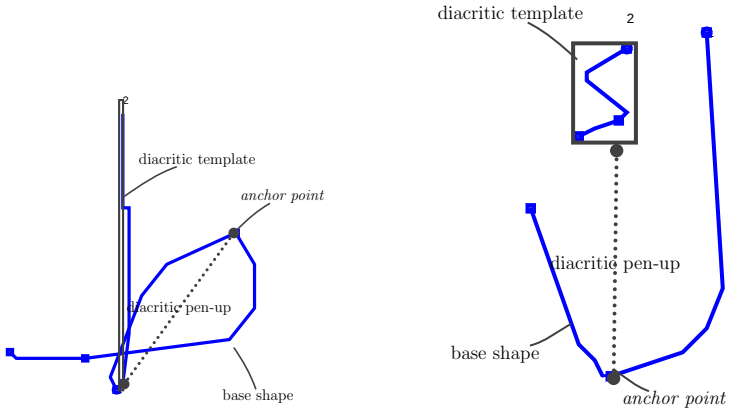
In short diacritic errors can be said to belong to one of three types

1. Letter confusion - a diacritic mark has been matched as a normal letter or vice versa.
2. Association error - a correctly recognized diacritic - such as a point - has been associated as a modifier to the wrong part of input.
3. Diacritic confusion - a diacritic mark has been interpreted as the wrong type of diacritic mark.

Here the diacritic confusion corresponds to shape matching errors such as those that would arise in normal recognition with a template distance function on isolated characters. The other two confusion cases are however particular to how the writing of a diacritic is modeled compared to previous writing as shown in Section 6.4.

6.2 Diacritic Modeling

Diacritic strokes are defined as the small strokes such as accents and dots added to some letters to distinguish between similar words. In the template database presented in this system the diacritic components and the letters they modify are distinctly separated. The actual writing of diacritics may, just like the ordinary set of letters, vary considerably between writers. In order to simplify modeling, the preliminary assumption that a delayed stroke is added one at a time is made in Definition 6.2.1. In cases when a diacritic is composed of multiple strokes this means that each such stroke will be treated separately.



(a) 'Tah' template (b) 'Kaf' template

FIGURE 6.1: A plot of the pen-up modeling for attaching diacritics to templates.

Definition 6.2.1. A *diacritic shape* is a part of a character that is written distinctly separated into a single stroke.

Templates corresponding to letters without diacritic marks will here be called *base shapes*. All letters with variations in diacritic marks but with the same base shape will share this template. The modeling of the position of the diacritic relative the base shape is done by associating a *pen-up* movement (i.e. translation) from a designated point in the template, called the *anchor point* as seen in Figure 6.1. Let ϱ_a^T denote a pen-up attached to point index a of template T. As diacritic marks appear both below, above and even in the middle of characters, the endpoint of this movement may correspond to min, max or center value of the diacritic stroke. Two samples with pen-up movements in opposite directions connected to the vertical minimum of the diacritic template are shown in Figure 6.1.

In the case when the anchor point corresponds to the last point in the template and the diacritic in input is the next stroke after the base shape, recognition will correspond simply to connected character recognition as described in Chapter 5 of a base shape template and a diacritic template. For diacritics in connected writing however, this is almost never the case and therefore a more elaborate modeling as described in Section 6.3 is required to make sure that previous assumptions on additivity used for the connected character recognition are sensible.

6.3 Pen-up Attachment

The system for recognition of on-line sequences of connected characters presented in previous chapters is Markovian in the sense that recognition of a new part of a sample only depends on the recognition of the immediate predecessor. There is no way to associate delayed strokes to base shapes. This section describes a new stroke attachment paradigm to cope with this problem. In the discussion in previous chapters it has been assumed (although not explicitly mentioned) that characters in the connected character sequence in input follow a *chronological order*. This means that the pen-up segments discussed in Section 4.2.2 can be used directly for recognition. In view of the frame energy concept of Section 4.1 each such pen-up movement corresponds to the bar connecting two strokes. To provide a suitable basis for good recognition results these pen-up movements need to be stable features in input with as small variance as possible, and from this standpoint the chronological pen-up modeling is not always a good choice. Consider the chronological pen-up segments of the Arabic word حفظ in Figure 6.2 for instance. It is quiet clear that some of the pen-up movements in input corresponding to the diacritic attachment depends on the width of other characters. To improve this situation two new concepts for modeling stroke connections in input are introduced and the corresponding pen-up attachments are shown in Figure 6.2.

Recalculated pen-up This term has already been introduced in Section 5.5.1 and corresponds to changing the modeling of the attachment point between base shape templates.

Diacritic pen-up This remodeling is often accompanied by a recalculated pen-up since it only models the pen-up movement from the anchor point in the template to the diacritic template and never from the diacritic template.

6.3.1 Input Segmentation

Clearly as seen in Figure 6.2 the proposed remodeling schemes only modify the pen-up movements between strokes. How does this impact on the task of comparing and ranking the *approximating* sequences of templates to input? Since each pen-up remodeling corresponds to another set of pen-up segments in X it is natural to introduce a remodeling function $\mathcal{X} : \mathbb{S}(X) \rightarrow \mathbb{S}(X)$, where the space $\mathbb{S}(X)$ consists of all variations in pen-up modeling of identical segmentation points $\mathcal{S}(X)$ so that even though $\mathcal{S}(X) = \{p\}$ is constant, the resulting segments $\{\Lambda\}_{i=1}^{|\mathcal{S}(X)|-1}$ will differ. In the view of the optimization

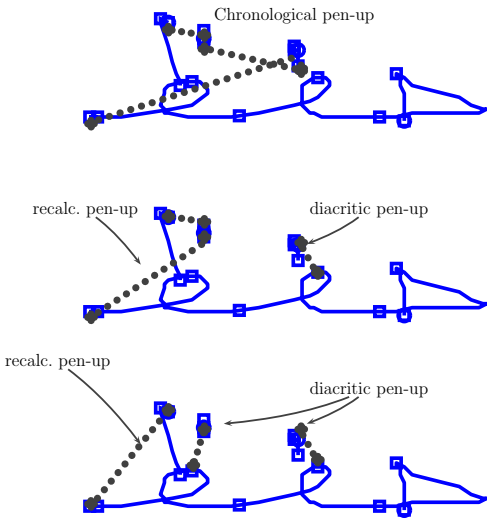


FIGURE 6.2: Some examples of pen-up reorderings and reassociations in a sample containing diacritic strokes for the Arabic word حفظ.

problem of finding the best approximating template sequence. This will add $\mathcal{X}$ as a subject of minimization and the best approximating sequence in this respect will now be the solution to

$$\underset{\mathcal{T},\mathcal{X}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \mathfrak{G}(\mathcal{T}_i, \Phi_{\phi_{\mathcal{T}_i}(1)}(\mathcal{T}_i, \mathcal{X}(X))) + \sum_{i=1}^{|\mathcal{T}|-1} \hat{g}(\mathcal{T}_i, \mathcal{T}_{i+1}, \mathfrak{O}_i, \mathcal{X}(X)). \tag{6.1}$$

6.4 Dynamic Treatment Of Diacritic Strokes

In view of (6.1) it is clear that Algorithm 6 needs modification in order to handle the problem of finding the best template sequence. The actual pen-up remodeling implications to template distance is easily handled by the recalculated distance function in (5.4), the chronological dispersion of diacritics from their base shapes however, will cause problems. The reason for this is the sharing of diacritic templates for the base shapes. Since a base shape may correspond to several different symbols solely differing by their diacritic variations, not

including the diacritic information in matching the base shape may cause a combinatorial explosion of possible symbol sequences and without the diacritic matching information each such symbol sequence will have the same distance. There are a few intuitively direct ways to attack this problem.

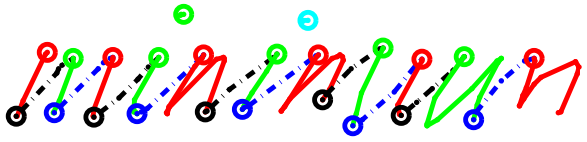
1. Insert heuristically designated delayed strokes into a chronological point sequence [16, 124].
2. Handle symbol ambiguity at a path level to avoid the combinatorial explosion of symbol sequences. This corresponds to matching completely without diacritics only applying such knowledge in a postprocess to discriminate between ambiguous words [36, 50, 90, 104].
3. Include the *future* diacritic matching information in the path calculation. This strategy is the most common for HMM and neural network based methods where all delayed strokes are removed by simple heuristics and incorporated into the feature sequences through the *hat* feature [55, 68, 101].

Only the second method is covered in this thesis, and this choice is motivated by two strong arguments. Firstly, handling ambiguity at the symbol sequence level would still not aid in taking early decisions as to whether a complete template with a diacritic matches well. Thus it would force the recognition graph to store more paths in every node to handle sequences that would be very unlikely given the complete input with diacritics. An example of this in English for instance is the recognition of a sample of the word *minimum* as shown in Figure 6.3. Since the letter 'i' without a dot here matches well to parts of 'n' and 'm' the recognition graph will have to keep track of lots of paths that have a very large matching distance if the required diacritic matching were to be included. An example of such a sequence that would be unlikely given a priori diacritic associations is the segmentation corresponding to template sequence *'iciininciun'* shown in Figure 6.3b. The other argument for including diacritic matching information in the path calculations involves the interaction with lexical knowledge presented in Chapter 7.

6.4.1 Diacritic Matching

The diacritic templates themselves are not treated as ordinary base shapes and the main difference are the implementational aspects of Definition 6.2.1. This definition implies that matching of diacritic templates to input should be restricted so that diacritic edges containing the match of one diacritic always start at the first node of a stroke and ends in the last node of the same stroke.

(a) Sample



(b) Segmentation from template sequence

FIGURE 6.3: A sample of the cursive word *minimum* demonstrating the typical problem with diacritic association, here shown with a different template segmentation.

Thus only the best path containing precisely one diacritic is calculated for each stroke j, i.e.

$$\mathbb{S}_j = \underset{v^{i_1 \to i_{|\mathcal{S}(X^j)|}}}{\operatorname{argmin}} \; D_v(v^{i_1 \to i_{|\mathcal{S}(X^j)|}}), \tag{6.2}$$

where $|\mathcal{S}(X^j)|$ is the number of nodes in stroke j, and $i_1, i_{|\mathcal{S}(X^j)|}$ denote the indexes of the first and last segmentation points of stroke j respectively. The best path for a diacritic template T in (6.2) will be referred to as a diacritic edge $e_T^{X^j}$. Let $\hat{d}(e_T^{X^j})$ denote the distance for the diacritic edge of template T matched to stroke j. To denote the category of templates matched from the template database $\mathbb{D}$ the notation $\mathbb{D}_D, \mathbb{D}_B$ for the diacritic and base shape templates is introduced respectively. The set $\mathbb{S}_i$ can be calculated using Algorithm 6 with $\mathbb{D} = \mathbb{D}_D$ and with restrictions on symbol sequences so that it

contains precisely one edge corresponding to a diacritic template (but possibly multiple noise edges).

The best paths at stroke level for non-diacritic matches are also stored thus providing a set of conditional best stroke-level paths $\mathbb{S}_i|_{\mathbb{D}=\mathbb{D}_B}, \mathbb{S}_i|_{\mathbb{D}=\mathbb{D}_D}$.

6.4.2 Diacritic Attachment

The diacritic stroke paths are added to the segmentation graph as extra edges between start and end-point but with specific connection properties which also involve stroke pen-up reattachment as seen in Figure 6.2. Unlike the connection properties in (5.1) the connection properties for a diacritic stroke path depend on the complete previous path and the diacritic properties of its matched base shape template. Let $\mathfrak{D}_a(e)$ denote a diacritic property with anchor point a of the base template for the symbol corresponding to the edge e. The distance for a complete diacritic edge match is complemented by the matching of the diacritic pen-up as

$$d(e_{T,a}) = \hat{d}(e_T) + \breve{g}(\varrho_a^T, T_e, \mathcal{X}_{\varrho_a^T}(X)). \tag{6.3}$$

Now every path v will contain not only the set of matched edges $\{e_k\}$ but possibly also a set of *unmatched* templates $\{\mathfrak{D}_a(e)\} = U$. With this notation in mind the connective function in (5.1) can be extended to act on diacritic edges e corresponding to template T and path v corresponding to template sequence $\mathcal{T} = \{T_j\}_{j=1}^{|v|}$ as

$$\hat{f}_{\text{conn}}(v, T, \mathfrak{O}) = \begin{cases} f_{\text{conn}}(T_{|v|}, T, \mathfrak{O}), & \text{if } T \in \mathbb{D}_B \\ 1, & \text{if } T \in \mathbb{D}_D, T \in U(v), \mathfrak{O} = + \\ 0, & otherwise \end{cases}. \tag{6.4}$$

Limitations to the diacritic modelings searched introduces a diacritic beam $B_{\mathfrak{D}}$. Here this beam is defined as the size of the set of diacritic edges $\{e_{T,a}^{X^j}\}$ for each stroke j. This set can be ordered according to the complete diacritic edge distance $d(e_{T,a}^{X^j})$ and limited so that only the $B_{\mathfrak{D}}$ edges with smallest distance are kept. Since the set of possible stroke reattachments $\mathcal{X}$ change only the pen-up movement between strokes, matching characteristics will differ only after a pen-up segment has been passed during the construction of the recognition graph. Thus for every such path expansion (adding an edge after pen-up segment) the set of matching reattachments going forward will depend on if the chronological, some recalculated pen-up or diacritic pen-up was used for the pen-up

matching. After the final pen-up has been matched the particular reattachment used will be unique. In order for the previous discussions of approximating best template sequences with graph techniques covered in Chapters 4 and 5 to be valid the compared input sample must remain the same. This is no longer true for stroke reattachments since each reattachment alters the pen-up segments. In view of the Frame Energy concept the stroke attachments can be seen as a way to model input and in this sense the comparison of path scores resulting from different stroke attachments will correspond to comparing different graphs each corresponding to a unique model as seen in Figure 6.6. The corresponding recalculated connection distance will also depend on the attachment and takes the form of

$$D_v(v^{1\to k}) = D_v(v^{1\to j}) + d(e^{j\to k}) + \hat{g}(T_{|v^{1\to j}|}, T_{e^{j\to k}}, \mathcal{X}_{v^{1\to j}}(X)), \qquad (6.5)$$

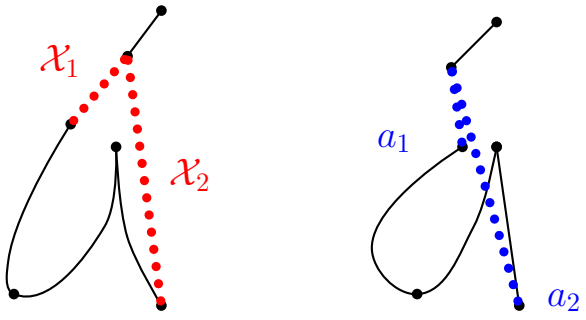
where $\mathcal{X}_{v^{1\to j}}(X)$ is the set of stroke attachments that contain the pen-up segments matched in $v^{1\to j}$. With this notation at hand, there are a number of issues to attend to before presenting the dynamic diacritic recognition algorithm. First of all is the validity in comparing different version of input. In terms of finding the template sequence and stroke attachment fulfilling (6.1) this proposed strategy simply corresponds to sequential optimization of the discrete variables $\mathcal{X}$ and $\mathcal{T}$ and thus the solution will be found with an exhaustive search. The proposed algorithm will merely bound this search in time.

6.4.3 Attachment Point Invariance

In the introduction of the additivity concept it is stated that template sequences should be treated *fairly* i.e. independently of their segmentation when matched. It is possible to define a similar property for the stroke attachments.

Definition 6.4.1. A pen-up segment distance function is *attachment point invariant* if its output depends only on the relative position of the pen-down parts on either side of the pen-up and not on the actual pen-up segment itself.

It is clear that the conventional distance function with the frame features presented in Section 4.2.1 will produce different distance values for the case seen in Figure 6.4 as the difference in angle is very small for the case of matching the template with anchor point a_2 to attachment $\mathcal{X}_2$ and quite large when comparing the template with anchor point a_1 to attachment $\mathcal{X}_1$. Lacking this property the matching properties for diacritical marks will depend not only on their position (including the diacritic pen-up as seen in Figure 6.1) but also on their anchor points. Due to the angular feature a harsher handling will be



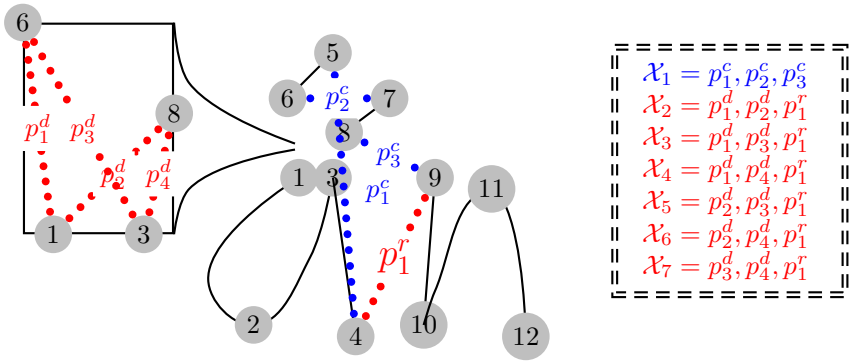
(a) Sample (b) Diacritic template

FIGURE 6.4: An example of the attachment point invariance problem. Even though the two attachment points a_1, a_2 result in identical templates, the template distance to the corresponding stroke reorderings $\mathcal{X}_1, \mathcal{X}_2$ of sample in 6.4a may differ due to the differing differences between pen-up segments.

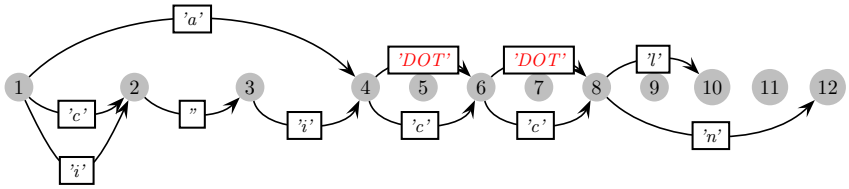
achieved by minimizing the length of the diacritic pen-up segments as seen in the comparison of the anchor points a_1, a_2 in Figure 6.4.

