

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

A THESIS PRESENTED TO THE
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN FULFILLMENT OF THE THESIS REQUIREMENT
FOR THE DEGREE OF
PHILOSOPHIAE DOCTOR IN ENGINEERING
Ph.D.

BY
ALESSANDRO L. KOERICH

LARGE VOCABULARY OFF-LINE HANDWRITTEN WORD RECOGNITION

MONTREAL, AUGUST 19, 2002

© Copyright reserved by Alessandro L. Koerich

THIS THESIS WAS EVALUATED
BY THE COMMITTEE COMPOSED BY:

Mr. Robert Sabourin, Thesis Supervisor
Département de Génie de la Production Automatisée, École de Technologie Supérieure

Mr. Ching Y. Suen, Thesis Co-Supervisor
Centre for Pattern Recognition and Machine Intelligence, Concordia University

Mr. Jacques-André Landry, President
Département de Génie de la Production Automatisée, École de Technologie Supérieure

Mr. Adam Krzyzak, External Examiner
Department of Computer Science, Concordia University

Mr. Mohamed Cheriet, Examiner
Département de Génie de la Production Automatisée, École de Technologie Supérieure

THIS THESIS WAS DEFENDED IN FRONT OF THE EXAMINATION
COMMITTEE AND THE PUBLIC
ON AUGUST 19, 2002
AT THE ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

RECONNAISSANCE HORS-LIGNE DE MOTS MANUSCRITS DANS UN LEXIQUE DE TRÈS GRANDE DIMENSION

Alessandro L. Koerich

Sommaire

Au cours des dernières années, des progrès considérables ont été accomplis dans le domaine de la reconnaissance de l'écriture manuscrite. Ainsi, il est intéressant de constater que la plupart des systèmes existants s'appuient sur l'utilisation d'un lexique pour effectuer la reconnaissance de mots. Or, dans la plupart des applications le lexique utilisé est de petite ou de moyenne dimension. Bien entendu, la possibilité de traiter efficacement un très grand vocabulaire permettrait d'élargir le champ des applications, mais cette extension du vocabulaire (de quelques dizaines à plus de 80 000 mots) a pour conséquence l'explosion de l'espace de recherche et bien souvent la dégradation des taux de reconnaissance.

Ainsi, le thème principal de cette thèse de doctorat est la reconnaissance de l'écriture manuscrite dans le cadre de l'utilisation de lexique de très grande dimension. Nous présentons tout d'abord, plusieurs stratégies pour améliorer en termes de vitesse de reconnaissance les performances d'un système de référence. L'objectif sera alors de permettre au système de traiter de très grands lexiques dans un temps raisonnable. Par la suite, nous améliorons les performances en termes de taux de reconnaissance. Pour ce faire, nous utiliserons une approche neuronale afin de vérifier les N meilleurs hypothèses de mots isolés par le système de référence. D'autre part, toutes les caractéristiques du système initial ont été conservées: système omni-scripteurs, écriture sans contraintes, et lexiques générés dynamiquement.

Les contributions majeures de cette thèse sont l'accélération d'un facteur 120 du temps de traitement et l'amélioration du taux de reconnaissance d'environ 10% par rapport au système de référence. Le gain en vitesse est obtenu grâce aux techniques suivantes: recherche dans un arbre lexical, réduction des multiples modèles de caractères, techniques de reconnaissance guidée par le lexique avec et sans contraintes, algorithme "level-building" guidé par le lexique, algorithme rapide à deux niveaux pour effectuer le décodage des séquences d'observations et utilisation d'une approche de reconnaissance distribuée. Par ailleurs, la précision du système est améliorée par le post-traitement des N meilleures hypothèses de mots à l'aide d'un module de vérification. Ce module est basé sur l'utilisation d'un réseau de neurones pour vérifier la présence de chacun des caractères segmentés par le système de base. La combinaison des résultats du système de référence et du module de vérification permet alors d'améliorer significativement les performances de reconnaissance. Enfin, une procédure de rejet est mise en place et permet d'atteindre un taux de reconnaissance d'environ 95% en ne rejetant que 30% des exemples.

LARGE VOCABULARY OFF-LINE HANDWRITTEN WORD RECOGNITION

Alessandro L. Koerich

Abstract

Considerable progress has been made in handwriting recognition technology over the last few years. Thus far, handwriting recognition systems have been limited to small-scale and very constrained applications where the number of different words that a system can recognize is the key point for its performance. The capability of dealing with large vocabularies, however, opens up many more applications. In order to translate the gains made by research into large and very-large vocabulary handwriting recognition, it is necessary to further improve the computational efficiency and the accuracy of the current recognition strategies and algorithms.

In this thesis we focus on efficient and accurate large vocabulary handwriting recognition. The main challenge is to speedup the recognition process and to improve the recognition accuracy. However, these two aspects are in mutual conflict. It is relatively easy to improve recognition speed while trading away some accuracy. But it is much harder to improve the recognition speed while preserving the accuracy.

First, several strategies have been investigated for improving the performance of a baseline recognition system in terms of recognition speed to deal with large and very-large vocabularies. Next, we improve the performance in terms of recognition accuracy while preserving all the original characteristics of the baseline recognition system: omniwriter, unconstrained handwriting, and dynamic lexicons.

The main contributions of this thesis are novel search strategies and a novel verification approach that allow us to achieve a 120 speedup and 10% accuracy improvement over a state-of-art baseline recognition system for a very-large vocabulary recognition task (80,000 words). The improvements in speed are obtained by the following techniques: lexical tree search, standard and constrained lexicon-driven level building algorithms, fast two-level decoding algorithm, and a distributed recognition scheme. The recognition accuracy is improved by post-processing the list of the candidate N -best-scoring word hypotheses generated by the baseline recognition system. The list also contains the segmentation of such word hypotheses into characters. A verification module based on a neural network classifier is used to generate a score for each segmented character and in the end, the scores from the baseline recognition system and the verification module are combined to optimize performance. A rejection mechanism is introduced over the combination of the baseline recognition system with the verification module to improve significantly the word recognition rate to about 95% while rejecting 30% of the word hypotheses.

RECONNAISSANCE HORS-LIGNE DE MOTS MANUSCRITS DANS UN LEXIQUE DE TRÈS GRANDE DIMENSION

Alessandro L. Koerich

Résumé

L'écriture reste l'un des modes de communication privilégiés par l'être humain. Elle est utilisée à la fois à des fins personnelles (lettres, notes, adresses postales, etc.) et dans un cadre professionnel (chèques bancaires, factures, formulaires d'impôts, etc.). Par ailleurs, les ordinateurs sont aujourd'hui omniprésents dans la vie quotidienne et chacun d'entre nous est amené à les utiliser. Ainsi, il est essentiel de démocratiser leur utilisation en facilitant les interactions entre l'homme et la machine. En effet, la majorité du traitement des informations étant actuellement réalisée électroniquement, il est indispensable de simplifier ce transfert d'informations entre l'utilisateur et l'ordinateur. L'écriture semble alors une solution intéressante, étant conviviale et n'exigeant aucune formation spécifique de l'utilisateur. D'autre part, en plus de l'aspect interface homme-machine, la reconnaissance d'écriture peut permettre le traitement automatique d'une importante quantité de documents manuscrits déjà en circulation. Qu'il s'agisse de chèques bancaires, de lettres, de déclarations d'impôts ou de tout autre type de documents manuscrits, la reconnaissance d'écriture offre des possibilités intéressantes d'améliorer la rentabilité des systèmes de traitements en limitant l'intervention humaine lors de la pénible tâche de transcription.

Une analyse approfondie du domaine de recherche nous indique que la majeure partie des travaux réalisés en reconnaissance d'écriture sont consacrés à la résolution de problèmes relativement simples telle que la reconnaissance de chiffres et de caractères isolés ou la reconnaissance de mots dans un petit lexique. Or, l'une des principales difficultés se situe au niveau du nombre de classes et de l'ambiguïté pouvant exister entre elles. En effet, plus le nombre de classes est grand, plus la quantité de données requises pour modéliser le problème augmente. D'autre part, plus l'on diminue le nombre de contraintes, plus la tâche de reconnaissance devient complexe. Ainsi, les meilleurs résultats rapportés dans la littérature ont généralement été obtenus en utilisant des lexiques de petite dimension et en se limitant à un faible nombre de scripteurs.

Malgré les nombreux progrès réalisés durant ces dernières années et l'augmentation de la puissance de calcul des ordinateurs, les performances des systèmes de reconnaissance d'écriture sont encore loin d'égaliser celles d'un expert humain. La reconnaissance de l'écriture manuscrite non-contrainte reste donc un défi d'actualité.

Une des principales difficultés de ce domaine est liée à la très grande variabilité existant entre les différents styles d'écriture. De plus, il existe une forte incertitude, non seulement en raison de la grande variété dans la forme des caractères, mais également à cause des éventuels recouvrements et des liaisons pouvant exister entre les caractères. Les lettres peuvent effectivement être soit isolées comme dans le cas de l'écriture bâton, soit regroupées par groupe de lettres ou encore former un mot composé de lettres entièrement connectées. Ainsi observés individuellement, les caractères sont souvent ambigus. Il est alors nécessaire d'intégrer des informations contextuelles pour pouvoir reconnaître la lettre. Or, bien que l'objectif final soit la reconnaissance du mot, il ne semble pas envisageable, dans le cadre des lexiques de grande dimension, de chercher à modéliser directement le mot complet. En effet, il faudrait pour cela disposer d'une quantité prohibitive de données d'apprentissage. Il est donc préférable de chercher à modéliser des caractères ou pseudo caractères qui seront utilisés pour effectuer la reconnaissance. Il faut donc nécessairement segmenter le mot ce qui peut s'avérer relativement délicat notamment dans le cas de l'écriture cursive.

L'objectif principal de cette thèse est de proposer un système omniscriteur de reconnaissance hors-ligne d'écriture manuscrite non-contrainte (bâton, cursif et mixte) capable de traiter un lexique de très grande dimension (80 000 mots) à l'aide d'ordinateurs standards tel que des PC.

Le principal défi se situe au niveau de la dimension du lexique, qui complique significativement la tâche de reconnaissance non seulement au niveau de la complexité de calcul, mais aussi de l'augmentation de l'ambiguïté et de la variabilité. Ainsi, il est important de prendre en compte à la fois la vitesse de traitement et la précision des résultats. Bien que ces deux aspects peuvent sembler antagonistes, nous avons démontré qu'il est possible de réduire considérablement le temps de traitement tout en conservant la précision du système de référence. Il est alors possible de traiter des lexiques de très grande dimension.

Les hypothèses de départ de cette thèse de doctorat étaient les suivantes :

- Il est possible d'incorporer au système de base des stratégies de recherche dans un lexique qui permettent d'accélérer le processus de reconnaissance sans affecter sa précision.
- Le post-traitement des N meilleures hypothèses de mots isolés par le système peut permettre d'améliorer la précision et la fiabilité de la reconnaissance.

L'idée directrice du premier axe de recherche, est de développer des stratégies de recherche rapide en éliminant les étapes de calcul répétées. Pour ce faire, nous utilisons les particularités de l'architecture des modèles de Markov cachés (MMC)

qui modélisent les caractères. Nous tiendrons aussi compte des spécificités des étapes d'extraction des primitives de segmentation des mots en caractères et de l'approche de reconnaissance guidée par le lexique. Ainsi, nous avons développé une stratégie de décodification rapide des MMC à deux niveaux. Cette technique découpe le calcul des probabilités associées aux mots en un niveau "états" et un niveau "caractères". Ceci permet la réutilisation des probabilités des caractères pour décoder tous les mots dans le vocabulaire et éviter ainsi le calcul répétitif des séquences d'état. Par ailleurs, nous avons en plus utilisé le concept de systèmes répartis afin de diviser la tâche parmi plusieurs processeurs.

Concernant la deuxième hypothèse de recherche, l'idée est de développer une stratégie de post-traitement de manière à compenser les faiblesses du système de référence concernant la capacité de discrimination des hypothèses très semblables de mots isolés. Nous avons donc développé une stratégie de vérification qui s'appuie uniquement sur les N meilleures hypothèses de mots. Nous utilisons alors une approche neuronale pour vérifier la présence de chacun des caractères segmentés par le système de base. La combinaison des résultats du système de référence et du module de vérification permet alors d'améliorer significativement les performances de reconnaissance. L'utilisation d'un réseau de neurones permet notamment de surmonter une partie des limitations des modèles cachés de Markov en réduisant l'ambiguïté entre des formes de caractères semblables.

Les contributions originales de ce travail portent sur :

- La conception d'un système utilisant un lexique de très grande dimension. En effet, les plus grands lexiques rencontrés dans la littérature comprennent environ 40 000 mots.
- L'intégration d'une technique de recherche en arbre gérant efficacement les multiples modèles de caractères.
- L'utilisation d'un algorithme "level-building" pour réduire la complexité de calcul du processus de reconnaissance en choisissant et en poursuivant à chaque niveau uniquement le modèle de caractère (majuscule ou minuscule) le plus probable.
- L'intégration dans l'algorithme "level-building" de trois contraintes et l'utilisation de méthodes statistiques pour déterminer les paramètres de contrôle qui maximisent la précision et la vitesse de reconnaissance.
- L'adaptation des contraintes à chaque type de caractères afin d'améliorer les performances de l'algorithme "level-building".

- Un nouvel algorithme rapide à deux niveaux pour décoder les séquences d'observations du modèle de Markov caché et accélérer ainsi le processus de reconnaissance tout en préservant la précision.
- Les concepts de modularité et de réutilisation des modèles de caractères décodés dans une approche de reconnaissance guidée par le lexique.
- L'utilisation du concept de calculs distribués pour développer un système de reconnaissance réparti, où la tâche est décomposée en plusieurs sous tâches.
- Le développement d'un paradigme de post-traitement pour la vérification des N meilleures hypothèses de mots générés par le système de reconnaissance d'écriture non-contrainte.
- L'utilisation du système de reconnaissance pour effectuer la segmentation des mots en caractères et l'utilisation d'une approche neuronale pour vérifier la segmentation en estimant la probabilité a posteriori associée à chacun des différents caractères isolés.
- La combinaison d'un système de reconnaissance de mots basé sur les modèles de Markov cachés avec un module de vérification utilisant des réseaux de neurones afin d'améliorer la fiabilité du système global.

Enfin, nous avons développé un système de reconnaissance automatique de l'écriture manuscrite non-contrainte qui permet d'identifier un mot parmi un vaste vocabulaire avec un taux de réussite d'environ 78%. ce qui correspond à une amélioration d'approximativement 10% par rapport au système de base. D'autre part, nous avons amélioré d'un facteur 120 le temps de traitement des données en phase de test. Ainsi, la reconnaissance d'un mot prend environ 60 secondes sur un ordinateur conventionnel (Sun Ultra1 à 173 MHz) et moins de cinq secondes sur un ordinateur parallèle à 10 processeurs du même type.

ACKNOWLEDGEMENTS

It is an impossible task to not forget the many people who have contributed, directly or indirectly, to this work. As I could not simply omit such a page, I would like then to express all my thanks.

First of all I would like to express my gratitude to Prof. Robert Sabourin for supervising me during this thesis. I am also indebted to Prof. Ching Y. Suen who has also supervised my work.

I would like to thank everyone else in the *Laboratoire de Vision, d'Imagerie et d'Intelligence Artificielle*. The laboratory has been an ideal environment, both socially and technically, in which to conduct research.

Special thanks must go to Abdenaim El-Yacoubi and Frederic Grandidier for all the help concerning the baseline recognition system, Yann Leydier for helping with the verification module, and particularly to Alceu de Souza Britto, Alessandro Zimmer, Marisa Morita and Luiz E. Soares for their invaluable support and friendship.

I would also like to thank everyone in the *Centre for Pattern Recognition and Machine Intelligence* (CENPARMI) specially Christine Nadal for her invaluable support and friendship.

The Brazilian National Council for Scientific and Technological Development (CNPq) is to be thanked for providing the financial support necessary for me to carry out this work. I would also like to thank the Ministry of Education of Québec and the Decanat des Études Supérieures e de la Recherche de l'École de Technologie Supérieure for the partial financial support.

The Service de Recherche Technique de la Poste (SRTP), is to be thanked for providing the baseline recognition system and the database used in this work.

Finally, I would like to dedicate this thesis to Soraya and Karl Michel for the endless support during the last four years.

CONTENTS

ABSTRACT	i
SOMMAIRE	ii
RÉSUMÉ	iii
ACKNOWLEDGEMENTS	vii
CONTENTS	ix
LIST OF TABLES	xiv
LIST OF FIGURES	xx
LIST OF ABBREVIATIONS	xxx
LIST OF NOTATIONS	xxxi
INTRODUCTION	1
CHAPTER 1 STATE OF THE ART IN HANDWRITING RECOGNITION	17
1.1 Recognition Strategies	17
1.2 The Role of Language Model	23
1.3 Large Vocabulary Handwriting Recognition	24
1.3.1 Lexicon Reduction	25
1.3.1.1 Other Sources of Knowledge	26
1.3.1.2 Word Length	28
1.3.1.3 Word Shape	29
1.3.1.4 Other Approaches	31
1.3.2 Search Space Organization	34
1.3.2.1 Flat-Lexicon	34

1.3.2.2	Lexical Tree	35
1.3.2.3	Other Approaches	37
1.3.3	Search Techniques	37
1.3.3.1	Dynamic-Programming Matching	39
1.3.3.2	Beam Search	41
1.3.3.3	A*	43
1.3.3.4	Multi-Pass	43
1.4	Verification and Post-Processing in Handwriting Recognition . .	44
1.4.1	Isolated Handwritten Character Recognition	44
1.4.1.1	Neural Network Classifiers	45
1.4.1.2	Statistical Classifiers	47
1.4.1.3	Other Recognition Strategies	48
1.4.2	Verification in Handwriting Recognition	51
1.5	Summary	54
CHAPTER 2 PROBLEM STATEMENT		56
2.1	Large Vocabulary Problems	57
2.1.1	The Complexity of Handwriting Recognition	59
2.1.2	Discussion on the Current Methods and Strategies for Large Vo- cabulary Handwriting Recognition	62
2.2	Problems of Recognition Accuracy	64
2.2.1	Error Analysis	65
2.3	Other Related Problems	67
2.4	Large Vocabulary Handwriting Recognition Applications	68
2.5	A More Precise Overview of This Thesis	69
CHAPTER 3 BASELINE RECOGNITION SYSTEM		75
3.1	System Overview	75
3.2	Pre-Processing	76

3.3	Segmentation of Words into Characters	78
3.4	Feature Extraction	79
3.5	Character and Word Models	81
3.6	Training of Character Models	83
3.7	Recognition of Unconstrained Handwritten Words	84
3.7.1	The Viterbi Algorithm	85
3.7.2	Computational Complexity and Storage Requirements	89
3.8	Performance of the Baseline Recognition System (BLN)	91
3.9	Baseline System Summary	92
CHAPTER 4 THESIS CONTRIBUTIONS		93
4.1	Speed Issues in Handwriting Recognition	94
4.1.1	Tree-Structured Lexicon	96
4.1.2	Multiple Character Class Models	98
4.1.3	Best Model Selection	101
4.1.4	A Review of the Handwriting Recognition Problem	102
4.1.5	Lexicon-Driven Level Building Algorithm (LDLBA)	104
4.1.5.1	Computational Complexity and Storage Requirements of the LDLBA	110
4.1.5.2	Summary of the LDLBA	112
4.1.6	Time and Length Constraints	114
4.1.6.1	Time Constraint	115
4.1.6.2	Length Constraint	116
4.1.6.3	Contextual Time and Length Constraints	118
4.1.6.4	Computational Complexity and Storage Requirements for the CLBA and the CDCLBA	121
4.1.6.5	Summary of the the CLBA and the CDCLBA	122
4.1.7	Fast Two-Level Decoding Strategy	123
4.1.7.1	Principles of the Fast Two-Level Decoding Strategy	123

4.1.7.2	First Level: Decoding of Character Models	126
4.1.7.3	Second Level: Decoding of Words	131
4.1.7.4	Computational Complexity and Storage Requirements of the Fast Two-Level Decoding Algorithm	133
4.1.7.5	Summary of the Fast Two-Level Decoding Algorithm	136
4.1.8	Extension: A Distributed Recognition Scheme	137
4.1.8.1	Exploiting Concurrency	137
4.1.8.2	Task Partitioning	138
4.1.8.3	Combination of the Results	140
4.1.8.4	Computational Complexity and Storage Requirements of the Distributed Recognition Scheme	140
4.1.9	Summary of the Speed Improvements	141
4.2	Verification of Handwritten Words	144
4.2.1	Characteristics of the Word Recognizer	146
4.2.2	Formalization of the Verification Problem	148
4.2.3	Architecture of the Verification Module	151
4.2.4	Feature Extraction	153
4.2.4.1	Profiles	155
4.2.4.2	Projection Histograms	156
4.2.4.3	Contour-Directional Histogram	157
4.2.4.4	Selection of Characteristics	157
4.2.5	NN Classifier	159
4.2.5.1	Network Architecture	160
4.2.5.2	Frequency Balancing	163
4.2.5.3	Stroke Warping	165
4.2.5.4	Training the Neural Network Classifier	167
4.2.5.5	Compensating for Varying <i>a priori</i> Class Probabilities	170
4.2.6	Combination of the Character Scores	171

4.2.6.1	Problems of Character Omission and Undersegmentation	171
4.2.6.2	Duration Modeling	172
4.3	Combination of the Word Classifiers Decision	174
4.3.1	Outline of the Verification Scheme	179
4.3.2	Summary of the Verification of Handwritten Words	180
4.4	Rejection Mechanism	181
4.4.1	Principles of the Rejection Mechanism	182
4.4.2	Summary of the Rejection Mechanism	185
CHAPTER 5	PERFORMANCE ANALYSIS	186
5.1	Measuring Performance	187
5.1.1	Recognition Accuracy	188
5.1.2	Recognition Time	188
5.2	Databases	189
5.2.1	NIST Database	190
5.2.2	SRTP Database	190
5.2.2.1	Construction of an Isolated Character Database from Handwritten Words	194
5.3	Improving Recognition Speed	200
5.3.1	Testing Conditions	200
5.3.2	Recalling the Baseline Recognition System Performance (BLN)	201
5.3.3	Lexicon-Driven Level Building Algorithm (LDLBA)	202
5.3.4	Time and Length Constraints (CLBA)	204
5.3.5	Class Dependent Constraints (CDCLBA)	207
5.3.6	Fast Two-Level HMM Decoding (FS)	208
5.3.7	Experiments with a Very-Large Vocabulary	211
5.3.8	Distributed Recognition Scheme (DS)	212
5.3.9	Summary of Speed Improvements	216
5.4	Improving Recognition Accuracy	218

5.4.1	Testing Conditions	220
5.4.2	Reevaluation of the Word Recognition Performance	222
5.4.3	Recognition of Isolated Handwritten Characters	223
5.4.3.1	k Nearest Neighbor Classifier (k -NN)	224
5.4.3.2	k -Nearest Prototype Classifier (k -NP)	226
5.4.3.3	Neural Network Classifier (NN)	231
5.4.3.4	Comparison of the Classifier Results	232
5.4.4	Recognition of Handwritten Words by NN Classifier	233
5.4.5	Verification of Unconstrained Handwritten Words	238
5.4.6	Error Analysis	239
5.4.7	Computation Breakup	244
5.4.8	Summary of Accuracy Improvements	246
5.5	Rejection of Word Hypotheses	247
	CONCLUSION	255
	APPENDICES	
1	Determination of Control Factors	261
2	Stroke Warping Algorithm	272
3	Hidden Markov Models	275
	BIBLIOGRAPHY	293

LIST OF TABLES

Table I	Recent results on off-line handwritten word recognition . . .	4
Table II	Comparison of lexicon reduction approaches based on word shape for off-line handwriting recognition. Results are for cursive (lowercase) and handprinted (uppercase) words respectively	32
Table III	Comparison of lexicon reduction approaches based on word shape for on-line recognition of cursive words	32
Table IV	Other pruning and lexicon reduction strategies for unconstrained off-line handwritten word recognition	34
Table V	Type of classifiers and features used in handwritten character recognition	49
Table VI	Summary of results in isolated handwritten character recognition	50
Table VII	A summary of the approximate computational complexity and storage requirements for the baseline recognition system based on the Viterbi decoding and a flat lexicon	90
Table VIII	Word recognition rate and recognition time for the baseline recognition system (BLN)	91
Table IX	Average number of characters for several different sizes of lexicons of French city names represented as a flat structure and as a tree structure	98
Table X	A summary of the approximate computational complexity and storage requirements for the LDLBA	112
Table XI	An example of the search limits for the CLBA and CLDLBA	

	for the word “St_Malo” and a 12-observation sequence . . .	118
Table XII	Number of segments resulting from the loose segmentation of words into characters or pseudo-characters for each character class [32]	120
Table XIII	A summary of the approximate computational complexity and storage requirements for the CLBA and the CDCLBA . . .	122
Table XIV	Computational complexity and storage requirements for the fast two-level decoding algorithm considering a flat structure and a tree-structured lexicon	135
Table XV	Computational complexity and storage requirements for the distributed recognition scheme using the fast two-level decoding algorithm and considering a flat structure and a tree-structured lexicon	141
Table XVI	Computational complexity and storage requirements for all strategies presented in this section	141
Table XVII	Computational complexity and storage requirements for all strategies presented in this section	142
Table XVIII	Several combination of features to determine the best feature vector for isolated handwritten character recognition	158
Table XIX	Character recognition rates achieved by different combinations of features on the NIST database, considering 26 metaclasses	159
Table XX	Results for isolated character recognition on NIST database considering uppercase and lowercase characters separately, considering uppercase and lowercase as a single class (Mixed) and considering the combination of two specialized classifiers . .	163
Table XXI	Training dataset of NIST database (184,033 uppercase + 155,215 lowercase characters)	191

Table XXII	Test dataset (TD1 dataset) of NIST database (11,941 uppercase + 12,000 lowercase characters)	192
Table XXIII	Validation dataset of NIST database (12,092 uppercase + 11,578 lowercase characters)	193
Table XXIV	The number of samples (city name images) in each dataset of the SRTP database	193
Table XXV	Training dataset of the SRTP database (115,088 characters from 12,022 words)	196
Table XXVI	Validation dataset of the SRTP database (36,170 characters from 3,475 words)	197
Table XXVII	Test dataset of the SRTP database (46,700 characters from 4,674 words)	198
Table XXVIII	Character recognition rates for isolated characters of SRTP database by using a NN classifier trained on the training dataset of NIST database	199
Table XXIX	Cleaned training dataset of the SRTP database (43,365 uppercase + 41,446 lowercase characters = 84,811 characters from 8,043 words)	199
Table XXX	Word recognition rate and recognition time for the baseline recognition system (BLN) on the SRTP test dataset	202
Table XXXI	Word recognition rate and recognition time of the system based on the level building algorithm and a lexical tree (LDLBA) and speedup over the Viterbi flat lexicon of baseline recognition system (BLN)	203
Table XXXII	Difference in the word recognition rates between the system based on the level building algorithm and a lexical tree (LDLBA) and the baseline recognition system (BLN)	203

Table XXXIII	Values of the constraints of the CLBA determined by the statistical experimental design technique	205
Table XXXIV	Word Recognition rate and recognition time for the system based on the constrained level building algorithm and a lexical tree (CLBA)	206
Table XXXV	Difference in the word recognition rates between the system based on the constrained level building algorithm and the baseline recognition system (BLN)	207
Table XXXVI	Individual influence of the control factors s^* and Lv^* on the recognition rate ($TOP\ 1$) and on the recognition time of the LDLBA	207
Table XXXVII	Word recognition rate and recognition time using character class dependent constraints incorporated to the level building algorithm	208
Table XXXVIII	Difference in the word recognition rates between the system based on the class dependent constrained level building algorithm and the baseline recognition system (BLN)	209
Table XXXIX	Word recognition rate and recognition time for the system based on the fast two-level HMM decoding and a flat lexicon (FSFlat) 36k1 task	210
Table XL	Word recognition rate and recognition time for the system based on the fast two-level HMM decoding and a lexical tree (FSTree) 36k1 task	210
Table XLI	Word recognition rate and recognition time for the system based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task on a Sun Ultra1	211
Table XLII	Figure of performance for the distributed recognition scheme (DS-FSTree) for a different number of processors	215

Table XLIII	Word recognition rate and recognition time for the system based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task on an Athlon 1.1GHz	223
Table XLIV	Results for isolated character recognition on the NIST database using different number of neighbors (k) for a k -NN classifier	224
Table XLV	Recognition time for isolated character recognition on the NIST database using different number of neighbors (k) for a k -NN classifier	225
Table XLVI	Results for isolated character recognition on the SRTP database using different number of neighbors (k) for a k -NN classifier	225
Table XLVII	Recognition time for isolated character recognition on the SRTP database using different number of neighbors (k) for a k -NN classifier	226
Table XLVIII	Character recognition rates on the NIST database using different prototype configurations for a 1-NP classifier	230
Table XLIX	Character recognition times on the NIST database for different prototype configurations for a 1-NP classifier	230
Table L	Character recognition rate on the SRTP database using different number of neighbors (k) for a k -NP classifier	231
Table LI	Character recognition time on the SRTP database using different number of neighbors (k) for a k -NP classifier	231
Table LII	Character recognition rates and character recognition time on the NIST database for the MLP classifier	232
Table LIII	Character recognition rate and character recognition time on the SRTP database for the MLP classifier	233
Table LIV	Word recognition rates using the NN classifier alone for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words	236

Table LV	Word recognition rates using the the MLP with character duration modeling for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words	237
Table LVI	Word recognition rates from the baseline recognition system alone, the MLP classifier alone and the combination of both classifiers by two different rules for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words	240
Table LVII	Computation breakup for the verification of handwritten words for a <i>TOP</i> 10 word hypothesis list and an 80,000-word vocabulary measured on an Athlon 1.1GHz	245
Table LVIII	Computation breakup for the recognition of handwritten words for a <i>TOP</i> 10 word hypothesis list and an 80,000-word vocabulary on an Athlon 1.1GHz	245
Table LIX	The level of the control factors.	263
Table LX	The full factorial 3^3 array for control factors with one replication.	264
Table LXI	Analysis of Variance for Recognition Rate (y_{RR}).	266
Table LXII	Analysis of Variance for Recognition Time (y_{RS}).	267
Table LXIII	Coefficients of fitted regression models for recognition rate (y_{RR}).	268
Table LXIV	Coefficients of fitted regression models for recognition time (y_{RS}).	269
Table LXV	Combination of factor levels that maximize the desirability function.	271
Table LXVI	Optimal responses for the optimum setup of the control factors.	271

LIST OF FIGURES

Figure 1	The complexity and accuracy of the sub-problems in handwriting recognition	5
Figure 2	An overview of the basic components of a handwriting recognition system	7
Figure 3	The word "PARIS" written in different styles by different writers . . .	9
Figure 4	Examples of easy cases to segment words into characters where the words are written by well-separated characters	11
Figure 5	Some examples of words which are very difficult to be segmented into characters	12
Figure 6	A trellis structure that illustrates the problem of finding the best matching between a sequence of observations O (or sequence of primitives S) and a reference pattern $R_v = (c_1c_2 \dots c_L)$	20
Figure 7	A summary of the strategies for large vocabulary handwriting recognition	25
Figure 8	Number of characters for different lexicon sizes generated from an 85.1k entry lexicon and organized as a flat structure and as a tree structure	36
Figure 9	Reduction of the recognition rate with an increasing number of words in the lexicon [87]	58
Figure 10	Increasing of the recognition time with an increasing number of words in the lexicon [87]	58
Figure 11	Example of similar words present in a lexicon of French city names	59
Figure 12	Possible representations of a word: (a) assuming only one writing	

	style: entirely handprinted or cursive: (b) assuming all possible combinations of handprinted and cursive characters: mixed	61
Figure 13	An example of a page of handwritten text taken from the LOB database [153]	70
Figure 14	An example of a postal envelope taken from the testing set of the SRTP database	71
Figure 15	Some examples of handwritten notes and letters written by different persons	72
Figure 16	An example of a fax page with handwritten text, imprints and printed text	73
Figure 17	Two examples of business documents where the fields are filled in with handwritten text	73
Figure 18	An overview of the main modules of the baseline SRTP recognition system	77
Figure 19	An example of the pre-processing steps applied to cursive words: (a) original word images, (b) baseline slant normalization, (c) character skew correction, (d) lowercase character area normalization, (e) final images after smoothing	78
Figure 20	Two examples of the loose segmentation of handwritten words into characters or pseudo-characters	79
Figure 21	An example of observation sequences extracted from two handwritten words: (a) sequence of global features; (b) sequence of bidimensional transition histogram features; (c) sequence of segmentation features	81
Figure 22	The left-right character HMM with 10 states and 12 transitions where five of them are null (dotted lines) [35]	83
Figure 23	The inter-word space model with 2 states and 2 transitions where	

	one is null (dotted line) [35]	83
Figure 24	The global recognition model formed by the concatenation of uppercase and lowercase character models where “B” and “E” denote the beginning and the end of a word respectively	86
Figure 25	An overview of the modules of the baseline recognition system that are used through this thesis (light gray) and the modules introduced by the author (dark gray)	95
Figure 26	Two 4-character words (<i>BILY</i> and <i>BIME</i>) represented as: (a) a flat structure. (b) a tree structure where the prefix <i>BI</i> is shared by both words	97
Figure 27	Two 4-character words (<i>BILY</i> and <i>BIME</i>) represented as: (a) a flat structure where each character class (e.g. “M”) has two models (e.g. an uppercase λ_M and a lowercase λ_m model) and all combinations of models are allowed. (b) a tree representation of the same two words	100
Figure 28	A simplified representation of the lexical tree of Figure 27b by using the maximum approximation and selecting only the one character model at each tree level	101
Figure 29	A simplified overview of the unified structure. The word <i>ABBECOURT</i> is represented in the lexical tree by a sequence of linked characters (nodes) that may be shared by other words. The characters are represented by one or more HMMs (e.g. “A” and “a”). The LBA finds at each level which is the best model (“A” or “a”) for each t as well as the overall combination of models that best matches the entire sequence of observation (e.g. <i>Abbecourt</i>) . . .	111
Figure 30	The inclusion of several models for a unique character class in parallel: (a) with level reduction only the model with highest likelihood is expanded. (b) without level reduction all models are expanded	113
Figure 31	Levels of the LDLBA incorporating the time constraints $s(l)$ and	

- $e(l)$ that limit the number of observations aligned at each level, and the length constraint L_v that limits the number of levels of the LDLBA 117
- Figure 32 Reduction in the search space for a 7-character word (ST_MALO) and a 12-observation sequence. The search space delimited by the CLBA is shown in gray, while the line surrounding the search space represents the region delimited by the CDCLBA 121
- Figure 33 (a) A small lexicon with 5 words. (b) Organization of the lexicon as tree structure with the computation order for the time-asynchronous search. (c) Computation order for each word for the conventional time-synchronous Viterbi search 124
- Figure 34 A 3-character word HMM formed by the concatenation of three 2-state character HMMs, and the corresponding lattice structure for a 10-observation sequence. The gray arrows indicate the possible state transitions, the black arrows indicate the best path, and the numbers indicate the computation order 126
- Figure 35 An HMM topology with a unique initial state and a unique final state 127
- Figure 36 (a) Computation of the probabilities for a single 3-state character HMM for each entry point s . (b) The resulting forward probability array $\chi(s, e)$ that models single characters for a specific observation sequence, (c) The resulting best state sequence array $\epsilon(s, e)$, (d) Example of backtracking for two exit points $(s, e) = (3, 6)$, and $(s, e) = (3, 7)$. The resulting MAP state sequence is stored in the array $\epsilon(s, e)$ 129
- Figure 37 A character model represented as a block with an associated transfer function that maps input probabilities to output probabilities . . . 130
- Figure 38 (a) Computation of the probabilities for a word formed by the concatenation of characters represented as building blocks, (b) An example of backtracking for a 3-character word and a 9-observation

	sequence	134
Figure 39	An overview of the distributed recognition scheme where several similar classifiers execute a relatively less complex recognition task for a partitioned lexicon	139
Figure 40	Some examples of 10-best word hypothesis lists generated by the baseline recognition system	147
Figure 41	An overview of the main components and inputs of the verification module	150
Figure 42	The relation between the recognition rate and the number of N -best word hypotheses	151
Figure 43	An example of the four projection profiles for the letter “a”: top, bottom, left and right hand side profiles	155
Figure 44	An example of the two projection histograms for the letter “a”: vertical and horizontal projection histograms	156
Figure 45	(a) An example of the contour extracted from the letter “a”, (b) the contour split in 6 parts corresponding to the 6 zones	157
Figure 46	The eight directions used in the directional-contour histogram	158
Figure 47	Resulting 108-dimensional feature vector for isolated handwritten character recognition	159
Figure 48	Number of errors in the correct case recognition at the output of the word recognizer for the validation dataset	162
Figure 49	Architectural graph of the multilayer perceptron with 108 inputs, 90 neurons in the hidden layer and 26 outputs	164
Figure 50	A few random stroke warpings of the same original “a” character (inside the gray box)	166

- Figure 51 A few random stroke warpings of the same original “W” character (inside the gray box) 166
- Figure 52 Training and validation curves for the NIST database. The dashed line indicates where the training procedure was stopped 169
- Figure 53 Training and validation curves for the SRTP database. The dashed line indicates where the training procedure was stopped 169
- Figure 54 An example of undersegmentation where the characters “or” and “te” are linked within the word “Le Portet”. and during the recognition they are recognized as “O” and “R” respectively. The character “E” is omitted 172
- Figure 55 Some examples of character omission where the character “t” is omitted in the prefix “St” for the city names “ST ROMAN DE MARGARDE”, “St Georges S Erve”, and “St Nicolas les Arras” 173
- Figure 56 The variation of the word recognition rate of the combination of the HMM-MLP classifiers as a function of the combination weights for the SRTP validation dataset and 3 different sizes of dynamically generated lexicons: 10, 10,000 and 80,000 words 177
- Figure 57 An overview of the integration of the baseline recognition system with the verification module to form a hybrid HMM/NN handwriting recognition system 178
- Figure 58 (a) *N*-best list of word hypotheses generated by the baseline recognition system, (b) input handwritten word, (c) loose segmentation of the word into pseudo-characters, (d) final segmentation of the word into characters, scores assigned by the MLP classifier to each segment, and the composite score obtained by summing the scores of each segment, (e) *N*-best list of word hypotheses scored by the MLP classifier, (f) rescored *N*-best word hypotheses 179
- Figure 59 An overview of the rejection mechanism and the final recognition result either accepting or rejecting an input image 182