6.4.4 Graph Handling

Not only does the pen-up recalculation dependency in $\hat{g}$ of (6.5) imply that paths corresponding to different reattachments of the input sample compete in the same graph. In order to suppress the size of the beam when building the recognition graph, i.e. the number of kept paths in each node, some kind of preliminary attention must be given to the set of *unmatched* edges $U(v^{1\rightarrow})$. To accomplish this, the *branch-and-bound* strategy from combinatorial optimization is introduced. For each path it is possible to compute the best possible path forward by utilizing the best stroke-level paths as determined by (6.2) and further conditioned by U. This way the list of best paths in Algorithm 7 can always be sorted by the best possible distance from start to end, albeit somewhat optimistic. This combined distance is referred to as the *sorting distance* D_{sort} as defined in (6.6).



(a) Sample



(b) Simple segmentation graph

FIGURE 6.5: A graphic display of Algorithm 7 showing the gradual decrease in number of stroke attachment models evaluated during graph expansion.

$$D_{\text{sort}}(v^{1\to k}, U(v^{1\to k})) = D_v(v^{1\to k}) + \sum_{m=s+1}^{S} \min_{e^{X^m} \in U(v^{1\to k})} (d(\mathbb{S}_m)|_{\mathbb{D}_B}, d(e^{X^m})), \quad (6.6)$$

where S are the number of strokes in input and s is the index of the stroke containing node k.

Another issue is what to do with the set U when no suitable diacritics are found. The simple intuitive approach to this case taken in this thesis is to introduce a penalty to each path as soon as it becomes clear that an element of $U(v^{1\to})$ is impossible to match given the remainder of the input sample. This is called

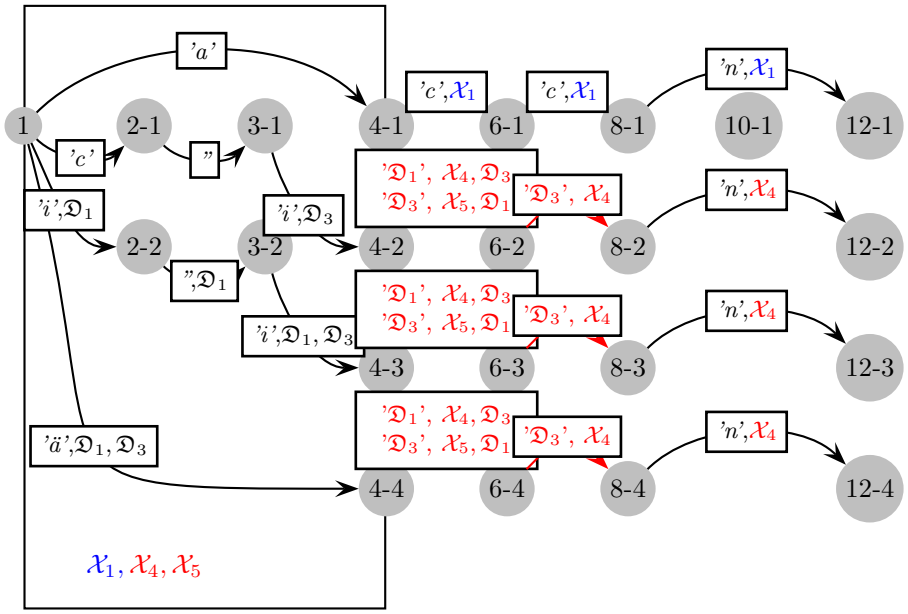


FIGURE 6.6: Recognition graph expansion of the segmentation graph in Figure 6.5b.

missing diacritic punishment and is added as $f_{\text{pen}}(U(v^{1\rightarrow}))$ to (6.6).

Ambiguous Path Expansion

Another difficulty with the unmatched edges is the embedded ambiguity which result in that the path expansion with a diacritic edge is ambiguous (unlike the base shape case as seen in Figure 5.8. The ambiguity is caused by two factors:

- One template with diacritics can have several alternative ways of doing this, such as two dots converted into one 'two-dot' stroke.
- If there were several unmatched edges corresponding to the same diacritic template, the same diacritic edge can be added in different places for the same sequence of edges resulting in identical paths apart from stroke attachment. A symbol requiring two dots can for instance use either stroke for each of the dots.

The first ambiguity implies that the same path can have several alternative sets of unmatched edges denoted by $\mathbb{U}(v^{1\rightarrow}) = \{U_m(v^{1\rightarrow})\}_{m=1}^{|\mathbb{U}(v^{1\rightarrow})|}$, and thus contain many *alternative futures*. The second ambiguity is not an ambiguity in a real sense since the stroke attachments differ and thus the distance values. On the other hand with respect to the performance of the beam search conducted by the recognition graph algorithm, there should not be multiple paths in the graph corresponding to exactly the same edges. To cope with this problem a secondary beam search within the set of identical edges sequences is performed by simply limiting the set for every new diacritic edge (corresponding to such an expansion), so that at most the best B_U different stroke attachments $\mathcal{X}$ are stored per sequence as seen in Figure 6.6 and in Algorithm 7.24. Note also that each such stroke attachment will carry its own set of unmatched edges which is thus also conditioned by the stroke attachment $\mathbb{U}(v^{1\rightarrow}, \mathcal{X})$.

Time Complexity Considerations

The two time consuming part of the added diacritic handling is searching for attachment points whenever a new diacritic edge is encountered and recalculations of the sorting distance. Each stroke attachment alternative carries its own set of unmatched edges and each of these must be searched for possible ways to attach the new edge. Similar operations are performed for determining the sorting distance as the minimum stroke distance with a diacritic in the future will depend on if that diacritic can attach to an attachment points amongst the unmatched edges. As seen in the experiments conducted in Section 9.4.1 the memory and speed effect of the two diacritic related beams $B_{\mathfrak{D}}$ and B_U is marginalized by the other beam sizes in the graph structures.

Algorithm 7 Constructing $\mathfrak{R}_{\mathbb{D}}(\{X^i\}_{i=1}^S)$ with dynamic diacritic handling

Create a segmentation graph according to Algorithm 3 or 5
Determine $\mathbb{S}_i|_{\mathbb{D}=\mathrm{T}_B}, i = 1, \ldots, S$
for $r = 1, \ldots, S$ **do**
    $k = \sum_{j=1}^{r} |\mathcal{S}(X^j)|$ *% k is the index of the last node in stroke r*
    Set $E_k = E_k \cup \{e_i^{X^r}\}_{i=1}^{B_{\mathfrak{D}}}$ *% Add diacritic edges*
end for
% Initialize recognition graph
Set $V_k = \{e^{1\to k} | e^{1\to k} \in E_k\}, k = 1, \ldots, |\mathcal{S}(X)|$
% Loop strokes
for $r = 1, \ldots, S$ ***do***
    % Loop segmentation points per stroke
    for $j = s_1, \ldots, s_{|\mathcal{S}(X^r)|}$ ***do***
        for $k = 1, \ldots, |E_j|$ ***do***
            Set i_k *to start node of* $e_k^{i_k\to j}$
            if $i_k < s_1$ ***then*** *% This is a new stroke*
                Update $\mathcal{X}_v(X), v \in V_{i_k}$
            end if
            for $l = 1, \ldots, |V_{i_k}|$ ***do***
                if $\hat{f}_{conn}(v_l^{1\to i_k}, T_{e_k^{i_k\to j}}, \mathfrak{O}_{i_k}) = 1$ ***then***
                    Calculate $D_v(v_{lk}^{1\to j})\Big|_{\mathcal{X}}, \forall \mathcal{X} \in \mathcal{X}_{v_{lk}^{1\to j}}(X)$
                end if
                Update $\mathbb{U}(v_{lk}^{1\to j}, \mathcal{X}_{v_{lk}^{1\to j}}(X))$
                Set $D^*_{sort}(v_{lk}^{1\to j}) = \min_{\substack{\mathcal{X}\in\mathcal{X}_{v_{lk}^{1\to j}}(X) \\ U\in\mathbb{U}(v^{1\to j},\mathcal{X})}} D_{sort}(v^{1\to j}, U)$
                Limit $|\mathbb{U}(v_{lk}^{1\to j}, \mathcal{X}_{v_{lk}^{1\to j}}(X))| \le B_U$
                $V_j = V_j \cup v_{lk}^{1\to j}$
                if $|V_j| > B_{\mathfrak{R}}$ ***then***% *Apply beam width*
                    $V_j = V_j - \mathrm{argmax}_{v\in V_j} D^*_{sort}(v)$
                end if
            end for
        end for
    end for
end for

CHAPTER 7

Application of Linguistic Knowledge

Acorns were good until bread was found.
Francis Bacon

Apart from the superposed complexity of segmentation and diacritic association problems, another difficulty with connected character recognition is that the nature of such writing is prone to producing less distinct individual letters within the word. This type of shape degradation makes purely shape based recognition insufficient for the task of recognizing connected cursive script and all state-of-the-art recognition methods have added linguistic components to the modeling in order to achieve acceptable word recognition results [104].

7.1 Introduction

It is commonly known that humans possess an exceptional ability to read even severely misspelled text and given just the first and last letter of every word in place even phrases such as

'*Tihs is an eaxxplme of sverelly dgrdaded txeet!*'

can be correctly deciphered. The reason for this is of course that humans use more information than just the visible sequence of letters when reading a text. Three typical categories for this extra linguistic information are *Semantic*, *Grammatical* and *Lexical*. Handwriting recognition systems today are still not as good to decipher writing as humans and it is likely that a large part of the difference is owed to the ingenious utilization of such extra information layers in humans. Some researchers have tried to explicitly remove these layers when comparing humans and computers, and although the actual results are very much dependent on their particular handwriting solutions as well as the datasets used, it is interesting (but expected) that humans also make a considerable amount of errors when stripped of these extra reading accuracy tools [79, 88].

The application of linguistic knowledge in the on-line handwriting community so far has mainly been focused the lexical part [92]. Previous work addressing the other categories usually model this jointly and implicitly by employing some form of N-gram statistical analysis of word contexts [68, 95]. The handwriting community however, has seemingly reached the unanimous conclusion that lexical knowledge is utterly important when recognizing cursive script. Some systems have exploited this fact completely and use a lexically determined set of words as input to the matching process [36, 94]. Systems producing results without a dictionary can filter the list of possible word hypothesis through a dictionary [50, 85, 107]. The filtering can also be conducted on the basis of some fuzzy string matching algorithm. DP-based methods utilizing handwriting recognition customized versions of the Damerau-Levenshtein metric [24] are

examples of this. Seni et al. construct a segmentation graph of the detection matrix and checks the edit distance of paths from this graph with a reduced word list [104], a procedure also used for off-line word recognition [27]. Schenkel et al. use a similar strategy but start off with the best string from the detection matrix and then calculates the complete HMM-paths from the set of words with a fixed edit distance to this initial hypothesis [101]. A big problem with evaluating complete continuous word-level HMMs for each hypothesis is that the computation time scales with dictionary size. This situation can be greatly improved by a beam search in combination with a tree-based lexicon structure [73, 98].

Filtering lists of candidates from a graph structured word hypothesis list such as those produced by the recognition graph is particularly efficient. Previous work for the on-line case is comparatively scarce [50, 104], but there has been work with segmentation graphs also in the off-line case [26, 44, 71]. Surprisingly it seems as though only Lucas realized (possibly Ford but details are scarce) that maintaining tree structure in the dictionary enables dictionary filtering with time complexity bounded by the alphabet size and independent of dictionary size [72]. Unfortunately these results are not applicable to the dual graph strategy presented in Chapter 5 since it makes assumptions on a multiplicative edge connection distance as well as the probabilistic independence of edges. This is obviously not true in the recognition graph approach since the connection distance requires recalculation of all edge to edge connections. The time complexity properties of the actual lexical validation is however the same as for the simple procedure presented later in this chapter. For this reason it is possible to check results in very large dictionaries, something which has been a crucial problem for other on-line methods in the past [98, 104].

Another issue which has received some attention and is of particular interest for lexical strategies dependent on dictionary size is the task of dynamic lexicon reduction. In the field of Chinese character recognition this goes by the name of coarse classification. This is an important task in terms of time complexity since most methods for recognizing Chinese script are model based [66]. Lexicon reduction methods for cursive alphabetic script also make use of simple holistic features stored for each lexicon word, and compare input features with lexicon to produce a reduced list of words [23, 94, 104].

The lexical processing presented in this chapter is strictly confined to being a postprocess in no way interfering with the shape matching algorithms in Chapters 4 and 5. Lexical processing is designed to take care of the aspects of connected character recognition that doesn't involve the actual shape matching itself. As shown in previous research it should be possible to construct a system that performs well without a dictionary given neat writing [89]. Given the

lexical and possibly higher level linguistic processing it should also be possible to recognize writing that does not even portray the letter sequence intended by the writer such as the above typing example. Although the actual implementation of linguistic processing for the strategy in this thesis has been limited to a simple lexical lookup it is a good baseline for further innovation.

(a) An ambiguous sample of the word *thread*. Without linguistic context the word can be interpreted as *tluearl*.

(b) An ambiguous sample of the word *under*. The 'n' is here a typical contextual shape that has a shape appearance in between 'n' and 'u'.

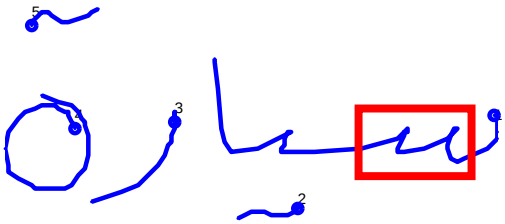
FIGURE 7.1: Two typical samples of writing that is ambiguous due to the writing style of the connected characters.

7.2 Contextual Shape

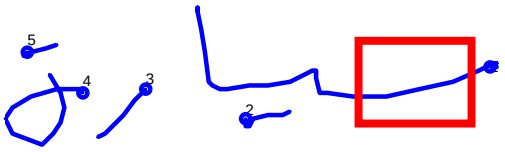
As mentioned in the introduction it is very common for unconstrained connected handwriting not to portray the exact sequence of symbols (shape-wise) that the writer intended to write. Here parts of handwriting such that the recognition problem is simply not solvable with just shape matching methods will be referred to as contextual shape. The main reasons that a writer will produce samples with contextual shape are:

- The particular sequence of character (in the users writing style) is ambiguous when connected, containing parts such as cl⇆d and lc⇆k or even hr⇆lu as seen in Figure 7.1a.
- Sloppy writing that degrades the shape appearance so that it is unrecognizable or even non-existent (flat part of word) such as the deformation seen in Figure 7.2.
- Misspelling

In addition to these cases the segmentation algorithms may also inhibit the actual shape information by adding extra segments or missing segmentation points (causing the boundary between two characters to disappear).



(a) Legible sample of سيارة



(b) Contextual sample of سيارة with missing parts of the letter سـ ('Seen').

FIGURE 7.2: Two samples of the same Arabic word سيارة where the shape information within the box is missing in Figure 7.2b.

In terms of the string matching procedures required to cope with the above problems there are four distinct operations conventionally used in the dictionary interaction in the literature:

i Static. No string operations on the hypothesis sequence.

ii Replacement. Exchanging one character for another in the hypothesis sequence.

iii Insertion. Adding a character to the hypothesis sequence without the support of shape matching.

iv Deletion. Skipping a part of input and the character representing it resulting in a deleted character in the hypothesis sequence.

The last three of the above operations involve some form of added fuzzyness to the recognition algorithm since they aim at recognizing more than the shape information contains. In this thesis only the first static part has been tested in the framework presented. Work with the other three are however clearly possible and is a subject that will be pursued in the near future. A short discussion on how to proceed is included in Section 7.4.

In previous chapters it has been shown how the dual graph structure effectively stores the best paths with some constraints on template connectivity serving the purpose of excluding sequences of templates that would not occur in writing. Since that solution puts no linguistic constraints on the best path it is equivalent to checking the resulting strings with an infinite dictionary containing all symbolic sequence combinations of arbitrary length. The connectivity function for templates uses heuristic shape context information to exclude certain sequences. This section will define a similar function but acting on linguistic context instead.

Definition 7.2.1. The linguistic connectivity function $\mathcal{L}(\{l_i\}^k, q)$ between a sequence of symbols $\{l_i\}^k$ and new symbol q is a binary function s.t. $\mathcal{L}(\{l_i\}^k, q) = 1$ if the new sequence $(l_1, \ldots, l_k, q)$ constitute a linguistically valid sequence and 0 otherwise.

Definition 7.2.1 makes use of the term *linguistically valid* and by this is simply meant a linguistic rule such as an ordinary dictionary or truncation of letter connection probabilities. The linguistic connectivity function will not induce any difference in the actual distance values when validating a symbol sequence, but merely provide exclusion of non-words from a linguistic perspective.

Note that this definition is independent of the actual shape matching graph used. It can therefore be used regardless of whether the symbolic input derives from shape matching directly or through some form of string edit as in [23] or through zero length edges in the segmentation graph as in [72].

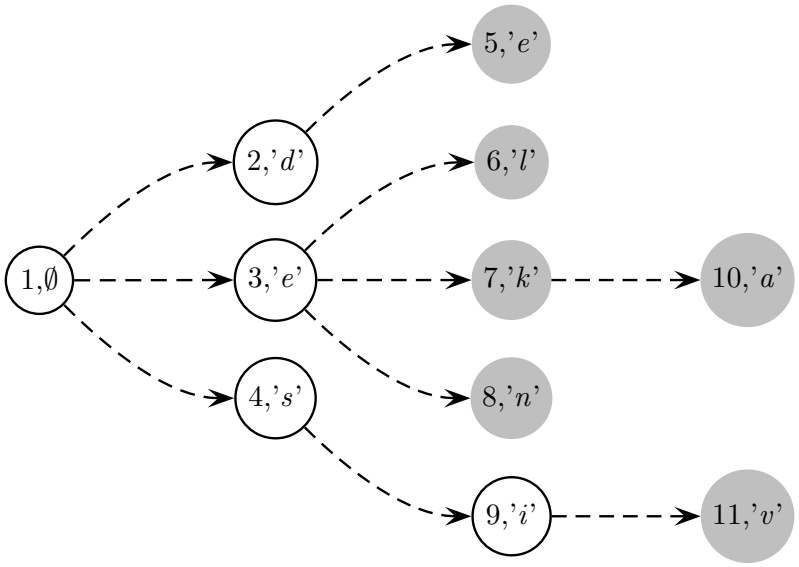


FIGURE 7.3: A simple example of a Trie dictionary containing the words *'de','el','ek','eka','en','siv'*. Nodes corresponding to end of words are shaded in lightgray.