Figure 60	Word recognition rate (TOP 1) as a function of rejection rate for the baseline recognition system alone and the baseline + verification on the SRTP validation dataset and 10k-word dynamically generated lexicon	184
Figure 61	An overview of the bootstrapping process to automatically generate a character database from unconstrained handwritten words from the SRTP database	195
Figure 62	Some examples of bad characters shapes that were eliminated from the database during the cleaning process	200
Figure 63	Comparison of average processing times achieved by the baseline SRTP recognition system implemented with a conventional Viterbi search (VT flat) and the implementation based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task (85.1k) over a limited test set of 1,000 words	213
Figure 64	Average processing time for the distributed recognition scheme according to the number of threads (processors) for a 30k-entry dynamically generated lexicon	214
Figure 65	Word recognition rate <i>versus</i> speedup of various systems for a 10-word vocabulary	217
Figure 66	Word recognition rate <i>versus</i> speedup of various systems for a 1k-word vocabulary	217
Figure 67	Word recognition rate <i>versus</i> speedup of various systems for a 30k-word vocabulary	218
Figure 68	Comparison of the search strategies for a 30,000 -word vocabulary recognition task: (a) recognition accuracy for the <i>TOP</i> 1 choice, (b) recognition time	219
Figure 69	Character recognition rates of the <i>k</i> -NP on the NIST validation dataset for different configurations of prototypes and number of	

	neighbors	228
Figure 70	Number and configuration of class prototypes for the 1-NP classifier on the NIST validation dataset	229
Figure 71	Comparison of character recognition rate and character recognition time on the NIST test dataset using k -NN, k -NP and MLP classifiers	234
Figure 72	Comparison of character recognition rate and character recognition time on the SRTP test dataset using k -NN, k -NP and MLP classifiers	235
Figure 73	Word recognition rates (TOP 1) for the MLP classifier and for the baseline recognition system based on the FSTree search strategy for different lexicon sizes (log scale)	237
Figure 74	Word recognition rates (TOP 1) for the MLP classifier with and without duration model and for the baseline recognition system based on the FSTree search strategy for different lexicon sizes (log scale)	238
Figure 75	Word recognition rates (TOP 1) of the recognition-verification system compared with the baseline recognition system and MLP classifier (log scale)	241
Figure 76	An example of a recognition error corrected by the verification module. The truth word hypothesis (<i>gourette</i>) was shifted up from TOP 4 position to the TOP 1 position after the verification	242
Figure 77	An example of a recognition error not corrected by the verification module. The truth word hypothesis (<i>livarot</i>) was shifted up from TOP 5 position to the TOP 2 position after the verification	243
Figure 78	An example of a recognition error caused by the verification module. The truth word hypothesis (<i>SOUES</i>) was shifted down from TOP 1 position to the TOP 2 position after the verification	244

Figure 79	The rejection criterion applied at three different levels: output scores of the baseline recognition system alone, output scores of the verification module alone and output scores of the the combination of the baseline recognition system with the verification	249
Figure 80	Word recognition rate (<i>TOP 1</i>) as a function of rejection rate for the baseline recognition system alone, the verification module alone and the baseline+verification scheme on the SRTP test dataset for an 80k-word lexicon	250
Figure 81	Word recognition rate (<i>TOP 1</i>) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for a 10-word lexicon	251
Figure 82	Word recognition rate (<i>TOP 1</i>) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for a 10k-word lexicon	252
Figure 83	Word recognition rate (<i>TOP 1</i>) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for an 80k-word lexicon	253
Figure 84	Main effects of the control factors on the recognition rate for a 100-word vocabulary.	265
Figure 85	Main effects of the control factors on the recognition speed for a 100-word vocabulary.	266
Figure 86	Estimated response surface for $Lv^* = 0.0$	270
Figure 87	The character image resulting from each step of the stroke warping algorithm: (a) original character image, (b) resized character image, (c) re-scaled character image, (d) final character image after rotation.	274

LIST OF ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
ASSOM	Adaptive Subspace Self Organizing Map
BLN	Baseline recognition system
CDCLBA	Contextual Dependent Constrained Level-Building Algorithm
CLBA	Constrained Level-Building Algorithm
DAWG	Direct acyclic word graph
DS	Distributed system
DP	Dynamic Programming
HMM	Hidden Markov Models
k -NN	k Nearest Neighbors
k -NP	k Nearest Prototypes
LBA	Level Building Algorithm
LDLBA	Lexicon-Driven Level Building Algorithm
MAP	Maximum <i>a posteriori</i>
MLP	Multilayer Perceptron
NIST	National Institute of Standards and Technology
NN	Neural Network
OCR	Optical Character Recognizer
PC	Personal Computer
RBF	Radial Basis Function
SNN	Segmental Neural Networks
SRTPT	Service de Recherche Technique de la Poste
SVM	Support Vector Machines

LIST OF NOTATIONS

a_{ij}^{Φ}	Probability to pass from a state i at frame t to a state j at frame t , and producing a null observation symbol Φ
$a_{ij}^{O_{t-1}}$	Probability to pass from a state i at frame $t - 1$ to a state j at frame t , and producing an observation symbol O_{t-1}
α	Weight factor associated with the baseline recognition system
α_t	Back pointer array
$A = \{a_{ij}\}$	State transition probability distribution
$B = \{b_{ij}\}$	Observation symbol probability distribution
B_t	Array that stores the backtrack pointer
B_t^*	Level output back pointer
β	Weight factor associated with the verification module
β_i	Network internal activity level of neuron i
c	Character class
C	Number of character classes
\mathcal{C}	Set of sub-word reference patterns
c_l^v	The l -th sub-word unit in a word reference pattern R_t
$ctr(.,.)$	Probability of changing or keeping the same writing style within a word
$\chi(.,.)$	Best score (accumulated probability) along a single path
d	Duration of a segment
$d(.,.)$	Minimum distance between a sequence of primitives and a reference pattern
D_{max}	Maximum number of observation that an HMM can emit
$del(.)$	Cost parameter for deletion
δ_t	Best score (accumulated probability) along a single state path that accounts for the first t observations

$\hat{\delta}_t$	Best score (accumulated probability) along a single character path that accounts for the first t observations
e	Time index or ending time index
e^*	Time constraint control factor
E_f	Efficiency of the distributed recognition scheme
$\epsilon(\dots)$	Best character state sequence
$\hat{\epsilon}(\dots)$	Best word state sequence
f	State index or final state index
f_n^l	The n -th parameter in a character reference pattern λ_l
F_c	Normalized frequency of a character class c
Φ	Null observation symbol
G	Number of distinct observation symbols
H	Number of models (HMMs) for a character class
\mathbb{H}	Number of different models for a single character class
i	State index or initial state index
$ins(\cdot)$	Cost parameter for insertion
k	Number of candidates at each observation frame
κ	Rejection threshold
l	Character index or character position within a word
L	Length of a word (number of sub-word units or characters)
\hat{L}_n	Label for the n -th word hypothesis
L_n	Label for the n -th word hypothesis
L_n^*	Label for the n -th word hypothesis
\mathbb{L}	Average length of the words in \mathcal{R}
\mathbb{L}'	Average length the words in \mathcal{R} represented as a trie
\mathbb{L}''	Reduced average length the words due to length constraint

\mathbb{L}_s	Average number of shared characters per word in a lexicon
\mathbb{L}_v^*	Length constraint control factor
λ	Character model
$\hat{\lambda}$	Word model
λ_m	The m -th sub-word reference pattern
m	Model or reference pattern index
M	Number of sub-word reference patterns or HMMs
\mathcal{M}	Number of parameters in sub-word models
\mathcal{N}	Number of parameters in the character reference patterns or states in HMMs
N	Dimension of the reference patterns
\mathcal{N}_p	Number of processors
N'	Dimension of the reduced reference patterns
\mathcal{N}_{cl}	Number of characters in a flat lexicon
\mathcal{N}_{ct}	Number of characters in a tree-structured lexicon
O	Sequence of observations
o_t	The t -th symbol in a sequence of observations
\mathcal{O}	Number of operations
ω	Best state along a single path
$\hat{\omega}$	Best character along a single path
P	Length of a sequence of primitives
P^*	Composite probability score
\mathbb{P}	Number of threads or processors
$P(\cdot)$	Probability
\hat{P}_w	Word probability
\hat{P}_n	Estimate of the <i>a posteriori</i> probability assigned to the n -th word hypothesis
P_t	Array that stores the resulting probabilities

P_t^*	Best level output probability
$Ph(x_i)$	Number of pixels with $x = x_i$ for a horizontal histogram
$Ph(y_i)$	Number of pixels with $y = y_i$ for a vertical histogram
$\Pi = \{\pi_i\}$	Initial state distribution
Ψ_{HMM}	Output of the baseline recognition system
Ψ_{NN}	Output of the character recognizer
q_t	State at time t
\hat{q}	Character output backpointer
\mathcal{R}	Set of word reference patterns
\mathcal{R}'	A reduced set of word reference patterns
R_v	The v -th reference pattern
R_c	Repetition factor for a character class c
R_{th}	Rejection threshold
rf	Reduction factor for the average length of the words in a lexicon
s	Time index or beginning time index
s^*	Time constraint control factor
S	Sequence of primitive segments
\hat{S}	Average number of samples over all character classes
S_c	Number of samples in a character class c
S_p	Speedup of the distributed recognition scheme
\hat{S}_n	Set of segments that corresponds to the segmentation of words into characters
s_p	The p -th element in a sequence of primitives
$sub(\dots)$	Cost parameter for substitution
T	Length of a sequence of observations
\mathbb{T}	Dimension of the sequence of observations
\mathbb{T}'	Reduced dimension of the sequence of observations due to the time constraint

τ	Writing style transition probability
V	Number of words or word reference patterns
\mathbb{V}	Number of words in the vocabulary
\mathbb{V}'	Number of words in a reduced vocabulary
φ	Minimum cost to go from a grid point to another
φ_l^*	Best hypothesis score at l
ς	Cost to associate a character with an aggregation of primitives
w	Test pattern (or word to be recognized)
\hat{w}	Best word hypothesis
W_t^*	Level output character indicator
X	Input vector of the neural network
x_t	Segment that represents a character or a pseudo-character or a grid point
y	Output of the neural network
\hat{y}	Corrected neural network output
z_g	Observation symbol
\mathcal{Z}	Set of discrete observation symbols
ζ_t	Variable that either a null or a non-null transition.

INTRODUCTION

Handwriting is one of the most important ways in which civilized people communicate. It is used both for personal (e.g. letters, notes, addresses on envelopes, etc.) and business communications (e.g. bank checks, tax and business forms, etc.) between person and person and for communications written to ourselves (e.g. reminders, lists, diaries, etc.).

Our handwriting is the product of brain and hand, mind and body - thoughts expressed on paper using the muscles of the arm and hand, physical movements controlled by the brain. A person's handwriting is as unique as his/her fingerprints and facial features. However, it varies depending upon the importance: for instance a reminder note is likely to be somewhat different from a legal amount written on a bank check.

Writing is a physical process where the brain sends an order through the nervous system to the arm, hand and fingers, where together they manipulate the writing tool. In this way, the intent to write forms deep within the creative processes of the mind and makes writing an expressive gesture representative of the mind behind the pen. Despite teaching of a standard letter model to form the letters and words necessary to express our ideas, no two writings are exactly alike.

Why Handwriting Recognition

Computers are becoming ubiquitous as more people than ever are forced into contact with computers and our dependence upon them continues to increase, it is essential that they become more friendly to use. As more of the world's information processing is done electronically, it becomes more important to make the transfer of information between people and machines simple and reliable [153].

Since handwriting is one of the most important ways in which people communicate, it would provide an easy way of interacting with a computer, requiring no special training to use effectively. A computer able to read handwriting would be able to process a great amount of data which at the moment is not accessible to computer manipulation.

In addition to a potential mode of direct communication with computers, handwriting recognition is essential to automate the processing of a great number of handwritten documents already in circulation. From bank checks and letters to tax returns and market research surveys, handwriting recognition has a huge potential to improve efficiency and to obviate tedious transcription.

Limitations

Although many researchers say that handwriting recognition is already a mature field, this is not entirely true. Handwriting recognition technology is still far from broad applications. A few dedicated systems are already working, but in industrial applications with dedicated hardware such as recognition of postal addresses [35, 115] or bank check applications [57, 83]. To reach wide application by regular users and to run on regular personal computers or portable gadgets, many advances are still required. Furthermore, the performance of current systems meet the reliability requirements only in very constrained applications. The main constraints that are currently used in handwriting recognition are:

- Well-defined application environments;
- Small vocabularies;
- Constrained handwriting styles (cursive or handprinted);
- User-dependent recognition (or writer-dependent).

A careful analysis of the handwriting recognition field reveals that most of the research has been devoted to relatively simple problems e.g. recognition of isolated digits and characters, recognition of words in a small lexicon. The key point is the number of classes and the ambiguity among them. As the number of classes increases, the amount of data required to develop a good recognition approach increases. Table I presents some recent results from the literature for the problem of handwritten word recognition. It should be stressed that these studies have used different datasets and experimental conditions, which makes a direct comparison of the results very difficult. Furthermore, many of the results presented in Table I are not very significant because they were obtained over very small datasets that may not represent real-life data. However, the results are very helpful to illustrate the current state of the field. Generally the best accuracies reported in Table I are obtained with small lexicons and writer-dependent systems. Examples on how difficult the unconstrained word recognition task can become when large lexicons are used can be seen in the accuracies achieved by the recognition systems that deal with such a problem. A rough representation of the relation between the constraints and the two main criteria to evaluate the performance of handwriting recognition systems is shown in Figure 1. As the number of constraints decreases, the more complex and less accurate the recognition becomes.

A Paradigm for Handwriting Recognition

A wide variety of techniques are used to perform handwriting recognition. A general model for handwriting recognition, shown in Figure 2, is used to highlight the many components of a handwriting recognition system. The model begins with an unknown handwritten word that is presented at the input of the recognition system as an image. To convert this image into information understandable by computers requires the solution to a number of challenging problems. Firstly, a front-end parameterization is needed which extracts from the image all of the necessary mean-

Table I
Recent results on off-line handwritten word recognition

Author	Method	Lexicon Size	Accuracy (%)	Test Set (#)	Comments
Cai and Liu [17]	DP/Fuzzy	14	64.6	113	UNC
Saon [148]	HMM	26	82.50	4,098	UNC, OMNI
Augustin et al. [2]	HMM/NN	28	92.9	40,000	CUR, OMNI
Guillevic et al. [57]	HMM/kNN	30	86.7	4,500	UNC, OMNI
Mohamed et al. [124]	HMM/Fuzzy	100	78.2	317	UNC, OMNI
Gader et al. [47]	NN/DP	100	85.8	1,000	UNC, OMNI
Mohamed et al. [123]	DP	100	89.3	317	UNC, OMNI
Bunke et al. [15]	HMM	150	98.40	3,000	CUR, WD, 5 Writers
Chen et al. [23]	HMM	271	72.3	94	UNC, OMNI
Bippus et al. [7]	HMM	400	89.0	2,000	UNC, OMNI
Procter et al. [137]	HMM	713	88.4	2,031	CUR, WD, 1 Writer
Gader et al. [46]	DP	746	80.41	550	HAND, OMNI
Farouzi et al. [39]	HMM/NN	1,000	95.5	4,058	UNC, OMNI
Burges et al. [16]	DP/NN	1,000	47	3,000	UNC, OMNI
Favata [40]	DP	1,000	82.0	500	UNC, OMNI
Kim et al. [75]	DP	1,000	73.8	3,000	UNC, OMNI
Chen et al. [22]	HMM	1,000	59.6	94	UNC, OMNI
Madhvanath et al. [110]	DP	1,000	74.0	3,000	CUR, OMNI
Scagliola et al. [149]	DP	1,000	83.8	500	CUR, OMNI
Marti et al. [120]	HMM	7,719	60.05	44,019	UNC, OMNI, 250 Writers
Cho et al. [27]	HMM	10,000	67.09	700	CUR, OMNI
Brakensiek et al. [13]	HMM	30,000	89.2	800	CUR, WD, 4 Writers
Dzuba et al. [33]	DP	40,000	60.7	3,000	CUR, OMNI

UNC: Unconstrained, OMNI: Omnigrapher, CUR: Cursive, WD: Writer-Dependent
HAND: Handprinted.

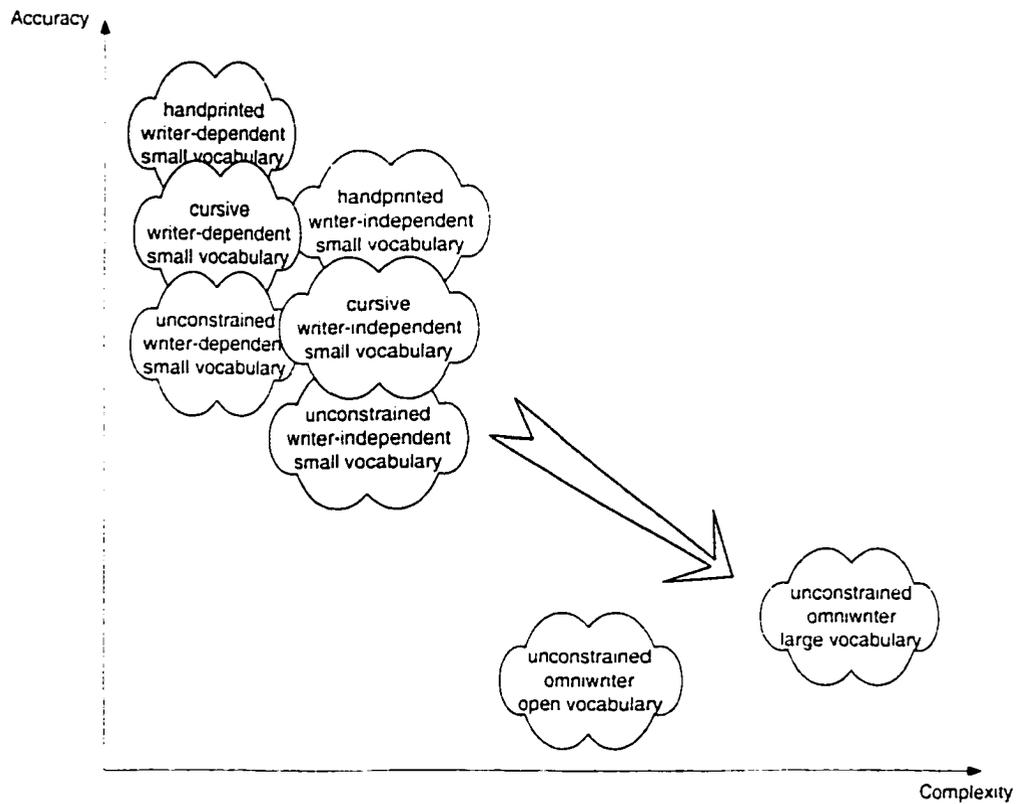


Figure 1 The complexity and accuracy of the sub-problems in handwriting recognition

ingful information in a compact form compatible with the computer language. This involves the pre-processing of the image to reduce some undesirable variability that only contributes to complicate the recognition process. Operations like slant correction, smoothing, normalization, etc. are carried out at this stage. The second step in the front-end parameterization is the segmentation of the word into a sequence of basic recognition units such as characters or pseudo-characters. However, segmentation may not be present in all systems. Some approaches treat words as single entities and attempt to recognize them as a whole [33, 63, 109]. The final step is to extract discriminant features from the input pattern to either build up a feature vector or to generate graphs, string of codes or sequence of symbols. However, the characteristics of the features depend on the preceding step, say whether segmentation of words into characters was carried out or not.

The pattern recognition paradigm to handwriting recognition consists of pattern training, that is, one or more patterns corresponding to handwritten words of the same known class are used to create a pattern representative of the features of that class. The resulting pattern, generally called, reference pattern or class prototype can be an exemplar or template, derived from some type of averaging technique, or it can be a model that characterizes the statistics of the features of the reference pattern. In spite of the goal of most recognition systems to recognize words, sometimes it is difficult to associate one class to each word, then sub-word models (e.g. characters or pseudo-characters models) are trained instead and standard concatenation techniques are used to build up word models during the recognition.

The recognition includes a comparison of the test pattern with each class reference pattern and measuring a similarity score (e.g. distance, probability) between the test pattern and each reference pattern. The pattern similarity scores are used to decide which reference pattern best matches the unknown pattern. Recognition can be achieved by many methods such as dynamic programming (DP) [33, 123], hidden

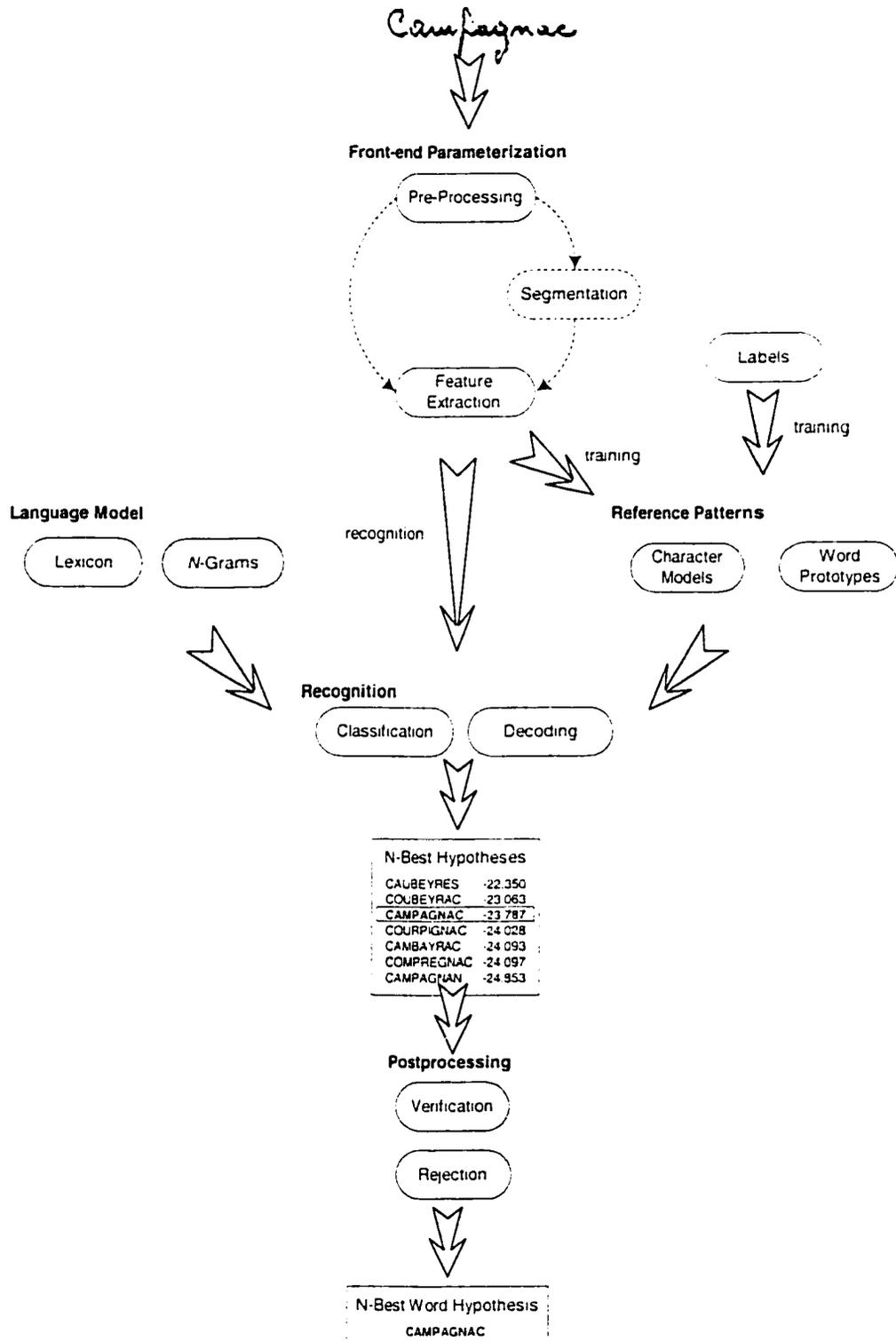


Figure 2 An overview of the basic components of a handwriting recognition system

Markov modeling (HMM) [15, 22, 35], neural networks (NN) [9], k nearest neighbour (kNN) [40], expert systems [63, 136] and combinations of techniques [57, 79]. The recognition process usually provides a list of best word hypotheses. Such a list can be post processed or verified to obtain a more reliable list of word hypotheses [84, 94]. The post-processing or verification may also include some rejection mechanism to discard unlikely hypotheses.

However, for meaningful improvements in recognition, it is necessary to incorporate to the recognition process other sources of knowledge such as language models. A lexicon representing the recognition vocabulary, that is, the words that are expected (allowed) at the input of the recognition system is the most commonly used source of knowledge. Notably, a limited vocabulary is one of the most important aspects of systems that rely on large vocabularies because it contributes to improve the accuracy as well as to reduce computation. In the case of systems that deal with large vocabularies, other additional modules may be included such as pruning or lexicon reduction mechanisms.

Although the above description is not a standard, it is typical of most modern recognition systems. Many of the issues related to the basic modules are common to small and medium vocabularies and they are somewhat overlooked in this thesis. We recommend the interested reader to look at other references that cover these subjects with more details [1, 19, 96, 106, 161, 164, 165].

Why Recognition of Handwritten Words is Difficult

The recognition of handwritten words by computers is a challenging task. Despite the impressive progress achieved during the last years and the increasing power of computers, the performance of the handwriting recognition systems is still far from human performance. Words are fairly complex patterns and owing to the great variability in handwriting style, handwritten word recognition is a difficult one.

The first sort of difficulties is due to the high variability and uncertainty of human writing. Not only because of the great variety in the shape of characters, but also because of the overlapping and the interconnection of the neighboring characters. In handwriting we may observe either isolated letters such as handprinted characters, groups of connected letters, i.e. sub-words, or entirely connected words. Furthermore, when observed in isolation, characters are often ambiguous and require context to minimize the classification errors. Figure 3 shows some examples of the word "PARIS" written by different persons. It clearly illustrates the variability and ambiguity that computers have to deal with to recognize handwritten words.

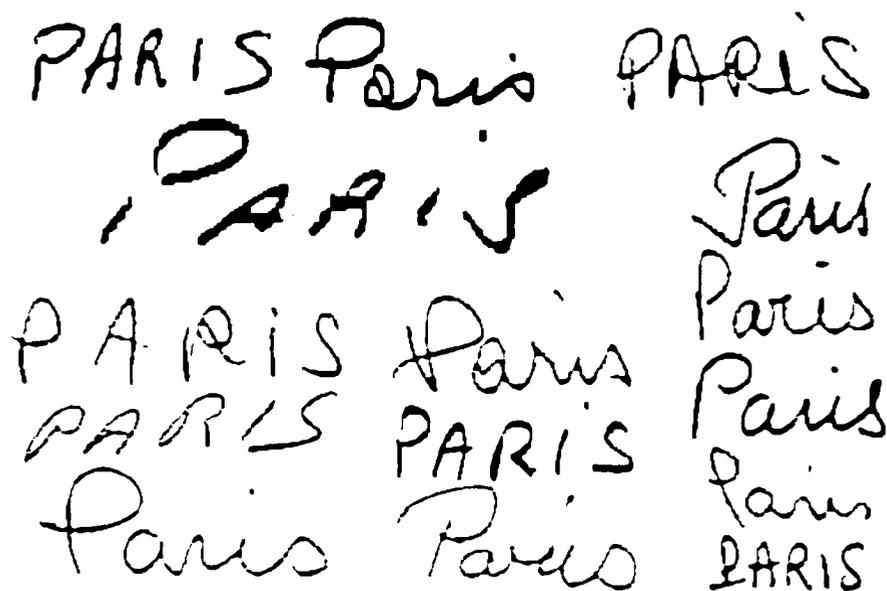


Figure 3 The word "PARIS" written in different styles by different writers

The most natural unit of handwriting is the word, and it has been used for many recognition systems [59, 77, 109]. One of the main advantage of using whole-word models is that these are able to capture within-word co-articulation effects [109]. When whole-word models are adequately trained, they will usually yield the best recognition performance. Global or holistic approaches treat words as single, indi-

visible entities and attempt to recognize them as whole, bypassing the segmentation stage [63,109]. Therefore, for small vocabulary recognition, such is the case of bank check applications where the lexicons do not have more than 30–40 entries [79], whole–word models are the preferred choice.

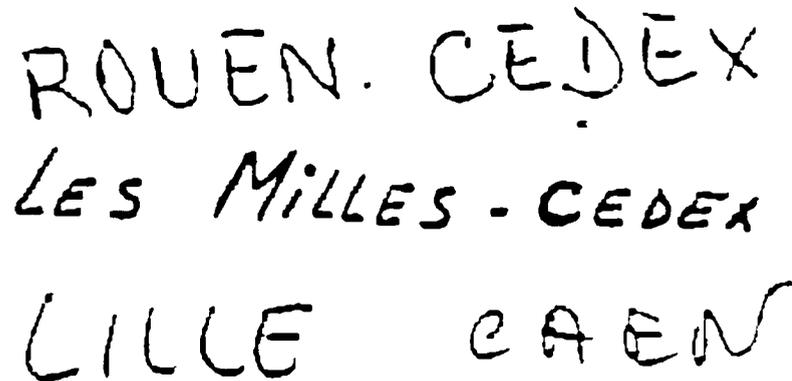
Nevertheless, many practical applications require larger vocabularies with hundreds or thousands of words [51,68,77]. One of the most common constraints of current recognition systems is that they are only capable of recognizing words that are present in a restricted vocabulary, typically comprised of 10–1,000 words [15,23,35,76,138,154]. The restricted vocabulary, usually called *lexicon*, is a list of all valid words that are expected to be recognized by the system. There are no established definitions, however, the following terms are usually used:

- Small Vocabulary - tens of words;
- Medium Vocabulary - hundreds of words;
- Large Vocabulary - thousands of words;
- Very-Large Vocabulary - tens of thousands of words.

While words are suitable units for recognition, they are not a practical choice for large vocabulary handwriting recognition. Since each word has to be treated individually and data cannot be shared between word models, this implies a prohibitively large amount of training data. In addition the recognition vocabulary may consist of words that have not appeared in the training procedure. Instead of using whole–word models, analytical approaches use sub–word units such as characters or pseudo–characters as the basic recognition units, requiring the segmentation of words into these units.

Therefore, the second sort of difficulties lies in the segmentation of handwritten words into characters. Figures 4 and 5 show some examples of words that are easy

and difficult to be segmented into characters respectively. The words in Figure 4 are naturally segmented, that is, each compounding letter was written separately. Such words do not pose a problem to the segmentation. On the other hand, the cursive nature of the words in Figure 5 makes the segmentation very difficult.



Handwritten examples of words segmented into characters. The first line shows 'ROUEN. CEDEX' with a period between the words and a dot under the 'E' in 'CEDEX'. The second line shows 'LES MILLES - CEDEX' with a hyphen between 'MILLES' and 'CEDEX'. The third line shows 'LILLE CAEN' with a space between the words.

Figure 4 Examples of easy cases to segment words into characters where the words are written by well-separated characters

Even with the difficulty and the errors introduced by the segmentation of words, most successful approaches are segmentation-recognition methods in which words are first loosely segmented into characters or pieces of characters, and dynamic programming techniques with a lexicon are used in recognition to choose the definitive segmentation as well as to find the best word hypotheses. As a result, this approach is the preferred choice for applications where large vocabularies are required.

This Thesis

After this brief overview of the current state of the handwriting recognition field and the difficulties that arise when the number of constraints are reduced, we are ready to present what this thesis is all about.

The primary goal of this thesis is to propose a writer-independent (omniwriter) off-

Sigmeres-Chatelain
 plouzane Jeumont
 ST Amand les Eux
 Illkirch Cedex
 Saint Dizier

Figure 5 Some examples of words which are very difficult to be segmented into characters

line handwritten word recognition system that deals efficiently with large and very-large vocabularies (80,000 words), unconstrained handwriting (handprinted, cursive and mixed), and runs on regular personal computers (PC's) with an acceptable performance both in terms of recognition accuracy and recognition speed. However, we do not propose a complete computer system build from scratch that performs handwriting recognition, but instead we use many modules of a baseline recognition system, which is described in Chapter 4.

The main challenge with no doubts is the aspect of large vocabulary. Passing from a small lexicon of tens of words to ten thousands of words is not as simple as it seems. Besides the problem of ambiguity and variability that are boosted, another factor that has been neglected comes up, the computational complexity. To tackle the problem of computational complexity we have developed some novel search and recognition strategies that are able to deal efficiently with large vocabularies and multiple character class models, achieving speedup factors of 10 to 120. To tackle the problem of recognition accuracy we have developed a post-processing module that corrects many errors in the recognition of words, improving the recognition rate by almost 10%.

Original Contributions

Until now, large vocabulary off-line handwriting recognition was not feasible due to the lack of computational efficient and fast decoding algorithms. So, researchers have avoided to deal with large vocabularies. Larger vocabularies reported in the literature (see Table I) have about 40,000 words. So, the subject of this thesis is already itself original. Besides that, the algorithms and methods that have been proposed to accomplish our goal of building a large vocabulary handwritten word recognition system are also original contributions. The original contributions of this work are twofold and they can be summarized as follows.

Recognition Speed:

- The integration of a lexical tree search and multiple character class models in an efficient way [90];
- The use of a level-building algorithm as a means to alleviate the complexity of including multiple character class models and contextual-dependent models during the recognition process by choosing and expanding only the more likely character model (uppercase or lowercase) at each level [89];
- The integration of three constraints into the level-building algorithm and the use of statistical experimental design techniques to setup the control parameters that maximize both the recognition accuracy and recognition speed [86];
- The adaptation of the constraints to the identity of the characters to further improve the performance of the constrained level-building algorithm [89];
- A novel fast two-level decoding algorithm devoted to decode observation sequences in an HMM framework that speedup the recognition process while preserving the accuracy [87];
- The concepts of modularity and reusability of character models for lexicon-driven word recognition [87];
- The use of distributed computation concepts to build up a distributed recognition system where the recognition task is split into several sub-tasks [85];
- Speedup factor about 120 while preserving the recognition accuracy;

Recognition Accuracy:

- The development of a post-processing paradigm for the verification of unconstrained handwritten word based on the N -best word hypotheses [84];
- The use of an isolated character recognizer to verify the segmentation of the N -best word hypotheses [84];
- The use of the segmentation information provided by a baseline recognition system to segment words into characters [84];

- The use of neural networks as a back-end classifier to verify the segmentation and assign *a posteriori* probabilities to isolated characters [84]:
- The combination of a baseline recognition system based on HMMs with a verification module based on NNs to optimize the performance and improve the recognition accuracy:
- A rejection mechanism based on the combination of a baseline recognition system with a verification module to improve the reliability.

How to Read this Thesis

The thesis is divided into seven chapters. This chapter describes the aim and contents of this thesis and gives an overview of the main results. Chapter 1 describes the aims and achievements of other work in the field of handwriting recognition focusing on three aspects that are important to understand the contributions of our own work: large vocabulary handwriting recognition, isolated character recognition and verification of handwritten words. Chapter 2 presents a concise statement of the problems that this thesis tackles, and an analysis of the state of the art of the proposed methods and strategies presented in Chapter 1 as well as a number of practical applications that may profit from the results of our research. Chapter 3 presents an overview of the baseline recognition system used from which many modules were borrowed and with which the performance of the algorithms and methods proposed in this thesis are compared.

The contributions of this thesis are presented in Chapter 4. This chapter describes the algorithms and techniques used and developed for the work presented in this thesis, such as the lexicon-driven level building algorithm (LDLBA), the constrained level building algorithm (CLBA), the fast two-level decoding algorithm (FS), the distributed recognition scheme (DS), the recognition-verification approach based on the combination of hidden Markov models and neural networks, and the rejection

mechanism. Chapter 5 presents the details of databases, conditions and experiments carried out to assess the performance of the techniques presented in Chapter 4 as well as the results of the application of the proposed recognition strategies to large and very-large vocabulary handwriting recognition.

The final chapter draws together the conclusions of the chapters and summarizes the results to put the achievements of this thesis into the context of the findings of other groups. A brief outlook on the future work which could be carried on from this thesis is also included.

We have assumed that the reader is familiar with the theory of HMMs and neural networks. For this reason, we do not devote much attention to such subjects. For those who do not feel comfortable with such subjects we recommend the reading of following references: [36, 66, 139] for HMMs and [8, 43] for NNs.

CHAPTER 1

STATE OF THE ART IN HANDWRITING RECOGNITION

In this chapter we present an overview of the state of the art in handwriting recognition focusing on the aspects that are closely related to our research: methods and strategies for large vocabulary handwriting recognition [88], isolated handwritten character recognition and verification in handwriting recognition.

First we review the recognition strategies and methods to handwriting recognition. Next we focus on the approaches that have been proposed to deal with large vocabularies. These approaches mainly address the problems related to the computational complexity of the recognition process. Following the aspect large vocabulary, we review some approaches to recognize isolated handwritten characters since we address such an aspect later in this thesis. In the last part we review some approaches used at the back end of handwriting recognition systems, that are devoted to the postprocessing of multiple hypotheses generated during the recognition process.

1.1 Recognition Strategies

A key question in handwriting recognition is how test and reference patterns are compared to determine their similarity. Depending on the specifics of the recognition system, pattern comparison can be done in a wide variety of ways. The goal of a classifier is to match a sequence of observations derived from an unknown handwritten word against the reference patterns that were previously trained, and obtain confidence scores (distance, cost, or probabilities) to further decide which model best represents the test pattern. Here, we have to distinguish between word models and sub-word models. As we have pointed out in the preceding chapter, approaches that use sub-word models are more suitable for large vocabulary applications. So, therefrom we assume that the reference patterns are related to sub-word units or

characters.

An observation sequence can be represented by different ways: by low-level features, such as smoothed traces of the word contour, stroke direction distributions, pieces of strokes between anchor points, local shape templates, etc [52, 53]; by medium-level features that aggregate low-level features to serve as primitives include edges, end-points, concavities, diagonal and horizontal strokes, etc. [33, 59]; by high-level features such as ascenders, descenders, loops, dots, holes, t-bars, etc. Moreover, such features can be used in different manners to build up feature vectors, graphs, string of codes or sequence of symbols. Here, it is convenient to distinguish between two particular representations of the test pattern: as a sequence of observations or as a sequence of primitive segments. We define a test pattern w as a sequence of observations such that $O = (o_1 o_2 \dots o_T)$ in which T is the number of observations in the sequence and o_t represents the t -th symbol. We define S as a sequence of primitive segments of the image such that $S = (s_1 s_2 \dots s_p)$, in which P is the number of segments in the sequence and s_p represents the p -th primitive. In a similar manner we define a set of reference patterns $\mathcal{R} = \{R_1, R_2, \dots, R_V\}$ in which V is the number of reference patterns, R_v is the reference pattern that represents a word that is formed by the concatenation of sub-word units (or characters) such that $R_v = (c_1^v c_2^v \dots c_L^v)$ in which L is the total number of sub-word units that form a word, and c_l^v represents the l -th sub-word unit. The goal of the pattern comparison stage is to determine a similarity score (cost, distance, probability, etc.) of O or S to each $R_v \in \mathcal{R}$, in order to identify the reference pattern that gives the best score, and to associate the input pattern with the class of this reference pattern. Since words are broken up into sub-word units, the recognition strategies used are essentially based on dynamic programming (DP) methods that attempt to match primitives or blocks of primitives with sub-word units to recognize words. Depending on how the words are represented, statistical classification techniques, heuristic matching techniques, symbolic matching methods, or graph matching methods are some of

the possible matching methods that can be used [109]. In handwriting recognition, the optimal interpretation of a word image may be constructed by concatenating the optimal interpretation of the disjoint parts of the word image.

In terms of optimal path problem, the objective of the DP methods is to find the optimal sequence of a fixed number of moves, say L , starting from point i and ending at point j , and the associated minimum cost $\varphi_L(i, j)$. The P points representing the sequence of P primitives are plotted horizontally and the L points representing the sub-word models (or the moves) are plotted vertically as shown in Figure 6¹. The Bellman's principle of optimality [6] is applied in this case, and after having matched the first l -th moves, the path can end up at any point k , $k = 1, 2, \dots, P$, with the associated minimum cost $\varphi_l(i, k)$. The optimal step, associating the first $l + 1$ characters with the first p primitives is given as:

$$\varphi_{l+1}(i, p) = \min_k [\varphi_l(i, k) + \zeta(k, p)] \quad (1.1)$$

where $\varphi(\cdot)$ is the minimum cost (or best path) and $\zeta(\cdot)$ represents the cost to associate the $(l + 1)$ -th character to the aggregation composed by primitives $i + 1, i + 2, \dots, p$.

Besides the matching strategy, the segmentation-based methods used in large vocabulary handwriting recognition lie within two categories:

- Character recognition followed by word decoding - Characters or pseudo-characters are the basic recognition units and they are modeled and classified independently of the words, that is, the computation of the cost function is

¹Note that this is a generic representation of the DP method and it may not reflect a practical case of aligning a sequence of primitives and sub-word models, since usually spatial and temporal constrains can be used to limit the search effort

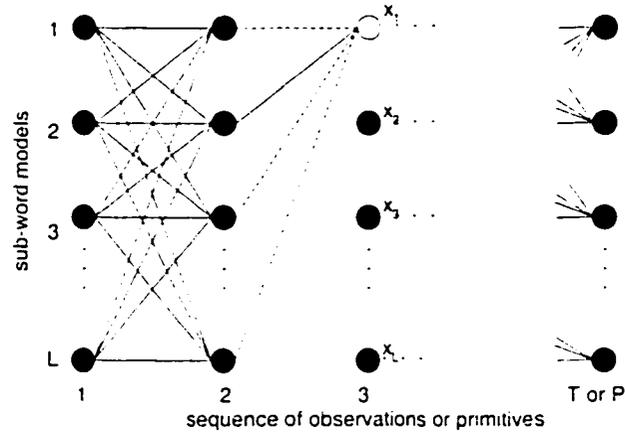


Figure 6 A trellis structure that illustrates the problem of finding the best matching between a sequence of observations O (or sequence of primitives S) and a reference pattern $R_r = (c_1 c_2 \dots c_L)$

replaced by an ordinary optical character recognizer (OCR) that outputs the most likely character and its confidence level given a primitive or a block of primitives. To this aim, pattern recognition approaches such as template matching, structural techniques, neural networks and statistical techniques [70] can be used. Further, character scores are matched with lexicon entries by dynamic programming methods [9, 40];

- Character recognition integrated with word decoding - Characters or pseudo-characters are the basic recognition units and they are concatenated to build up word models according to the lexicon. The classification is carried out by dynamic programming methods that evaluate the best match between the whole sequence of observations and word models [15, 22, 35].

A simplified dynamic programming approach relies on minimum edit-distance classifiers (usually using the Levenshtein's metric) that attempt to find a reference pattern R_r that has the minimum cost with respect to the input pattern O (or S) [161] as:

$$d(O, R_v) = \min \begin{cases} d(o_1 \dots o_{T-1}, c_1^v \dots c_{L-1}^v) + \text{sub}(o_T, c_L^v) \\ d(o_1 \dots o_{T-1}, c_1^v \dots c_L^v) + \text{ins}(o_T) \\ d(o_1 \dots o_T, c_1^v \dots c_{L-1}^v) + \text{del}(c_L^v) \end{cases} \quad (1.2)$$

where $d(O, R_v)$ is the minimum distance between O and R_v , $\text{del}(c_L^v)$, $\text{sub}(o_T, c_L^v)$, and $\text{ins}(o_T)$ are the cost parameters for deletion, substitution, and insertion respectively.

So far, handwriting recognition using neural networks (NN) has mostly been aimed at digit recognition [97, 131], isolated character recognition [9], and small vocabulary word recognition [132] because in large vocabulary handwriting recognition, words must be segmented before neural network modeling [16]. With large vocabularies, NNs are not frequently used as front-end classifiers, but as part of hybrid approaches, where they are used to estimate *a priori* class probabilities [79, 83, 154], *a priori* grapheme probabilities [39], or to verify results of previous classifiers (as a back end classifier) [84].

Statistical techniques use concepts from statistical decision theory to establish decision boundaries between pattern classes [70]. Techniques such as k nearest neighbor decision rule [40], Bayes decision rule, support vector machines [168], and clustering [167] have been used in handwriting recognition but mostly aiming the recognition of isolated characters and digits or words in small vocabularies. However, during the last decade, hidden Markov models (HMMs), which can be thought of as a generalization of dynamic programming techniques, have become the predominant approach to automatic speech recognition [128]. The HMM is a parametric modeling technique in contrast with the non-parametric DP algorithm. The power of the HMM lies in the fact that the parameters that are used to model the signal can be well optimized, and this results in lower computational complexity in the decoding procedure as well as improved recognition accuracy. Furthermore, other

knowledge sources can also be represented with the same structure, which is one of the important advantages of the hidden Markov modeling [66].

The success of HMMs in speech recognition has led many researchers to apply them to handwriting recognition by representing each word image as a sequence of observations. The standard approach is to assume a simple probabilistic model of handwriting production whereby a specified word w produces an observation sequence O with probability $P(w, O)$. The goal is then to decode the word, based on the observation sequence, so that the decoded word has the maximum *a posteriori* (MAP) probability, i.e.:

$$w \ni P(w|O) = \max_{w \in \mathcal{R}} P(w|O) \quad (1.3)$$

The way we compute $P(w|O)$ for large vocabularies is to build statistical models for sub-word units (characters) in an HMM framework, build up word models from these sub-word models using a lexicon to describe the composition of words, and then evaluate the model probabilities via standard concatenation methods and DP-based methods such as the Viterbi algorithm [23, 35]. This procedure is used to decode each word in the lexicon.

In fact the problem of large vocabulary handwriting recognition is turned into an optimization problem that consists of evaluating all the possible solutions and choosing the best one, that is, the solution that is optimal under certain criteria. The main problem is that the number of possible hypotheses grows as a function of the lexicon size and the number of sub-word units and that imposes formidable computation requirements on the implementation of search algorithms [23].

1.2 The Role of Language Model

The fact is that whatever the recognition strategy is, contextual knowledge (linguistic, domain, or any other pertinent information) needs to be incorporated to the recognition process to reduce the ambiguity and achieve acceptable performance. The lexicon is such a source of linguistic and domain knowledge. The majority of the recognition systems rely on a lexicon during the recognition, the so-called lexicon-driven systems, or also after the recognition as a post-processor of the recognition hypotheses [138, 156, 169]. However, systems that rely on a lexicon at the early stages have obtained more success since they look directly for a valid word [138]. Lexicons are very helpful in overcoming the ambiguity involved in the segmentation of words into characters and variability of character shapes [20, 107]. Furthermore, lexicons are not only important in improving the accuracy, but also in limiting the number of possible word hypotheses to be searched [90, 138]. This is particularly important to limit the computational complexity during the recognition process. One of the major problems of lexicon-driven systems is the encounter of an out-of-vocabulary word [138, 153]. In such cases, words cannot be correctly recognized. So, the system must provide some sort of mechanism to recognize them as out-of-vocabulary words instead of misclassifying them or even to reject them. However, for many applications, the size and content of vocabulary is controllable, and this problem can be managed (handled).

There are systems that do not use a limited vocabulary, but a lexicon of characters and statistical language model at the character level (n -grams on sequences of characters) [4, 12, 13]. For isolated word recognition, the idea is to use character-based language properties to model a word as a sequence of characters rather than word models. The use of n -grams enables the recognition of words with open vocabulary [4, 13, 29, 153, 162]. In spite of the flexibility of the open-vocabulary recognition systems, the achieved performance is still far below the approaches that rely on

limited lexicons, and it is not good enough for practical use [29]. Brakensiek et al. [12, 13] report a decrease of about 26% in word accuracy using no lexicon for an off-line handwritten word recognition system.

The recognition of isolated words in a large vocabulary is already a demanding problem. If we attempt to recognize phrases and sentences, linguistic constraints have to be used to limit the search space. The use of a language model brings great gains in recognition accuracy. But a lexicon which limits the search to a set of permitted words is not the only solution. Grammars can also be used to limit which words are permissible in a given context to account for the frequencies of different words. However, grammars are typically used at the word level in the recognition of phrases and sentences, but not on isolated words [77, 153].

1.3 Large Vocabulary Handwriting Recognition

In this section the current state of large vocabulary off-line handwriting recognition is presented. We review many recent advances that have occurred in this field, particularly over the last decade. The methods and techniques that have been developed basically attempt to reduce the computational complexity of the recognition process by reducing the lexicon size, or reorganizing the search space, or using heuristics to limit the search efforts. Most of the methods that will be discussed in this section are shown in Figure 7.

However, this literature review is not exhaustive since there are other ways to deal with the large vocabulary complexity such as the approaches related to feature vector reduction. Feature selection methods primarily aims to select more discriminant features to improve the recognition accuracy but these methods can also be used to reduce the dimensionality of feature vectors, leading to less complex recognition tasks. The readers interested in investigating this particular aspect may refer to a more general description of feature selection methods [69, 92] as well as to appli-

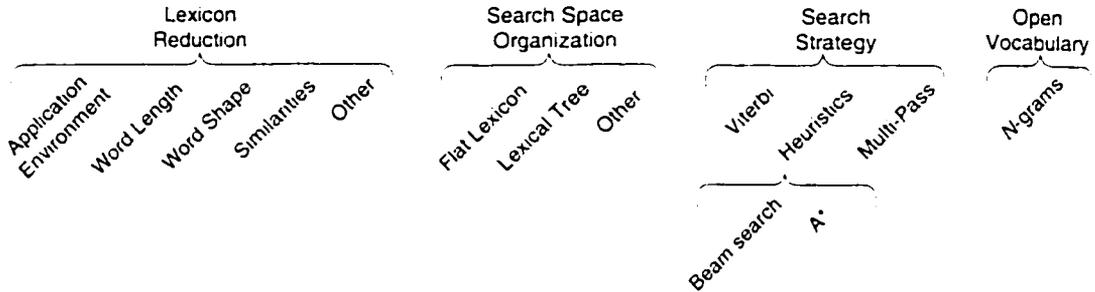


Figure 7 A summary of the strategies for large vocabulary handwriting recognition

cations on handwriting recognition [78, 130]. In the same manner, we do not cover methods related to the reduction of the number of class models (or class prototypes), since this approach is not very common [89, 166].

1.3.1 Lexicon Reduction

One of the elements that contributes more to the computational complexity of the recognition task is the size of the lexicon. The problem with large lexicons is the number of times that the observation sequence extracted from the input image has to be matched against the words (or reference vectors) in the lexicon. So, the more intuitive approach attempts to limit the number of words to be compared during the recognition. Basically, pruning methods attempt to reduce the lexicon prior to recognition, that is, to reduce a global lexicon \mathcal{R} to \mathcal{R}' where $\mathcal{R}' \subset \mathcal{R}$.

There is a chance that the pruning methods may throw away the true word hypothesis. Here we introduce the definition of *coverage* that refers to the capacity of the reduced (pruned) lexicon to include the right answer. So, the coverage indicates the error brought about by pruning (reducing) the lexicon. The effectiveness of a lexicon reduction technique can be measured by its coverage which ideally has to be kept at

100% to avoid the introduction of errors. However, many authors do not report the performance of the lexicon reduction in terms of coverage but looking directly the effect on the recognition accuracy. This is the case of schemes that embed pruning mechanisms into the recognition process.

There are some basic ways to accomplish such lexicon reduction task: knowledge of the application environment, characteristics of the input pattern, and clustering of similar lexicon entries. Unquestionably, the application environment is the main source of information in limiting the lexicon size. In some cases, such as in bank check processing, the size of the lexicon is naturally limited to tens of words. Sometimes, even for applications where the number of words in the lexicon is large, additional sources of knowledge are available to limit the number of candidates to tens or hundred words. Other methods attempt to perform a pre-classification of the lexicon entries to evaluate how likely is the matching with the input image. These methods basically look at two aspects: word length and word shape. Other approaches attempt to find similarities between lexicon entries and organize them into clusters. So, during the recognition, the search is carried out only on words that belong to more likely clusters. The details of some methods are presented as follows.

1.3.1.1 Other Sources of Knowledge

Basically, in handwriting recognition the sources of knowledge that are commonly used depend on the application environment. The application environment usually is a rich source of contextual information that helps us to reduce the complexity of the problems to more manageable ones. Typical examples are banking and postal applications and language syntax.

1.3.1.1.1 Banking Applications

One of the areas where the researchers have devoted attention is the recognition of legal amounts on bank checks. The reason is very simple: the lexicon size is limited to tens of words. This facilitates the gathering of the data required for training and testing. Furthermore, there is also the courtesy amount that can be used to constrain (parse) the lexicon of the legal amount [58, 59, 99, 147] or to improve the reliability of the recognition.

1.3.1.1.2 Postal Applications

Maybe the postal application is the area where handwriting recognition techniques have been used more often [35, 51, 71, 159]. Most of the proposed approaches attempt to first recognize the ZIP codes to further read other parts of the address, depending on the reliability in recognizing the ZIP code. Conventionally, the ZIP code allows the system to reduce the lexicons of thousands entries to a few hundred words [35, 38, 47, 51, 71, 98]. So, the reduced lexicon can be processed using the conventional search techniques such as Viterbi and Dynamic Programming methods (DP).

1.3.1.1.3 Language Syntax

However, when no additional source of knowledge is available, other alternatives are necessary. In the case of generic content recognition, where the words are associated to form phrases and sentences, the application environment contributes little to reduce the lexicon. But here, the linguistic knowledge plays an important role in limiting the lexicon. The use of language models based on grammars is very important to not only reduce the number of candidate words at each part of the text, but also to improve the accuracy [12, 13, 118–120]. However this source is more suitable for the recognition of sentences than isolated words.

1.3.1.2 Word Length

Short words can be easily distinguished from long words by comparing their lengths only. So, the length is a very simple criterion for lexicon reduction. The length of the observation sequence (or feature vector) extracted from the input image has intrinsically a hint about the length of the word from which the sequence was extracted. Many lexicon reduction methods make use of such information to reduce the number of lexicon entries to be matched during the recognition process [56,71,73,82,86,136]. Kaufmann et al. [73] use a length classifier to eliminate from the lexicon the models that differ significantly from the unknown pattern in the number of symbols. For each model, a minimal and a maximal length are determined. Based on this range, a distance between a word and the model class is defined and used during the recognition process to select only the pertinent models. Kaltenmeier et al. [71] use the word length information given by a statistical classifier adapted to features derived from Fourier descriptors for the outer contours to reduce the number of entries in vocabulary of city names.

Other methods do not rely on the feature vector to estimate the length of words, but on particular techniques. Kimura et al. [82] estimate the length of the possible word candidates using the segments resulting from the segmentation of the word image. Such estimation provides a confidence interval for the candidate words, and the entries outside of such an interval are eliminated from the lexicon. An overestimation or an underestimation of the interval leads to errors. Furthermore, the estimation of the length requires a reliable segmentation of the word, what still is an ill-posed problem. Powalka et al. [136] estimate length of cursive words based on the number of times an imaginary horizontal line drawn through the middle of the word intersects the trace of the pen in its densest area. A similar approach is used by Guillevic et al. [56] to estimate word length and reduce the lexicon size. The number of characters is estimated using the counts of stroke crossings within

the main body of a word.

1.3.1.3 Word Shape

The shape of the words is another good hint about the length and the style of the words. Zimmerman and Mao [179] use key characters in conjunction with word length estimation to limit the size of the lexicon. They attempt to identify some key characters in cursive handwritten words and use them to generate a search string. This search string is matched against all lexicon entries to select those best matched. A similar approach is proposed by Guillevic et al. [56], but instead of cursive script, they consider only uppercase words. First they attempt to locate isolated characters that are further preprocessed and passed to a character recognizer. The character recognition results are used along with the relative position of the spotted characters to form a grammar. An HMM module is used to implement the grammar and generate some entries that are used to dynamically reduce the lexicon. Kaufmann et al. [73] proposed a method of reducing the size of vocabulary based on the combination of four classifiers: length classifier, profile range, average profile, and transition classifier. All the classifiers use as input the same feature vector used by the recognition system. Seni et al. [152] extract a structural description of a word and use it to derive a set of matchable words. This set consists of entries from the lexicon that are similar in shape or structure to the input word. The set of matchable words forms the reduced lexicon that is employed during the recognition process.

Madhvanath and Govindaraju [107] present a holistic lexicon filter that takes as input a chain code of a word image and a lexicon and returns a ranked lexicon. First the chain code is corrected for the slant and the skew and features such as natural length, ascenders, and descenders are extracted as well as assertions about the existence of certain features in certain specific parts of the word. The same features are extracted from lexicon entries (ASCII words) by using heuristic rules to combine

the expected features of the constituent characters. A graph-based framework is used to represent the word image, the lexicon entries and their holistic features and for computing three different distance measures (confidence of match, closeness, and degree of mismatch) between them. These three measures are computed for each lexicon entry and used to rank the hypotheses. A 50% reduction in the size of the lexicon with 1.8% error is reported for a set of 768 lowercase images of city names. The same idea is used by Madhvanath et al. [108, 113] for pruning large lexicons for the recognition of off-line cursive script words. The holistic method is based on coarse representation of the word shape by downward pen-strokes. Elastic matching is used to compute the distance between the descriptor extracted from the image and the ideal descriptor corresponding to a given ASCII string. Distance scores are computed for all lexicon entries and all words greater than a threshold are discarded. Hennig and Sherkat [63] also use several holistic methods to reduce the lexicon in a cursive script recognition system. Features such as word length, diacritical marks, ascenders, descenders, combined as/descenders, and segments crossing the word's axis as well as several tolerance factors are used. The first method was based on the letter candidates produced by a hierarchical fuzzy inference method [64] and the known distribution of the width of those candidates achieved a reduction of 44% of the hypotheses with an error rate of 0.5% for a 4,126-word vocabulary. Using the number of possible axis crossings instead of letter candidates leads to a reduction of 30% with the same error rate. An extension of the method evaluates the occurrence of other physical features, such as ascenders, descenders, diacritical marks and as/descenders. The resulting reduction in the lexicon ranged from 53% to 99%.

Leroy [101] presents an approach for lexicon reduction in on-line handwriting based on global features. First, alphabetic characters are encoded using global features (silhouette) with an associated probability. Using the silhouette of the characters, the words in the lexicon are encoded and the probability of each word is given by the product of the compounding character silhouettes. Next, an epigenetic network

is used to select the words in the lexicon that are best described by the silhouettes. An extension of this approach is presented by Leroy [102] where the silhouettes are combined to form words and a neural network is used to relate words to silhouettes. Those words that give the best scores are selected and encoded by more sophisticated features, such as elliptic arcs and loop-shapes. Another neural network is used to select a new subset of words that give the best scores. Finally, diacritic marks, as proposed in [101] are used to select the final word candidates.

Table II summarizes the experimental results obtained by some of the lexicon reduction methods presented above. The elements presented in Table II are the number of words in the lexicon, the number of samples used to test the proposed approach, the lexicon reduction achieved, the coverage of the resulting lexicon, and the speedup in the recognition obtained by using the lexicon pruning. The speedup is not available for all methods, because most of them are presented separately from the recognition system. Table III shows some results of pruning methods for on-line handwriting recognition. Notice that the tables have different columns because the respective information is not always available for the methods presented.

1.3.1.4 Other Approaches

Other approaches work toward different principles. Some of them avoid matching the input data against all lexicon entries during the search based on some measure of similarity of the lexicon entries.

Gilloux [50] presented a method to recognize handwritten words from a large vocabulary. The proposed approach uses degraded word models which models zones of the words instead of characters. The decoding is made between the sequence of observations and these degraded models. This allows a fast computation of word conditioned sequence probabilities. Furthermore, models are clustered according to their length and features. So, words may share a reduced number of base models

Table II

Comparison of lexicon reduction approaches based on word shape for off-line handwriting recognition. Results are for cursive (lowercase) and handprinted (uppercase) words respectively

Author	Lexicon Size	Test Set	Lexicon Reduction (%)	Coverage (%)	Speedup Factor
Madhvanath et al. [107]	1k	768	50-90	98.2-75.0	—
Zimmermann et al. [179]	1k	811	72.9	98.6	2.2
Madhvanath et al. [108]	21k	825	99	74	—
Madhvanath et al. [112]	21k	825	95	95	—
Madhvanath et al. [113]	23.6k	760	95	75.5	—
Guillevie et al. [56]	3k	500	3.5	95.0	—

Table III

Comparison of lexicon reduction approaches based on word shape for on-line recognition of cursive words

Author	Lexicon Size	Test Set	Lexicon Reduction (%)	Coverage (%)
Hennig et al. [63]	4k	3,750	97.5	84
Leroy [101]	5k	250	99.9	22
Leroy [102]	6.7k	600	99.8	76
Seni et al. [151]	21k	750	85.2-99.4	97.7

that are matched only once to the data resulting in a speedup of the recognition process. The original word HMMs and the degraded word models are compared using a 59k-word vocabulary and 3,000 word images. In spite of being 20 times faster, the use of the degraded model causes a significant drop of 30% in the recognition rate. Gilloux [51] also proposes the use of Tabou search to reduce the number of words in the same 59k-word vocabulary. The approach consists of organizing the search space according to the similarity of the lexicon entries. The Tabou method is a strategy for iterative improvement based on the local optimization of an objective function. The criteria to be optimized is the likelihood of the observation sequence that represents a handwritten word, the HMMs associated with the lexicon entries and also the closeness between the HMMs. Lexicon reduction rates of 83-99.2% that correspond to 1.75-28 speedup factors are reported. But this improvement in speed is at the expense of reducing the coverage of the lexicon in 90-46% what implies in a reduction in recognition rate of 4-23%. Farouz [38] presents a method for lexicon filtering based on bound estimation of HMM probability from some properties of the observation sequence extracted from the word image. In the first step of the recognition process, the method estimates this bound for each lexicon entry, and as the entry comes close to the word image, unlikely candidates are eliminated from the lexicon. A lexicon reduction rate of 69% is reported with a drop of 1% in recognition rate. A similar approach is proposed by Wimmer et al. [169] where an edit distance which works as similarity measure between character strings is used to preselect the lexicon to perform a post-processing of a word recognition system. Experimental results show a 100-speedup factor for a 10,600-entry lexicon with a reduction of 3-4% in accuracy. A summary of the results achieved by some of the methods described in this section is presented in Table IV.

Table IV

Other pruning and lexicon reduction strategies for unconstrained off-line handwritten word recognition

Author	Lexicon Size	Test Set	Lexicon Reduction (%)	Coverage (%)	Reduction in Recognition Rate (%)	Speedup Factor
Wimmer et al. [169]*	10.6k	1.5k	—	—	3-4	100
Gilloux [50]	29.4k	3k	—	45-64	30	24
Farouz [38]	49k	3.1k	69	—	1.0	1.75
Gilloux [51]	60k	4.2k	99.2-83.0	46.0-90.0	23-4	28-1.7

* Results refer to post-processing

1.3.2 Search Space Organization

In this section we present some approaches that attempt to reorganize the search space, that is, to reorganize all word hypotheses that have to be decoded during recognition to exploit the presence of common prefixes in words that have similar spellings and avoid repeated computation of the same sequence of characters against the sequence of observations generated from the input image [68, 71, 90, 117]. This approach avoids a reduction in the coverage, since, the number of words in the lexicon is not changed. There are two basic ways of organizing a lexicon: as a flat structure and as a tree structure.

1.3.2.1 Flat-Lexicon

The term flat lexicon or linear lexicon denotes the fact that the words are kept strictly separate in the recognition process, that is, the matching between a given sequence of observations of unknown class and each word model is calculated independently. Word models are built a priori by concatenating the sub-word units or characters and further the matching between the sequence of observations and each model is

computed. So, the complexity increases linearly with the length and the number of words in the vocabulary.

1.3.2.2 Lexical Tree

Organizing the lexicon to be searched as a character tree instead of a linear structure of independent words has some advantages. This structure is referred to as lexical tree, tree-structured lexicon, or lexicon trie. If the spellings of two or more words contain the same initial characters, in the lexical tree they will share this sequence of characters. If the search strategy exploits adequately the shared parts, the repeated computation of the shared parts can be avoided, reducing the complexity of the recognition. It can be viewed as a reduction of the average word length that is given as:

$$L' = \frac{L}{rf} \quad (1.4)$$

where L' is the new average word length and rf is the reduction factor that is given as:

$$rf = \frac{Ncl}{Nct} \quad (1.5)$$

where Ncl is the total number of characters in the flat lexicon and Nct is the total number of characters in tree structured lexicon.

It is clear that as more words in the lexicon have common prefixes, more advantageous it will be to use a lexical tree. This is more likely to happen when large vocabularies are used. Figure 8 shows the average number of characters for both linear and tree-structured lexicon, for different vocabulary sizes. However, the search

technique must be adapted to exploit such shared parts and avoid unnecessary computation. The matching scores must be computed at the character level and they must be retained during the search to be used further by other words with similar prefixes.

Many authors have used tree-structured lexicons both in off-line [7, 20, 33, 41, 45, 71, 85, 86, 90, 113] and on-line handwriting recognition [45, 68, 117, 142]. An example is the work of Chen et al. [20]. They have presented an algorithm for lexicon-driven handwritten word recognition where word images are represented by segmentation graphs and the lexicon is represented as a tree. The proposed approach saves about 48% and 15% of computation time from the standard DP algorithm when static and dynamic lexicons are used respectively.

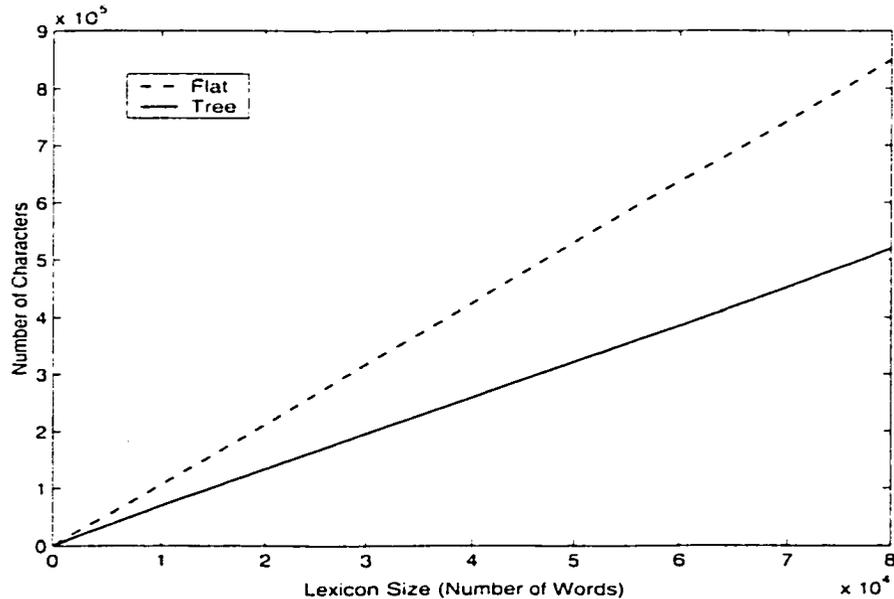


Figure 8 Number of characters for different lexicon sizes generated from an 85.1k-entry lexicon and organized as a flat structure and as a tree structure

1.3.2.3 Other Approaches

Other sorts of lexicon organization have also been proposed, such as the one based on a self-organizing feature map [122] and on n -gram analysis [162]. The former maps the dictionary (build word clusters) in a two-dimensional space in an unsupervised way, preserving neighborhood relationships. Results on a lexicon of 1,125 words show that the number of lexicon entries tested decreases by using the proposed mapping, however, the effects of the lexicon reduction on the coverage and recognition accuracy are overlooked. Procter et al. [138] incorporate a lexicon directly with the search procedure, but instead of searching through all the lexicon entries, they use a list derived from the lexicon with the models that may legally follow the preceding recognized subsequence. Another way to represent the search space is by a Direct Acyclic Word Graph (DAWG) which shares not only the prefixes, but also other common parts of the words such as terminations [61, 104]. However, it is not clear how such a technique can be used to the problem of large vocabulary handwriting recognition and further investigation is necessary.

1.3.3 Search Techniques

Generally, the matching between the test and the reference patterns through a decoding algorithm is the most time-consuming procedure in a large vocabulary handwriting recognition system. The motivation of investigating search strategies comes from the fact that the majority of the current search techniques used in handwriting recognition are based on expensive DP methods originally designed for small and medium-scale problems. When the recognition task is extended from a 100-word vocabulary to a 10,000-word vocabulary, the number of hypotheses to be searched blows up and these techniques can no longer handle efficiently such a large space. In handwriting recognition, the aspect of speed has been neglected because most of the researchers focus on small and medium vocabularies, where this aspect is not so

important². For this reason it is difficult to find a paper that reports the processing time together with the accuracy. Recognition rate has been the sole parameter to evaluate the proposed systems.

Researchers in speech recognition have devoted much more attention to large vocabularies because they reflect the real-life problems. So, they have developed alternative solutions to search in large vocabularies. While in speech recognition, throwing away some accuracy while improving the speed and reducing the memory usage is widely accepted; the researchers in handwriting recognition have been stricter maybe because they are still focusing on improving the accuracy of small-scale problems. Most of the search techniques in handwriting recognition are inherited from speech recognition. However, in spite of the similarities between the recognition tasks, the inputs are quite different. While high level features that yield to sequence of observations that are considerably short (e.g. 40-60 observations for a 13-character word) can be extracted from handwritten words, the speech waveform is converted to a sequence of acoustic vectors representing a smoothed log spectrum computed every 10 ms. A number of additional transformations are applied in order to generate the final acoustic vector [173] that usually has hundreds of observations. Furthermore, phone models are usually modeled by 3 to 5-state HMMs while the models used in handwriting recognition can be based on structural assumptions and easily have a high number of states (more than 10 states) [35, 71, 147].

To solve the handwriting recognition problem we have to resolve the following sub-problems:

- The number of characters L that are encoded in the sequence of observations is usually not known (although it is possible to estimate it);

²Depending on the constraints and experimental conditions, many small and medium vocabulary handwriting recognition systems are able to recognize words on personal computers in milliseconds.

- The character boundaries within the word are not known (except the beginning of the first character and the end of the last character in the word):
- For a set of sub-word reference patterns $\mathcal{C} = \{\lambda_1, \lambda_2, \dots, \lambda_M\}$ in which M is the number of reference patterns and λ_m represents the m -th pattern, and for a given value of L , there are M^L possible combinations of composite matching patterns (although this problem can be solved easily with a lexicon).

Large vocabulary handwriting recognition is a search problem that can be formulated as: select a word from the lexicon with the highest score, given a test pattern corresponding to an unknown handwritten word represented as a sequence of observations³ O , and a set of character reference patterns denoted as \mathcal{C} . Word reference patterns are formed by the concatenation of character reference patterns as $R_w = (\lambda_1^r \lambda_2^r \dots \lambda_L^r)$. However, characters usually are modeled by many parameters as $\lambda_l = (f_1^l f_2^l \dots f_N^l)$ in which N is the number of parameters in the character models and f_n represents the n -th parameter. Next, we provide a description of the search methods and techniques that have been used in large vocabulary handwriting recognition.

1.3.3.1 Dynamic-Programming Matching

Dynamic programming (DP) methods are based on the principle of optimality and are the most used search strategy both in speech and handwriting recognition. Depending on how the reference and test patterns are represented, distances or probability scores can be evaluated. DP methods compute the distance between a sequence of observations generated from the test pattern and all possible words in a lexicon. Each reference pattern is computed recursively by Equation 1.1 allowing the optimal path search to be conducted incrementally, in a progressive manner. Although there are P possible moves that end at point l , the optimality principle indicates that only

³Alternatively, the test pattern can be defined as a sequence of primitive segments of the image such that $S = (s_1 s_2 \dots s_P)$ in which P is the length of the sequence and s_p is the p -th primitive.

the best move is necessary to be considered. At the end, the best word hypotheses are those that have the minimum score with respect to the test pattern [112]. However, DP methods are mostly used with small and medium lexicons [47, 75, 82, 155], since they perform a non-exhaustive but still expensive search procedure.

The computational complexity of the recognition process, that is, the number of basic mathematical operations (additions, multiplications, and divisions) is $\mathcal{O}(N^2TVL)$ where T is the dimension of the sequence of observations (or primitives), V is the vocabulary size, L is the dimension of the words in the vocabulary, and assuming character reference patterns with dimension N . This is a rough approximation considering that each character has only one character reference pattern. This may be true if we consider only one type of handwriting style, e.g. handprinted words. However, in the unconstrained handwritten case, more than one reference pattern per character is usually necessary because a single one is not enough to model the high variability and ambiguity of the human handwriting.

1.3.3.1.1 Viterbi Search

The Viterbi algorithm is actually the same as the DP algorithm except that the matching between the test and reference patterns is computed in the HMM rather than the distance measure between primitives. Viterbi search is mostly used when the reference patterns are represented by statistical models and it has been used widely in handwriting recognition [15, 23, 35, 90, 123, 147]. Viterbi search belongs to a class of breadth-first search techniques where all hypotheses are pursued in parallel. It exploits the time invariance of the probabilities to reduce the complexity of the problem by avoiding the necessity for examining every route through the trellis. This procedure is known as time synchronous Viterbi search because it completely processes at frame t before going into the frame $t + 1$. At the end, a backtracking pass gives the required state sequences. The performance of the

conventional Viterbi algorithm in terms of recognition accuracy and speed is reported in some references [68, 86, 87, 90].

The Viterbi algorithm provides a computationally efficient way of analyzing observations of HMMs that exploits recursion to reduce computational load, but its computational complexity is comparable to other DP methods, that is, $\mathcal{O}(N^2TVL)$. More details about character and word modeling with HMMs and the use of the Viterbi algorithm in handwriting recognition are given in Chapter 4.

1.3.3.2 Beam Search

In many applications, a complete Viterbi search is impractical due to the large size of the state space. Many variations of the Viterbi search have been proposed to improve its performance.

Instead of retaining all L candidates at every frame, a threshold can be used to consider only a group of likely candidates. The state with the highest probability can be found first and each state with smaller probability than the highest one can then be discarded from further consideration. This is the *beam search* algorithm which can lead to substantial savings in computation with little loss of accuracy. Referring to the Equation 1.1, it consists in setting pruning beams based on the best hypothesis score at l denoted as $\varphi_l^*(i, n)$. So, all points with scores lower than $\varphi_l^*(i, n) + beam$ are activated at $l + 1$ and expanded, while all other points are pruned. In Figure 6 that corresponds to instead of keeping all candidates at every frame (e.g. x_1, x_2, \dots, x_L), only the k candidates with the highest scores are allowed to remain (e.g. $x_1, x_3, x_8, \dots, x_k$, where $k < L$). The problem is that the beam size is determined empirically and sometimes can throw away the optimal solution.

The idea of the beam search can be expanded to the character [67] and word levels. Since word recognition must evaluate the matching between the test pattern and

all possible reference patterns, the threshold can be derived from the final scores of words previously decoded, to discard from further consideration words in which the partial match falls below such a threshold.

Bippus et al. [7] presented a scheme to reduce the computational load based on a lexical tree Viterbi beam search. However, for a task of recognizing German city names with lexicon sizes between 150 and 400 words, the proposed approach resulted in an inefficient search. Manke et al. [117] presented a fast search technique for large vocabulary on-line handwriting recognition that combines a tree representation of the vocabulary, with efficient pruning techniques to reduce the search space without losing much recognition performance compared to a flat exhaustive search. Dolfing [29] uses the same technique to on-line recognition of handwritten words. Jaeger et al. [68] report a speedup factor of 10 over conventional Viterbi search in on-line handwriting recognition.