7.3 Static Lexical Lookup

The linguistic connectivity function defined in Def. 7.2.1 can be inserted directly into the connectivity check in Algorithm 6 to limit results to an ordered list of linguistically valid strings. All this does however is to exclude non-valid strings and it does not impose any probability-driven punishment on string retrieval. The stance taken in this thesis is that given a context of only the word, just like a human reading an isolated word, the only applicable linguistic process is the exclusion of non-dictionary words.

Given the combinatorial nature of graphs, the effectiveness of string validation through a dictionary is utterly important. One intuitive way of guaranteeing such efficacy is to represent the dictionary in a suitable format. For the dual

graph approach it will be shown that one suitable such format for the dictionary is a Trie structure [41] as for many previous systems [50, 72, 73]. This is a structure almost identical to the segmentation graph in that edges correspond to some symbol (although it can be an ambiguous symbol) and that common edges are shared, see Figure 7.3.

7.3.1 Trie dictionary implementation

A small Trie dictionary containing 6 words is shown in Figure 7.3. As the figure shows, the Trie dictionary is a directed acyclic graph structure $G_L = (N_L, E_L)$, where each node corresponds to a letter and each edge implies that the end-node may follow after the start node. Each node in the Trie structure corresponds to the last letter of a uniquely defined start sequence and will be denoted by $n_l^k = n(\{l_1, \ldots, l\})$ where k is an index and l the last symbol. An edge between two nodes will be denoted by $e^{k \to p}$, where k, p are the node indexes. When a Trie dictionary or similar graph-like dictionary structure is used for checking the linguistic validity of adding a symbol to a sequence the linguistic connective function can be simplified into (7.1).

$$\mathcal{L}(\{l_i\}^L, q) = \begin{cases} 1, & \text{if } \exists n_{l_L}^k = n(\{l_i\}^L), n_q^i \in N_L, e^{k \to i} \in E_L \\ 0, & \text{otherwise.} \end{cases} \tag{7.1}$$

The extra computation time required by linguistic validation with a dictionary thus amounts to checking the succeeding nodes from a node corresponding to the last letter in the path in the recognition graph. As reported in Lucas implementation of path algebras [72] this means that the validity check of this technique is independent of the dictionary size. As seen in Figure 7.4 the implementation is simply accomplished by associating a Trie node index to every expanded path. For every new symbol added to the sequence only the edge forward from the given Trie node index need to be checked. However, the number of strings to check in each node is of course dependent upon the dictionary size - the fewer words in the dictionary the larger number of strings can be discarded. Something which is of course even more useful when excluding strings with only slightly differing sorting distances in Algorithm 7.

To further optimize the size of the dictionary dynamically, the shortest and longest succeeding path endnodes (complete words) can also be stored for each node in the Trie. Let $W(n_l^k) = \{w\}$ be all words in the Trie dictionary passing through node n_l^k. Then write the number of symbols (nodes) in each such word by $|w|$. The largest $R_{\max}$ and smallest $R_{\min}$ possible number of edges forward from a given path in the recognition graph can also be calculated by

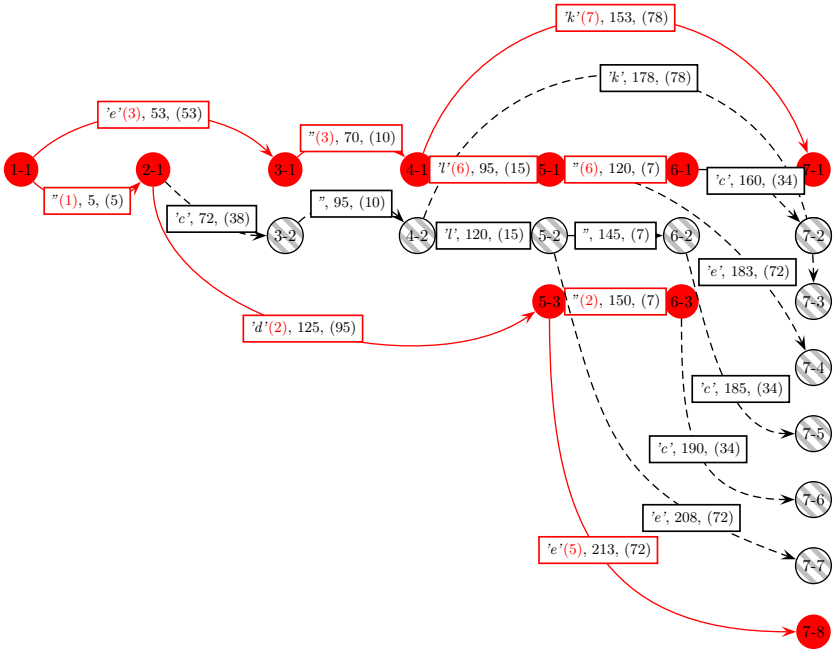


FIGURE 7.4: The recognition graph expansion from Figure 5.8 reproduced with added lexical constraints using the Trie dictionary of Figure 7.3. The dashed edges and nodes correspond to lexically invalid path expansions and the Trie node indexes for the valid paths are included within parenthesis.

analyzing the edges in the segmentation graph and possibly also fuzzy criteria for handling missing edge information. This means that the Trie connective function in (7.1) can be updated with the extra constraint (7.2).

$$\mathcal{L}_{\text{length}}(\{l_i\}^L, q) = \begin{cases} 1, & \text{if } max_{w \in W(n_l^k)}|w| > R_{\min}, min_{w \in W(n_l^k)}|w| < R_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

7.4 Dynamic Lexical Lookup

The static lexical lookup treated in the previous section only performs exclusions of lexically invalid strings in the recognition graph and thus will not enable recognition of symbols corresponding to shape sequences not present in the segmentation graph. To enable such *fuzzy* matching, the string edit operations listed in the beginning of this chapter, i.e. *replacement*, *insertion* and *deletion* must be treated. To some extent the last of these has already been treated with the noise handling described in Section 5.4. Deletion implies moving forward shape-wise in the input sample without matching this to a letter,i.e. without stepping forward in the dictionary. The difficult task here is therefore a matter of tuning the noise distance parameters to a desirable level. Replacement means replacing one edge by another and thus is something not handled by the segmentation graph when the edge of the desired hypothesis is discarded in the beam-limitation in the segmentation graph. A typical such case could be the letter 'n' written as an 'n' in Figure 7.1b. A reasonable way to treat this problem is to calculate and store such fuzzy edit costs for each such pair of confusion pairs. The most difficult case to treat is probably insertion of letters not corresponding to shape in the input sample. This could be caused by flattened shapes such as in Figure 7.2 or by misspelling. It is probable that it is best to use whatever shape information that is available, such as matching a character to a flat stroke, descending stroke etc, and make some assessment of how much it should cost to transform a given shape (template) from a sequence into each such coarse shape descriptor.

So although such dynamic lexical lookup operations are yet to be implemented and experimented with for the system presented in this thesis, the system itself contains all the necessary characteristics to enable such operations.

7.5 Conclusions

In this chapter it has been shown that the recognition graph presented in Chapter 5 can be searched efficiently for dictionary entries. It has been shown that such a search can be independent of dictionary size and in fact it is possible to control the computational speed by setting $B_{\mathfrak{R}}$ on the candidates for each node in the recognition graph as presented in Section 5.5. Since a dictionary check severely limits the space of possible template sequences it is possible to run recognition with a lower value of $B_{\mathfrak{R}}$ than when using Algorithm 6 without the linguistic connectivity function.

CHAPTER 8

Clustering And Automatic Template Modeling Techniques

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.

George Bernard Shaw

For implicit modeling methods such as those presented in Section 2.6 the availability of a training set of decent size and quality is absolutely vital. This is one of the strong arguments for template based methods. It does however not mean that recognition methods based on template matching are indifferent to training data. This chapter will deal with methods for improving the template modeling scheme presented in previous chapters by incorporating observations from a training set.

8.1 Introduction

It goes without saying that a set of templates based completely on one persons handwriting will have a strong bias towards that persons specific writing style and in that sense will suffer from *overtraining*. Adaptive classifiers can therefore also easily be constructed for template matching methods by simply adjusting the template set by incorporating templates of the current writer as in [36, 50, 94, 132]. For several reasons, a writer independent system is however generally preferable. One of the more striking problems with adaptation is that it somehow relies on the users intuition of how the system works. The reason for this is that writers may not be aware of the fact that they write two letter shapes identically for instance. When adapting the system to such writing there is a risk that the recognition accuracy for such letters will drop rather than increase when conducting adaptation.

In the quest for a writer independent system, one in general seeks to model each template so that it corresponds to some form of *mean* shape with respect to the feature space used and the distance function used to discriminate. For on-line handwriting recognition this research is often categorized under the somewhat misleading term *clustering* [8, 56, 134] (cf. Section 2.2.2). Originally the term is used in pattern recognition to denote the process of unsupervised classification, i.e. the problem of dividing observations into classes without prior class knowledge [56]. So what does this mean when applying clustering to a set of handwriting samples where the classes (i.e. the symbol labels) are already known? For template based recognition this amounts to the task of finding a set of templates corresponding to salient shape variations for the given symbol. Such significant variations in writing the same letter, i.e. different templates corresponding to the same symbol are often referred to as *allographs*.

Since template matching schemes in the past have been applied mainly to the single character recognition problem the clustering techniques for on-line characters have generally employed the same globally defined distance function to conduct clustering. The main clustering strategies used to achieve this have been Hierarchical Agglomerative Clustering as in [7, 134] and the k-means al-

gorithm [56, 91]. The k-means algorithm is robust but suffers from the fact that the number of clusters need to be statically predetermined. The hierarchical algorithm on the other hand suffers from the requirement to find a suitable threshold value. If the goal is to produce clusters (allograph sets) that maximize recognition performance it makes sense to inspect the bearing that the choice of allographs has on recognition accuracy. The above two methods belong to the category of *generative* clustering methods in that they solely take the class to be modeled into account when generating clusters. There are other methods for clustering which also examine the effects of cluster choices in the neighboring clusters of other symbols such as [117]. These belong to the group of *discriminative* clustering methods. Since the discriminative methods aim at producing optimal class borders in sample feature space they are indeed very close conceptually to SVM [126] and LVQ [60]. Indeed the generative nature of HMMs and template methods is often posed as an argument that makes intrinsically discriminative neural networks more suited for handwriting recognition [22]. This is especially true for genuinely discriminative networks like Kohonen maps [59].

This chapter will present some clustering techniques specialized for the segmented template matching strategy presented in this thesis.

8.2 Holistic Clustering

In contrast to the segmental operations treated in this thesis the conventional type of single character clustering will here be referred to as *holistic* clustering since it treats one complete shape at a time. By extending the support of the distance function so that it can compare samples with *dissimilar* segmentations it is natural to use this function also for clustering with segmented samples. A natural such extension would be to set $\mathfrak{G}(X, Y) = \infty$ unless $\mathcal{S}(X) \sim \mathcal{S}(Y)$. When applying holistic clustering techniques to a set of segmented samples with that kind of a strategy, naturally it will also generate separate clusters for samples that incorporate noise segments. It is not a good idea to try to keep models of all possible combinations that noise segments can be added to templates and as expected such clustering techniques fail to provide good results for a segmented distance function [116].

By removing all such clusters that correspond to allographs with incorporated noise, a good initial candidate for a definition of supported noise-free allographs is acquired. In other words for segmented input, holistic clustering may be a useful tool for extracting an initial collection of templates with varying segmentation. Improvements to such a database that makes more explicit use of the segmentation will be discussed in the remainder of this chapter.

8.3 Segmentation Definition Database

The *segmentation definition database* of a segmented template matching system is a set of templates designated to define the supported set of segmentations for each template. Originally such a database could be generated from automatically some kind of clustering process as described in Section 8.2, it could be hand-built or a combination of both. In this database the actual discriminatory power of each template does not have to be great, the main objective here is that it contains all the desired allograph and segmentation variations for each character. The allograph corresponds to perceptually different ways of writing a character whereas the segmentations are different segmentations that may be observed within each such class of allographs as seen in Figure 8.1.

In a way this segmentation definition database can be interpreted as a dynamic way of defining the type of syntactic rules used in the recognition by synthesis paradigm such as the AHD language (cf. [89]) or RecifMOT [6]. In the context presented here rules are implicitly defined by the matching properties of the elements in the database.

8.4 Forced Recognition And Labeling

In order to present algorithms specialized at training segmented templates, some new terminology needs to be introduced. Let the set of all available samples be $\mathbb{X}$ and the subset with the known class index i be denoted by $\mathbb{X}^i$. This can be further divided into which allograph (which in turn corresponds to a template in the database) each sample adheres to. Denote this by adding an extra template index so that $\mathbb{X}^{(i,j)}$ denotes all samples of class i which are closest in shape to template T_j. Furthermore let $\mathbb{X}^{(i,k,s)} = \{\Lambda_s \in X, X \in \mathbb{X}^{(i,k)}\}$ Normally the class label of a sample is known in training data whereas the template index depends on the database and can therefore not be assigned by writers that are unaware of the underlying recognition system. This poses a new challenge in how to find $X \in \mathbb{X}^{(i,j)}$ such that $\mathcal{S}(X) \nsim \mathcal{S}(T_j)$. A very simple way of obtaining such a set by means of a database is by running recognition with a dictionary as described in Chapter 7 and restricting the dictionary only to contain the known class label. This will force the recognition system to find the best path with a template of the correct class to be associated with each sample. In particular this enables extraction of samples which are part of connected script. For Arabic for instance this is crucial for training since characters vary in shape depending on the placement in the word. Figure 8.2 shows one such example of automatically extracting a *medial* (the form of a character when connected on both sides) of the character .

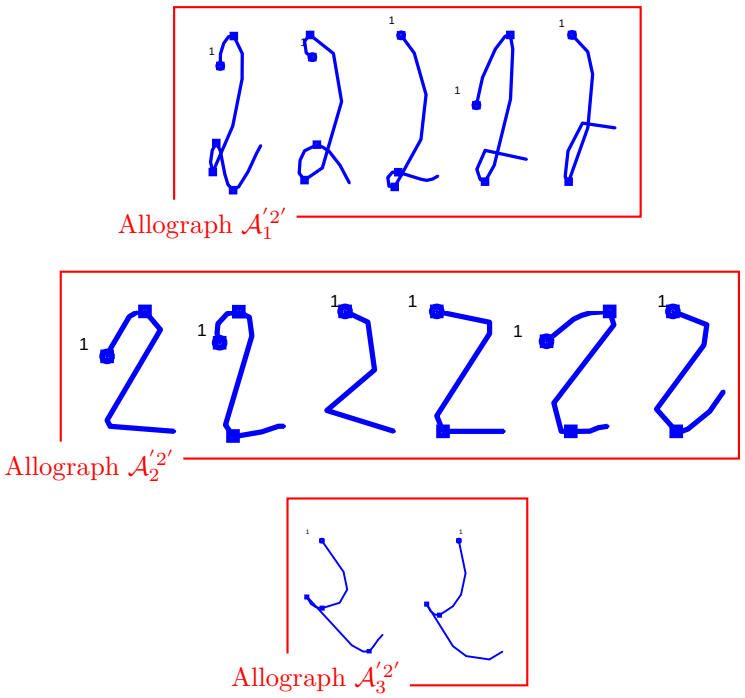


FIGURE 8.1: An example of a segmentation definition database content for the class '2' showing 3 distinct allographs each with a number of different segmentation definitions.

8.4.1 Erroneous Training Data

There are mainly three types of errors that the forced recognition process needs to cope with.

1. Mislabeled samples
2. Samples with extraneous noise
3. Partial samples or otherwise severely deformed samples.

Samples with extraneous noise in the form of extra segments can be handled simply by enabling the noise handling described in Section 5.4 during the forced recognition. An example of this noise filtering process is seen in Figure 8.3.

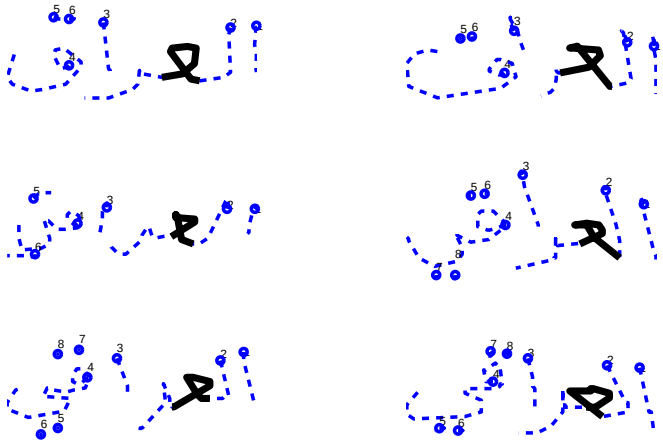
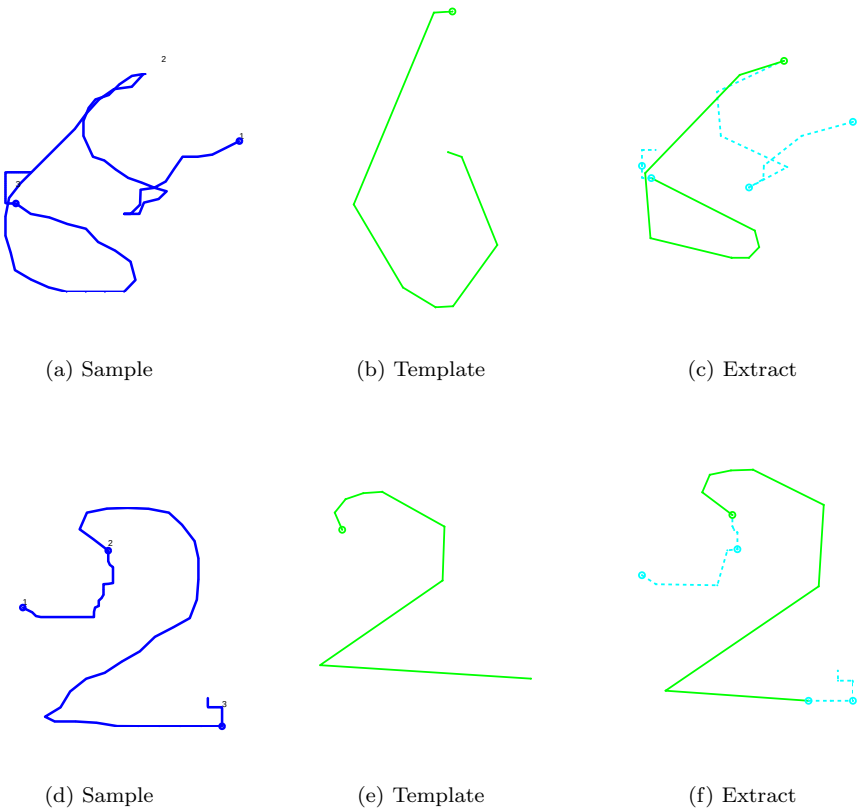


FIGURE 8.2: An example of forced recognition followed by allograph labeling and extraction of the Arabic character ـه from a set of word samples.