Another sort of beam search is proposed by Favata [40] which is a compromise between exhaustive search and DP methods. The idea is not to speed up the recognition process, but to improve accuracy by carrying the k best partial matches forward by using a queue structure to hold each partial match. The proposed algorithm takes each of the current k matches and expands them to find the next incremental match between a character and segment. In Figure 6 it corresponds to expand for example not only the best path that reaches the node x_1 , but the k -best. This procedure is more complex than the conventional DP, because instead of expanding only the L best matches at every frame, it expands the k best matches where $k > L$.

The computational complexity of the Beam search can be approximated by $\mathcal{O}(N'^2TVL)$, where $N' < N$ is the reduced number of states resulting from the beam heuristics.

1.3.3.3 A*

The A* algorithm belongs to a class of depth-first or best-first search techniques where the most promising hypothesis is pursued until the end of the observation sequence is reached [11]. The A* requires an evaluation function to compare hypotheses of different lengths and this is one of the key problems in heuristic search. For example, stack decoding is a variant of the heuristic A* search based on the forward algorithm, where the evaluation function is based on the forward probability.

The search begins by adding all possible word prefixes to the stack. Then, the best hypothesis is removed from the stack and all paths from it are extended, evaluated, and placed back in the stack. This search continues until a complete path that is guaranteed to be better than all paths in the stack has been found. A pruning mechanism can be used to save only a fixed number of hypotheses in the stack.

Bozinovic and Srihari [11] applied an A* search in off-line handwritten word recognition to match sequence of features with words in a lexicon. Hypotheses are generated from the matching between prefixes of lexicon words and sequence of features. For each hypothesis a score is assigned and at each step, the current best hypothesis is expanded and the list of hypotheses is resorted. The resulting list undergoes to a lexicon lookup where inadmissible hypotheses are discarded and the hypothesis with the best score continues to be expanded. Fujisaki et al. [45] have also used a decoder based on the A* search algorithm to on-line unconstrained handwriting recognition.

1.3.3.4 Multi-Pass

Multi-pass search algorithms (also called fast-match) employ a coarse-to-fine strategy, where computationally inexpensive models are initially used to produce a list of likely word hypotheses that are later refined using more detailed and computa-

tionally demanding models [21]. Ratzlaff et al. [142] have used this search strategy in the recognition of unconstrained on-line handwritten text, and Bazzi et al. [4] in an open-vocabulary OCR system. The computational complexity of the first pass of such an approach can be approximated by $\mathcal{O}(N'TVL)$, where $N' < N$ and N' is the dimension of such a computationally inexpensive model. At the end to this first pass, only V' word hypotheses are selected, where $V' \ll V$. These word hypotheses represent the reduced vocabulary that is used in the second stage with more complex models N . So, the computational complexity of the second pass is $\mathcal{O}(NTV'L)$.

To be efficient the combined computational complexity $\mathcal{O}(N'TVL + NTV'L)$ must be lower than the conventional DP method. However, there is a risk of reducing the accuracy due to the use of heuristics and coarse models at the first pass.

Another example of multi-pass search is the forward-backward which uses an approximate time-synchronous search in the forward direction to facilitate a more complex and expensive search in the backward direction [21].

1.4 Verification and Post-Processing in Handwriting Recognition

There exists many different ways to improve accuracy of a handwriting recognition system, such as optimization of the feature set, improvements on the modeling of words and characters, improvements of classifiers and classification methods, combination of different classifiers, etc. Since the approach proposed in this thesis is based on the recognition of isolated handwritten characters, first we present the main investigations on this topic and next we address the main topics related to the verification of handwritten words.

1.4.1 Isolated Handwritten Character Recognition

The recognition of handwritten characters is very challenging and it has been the subject of much attention in the field of handwriting recognition. Many proposals

to solve this problem have been presented throughout the last decade [9, 18, 25, 30, 46, 65, 81, 95, 129, 134, 153, 168, 172, 175]. However, most of the research efforts in recognition of characters have been focused on the recognition of digits [14, 60, 100, 131]. In fact, digit recognition is just a subset, for which the solutions are more simple and robust. But when we talk about the recognition of alphabetic characters, this problem becomes more complicated. The most obvious difference is the number of classes that can be up to 52, depending if uppercase (A-Z) and lowercase (a-z) characters are distinguished from each other. Consequently, there are a larger number of ambiguous alphabetic characters other than numerals. Character recognition is further complicated by other differences such as multiple patterns to represent a single character, cursive representation of letters, and the number of disconnected and multi-stroke characters.

The problem of recognition of isolated handwritten characters is quite simple if compared with handwritten words because the characters are already isolated and there is no need to perform a segmentation. Furthermore, the elements to be recognized are supposed to be entire characters and not fragments or connected shapes. So, the solutions to the recognition problem are more flexible and many different recognition strategies and classifiers can be used. Several strategies have been adopted by researchers to recognize isolated characters and digits. We can distinguish between two main classes of classifiers: statistical and neural classifiers. Furthermore, some other approaches use hybrid strategies that combine different types of classifiers.

1.4.1.1 Neural Network Classifiers

Neural networks classifiers (NN) have been used extensively in character recognition [9, 25, 30, 44, 46, 47, 81, 134, 176]. These networks can be used as a combined feature extractor and classifier, where the inputs are scaled or sub-sampled input image, or as a “pure” classifier where the inputs are extracted features. One problem of using

neural networks in character recognition is that it is difficult to analyze and fully understand decision-making process.

Many recognition systems are based on multilayer perceptrons (MLPs) [9, 30, 46, 47, 134]. Gader et al. [46] describe an algorithm for handprinted word recognition that uses two 27-output-4-layer backpropagation networks, one for uppercase and the other for lowercase characters, to account for the cavity features and other two 27-output-4-layer backpropagation networks to account for the direction-value features. Cavities are computed using mathematical morphology and yields up to a 105-dimensional vector. Direction-value features are derived from zones using boundary and skeletal pixels, and yield up to a 60-dimensional feature vector. Further, the outputs of the networks are combined to optimize recognition accuracy. Recognition rates of 94% and 82% were achieved for uppercase and lowercase characters respectively. A similar approach also using four neural networks to account for uppercase and lowercase characters has been proposed in [47]. A different feature set formed by bar features and transition features is used. Recognition rates of 86.24% for uppercase characters and 83.45% for lowercase characters are reported. Instead of using separated networks to account for uppercase and lowercase characters, Blumenstein et al. [9] have used a 52-output neural network representing 26 uppercase and 26 lowercase characters. Case sensitive and case non-sensitive experiments were conducted and the recognition accuracy attained was almost 60%. Dong et al. [30] have presented a local learning framework consisting of quantization and ensemble layer for recognition of lowercase handwritten characters. A 160-dimensional feature vector is built from 32×32 normalize character images and directional features based on the gradient. The quantization layer uses learning vector quantization (LVQ) to partition the pattern space into clusters or subspaces. In the ensemble layer, MLPs are used as local learning machines. Recognition rate of 92.34% on a cleaned set of NIST database is reported. Pedrazzi and Colla [134] present a simple feature set for handprinted character representation build by the combination of average

pixel density and measures of local alignment along some directions. Classification is carried out by a MLP and recognition rates of 96.08% and 87.40% are achieved for uppercase and lowercase characters respectively.

Other types of neural networks have also been used [25, 176]. Zhang et al. [176] have applied an adaptive-subspace self organizing map (ASSOM) to classify handwritten digits. Chim et al. [25] present a dual classifier handprinted character recognition system that uses radial basis function (RBF) networks. Image projections and four-directional edge maps are extracted from images and used as feature vectors. These features are used in two independent classifiers from which the outputs are further for a final decision. Recognition rates over 96% for the combination of both classifiers are reported.

These are just some few examples of the use of neural networks in isolated character recognition. Table V shows the performance of these and other recognition systems.

1.4.1.2 Statistical Classifiers

Stochastic modeling is a flexible and general method for modeling handwritten characters and entails the use of probabilistic models with uncertain or incomplete information. Some strategies that employ such kind of classifiers can be found in [65, 80, 133, 168].

Kim et al. [80] presented a recognition system of constrained handwritten Hangul and alphanumeric characters using discrete HMMs. Park et al. [133] present a scheme for off-line recognition of large set handwritten characters in the framework of first order HMMs. Wang et al. [168] present new techniques using SVMs for the recognition of Western handwritten capitals. The performance of two SVM classifiers is compared with 1-NN, k -NN, HMM and MLP classifiers. The recognition rates achieved show that the SVM classifier outperforms other methods, however, they

are considerably slower both for training and recognition tasks. Heutte et al. [65] describe the combination of structural/statistical feature based vector and a linear discriminant classifier to the recognition of digits and uppercase characters. Global features such as invariant moments, projections and profiles and local features such as intersection with straight lines, holes and concave arcs, extrema and junctions were used to build a 124-dimensional feature vector. Recognition rates over 97% have been reported.

1.4.1.3 Other Recognition Strategies

Other strategies to recognize isolated handwritten characters use hybrid schemes that combine different types of classifiers such as neural and statistical classifiers to build more robust classifiers.

Kimura et al. [81] used two different character classifiers: one statistical classifier that calculates the likelihood of any character using a modified quadratic discriminant function and other based on a 3-layer backpropagation neural network with a 52-output layer to account for the 52 letters of alphabet. Fu et al. [44] presented a Bayesian decision-based neural network for multilingual handwritten character recognition that adopts a hierarchical network structure with nonlinear basis functions and a competitive credit-assignment scheme. This modular NN deploys one subnet to take care one object and it is able to approximate the decision region of each class locally and precisely. Such kind of scheme has the merits of both neural networks and statistical approaches. Kato et al. [72] presented a system for Chinese and Japanese character recognition that uses city block distance with deviation and asymmetric Mahalanobis distance for rough and fine classification respectively. Lazzerini and Marcelloni [95] presented a linguistic fuzzy recognizer of off-line isolated lowercase handwritten characters where the shape of each character is described by linguistic expressions derived from a fuzzy partition of the character

image. Recognition rate of 69.5% is reported for a small-scale problem. Camastra and Vinciarelli [18] presented a cursive character recognizer composed of feature extraction and learning vector quantization where three consecutive learning techniques: NN decision rule, pairwise adjustment of the codevectors, and other set of rules are used to approximate the codebook to the class distribution. A recognition rate of 81.72% is reported for lowercase characters of NIST database.

Table V presents some types of classifiers and features used in handwritten character recognition. Table VI presents a summary of the performance of isolated handwritten character recognition methods. The character recognition rates are presented for case sensitive and case non-sensitive. Some authors give separated recognition rates for uppercase and lowercase characters. It should be stressed that these studies have used different datasets and experimental conditions, which makes a direct comparison of the results very difficult.

Table V

Type of classifiers and features used in handwritten character recognition

Author	Classifiers	Features
Heutte et al. [65]	Linear discriminative functions	Invariant moments, projections, profiles Intersection, holes, concave arcs extrema, end points, junctions
Takahashi et al. [163]	NN	Contour direction, bending point
Gader et al. [46]	Multilayer feedforward NN	Cavities, direction-value
Gader et al. [47]	Multilayer feedforward NN	Bar and transition
Wang et al. [168]	SVM	Raw bitmap
Oh et al. [129]	Modular	Distance transformation directional distance distribution
Guillevic et al. [59]	k -NN Classifiers	Pixel distance metric
Chim et al. [25]	RBF networks	Projections and edge maps

Table VI

Summary of results in isolated handwritten character recognition

Reference	Year	Datasets			Recognition Rate (%)		
					Case Sensitive		Case Insensitive
		Database	Training	Test	Uppercase	Lowercase	
Kimura [82]	93	CEDAR	22,606	—	60.83		69.53
		CEDAR	34,206	—	61.30		70.03
					69.45		77.57
Yamada [172]	96	CEDAR	20,800	1,219	67.8		75.7
Kimura [81]	97	USPS	34,139	9,539	76.31		83.87
Blumenstein [9]	99	CEDAR	15,297	1,212	56.11		58.50
Camastra [18]	01	CEDAR	32,426	16,213	—	—	81.72
Pedrazzi [134]	95	NIST	29,969	11,941	96.08	87.40	—
Gader [47]	97	USPS	13,000	13,000	86.24	83.45	—
Gader [46]	95	NIST	13,000	13,000	94	82	—
Heutte [65]	98	USPS	12,765	3,195	97.27	—	—
Chim [25]	98	DBP	1,860	620	97.42	—	—
Wang [168]	00	UNIPEN	5,200	9,767	81.2	—	—
Takahashi [163]	93	NIST	42,176	11,941	94.6	—	—
		NIST			95.8	—	—
Drucker [31]	93	NIST	10,000	2,000	96.0	—	—
					97.6	—	—
Schwenk [150]	96	NIST	26,969	17,982	97.45	—	—
Oh [129]	98	NIST	26,000	11,941	90.06	—	—
Srihari [159]	93	NIST	10,134	877	—	85	—
Matsui [121]	94	NIST	10,968	8,284	—	89.64	—
Lazzerini [95]	00	NIST	1,560	1,040	—	69.5	—
Dong [30]	01	NIST	23,937	10,688	—	92.34	—
		CENPARMI	14,810	8,580	—	89.30	—

1.4.2 Verification in Handwriting Recognition

The last aspect that we review is the verification in handwriting recognition. Our particular interest is on the verification of handwritten words. While the concept of verification has been adopted in other related domains, it is a relatively new subject in handwriting recognition [14, 26, 59, 110, 111, 131, 136, 157, 163, 178]. So, many of the results that we review in this section come from related domains, such as verification of handwritten characters [26, 163] and digits [14, 131, 178], and verification of spoken utterances [3]. Verification of handwritten words and phrases are presented in [59, 110, 111, 136, 157].

Takahashi and Griffin [163] are among the earliest to mention the concept of verification in handwriting recognition. They describe a linear tournament verification for handwritten characters. The classifier is based on a MLP and a 184-dimensional feature vector made up by the combination of zonal (contour direction) and geometrical features (bending points), and recognition rate of 94.6% is achieved for uppercase characters of NIST database. Based on an error analysis the authors present a verification by linear tournament with one-to-one verifiers between two categories. With such a verification scheme, the recognition rate was increased by 1.2%. Zhou et al. [178] describe a system for recognition of handwritten numerals that uses a verifier to overcome some problems of confused digits. Improvements in the recognition rate of about 6% are achieved with the use of the verifier. Britto et al. [14] have presented a two-stage HMM-based system for recognizing handwritten numeral strings. The first stage finds the N -best segmentation-recognition paths for a numeral string. Further, these paths are re-ranked by the verification stage. The verification stage is composed of 20 numeral HMMs where 10 are based on the image columns and the other 10 are based on image rows. A global improvement of about 10% (from 81.65% to 91.57%) in the recognition rate is reported after the verification stage. Oliveira et al. [131] present a recognition and verification strat-

egy for automatic recognition of handwritten numerical strings. The verification module contains two verifiers to deal with the problems of oversegmentation and undersegmentation. An improvement of 5–12% is reported when both verifiers are employed. Cho et al. [26] present a neural network based verification module in an HMM-based on-line handwriting recognition system that penalizes unreasonable grapheme hypotheses. The verifier is incorporated to the Viterbi search algorithm and it takes as an input the grapheme hypothesis generated by the HMM and outputs *a posteriori* probability as its validity. An improvement of about 4% in the recognition of Korean characters is achieved.

In speech recognition, verification schemes have also been proposed. Austin et al. [3] present a segmental neural net (SNN) for phonetic modeling in continuous speech recognition that is used to score frames of a phonetic segment provided by an HMM. A spoken utterance is processed by the HMM recognizer to produce a list of the N -best scoring sentences hypotheses. Thereafter, the recognition task is reduced to select the best hypothesis from the N -best list. The author use the SNN to generate a score for the hypothesis as the logarithm of the product of the appropriate SNN scores outputs for all the segments. Such a scheme reduced the error rate from 4.1% to 3.0%.

Madhvanath et al. [110,111] describe a system for rapid verification of unconstrained off-line handwritten phrases using perceptual holistic features. Having as input a binary image of a handwritten phrase and a verification lexicon containing one or more ASCII verification strings, holistic features are predicted from the verification ASCII strings and matched with the feature candidates extracted from the binary image. The system rejects errors with 98% accuracy at a 30% accept level. The verification scheme cannot handle handprinted phrases in either upper or mixed case, as well as words that are not well written. Srihari [157] presents several methods for classifier combination in a handwritten phrase recognition system. Two handwritten

word classifiers are combined to maximize the accept rate while keeping error rate and the average processing cost within acceptable limits. A lexicon-driven word model recognizer provides a list of lexicon entries ordered according to the goodness of match that is obtained by matching a sequence of features extracted from the word images by dynamic programming. A character model recognizer that attempts to isolate and recognize each character in the word provides the possible choices of characters with the respective confidence values. The two classifiers are combined in a hierarchical strategy. The character model recognizer is called only if a decision cannot be made based on the results of the word model recognizer. Classifier decisions are combined only if a decision cannot be made based on the results of the character model recognizer.

Powalka et al. [136] present a hybrid recognition system for on-line handwritten word recognition where letter verification is introduced to improve disambiguation among the resulting word alternatives. Letter alternatives located within the words are used by the letter verification as compound features. A multiple interactive segmentation process identifies parts of the input data which can potentially be letters. Each potential letter is recognized and after they are concatenated to form strings. A lexicon is used to eliminate invalid words. The letter verification produces a list of word alternatives that are constrained to be the same words provided by an holistic word recognizer. Scores of the word recognizer and letter verification are integrated into a single score using a weighted arithmetic average. An improvement of 5–12% is reported for the top word hypothesis. Guillevic [59] has presented a verification scheme at character level for handwritten words from a restricted lexicon of legal amount of bank checks. Only the first and the last characters of the words are verified using two k -NN classifiers, one for uppercase character and other for lowercase characters. There is a limited number of possible character classes that may appear at the first and last positions of words for his specific lexicon (9 for first and 9 for last position). For a given character, both the lowercase and the

uppercase classifiers are called and the output results from the classifier with the lowest distance score. The results of the character recognition are integrated with a word recognition module to shift up and down the word hypotheses.

1.5 Summary

In this chapter we have presented the recognition strategies commonly used in handwriting recognition as well as the importance of using a language model in the recognition process. We have also reviewed a number of works in handwriting recognition focusing on three main subjects: large vocabulary handwriting recognition, isolated handwritten character recognition, and verification in handwriting recognition.

We have seen that most of the methods and strategies proposed in large vocabulary handwriting recognition system attempt to circumvent the problem by reducing the size of the vocabulary prior to the recognition process. This reduction allows the utilization of conventional techniques currently used with small and medium vocabulary recognition tasks. However, the improvements in recognition speed brings also reduction in recognition accuracy. The same is valid for the methods that attempt to reduce the search effort by introducing heuristics into the search algorithms.

Following the works in large vocabulary, we have presented some relevant works on recognition of isolated handwritten characters. A diversity of classification methods and features can be used to tackle this problem with relative success. However, most of them deal with either uppercase characters or lowercase characters. Very few works have considered unconstrained handwritten characters which is a more difficult problem.

Finally, the last aspect that we have reviewed was the verification in handwriting recognition. Although this aspect is relatively new, some works have employed verification modules to overcome problems of classifier and improve the performance

in terms of accuracy and reliability.

In the next chapter we will discuss more extensively the methods and techniques proposed in this chapter to large vocabulary handwriting recognition and verification in handwriting recognition. Based on such a discussion the goals of our research will be established more precisely.

CHAPTER 2

PROBLEM STATEMENT

Most of the research effort on handwriting recognition has primarily been in improving recognition accuracy of systems where the restricted size of the vocabulary and specific application environments have been the most common constraints. We have seen in the previous chapter that current recognition systems are several tens of times too slow as well as much less accurate for large vocabulary tasks. In order to be practically useful, handwriting recognition systems have to be efficient in their usage of computational resources and deliver reasonable recognition accuracy as well.

One goal of this thesis is to demonstrate that it is possible to achieve high recognition speed on large vocabulary handwriting recognition without compromising the recognition accuracy offered by the baseline recognition system. Notwithstanding the aimed improvement in recognition speed, another goal of this research is to demonstrate that it is possible to achieve better recognition accuracy without compromising the gains in recognition speed.

In this chapter we discuss the problems that have to be solved to build up a large vocabulary handwriting recognition system. In Section 2.1 we analyze the effects of an increasing vocabulary in both recognition accuracy and recognition speed. This section also presents a discussion about the computational complexity of the recognition process and an analysis of the methods and strategies that have been proposed to deal with the problem of recognition speed in large vocabularies. In Section 2.2 we discuss many possible issues to improve the recognition accuracy of handwriting recognition systems, but focusing on the concept of post-processing. The relevance of having handwriting recognition systems capable to deal with large

vocabularies is presented in Section 2.4 where many applications that require large vocabularies are presented. This chapter concludes with Section 2.5 that gives a more precise overview of our research hypotheses and the goals we intend to achieve.

2.1 Large Vocabulary Problems

There are two main problems that have to be solved to build up a large vocabulary handwriting recognition system: the first one is related to the recognition speed while the second is related to the recognition accuracy. However, these two aspects of performance are in mutual conflict. It is relatively easy to improve recognition speed while trading away some accuracy and vice versa.

Recognition speed is not a serious issue in the case of small and medium vocabulary tasks of a few tens or hundreds of words. They are dominated by concerns of achieving high accuracy. But as the size of the vocabulary increases, the recognition speed does become an issue. Figures 9 and 10 clearly illustrate the effects of an increasing vocabulary size on both recognition accuracy and recognition speed respectively for a baseline recognition system that will be presented in Chapter 3. While the recognition rate is as high as 90% for vocabularies of hundreds of words, it drops to less than 80% for vocabularies of ten thousands of words. The negative effects are also observed on the recognition speed. While the recognition time is on the order of tens of seconds for small vocabularies, it increases to few minutes for very large vocabularies.

The problems related to the accuracy are common to small and medium vocabularies. However the task of recognizing words from a small vocabulary is much easier than from a large lexicon (where more words are likely to be similar to each other). With an increasing number of word candidates, the ambiguity increases due to the presence of more similar words in the vocabulary and that causes more confusion to the classifiers. A common behaviour of the actual systems is that the accuracy

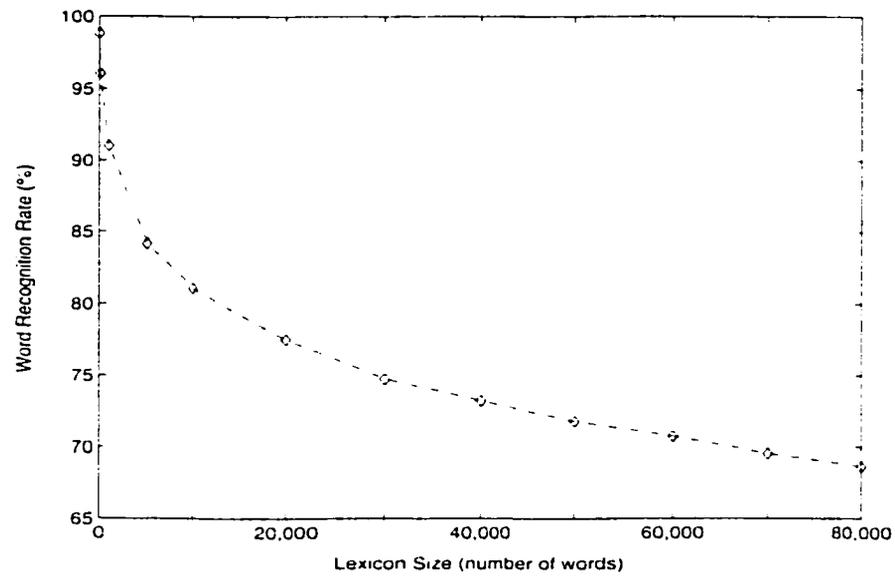


Figure 9 Reduction of the recognition rate with an increasing number of words in the lexicon [87]

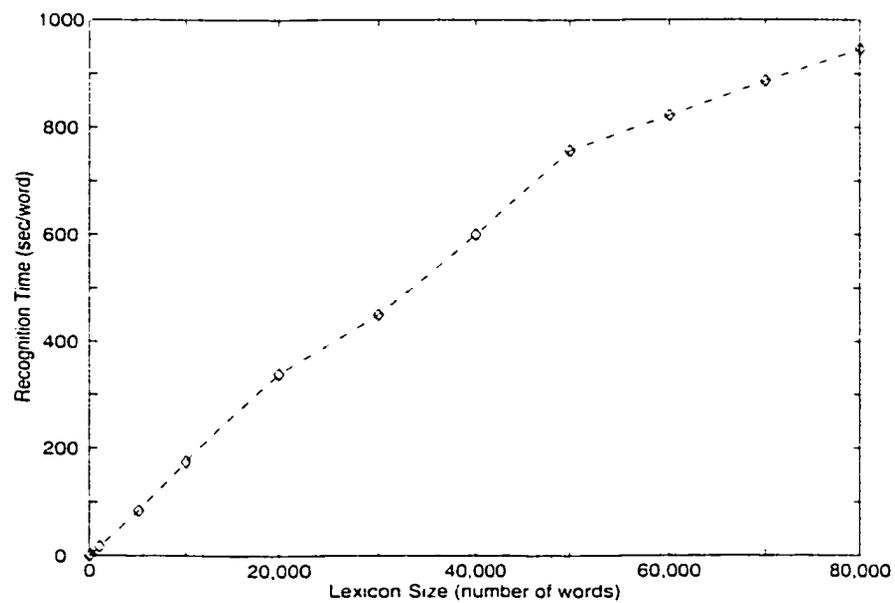


Figure 10 Increasing of the recognition time with an increasing number of words in the lexicon [87]

decreases as the number of words in the lexicon grows (Figure 9). However, there is not a clear relation between these two factors. It depends on the particular characteristics of the systems as well as on the nature of the words in the vocabulary. Figure 11 shows an example of words taken from a real lexicon of French city names where some of them differ only by one or two characters.

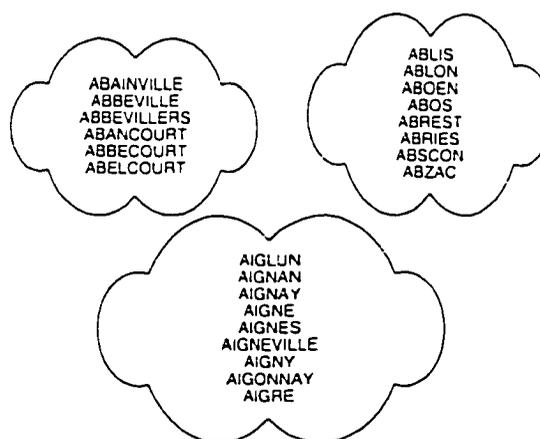


Figure 11 Example of similar words present in a lexicon of French city names

Most of the problems with computational complexity and recognition time in handwriting recognition arise from the fact that most of the current recognition systems rely on very time-consuming search algorithms such as standard dynamic programming, Viterbi, or forward algorithms. However, the speed aspect has not been considered by many researchers in handwriting recognition, mainly because they have not been dealing with large vocabulary problems. This aspect is overlooked in small and medium vocabularies because typical recognition times are in the order of milliseconds [85].

2.1.1 The Complexity of Handwriting Recognition

One of the most important aspect in large vocabulary handwriting recognition is the computational complexity of the recognition process. This aspect is usually

overlooked in applications that deal with smaller vocabularies. It is worthwhile to identify the elements responsible for the computational complexity of a handwriting recognition system. Indeed, the complexity of the recognition is strongly dependent on the representation used for each of the elements (e.g. reference pattern, input pattern, language model, etc) as well as on the algorithm used to decoding. Recalling that the basic problem in handwriting recognition is given an input pattern represented by a sequence of observations (or primitives) O and a recognition vocabulary represented by \mathcal{R} , find the word $w \in \mathcal{R}$ that best matches the input pattern. In describing the computational complexity of the recognition process, we are interested in the number of basic mathematical operations, denoted as \mathcal{O} , such as additions, multiplications, and divisions it requires. The computational complexity for a generic recognition process, denoted as \mathcal{C} , can be approximated by:

$$\mathcal{C} = \mathcal{O}(\mathbb{T}\mathbb{V}\mathbb{L}\mathbb{M}) \quad (2.1)$$

where \mathbb{T} is the dimension of the observation sequence, \mathbb{V} is the size of vocabulary, \mathbb{L} is the average size (length) of the words in the vocabulary, and \mathbb{M} is the dimension of the sub-word models¹. This is a rough approximation considering that each word/character has only one model. This may be true if we consider only one type of handwriting style, e.g. handprinted words. However, if handwriting is unconstrained, more than one reference pattern per character is usually necessary because a single one is not enough to account for the high variability and ambiguity of the human handwriting. Assuming that each word is either entirely handprinted or cursive (Figure 12a) and that each character class has a cursive and a handprinted reference pattern, the computational complexity increases linearly as:

¹We assume that words are modeled by sub-word models since this is the case for most of the large vocabulary handwriting recognition systems.

$$\mathcal{C} = \mathcal{O}(\mathbb{H}\mathbb{T}\mathbb{V}\mathbb{L}\mathbb{M}) \quad (2.2)$$

where \mathbb{H} denotes the number of models per character class. However, if we assume that words can be made up by the combination of both writing styles, that is, a mixture of handprinted and cursive characters (Figure 12b), the computational complexity blows up exponentially as:

$$\mathcal{C} = \mathcal{O}(\mathbb{H}^2\mathbb{T}\mathbb{V}\mathbb{M}) \quad (2.3)$$

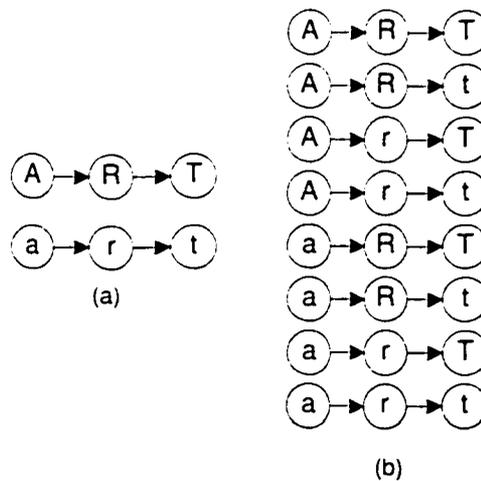


Figure 12 Possible representations of a word: (a) assuming only one writing style: entirely handprinted or cursive; (b) assuming all possible combinations of handprinted and cursive characters: mixed

To get a feeling for how impractical the computation of Equations 2.2 and 2.3 actually is, consider typical values of $\mathbb{H} = 2$ models per character class, $\mathbb{L} = 10$ characters per word, $\mathbb{M} = 10$ primitives or states, $\mathbb{V} = 80,000$ words, and $\mathbb{T} = 40$ observations. With these values we get $\mathcal{C} \approx \mathcal{O}(6.4 \cdot 10^8) - \mathcal{O}(3.27 \cdot 10^{10})$. This computation to rec-

ognize a single word is already excessive for most modern machines². In spite of the size of the vocabulary is only one of the several factors that contribute to the high computational complexity of the recognition process. it is the most important factor affecting the development of more general applications. Therefore, management of the complexities in large vocabulary handwriting recognition systems, especially in real-time applications, poses a serious challenge to the researchers.

2.1.2 Discussion on the Current Methods and Strategies for Large Vocabulary Handwriting Recognition

The methods presented in Chapter 1 attempt to prune the lexicon prior to the recognition to reduce the number of words to be decoded during the recognition process. The knowledge about the application environment is a very efficient approach to reduce the lexicon, since it does not incur any error, because the reduced lexicon contains only the words that the system has to recognize effectively. However, the other methods rely on heuristics and the lexicon is reduced at the expense of accuracy.

The approaches based on the estimation of the word length are very simple and they can also be efficient. However, as they depend on the nature of the lexicon, their use may be preceded by an analysis of the distribution of the length of the words. The same remark is valid for the methods based on the analysis of the word shape. The main drawback of these methods is that they depend on the writing style and they seem to be more adequate for cursive handwriting. Another remark is that some approaches involve the extraction of different features from those used to recognition from the word image. Moreover, the robustness of some methods had not been demonstrated in large databases and large lexicons and their influence on the recognition process as a whole have not been clearly stated.

²Current personal computers can perform between 1,000 and 3,000 million floating-point operations per second (MFLOPS).

In spite of the fact that larger lexicons may cause more confusion in the recognition process due to the presence of more similar words, reducing the lexicon by these proposed approaches implies a reduction in coverage, so the recognition accuracy also drops. It is easy to reduce the search space and improve the recognition speed by trading away some accuracy. It is much harder to improve recognition speed without losing some accuracy. The same conflict is observed in speech recognition systems [61, 74, 143, 145, 173].

The results presented in Tables II, III, and IV in Chapter 1 are not directly comparable but they only show the effectiveness of some of the proposed methods under very particular experimental conditions. There is a lack of information concerning the effects of the pruning methods on the recognition system. Aspects such as propagation of errors, selectiveness of writing styles, time spent to reduce the lexicon, etc. are usually overlooked. What is the computational cost of including lexicon reduction in the recognition process? How reliable are the lexicon reduction mechanisms, and what happens when they fail to reduce the number of lexicon entries? Most of the proposed approaches that we have presented in Chapter 1 overlooked these aspects.

On the other hand, the organization of the vocabulary seems to be a very effective approach specially for large vocabularies. For small to medium size vocabularies, it is quite reasonable to use linear lexicons [23, 35, 154]. However, for larger vocabularies, the search space blows up linearly as a function of the number of entries, and that requires a formidable effort to search the entire vocabulary during the recognition. The organization of the lexicon as a tree structure may minimize such a problem. However its effectiveness depends on the nature of the lexicon and an analysis of the number of common prefixes within words in the lexicon should be considered. Other methods still have to prove their effectiveness in maintaining or not the same coverage of the whole vocabulary.

Concerning the search strategies, the Viterbi algorithm and DP methods are the search strategies used more often in small and medium vocabulary applications. Although, calculating the probability in this manner is computationally expensive, particularly with large models or long observation sequences. For large vocabulary handwriting recognition, a complete DP search is clearly impractical. The other search techniques (A*, beam search, and multi-pass) have been used widely in speech recognition but not in handwriting recognition. Because they are faster, generally they are less accurate, providing sub-optimal solutions. Most of the research in handwriting recognition is focused on small and medium vocabulary problems that do not justify the use of less accurate search techniques. The Viterbi beam search has been the preferred choice of the researchers dealing with large vocabularies. The stack decoder suffers from problems of speed, size, accuracy and robustness. For example, a bad choice of heuristic can lead to an explosion of the effective search space and the possibility of repeated computation [143]. In fact, there is a lack of studies that compare the advantages and disadvantages of different search methods applied to handwriting recognition both in terms of accuracy and speed.

2.2 Problems of Recognition Accuracy

The main issue in improving the accuracy of a working handwriting recognition system is to preserve the computational complexity of the recognition process. Unfortunately, the complexity of the recognition process has been steadily increasing along with the recognition accuracy, more often than not at the expense of recognition speed. How useful might be a recognition system that needs 10 minutes to recognize a single word with recognition accuracy as high as 90% ?

For small vocabularies, the nature of the problem also allows the use of *ad hoc* techniques, such as HMM models for whole words instead of characters. As the vocabulary size increases, words tend to have more common spellings and to achieve

reasonable accuracy it is necessary to have more discriminant classifiers because the confusions are also increased. Usually, improvements in accuracy are associated with improvements in the feature set, better modeling of reference patterns, or by the combination of different feature vectors and/or classifiers. It has been shown that when a large amount of training data is available, the performance of a word recognizer generally can be improved by creating more than one model for each of the recognition units [28, 140] because it provides more accurate representation of the variants of handwriting. However, this can be one of the many possible ways to improve the accuracy of large vocabulary recognition systems. Furthermore, the use of contextual-dependent models may be another feasible solution. On the other hand, while multiple models may improve the accuracy, they also increase the computational complexity. Recently, methods based on combination of multiple classifiers have become very popular. The claim is that a single classification strategy is not able to account for the high variability of handwriting. The combination of different classifiers with different feature sets usually is very helpful to improve the accuracy in handwriting recognition systems.

These are some examples of the problems that have to be circumvented to improve the recognition accuracy. One of the main constraints is always the amount of data available for training. So we may conclude that to improve the recognition accuracy of a large vocabulary handwriting recognition system while preserving the recognition speed is an even harder challenge.

2.2.1 Error Analysis

Improvements in the recognition accuracy of a working system shall be preceded by an error analysis that first identifies the weaknesses to further carry out improvements that correct some of these errors. Recently, two distinct studies have been conducted to identify the sources of errors and confusions in the baseline SRTP

recognition system [38,54], which is described in Chapter 3.

Farouz [38] has found that many confusions arise from the features extracted from the loose-segmented characters. These features that are not able to give a good description of characters when oversegmentation is present, specially for cursive words. The reason for that is that the feature set used to characterize cursive words is based on topological information such as loops and crossings which are difficult to be detected on character fragments. According to him, these features only provide a good representation of words correctly segmented.

A summary of the main causes of recognition errors is given as follows.

- Presence of upper and lower guidelines on the word image;
- Underlined words;
- Problems with the pre-processing;
- Ambiguity due to the discriminant power of the features: fragments or pseudo-characters may be a source of confusion with whole characters;
- Errors of undersegmentation: the undersegmentation is not correctly modeled in the character models. It is partially modeled by the probability of character omission. However, the features resulting from a grouping of characters is supposed to be generated by undersegmented characters. Moreover, the features extracted from a group of connected characters creates a high ambiguity because it may be confused with other whole characters;
- Irregular behaviour of the oversegmentation: it arises from the irregular splitting of some characters that generate segments non useful to the recognition. This is the case of loops that are not completely closed by discontinuity on the tracing.

A similar error analysis conducted by Grandidier [54] but using a slightly different feature set that incorporates concavities features, has drawn similar conclusions

about the sources of errors and confusions. So, tackling one or more of the problems mentioned above might improve the recognition accuracy.

2.3 Other Related Problems

Recognition accuracy and recognition speed might be the main goals while developing a large vocabulary handwriting recognition system, however some other important issues exist:

- Memory Usage;
- Reliability;
- Out-of-Vocabulary Words.

Memory usage is actually not so important issue because most of the recognition strategies require only tens of megabytes of main memory which is found on most of the current personal computers. Furthermore, it is always possible to maintain on disk not frequent used information and use some caching strategy to avoid delays due to disk access.