An intuitive way to handle other types of errors bearing some resemblance to the dynamic handling of threshold values in hierarchical clustering, is to set some form of a distance threshold value and simply discard any samples with a total matching distance exceeding this (not including noise parts) from the training set. This can also be used to automatically sift out contextual allographs as discussed in Section 7.4. It may however be very difficult to determine such a threshold and it is most likely that a zero false acceptance rate would discard many samples displaying large variations. A different way to handle this explored in this thesis is to add non-class templates with the aim of catching mislabeled and severely deformed samples. Such templates can easily be discarded before actually using the generated database for recognition. An obvious flaw of this technique is of course that such templates need to be manually determined, furthermore they need to be very dissimilar from the true templates. Nevertheless this technique manages to easily extract and discard mislabeled samples from the training set.



(a) Sample (b) Template (c) Extract

(d) Sample (e) Template (f) Extract

FIGURE 8.3: Examples of extraneous noise in UNIPEN/1a (cf. Section 9.2) automatically removed through allograph labeling with forced recognition as described in Section 8.4.

8.5 Segmental clustering

A problem with the vast space of synthetic samples from the combinatorial expansion of the sample segmentation is that the space quickly becomes too large for conventional holistic clustering methods. Due to the graph nature of the modeling, however, there is also a naive alternative for clustering, which has been utilized in the experiments of Chapter 9 - clustering segment-by-segment. By clustering segment-by-segment, the complete set of synthetic samples are implicitly included in the clustering procedure, while still avoiding the imprac-

tical issues of the immense set of data. Various data set expansion methods have been studied before and for neural networks this has been shown to improve recognition results [106]. The obvious set back of this is of course that a large part of the distance value attributes to the actual connection between segments as seen in (4.3) on page 54. For a more optimal clustering in terms of the trade off recognition accuracy/database size/time complexity it is probable that some allographs would benefit from a varied clustering precision per segment. This is not in the scope of this thesis, however, and is left as an open topic for future research.

In order to cluster segment-by-segment the connective distance component from (4.3) can not be used. However the equivalents of the features *direction* $\vec{d}$, *length* λ, *connection angle* α and the *curve segment shape* $\mathcal{S}$ can be compared at segment level. Here these features have been used to produce a clustering of the segmental variations.

Effectively the segmental distance function used to cluster the segments used here is, just like in (4.3), a linear combination of the squared differences and can be written as

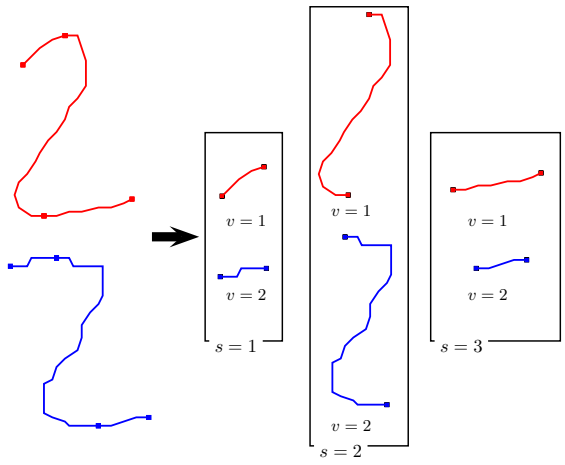
$$\begin{aligned} d_{\text{seg}}(X,Y) = w_{\vec{d}} d_{\text{angle}}(\vec{d}(X) - \vec{d}(Y))^2 + w_\lambda d_\lambda(\lambda(X) - \lambda(Y))^2 \\ + w_\alpha(d_{\text{angle}}(\alpha^-(X) - \alpha^-(Y))^2 + d_{\text{angle}}(\alpha^+(X) - \alpha^+(Y))^2) + \\ w_{\mathcal{S}} d_{\mathcal{S}}(X,Y), w \geq 0, \quad (8.1) \end{aligned}$$

where d_{angle} is an angle distance function and the α^- and α^+ denotes the previous connection angle and the subsequent connection angles. $d_{\mathcal{S}}$ is the same as in the template distance function in (4.3).

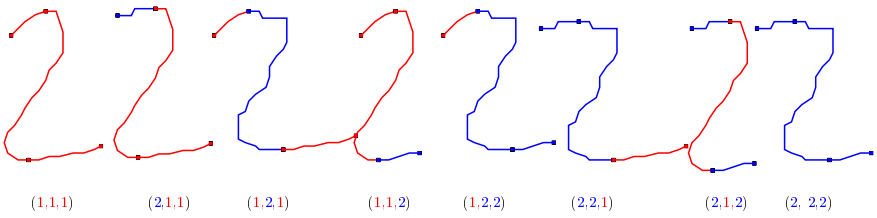
8.6 The Variation Graph

Given previous graph techniques for segmented recognition, a natural way to combine the segment clusters produced by segmental clustering is to construct a graph. This graph is called the *Variation Graph* since it contains the model variations for each segment of a given allograph template. To distinguish a variation graph template the notation $\mathcal{A}_k^i = (N_{\mathcal{A}}^{(i,k)}, E_{\mathcal{A}}^{(i,k)})$ will be used for the variation graph modeling allograph k of class i. Just as for segmented samples $\mathcal{S}(\mathcal{A}_k^i)$ denotes the segmentation of this graph model. A particular variation of a segment s from $\mathcal{A}_k^i$ is written as $n^{(i,k,s)}$. In the variation graph each node corresponds to such a variation and an edge $e_s^{m \to r}$ corresponds to a link between variations $n_m^{(i,k,s-1)}$ and $n_r^{(i,k,s)}$. Let $\mathcal{N}_{\mathcal{A}}^{(i,k)} = (n_1, \ldots, n_{|\mathcal{S}(A_k^i)|-1})$ denote a sequence of variation nodes in $\mathcal{A}_k^i$ and let $\mathbb{N}_{\mathcal{A}}^{(i,k)}$ be the set of all such

possible sequences. The set of edges E_s^r of variation $v_r^{(i,k,s)}$ in the graph will thereby correspond to the indexes of the set of variations from segment $s-1$ that can connect forward.



(a) Graph base and segment variations



(b) All variation sequences

FIGURE 8.4: An example of a variation graph based on two samples and all sequences of the fully connected graph.

8.6.1 Generative Training

The generation of variation graphs described here assumes an available set of allograph labeled samples as described in Section 8.4.

Given the segmental distance function in (8.1) the generation of the variation graph can be conducted through Algorithm 8. Each segment is clustered with a generative clustering algorithm such as Hierarchical Agglomerative clustering [7] or k-Means [56] and a choice of n cluster representatives are chosen as variations for each segment.

The mixture of segments of different samples may however correspond to segment shapes that are not representative of the class. To cope with this problem the closest variation sequence is found for each sample in the training set. All such observed variation connections in the training set can be recorded and after that the edges between variations not observed in the training set can be excluded. Figure 8.4 shows all the possible variation sequences when mixing two arbitrary samples of the same allograph (and segmentation) of the digit '2'.

Algorithm 8 Generative Variation Graph Training

for $\mathcal{A}_k^i \in \mathbb{D}$ **do**
  for $s = 1, \ldots, |\mathcal{S}(\mathcal{A}_k^i)|$ **do**
    Generate clusters $(c_1, \ldots, c_n), c_i \in \mathbb{X}^{(i,k,s)}$
    for $p = 1, \ldots, n$ **do**
      Set $n_p^{(i,k,s)} = c_p$ in $\mathcal{A}_k^i$
      if s > 1 **then**
        $E_p^s = (1, \ldots, n)$ *% Initialize edges*
        $r_j(E_p^s) = 0, j = (1, \ldots, n)$ *% Initialize edge counter*
      end if
    end for
  end for
  for $X \in \mathbb{X}^{(i,k)}$ ***do***
    Set $\mathcal{N}^* = \text{argmin}_{\mathcal{N} \in \mathbb{N}_{\mathcal{A}}^{(i,k)}} \mathfrak{G}(X, \mathcal{N})$
    Set $I = (i_1, \ldots, i_{|\mathcal{S}(\mathcal{A}_k^i)|-1})$ *to the indexes of* $\mathcal{N}^*$
    for $s = 2, \ldots, |\mathcal{S}(\mathcal{A}_k^i)|$ ***do***
      $r_{i_{s-1}}(E_{i_s}^s) = r_{i_{s-1}}(E_{i_s}^s) + 1$ *% Increment edge counter*
    end for
  end for
  Update $E_p^s = \{j \in (1, \ldots, n) \,|r_j(E_p^s) > T\}, \forall s, p$
end for

8.6.2 Discriminative Training

The procedure for generating a variation graph described in Section 8.6.1 is a generative approach to modeling not taking the possible conflicts with surrounding classes into account. The experiments in Section 9.3 however show that the variation graph could benefit from a second layer of more discriminative methods such as CDCR [117], LVQ [18, 60] or SVM [1, 9] since top-2 recognition candidates are significantly better than the best candidate. This section presents another approach to dealing with this problem already at the template design phase. The basis for the rudimentary algorithm presented here is that the cause for many of the top candidate confusions is that the contributions of discriminative features between two very similar classes often *drown* in the general noise caused by the normal variations of other feature values. The strategy of the simple algorithm proposed here is therefore quite intuitive and aims at assimilating similar parts of confused characters while the discriminative parts are accentuated. The full algorithm, here referred to as *Discriminative Assimilation (DA)*, is schematically described in Algorithm 9. For notational purposes some new terminology is introduced. Let $\mathcal{V}_k^i(X) = \text{argmin}_{\mathcal{N} \in \mathbb{N}_{\mathcal{A}}^{(i,k)}}$ and let $\{\Lambda_j\}_{j=1}^{|\mathcal{S}(X)|-1}$ be the sample segments of X. For each segment s associate the sample segments in $\mathbb{X}^{(i,k,s)}$ according to the closest variation in $\mathcal{A}_k^i$ as $\mathbb{X}_p^{(i,k,s)} = \{\Lambda_s | n_p^{(i,k,s)} \in \mathcal{V}_k^i(X)\}$ and order by the distance value as determined by the function in (8.1). The algorithm will skim through the feature space of every segment and determine a new sequence of variations to add for one of the two variation graphs being discriminatively compared. Let the $\mathfrak{f}_f$ operator denote the retrieval of feature f from a variation or a set of sample segments. One iteration of the algorithm is defined as allowing each allograph to get such an update one time if the number of errors caused by the neighboring class exceeds a certain threshold.

8.7 Recognition with Variation Graphs

Just as the linguistic properties optimally are modeled with a graph when graph structures are used with recognition as shown in Chapter 7, this is also a very efficient way to match multiple templates. As seen in Figure 8.4 each variation graph can contain a vast amount of templates through the multitude of variation sequences, but the segmental nature of the recognition process described in previous chapters enables *sharing* of partial recognition results. In particular the same beam search paradigms can be used within the variation graph match to find the best sequence of variations matching input. In the segment-by-segment approach of Algorithm 5 this is handled by the incomplete

Algorithm 9 Discriminative Assimilation (DA) graph $\mathcal{A}_k^i$ w.r.t $\mathcal{A}_m^j$

Find most common sequence $\mathcal{V}_m^j(X)$ s.t. $\mathfrak{G}(\mathcal{A}_m^j, X) < \mathfrak{G}(\mathcal{A}_k^i, X), X \in \mathbb{X}^{(i,k)}$
for $s = 1, \ldots, |S(X)| - 1$ **do**
  Find $\mathbb{X}_{[j,m,p]}^{(i,k,s)} = \{\Lambda_s \in X | X \in \mathbb{X}^{(i,k)}, \mathcal{V}_{(m,s)}^j(X) = n_p^{(j,m,s)}\}$
  Find $\mathbb{X}_p^{(j,m,s)}$
  for $f = 1, \ldots, \dim \mathfrak{F}$ **do**
    if Span of feature f in $\mathbb{X}_{[j,m,p]}^{(i,k,s)}$ overlaps that of $\mathbb{X}_p^{(j,m,s)}$ **then**
      Set $a_f^s = \mathfrak{f}_f(n_p^{(j,m,s)})$
    else
      Find $\mathfrak{m}$ as middle of margin δ between $\mathfrak{f}_f(\mathbb{X}_{[j,m,p]}^{(i,k,s)})$ and $\mathfrak{f}_f(\mathbb{X}_p^{(j,m,s)})$
      Set

$$a_f^s = \begin{cases} \mathfrak{m} - \delta/2, & \text{if } \delta > \max_{F \in \mathfrak{f}_f(\mathbb{X}_{[j,m,p]}^{(i,k,s)})} F \\ \mathfrak{m} + \delta/2, & \text{otherwise} \end{cases}$$

    end if
  end for
end for
if At least one feature of one segment differs **then**
  Add segment variation sequence $\mathbf{a} = \{a^s\}_{s=1}^{|\mathcal{S}(X)|-1}$ to $\mathcal{A}_k^i$
end if

beam B_I which is common for all templates. For the template-by-template strategy of Algorithm 3, B_I is instead the number of sequences allowed to continue to the next segment per template. Unlike the segment-by-segment strategy partial templates do not compete for space within the beam. Since each such sequence may have common segment variations the effective number of matches need to be made during the segmentation graph construction scales well with the total number of sequences enabled through the variation graph. A potential flaw of this property is that the Markov characteristic (edges only between successive segments) will also make it impossible to directly exclude some sequences, since an edge between two segment variations will allow such a connection for all segment variations in the future. This could cause problems for characters that have distinctly different appearances in a non-connected part (i.e. with at least one intermittent segment). In these cases such allograph types can be separated into two variation graphs in order to allow exclusion of disallowed interconnections.

CHAPTER 9

Experiments

The training of children is a profession, where we must know how to waste time in order to save it.

Jean Jacques Rousseau

One of the satisfactory things about handwriting recognition research is the possibility of instant quantitative feedback. It is however always important to keep in mind that such results by no means are absolute in their judgement. There are lots of aspects to consider. Different datasets contain different types of data posing varying problems which may be handled differently by different strategies. This chapter will begin with a short general discussion on recognition experiments with on-line characters followed by experiments conducted with the algorithms presented in the thesis.

9.1 Introduction

The complexity of the handwriting recognition problem is largely due to the notable variance in shape that samples of handwriting display. Thus the significance of experimental recognition results is also utterly dependent on the content of the data set used. In the early days of on-line handwriting recognition capturing devices were not as reliable and various preprocessing steps focusing on noise removal were mandatory elements of the recognition systems [20, 47, 125]. The lack of publicly available data sets along with hardware dependency thus made it virtually impossible to make quantitative comparisons of recognition experiments. Today capturing devices are much more reliable and sampling artifacts can generally be attributed to the human factor. A shaky writing environment can cause jagged curves and a slip of the pentip can still cause extra noise segments (cf. Section 5.4).

9.2 Datasets

In the past fifteen years this situation has been greatly improved starting off with the ambitious UNIPEN project in the mid-90's [48]. Unfortunately this project started off when the digitizing tablets used for collection were somewhat unreliable and does contain fair amounts of noisy data [97]. The UNIPEN dataset is divided into a training set `Train-R01/V07` and a test set `DevTest-R01/V02`. Regrettably only the train set has been available for public use so far, even though some recent work has been conducted to automatically clean the DevSet in order to simplify comparison of recognition performance [133]. Other public datasets now include the MIT database [57], the IRONOFF database [128], the Ethem-Alpaydin data set [2] for digits and the Kuchibue database for on-line Japanese characters [81]. Recently Liwicki et al. also made a publicly available set of on-line handwritten sentences [67]. The author has also contributed with a dataset collected with audio-playback. The writers

were instructed to write the words provided through audio feedback instead of showing the words to be written visually [113]. The aim of this technique is not to influence the writers style of writing by showing a particular font, and it was also used for collecting the MIT data set [57]. Unfortunately there are no actual results showing if this has any effect.

9.2.1 Dataset Partitions

Datasets are often partitioned into 2 or 3 sets (train, test, validation). These partitions can be either single-writer,multi-writer or omni-writer [97]. In most cases the last two are the most interesting from a recognition perspective. The multi-writer case implies that there are multiple writers both in the train and test set whereas the omni-writer case also implies writer independency, i.e. that no data from a writer in the test set can be part of the train set. With the aim of a general handwriting solution naturally the omni-writer case will be the common case for a commercial system, since no system will be able to contain training data from all possible users. For partitioning of fixed datasets however, it is often recommendable to use some kind of a multi-writer partitioning scheme, since the limited size of data may otherwise cause some allographs for some classes to be absent in either train or test set.

9.2.2 Allograph Distribution Analysis

Apart from the influence of the hardware used for capturing on-line information, the writing styles of the writers contributing to the collection is also of great importance for the properties on the data set. The discrimination power of different recognition strategies depend greatly of the problem and the type of salient features. For this reason it is of interest not only to know the data set used, but also the distribution of styles (or allographs) contained in the set. The success of different recognition methods on a certain partition into training and testing set may depend greatly on the correspondence of allograph distribution between these two sets.

By making use of the allograph labeling scheme developed for training noise free models of segmented characters in Section 8.4, it is also possible to directly analyze the distribution of the labels. Such allograph content analysis has been conducted in the past [57] but no previous results have been found for the noise-free allograph analysis shown here. In recent years a data set used often when comparing the state-of-the-art in on-line single character recognition is the digit collection in the UNIPEN `Train-R01/V07` data set, i.e. UNIPEN/1a [48]. Furthermore many researchers have used a similar way to divide this set

into multi-writer 2:1 ratio of train and test data [77, 97][1]. The same 2:1 ratio of train/test multi-writer data but with a different partitioning scheme has been employed in other experiments [3, 8]. By conducting forced recognition on both the test set and the train set the allograph distributions can be compared. The most frequent allographs for the symbols 0, 2 for the different sets are shown in Tables 9.1 and 9.2. The complete lists are included in Appendix A.

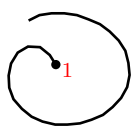
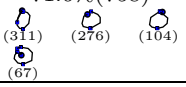
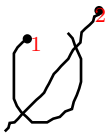
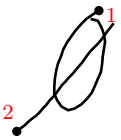
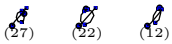
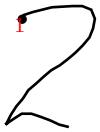
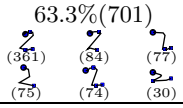
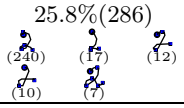
Class	$\|\mathbb{X}\|$	Allographs		
0	1059	 71.6%(758) 	 11.9%(126) 	 5.8%(61) 
2	1107	 63.3%(701) 	 25.8%(286) 	 5.5%(61) 

TABLE 9.1: The three most common allographs with their respective segmentations for the classes 0,2. The complete table is found in Appendix A.1

[1]Dataset partition through utils2compareHWR tools, `http://www.alphaworks.ibm.com/tech/comparehwr`

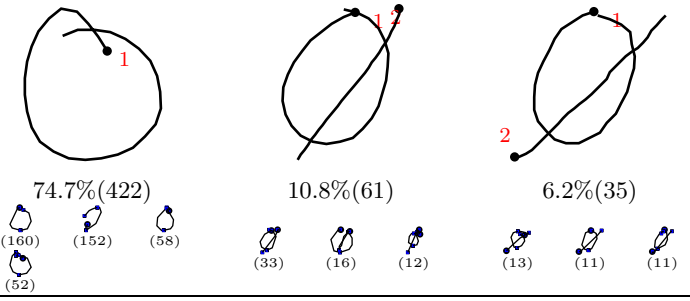
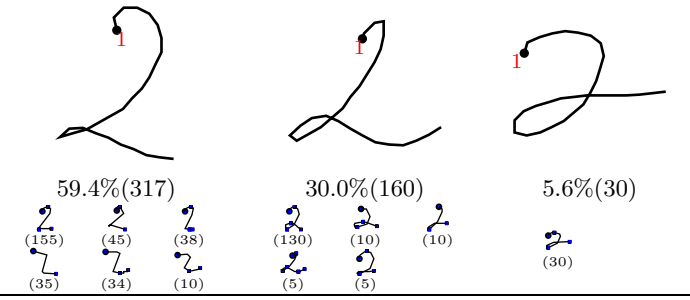
Class	$\|\mathbb{X}\|$	Allographs
0	565	
2	534	

TABLE 9.2: The three most common allographs in the test part of the UP1a data with their respective segmentations.

9.3 Single Character Recognition

Single character recognition experiments have been conducted both on the digits in the `UNIPEN/1a` dataset and on a proprietary set of isolated Arabic single characters supplied by Zi Decuma.

Digits The `UNIPEN/1a` was divided into the training and test set with multi-writer paradigm as described above. The training set was used to alter the modeling of the observed segmentation in the manually defined segment definition database with variation graphs as described in Chapter 8. For the digit set both the generative and discriminative variation graph modeling methods are tested, whereas only the generative approach has been tested on the Arabic single character problem. The digit experiments were conducted using the template-by-template matching strategy (cf. 5.3.1) during the segmentation

k	top-1	top-2	top-3
0 (org)	84.8%	94.3%	97.0%
5	92.9%	97.2%	98.2%
10	94.0%	97.7%	98.3%
15	94.3%	97.5%	98.2%

TABLE 9.3: The effect of varying number of clusters for generative training on the independent test set

k	top-1	top-2	top-3
0 (org)	86.0%	94.5%	97.0%
5	93.9%	97.9%	98.7%
10	95.7%	98.4%	98.9%

TABLE 9.4: The effect of varying number of clusters for generative training on the training set

graph construction and for all these experiments the beam width B_I for the variation graph matching was set to 15. The beam sizes used for the segmentation and recognition graphs (i.e. the number of edges in each node in the segmentation graph and the number of paths in each node in the recognition graph) were large enough ($B_{\mathfrak{S}} = 10, B_{\mathfrak{R}} = 15$) not to have a large impact on the recognition accuracy of isolated digits. Recognition results are reported as top-n results for various n, each indicating that the correct interpretation is among the n best matching template classes.

The recognition results in Table 9.3 are a few percent lower than the best reported in the literature for this split into training and test data [3, 77]. There is however a significant difference for the top-2 results, and these results are very close to the best recognition results reported in the literature. There are several possibilities to improve upon the recognition results conducted here. Firstly, recognition parameters (i.e. weights in (4.3 on page 54) have only been set uniformly to 1. The recognition on the digit set does not include the relative x and relative y features from Table 4.1 which give improvement in the Arabic case below. The noise distance thresholds are also ad hoc values and so are the parameters (constants) in the dcmDTW function of Section 3.2.4. Furthermore it is possible that the ratio normalization function in Figure 4.1 on page 4.1 can be optimized further to improve recognition accuracy.

A minor improvement can be achieved by applying discriminative training with Algorithm 9. Results of a few iterations with this algorithm on the databases

k-base	nIter	top-1	top-2	top-3
5	1	92.6%	97.3%	98.2%
5	5	93.7%	97.3%	98.2%
10	1	94.1%	97.7%	98.3%
10	2	94.2%	97.7%	98.3%
10	4	94.4%	97.7%	98.3%

TABLE 9.5: The effect of additional discriminative training. The error threshold was set to 3 and only single stroke allographs were regarded.

trained with the generative approach for 5, 10 cluster centers are shown in Table 9.5. As expected the improvement is greater for the base of 5 generative segmental clusters. In total only 5 extra sequences are needed for improving hitrates with 0.4%, which corresponds to an error reduction of 15%. A complication in running the naive feature by feature optimization of Algorithm 9 on the set of features presented in Section 4.2 is that they depend on each other. This was particularly limiting in the system implemented for the experiments shown in Table 9.5 since each feature was recalculated from each segment thus making it impossible to give values to dependent features that would cause conflicts in the segment shape representation.

Arabic Single Characters The Arabic single character set training set contained 27 writers and 2150 samples- The writer independent test set consisted of 12 writers and a total of 948 samples. This means that this is an omni-writer setting as opposed to the multi-writer case for the UNIPEN set above. The most important difference in the actual recognition settings for the UNIPEN digit experiment above are: (1) the use of virtual reference segments to determine size and position (cf. Section 4.2.3); (2) use of the segment-by-segment strategy for the segment graph construction (cf. Section 5.3.1); (3) the use of the Relational x, y features from Table 4.1 on page 51 and (4) the beam widths - in particular the diacritic beam since there are no delayed strokes in the isolated digit set. Unfortunately the data set used for testing is a proprietary data set supplied by Zi Decuma and it is not publicly available, but a recent Master Thesis with a current state-of-the-art recognition technique has enabled recognition performance comparison [10]. In this test the strategy presented here provides comparable recognition accuracy (94.1%) to the state-of-the art technique (94.4%) based on similarity matching (cf. Section 2.3.2) complemented with a multi-stage combination paradigm (cf. Section 2.7) as seen in Table 9.6. It remains to be seen whether this means that the settings used for the Arabic

k-Means	top-1	top-2	top-3
0	91.9%	96.7%	97.6%
5	94.1%	96.5%	97.7%

TABLE 9.6: Recognition results for isolated Arabic single characters test set

k-Means	Dictionary	top-1	top-2	top-3
5	65k	87.5%	-%	-%
5	∞	54.9%	-%	-%

TABLE 9.7: Recognition results for online Arabic cursive words with varying dictionary sizes and for two different databases on the training independent set of data

single characters have superior discriminative power or whether the segmental matching of this thesis is naturally better at the slightly more complex problem of Arabic single characters as opposed to Latin digits.

9.4 Unconstrained Character Recognition

Unconstrained character recognition here means that data was collected by instructing users to write a specific word without imposing restrictions on how to write (as opposed to the single character case). For Arabic this in principle means that unconstrained character recognition is the same as cursive Arabic handwriting recognition. Since dynamic linguistical methods (cf. Section 7.4) have not yet been implemented in the experimental system used here, the dataset was filtered only to contain writing such that each individual character was discernible. This implied removal of about 28% of the original set of 2513 samples from 66 words and 39 writers. The tests performed on the remaining set were multi-writer so that a set of 17 words (408 word samples) were used purely for testing and parts of the remaining words were used for training. Word recognition was conducted both with and without the application of lexical knowledge according to the procedure described in Chapter 7. Running recognition without a dictionary corresponds to an infinite dictionary size in Tables 9.7 and 9.8. As for the single character recognition case, experiments have been conducted both with the segment definition database and with a database segment-wise clustered with k-Means ($k = 5$).

It is interesting to note that the segmental clustering technique effectively clus-

k-Means	Dictionary	top-1	top-2	top-3
0	65k	78.6%	86.6%	87.9%
0	∞	40.9%	54.2%	58.9%
5	65k	85.4%	89.8%	90.9%
5	∞	57.6%	69.8%	74.4%

TABLE 9.8: Recognition results for online Arabic cursive words with varying dictionary sizes and for two different databases on the complete set of data (including word samples in training set)

ters characters from the much larger space of all combinatorial variations of segments as previously seen in Figure 8.1. For this reason it is conceivable that the training procedure will be less dependent on the training set. This is given some support from the experiments where in fact recognition accuracy is higher on the test set of word samples not used in training (using a 65k word dictionary).

9.4.1 Varying the beam widths

The three graph algorithms (Alg. 3, 6, 7) in Chapters 5 and 6 all contain parameters in the form of *beam widths* to limit time complexity and memory consumption in the graph structures. This section will deal with experiments conducted with various settings for these parameters. Since word recognition is the most complex problem in view of the graph algorithms only this recognition setup has been used in this experiment and results are reported for tests on the complete data set as in Table 9.8. In view of commercial applications the triplet of recognition accuracy, memory consumption and recognition response time (speed[2]) have all been analyzed as functions of various values for the different beam widths. Both memory consumption values and speed measurements are reported with mean and max values. Naturally a larger beam width imply a slower response time and a more extensive memory consumption. The impact on recognition accuracy is not, however, as straightforward. Since memory and speed consumptions are also extremely dependent on the quality of the actual implemented code itself as well as the architecture of the platform used for testing, the values should not be used as absolute values, but merely values that enable comparison of the effects of different beam widths.

[2]The absolute speed calculated on 4 word representative samples on a ARM9TDMI, 208 MHz processor, comparable to that of many mobile platforms

The segmentation graph beam $B_{\mathfrak{S}}$

The segmentation graph beam $B_{\mathfrak{S}}$ limits the number of edges allowed to each node in the segmentation graph and thereby the size of the set of shape matches ending in the corresponding segmentation point. The effect of varying $B_{\mathfrak{S}}$ while keeping all other beam widths fixed is shown in Table 9.9. As expected memory and response time increases with increasing beam size while there seems to be a fairly protrudent optimum in terms of recognition accuracy both with and without a dictionary. It is likely this decrease in recognition accuracy after the optimum value is reached is caused by increased competition of hypothesis in the recognition graph.

$B_{\mathfrak{S}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
10	226	510	460	620	83.0%	87.1%	87.6%	88.4%
20	311	648	557	720	85.3%	**89.8**%	**90.9**%	92.0%
28	314	722	607	780	**85.4**%	89.8%	90.9%	92.2%
40	350	760	657	820	84.9%	89.5%	90.9%	**92.4**%
60	401	878	703	870	84.0%	88.9%	90.4%	92.2%
Dictionary size = ∞								
10	242	579	433	570	55.8%	67.6%	71.0%	76.7%
20	289	693	480	620	**58.0**%	**70.0**%	73.9%	80.7%
28	316	740	503	640	57.6%	69.8%	**74.4**%	81.1%
40	349	811	530	670	57.5%	69.8%	74.1%	**81.4**%
60	396	877	553	700	57.5%	69.6%	73.8%	80.7%

TABLE 9.9: The effect of varying the SG beam width $B_{\mathfrak{S}}$ while fixing other parameters ($B_{\mathfrak{R}}$=80, B_I=200,$B_{\mathfrak{D}}$=15,B_U=4) on the complete data set.

The recognition graph beam $B_{\mathfrak{R}}$

The beam in the recognition graph limits the number of allowed hypothesis in each node corresponding to the height of the Trie structure shown in Figure 5.8. Increasing this size therefore reduces the risk that the *correct* hypothesis is prematurely excluded. As seen in Table 9.10 this is more of a problem for recognition with an infinite dictionary as the competition in the recognition graph depends on the size of the set of allowed hypothesis.

$B_{\mathfrak{R}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
20	184	350	457	590	82.3%	85.9%	86.9%	87.6%
40	227	468	513	660	84.3%	88.5%	89.3%	90.4%
60	271	589	570	730	85.0%	89.6%	90.5%	91.7%
80	314	722	607	780	85.4%	89.8%	90.9%	92.2%
100	357	842	653	820	**85.5**%	90.0%	91.2%	92.6%
120	400	967	693	880	85.5%	90.0%	91.2%	**92.7**%
140	462	1145	740	930	85.5%	**90.1**%	**91.3**%	92.7%
Dictionary size = ∞								
20	181	357	430	550	55.0%	65.0%	67.9%	71.1%
40	225	492	450	580	56.8%	68.1%	72.2%	77.2%
60	271	628	480	620	57.4%	69.5%	73.8%	79.8%
80	316	761	503	640	57.6%	69.8%	74.4%	81.1%
100	362	890	527	680	57.9%	70.0%	74.7%	81.6%
120	407	1019	557	720	57.9%	70.2%	74.8%	82.0%
140	476	1204	587	760	**58.1**%	**70.3**%	**75.0**%	**82.4**%

TABLE 9.10: The effect of varying the RG beam width ($B_{\mathfrak{R}}$) while fixing other parameters ($B_{\mathfrak{S}}$=28, B_I=200,$B_{\mathfrak{D}}$=15,B_U=4) on the complete data set.

The incomplete beam B_I

The incomplete beam regulates the set of incomplete hypothesis maintained within the segmentation graph. For Algorithm 5 on page 73 implemented for the cursive word recognition all partial matches including noise segments compete in this beam. As seen in Table 9.11 the width of the incomplete beam needs a to be of considerable size (80) not to cause a drastic drop in recognition accuracy. Since the incomplete beam can be reused for every segment (there is no need to store incomplete edges since they are only used as input to continued matching), an increase in the incomplete beam has virtually no effect on memory size but seems scale approximately linearly with response time after the crucial value of 80 has been reached.

B_I	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
40	302	685	517	670	78.6%	82.4%	83.5%	84.9%
80	306	700	553	710	82.8%	87.0%	88.1%	89.3%

continued on next page

continued from previous page

B_I	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
120	309	702	573	730	84.1%	88.6%	89.7%	90.8%
160	311	709	590	750	85.0%	89.4%	90.4%	91.8%
200	314	722	607	780	85.4%	89.8%	90.9%	92.2%
240	316	718	627	800	**85.5**%	**90.0**%	**91.1**%	**92.3**%
Dictionary size = ∞								
40	306	728	423	550	54.9%	66.7%	70.3%	75.4%
80	309	742	450	580	57.0%	68.8%	72.8%	79.0%
120	312	750	463	590	57.0%	69.4%	73.6%	79.7%
160	314	755	483	620	57.5%	**69.8**%	74.1%	80.8%
200	316	761	503	640	**57.6**%	69.8%	**74.4**%	**81.1**%
240	318	750	520	660	57.5%	69.8%	74.1%	80.8%

TABLE 9.11: The effect of varying the incomplete beam width (B_I) while fixing other parameters ($B_{\mathfrak{S}}$=28, $B_{\mathfrak{R}}$=80,$B_{\mathfrak{D}}$=15,B_U=4) on the complete data set.

The diacritic beam $B_{\mathfrak{D}}$

The diacritic beam width is an interesting parameter which has virtually no bearing on recognition speed and only negligible effect on memory consumption. Like for the incomplete beam the recognition accuracy quickly levels off after a certain width (10 in Table 9.12). But for the diacritic beam there also seems to be a local optimum for recognition with infinite dictionary not present for the 65k case. This shows the clear connection between the diacritic beam and the recognition graph beam. Increasing the diacritic beam allows more edges to utilize strokes as diacritics and thereby obtain a lower sorting distance as seen in (6.6) on page 6.6. This increases the competition in the recognition graph of strings by reducing the distance values for a large number of hypothesis. From Table 9.12 it is clear that this does not cause problems for the limited hypothesis space of the 65k dictionary, but certainly for the case with an infinite dictionary.

$B_{\mathfrak{D}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
5	302	684	613	790	68.1%	74.9%	78.0%	83.7%
10	308	697	610	780	82.5%	87.5%	88.8%	90.9%
15	311	716	610	790	85.4%	89.9%	90.9%	92.2%

continued on next page

continued from previous page

$B_{\mathfrak{D}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
18	313	724	607	780	85.4%	90.2%	91.1%	92.0%
20	314	728	610	790	85.6%	90.3%	91.1%	92.1%
25	316	730	610	790	85.8%	90.5%	**91.3**%	**92.3**%
30	317	739	610	780	**86.0**%	**90.6**%	91.3%	92.3%
35	318	740	610	790	86.0%	90.5%	91.3%	92.1%
Dictionary size = ∞								
5	304	718	513	670	46.5%	54.8%	58.2%	65.1%
10	337	746	530	640	56.7%	68.3%	72.4%	79.3%
15	313	754	503	640	57.7%	70.0%	**74.5**%	**81.2**%
18	315	757	503	640	**57.8**%	**70.1**%	74.3%	81.2%
20	316	756	503	640	57.5%	69.9%	74.1%	80.3%
25	318	768	507	650	57.2%	69.1%	73.0%	79.0%
30	319	765	503	640	57.2%	69.0%	72.8%	78.6%
35	320	766	503	640	57.0%	68.8%	72.5%	78.1%

TABLE 9.12: The effect of varying the diacritic beam width ($B_{\mathfrak{D}}$) while fixing other parameters ($B_{\mathfrak{S}}$=28, $B_{\mathfrak{R}}$=80, $B_I = 200$, $B_U = 3$) on the complete data set.