Reliability is a very important issues, and it is related to the capability of the system to not accept false word hypothesis and to not reject true word hypothesis. The question is not only to find a word hypothesis, but the most important is to find out how trustworthy is the hypothesis provided by the handwriting recognition system. However, this problem may be regarded as difficult as the recognition itself is. For such an aim, rejection mechanisms are usually used to reject word hypotheses according to an established threshold. The aspect reliability is discussed later in Section 4.4.

Alternatively, it may also be interesting to incorporate to the recognition system some sort of mechanism to deal with words that do not take part of the recognition

vocabulary, either by identifying them as such or attempting to reject them or attempting to recognize them. However, in this thesis we are considering a large vocabulary where all possible words are present. So, out-of-vocabulary words are out of the scope of this thesis and no further attention will be given to such a topic.

2.4 Large Vocabulary Handwriting Recognition Applications

According to Plamondon and Srihari [135], the ultimate handwriting computer will have to process electronic handwriting in an unconstrained environment, deal with many handwriting styles and languages, work with arbitrary user-defined alphabets, and understand any handwritten message provided by any writer. So, it is unquestionable the importance of large vocabulary handwriting recognition techniques to reach some of these goals. The capability of dealing with large vocabularies, however, opens up many more applications.

Most of the actual research in handwriting recognition focuses on specific applications where the recognition vocabulary is relatively small. Clearly, the size of the vocabulary depends on the application environment. The larger the vocabulary is, the more flexible the application that utilizes it can be. More generic applications need to quickly access large vocabularies of several thousand words. For a general text transcription system, a lexicon of 60,000 words would cover 98% of occurrences [154]. Future applications [77, 120] have to be flexible enough to deal with dynamic lexicons and also words outside of the vocabulary. Typical applications that require large vocabularies are:

- Postal applications: recognition of postal addresses on envelopes (city names, street names, etc.) [5, 33, 35, 91];
- Reading of handwritten notes [120, 154];
- Fax transcription [37];

- Generic text transcription: recognition of totally unconstrained handwritten [77, 120];
- Information retrieval: retrieval of handwritten field from document images;
- Reading of handwritten fields in forms: census forms [114], tax forms [160], visa forms and other business forms;
- Pen-pad devices: recognition of words written on pen-pad devices [67, 171].

In postal applications, the potential vocabulary is large, containing all street, city, county and country names. One of the main reasons for using word recognition in address reading is to disambiguate confusions in reading the ZIP code [154]. If the ZIP code is reliably read, the city will be known, but if one or more digits are uncertain, the vocabulary will reflect this uncertainty and expand to include other city names with ZIP codes that match the digits that were reliably read. However, as pointed out by Gilloux [51], when the recognition of the ZIP code fails, the recognition task is turned into a large vocabulary problem where more than 100,000 words need to be handled. Other applications that require large vocabularies are reading handwritten phrases on census forms [114], reading names and addresses on tax forms [160], reading fields of insurance and healthcare forms and claims, and reading information from subscription forms and response cards. Figures 13 to 17 illustrate some of the possible applications of the large vocabulary handwriting recognition mentioned through this section.

2.5 A More Precise Overview of This Thesis

Considerable progress has been made in handwriting recognition technology over the last few years. Although many issues still need to be solved, especially those concerning large vocabulary applications since the capability of dealing with large vocabularies opens up many more applications. In this thesis, we address the problems related to large vocabulary handwriting recognition by first improving the

Now who's tipped for number ten? by Walter Terry
 With one mighty spurt Mr Selwyn Lloyd has dashed from his
 rut and is now in the race for real power within the Conservative
 party. In so intense a contest the most difficult task is to judge
 one's tongue properly. Mr Lloyd has done this superbly with his budget.
 Once he was a non-starter today he is running well along the track
 towards number ten Downing Street. But wait a minute - Selwyn Lloyd,
 the little Liverpool lawyer, as he was contemptuously described a few years
 back, as prime minister? Laughable, they used to say. The man could
 hardly make a decent speech, fluffing and fawcendering over a dreary brief.
 Dominant. But Mr Lloyd as Prime Minister is ridiculous no more. The
 very thought, I am sure, has struck Mr R A Butler, home secretary and
 apparently the heir to Downing Street. For Mr Lloyd, old nerves gone
 and seemingly dominant for the first time in his political career, has made
 a tremendous impact on the tones of Westminster with his budget.
 Maybe they don't like some of its detail, specially the payroll tax. But
 the key significance of it is that for the first time in ten years of

Figure 13 An example of a page of handwritten text taken from the LOB database [153]

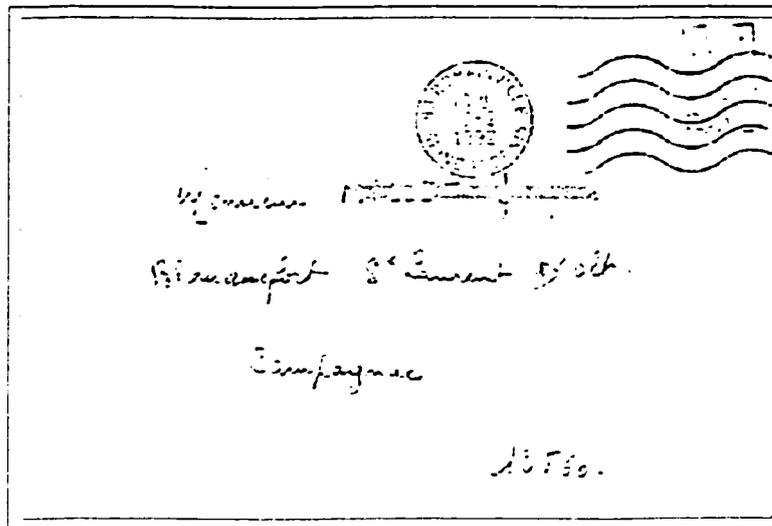


Figure 14 An example of a postal envelope taken from the testing set of the SRTP database

recognition speed and next the recognition accuracy and reliability of a baseline recognition system. The research hypotheses that will be investigated through this thesis can be summarized as:

- New search strategies incorporated to the baseline recognition system will be able to speedup the recognition process while preserving its original recognition accuracy;
- Post-processing of the N -best word hypothesis lists provided by the baseline recognition system will be able to recover some accuracy;
- The combination of a recognition and verification strategy will be able to improve the reliability of the recognition of handwritten words.

For the first research hypothesis, the idea is to take into account the particularities of the architecture of the HMMs, feature extraction, segmentation and lexicon-driven recognition to eliminate the repeated computation steps and develop fast search strategies. Our efforts to achieve such a goal are presented in Section 4.1 and the

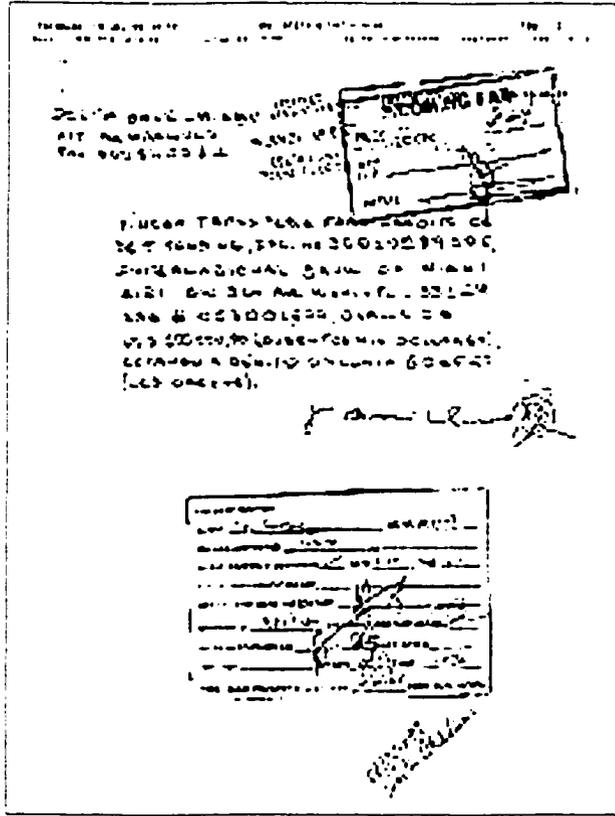


Figure 16 An example of a fax page with handwritten text, imprints and printed text

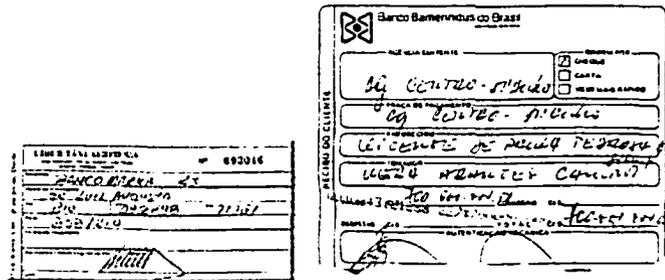


Figure 17 Two examples of business documents where the fields are filled in with handwritten text

experimental results are presented in Section 5.3.

For the second research hypothesis the idea is to develop a post-processing strategy that attempts to overcome the weaknesses of the baseline recognition system, that is, in the discrimination of very similar word hypotheses that usually are present in the N -best word hypothesis list provided at the output of the baseline recognition system. Our efforts to achieve such a goal are presented in Section 4.2 and the experimental results are presented in Section 5.4.

For the third research hypothesis the idea is to use the composite scores provided by the recognition-verification scheme to build up a more reliable rejection mechanism. Our efforts to achieve such a goal are presented in Section 4.4 and the experimental results are presented in Section 5.5.

In the context of research hypotheses, the challenge is to improve at the same time two aspects that are in mutual conflict: the recognition speed and the recognition accuracy.

Before going through the research hypotheses, it is very instructive to have an overview of the baseline SRTP recognition system, which will serve as our test and developing framework. In Chapter 3 we present the main components of the baseline recognition system as well as a figure of its performance on small, medium and large vocabularies.

CHAPTER 3

BASELINE RECOGNITION SYSTEM

The purpose of this chapter is to establish a comprehensive description of a baseline recognition system that has been used as the development environment throughout this work. We review the details of the baseline SRTP recognition system needed to understand the performance characteristics. The current baseline recognition system is an evolution of the approach originally proposed by Gilloux [49] and that has been improved in the last years by different researchers [34, 35, 38, 55, 86, 90].

First we present the main characteristics of the recognition system and an overview of its main components followed by a description of the pre-processing, segmentation and feature extraction steps. Next, the modeling of characters and words and the training of the character models are presented. We give particular attention to the recognition process that is followed by a computational complexity analysis. The performance of this baseline system in terms of recognition accuracy and recognition speed for several dynamically generated lexicons is assessed in Section 3.8. We finally conclude with some final remarks in Section 3.9.

3.1 System Overview

The SRTP recognition system is a handwritten word recognition system designed to deal with unconstrained handwriting (handprinted, cursive and mixed styles), multiple writers (writer-independent), and dynamic vocabularies of moderate size ($\approx 1,000$ words).

For these reasons it uses a segmentation-recognition approach, where handwritten words are loosely segmented (oversegmented) into sub-word units (characters or pseudo-characters). This sub-word units are modeled in a probabilistic framework

by elementary HMMs. The Markovian modeling employed in the baseline system assumes that a word image is represented by a sequence of observations. These observations should be statistically independent once the underlying hidden state sequence is known. Therefore, before segmentation, the input images are preprocessed to get rid of information that is not meaningful to recognition and that may lead to dependence between observations. Following the segmentation, two sequences of high level features are extracted from the segments to form an observation sequence. During training, since only the word labels are available, word models are built up of the concatenation of the appropriate character models and the training algorithm decides itself what the optimal segmentation might be. Recognition is carried out by a lexicon-driven decoding algorithm where each word in the lexicon is modeled by an HMM created by concatenating character HMMs. The decoding algorithm finds the N -best word hypotheses which have the highest *a posteriori* probability given the observation sequence. An overview of the main components of the baseline recognition system is presented in Figure 18. In the following sections we describe the main components of the baseline system.

3.2 Pre-Processing

The pre-processing attempts to eliminate some variability related to the writing process and that are not very significant under the point of view of the recognition, such as the variability due to the writing environment, writing style, acquisition and digitizing of image. Besides that, the pre-processing steps are also welcome to respect the assumption of the Markovian modeling: the observations in the sequence should be statistically independent, once the underlying hidden state is known.

The pre-processing module includes the following steps:

- Baseline slant normalization;
- Lowercase character area normalization (for cursive words);

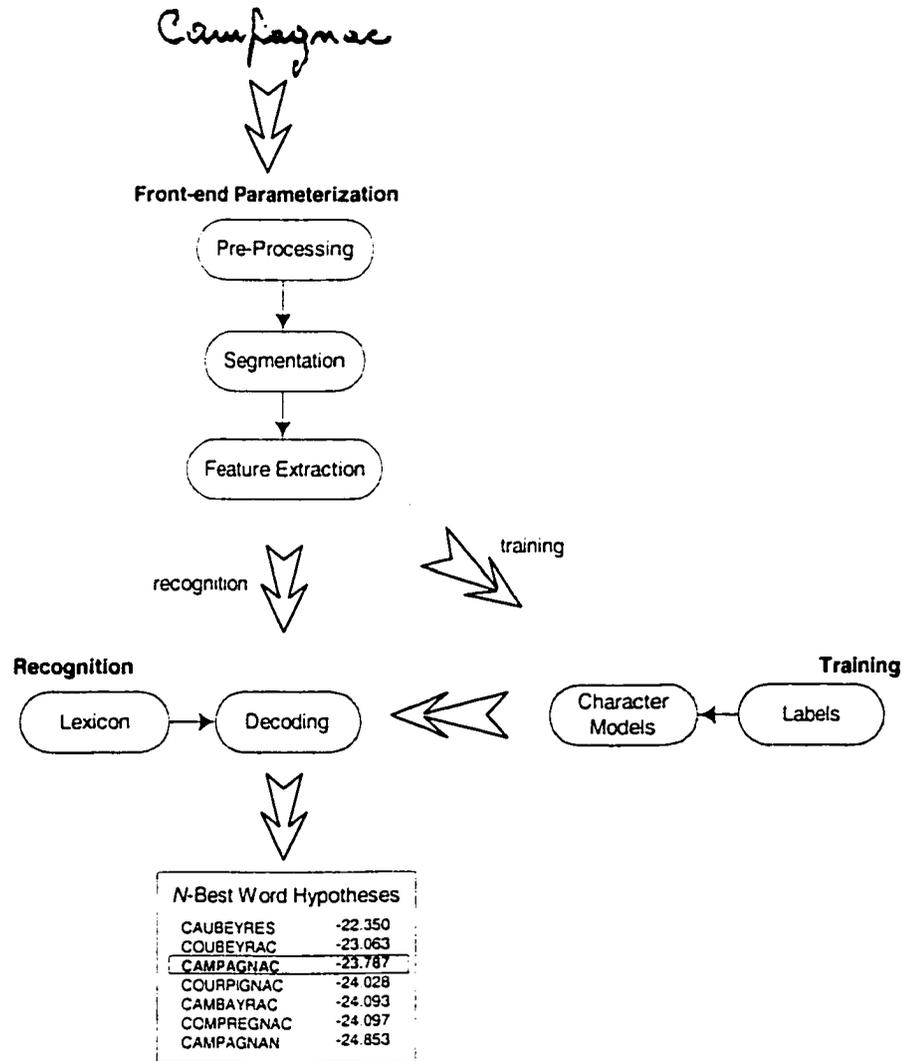


Figure 18 An overview of the main modules of the baseline SRTP recognition system

- Character skew correction:
- Smoothing.

The first two steps attempt to ensure a robust extraction of the first feature set, mainly ascenders and descenders, while the third step is required since the second feature set shows a significant sensitivity to character slant. Figure 19 shows two examples of the pre-processing steps applied to cursive handwritten words. More details about the preprocessing can be found in [34, 35].

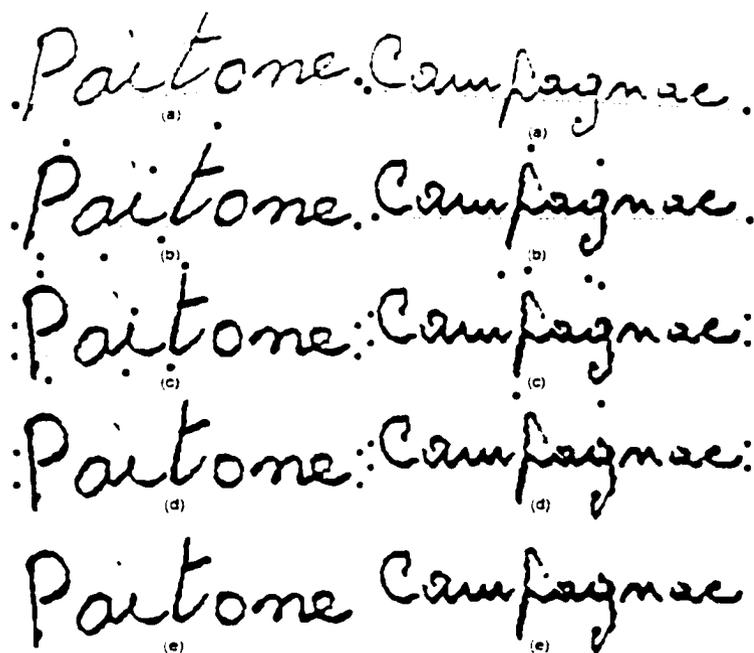


Figure 19 An example of the pre-processing steps applied to cursive words: (a) original word images, (b) baseline slant normalization, (c) character skew correction, (d) lowercase character area normalization, (e) final images after smoothing

3.3 Segmentation of Words into Characters

Segmentation of words into basic units is necessary when dealing with dynamic vocabularies of moderate size. The segmentation method performs an explicit seg-

mentation of the words that deliberately proposes a high number of segmentation points, offering in this way several segmentation options, the best ones to be validated during recognition. The segmentation algorithm is based on the analyses of the upper and lower contours, loops, and contour minima. Then, each minimum satisfying some empirical rules gives rise to a segmentation point. Mainly, the algorithm looks in the neighborhood of this minimum for the upper contour point that permits a vertical transition from the upper contour to the lower one without crossing any loop, while minimizing the vertical transition histogram of the word image. If the crossing of a loop is unavoidable, no segmentation point is produced. This strategy may produce correctly segmented, undersegmented, or oversegmented characters. Figure 20 illustrates the behaviour of the segmentation algorithm.

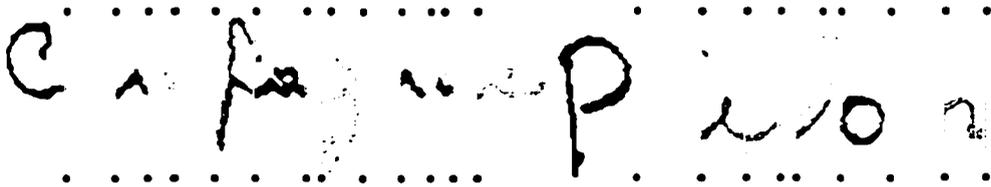


Figure 20 Two examples of the loose segmentation of handwritten words into characters or pseudo-characters

3.4 Feature Extraction

The aim of the feature extraction phase is to extract, in an ordered way, a set of relevant features that reduce redundancy in the word image while preserving the discriminative information for recognition. The main philosophy in this step is that, unlike isolated character recognition, lexicon-driven word recognition approaches do not require features to be very discriminative at the character or pseudo character level because other information, such as context, word length, etc., are available and permit high discrimination of words. Thus, the baseline system considers features at the grapheme level with the aim of clustering letters into classes. A grapheme may consist of a full character, a fragment of a character or more than a character.

The sequence of segments obtained by the segmentation process is transformed into a sequence of symbols by considering two sets of features where the first feature set is based on global features, namely loops, ascenders, and descenders. Ascenders (descenders) are encoded in two ways according to their relative size compared to the height of the upper (lower) writing zone. Loops are encoded in various ways according to their membership in each of the three writing zones and their relative size compared to the sizes of these zones. The horizontal order of the median loop and the ascender (or descender) within a segment are also taken into account to ensure a better discrimination between letters. Each combination of these features within a segment is encoded by a distinct symbol, leading in this way to an alphabet of 27 symbols [34]. The second feature set is based on the analysis of the bidimensional contour transition histogram of each segment in the horizontal and vertical directions. After a filtering phase consisting of averaging each column (row) histogram value over a five pixels-wide window centered in this column (row) and rounding the result, the histogram values may be equal to 2, 4, or 6. In each histogram it is focused only on the median part, representing the stable area of the segment, and it is determined the dominant transition number for which the number of columns (rows) with a histogram value equal to the dominant transition number is maximum. Each different pair of dominant transition numbers is then encoded by a different symbol or class. After having created some further subclasses by a finer analysis of the segments, this coding leads to a set of 14 symbols [34]. The baseline system also uses five segmentation features that try to reflect the way segments are linked together. For connected segments, two configurations are distinguished: If the space width is less than a third of the average segment width, it is considered that there is no space. Otherwise, the space is validated and encoded in two ways, depending on whether the space width is smaller than average segment width. If the two segments are connected, the considered feature is the segmentation point vertical position which is encoded in two ways depending on whether the segmentation point

is close to or far from the writing baseline. Finally, given an input word image, the output of the feature extraction process is a pair of symbolic descriptions of equal length, each consisting of an alternating sequence of segment shape symbols and associated segmentation point symbols. Figure 21 shows two examples of sequences of observations generated from handwritten words.

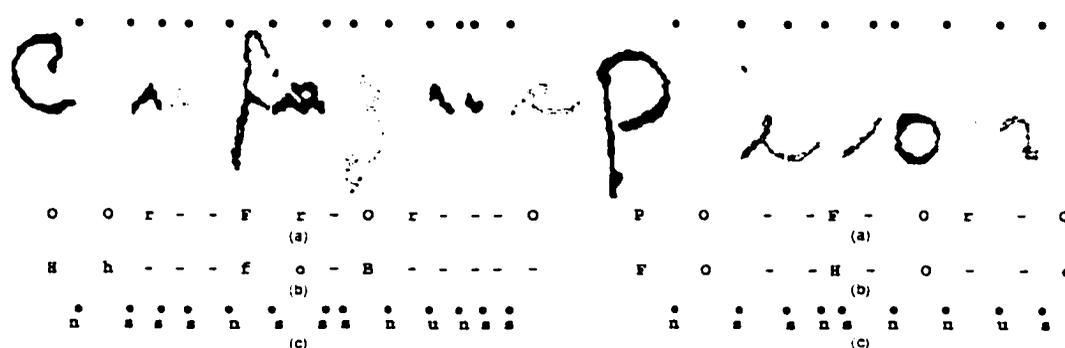


Figure 21 An example of observation sequences extracted from two handwritten words: (a) sequence of global features: (b) sequence of bidimensional transition histogram features: (c) sequence of segmentation features

3.5 Character and Word Models

Several HMM architectures can be considered for handwritten word recognition. This stems from the fact that handwriting is certainly not a Markovian process and, even if it were so, the correct HMM architecture is actually not known. The usual solution to overcome this problem is to first make structural assumptions and then use parameter estimation to improve the probability of generating the training data by the models.

The elementary HMMs can be completely characterized by the state-transition probability distribution matrix $A = \{a_{ij}\}$, the observation symbol probability distribution $B = \{b_j\}$, and the initial state distribution $\Pi = \{\pi_i\}$ as:

$$\lambda = \{A, B, \Pi\} = \{a_{ij}, b_j, \pi_i; i, j = 1, \dots, N\} \quad (3.1)$$

where N is the total number of states.

However, the baseline system uses discrete HMMs where observations are produced by transitions rather than by states. Furthermore, transitions with no output are also incorporated into the model. In this case, the classic definition of an HMM is modified and now the compact notation is $\lambda = \{A, A', \pi_i\}$, where $A = \{a_{ij}^{vk}\}$ is the probability distribution associated with transitions from states s_i to state s_j and at the same time producing observation symbol v_k and $A' = \{a_{ij}''\}$ is the probability distribution associated with null transitions from states s_i to state s_j and at the same time producing null observation symbol. More details about this peculiar definition of HMMs can be found in Appendix 3.

In the baseline recognition system, the assumptions to be made are related to the behavior of the segmentation process. As the segmentation process may produce either a correct segmentation of a letter, a letter omission, or an oversegmentation of a letter into two or three segments, a 10-state left-right HMM having three paths to take into account these configurations (Figure 22) is adopted. In this model, observations are emitted along transitions. Transition t_{1-10} , emitting the null symbol, models the letter omission case. Null transition t_{1-2} models the case of correctly segmented character while transition t_{2-9} and t_{9-10} emit symbols encoding the correctly segmented letter shape and the nature of the segmentation point associated with this shape respectively. Null transition t_{1-3} models the case of oversegmentation into two or three segments. Transitions t_{3-4} , t_{5-6} , and t_{8-9} are associated with the shapes of the first, second, and third parts of an oversegmented letter, while t_{4-5} , t_{7-8} and t_{9-10} model the nature of the segmentation points that gave rise to this oversegmentation.

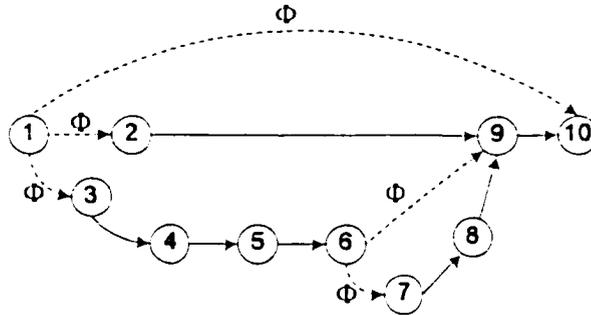


Figure 22 The left-right character HMM with 10 states and 12 transitions where five of them are null (dotted lines) [35]

In addition, there is a special model for inter-word space, in the case where the input image contains more than one word. This model simply consists of two states linked by two transitions, modeling a space or no space between a pair of words (Figure 23).

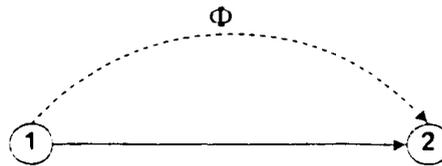


Figure 23 The inter-word space model with 2 states and 2 transitions where one is null (dotted line) [35]

In summary, the baseline system has 70 HMMs that represent 26 uppercase characters (A-Z), 26 lowercase characters (a-z), 10 digits (0-9), and other 8 special symbols [34,35].

3.6 Training of Character Models

The goal of the training phase is to estimate the best parameter values of the character models, say A and A' in λ , given a set of training examples and their associated word labels. Since the exact orthographic transcription of each training word image

is available, the word model, denoted as $\hat{\lambda}$, is made up of the concatenation of the appropriate character models $\hat{\lambda} = \lambda_1 \oplus \lambda_2 \oplus \dots \oplus \lambda_L$, where L is the number of character that form a word. In such a scheme, the final state of an HMM becomes the initial state of the next one, and so on.

The baseline system uses a variant of the Baum–Welch algorithm [35,36] for which the segments produced by the segmentation algorithm need not be manually labeled. Since a sufficient training database is available, the Baum–Welch training procedure allows the recognizer to capture contextual effects and permits the segmentation of the feature sequence into characters and the reestimation of the associated transitions so as to optimize the likelihood of the training database. Thus, the recognizer decides for itself what the optimal segmentation might be, rather than being heavily constrained by a priori knowledge based on human intervention. From an implementation point of view, given a word composed of L characters a new index corresponding to the currently processed character is added to the probabilities involved in the Baum–Welch algorithm. Then, the results of the final forward (initial backward) probabilities at the last (initial) state of the elementary HMM associated with a character are moved forward (backward) to become the initial forward (final backward) probabilities at the initial (last) state of the elementary HMM associated with the following (previous) character. In addition to the training dataset, a validation set is used to test the reestimated model after each training iteration. The training stops when the likelihood of the training set becomes sufficiently small. In Appendix 3 the training procedure is described briefly.

3.7 Recognition of Unconstrained Handwritten Words

The recognition process consists of determining the word maximizing the *a posteriori* probability that a word w (modeled by $\hat{\lambda}$) has generated an unknown observation sequence O , that is:

$$\hat{w} \ni P(\hat{w}|O) = \max_{w \in \mathcal{R}} P(w|O) \quad (3.2)$$

where \mathcal{R} is the recognition vocabulary. Using Bayes' Rule and assuming that $P(O)$ does not depend on w , and equal *a priori* probabilities of words $P(w)$, the MAP decoding rule can be approximate by:

$$\hat{w} = \arg \max_{w \in \mathcal{R}} P(O|w) \quad (3.3)$$

The estimation of $P(O|w)$ requires a probabilistic model that accounts for the shape variations O of a handwritten word w . Such a model consists of a global Markov model created by concatenating character HMMs at state level. The architecture of this model remains the same as in training. However, as the writing style is unknown during the recognition, a character model here actually consists of two models in parallel, associated with the uppercase and lowercase representations of a character as shown in Figure 24. As a result, two consecutive character models are now linked by four transitions associated with the various ways two consecutive characters may be written: uppercase–uppercase (UU), uppercase–lowercase (UL), lowercase–uppercase (LU), and lowercase–lowercase (LL). The probabilities of these transitions, as well as those of starting a word with an uppercase (OU) or lowercase (OL) letter, are estimated by their occurrence frequency in the training database. Recognition is carried out using a variant of the Viterbi algorithm as described next.

3.7.1 The Viterbi Algorithm

The baseline recognition system [35, 36] uses a variant of the Viterbi algorithm that accounts for the specifics of the HMMs, that is, the concatenation of character HMMs, observations generated along transitions, the presence of null transitions,

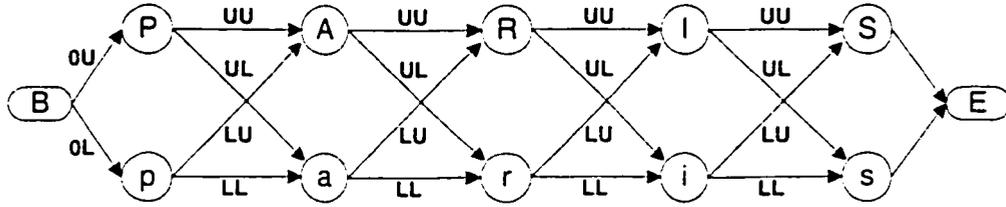


Figure 24 The global recognition model formed by the concatenation of upper-case and lower-case character models where “B” and “E” denote the beginning and the end of a word respectively

left-right transitions, and well-defined initial (s_1) and final states (s_{10}). The complete decoding procedure to decode a single word and considering log-probabilities (likelihoods) is described as follows.

1) *Initialization:* For $j = 1, 2, \dots, N$

$$\begin{aligned}
 \delta_1(1, 1) &= 1.0 \\
 \omega_1(1, 1) &= 0 \\
 \delta_1(1, j) &= \max_{1 \leq i < N} [\delta_1(1, i) + a_{ij}^{\Phi}] \\
 \omega_1(1, j) &= \arg \max_{1 \leq i < N} [\delta_1(1, i) + a_{ij}^{\Phi}]
 \end{aligned} \tag{3.4}$$

where $\delta_t(l, i)$ is the probability of the best path that accounts for the t first observations and ends at state s_i at time t and $\omega_t(l, i)$ keeps track of the argument that maximizes Equation 3.4. The indexes l , t , and i (or j) denote the character, observation and state index respectively.

Note that the the initialization in Equation 3.4 is valid only for the first character within a word ($l=1$). Since character HMMs are concatenated, for the remaining characters the initialization differs slightly.

For $l = 2, 3, \dots, L$ and $t = 1, 2, \dots, T + 1$

$$\begin{aligned}\delta_t(l, 1) &= \delta_t(l-1, N) \\ \omega_t(l, 1) &= 1\end{aligned}\tag{3.5}$$

2) *Recursion:* For $l = 1, 2, \dots, L$, $t = 2, \dots, T+1$, and $j = 1, 2, \dots, N$

$$\begin{aligned}\delta_t(l, j) &= \max \left\{ \max_{1 \leq i < N} [\delta_{t-1}(l, i) + a_{ij}^{O_{t-1}}] ; \max_{1 \leq i < N} [\delta_t(l, i) + a_{ij}^{*P}] \right\} \\ \omega_t(l, j) &= \begin{cases} \arg \max_{1 \leq i < N} [\delta_t(l, i) + a_{ij}^{*P}] & \text{if } ij \text{ is null} \\ \arg \max_{1 \leq i < N} [\delta_{t-1}(l, i) + a_{ij}^{O_{t-1}}] & \text{otherwise} \end{cases}\end{aligned}\tag{3.6}$$

$$\zeta_t(l, j) = \begin{cases} 1 & \text{if } ij \text{ is null} \\ 0 & \text{otherwise} \end{cases}$$

Here we have introduced a new variable $\zeta_t(l, j)$ that keeps track if the argument that maximizes Equation 3.6 comes either from a null or from a non-null transition.

3) *Termination:*

$$\begin{aligned}P^*(L) &= \delta_{T+1}(L, N) \\ q_{T+1}^*(L) &= N\end{aligned}\tag{3.7}$$

4) *Backtracking:* For $l = L-1, L-2, \dots, 1$, and $t = T, T-1, T-2, \dots, 1$

$$q_t^*(l) = \omega_{t+1}(l, q_{t+1}^*)\tag{3.8}$$

However, to account for the concatenation of characters when the first state of a character model is reached during the backtracking procedure, the level must be decreased and $q_t^*(l-1)$ is given as follows:

$$q_t^*(l-1) = s_N \quad \text{if} \quad q_t^*(l) = s_1 \quad (3.9)$$

Besides the level changing we also have to account for the presence of null transitions. So, a new variable that accounts for the presence of null transitions, denoted as $q'_t(l)$ is introduced. After computing $q_t^*(l)$ we have to check if the variable ζ indicates the presence of a null transition, that is, $\zeta_t(l, q_t^*) = 1$. In such a case, without decreasing the time index t , $q'_t(l)$ is computed as:

$$q'_t(l) = \omega_t(l, q_t^*) \quad \text{if} \quad \zeta_t(l, q_t^*) = 1 \quad (3.10)$$

Following the presence of a null transition during the backtracking procedure, that is $\zeta_{t+1}(l, q_{t+1}^*) = 1$, the computation of $q_t^*(l)$ is modified slightly as follows (now with a decrease in the time index t):

$$q_t^*(l) = \omega_{t-1}(l, q'_{t+1}) \quad \text{if} \quad \zeta_{t+1}(l, q_{t+1}^*) = 1 \quad (3.11)$$

So, the best state sequence denoted as Q^* will be given by:

$$Q^* = [q_T^*(L)q'_T(L)q_{T-1}^*(\cdot)q'_{T-1}(\cdot) \dots q_2^*(\cdot)q'_2(\cdot)q_1^*(1)q'_1(1)] \quad (3.12)$$

The same decoding procedure is applied repeatedly for each word in the lexicon and

at the end we pick up the words that give the highest probabilities. Note that we have not considered the parallel association of uppercase and lowercase HMMs as described before (Figure 24). However, to decode unconstrained handwritten words it is necessary to use such a parallel decoding. In practice, we have to introduce two new arrays, say δ' and ω' and change slightly the initialization procedure. More details about the integration of contextual information into the parallel model during the recognition are presented in Chapter 4.

The complete procedure to recognize an unknown word represented by a sequence of observations O can be summarized as follows:

1. For a given sequence of observations $O = (o_1 o_2 \dots o_T)$:
2. Pick up a word from the lexicon;
3. Build up the corresponding parallel word HMM by concatenating $2 \times L$ character HMMs;
4. Decode the such a word HMM and compute its probability score;
5. Compare its probability score with the probabilities scores of the words previously decoded;
6. Add the word to a list of the N -best word hypotheses if its probability score is higher than the scores of the words previously decoded;
7. Repeat from step 2 until all words in the lexicon have been decoded;
8. End.

In Appendix 3 we present an overview of the HMM concepts and the algorithm for training, evaluation, and recognition considering the conventional definition of HMMs in which outputs are generated by states (state model) and the alternative definition in which outputs are generated by transitions into states (transition model).

3.7.2 Computational Complexity and Storage Requirements

We have pointed out in the previous chapters that a key point of dealing with large vocabularies is the computational complexity of the recognition process. So, it is

very instructive to know the computational complexity and the storage requirements of the decoding algorithms. The computational complexity is measured as the number of basic operations (additions, multiplications, divisions, etc.), denoted as \mathcal{O} , performed by the algorithm.

The computational complexity of the conventional Viterbi algorithm is $\mathbb{T}\mathbb{N}^2$ for an elementary \mathbb{N} -state HMM. However, to make up a word, we have the concatenation of \mathbb{L} characters where each character may have \mathbb{H} models ($\mathbb{H}=2$ in the case of having models for uppercase and lowercase characters). Furthermore, the decision between \mathbb{H} models is taken after each character is decoded. The computational complexity to decode the whole vocabulary of size \mathbb{V} is shown in Table VII. Table VII shows the storage requirements to decode a whole word, considering the parallel recognition model. The final word probability and the best state sequence have also to be kept during the decoding.

Table VII

A summary of the approximate computational complexity and storage requirements for the baseline recognition system based on the Viterbi decoding and a flat lexicon

Computational Complexity	Storage Requirements
$\mathcal{O}(\mathbb{H}\mathbb{T}\mathbb{N}^2\mathbb{L}\mathbb{V} + \mathbb{H}\mathbb{T}\mathbb{L}\mathbb{V})$	$2\mathbb{H}\mathbb{T}\mathbb{L}\mathbb{N}$

To get a feeling on how complex the decoding of a large vocabulary is, typical values of $\mathbb{H}=2$, $\mathbb{T}=25$, $\mathbb{L}=10$, $\mathbb{N}=10$, and $\mathbb{V}=80,000$ result in $\mathcal{O}(4 \times 10^9)$. This computation to recognize a single word is already excessive for most modern machines since current personal computers can perform between 1,000 and 3,000 million floating-point operations per second (MFLOPS). Thus, it is necessary to find an alternative decoding scheme to deal with applications that require large vocabularies.

3.8 Performance of the Baseline Recognition System (BLN)

The baseline recognition system described in through this chapter was evaluated in terms of both recognition accuracy and recognition speed. Table VIII summarizes the results for the recognition accuracy and recognition speed for the baseline recognition system (BLN) considering five different sizes of dynamically generated lexicons (10, 100, 1k, 10k, and 30k entries). These lexicons are built by randomly selecting a number of words from the global lexicon with 36,116 French city names. The details about the databases and testing conditions that have been used to evaluate the performance of the baseline recognition system are presented in Chapter 5.