The unmatched edge beam B_U

The unmatched edge beam width B_U corresponds to the number of alternatives for a fixed set of diacritic edges to be used within a given hypothesis. This could correspond to swapping anchors for multiples of the same diacritic for instance as described in Section 6.4.4. As seen in Table 9.13 this parameter has negligible effect both on memory and response time. Although its existence (setting $B_U > 1$) gives an increase in recognition accuracy, increasing the parameter further does not provide better recognition accuracy in these experiments.

B_U	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
1	307	704	607	780	83.2%	88.2%	90.0%	92.1%
2	309	711	607	780	85.3%	89.7%	**90.9**%	**92.2**%
3	311	716	607	780	**85.4**%	**89.9**%	90.9%	92.2%
4	314	722	607	780	85.4%	89.8%	90.9%	92.2%
6	318	733	610	780	85.4%	89.8%	90.9%	92.2%

continued on next page

continued from previous page

B_U	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = ∞								
1	308	741	497	620	56.4%	67.9%	71.4%	79.7%
2	311	748	500	630	**58.0**%	**70.0**%	74.3%	**81.2**%
3	313	754	503	640	57.7%	70.0%	**74.5**%	81.2%
4	316	761	503	640	57.6%	69.8%	74.4%	81.1%
6	322	773	507	650	57.6%	69.8%	74.4%	81.1%

TABLE 9.13: The effect of varying the diacritic unmatched alternative beam width (B_U) while fixing other parameters ($B_{\mathfrak{S}}$=28, $B_{\mathfrak{R}}$=80, B_I = 200, $B_{\mathfrak{D}}$ = 15) on the complete data set.

9.4.2 Optimizing performance

Given the effects of various beam width parameters, it would clearly be preferable to have a system for optimizing such parameters given a set of requirements. Different platforms may have different requirements in terms of computational power and storage. If the requirements are not limiting to the system (like on a PC for instance) it would be beneficial to know the parameter setting to reach optimal recognition accuracy. For other platforms this may be conditioned on memory and/or response time. In this thesis no work has been put into deriving automatic methods to answer these questions but some general directions can be given by studying Table 9.14.

$B_{\mathfrak{S}}/B_I/B_{\mathfrak{R}}/B_{\mathfrak{D}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
Dictionary size = 65k								
10/40/10/5	91	165	310	420	71.0%	73.1%	73.5%	73.6%
10/40/10/10	96	164	307	420	71.1%	73.2%	73.6%	73.4%
20/40/10/10	125	219	340	450	72.9%	75.3%	75.9%	76.2%
20/80/10/10	128	229	360	470	76.6%	79.5%	80.0%	80.3%
10/40/20/10	114	206	300	400	74.8%	77.6%	78.3%	78.7%
10/80/20/10	116	215	343	460	78.5%	82.0%	82.5%	83.0%
10/120/20/10	118	217	363	480	79.0%	82.7%	83.2%	83.7%
10/120/40/10	152	303	387	510	81.1%	85.0%	85.4%	86.1%
20/80/40/10	191	390	387	510	81.1%	84.7%	85.7%	87.0%
10/120/60/10	185	384	407	540	81.8%	85.9%	86.3%	87.2%
20/80/60/15	237	508	467	620	83.1%	87.0%	87.8%	88.6%
20/160/60/20	243	507	503	660	85.2%	89.4%	90.0%	91.0%
20/200/60/25	246	533	517	680	85.5%	89.5%	90.4%	91.3%
20/120/80/15	280	638	520	670	84.3%	88.5%	89.4%	90.4%
20/80/100/15	320	764	530	680	83.0%	87.4%	88.5%	89.4%
30/120/60/15	272	583	537	690	83.7%	88.4%	89.4%	90.4%
20/160/80/20	284	650	543	710	85.5%	89.9%	90.4%	91.6%
20/200/80/25	288	639	557	730	85.9%	90.1%	90.9%	92.0%
30/80/80/10	309	689	560	720	79.6%	84.4%	85.4%	88.1%

continued on next page

continued from previous page

$B_{\mathfrak{S}}/B_I/B_{\mathfrak{R}}/B_{\mathfrak{D}}$	Memory(kB)		Speed(ms)		top-1	top-2	top-3	top-10
	mean	max	mean	max				
20/200/100/35	331	762	597	780	85.9%	90.2%	91.1%	92.2%
30/240/80/25	327	756	637	810	86.0%	90.9%	91.6%	92.5%
30/200/100/35	372	883	660	840	86.3%	90.7%	91.5%	92.6%
30/240/100/40	375	892	677	860	86.4%	91.1%	91.8%	92.8%
40/200/100/30	402	924	700	870	86.2%	90.9%	91.7%	92.8%
40/280/100/40	410	949	743	920	86.4%	91.1%	91.9%	92.8%
40/240/120/40	453	1074	760	950	86.6%	91.4%	92.1%	93.2%
40/280/120/30	453	1077	790	980	86.5%	91.4%	92.1%	93.2%
50/240/120/30	486	1105	797	990	86.7%	91.7%	92.5%	93.5%
40/240/140/30	500	1203	813	1020	86.6%	91.4%	92.3%	93.4%
40/280/140/40	506	1216	840	1040	86.6%	91.4%	92.3%	93.4%
50/240/140/40	537	1239	850	1060	86.8%	91.6%	92.5%	93.5%
50/240/160/40	584	1369	903	1030	86.8%	91.6%	92.5%	93.5%
100/320/400/100	1303	3179	1893	2390	87.1%	92.2%	93.3%	94.4%
Dictionary size = ∞								
10/40/10/5	95	162	297	380	48.9%	55.8%	57.0%	58.6%
10/40/10/10	92	167	293	380	49.0%	55.8%	57.1%	58.7%
20/40/10/10	120	225	323	420	49.9%	57.2%	58.9%	61.1%
20/80/10/10	122	230	343	440	51.8%	59.5%	61.2%	63.6%
10/40/20/10	113	222	320	440	52.2%	61.0%	63.2%	65.5%
10/80/20/10	114	267	327	430	53.8%	62.9%	65.3%	68.3%
10/120/20/10	117	232	343	440	53.9%	63.1%	65.5%	68.6%
10/120/40/10	155	339	367	470	55.3%	66.2%	69.3%	74.0%
20/80/40/10	190	428	367	470	56.5%	67.4%	70.8%	76.2%
10/120/60/10	194	443	383	490	56.0%	67.5%	70.6%	76.2%
20/80/60/15	240	574	403	520	57.2%	68.4%	71.9%	77.6%
20/160/60/20	245	565	437	560	56.8%	68.4%	71.5%	77.5%
20/200/60/25	249	580	457	590	56.9%	68.1%	71.6%	77.0%
20/120/80/15	287	696	443	570	57.4%	69.3%	73.0%	79.3%
20/80/100/15	330	836	453	590	57.8%	69.3%	73.1%	79.2%
30/120/60/15	272	626	447	570	56.6%	68.8%	72.8%	78.5%
20/160/80/20	290	691	460	590	57.9%	69.5%	73.1%	79.5%
30/80/80/10	312	740	450	580	54.9%	65.7%	69.5%	76.3%
20/200/80/25	294	707	480	620	57.6%	69.1%	72.8%	78.9%
20/200/100/35	341	835	500	650	57.9%	69.6%	73.3%	79.2%
30/240/80/25	329	790	523	680	57.1%	69.1%	73.3%	79.3%
30/200/100/35	376	917	537	690	57.4%	69.3%	73.3%	79.3%
30/240/100/40	379	925	547	700	57.2%	69.2%	73.3%	79.2%
40/200/100/30	403	957	557	700	57.7%	69.6%	73.6%	80.3%
40/280/100/40	410	975	587	740	57.2%	69.2%	73.0%	93.2%
40/240/120/40	456	1103	597	750	57.7%	69.7%	73.9%	80.4%
40/280/120/30	455	1105	617	780	57.9%	70.0%	74.3%	81.1%
50/240/120/30	486	1137	617	780	57.9%	70.0%	74.5%	81.5%
40/240/140/30	505	1244	623	780	58.1%	70.0%	74.5%	81.8%
40/280/140/40	510	1225	640	810	57.8%	69.8%	74.2%	81.1%
50/240/140/40	587	1410	670	850	58.0%	70.2%	74.6%	81.6%
100/320/400/100	1313	3189	1183	1550	58.2%	70.7%	75.0%	83.0%

TABLE 9.14: The effect of varying graph beams simultaneously on the complete data set.

As seen in Figure 9.1, the plots of recognition accuracy as a function of memory usage clearly levels off at an early level. Since memory usage is approximately proportional to response time as seen in Figure 9.5, it is expected that the curve levels off for corresponding values in recognition accuracy. Even though

the beam widths are increased to significantly larger numbers as seen in the last rows of dictionary and non-dictionary results in Table 9.14, recognition accuracy only improves marginally. These upper limitations are not caused by restrictive hypothesis beam search. Instead, the upper limit especially of top-10 results should be interpreted as the limitations put on recognition accuracy of other components in the recognition system. These may range from preprocessing and segmentation errors (failing to find a segmentation point) to missing database coverage, unfavorably initialized distance function weights and noise distance parameters. From a performance perspective it is therefore quite assuring to see that the graphs in Figures 9.1-9.4 level off quickly, since this gives indications that it is possible to achieve a higher hitrate with the strategy presented in the thesis without compromising memory and speed performance. The points in the plots corresponding to the maximization of hitrate while minimizing memory and/or speed have been marked with a red frame in Table 9.14.

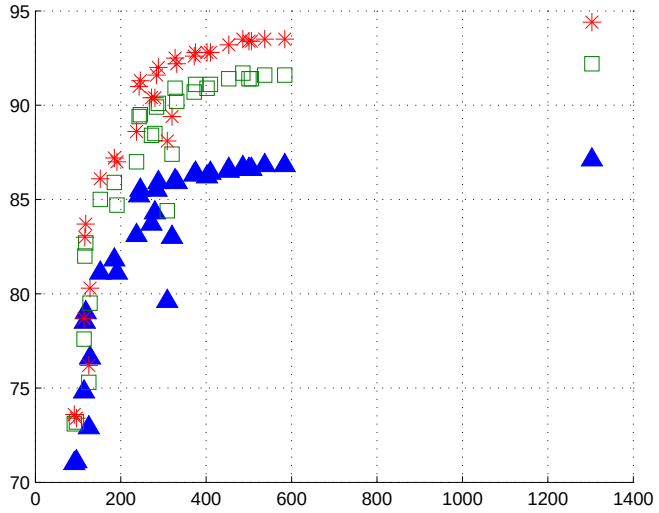


FIGURE 9.1: Plot with recognition accuracy as a function of memory usage for recognition with a dictionary of 65k words. The three markers are the different recognition accuracy levels (Triangle) Top-1, (Square) Top-2 and (Star) Top-10.

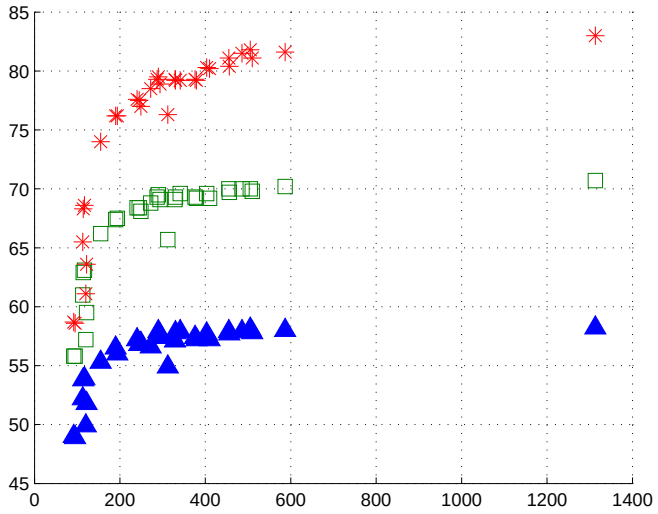


FIGURE 9.2: Plot with recognition accuracy as a function of memory usage for recognition without a dictionary. The three markers are the different recognition accuracy levels (Triangle) Top-1, (Square) Top-2 and (Star) Top-10.

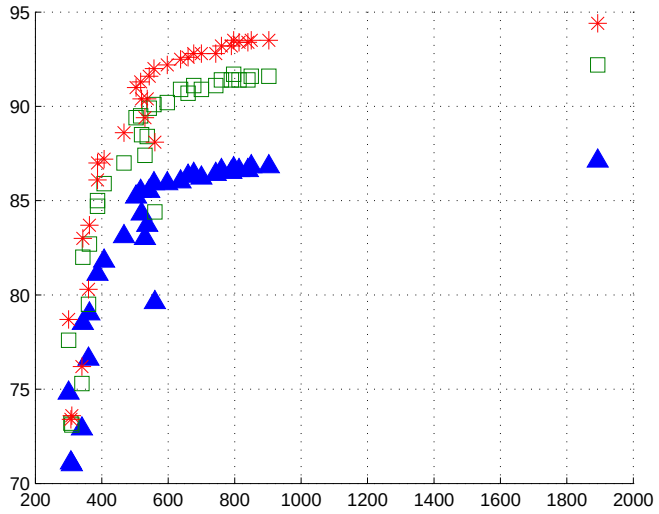


FIGURE 9.3: Plot with recognition accuracy as a function of response time for recognition with a dictionary of 65k words. The three markers are the different recognition accuracy levels (Triangle) Top-1, (Square) Top-2 and (Star) Top-10.

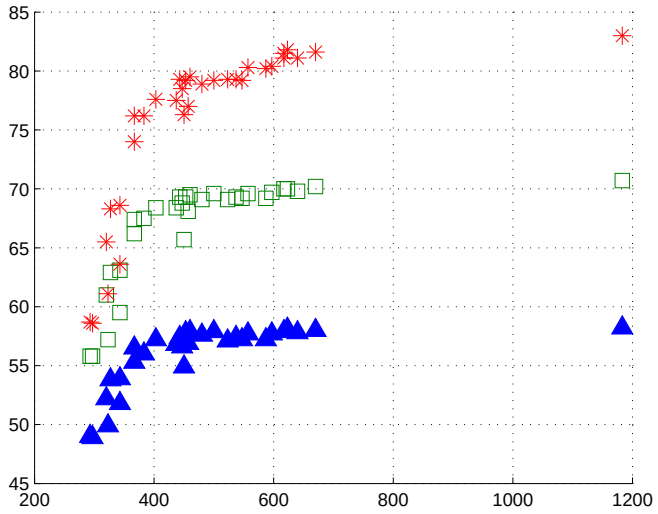


FIGURE 9.4: Plot with recognition accuracy as a function of response time for recognition without a dictionary. The three markers are the different recognition accuracy levels (Triangle) Top-1, (Square) Top-2 and (Star) Top-10.

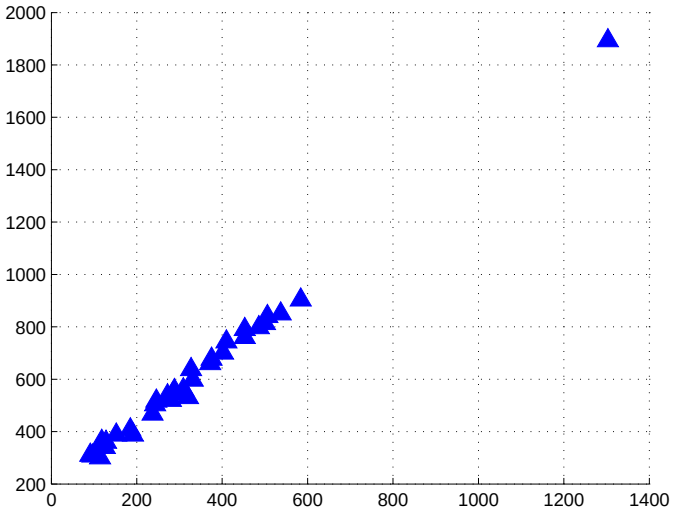


FIGURE 9.5: Plot with memory as a function of response time.

CHAPTER 10

Conclusions and Future Prospects

I think and think for months and years. Ninety-nine times, the conclusion is false. The hundredth time I am right.
Albert Einstein

IT HAS BEEN SHOWN in this thesis that it is possible to construct a fast and effective recognition system with an additive template matching applicable to the unconstrained handwriting recognition problem. Neither the algorithms and the system presented here nor the experimental results should be thought of as the products of optimizations maximizing the full potential of the proposed recognition strategy. On the contrary, each part of the strategy has just been defined in its most simple realization. The aim of this has been to produce all required parts of a complete system within a limited time frame. Yet, the experimental part reveals surprisingly powerful recognition results even with a completely hand-built template database. In the early days of on-line handwriting recognition, template based methods were very popular but have lately grown out of fashion. This is especially true for the connected character recognition problem. Although some recognition systems in the past have used templates for recognition of cursive writing it is increasingly difficult to deduce the exact recognition settings from older publications. Hopefully the work presented in this thesis can inspire rejuvenated efforts in this area.

As stated above, several parts of the recognition system have just been defined in their most simple form and this leaves several obvious areas of improvements for future research. Some of the most striking items are listed below:

- The segmentation strategy. In the strive for a completely script independent recognition engine (apart from template database properties), the segmentation strategy has to be able to deal with alternative writing directions. This in turn imposes a requirement on a more generic way of finding segmentation points, possibly such as those used to identify feature points in on-line signature verification.

- Dynamic lexical lookup as sketched in Section 7.4. The current system can not handle degraded shapes in samples. The reason for this is that the recognition hypotheses are restricted to the shape matching information made available by the segmentation graph.

- Optimization of weights in the segmental distance function in Section 4.3

- Tuning of settings for the noise distance in Section 5.4

- Inclusion of various parameters used in preprocessing, such as baseline adjustment and slant correction, into the recognition system at a local template level.

Successful treatment of the above mentioned issues will not only increase recognition accuracy and relax constraints on neatness of writing, compared to the

system in this thesis. It would also provide a versatile and script independent system for applications to any on-line shape sequence segmentation problem.

Bibliography

[1] Ahmad, A. R., Khalia, M., Viard-Gaudin, C., and Poisson, E. Online handwriting recognition using support vector machine. In *TENCON 2004. 2004 IEEE Region 10 Conference* (2004), vol. A, pp. 311–314 Vol. 1.

[2] Alimoglu, F., and Alpaydin, E. Combining multiple representations and classifiers for pen-based handwritten digit recognitio. In *Proc. of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany* (Washington, DC, USA, August 1997), IEEE Computer Society, pp. 637–640.

[3] Alon, J., Athitsos, V., and Sclaroff, S. Online and offline character recognition using alignment to prototypes. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 839–843.

[4] Andersson, J. Hidden markov model based handwriting recognition. Master's thesis, Dept. of Mathematics, Lund Institute of Technology, Sweden, 2002.

[5] Anquetil, E., and Bouchereau, H. Integration of an on-line handwriting recognition system in a smart phone device. In *Proc. 16th International Conference on Pattern Recognition* (Washington, DC, USA, 2002), vol. 3, IEEE Computer Society, p. 30192.

[6] Anquetil, E., and Lorette, G. Perceptual model of handwriting drawing application to the handwriting segmentation problem. In *Proc. of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany* (Washington, DC, USA, 1997), IEEE Computer Society, p. 112.

[7] Bahlmann, C. *Advanced Sequence Classification Techniques Applied to Online Handwriting Recognition*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2005.

[8] Bahlmann, C., and Burkhardt, H. The writer independent online handwriting recognition system *frog on hand* and cluster generative statistical dynamic time warping. *IEEE Trans. Pattern Analysis and Machine Intelligence 26*, 3 (Mar. 2004), 299–310.

[9] Bahlmann, C., Haasdonk, B., and Burkhardt, H. On-line handwriting recognition with support vector machines - a kernel approach. In *Proc. of the Eighth International Workshop on Frontiers in Handwriting Recognition* (August 2002), pp. 49–54.

[10] Bakhtiari-Haftlang, C. Arabic online handwriting recognition. Master's thesis, Dept. of Mathematics, Lund Institute of Technology, Sweden, 2007.

[11] Bellegarda, E. J., Bellegarda, J. R., Nahamoo, D., and Nathan, K. A fast statistical mixture algorithm for on-line handwriting recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence 16*, 12 (1994), 1227–1233.

[12] Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Analysis and Machine Intelligence 24*, 24 (2002), 509–522.

[13] Bengio, Y., and LeCun, Y. word normalization for on-line handwritten word recognition. In *Proc. 12th International Conference on Pattern Recognition* (Jerusalem, October 1994), IAPR, Ed., vol. II, IEEE, pp. 409–413.

[14] Bengio, Y., LeCun, Y., Nohl, C., and Burges, C. Lerec: a nn/hmm hybrid for on-line handwriting recognition. *Neural Comput. 7*, 6 (1995), 1289–1303.

[15] Berthilsson, R., and Åström, K. Extension of affine shape. *J. Math. Imaging Vis. 11*, 2 (1999), 119–136.

[16] Biadsy, F., El-Sana, J., and Habash, N. Online arabic handwriting recognition using hidden markov models. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 85–90.

[17] Bishop, C. *Artificial Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[18] Bote-Lorenzo, M. L., Dimitriadis, Y. A., and Gómez-Sánchez, E. A hybrid two-stage fuzzy artmap and lvq neuro-fuzzy system for online handwriting recognition. In *ICANN '02: Proceedings of the International Conference on Artificial Neural Networks* (London, UK, 2002), Springer-Verlag, pp. 438–443.

[19] Brault, J.-J., and Plamondon, R. Segmenting handwritten signatures at their perceptually important points. *IEEE Trans. Pattern Analysis and Machine Intelligence 15*, 9 (1993), 953–957.

[20] Brown, M., and Ganapathy, S. Preprocessing techniques for cursive word recognition. *Pattern Recognition 16*, 5 (1983), 447–458.

[21] Caillault, E., and Viard-Gaudin, C. Using segmentation constraints in an implicit segmentation scheme for on-line word recognition. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 607–612.

[22] Caillault, E., Viard-Gaudin, C., and Ahmad, A. R. Ms-tdnn with global discriminant trainings. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 856–861.

[23] Carbonnel, S., and Anquetil, E. Lexical post-processing optimization for handwritten word recognition. In *Proc. of the 7th International Conference on Document Analysis and Recognition* (2003), pp. 477–481.

[24] Carbonnel, S., and Anquetil, E. Lexicon organization and string edit distance learning for lexical post-processing in handwriting recognition. In *Proc. of the Ninth International Workshop on Frontiers in Handwriting Recognition* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 462–467.

[25] Casey, R., and Lecolinet, E. A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence 18*, 7 (July 1996), 690–706.

[26] Chen, D. Y., Mao, J., and Mohiuddin, K. M. An efficient algorithm for matching a lexicon with a segmentation graph. In *Proc. of the 5th International Conference on Document Analysis and Recognition* (1999), pp. 543–546.

[27] Chen, M., Kundu, A., and Srihari, S. Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE Trans. Image Processing 4*, 12 (December 1995), 1675–1688.

[28] Connell, S. *Online Handwriting Recognition Using Multiple Pattern Class Models*. PhD thesis, The Michigan State University, 2000.

[29] Connell, S. D., and Jain, A. K. Template-based online character recognition. *Pattern Recognition 34*, 1 (2001), 1–14.

[30] Costin, H., Ciobanu, A., and Todirascu, A. Handwritten script recognition system for languages with diacritic signs. In *Proceedings of The 1998 IEEE International Joint Conference on Neural Networks* (1998), pp. 1188–1193.

[31] Danowsky, D. Cyrillic handwriting recognition using support vector machines. Master's thesis, Dept. of Mathematics, Lund Institute of Technology, Sweden, 2006.

[32] Deepu, V., Madhvanath, S., and Ramakrishnan, A. G. Principal component analysis for online handwritten character recognition. In *Proc. 17th International Conference on Pattern Recognition* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 327–330.

[33] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[34] Do, T. M. T., and Artiéres, T. Conditional random fields for online handwriting recognition. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 197–202.

[35] Dolfing, H., and Haeb-Umbach, R. Signal representations for hidden markov model based on-line handwriting recognition. In *ICASSP '97: Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '97) - Volume 4* (Washington, DC, USA, 1997), IEEE Computer Society, pp. 3385–3388.

[36] Duneau, L., and Dorizzi, B. Online cursive script recognition: A user-adaptive system for word identification. *Pattern Recognition 29*, 12 (December 1996), 1981–1994.

[37] Elgammal, A., and Ismail, M. A. A graph-based segmentation and feature-extraction framework for arabic text recognition. In *Proc. of the 6th International Conference on Document Analysis and Recognition* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 622–626.

[38] Eriksson, A. P., and Åström, K. On the bijectivity of thin-plate splines. In *Proceedings SSBA'05 Symposium on Image Analysis* (Malmö, March 2005), A. Heyden, Ed., SSBA, pp. 109–112.

[39] Favata, J. T. Offline general handwritten word recognition using an approximate beam matching algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence 23*, 9 (2001), 1009–1021.

[40] Forney, G. D. The viterbi algorithm. In *Proc. of the IEEE* (1973), pp. 268–278.

[41] Fredkin, E. Trie memory. *Communications of the ACM 3*, 9 (Sept. 1960), 490–499.

[42] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci. 55*, 1 (1997), 119–139.

[43] Funada, A., Muramatsu, D., and Matsumoto, T. The reduction of memory and the improvement of recognition rate for hmm on-line handwriting recognition. In *Proc. of the Ninth International Workshop on Frontiers in Handwriting Recognition* (2004), pp. 383–388.

[44] Gader, P. D., Keller, J. M., Krishnapuram, R., Chiang, J.-H., and Mohamed, M. A. Neural and fuzzy methods in handwriting recognition. *IEEE Computer 30*, 2 (1997), 79–86.

[45] Goodrich, M. T. Efficient piecewise-linear function approximation using the uniform metric: (preliminary version). In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry* (New York, NY, USA, 1994), ACM Press, pp. 322–331.

[46] Gran, F. Pattern recognition using support vector machines. Master's thesis, Dept. of Mathematics, Lund Institute of Technology, Sweden, 2002.

[47] Guerfali, W., and Plamondon, R. Normalizing and restoring on-line handwriting. *Pattern Recognition 26*, 3 (1993), 419–431.

[48] Guyon, I., Schomaker, L., Plamondon, R., Liberman, M., and Janet, S. Unipen project of on-line data exchange and recognizer benchmarks. In *Proc. 12th International Conference on Pattern Recognition* (Jerusalem, Israel, October 1994), pp. 29–33.

[49] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer, New York, 2001.

[50] Higgins, C., and Ford, D. On-line recognition of connected hnadwriting by segmentation and tempalte matching. In *Proc. 11th International Conference on Pattern Recognition* (1992), pp. 200–203.

[51] Hill, A., Cootes, T. F., and Taylor, C. J. Active shape models and the shape approximation problem. *Image Vision Comput. 14*, 8 (1996), 601–607.

[52] Ho, T. K., Hull, J. J., and Srihari, S. N. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Analysis and Machine Intelligence 16*, 1 (1994), 66–75.

[53] Hu, J., Lim, S., and Brown, M. K. Writer independent on-line handwriting recognition using an hmm approach. *Pattern Recognition*, 33 (2000), 133–147.

[54] Jäger, S., Liu, C.-L., and Nakagawa, M. The state of the art in japanese online handwriting recognition compared to techniques in western handwriting recognition. *IJDAR 6*, 2 (2003), 75–88.

[55] Jäger, S., Manke, S., and Waibel, A. Npen++: An on-line handwriting recognition system. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, Netherlands* (September 2000), pp. 249–260.

[56] Jain, A. K., Murthy, M. N., and Flynn, P. J. Data clustering: A review. Tech. Rep. MSU-CSE-00-16, Department of Computer Science, Michigan State University, East Lansing, Michigan, August 2000.

[57] Kassel, R. *A Comparison of Approaches to On-Line Handwritten Character Recognition*. PhD thesis, Massachussetts Institute of Technology, 1995.

[58] Klassen, T., and Heywood, M. Towards the on-line recognition of arabic characters. In *Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02* (Hilton Hawaiian Village Hotel, Honolulu, Hawaii, 12-17 May 2002), IEEE Press, pp. 1900–1905.

[59] Kohonen, T. The self-organizing map. *Proc. IEEE 78*, 9 (1990), 1464–1480.

[60] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., and Torkkola, K. Lvq_pak: The learning vector quantization program package. Technical Report A30 FIN-02150, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996.

[61] Kosmala, A., and Rigoll, G. Tree-based state clustering using self-organizing principles for large vocabulary on-line handwriting recognition. In *Proc. 14th International Conference on Pattern Recognition* (Washington, DC, USA, 1998), vol. 2, IEEE Computer Society, p. 1313.

[62] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning* (2001), pp. 282–289.

[63] LeCun, Y., Huang, F.-J., and Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04* (2004), IEEE Press.

[64] Lee, J. J., Kim, J., and Kim, J. H. Data-driven design of hmm topology for on-line handwriting recognition. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, Netherlands* (September 2000).

[65] Li, X., Parizeau, M., and Plamondon, R. Segmentation and reconstruction of on-line handwritten scripts. *Pattern Recognition 31*, 6 (1998), 675–684.

[66] Liu, C.-L., Jäger, S., and Nakagawa, M. Online recognition of chinese characters: The state-of-the-art. *IEEE Trans. Pattern Analysis and Machine Intelligence 26*, 2 (2004), 198–213.

[67] Liwicki, M., and Bunke, H. IAM-OnDB – an on-line English sentence database acquired from handwritten text on a whiteboard. In *Proc. 8th Int. Conf. on Document Analysis and Recognition* (2005), vol. 2, pp. 956–961.

[68] Liwicki, M., and Bunke, H. HMM-based on-line recognition of handwritten whiteboard notes. In *Proc. 10th Int. Workshop on Frontiers in Handwriting Recognition* (2006), pp. 595–599.

[69] Liwicki, M., and Bunke, H. Combining on-line and off-line systems for handwriting recognition. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* (2007), vol. 1, pp. 372–376.

[70] Liwicki, M., and Bunke, H. Feature selection for on-line handwriting recognition of whiteboard notes. In *Proc. 13th Conf. of the Int. Graphonomics Society* (2007), pp. 101–105.

[71] Lucas, S. Efficient best-first dictionary search given graph-based input. In *Proc. 15th International Conference on Pattern Recognition* (Piscataway, NJ, 2000), vol. 1, IEEE Press, pp. 434–437. Barcelona, Spain, September 3–8, 2000.

[72] Lucas, S. Efficient graph-based dictionary search and its application to text-image searching. *Pattern Recognition Letters 22* (April 2001), 551–562(12).

[73] Manke, S., Finke, M., and Waibel, A. A fast search technique for large vocabulary on-line handwriting recognition. In *Proc. of the International Workshop on Frontiers in Handwriting Recognition (IWFHR)* (Colchester, 1996).

[74] Marinai, S., Gori, M., and Soda, G. Artificial neural networks for document analysis and recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence 27*, 1 (2005), 23–35.

[75] Marshall, S. Review of shape coding techniques. *Image and Vision Comp. 7*, 4 (1989), 281–294.

[76] Mezghani, N., Mitiche, A., and Cheriet, M. A new representation of character shape and its use in on-line character recognition by a self-organizing map. In *IEEE International Conference on Image Processing (ICIP'04)* (Singapore, October 2004), pp. 2123–2126.

[77] Mitoma, H., Uchida, S., and Sakoe, H. Online character recognition based on elastic matching and quadratic discrimination. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 36–40.

[78] Morasso, P., Barberis, L., Pagliano, S., and Vergano, D. Recognition experiments of cursive dynamic handwriting with self-organizing networks. *Pattern Recognition 26*, 3 (1993), 451–460.

[79] Morwing, J. Recognition of cursive handwriting. Master's thesis, Dept. of Mathematics, Lund Institute of Technology, Sweden, 2001.

[80] Nakagawa, M., Akiyama, K., Tu, L. V., Homma, A., and Kigashiyama, T. Robust and highly customizable recognition of on-line handwritten japanese characters. In *Proc. 13th International Conference on Pattern Recognition* (Washington, DC, USA, 1996), IEEE Computer Society, pp. 269–273.

[81] Nakagawa, M., Higashiyama, T., Yamanaka, Y., Sawada, S., Higashigawa, L., and Akiyama, K. On-line handwritten character pattern database sampled in a sequence of sentences without any writing instructions. In *Proc. of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany* (1997), pp. 376–381.

[82] Neskovic, P., and Cooper, L. Neural network-based context driven recognition of on-line cursive script. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, Netherlands* (September 2000), pp. 353–362.

[83] Neskovic, P., Davis, P. C., and Cooper, L. Interactive parts model: an application to recognition of on-line cursive script. *Advances in Neural Information Processing Systems* (2000), 974–980.

[84] Niels, R. Dynamic time warping - an intuitive way of handwriting recognition? Master's thesis, Radboud University Nijmegen, 2005.

[85] Oh, J. *An On-Line Handwriting Recognizer with Fisher Matching, Hypotheses Propagation Network and Context Constraint Models.* PhD thesis, New York University, 2001.

[86] Parizeau, M., Lemieux, A., and Gagne, C. Character recognition experiments using unipen data. In *Proc. of the 6th International Conference on Document Analysis and Recognition* (Los Alamitos, CA, USA, 2001), vol. 00, IEEE Computer Society, pp. 481–485.

[87] Parizeau, M., and Plamondon, R. A handwriting model for syntactic recognition of cursive script. In *Proc. 11th International Conference on Pattern Recognition* (August 31 to September 3 1992), vol. II, pp. 308–312.

[88] Parizeau, M., and Plamondon, R. Machine vs humans in a cursive script reading experiment without linguistic knowledge. In *Proc. 12th International Conference on Pattern Recognition* (1994), pp. 93–98.

[89] Parizeau, M., and Plamondon, R. A fuzzy-syntactic approach to allograph modeling for cursive script recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence 17*, 7 (1995), 702–712.

[90] Pawalka, R. K. *An algorithm toolbox for on-line cursive script recognition.* PhD thesis, The Nottingham Trent University, 1995.

[91] Perrone, M., and Connell, S. K-means clustering for hidden markov models. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, Netherlands* (September 2000), pp. 229–238.

[92] Plamondon, R., and Srihari, S. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Analysis and Machine Intelligence 22*, 1 (January 2000), 63–84.

[93] Prevost, L., and Milgram, M. Modelizing character allographs in omni-scriptor frame: a new non-supervised clustering algorithm. *Pattern Recognition Letters 21*, 4 (2000), 295–302.

[94] Qian, G. An engine for cursive handwriting interpretation. In *Proceedings of the 11th International Conference on Tools with Articificial Intelligence* (nov 1999), pp. 271–278.

[95] Quiniou, S., and Anquetil, E. Use of a confusion network to detect and correct errors in an on-line handwritten sentence recognition system. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* (Washington, DC, USA, 2007), vol. 1, IEEE Computer Society, pp. 382–386.

[96] Rabiner, L. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE 77*, 2 (1989), 257–286.

[97] Ratzlaff, E. H. Methods, report and survey for the comparison of diverse isolated character recognition results on the unipen database. In *Proc. of the 7th International Conference on Document Analysis and Recognition* (Los Alamitos, CA, USA, 2003), vol. 01, IEEE Computer Society, pp. 623–628.

[98] Rigoll, G., Kosmala, A., and Willett, D. *A Systematic Comparison of Advanced Modeling Techniques for Very Large Vocabulary On-line Curisve Handwriting Recognition.* World Scientific, 1999, ch. 2, pp. 69–78.

[99] Ripley, B. *Pattern Recognition and Neural Networks.* Press Syndicate of the University of Cambridge, Cambridge, 1996.

[100] Rowley, H., Goyal, M., and Bennett, J. The effect of large training set sizes on online japanese kanji and english cursive recognizers. In *Proc. of the Eighth International Workshop on Frontiers in Handwriting Recognition* (Washington, DC, USA, 2002), IEEE Computer Society, p. 36.

[101] Schenkel, M., and Guyon, I. On-line cursive script recognition using time delay networks and hidden markov models. *Machine Vision and Applications 8* (1995), 215–223.

[102] Schwenk, H., and Bengio, Y. Adaboosting neural networks: Application to on-line character recognition. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks* (London, UK, 1997), Springer-Verlag, pp. 967–972.

[103] Seni, G., and Seybold, J. *Diacritical Processing Using Efficient Accounting Procedures in a Forward Search.* World Scientific, 1999, ch. 2, pp. 49–58.

[104] Seni, G., Srihari, R. K., and Nasrabadi, N. Large vocabulary recognition of on-line handwritten cursive words. *IEEE Trans. Pattern Analysis and Machine Intelligence 18*, 7 (1996), 757–762.