Table VIII

Word recognition rate and recognition time for the baseline recognition system (BLN)

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)
	TOP 1	TOP 5	TOP 10	
10	98.93	99.93	100	0.222
100	95.89	98.99	99.40	1.989
1k	89.79	95.97	97.30	19.50
10k	79.50	89.53	91.89	182.5
30k	73.70	85.17	88.15	493.1

The results presented in Table VIII are the mean values that were obtained according to testing procedure describe in Chapter 6. The standard deviation of the recognition rate, denoted as σ_{RR} is less than 0.4%, 0.2%, and 0.1% for *TOP 1*, *TOP 5*, and *TOP 10* respectively. The standard deviation of the recognition speed, denoted as σ_{RS} is less than 5%.

The performance reported in Table VIII is the landmark for our work. One of the

goals of the algorithms and methods proposed in the next chapter is to improve such a performance, both in terms of recognition accuracy and recognition speed.

3.9 Baseline System Summary

The purpose of this chapter has been to present the main components of the baseline SRTP recognition system, as well as to evaluate its performance on small, medium and large vocabulary tasks. We have obtained a measure of its efficiency along two basic dimensions: recognition accuracy and recognition speed. The evaluations were carried out on a test set of unconstrained handwritten city names from the SRTP database. The tests are run with five different lexicon sizes that are generated dynamically from a 36,116-word vocabulary.

The immediate conclusion from these measurements is that the baseline recognition system is powerful and deals very well with the variability of different writing styles. One interesting characteristic is that it employs parallel models to deal with uppercase and lowercase characters. Of course, there is a serious drawback: the system has a good performance to deal only with small and medium vocabularies. It cannot be used in practical, large vocabulary handwriting recognition applications. While it is possible to improve the recognition speed by pruning the search and using less sophisticated character models, such measures cannot overcome the inherent algorithm complexities of the system. Moreover, they may result in an unacceptable reduction in the recognition rate.

In the context of our research goals, this system is the basis of the large vocabulary handwriting recognition system that will be presented in Chapter 4 where we propose several modifications on the baseline SRTP recognition system in order to improve its performance when dealing with vocabularies with tens of thousands of words. The proposed modifications are not only for improving the recognition speed, but also the recognition accuracy.

CHAPTER 4

THESIS CONTRIBUTIONS

We have seen in the previous chapter that the baseline SRTP recognition system is several tens of times too slow for large vocabulary tasks since it takes more than 8 minutes to recognize a word in a 30,000-word vocabulary compared to 2 seconds for a 100-word vocabulary, as well as much less accurate since the recognition rate is about 74% for 30,000-word vocabulary compared to about 96% for a 100-word vocabulary. In order to be practically useful, large vocabulary handwriting recognition systems have to be efficient in their usage of computational resources and achieve a reasonable recognition accuracy as well.

This chapter presents the contribution of the author to overcome problems of computational complexity and recognition accuracy to build up a large vocabulary handwriting recognition system with the following characteristics: writer-independent (or omniwriter), unconstrained handwriting (handprinted, cursive, or mixed), and very-large vocabulary (80,000 words).

First, speed issues in handwriting recognition like the implementation of a tree search, elimination of character models during the decoding, pruning strategies, a novel paradigm for word decoding, and task partitioning by incorporating concepts of distributed computation will be addressed. The main idea behind the strategies presented in Section 4.1 is to reduce the recognition time while preserving the recognition accuracy to make feasible large vocabulary handwriting recognition tasks.

Sections 4.2 and 4.3 are about improving the recognition accuracy by postprocessing lists of N -best word hypotheses generated by the handwriting recognition system based on HMMs by a different recognition strategy based on the recognition of isolated handwritten characters by neural networks and further combining the results

with the former to optimize performance.

Finally, improvement on the reliability of the large-vocabulary handwriting recognition system is introduced in Section 4.4 where a rejection mechanism evaluate the confidence scores of the words and reject those with low confidence scores.

Before presenting our contributions it is worthwhile to identify which parts of the baseline recognition system will remain unaltered and which parts will be replaced or even introduced as a result of the research that has been carried out and is described in this chapter. Figure 25 shows the main components of the baseline recognition system where those inside the light gray box will not be modified, and those inside the dark gray box will be proposed and introduced as result of this thesis.

4.1 Speed Issues in Handwriting Recognition

One goal of this thesis is to demonstrate that it is possible to achieve high recognition speed on large vocabulary tasks without compromising the recognition accuracy offered by current recognition systems.

Handwriting recognition systems consist of several components: pre-processing, segmentation, feature extraction, character models, learning and decoding algorithms and language models. Clearly, these components contribute to the various dimensions of efficiency of the systems – accuracy and speed. But much of the research in handwriting recognition has been focused on the first four components towards improving recognition accuracy. This section concentrates on improving the decoding algorithms while preserving the gains made in the other components.

Another reason to concentrate on the decoding problem is that much of the research in handwriting recognition has been focused on constrained problems where the vocabularies are relatively small. It seems reasonable to expect that research will move to more unconstrained problems in a near future as the results achieve in these

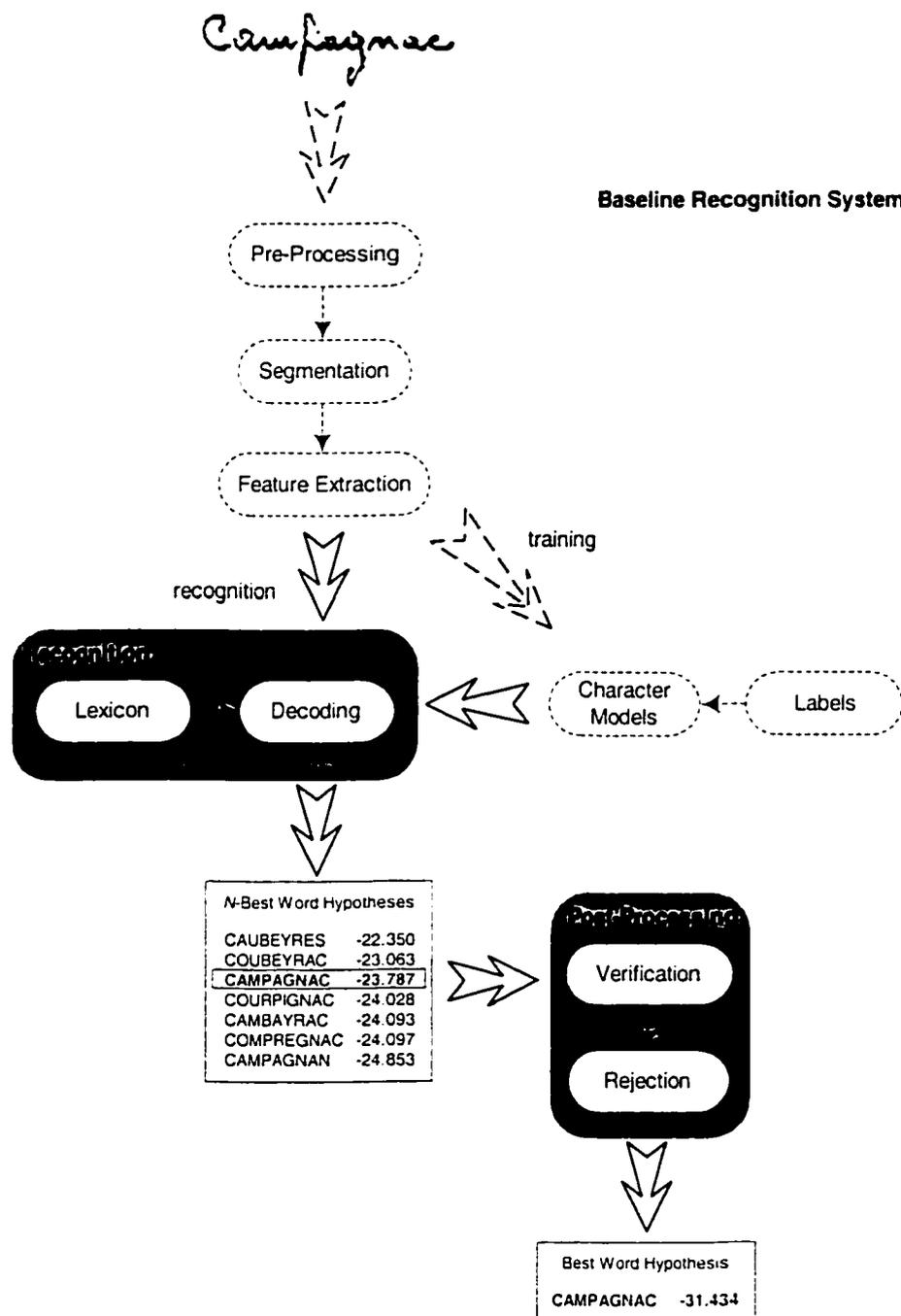


Figure 25 An overview of the modules of the baseline recognition system that are used through this thesis (light gray) and the modules introduced by the author (dark gray)

constrained problems cannot be further improved.

In this section we discuss several strategies, algorithms and heuristics for improving the efficiency of a recognition system. We use the SRTP recognition system described in Chapter 3 as a baseline for comparison. Since the focus of this work is in improving search algorithms, we use the same pre-processing, segmentation and feature extraction methods, as well as the same character models as in the baseline recognition system.

The outline of this section is as follows. In Section 4.1.1 the representation of the lexicon as a tree structure is addressed. The main idea behind this representation is to reduce the recognition time by avoiding repeated computations for common parts of the lexicon. Section 4.1.2 is about the management of the computational complexity when multiple models are used to represent the characters together with a lexical tree. In Section 4.1.5 we present an efficient decoding algorithm that searches for the best word hypotheses and integrates HMM state decoding, lexical tree search, character pruning and contextual information. Time and length heuristics are also integrated to such a decoding algorithm to reduce the search effort with relatively no loss of accuracy. In Section 4.1.7 we introduce a novel fast two-level HMM decoding strategy that breaks up the computation into two steps: decoding of the HMM states and decoding of word hypotheses. This search strategy accelerates the decoding of word with no loss in accuracy. In Section 4.1.8 we integrate concepts of distributed computation to the recognition process to further reduce recognition time while preserving the accuracy. The speed issues are concluded with Section 4.1.9 that summarizes the performance of all of the presented strategies.

4.1.1 Tree-Structured Lexicon

A single source of computational efficiency in performing search is in organizing the HMMs to be searched as a character tree (Figure 26b), instead of the flat structure

(Figure 26a) described in Section 1.3.2.

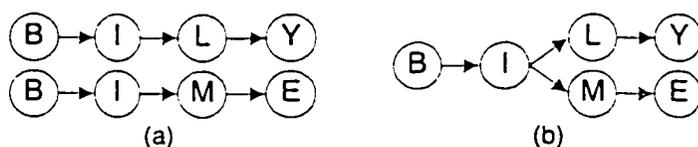


Figure 26 Two 4-character words (*BILY* and *BIME*) represented as: (a) a flat structure, (b) a tree structure where the prefix *BI* is shared by both words

While a flat lexicon is easy to implement and to integrate into an HMM framework [35] and it provides a good trade-off in terms of computational complexity for small to medium lexicons. Assuming a unique character model (HMM) for each character class (e.g. the characters “a” and “A” are modeled by the same HMM, say λ_A), the total number of HMM states to be searched during the decoding, of a 10-word lexicon is approximately 1,200¹. However, the extension to a 10,000-word lexicon increases the number of states linearly. As a consequence, the recognition time grows about 1,000 times [90].

It is also possible to organize the lexicon as a character tree instead of a flat structure. In this structure, referred to as tree-organized lexicon or lexical tree, if the spellings of two or more words contain the same n initial characters, they share a single sequence of n HMMs during the decoding process (Figure 26b). Now, for the same 10-word lexicon approximately 900 HMM states have to be searched during the decoding².

While for small to medium vocabularies this reduction in the number of HMM states to be searched is irrelevant, in the case of large vocabularies (>1,000 words), it is more likely to have words with similar prefixes, so it is very interesting to use a

¹ LNV , where $L=12$ is the average length of the words in the lexicon, $N=10$ is the number of states of the HMM that model characters, and $V=10$ is the lexicon size.

² $(L - L_s)NV$, where $L_s=3$ is the average number of shared characters per word in the lexicon.

tree-structured lexicon to avoid repeated computation of such prefixes. But notice that L_s depends strongly on the nature of the lexicon. Table IX shows the reduction in the number of characters by representing a vocabularies of different sizes as tree structures. The expected speedup in the recognition process by representing the vocabulary as a lexical tree is proportional to the *Reduction Ratio* shown in Table IX what is not so much significant. However the tree representation preserves the recognition accuracy.

Table IX

Average number of characters for several different sizes of lexicons of French city names represented as a flat structure and as a tree structure

Lexicon Size	Number of Characters		Reduction Ratio
	Flat Lexicon	Lexical Tree	
10	119	113	1.05
100	1,198	987	1.21
1k	11,998	8,361	1.43
10k	120,035	66,558	1.80
30k	360,012	173,631	2.07

4.1.2 Multiple Character Class Models

In Section 4.1.1 we have assumed a unique character model (HMM) for each character class. However, this assumption does not hold in the case of unconstrained handwriting recognition because it is very difficult that a single model captures the high variability and ambiguity of a large number of writing styles.

It has been shown that when a large amount of training data is available, the performance of a word recognizer generally can be improved by creating more than one model for each of the recognition units [28, 140] because it provides more accurate representation of the variants of handwriting. However, identification of writing styles prior to recognition is also a difficult task that may also introduce errors to

the recognition process. One alternative has been to incorporate to the recognition itself the capability of recognizing unconstrained handwriting [35]. The most intuitive modeling is to have different models for uppercase and lowercase characters [35]. On the other hand, while multiple models may improve the accuracy of a recognition system, they also increase the computational complexity. This is the point that we have to account for now.

Assuming that each character class is now modeled by multiple HMMs (e.g. the character “M” is modeled by $\lambda_m, \lambda_{m'}, \lambda_M, \lambda_{M'}$, etc.) that may represent an uppercase letter model or a lowercase letter model, a letter model at the beginning or at the end of a word, etc. So, the problem now is how to integrate these multiple character models to the lexical search.

Usually we do not know which model we have to use during the recognition since the writing style is not known. So, we have to account for all possible combinations of uppercase and lowercase models during the decoding. Figure 27a shows the words of Figure 26a but now considering all combinations of models. For the sake of simplicity, we assume that for each character class we have only two different models, one that model uppercase characters and another that models lowercase characters (e.g. the character “A” is modeled by λ_A and λ_a). For a 10-word lexicon approximately 4,096,000 HMM states³ have to be decoded. The extension to a 10,000-word vocabulary blows up the number of HMM to be searched. Even if we represent the lexicon by a tree structure as shown in Figure 27b, the computation required to decode the whole vocabulary will be enormous. This particular problem of how to manage the search complexity in lexical trees when multiple character class models are used has not been addressed in handwriting recognition.

³ $\mathbb{H}^{\mathbb{L}}\mathbb{N}\mathbb{V}$, where $\mathbb{H}=2$ is the number of different models for each character class, $\mathbb{L}=12$ is the average length of the words in the lexicon, $\mathbb{N}=10$ is the number of states of the character HMMs, and $\mathbb{V}=10$ is the size of the vocabulary.

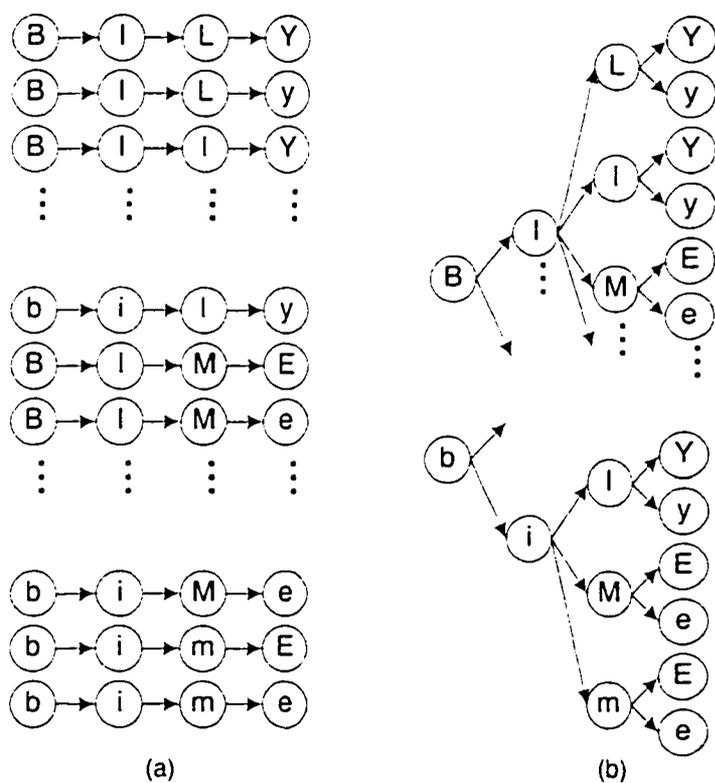


Figure 27 Two 4-character words (*BILY* and *BIME*) represented as: (a) a flat structure where each character class (e.g. "M") has two models (e.g. an uppercase λ_M and a lowercase λ_m model) and all combinations of models are allowed, (b) a tree representation of the same two words

4.1.3 Best Model Selection

Instead of keeping all possible combinations of models, we can make use of the maximum approximation to select only the more likely combinations. Figure 28 shows an approximated way to represent the lexical tree of Figure 27b. In such cases, the number of HMM states to be searched can be approximated by $H(L - L_s)NV$. By such a representation, the number of HMM states to be searched no longer grows exponentially with the number of models per character class, but linearly.

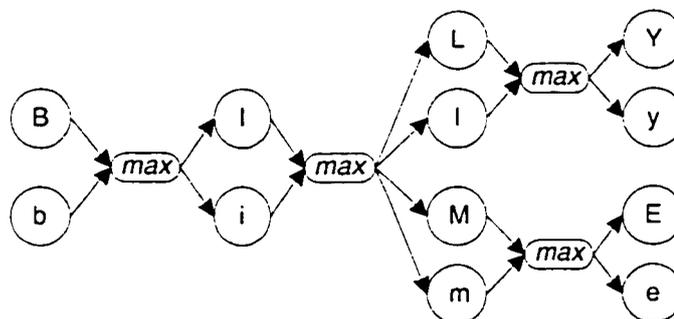


Figure 28 A simplified representation of the lexical tree of Figure 27b by using the maximum approximation and selecting only the one character model at each tree level

While the growth of the number of HMM states can be controlled by using the maximum approximation, it can cause pruning errors during search and the accuracy may not be preserved. This occurs because we are taking decisions about the best models based on the local context and not based on the whole word. It is obvious that the best model in a level is certainly not always the correct one since the sequence of observations that represents the word to be recognized may be partially garbled by noise. But the only manner to have a precise search is to evaluate each combination of character models independently and that is computationally very expensive.

The criteria to select the best model at each tree level is based on the *a posteri-*

ori probability of each model. So, throughout the Sections 4.1.5 and 4.1.6 we use this approximation to decode unconstrained handwritten words in a large vocabulary, considering that the character classes have multiple models, and the lexicon is represented by a tree structure.

4.1.4 A Review of the Handwriting Recognition Problem

The basic problem in large vocabulary handwriting recognition is given a handwritten word to recognize represented by a sequence of observations $O = (o_1 o_2 \dots o_T)$ where T is the number of observations in the sequence, and a recognition vocabulary represented by \mathcal{R} corresponding to V unique words, find the word $w \in \mathcal{R}$ that best matches to the input pattern. The standard approach is to assume a simple probabilistic model of handwriting production whereby a specified word, w , produces an observation sequence O with probability $P(w, O)$. The goal is then to decode the word, based on the observation sequence, so that the decoded word has the maximum *a posteriori* (MAP) probability, i.e.:

$$\hat{w} \ni P(\hat{w}|O) = \max_{w \in \mathcal{R}} P(w|O) \quad (4.1)$$

Using Bayes' Rule and assuming that $P(O)$ does not depend on w , and equal *a priori* probabilities of words $P(w)$, the MAP decoding rule can be approximate by:

$$\hat{w} = \arg \max_{w \in \mathcal{R}} P(O|w) \quad (4.2)$$

As we have pointed out in the preceding chapters, the way we compute $P(O|w)$ for large vocabularies is to build statistical models for sub-word units (characters) in an HMM framework, build up word models from these sub-word models using a lexicon

to describe the composition of words, and then evaluate the model probabilities via standard concatenation methods.

Considering a discrete symbol observation we can define a sub-HMM by its compact notation as $\lambda = (A, B, \Pi)$, where A is the state-transition probability distribution, B is the observation symbol probability distribution, and Π is the initial state distribution. The number of states in the models is denoted as N . In our framework of handwriting recognition, there are several sub-word HMMs that model characters, digits and symbols and a recognition vocabulary represented by \mathcal{R} which contains V words in which average length is L characters. So, a word model $\hat{\lambda}$, regarded as a "super-HMM" is build by the concatenation of L sub-word HMMs, i.e.:

$$\hat{\lambda} = \lambda_1 \hat{\oplus} \lambda_2 \hat{\oplus} \cdots \hat{\oplus} \lambda_L \quad (4.3)$$

Here, we have made use of the so-called maximum approximation which is also referred to as Viterbi approximation. In this maximum approximation, the search space can be described as a huge network through which the best time alignment path has to be found. The search has to be performed at two levels: at the state level and at the word level w . So, the decoding rule of Equation 4.2 can be rewritten as:

$$\hat{w} = \arg \max_{w \in \mathcal{R}} \{ \max_Q P(o_1 \dots o_T, q_1 \dots q_T | \lambda_1 \dots \lambda_L) \} \quad (4.4)$$

where q_t denotes the state at time t and $Q = (q_1 q_2 \dots q_T)$ is the best state sequence.

Conventional procedure employed in handwriting recognition is to concatenate character HMMs at state level to build up word HMMs that are further matched against

the sequence of observations using a strict left-to-right Viterbi algorithm. So, in such an approach, if an individual character appears repeated within a word or within several words, its likelihood scores have to be computed again because it appears in a different context (different neighbor characters). So, its initialization depends now on the Viterbi score of the immediate preceding character, that is, the likelihood score and the frame where the preceding character terminates. The repeated computation may be minimized if the lexicon is represented as a tree structure. The computation of the prefixes can be done once and shared by words with similar spellings. However, the remainder of the word still have to be computed in a conventional manner.

4.1.5 Lexicon-Driven Level Building Algorithm (LDLBA)

The use of a lexical tree can be much more complex than a flat lexicon considering the problems to integrate multiple character class models and the advantages of avoiding repeated computation of common prefixes may become insignificant. The main problem is that a tree node can have multiple models associated with it and that results in a new tree branch for each additional model, that is, \mathbb{H}^L search hypotheses for each word.

The difficulties can be overcome by using a level building algorithm (LBA) to decide locally the best model and reduce the amount of computation. The LBA was introduced by Myers and Rabiner [125, 126] to recognize connected words from a small vocabularies and highly constrained word syntax. Since the vocabulary is small they have used as the basic speech-recognition units whole words. Recently the LBA has also been used for the recognition of printed [37], cursively handwritten words [137] and numerical strings [14]. However, they do not have used the LBA in such a way to integrate multiple class character models, lexical tree search and contextual information.

Given a set of sub-word models $\mathcal{C} = \{\lambda_1, \lambda_2, \dots, \lambda_M\}$ where λ_m models character classes (letters, digits, and symbols), and a sequence of observations $O = (o_1 o_2 \dots o_T)$, word recognition means decoding O into a sequence of L models $\lambda_1 \oplus \lambda_2 \oplus \dots \oplus \lambda_L$. Namely, it is to match the observation sequence to a state sequence of models with maximum joint likelihood. In the same way that the Viterbi algorithm matches a model to a sequence of observations, determining the maximum likelihood state sequence of the model given the sequence of observations [42], the LBA is used to match an observation sequence to a number of models [141]. The LBA jointly optimizes the segmentation of the sequence into sub-sequences produced by different models, and the matching of the sub-sequences to particular models.

However, it is necessary to adapt this algorithm to take into account some particular characteristics of our character model since it is modeled by 10-state left-right transition-based HMMs (Figure 22 in Chapter 3) [35]. Moreover, we also take into account some contextual information that is given by the probability to start a word with an uppercase or a lowercase character and the probability of changing or keeping the same writing style within a word. Since we are using a lexicon-driven strategy the HMMs that will be evaluated at each level of the LBA depend on the sequence of nodes of the lexical tree, so we call such a strategy a lexicon-driven LBA. The problem of carrying out the search through a node that may have several different models is easily incorporated to the LBA.

It is possible to put into the LBA some contextual information as a means to adapt it to a particular application. In our case, we are interested in recognizing unconstrained handwritten words, so some information about the writing style may help to improve the recognition accuracy. For example, it is more likely that a writer starts to write a word using an uppercase character and continue in uppercase or change to lowercase, than that starts in lowercase and change to uppercase. From the learning database it is possible to estimate the probability of a word to start

with a specific writing style, as well as the probability of keeping or changing the writing style within the word. This contextual information contributes to improve the recognition accuracy [35].

Although the LBA has been presented before for statistical models like HMMs [141], the presentation is always based on simple left-right models with forward and self-loop transitions. Its extension to more complicated models with null transitions and observations emitted along transitions is not so trivial. So, we present the complete procedure for finding the maximum likelihood state sequence of a word built up from character HMMs and considering that the HMMs have null transitions, observations are emitted along transitions, and the logarithm of the model parameters (known as *likelihoods*).

1) *Initialization*: For $l = 1$, $t = 1$ and $j = 1$ we have:

$$\delta_t(l, j) = 0 \quad (4.5)$$

where $\delta_t(l, j)$ accumulates the likelihood scores for each frame t , state j , and position l of the model within the word (or level of the LBA). However, the null transitions must be initialized also for $l = 1$, $t = 1$, and $j = 2, 3, \dots, N$ as:

$$\delta_t(l, j) = \max_{1 \leq i < j} [\delta_t(l, i) + a_{ij}^{\Phi} + ctr(l, i)] \quad (4.6)$$

where a_{ij}^{Φ} is the probability to pass from a state i at frame t to a state j at frame t , and producing a null observation symbol Φ , N is the number of states in the model, and $ctr(l, i)$ is the probability of changing or keeping the same writing style within the word, and it is defined as:

$$ctr(l, i) = \begin{cases} \tau(0, \lambda_l) & \text{if } l = 1 \text{ and } i = 1 \\ \tau(\lambda_{l-1}, \lambda_l) & \text{if } l > 1 \text{ and } i = 1 \\ 0 & \text{otherwise} \end{cases} \quad (4.7)$$

where τ is the writing style transition probability and it depends on the position of the character within a word.

For higher levels the initialization differs slightly since we must take into account the information provided by the preceding level ($l - 1$). At levels ($l > 1$) we must pick up the likelihood score at the most suitable observation frame from the previous level ($l - 1$). For $l = 2, 3, \dots, L$, $t = 1, 2, \dots, T$ and $j = 1$, we have:

$$\delta_t(l, j) = \delta_t(l - 1, N) \quad (4.8)$$

where T is the length of the observation sequence, and L is the number of concatenated characters that form a word. It also corresponds to the number of levels of a specific branch of the lexical tree.

To allow the best backtracking path, a new back pointer array (α) is introduced to record the observation frame (t) at the previous level ($l - 1$) in which the character ended. For $l = 2, 3, \dots, L$, $t = 1$ and $j = 1$ we have:

$$\alpha_t(l, j) = 0 \quad (4.9)$$

For all other observation frames ($t = 2, 3, \dots, T$) we have:

$$\alpha_t(l, j) = t \quad (4.10)$$

2) *Recursion*: For $l = 1, t = 2, 3, \dots, T, j = 1, 2, \dots, N$ and considering the presence of null transitions, we have:

$$\delta_t(l, j) = \max \left\{ \max_{1 \leq i < j} [\delta_{t-1}(l, i) + a_{ij}^{O_{t-1}}] ; \max_{1 \leq i < j} [\delta_t(l, i) + a_{ij}^{\Phi}] \right\} \quad (4.11)$$

where $a_{ij}^{O_{t-1}}$ is the state transition probability distribution for which a_{ij} is the probability to pass from a state i at frame $t-1$ to a state j at frame t , and producing an observation symbol $O_{t-1} = z_j$, where $z_j \in \mathcal{Z} = \{z_1, z_2, \dots, z_G\}$ is the discrete set of possible observation symbols, and G is the number of distinct observation symbols.

During the recursion the back pointer $\alpha_t(l, j)$ is updated for $l = 1, 2, \dots, L, t = 2, 3, \dots, T$ and $j = 2, 3, \dots, N$ as:

$$\alpha_t(l, j) = \begin{cases} \alpha_t \left\{ l, \arg \max_{1 \leq i < j} [\delta_t(l, i) + a_{ij}^{\Phi}] \right\} & \text{if } ij \text{ is null} \\ \alpha_{t-1} \left\{ l, \arg \max_{1 \leq i < j} [\delta_{t-1}(l, i) + a_{ij}^{O_{t-1}}] \right\} & \text{otherwise} \end{cases} \quad (4.12)$$

For higher levels ($l > 1$), $\alpha_t(l, j)$ is also computed by Equation 4.12. However, $\delta_t(l, j)$ is computed by Equation 4.11 with a slight difference: now, only the states greater than 1 must be considered ($j = 2, 3, \dots, N$) since for $j = 1$, $\delta_t(l, j)$ was already computed by Equation 4.8.

3) *Termination*: For $l = 1, t = 1, 2, \dots, T$ and a given a model λ , we have:

$$\begin{aligned} P_t(l, \lambda) &= \delta_t(l, N) \\ B_t(l, \lambda) &= 0 \end{aligned} \tag{4.13}$$

For higher levels ($l = 2, 3, \dots, L$), $P_t(l, \lambda)$ is also computed by Equation 4.13, but $B_t(l, \lambda)$ now is given by:

$$B_t(l, \lambda) = \alpha_t(l, N) \tag{4.14}$$

At the output of the level we store the resulting probabilities in an array P , which is a function of the level, observation frame, and the character model. The array B stores the backtrack pointer for each frame and level. At level $l = 1$ and at higher levels ($l > 1$) we cycle multiple models of a character class (uppercase, lowercase, contextual dependent) in the manner described above, or once in case of character classes with a unique model.

4) *Level Reduction*: At the end of each level when all appropriate models have been used, we level reduce to form the array P_t^* . For all l and t we have:

$$P_t^*(l) = \max_{\lambda} [P_t(l, \lambda)] \tag{4.15}$$

where P_t^* is the best level output probability.

The above equation searches the model λ at level l that gives the highest likelihood at each frame t . The level output back pointer for all l 's and t 's is given by:

$$B_t^*(l) = B_t \left\{ l, \arg \max_{\lambda} [P_t(l, \lambda)] \right\} \tag{4.16}$$

where B_t^* is the level output back pointer.

Finally, we keep the output model λ for each level (l) that gives the highest likelihood score for each level and frame:

$$W_t^*(l) = \arg \max_{\lambda} [P_t(l, \lambda)] \quad (4.17)$$

where W_t^* is the level output character indicator.

In summary, the relation among words, characters, tree nodes, character HMMs and levels of the LBA can be simply stated: each word is composed of a sequence of L characters that are represented by a sequence of L linked nodes in the lexical tree. During the decoding, each tree node can be viewed as a level of the LDLBA where a character class can have more than one model. At the end, for each tree node the arrays P_t^* , B_t^* , and W_t^* are stored. Figure 29 illustrates the decoding of a word from the lexicon, given an observation sequence O .

The result of the LDLBA search is a list with the multiple recognition hypotheses. For each word hypothesis, we have a probability score, a state lattice that contains all characters that form the recognized word which is readily converted into an ASCII label that represents the recognized word and the segmentation of the word into characters.

4.1.5.1 Computational Complexity and Storage Requirements of the LDLBA

The computational complexity of the LDLBA is $\mathbb{T}\mathbb{N}^2$ for an elementary N -state HMM. However, we have the concatenation of \mathbb{L} characters and each character may have \mathbb{H} models. Table X shows the computational complexity to decode the whole

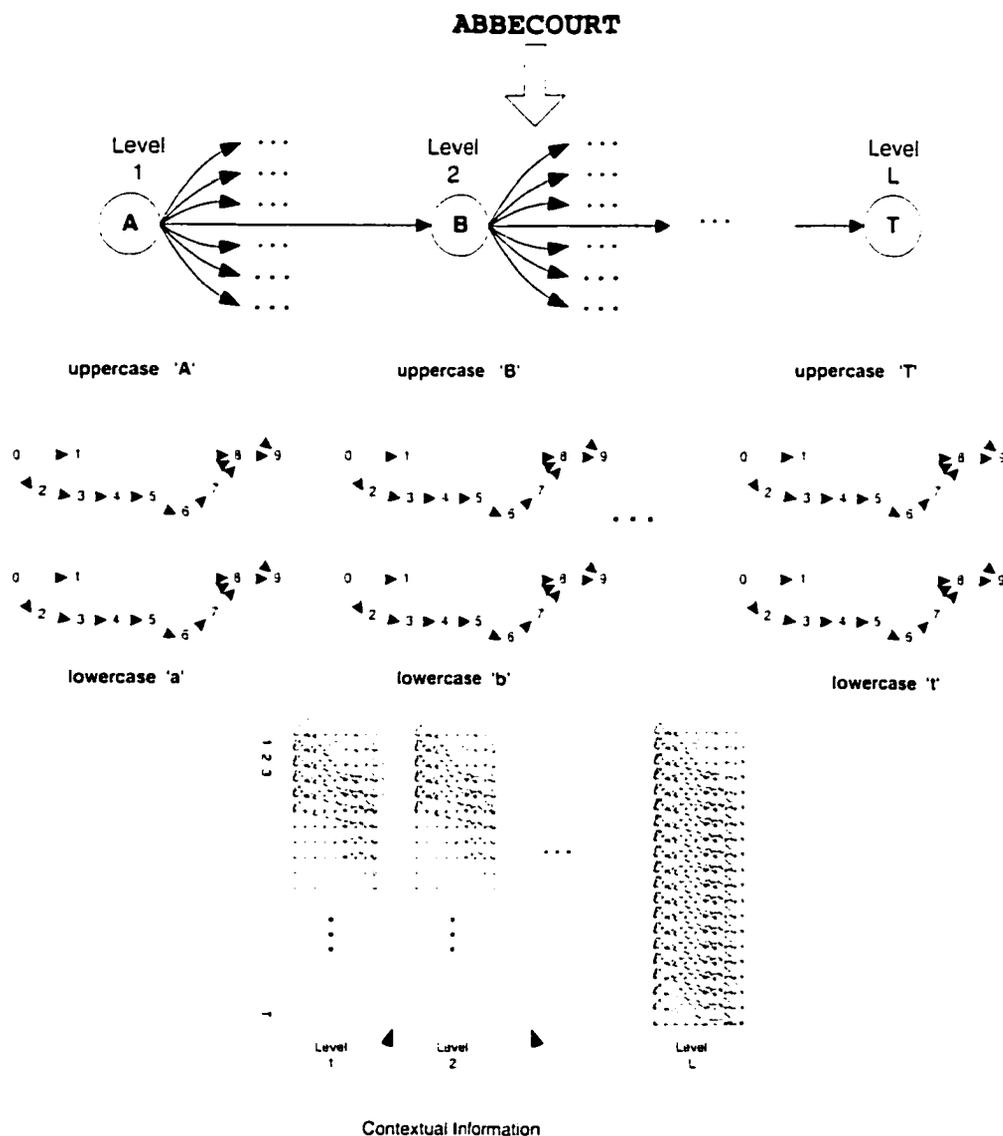


Figure 29 A simplified overview of the unified structure. The word *ABBECOURT* is represented in the lexical tree by a sequence of linked characters (nodes) that may be shared by other words. The characters are represented by one or more HMMs (e.g. “A” and “a”). The LBA finds at each level which is the best model (“A” or “a”) for each t as well as the overall combination of models that best matches the entire sequence of observation (e.g. *Abbecourt*)

lexicon of size \mathbb{V} .

The storage requirements refer to the arrays that are used during the decoding, that is, those that keep the probabilities and the best states. It requires the storage of only three arrays that keep the best states (B^*), probabilities (P^*), and best model (W^*). Furthermore, it is independent of the number of character class models (H). Additionally, it is necessary to keep the arrays that are required to decode the whole lexicon \mathbb{V} , that is, B^* , P^* , and W^* for the prefixes previously decoded and that are shared with other words. Table X shows the storage requirements to decode the whole lexicon considering the parallel recognition model, where $L' < L$ is the average number of characters in words with shared prefix.

Table X

A summary of the approximate computational complexity and storage requirements for the LDLBA

Computational Complexity	Storage Requirements
$\mathcal{O}(HTN^2L'\mathbb{V})$	$2HTN + 2HT + 3L'T\mathbb{V}$

4.1.5.2 Summary of the LDLBA

One of the main advantages of the LDLBA is that it is very easy to add contextual information during the search, as well as multiple models for each character class (e.g. contextual dependent models) without a significant increase in the computational complexity. Figure 30 shows an example of multiple models for the character "M". If the level reduction is not carried out as shown in Figure 30a, each model would generate a tree branch to be decoded during the recognition (Figure 30b) and that would be very time-consuming.

On the other hand, the main drawback of the LDLBA is that it gives a sub-optimal

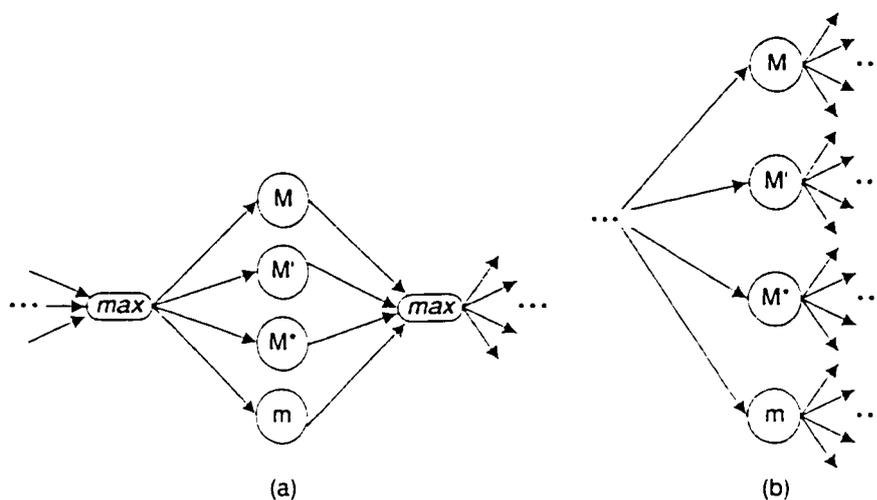


Figure 30 The inclusion of several models for a unique character class in parallel: (a) with level reduction only the model with highest likelihood is expanded. (b) without level reduction all models are expanded

solution due to the level reduction and as a consequence, a reduction in recognition accuracy is expected [127]. The sub-optimal solution arises from the level reduction operation, that is, at the end of each level we choose one among the multiple models (e.g. uppercase or lowercase character models), taking the one that gives the best likelihood score at each observation frame. As a consequence, for the subsequent levels ($l + 1$), only the character model that gives the highest likelihood scores at level (l) will be considered. On the other hand, with the Viterbi search, all models are kept and the decision is taken only after the last character model of the word is decoded.

A close analysis of the errors of the LDLBA indicates that the majority of the errors ($\approx 83\%$) occurs on relatively short words. For long words, even if wrong local decisions are taken, that is, the best model is wrongly selected at the level reduction, the final probability of the words is not severely affected since the errors are minimized along the decoding, while for short words, wrong local decisions are

more meaningful.

Notice that there are no parameters to adjust either to improve the accuracy or the recognition speed. However, some applications may require a higher accuracy or real-time processing. This is one of the limitations of the LDLBA, that is, it is not possible to define a better trade-off between speed and accuracy to satisfy the requirements of particular applications.

The evaluation of the performance of the lexicon-driven level building algorithm implementation and its comparison with the performance of the baseline recognition system is presented in Chapter 5.

4.1.6 Time and Length Constraints

Even though the computational complexity for the level building approach seems to be significantly less than the Viterbi approach of the baseline system presented in Section 3.7.2, there are several ways of further reducing the computational load of the algorithm. To such an aim we can rely on the particular characteristics of our application:

- We are decoding words from a lexicon from which length is known. So, the number of the levels l of the LBA is known a priori;
- The words are formed by the concatenation of character HMMs which architecture is associated with the shape of the characters;
- The high level features used to generate the sequence of observations imply that characters are usually represented by few observations:

Given these three remarks, it is clear that the conditions for the recursion of the LBA (Equation 4.11) are not realistic since it is carried out for almost the entire sequence of observations ($t = 2, 3, \dots, T$) at almost all levels ($l = 2, 3, \dots, L$). In

other words, this means that the whole observation sequence that represents a whole word is matched against individual character models. The reason for doing that is due to the difficulties related to the segmentation of words into characters, we do not know the boundaries of characters, that is, which part of the sequence of observations correspond to each character. However, since we are relying on a lexicon during the decoding and the search is carried out from left to right, we can estimate approximately the character boundaries and the alignment of the whole observation sequence with the HMMs at each level can be relaxed. Another point is that short observation sequences are more likely to be generated from short words. Therefore, there is not much sense in aligning short observation sequences with long words. Nevertheless, if we consider the architecture of the character model used in the baseline recognition system in which the number of observations that it can emit is very well determined, the limits to the search can be easily established.

Therefore, the two hypotheses to further improve the performance of the LDLBA presented in Section 4.1.5 are: by constraining the number of levels of the LDLBA (*length*) according to the length of the observation sequences; by limiting the number of observations aligned at each level (*time*).

4.1.6.1 Time Constraint

The time constraint concerns the limitation of the number of observations aligned at each level of LDLBA. We introduce two variables: $s(l)$ and $e(l)$. The former denotes the index of the first observation for which valid paths to a given level can begin, while the latter denotes the index of the last observation for which valid paths to a given level can end. Both variables are functions of the level (l). Figure 31 shows how these two constraints are incorporated to the levels of the LDLBA.

To incorporate these two constraints into the level building search, the equations of the LDLBA do not need to be modified, since the constraints are observation

indexes. Therefore, only the limits of the observation index t is modified as follows.

$$t = s(l), s(l) + 1, \dots, e(l) - 1, e(l) \quad (4.18)$$

where $1 \leq s(l) \leq T$, $1 \leq e(l) \leq T$, $1 \leq l \leq L$, and for a left-to-right HMM, $e(l) \geq s(l)$. Furthermore, both $s(l)$ and $e(l)$ must be positive integers and they are given by Equations 4.19 and 4.20 respectively:

$$s(l) = \begin{cases} 1 & \text{if } l=1 \\ l - s^* & \text{if } l > 1, s^* \geq l. \end{cases} \quad (4.19)$$

$$e(l) = \begin{cases} D_{max} \cdot [l + e^*] & \text{if } D_{max} \cdot [l + e^*] < T \\ T & \text{otherwise.} \end{cases} \quad (4.20)$$

where s^* and e^* are the two control factors to be determined and D_{max} is the maximum model duration that is associated with the HMM architecture.

4.1.6.2 Length Constraint

The length constraint concerns the limitation of the number of levels of the LBA. We introduce the variable Lv which denotes the maximum number of levels of the LBA, given an observation sequence with length T (Figure 31). To incorporate this constraint to the decoding, the equations of the LBA are preserved, but the range of the variable l that denotes the level index, is modified slightly. Now, instead of ranging from levels 1 to L , the range will be given by:

$$l = 1, 2, \dots, Lv, \quad 1 < Lv \leq L. \quad (4.21)$$

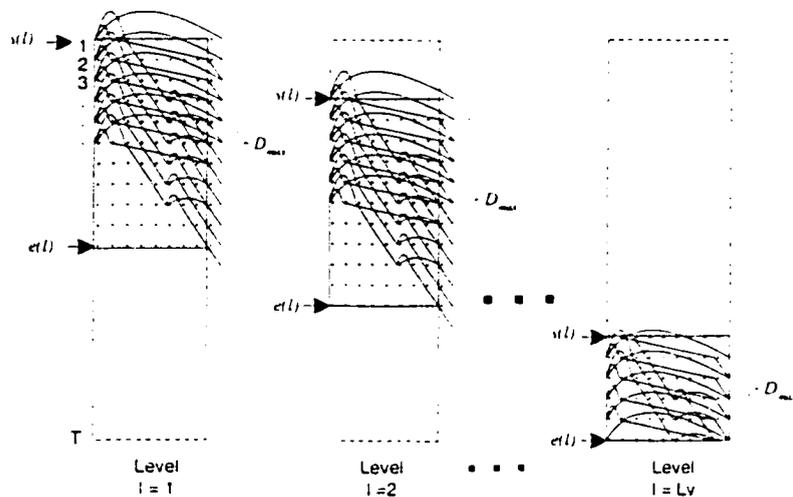


Figure 31 Levels of the LDLBA incorporating the time constraints $s(l)$ and $e(l)$ that limit the number of observations aligned at each level, and the length constraint L_v that limits the number of levels of the LDLBA

where L_v is an integer given by Equation 4.22 and its lower and upper limits are given by the shortest and the longest word in the lexicon respectively.

$$L_v = TLv^*, \quad Lv^* > 0. \quad (4.22)$$

where Lv^* is the control factor to be determined.

The control factors s^* , e^* , and Lv^* have to be adjusted to jointly optimize the system performance, that is, find the best trade-off between recognition accuracy and speed. Several approaches exist for optimization. A long established technique involves changing the value of each control factor in turn in an attempt to determine the effect of each on the response. Response surface techniques, hill climbing algorithms and genetic algorithms may also be employed. However most of these techniques are either inefficient or require a large number of experiments. To minimize the number

of experiments and at the same time obtain a satisfactory performance, we have used a statistical experimental design technique [10] that studies the effects of the multiple variables simultaneously and optimizes the performance of the constrained recognition system. The application of the statistical experimental design technique to determine the control factors and optimize the recognition system is described in Appendix 1 as well as in [86].

Table XI shows an example of the search limits imposed by the CLBA for an input word “St Malo”, given an observation sequence with 12 observations and considering that the lexicon word being decoded is the truth word hypothesis. Figure fig:contrapp shows the reduced search space delimited by the CLBA (in gray).

Table XI

An example of the search limits for the CLBA and CLDLBA for the word “St_Malo” and a 12-observation sequence

Parameter	Characters						
	S	t	-	M	a	l	o
l	1	2	3	4	5	6	7
s^*	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5
e^*	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$s(l)$	1	2	3	4	5	6	7
$e(l)$	3	9	13	13	13	13	13
$s(l, \lambda)$	1	2	2	3	5	6	9
$e(l, \lambda)$	4	6	7	12	13	13	13

4.1.6.3 Contextual Time and Length Constraints

The constraints introduced to the LBA depend only on l and t . They do not depend on the character classes. However, it would be advantageous to make these constraints dependent on the character classes. The three constraints can be associated with the shape of the characters and the with the behaviour of the segmentation

algorithm used in the pre-processing step to loose segment words into characters.

Table XII shows the probability of splitting characters into 0, 1, 2, or 3 and more segments for each character class. As expected, the probability of segment a character from class "I" into a segment is very high (91%), while for the character class "N" the probability is very low ($\leq 8\%$). If during the decoding we make use of such knowledge, we can limit the number of observations at each level, according to the character class that is being decoded.

Now, we make $s(l, \lambda)$ and $e(l, \lambda)$ depend on the character model (λ) and they are given by Equations 4.23 and 4.24 respectively:

$$s(l, \lambda) = \begin{cases} 1 & \text{if } l = 1 \\ l - s^*(\lambda) & \text{if } l > 1, s^*(\lambda) \geq l. \end{cases} \quad (4.23)$$

$$e(l, \lambda) = \begin{cases} D_{max}(\lambda) \cdot [l + e^*(\lambda)] & \text{if } D_{max}(\lambda) \cdot [l + e^*(\lambda)] < T \\ T & \text{otherwise.} \end{cases} \quad (4.24)$$

where $s^*(\lambda)$ and $D_{max}(\lambda)$ are now the two control factors to be determined and $D_{max}(\lambda)$ is the maximum model duration that is associated with the HMM architecture.

The control factors $s^*(\lambda)$ and $D_{max}(\lambda)$ have to be adjusted to jointly optimize the system performance, that is, find the best trade-off between recognition accuracy and speed. However, due to the high number of parameters to adjust, now it becomes difficult to use the statistical experimental design technique, and we have adjusted the value of each control factor in turn in an attempt to determine the effect of each factor on the response. In fact, we modify slightly the parameters which are initially set by the CLBA to adapt them to the particular characteristics of each

Table XII

Number of segments resulting from the loose segmentation of words into characters or pseudo-characters for each character class [32]

Character Class	Probability of splitting a character into N segments				Character Class	Probability of splitting a character into N segments			
	0	1	2	≥ 3		0	1	2	≥ 3
A	0.0033	0.8659	0.1248	0.0060	a	0.0171	0.7592	0.2146	0.0091
B	0.0006	0.8195	0.1550	0.0249	b	0.0019	0.4678	0.5152	0.0152
C	0.0057	0.9035	0.0840	0.0068	c	0.0136	0.9417	0.0431	0.0016
D	0.0141	0.7887	0.1807	0.0165	d	0.0028	0.4529	0.5135	0.0308
E	0.0299	0.8132	0.1481	0.0088	e	0.0448	0.9097	0.0433	0.0023
F	0.0374	0.7533	0.1938	0.0154	f	0.0221	0.9265	0.0515	0.0000
G	0.0020	0.2953	0.6731	0.0295	g	0.0014	0.7037	0.2828	0.0122
H	0.0026	0.0416	0.9104	0.0455	h	0.0075	0.3403	0.6254	0.0269
I	0.0758	0.9130	0.0104	0.0008	i	0.0933	0.8809	0.0259	0.0000
J	0.0047	0.7465	0.2488	0.0000	j	0.0132	0.9342	0.0526	0.0000
K	0.0000	0.1667	0.8333	0.0000	k	0.0000	0.6000	0.4000	0.0000
L	0.0080	0.7994	0.1857	0.0069	l	0.0507	0.9157	0.0316	0.0020
M	0.0011	0.2493	0.6351	0.1145	m	0.0033	0.0440	0.2692	0.6835
N	0.0018	0.0856	0.8681	0.0444	n	0.0096	0.2730	0.6832	0.0342
O	0.0290	0.9706	0.0004	0.0000	o	0.0342	0.8528	0.1114	0.0016
P	0.0190	0.8709	0.1071	0.0030	p	0.0000	0.6263	0.3636	0.0101
Q	0.0000	0.6992	0.1870	0.1138	q	0.0000	0.6329	0.3671	0.0000
R	0.0040	0.8469	0.1407	0.0084	r	0.0370	0.8327	0.1257	0.0046
S	0.0198	0.8624	0.1163	0.0015	s	0.0327	0.9230	0.0440	0.0003
T	0.0863	0.7993	0.1058	0.0086	t	0.1241	0.6898	0.1813	0.0048
U	0.0034	0.1296	0.8581	0.0089	u	0.0046	0.0845	0.8742	0.0367
V	0.0017	0.2376	0.7563	0.0043	v	0.0019	0.2486	0.7401	0.0094
W	0.0000	0.0000	0.2414	0.7586	w	0.0000	0.0000	0.1667	0.8333
X	0.0075	0.2728	0.7134	0.0063	x	0.0084	0.3382	0.6131	0.0403
Y	0.0044	0.2529	0.7338	0.0088	y	0.0043	0.1481	0.8219	0.0256
Z	0.0090	0.9186	0.0724	0.0000	z	0.0211	0.8211	0.1579	0.0000

character class. Table XI shows the search limits modified by the CDCLBA. Figure fig:contrapp shows the reduced search space delimited by the CDCLBA (surrounded by the black line).



Figure 32 Reduction in the search space for a 7-character word (ST.MALO) and a 12-observation sequence. The search space delimited by the CLBA is shown in gray, while the line surrounding the search space represents the region delimited by the CDCLBA

4.1.6.4 Computational Complexity and Storage Requirements for the CLBA and the CDCLBA

The computational complexity of the CLBA and CDCLBA are basically the same and it is difficult to establish it exactly since both depend on heuristics to reduce some of the elements that are involved in the number of computations.

Table XIII shows the computational complexity and the storage requirements of the CLBA and CDCLBA algorithms. Basically, the computational complexity is given by $T'N^2$, where $T' < T$ denotes the reduced number of operations due to the

time constraint, combined with $L'' < L'$ and H . The factor L'' denotes the reduced number of operations due to the length constraint. The storage requirements are the same of the LDLBA presented in Table X.

Table XIII

A summary of the approximate computational complexity and storage requirements for the CLBA and the CDCLBA

Computational Complexity	Storage Requirements
$\mathcal{O}(HT'N^2L''V)$	$2HTN + 2HT + 3L'TV$

4.1.6.5 Summary of the the CLBA and the CDCLBA

The use of the constraints allows the reduction of the search efforts, however, this method involves the optimization of several parameters which also entail a certain error. Furthermore, the constraints do not take into account the particularities of each character.

The use of constraints dependent on the character models can recover some accuracy that was lost by using global constraints and at the same time speedup the recognition process. On the other hand, the main drawback is the number of parameters to be adjusted. The optimization of the control factors is much more complicated and it is very difficult to use a statistical method to such an aim.

The evaluation of the performance of the constrained lexicon-driven level building algorithm and contextual dependent constrained lexicon-driven level building algorithm implementations and their comparison with the performance of the baseline recognition system and the LDLBA implementation are presented in Chapter 5.

4.1.7 Fast Two-Level Decoding Strategy

In Sections 4.1.5 and 4.1.6 we saw that although the level building algorithm together with the lexical tree reduces the computational complexity of the recognition process, there is also a possible degradation in the recognition accuracy relative to the baseline SRTP recognition system. This degradation could be attributed to the level reduction in the case of the LDLBA and also to the heuristics that limit the search in the case of the CLBA and CDCLBA.

However, we have also observed that there is still a great number of repeated computation during the decoding of the words in the lexicon. Therefore, it is possible to reduce further the computational complexity of the recognition process.

In this section we present a simple and efficient algorithm to search the lexical tree for a optimum decoding equivalent to the Viterbi implementation of the baseline recognition system. This novel fast search strategy breaks up the computation into two levels: state level and character level. Given an observation sequence, at the first level character models are decoded individually for each possible entry and exit points, and the results are stored into two arrays to further use. At the second level, words are decoded based on the previously pre computed character models without the necessity of decoding state sequences, but only character boundaries. We show that this algorithm brings the recognition accuracy to that of the baseline recognition system, at a reduced computational complexity.

4.1.7.1 Principles of the Fast Two-Level Decoding Strategy

The words in the lexicon are formed by the concatenation of alphabetic characters and each character class is modeled by at least an HMM. This so called “character HMMs” are global models, that is, they do not dependent on the words.

Considering the character “*a*” that appears repeatedly within the words of the five-

word lexicon shown in Figure 33. Assuming that it is modeled by a character HMM denoted as λ_a , during the decoding, each appearance of the character “a” in the words is always replaced by the same model λ_a . So, given an observation sequence and the five-word lexicon shown in Figure 33a, the character “a” will be replaced by λ_a eight times, and it will be equally decoded eight times. However, if we consider the tree representation of the lexicon as shown in Figure 33b, the prefix “la” is shared by all the words and consequently the character “a” appears only three times. So, it will be replaced by λ_a three times, and it will be equally decoded only three times.

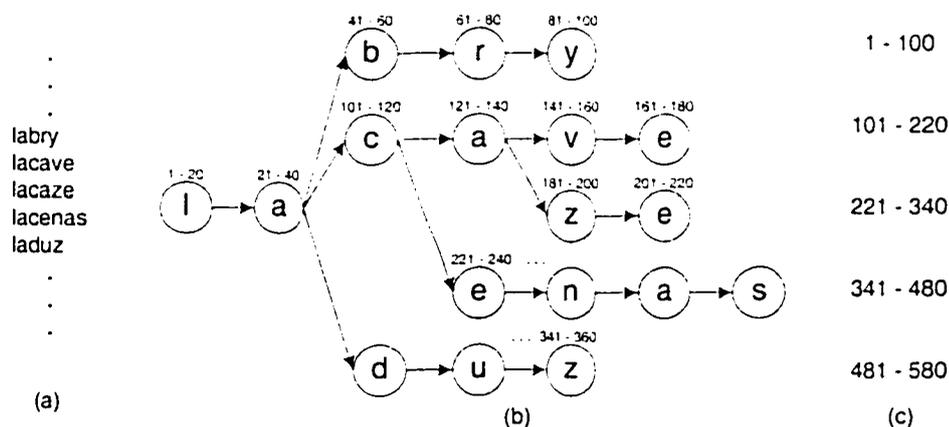


Figure 33 (a) A small lexicon with 5 words, (b) Organization of the lexicon as tree structure with the computation order for the time-asynchronous search, (c) Computation order for each word for the conventional time-synchronous Viterbi search

The question that one may pose is: “During the recognition, is it possible to decode λ_a only once, since it is always the same character model?”. The current decoding algorithms are no able to do that and the reason is that they decode an observation sequence in a time-synchronous fashion. Considering that a word is formed by the concatenation of HMM models as $\lambda_1 \oplus \lambda_2 \oplus \dots \oplus \lambda_l \oplus \dots \oplus \lambda_L$, under the conventional Viterbi algorithm it is not possible to calculate the Viterbi score of an observation

sequence for a character (λ_l) within the word without relying on the Viterbi score of the immediate preceding character (λ_{l-1}), even if the same character has appeared previously within the word or within another word for which the Viterbi score was already calculated [145].

Considering the lattice for a 3-character word formed by the concatenation of character HMMs shown in Figure 34, the initial probabilities of the second character ($l = 2$) depend on the final probability of the preceding character ($l = 1$), so the final probability of the second character depends on the probability of the preceding character, and so on, because the probabilities computed by the Viterbi algorithm are cumulative along the lattice. This dependence on the probabilities of the immediate preceding character is given in Equation 3.5 in Chapter 3. For this reason, even if the character HMMs do not depend on the words (position, neighbors characters, etc.) the probabilities computed along the model do. This implies the necessity of having multiple copies of the same character HMMs through the search space to account for the different contexts, say, different positions within a word due to the different preceding and succeeding characters. As a consequence, it is difficult to avoid the large number of copies and the repeated computation of best state sequences.

This problem was partially solved by the tree-structured lexicon and the LDLBA presented in Section 4.1.5 because the computation for the common prefixes is carried out only once and shared by all words with the same prefix, but it is still necessary to repeat the decoding of character HMMs for the remainder of the words.

To solve this problem and avoid repeated computation of the same HMMs and observation sequences we propose a new algorithm that breaks up the computation into two levels. At the first level character HMMs are decoded considering each possible entry and exit point and the result (best state sequences and probabilities) is stored in an array for further use. At the second level words are decoded but

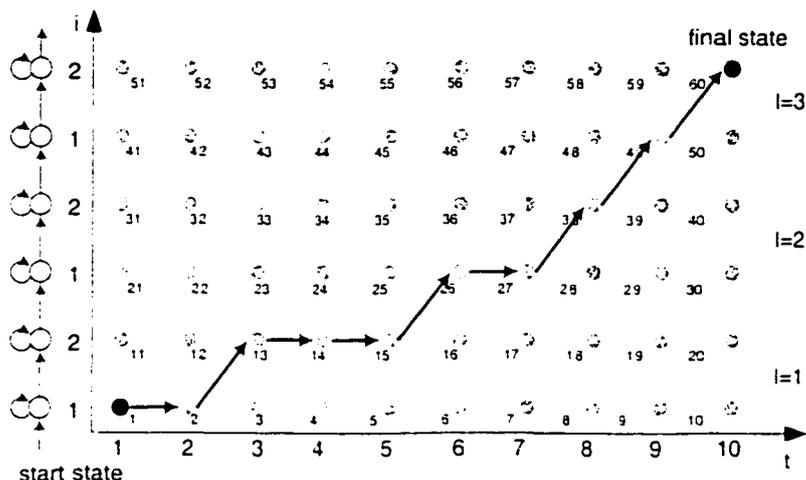


Figure 34 A 3-character word HMM formed by the concatenation of three 2-state character HMMs, and the corresponding lattice structure for a 10-observation sequence. The gray arrows indicate the possible state transitions, the black arrows indicate the best path, and the numbers indicate the computation order

reusing the results of the first level, say, the best state sequences and probabilities. So, only the character boundaries are decoded without the necessity of going through the HMM states. The details of each level are presented as follows.

4.1.7.2 First Level: Decoding of Character Models

The idea underneath the proposed search strategy lies in avoiding repeated computation of the best state sequences of sub-word HMMs. Given a set of sub-word models $\mathcal{C} = \{\lambda_1, \lambda_2, \dots, \lambda_M\}$ where λ_m models character classes (letters, digits, and symbols) and a sequence of observations $O = (o_1 o_2 \dots o_T)$, at the first level we evaluate the matching between O and each λ_m individually. To do that, we assume that each character HMM has only one initial state (or entry state) and only one final state (or exit state) as denoted in Figure 35 and we compute the best state sequences between initial and final states, considering one possible beginning frame

at each time, which is denoted as s . From this beginning frame s we compute the best state sequences between initial and final states for all possible ending frames, which is denoted as e . Furthermore, we store in an array the best state sequences and probability scores of all pairs of beginning and ending frames (s, e) .

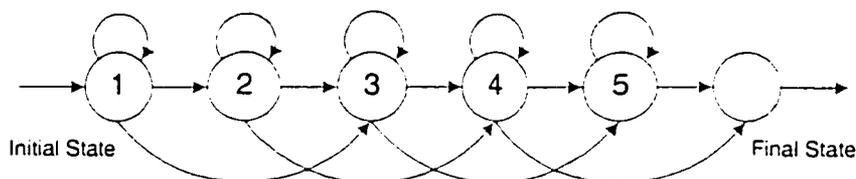


Figure 35 An HMM topology with a unique initial state and a unique final state

The complete description of the procedure to decode all character models is described as follows:

1. For a given sequence of observations $O = (o_1 o_2 \dots o_T)$;
2. Pick up a character model λ_m which has an initial state denoted as i and a final state denoted as f ;
3. Consider only a valid entry point, denoted as (s, i) at each time and compute all possible best state paths to the final state f using the Viterbi algorithm;
4. For each time e that the final state f is reached, store in an array $\chi(s, e)$ the accumulated probability at f ;
5. Find the best state sequence by backtracking from f to i and store it in an array $\epsilon(s, e)$;
6. Repeat from Step 4 until $s = T$;
7. Pick up another s and repeat from Step 3 until $s = T$;
8. Pick up another λ_m and repeat from Step 2 until $m = M$;
9. End.

Figure 36 illustrates the procedure presented above to decode an observation sequence with seven observations and a three-state HMM which initial state is $i = 1$ and final state is $i = 3$. Note that for this HMM topology some of the paths are

omitted because according to the HMM architecture they do not exist. In Figure 36a the decoding starts for $s = 1$ and the final state is reached at $e = 3, 4, \dots, 7$. So, the probability of each path from $i = 1$ to $i = 3$ for each beginning/ending frame (s, e) is stored in an array $\chi(s, e)$ as shown in Figure 36b. The best state sequences recovered by the backtracking procedure are also stored in an array $\epsilon(s, e)$ as shown in Figure 36c. Figure 36d shows two examples of the backtracking for $(e, j) = (6, 3)$ and $(e, j) = (7, 3)$. The array q_t^* keeps the best state at each t . An additional array, $\epsilon(s, e)$ is required to keep the whole state sequence.

Now that we have a good understanding of the steps carried out at the first level, we can present the complete decoding algorithm for finding the best state sequences of each character HMM, assuming that observations are emitted at states, as follows:

1. *Initialization:* for $1 \leq s \leq T$, $i = 1$

$$\begin{aligned}\delta_s(i) &= \pi_i b_i(o_s) \\ \omega_s(i) &= 0\end{aligned}\tag{4.25}$$

2. *Recursion:* for $s < t \leq T$, $1 \leq j \leq N$

$$\begin{aligned}\delta_t(j) &= \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t), \\ \omega_t(j) &= \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}]\end{aligned}\tag{4.26}$$

3. *Termination:* if $j = N$

$$\begin{aligned}e &= t \\ \chi(s, e) &= \delta_t(N) \\ q_e^* &= N\end{aligned}\tag{4.27}$$

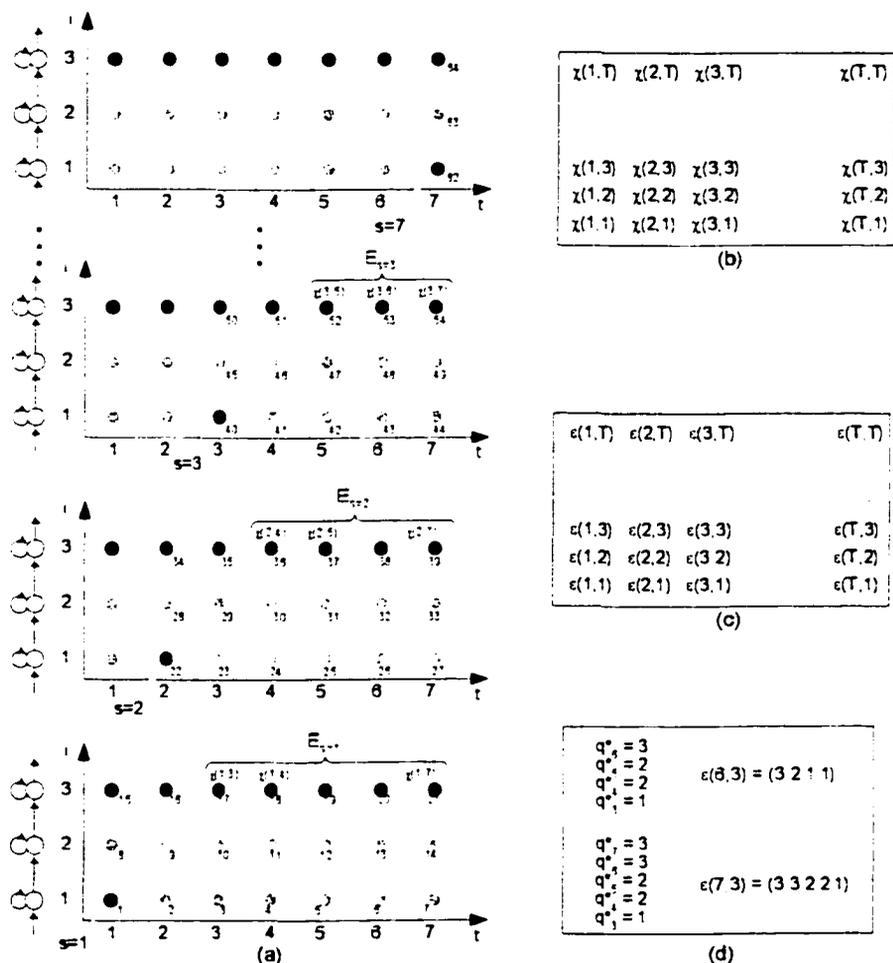


Figure 36 (a) Computation of the probabilities for a single 3-state character HMM for each entry point s . (b) The resulting forward probability array $\chi(s, e)$ that models single characters for a specific observation sequence. (c) The resulting best state sequence array $\epsilon(s, e)$. (d) Example of backtracking for two exit points $(s, e) = (3, 6)$, and $(s, e) = (3, 7)$. The resulting MAP state sequence is stored in the array $\epsilon(s, e)$

4. *Backtracking:* for $t = e - 1, e - 2, \dots, s$, q_t^* is determined recursively by:

$$\begin{aligned} q_t^* &= \omega_{t+1}(q_{t+1}^*) \\ \epsilon(s, e) &= [q_e^* q_{e-1}^* q_{e-2}^* \dots q_s^*] \end{aligned} \quad (4.28)$$

So, for a given observation sequence O we end up with M arrays that keep the best probability scores $\chi(s, e)$ and M arrays that keeps the best state sequences $\epsilon(s, e)$. Doing that, the probability scores of the character models are totally independent of the context (the word within the character may appear) and may be reused unrestrictedly to decode words in a lexicon.

This decoding algorithm can be viewed as the standard Viterbi algorithm but which is used to decode at each time the entire sequence of observations O for a specific range of beginning frames s , $1 \leq s \leq T$, and ending frames e , $1 \leq e \leq T$.

Note that so far any additional source of knowledge (e.g. lexicon, grammar, etc) was used, so this procedure is totally decoupled from context, lexicon, etc. By this process we avoid repeated computation of the best state sequences which may be regarded as the most time-consuming procedure in the recognition process. Once the best state sequences are computed for all character models, they can be reused freely to decode any lexicon because they are *position-independent*, so for this reason we call them, *reusable character models*. Now, the character models can be viewed as building blocks with an associated transfer function where the output probabilities depend only on the values at the input of such blocks (Figure 37).

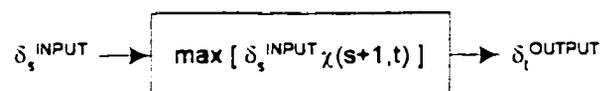


Figure 37 A character model represented as a block with an associated transfer function that maps input probabilities to output probabilities

The next section shows how to use the pre-computed best state sequences of all character models to decode the word in a lexicon.

4.1.7.3 Second Level: Decoding of Words

Considering our primary problem, that is to find the word $w \in \mathcal{R}$ that best matches with the sequence of observations O . Supposing that the words are formed by the concatenation of single character HMMs $(\lambda_1 \oplus \lambda_2 \oplus \dots \oplus \lambda_L)$, now, we no longer have observation sequences and character models but pre-decoded character arrays $\chi_1(s, e) \oplus \chi_2(s, e) \oplus \dots \oplus \chi_L(s, e)$ that hides the notion of observations and states.

Now, let us introduce a source of knowledge, say, a lexicon \mathcal{R} that contains all valid words that can occur at the input of the recognition system. So, we need to concatenate character arrays according to the orthographic form of the words in the lexicon and compute the character probabilities at character boundaries. The probability of each word is readily obtained from $\chi_L(e, T)$. This corresponds to decoding the sequence of observations O through the word $R_e \in \mathcal{R}$ formed by the concatenation of L single character models $R_e = (c_1 c_2 \dots c_L)$.

The decoding of words is carried out from left to right and since words have well-defined initial and terminations, that is, the first and the last word characters, the initialization condition specifies that for the first character of the word represented by the array $\chi_1(s, e)$ the only valid entry point is at $s = 1$. Here we make use of the index l to denote the position of the character models within a word, and now we can throw away the index s and keep only the paths for which $\hat{\delta}_t(l)$ is maximal.

The complete decoding algorithm for the second level is given as follows.

1. *Initialization:* for $1 \leq l \leq L$, $1 \leq t \leq T$:

$$\hat{\delta}_t(l) = \begin{cases} \chi(1, t) & \text{if } l = 1 \\ \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1)\chi(s+1, t)] & \text{if } 2 \leq l \leq L \end{cases} \quad (4.29)$$

$$\hat{\omega}_t(l) = \begin{cases} 1 & \text{if } l = 1 \\ \arg \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1)\chi(s+1, t)] & \text{if } 2 \leq l \leq L \end{cases}$$

2. *Termination: for* $l = L, t = T$:

$$\begin{aligned} \hat{P}^* &= \hat{\delta}_T(L) \\ \hat{q}^*(L) &= \hat{\omega}_T(L) \end{aligned} \quad (4.30)$$

3. *Character Backtracking: for* $L-1, L-2, \dots, 1$:

$$\hat{q}^*(l) = \hat{\omega}_{\hat{q}^*(l+1)}(l) \quad (4.31)$$

4. *State Backtracking: for* $L-1, L-2, \dots, 1$:

$$\begin{aligned} \hat{\epsilon}_L &= \epsilon[T, \hat{q}^*(L)] \\ \hat{\epsilon}_l &= \epsilon[(\hat{q}^*(l+1) - 1, \hat{q}^*(l))] \end{aligned} \quad (4.32)$$

At the initialization step, $\hat{\omega}_t(l)$ records the frame t in which the preceding character ended. To recover the character boundaries, we need to rely on the character output backpointer, $\hat{q}^*(l)$ that records the time t at which the previous character ended and it is determined recursively at the Step 3. Recall that the whole MAP state sequence for each single character was stored in an array $\epsilon(s, e)$. So, to recover the MAP state sequence we need the additional Step 4.

Figure 38 shows an example of word decoding where each array $\chi(s, e)$ models a single character. The start of a word is constrained to the first character and only the probabilities at $s = 1$ are considered, that is, the first row of the array. The remaining characters can start at any t where the precedent character terminates, that is, at all t 's, so we simply multiply the final probabilities of the preceding character $\delta_t(1)$ by all columns of the array $\chi_2(s, e)$ that models the second character, and take only the maximum value for each column. This procedure continues up to the last character of the word ($l = L$).

Figure 38b shows an example of backtracking for a three-character word and a 9-observation sequence. The variable $\omega_t(l)$ keeps the time in which the previous character terminated, (6, 3, and 0), so the corresponding best state sequence can be obtained in the array $\epsilon(s, e)$. The concatenation of the selected best state sequences of the array $\epsilon(s, e)$ gives the MAP state sequence of the word ($\hat{\epsilon}$).

In Appendix 3 the *fast two-level decoding algorithm* is presented considering the transition models and the presence of null transitions.

4.1.7.4 Computational Complexity and Storage Requirements of the Fast Two-Level Decoding Algorithm

In the case of the fast two-level decoding algorithm, the first level is independent of the lexicon, and its computational complexity is given by $\mathbb{T}\mathbb{N}^2$. However, this computation is repeated for each possible beginning frame s , where $1 \leq s \leq T$ and character models. So, the complete computational complexity of the first level is $\mathcal{O}(\mathbb{T}^2\mathbb{N}^2\mathbb{M})$. The first level requires the storage of the arrays $\chi(s, e)$ and $\epsilon(s, e)$ for all character models.

In the second level, the computation depends on the word in the lexicon, and the complexity is $\mathcal{O}(\mathbb{T}^2\mathbb{L}\mathbb{V})$. The second level requires the storage of the arrays δ_t and

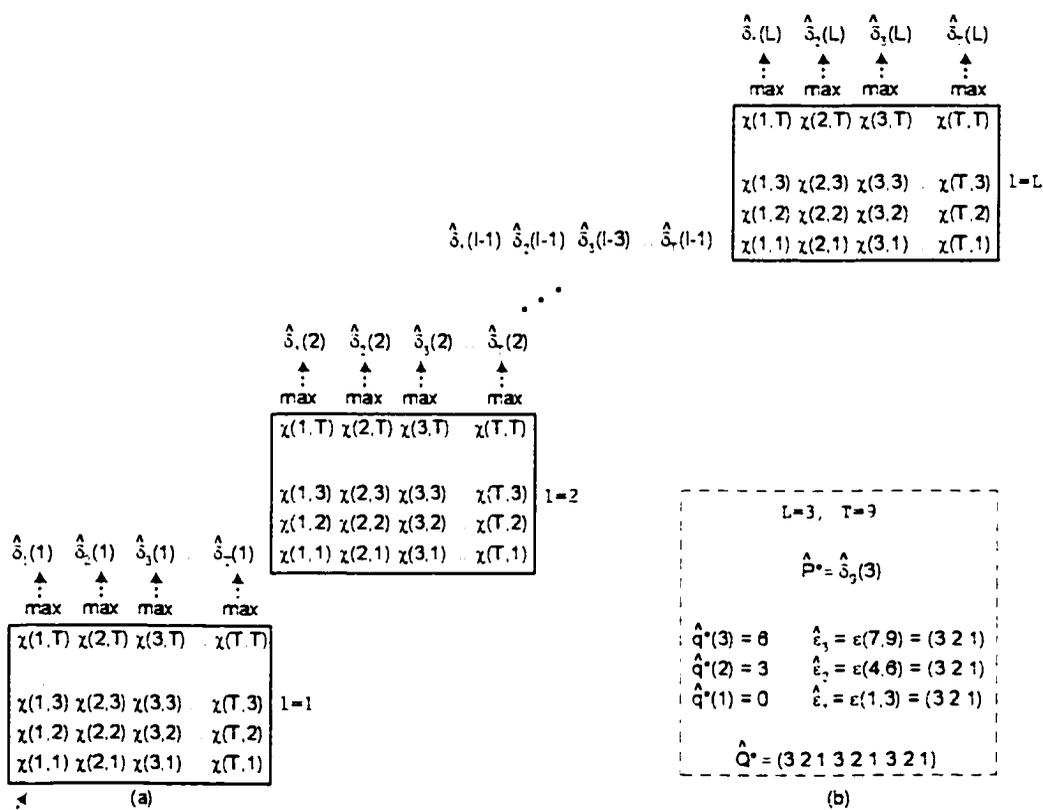


Figure 38 (a) Computation of the probabilities for a word formed by the concatenation of characters represented as building blocks, (b) An example of backtracking for a 3-character word and a 9-observation sequence

ω_t to decode the character boundaries. If a lexical tree is used instead a flat lexicon, we need two additional arrays δ_t and ω_t for the L' shared prefixes. Table XIV shows the approximate computational complexity and storage requirements for the fast two-level decoding algorithm considering a flat lexicon and a tree-structured lexicon.