[105] Shi, D., Gunn, S. R., and Damper, R. I. A radical approach to handwritten chinese character recognition using active handwriting models. In *CVPR (1)* (2001), pp. 670–675.

[106] Simard, P., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *Proc. of the 7th International Conference on Document Analysis and Recognition* (2003), pp. 958–962.

[107] Sin, B.-K., Ha, J.-Y., Oh, S.-C., and Kim, J. H. Network-based approach to online cursive script recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B 29*, 2 (1999), 321–328.

[108] Sin, B.-K., and Kim, J. H. Ligature modeling for online cursive script recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence 19*, 6 (1997), 623–633.

[109] Slavík, P., and Govindaraju, V. Equivalence of different methods for slant and skew corrections in word recognition applications. *IEEE Trans. Pattern Analysis and Machine Intelligence 23*, 3 (2001), 323–326.

[110] Sridhar, M., Mandalapu, D., and Patel, M. Active-dtw:a generative classifier that combines elastic matching with active shape modeling for online handwritten character recognition. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (La Baule, France, October 2006), vol. 1, pp. 193–196.

[111] Stefano, C. D., Garutto, M., and Marcelli, A. A saliency-based multiscale method for on-line cursive handwriting shape description. In *Proc. of the Ninth International Workshop on Frontiers in Handwriting Recognition* (2004), pp. 124–129.

[112] Sternby, J. On-line signature verification by explicit solution to the point correspondence problem. In *First International Conference on Biometric Authentication* (july 2004), pp. 569 – 576.

[113] Sternby, J. Core points - variable and reduced parameterization for symbol recognition. Tech. rep., Centre for Mathematical Sciences, 2005. Licentiate Thesis in Mathematical Sciences 2005:7.

[114] Sternby, J. Frame deformation energy matching of on-line handwritten characters. In *Proceedings of the 10th Iberoamerican Congress on Pattern Recognition* (Havanna, Cuba, 2005), pp. 128–137.

[115] Sternby, J. Structurally based template matching of on-line handwritten characters. In *Proc. of the British Machine Vision Conference 2005* (2005), pp. 250–259.

[116] Sternby, J. An additive single character recognition method. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 417–422.

[117] Sternby, J. Class dependent cluster refinement. In *Proc. 18th International Conference on Pattern Recognition* (Los Alamitos, CA, USA, 2006), vol. 2, IEEE Computer Society, pp. 833–836.

[118] Sternby, J. Prototype selection methods for on-line hwr. In *Proc. of the Tenth International Workshop on Frontiers in Handwriting Recognition* (2006), pp. 157–160.

[119] Sternby, J. Graph based shape modeling for on-line character recognition. Manuscript.

[120] Sternby, J., Andersson, J., Morwing, J., and Friberg, C. On-line arabic handwriting recognition with templates. In *Proc. of the First International Conference on Frontiers in Handwriting Recognition* (2008). Accepted.

[121] Sternby, J., and Ericsson, A. Core points - a framework for structural parameterization. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 217–221.

[122] Sternby, J., and Friberg, C. The recognition graph - language independent adaptable on-line cursive script recognition. In *Proc. of the 8th International Conference on Document Analysis and Recognition* (2005), pp. 14–18.

[123] Sternby, J., and Holtsberg, A. Core points for segmentation and recognition of on-line cursive script. In *Proceedings SSBA'05 Symposium on Image Analysis* (Malmö, March 2005), A. Heyden, Ed., SSBA, pp. 37–40.

[124] Tappert, C. Cursive script recognition by elastic matching. *IBM Journal of Research and Development 12* (1982), 765–771.

[125] Tappert, C. C., Suen, C. Y., and Wakahara, T. The state of the art in online handwriting recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence 12*, 8 (1990), 787–808.

[126] Vapnik, V. N. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1995.

[127] Varadarajan, K. R. Approximating monotone polygonal curves using the uniform metric. In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry* (New York, NY, USA, 1996), ACM Press, pp. 311–318.

[128] Viard-Gaudin, C., Lallican, P. M., Binter, P., and Knerr, S. The ireste on/off (ironoff) dual handwriting database. In *Proc. of the 5th International Conference on Document Analysis and Recognition* (Washington, DC, USA, 1999), IEEE Computer Society, p. 455.

[129] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*, 2 (1967), 260–269.

[130] Vuori, V. Adaptation in on-line recognition of handwriting. Master's thesis, Helsinki University of Technology, 1999.

[131] Vuori, V. Clustering writing styles with a self-organizing map. In *Proc. of the Ninth International Workshop on Frontiers in Handwriting Recognition* (2004), pp. 345–350.

[132] Vuori, V., Laaksonen, J., Oja, E., and Kangas, J. On-line adaptation in recognition of handwritten alphanumeric characters. In *Proc. of the 5th International Conference on Document Analysis and Recognition* (September 1999), pp. 792–795.

[133] Vuurpijl, L., Niels, R., van Erp, M., Schomaker, L., and Ratzlaff, E. Verifying the unipen devset. In *Proc. of the Ninth International Workshop on Frontiers in Handwriting Recognition* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 586–591.

[134] Vuurpijl, L., and Schomaker, L. Finding structure in diversity: a hierarchical clustering method for the categorization of allographs in handwriting. In *Proc. of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany* (1997), pp. 387–393.

[135] Vuurpijl, L., and Schomaker, L. Two-stage character classification: A combined approach of clustering and support vector classifiers. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, Netherlands* (Sept. 2000).

[136] Wakahara, T., and Odaka, K. On-line cursive kanji character recognition using stroke-based affine transformation. *IEEE Trans. Pattern Analysis and Machine Intelligence 19*, 12 (1997), 1381–1385.

[137] Wang, J., Wu, C., Xu, Y.-Q., and Shum, H.-Y. Combining shape and physical models for online cursive handwriting synthesis. *IJDAR* (2004).

Index

active shape, **16**
alignment
  function, 41
  problem, 32
allographs, 108
arclength, *see* parameterization
arctypes, 39

beam
  diacritic, 89, 130
  incomplete, 129
  recognition graph, 128
  search, 71
  segmentation graph, 72, 128
  unmatched, 94, 131
boosting, 24

clustering, **12**, 108
  discriminative, 109
  generative, 109
combination
  multi-stage, 24
  multiple experts, 23
CRF, 15

database, 70

DCM-DTW, 40
Dijkstra
  Curve Maximization, **36**
  shortest path, 69
distance
  additive, **55**
  function, 54
  sorting, 91
Dynamic Programming, 18
Dynamic Time Warping, 10, 18

Elastic Matching, 18

feature
  frame, **50**
  segmental, **52**
  space, **49**
features, 22
frame, 32
  deformation energy, **48**

graph
  recognition, 75
  segmentation, **65**
  trie, 104
  variation, 114

handwriting
  off-line, 8
  on-line, 9
helpline
  estimation, *see* preprocessing
Hidden Markov Models, **13**, 21
HWR, 7

ligature, 64
  pen-down, 64
  pen-up, 64

medial form, 110
modeling
  diacritic, **83**
  explicit, 11
  implicit, 11
  noise, **72**
  pen-up, **53**
  syntactic, **19**

Neural Networks, 20
  Self-Organizing Maps, **22**
  Time Delay Network, **21**
normalization
  ratio, **51**
  scale, *see* preprocessing

overtraining, 108

parameterization, 10
  arclength, **10**, 44
  by segmentation, 34
  polygonal approximation, **10**, 32
PCA, **15**, 34
pen-up
  attachment, **85**
  diacritic, 85
  recalculated, 85
preprocessing, **43**
  helpline estimation, **44**
  parameters, 79
  scale normalization, **45**
  slant correction, **45**
  smoothing, **43**

recognition
  graph, 75
relief function, **42**
reparameterization, 34

sampling, 9
SCR, 11
segment, 27
  virtual reference, **53**
segmentation, **27**
  generic, 31
  graph, 65, **65**
  script dependent, 28
  similar, **28**, 55, 71
slant correction, *see* preprocessing
smoothing, *see* preprocessing
spline
  thin plate, 33
SVM, **22**

template, 11, 48
  alignment, 66
  based, 17
  connectivity, 61
  database, 18
  segmented matching, **55**
thin plate spline, *see* spline
trie, 128
  dictionary, 104, 105

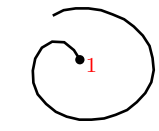
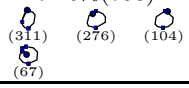
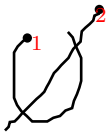
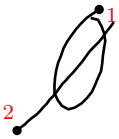
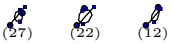
variation graph, 114

APPENDIX A

UNIPEN Train-R01/V07-1a Allograph Content

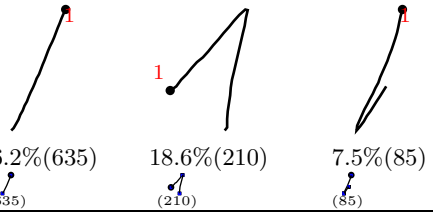
This appendix contains the complete tables of the 3 most common allographs, including the different segmentations as defined in the datasbase used in the experiments on the `UNIPEN\1a` dataset in Section 9.3. The dataset was divided into a writer-independent train and test with the tools suggested in [97].

A.1 Training Set Allograph Distribution

Class	$\|\mathbb{X}\|$	Allographs
0	1059	 71.6%(758)  (311) (276) (104) (67)  11.9%(126)  (62) (33) (31)  5.8%(61)  (27) (22) (12)

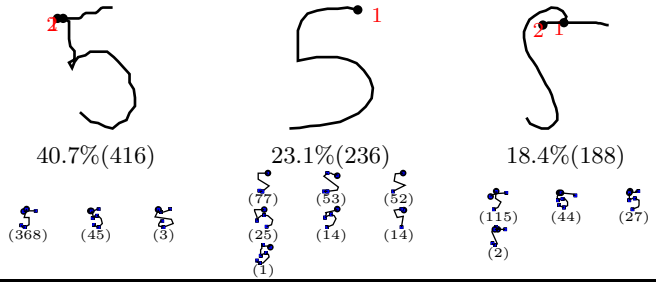
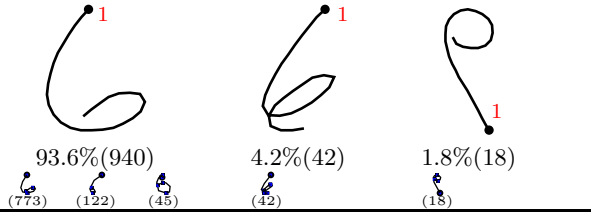
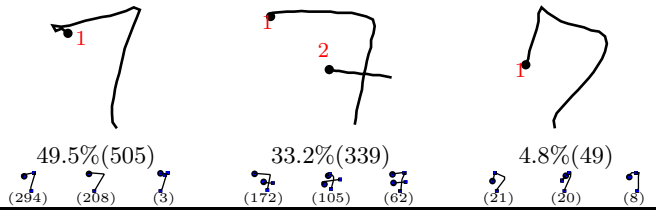
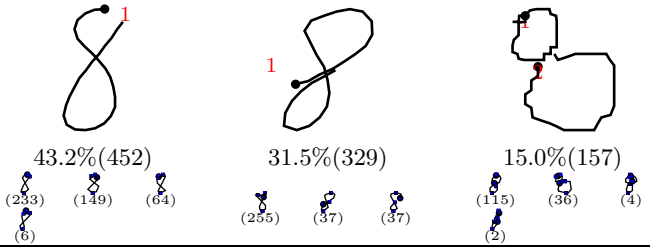
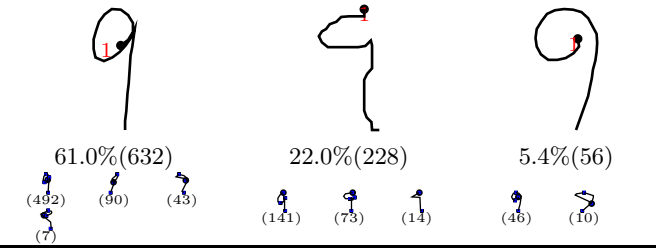
continued on next page

continued from previous page

Class	$\lvert\mathbb{X}\rvert$	Allographs
1	1130	 56.2%(635) 18.6%(210) 7.5%(85) (635) (210) (85)
2	1107	 63.3%(701) 25.8%(286) 5.5%(61) (361) (84) (77) (75) (74) (30) (240) (17) (12) (10) (7) (61)
3	1061	 64.6%(685) 14.4%(153) 6.8%(72) (303) (159) (149) (74) (98) (54) (1) (32) (20)
4	1052	 53.8%(566) 33.8%(356) 3.5%(37) (495) (40) (31) (169) (169) (17) (1) (30) (7)

continued on next page

continued from previous page

Class	$\lvert\mathbb{X}\rvert$	Allographs
5	1021	
6	1004	
7	1020	
8	1046	
9	1036	

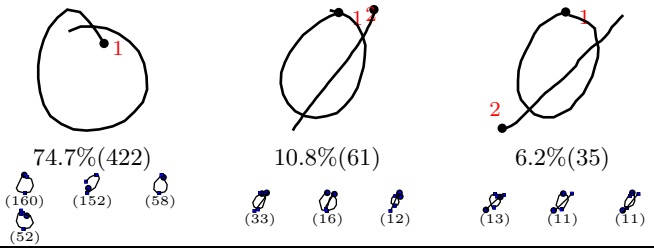
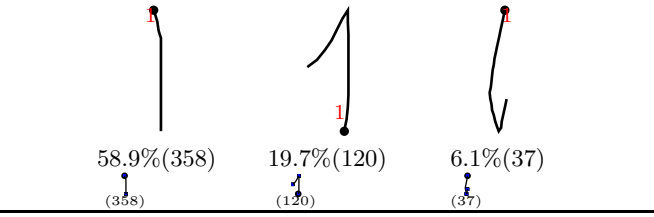
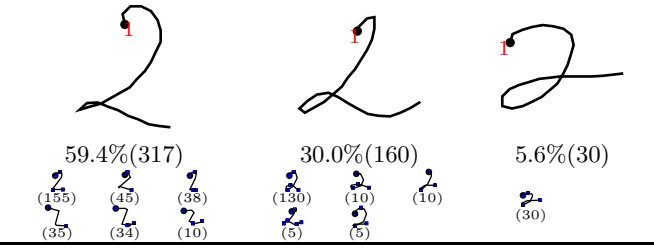
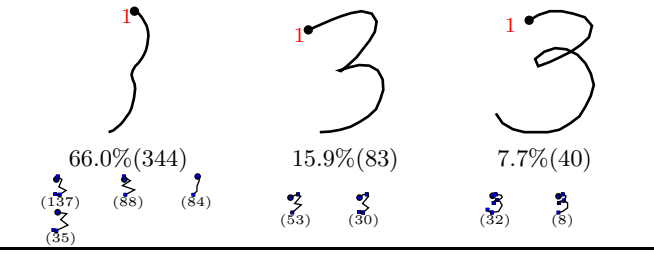
continued on next page

continued from previous page

Class	$\lvert\mathbb{X}\rvert$	Allographs

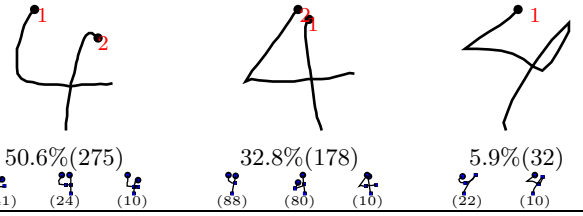
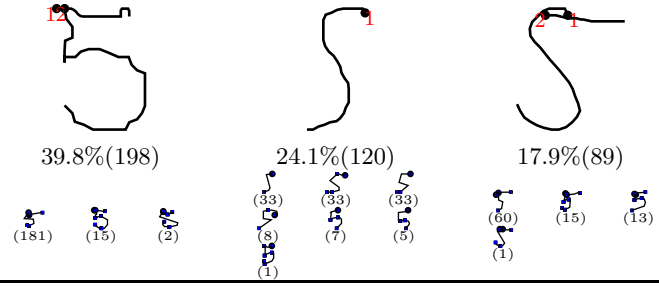
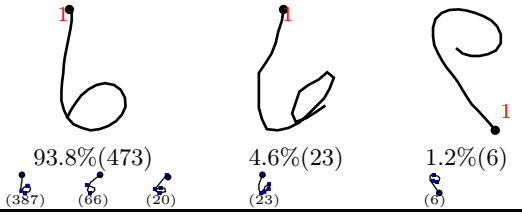
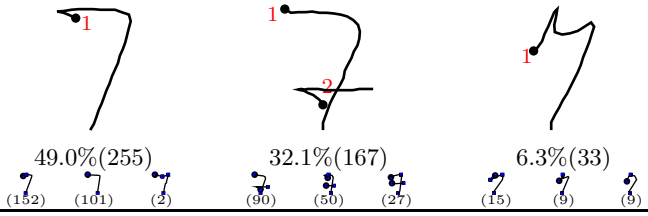
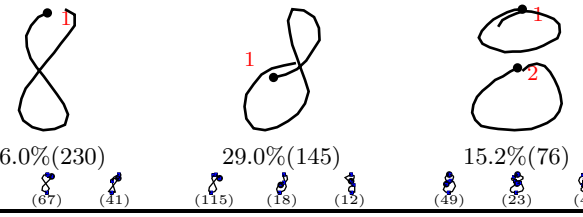
TABLE A.1: The three most common allographs with their respective segmentations.

A.2 Test Set Allograph Distribution

Class	$\lvert\mathbb{X}\rvert$	Allographs
0	565	
1	608	
2	534	
3	521	

continued on next page

continued from previous page

Class	$\lvert\mathbb{X}\rvert$	Allographs
4	543	
5	497	
6	504	
7	520	
8	500	

continued on next page

continued from previous page

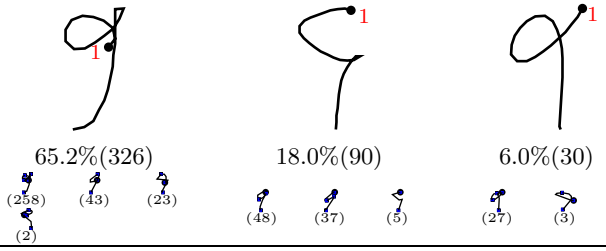
Class	$\lvert\mathbb{X}\rvert$	Allographs
9	500	

TABLE A.2: The three most common allographs of 0,2 in the test part of the Train-R01/07-1a data with their respective segmentations. The complete table is included in Section A.2.