Table XIV

Computational complexity and storage requirements for the fast two-level decoding algorithm considering a flat structure and a tree-structured lexicon

	Computational Complexity	Storage Requirements
FS Flat	$\mathcal{O}(T^2N^2M + T^2LV)$	$2T^2HN + 2MT^2$
FS Tree	$\mathcal{O}(T^2N^2M + T^2L'V)$	$2T^2HN + 2MT^2 + 3L'TV$

At first glance the fast two-level decoding algorithm seems to have a computational complexity greater than the algorithms presented in the earlier sections because it has two quadratic terms. In fact, the dimension of the variable is also very important. In Section 4.1.9 we compare the computational complexity of the algorithms presented through this chapter and there it will be clear the advantages of the proposed decoding algorithm.

It should be noticed that the results presented in Table XIV are generic and no assumption about the topology of the HMMs was made. To have a more exact estimation of the number of operations we have to know the topology of the HMMs as well as the other characteristics. For specific topologies, such as the left-to-right model it is possible to further reduce the computational complexity.

The algorithm is flexible enough to accommodate such particular characteristics of the HMM topologies, avoiding unnecessary computation. Some simplifications can be introduced to the algorithm according to the topology of the HMMs. For a left-

to-right model. the limits for t and i are $s \leq t \leq T$. and $1 \leq i \leq j$ respectively. Furthermore, if the duration of the models is known the limits of t can be further reduced to $s + D_{min} \leq t \leq s + D_{max}$. where D_{min} and D_{max} are the minimum and the maximum number of observations that can be emitted by a single HMM [86]. In some HMM topologies the duration is explicitly modeled by the HMM topology. notably in those models used in handwriting recognition that do not have self-loop transitions [23, 35, 51, 79].

4.1.7.5 Summary of the Fast Two-Level Decoding Algorithm

The fast two-level search introduces two concepts: modularity and reusability. Conventionally, in an HMM framework, the characters are modeled by a transition ($a_{i,j}$) and observation probability $b_j(o_i)$ that are obtained during the training. However, in the case of fast two-level decoding algorithm the characters are modeled by two arrays $\chi(s, e)$ and $\epsilon(s, e)$ and they can be considered as black-boxes (or building blocks), where the output is a function of the input (Figure 37). These building blocks can be used through the lexicon to account for the different contexts, avoiding repeated computations.

Note that the solution is exactly the same as the one obtained by using a classical Viterbi algorithm. Furthermore, in spite of having based our presentation on state-based HMMs, there is no restrictions, and the same search strategy can be used with transition-based HMMs, topologies including self-loop transitions, log-probabilities (likelihoods), etc. As an example, in Chapter 5 we present a practical implementation of the fast two-level decoding algorithm considering likelihoods, and transition-based models with null transitions. The formulation for the *fast two-level decoding algorithm* for transition-based HMMs and considering the presence of null transitions is presented in Appendix 3.

4.1.8 Extension: A Distributed Recognition Scheme

Our main goal in this section is to explore the potential for speeding up the recognition architecture via concurrency. It is relevant since multiprocessor machines and workstation clusters are becoming very common. To such an aim, one of the more intuitive attempts at speeding up a lexicon-driven handwriting recognition system is to exploit the large number of its algorithm steps. For example, in a lexicon-drive approach, the matching of a sequence of features against different words present in the lexicon can be executed in parallel on different processors. There are several other levels of concurrency in our recognition engine that can be exploited individually and in combination, for example, the computation of the model likelihoods of the fast two-level decoding algorithm presented in Section 4.1.7. However, the most natural way is to distribute the lexicon, partitioning the task of decoding among several processors.

4.1.8.1 Exploiting Concurrency

In the literature, it is hard to find a reference that describes distributed techniques applied to the handwriting recognition problem. Some authors just mention that distributed processing can be employed to achieve a better performance or that multiprocessors are employed to meet throughput requirements [5]. This is due to the fact that the majority of researchers have been focusing on the other important problem: improving the accuracy of such systems. To obtain high accuracy even for small and medium lexicons is a challenging problem. When the number of entries in the lexicon grows, both the accuracy and the processing time are affected significantly. Some authors claim to have worked with large vocabularies; indeed, just few entries are really matched against the sequence of features extracted from the input image [38]. Most of the lexicon entries are purged by taking into account other sources of knowledge [35]. For example, in postal applications, the ZIP code

is frequently used to reduce the number of candidate city names to be recognized. Some other authors use geometrical measurements of the input image to estimate, for example, the word lengths, and some heuristic to eliminate words over such a length from the lexicon before proceeding to the decoding (search) [38, 179]. But the problem occurs when such sources of knowledge are not available or provide some unreliable information. In such cases, it is no longer possible to reduce the lexicon size and the system is required to deal with a large vocabulary.

4.1.8.2 Task Partitioning

Threads are an efficient way to parallelize the recognition process. Threads provide a simple mechanism to parallelize the recognition engine by partitioning the lexicon among concurrent threads. In our implementation we have considered static partitioning schemes which are easier to manipulate on small-scale multiprocessors. The parallelization involves static partitioning of data and computation among multiple threads using the *C-threads* facility. Figure 39 illustrates the parallelization of the recognition engine through the partitioning of the lexicon.

We have a global lexicon that contains 36,116 entries. The sequential version of the recognizer is required to match the sequence of features against all these entries before making a decision about the best word candidates. In order not to introduce so many modifications to our classifier, we split the lexicon into N_p partial lexicons where each one will have V/N_p entries. N_p denotes the number of processors and V the number of entries of the global lexicon. A thread is created for each partial lexicon and a classifier is used just to match the sequence of features against the entries of such a lexicon. The same is done for the rest of the partial lexicons. In this work we did not address the problem of quantitative load balance, that is, we have not ensured that all processors get approximately equal numbers of character HMMs to minimize load imbalance and idling overhead. The partition of the lexicon

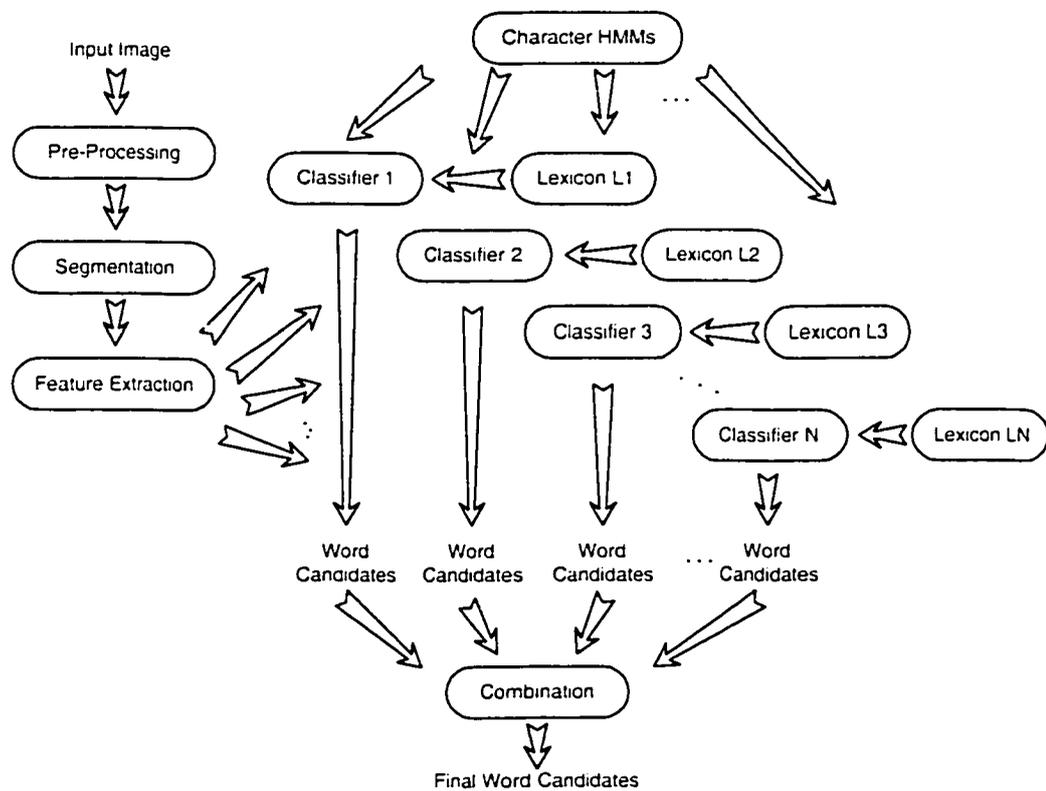


Figure 39 An overview of the distributed recognition scheme where several similar classifiers execute a relatively less complex recognition task for a partitioned lexicon

only uses the equal number of words in each partial lexicon as a criterion.

The output of each classifier is a list with the *TOP N* word candidates that give the highest likelihood score for that part of the global lexicon. The outputs of all classifiers must be combined to decide which are the best word candidates among all partial lexicons.

4.1.8.3 Combination of the Results

The goal of the combination module is just to combine the outputs of the N_P classifiers by taking into account the a posteriori probabilities. Such a combination problem is very simple to solve because the outputs of the classifiers are similar and provide comparable outputs. There is no need of normalization prior to the combination. Therefore, we just merge the N_P lists of the *TOP N* word candidates provided by each classifier and rank such a resulting list according to the likelihood scores in order to obtain the final *TOP N* word candidates. We observed exactly the same recognition rates in both the distributed and the sequential schemes. On the other hand, the processing time was reduced considerably.

4.1.8.4 Computational Complexity and Storage Requirements of the Distributed Recognition Scheme

Table XV shows the approximate computational complexity and storage requirements considering a flat lexicon and a tree-structured lexicon. The computational complexity is the same of the fast two-level decoding algorithm, except that the vocabulary size is $V' < V$, since it is partitioned among several processors.

However, it should be noticed that the computational complexities shown in Table XV only hold if the speedup is linear with the number of processors P .

Table XV

Computational complexity and storage requirements for the distributed recognition scheme using the fast two-level decoding algorithm and considering a flat structure and a tree-structured lexicon

	Computational Complexity	Storage Requirements
FS Flat	$\mathcal{O}(T^2N^2M + T^2LV')$	$2T^2HNP + 2MT^2P$
FS Tree	$\mathcal{O}(T^2N^2M + T^2L'V')$	$2T^2HNP + 2MT^2P + 3L'TV'P$

4.1.9 Summary of the Speed Improvements

The contents of this section can be summarized by comparing the computational complexities and storage requirements of the various approaches. Table XVI captures this information succinctly.

Table XVI

Computational complexity and storage requirements for all strategies presented in this section

Search Strategy	Computational Complexity	Storage Requirements
BLN	$\mathcal{O}(HTN^2LV + HTLV)$	$2HTLN$
LDLBA	$\mathcal{O}(HTN^2L'V)$	$2HTN + 2HT + 3L'TV$
CLBA & CDCLBA	$\mathcal{O}(HT'N^2L''V)$	$2HTN + 2HT + 3L'TV$
FS Flat	$\mathcal{O}(T^2N^2M + T^2LV)$	$2T^2HN + 2MT^2$
FS Tree	$\mathcal{O}(T^2N^2M + T^2L'V)$	$2T^2HN + 2MT^2 + 3L'TV$
DS FS Flat	$\mathcal{O}(T^2N^2M + T^2LV')$	$2T^2HNP + 2MT^2P$
DS FS Tree	$\mathcal{O}(T^2N^2M + T^2L'V')$	$2T^2HNP + 2MT^2P + 3L'TV'P$

H: Number of models per character class; T: Number of observations; N: Number of states;

L: Length of the words; V: Number of words in the vocabulary;

L': Reduced length of the words in a trie; V': Reduced number of words in the vocabulary;

M: Total number of models; T': Reduced number of observations;

L'': Reduce length of the words due to the length constraint; P: Number of processors.

However, just looking the expressions shown in Table XVI is hard to see which strategy is better. As we have mentioned before it is important to know the dimension of the variables involved in the computational complexity. To get a feeling on the computational complexity and storage requirements of each recognition strategy in Table XVII we use typical values: $H = 2$, $T = 30$, $L = 10$, $L' = 7$, $V = 80,000$, $V' = 8,000$, and $P = 10$. We assume that the speedup is linear with the number of processors.

Table XVII

Computational complexity and storage requirements for all strategies presented in this section

Search Strategy	MFLOPS	MBytes
BLN	4,800	0.768
LDLBA	3,300	22
CLBA & CDCLBA	1,200	22
FS Flat	700	1
FS Tree	500	24
DS FS Flat	70	10
DS FS Tree	50	34

Now, it is possible to have a better idea of the difference in the number of operations and memory usage of each recognition strategy. Compared with the recognition strategy of the baseline recognition system, the alternative search techniques that we have proposed in this chapter are very advantageous. On the other hand, they require a bit more memory, but they are compatible with most of the current personal computers.

We briefly review the results from this section:

- It is possible to improve the recognition speed by using a tree-structured lex-

icon with no effect on recognition accuracy. Organizing the lexicon as a tree-structure reduces the number of characters to be decoded in approximately 2 times relatively to the baseline recognition SRTP system that uses a flat lexicon:

- It is possible to reduce the complexity of the search by selecting only the best character models at each tree level with no significant effects on recognition accuracy;
- The lexicon-driven level building algorithms which integrates the lexical tree and the selection of best character models has a lower computational complexity relatively to the baseline recognition system for large vocabulary tasks;
- It is possible to further improve upon the recognition speed by limiting the number of observations aligned at each level as well as by limiting the number of levels of the level building algorithm;
- The contextual dependent constrained level building algorithm further improves upon the recognition speed by adjusting the constraints according to the character models;
- The fast two-level decoding algorithm deals with the limitations of the conventional decoding techniques and attempts to avoid repeated computation of character likelihood scores by decoupling the computation of the likelihood scores of individual character models and the computation of word likelihood scores. This approach is somewhat similar to the two-level DP matching used in speech recognition for connected word recognition [146];
- The fast search strategy handles the paradox of decoding the single characters separately from the context (words) but relying on the entire words to compute the overall probabilities, ensuring an optimal solution comparable to the one provided by the Viterbi search but with a reduced computational complexity that enables it to tackle the problem of large vocabulary handwriting recognition efficiently;

- We have exploited the potential for speedup through parallelism using a very simple approach that instead of parallelizing parts of the algorithm, it cuts up the recognition task and each processor is in charge of decoding part of the lexicon. The expected reduction in the computational complexity is proportional to the number of processors.

In Chapter 5 we compare the implementation of all these search strategies. So, there it will be possible to both the recognition speed, memory usage and accuracy of the search strategies. But before going to the experimental results, the following section presents our attempt to improve the recognition accuracy as well.

4.2 Verification of Handwritten Words

Another goal of this thesis is to demonstrate that it is possible to achieve high recognition rates on large vocabulary tasks without compromising the speedup achieved by the strategies presented in Section 4.1 (see Section 5.4.7). The recognition of handwritten words in a large vocabulary can be considered as a very complex classification problem, where a decision must be taken among several classes. As we have seen in the previous chapters, the baseline recognition system provides a list with the N -best word hypotheses as result. However, usually a single response is required and the most intuitive solution is to consider the *TOP* 1 word hypothesis as the identity of the unknown test pattern.

The baseline recognition system performs segmentation and recognition of handwritten words and it provides at the output a list with the N -best word hypotheses that best match against the sequence of observations generated from the input image. Based on the previous evaluation of the performance of such a system presented in Chapter 3, we know that as the number of words in the vocabulary increases, as the recognition accuracy decreases. However, if we take into account the list of the N best answers, it decreases slightly than the top choice. Considering only the *TOP*

1 word hypothesis, the difference in the recognition rate between a 10-word and a 30k-word vocabulary is 25.23%. On the other hand, if we consider the *TOP* 10 word hypotheses, the difference in the recognition rate is 11.85%. In practice these ranked lists are regarded as intermediate results and at the end of the recognition process, a decision on whether accept the *TOP* 1 word hypothesis as the correct or reject all hypotheses has to be taken. A common approach in handwriting recognition to deal with these ranked lists is to incorporate at the end a rejection mechanism to accept or reject the word hypothesis based on the confidence scores values assigned to each one [35]. If the confidence score of the *TOP* 1 choice is significantly higher than the other $N - 1$ choices, the *TOP* 1 label is assigned to the input word, otherwise, the input word is rejected. Only few researchers have used verification modules in handwriting recognition to improve the recognition accuracy of their systems based on these multiple hypotheses lists [14, 47, 110, 131].

In this section we propose a verification scheme which aims to improve the recognition accuracy of the baseline recognition system. We use the list of the N -best word hypotheses generated by the baseline recognition system which in addition to each word hypothesis it also provides the segmentation of such word hypotheses into characters. Based on the segmentation, the verification module attempts to recognize each segment as a whole character using different features, different character models and different recognition strategy.

Based on the error analysis presented in Chapter 2, we can summarize the main problems that have to be overcome to improve the accuracy of the recognition by combining the baseline SRTP recognition system with a different recognition strategy:

- Better discrimination of similar shapes;
- The results should be systematically integrable with the results of the HMMs;

- The other recognition strategy should focus on the weaknesses of the baseline recognition system to complement its results:
- Model each character as a whole to overcome the limitations of the first order Markov assumption:

The outline of this section is as follows. In Section 4.2.1 the characteristics of the output of the word recognizer are presented and analyzed. The main idea behind this analysis is to identify the information that could be helpful for the verification and to define a verification strategy. In Section 4.2.2 some practical constraints that have to be taken into account in designing the verifier and the problem of verification are discussed. Following such a discussion, in Section 4.2.3 the architecture of the verification module is described and its many components, such as the isolated character recognizer based on neural networks, the feature extraction, etc. are described. In Section 4.2.6 the characteristics of the output of the verification module are presented.

4.2.1 Characteristics of the Word Recognizer

The baseline recognition system described in Chapter 3 performs a huge task of classifying an input pattern of unknown class into one of the possible V classes, where V corresponds to the size of the vocabulary. The output of such a system is an N -best word hypothesis list ordered according to the confidence scores, i.e. the likelihood assigned to each word hypothesis. This list contains not only the ASCII label and the likelihood for each word hypothesis but also the segmentation of each word hypothesis into characters. The segmentation of each word hypothesis into characters is obtained by the backtracking of the best state sequence of the Viterbi algorithm or by one of the alternative decoding algorithms presented in Section 4.1.

So, the output of the baseline recognition system, denoted as Ψ_{HMM} can be represented by a triple:

$$\Psi_{\text{HMM}} = \{\hat{L}_n, \hat{S}_n, \hat{P}_n\} \quad (4.33)$$

where \hat{L}_n is the label for the n -th word hypothesis, and it is given as a sequence of characters $\hat{L}_n = c_1^n c_2^n \dots c_L^n$ in which L is the number of characters in the word and c_l^n denotes the l -th character of the n -th word hypothesis; $\hat{S}_n = x_1^n x_2^n \dots x_L^n$ is a set of segments that corresponds to the segmentation of the word into characters in which x_l is the segment that corresponds to the l -th word character of the n -th word hypothesis; \hat{P}_n is the confidence score (or likelihood) assigned to the n -th word hypothesis. Figure 40 shows some examples of 10-best word hypothesis lists generated by the baseline recognition system.

Label	Segmentation	Likelihood	Label	Segmentation	Likelihood
BOURS	11221	-14.378855	NERON	21112	-8.522033
DOURS	11221	-14.384986	VERON	21112	-8.530042
ADGER	11221	-14.453356	HERON	21112	-9.009810
JOUTEL	112111	-14.791832	MERON	21112	-9.354612
BAGAS	11221	-14.876367	GERTON	211012	-9.356061
SPURR	11221	-14.921134	SERON	21112	-9.358621
BAURA	11221	-14.965520	CERON	21112	-9.432840
BOUER	11221	-15.077375	VERTON	211012	-9.570351
DOUDS	11221	-15.164326	FERON	21112	-9.573650
BAGES	11221	-15.207915	USSON	21112	-9.583147

Label	Segmentation	Likelihood
RIVIERE_SUR_TARN	11211111211112	-25.591029
RIVIERE_SAINTE_JEAN	1121111101201112	-32.222140
RIVIERE_SAINTE_PAUL	1121111101211111	-34.300776
ASNIERES_SUR_SAONE	1121111012111120	-34.305419
LAYRAC_SUR_TARN	13210112111112	-34.554012
ATHIES_SOUS_LAON	11311110211112	-34.686680
RIVIERE_DU_MAT	112111113212	-35.098250
RIVIERE_DE_LA_SAVANE	112111111002110120	-35.333127
PERIERS_SUR_LE_DAN	121111112101112	-35.601541
SAINTE_BREVIN_L_OCEAN	10121111112110012	-35.743930

Figure 40 Some examples of 10-best word hypothesis lists generated by the baseline recognition system

4.2.2 Formalization of the Verification Problem

The main hypothesis in designing a verification scheme to improve the accuracy of the baseline recognition system is that the final segmentation of the words into characters carried out by the baseline recognition system is somewhat reliable, even if the input pattern is not recognized correctly (as the *TOP 1* hypothesis). Some previous studies have shown that the segmentation is reliable in most of the cases [32, 38, 54]. For instance, Duplantier [32] has visually inspected 10,006 word images from the training dataset of the SRTP database and compared with the alignment (segments–characters) provided by the baseline recognition system. In less than 20% of the cases (2,001 out of 10,006), segmentation problems mainly due to undersegmentation and labeling were identified.

Furthermore, if we look at the *N*-best list, the difference in the accuracy when considering only the best hypothesis (*TOP 1*) and the ten best hypotheses (*TOP 10*) is more than 15% (for an 80k-word lexicon). This is due to the presence of very similar words in the lexicon that may differ only by one or two characters. For this reason it seems that the most straightforward manner is to use the segmentation of the word hypotheses into characters and the word hypothesis labels to build an isolated character recognizer to carry out verification at the character level in an attempt to better discriminate characters, since such an aspect was somewhat overlooked by the word recognition based on HMMs.

Another possible strategy would be the use of an holistic verifier to verify the word hypotheses as a whole instead of as a concatenation of characters. The additional advantage of using such a strategy is that it can model coarticulation effects, i.e., the changes in the appearance of a character as a function of the shapes of the neighboring characters. However, a practical limitation prevent us to do use such a strategy for verification: the lack of data for training a holistic classifier. Never-

theless, the lack of data is also troublesome to build the verification strategy based on the recognition of isolated characters as well as many other issues such as the dimensionality of the feature vector, the high number of classes (26 to 62 classes), the presence of characters poorly segmented (undersegmented and oversegmented characters), recognition time, memory usage, etc.

Another strong justification to support this strategy is the limitation of the HMMs to model the handwriting signal: the assumption that neighboring observations are conditionally independent, which prevents an HMM from taking full advantage of the correlation that exists among the observations of a character; and the awkwardness with which segmental features such as duration can be incorporated into HMM systems. These two limitations can be overcome by modeling and recognizing a character as a single unit rather than a sequence of conditionally independent observations.

So, given the triple $\{\hat{L}_n, \hat{S}_n, \hat{P}_n\}$ provided by the HMM classifier, character alternatives are located within the word by using the segmentation \hat{S}_n . Another feature extraction module is used to extract new features from such segments, and a different feature vector is formed. The task of the isolated character classifier is to assign Bayesian *a posteriori* probabilities to such a new feature vector that represents a character given that its class is already known. Further, the probabilities of the individual characters can be combined to generate a new score to the word hypotheses. These new word scores can be combined with the output of the HMM classifier by a suitable rule to build up a hybrid recognition system. Figure 41 shows an overview of main components of the verification module. The details of the verification module and its main components are presented in the rest of this Section as well as its integration with the HMM-based word recognition building up a recognition-verification system.

Figure 42 shows the variation in the recognition rate when taking a greater number

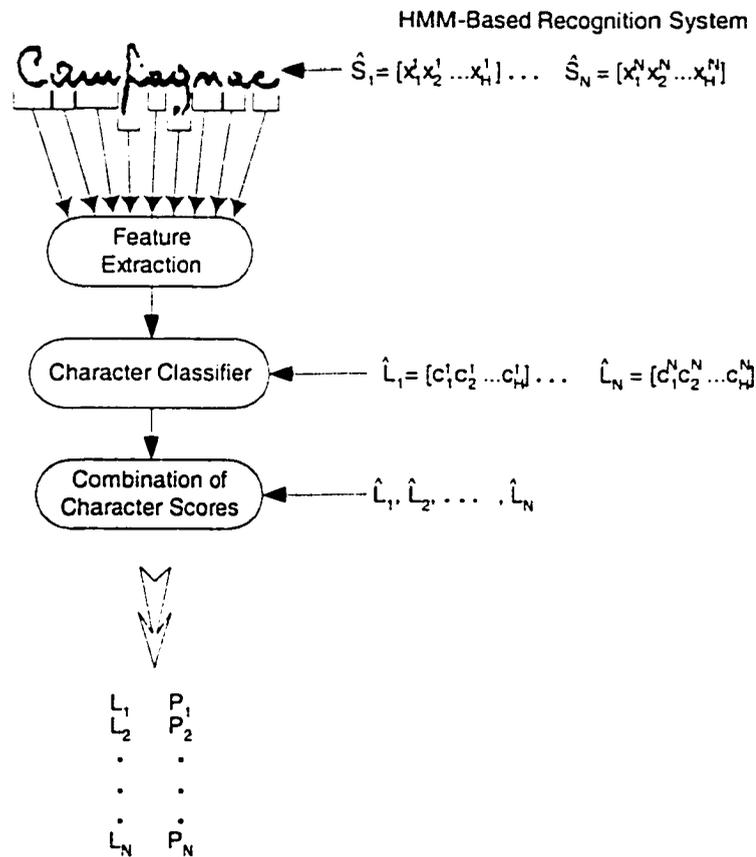


Figure 41 An overview of the main components and inputs of the verification module

of N -best word hypothesis. It is clear that the more word hypotheses we take, the more the recognition rate increases. For the verification we could consider for example the *TOP* 100 best hypotheses, however, in the scope of this thesis it would be impractical to consider such a number of word hypotheses and we consider only the *TOP* 10 word hypotheses.

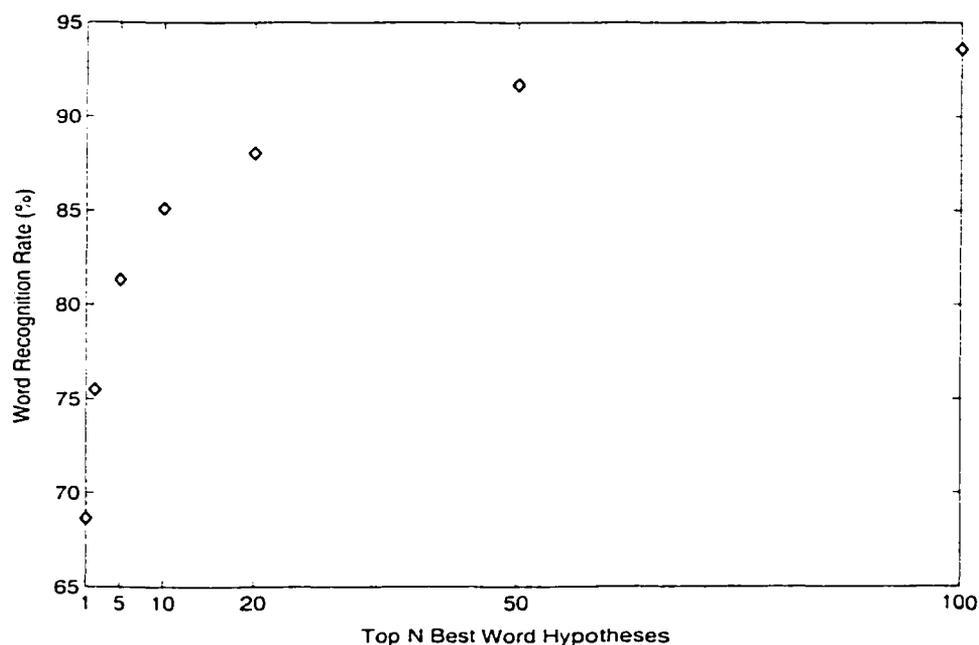


Figure 42 The relation between the recognition rate and the number of N -best word hypotheses

4.2.3 Architecture of the Verification Module

The main role of the verifier is to post-process the output generated by the handwritten word recognition system to improve its reliability and performance. It is also desirable that the verifier achieves both high accuracy and high speed. However, an important remark is that our main goal is not to develop a optimized classifier to achieve high recognition rates, but to focus on the conceptual aspect of the recognition-verification strategy to validate our research hypothesis. The main elements of the verification process are:

- Feature Extraction;
- Neural Network Classifier;
- Combination of Character Scores.

The verification scheme that we have proposed is based on the recognition of isolated handwritten characters which is introduced to improve the disambiguation among the resulting word hypotheses. Handwritten character recognition has been the subject of much attention in the field of handwriting recognition. Several proposals to solve this problem have been presented throughout the last decade [13, 65, 168]. However, our problem is slightly different, since we do not want to really perform character classification, that is, assign a class to a given feature vector. In fact, the character class is already known, and we only want to reestimate the probability of such a class to represent the feature vector. The information provided by the word recognizer is $(c_1, x_1), (c_2, x_2) \dots (c_H, x_H)$. So, the task of the character classifier is to assign an *a posteriori* probability to each segment (x_h) given that the character class c_h was assigned to such a segment. The output of the character classifier is only a probability score (*a posteriori* probability) for the given character class, $P(c_h|x_h)$.

Recalling that our main goal is to recognize handwritten words from the SRTP database. However, while for some classes of characters the number of samples is quite elevate (thousands of samples), for others, it is quite scarce (less than 10). Besides to this problem, we have a high number of classes (from 26 to 52 classes).

Therefore, the lack of data causes several restrictions to the development of our work, such as:

- Lack of data. The limited size of the training dataset is one of the most heavy constraints to the development of our work;
- Unbalanced sample class distribution.

These practical effects of the practical constraints mentioned above is that we have to take care about the dimensionality of the feature vector due to the *curse of dimensionality* [69] and we also have to work around to alleviate the problem of character class sample distribution during training. For the latter problem, we could not use samples from another database to increase the number of the samples of the SRTP character database for those classes that have few samples. Usually, the under represented classes coincide with those that show up less frequently within the words. Another possible solution could be to generate synthetic data for the under represented classes by applying transformations to the actual samples. This solution was adopted by Gader et al. [47] that have constructed balanced training and testing datasets from images of addresses from the USPS mail using 250 characters from each class for each set. For the classes with less than 500 samples, they randomly resized the existing characters to create new samples.

4.2.4 Feature Extraction

The selection of a feature extraction method is probably the single most important factor in achieving high recognition performance [165]. In feature extraction the aim is to represent the image in terms of some quantifiable measurement that may be easily used in the classification stage. Features extracted must minimize the within-class pattern variability and maximize the between-class pattern variability to provide for sufficient discrimination among the different characters. Often, a single feature extraction method alone is not sufficient to obtain good discrimination power. An obvious solution is to combine features from different feature extraction methods.

Several different types of features and feature extraction methods have been proposed in the last decade to represent isolated character handwritten recognition such as size normalization [30, 46, 95, 129, 134, 150, 168], zoning [9, 46, 81, 82, 134, 155],

concavities [159, 163], cavities [46], contour-directional histogram [46, 81], contour-direction [18, 25, 121, 163], chain-code histogram [155], gradient [30, 81], profiles [103], projections [25], distance map [59, 129], directional distance [129], tangent [150], end points [159, 163], cross points [159, 163], pixel density [9, 18, 134], bars [24, 47], transition [24, 47], h/w ratio [18], etc. Table V in Chapter 1 presents a summary of some features together with classifiers used in character recognition.

Besides all these proposed features, there is not a clear methodology to choose the more suitable feature set for a specific problem. The criteria that have been used are based on the empirical evaluation of the recognition rate by using different combinations of features. Recently, genetic algorithms have also helped in the selection of features in specific applications [131].

As we have mentioned earlier, our main interest is not to develop a “fully-optimized” classifier, but a good classifier. For this reason, an empirical evaluation of several types of features seems to be sufficient. To such an aim, we have evaluated many different features combined into different feature sets as described later in this Chapter. Such an empirical evaluation lead us to build a 108-dimensional feature vector by combining 3 different types of features:

- Projection histogram from whole characters;
- Profiles from whole characters;
- Directional histogram from 6 zones.

Next we give a brief description of the 3 feature types that we have chosen. We assume binary images in which the background is represented by white pixels and the character is represented by black pixels.

4.2.4.1 Profiles

The profile counts the number of pixels (distance) between the bounding box of the character image and the edge of the character. The profile of an character can be taken at any position, but usually profiles are taken at 4 positions: top, bottom, left and right hand sides as illustrated in Figure 43.

The profiles describe well the external shapes of characters and allow to distinguish between a great number of letters, such as “p” and “q”. Since the profiles depend on the image dimension, the features are made scale independent by using a fixed number of bins on each axis that is obtained by merging neighboring pixels and dividing by the total number of black pixels in the character image. We have normalized the profiles to ten bins at each axis to have an equal number of elements for all characters⁴.

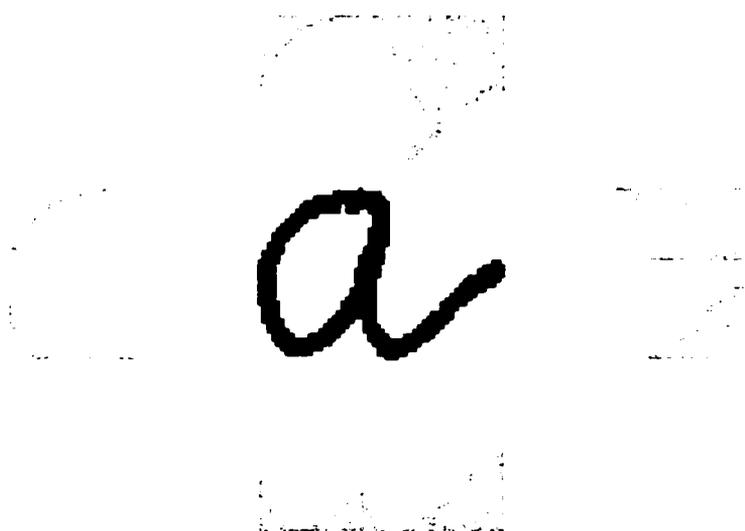


Figure 43 An example of the four projection profiles for the letter “a”: top, bottom, left and right hand side profiles

⁴The number of bins was determined empirically by exploratory experiments on the validation dataset where the character recognition rate and the dimensionality were used as evaluation criteria.

4.2.4.2 Projection Histograms

Projection histograms count the number of pixels in each column and row of a character image [25, 65, 165]. The projection can also be taken at any position, but usually projection are taken at the vertical and horizontal axis. For a horizontal projection, $Ph(x_i)$ is the number of pixels with $x = x_i$, while for a vertical histogram, $Ph(y_i)$ is the number of pixels with $y = y_i$. The features are made scale independent by using a fixed number of bins on each axis that is obtained by merging neighboring pixels and dividing by the total number of black pixels in the character image. Projection histograms can separate characters such as “m” and “n” (3 and 2 peaks in vertical projection respectively) or “E” and “F” (3 and 2 peaks in horizontal projection respectively). The projection histograms are normalized to ten bins⁵ at each axis to have an equal number of elements for all characters. Figure 44 shows the vertical and horizontal projection histograms for the letter “a”.

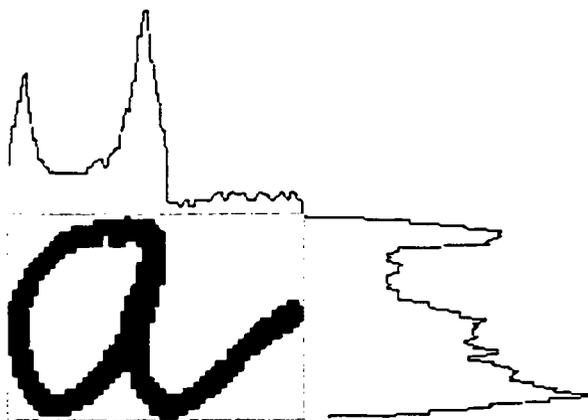


Figure 44 An example of the two projection histograms for the letter “a”: vertical and horizontal projection histograms

⁵The number of bins was determined empirically by exploratory experiments on the validation dataset where the character recognition rate and the dimensionality were used as evaluation criteria.

4.2.4.3 Contour-Directional Histogram

The contour of the character image is given by the outer and inner boundary pixels that can be easily found by examining each pixel within a 3×3 window. Considering a binary character image, if the center pixel is black and at least one of its neighborhood pixels is white, then the center pixel is a contour pixel and it is set as such (set to black). All other pixels are set to white. Figure 45a shows the contour for the letter "a". Next, the resulting contour is divided into $n \times m$ zones by superimposing an $n \times m$ grid as show in Figure 45b [165]. For each of these zones the contour is followed and a directional histogram is obtained by analyzing the adjacent pixels in the 3×3 neighborhood of each contour's pixel (center pixel). Figure 46 shows the contour-directional histogram. The goal of the zoning is to obtain local characteristics, instead of global characteristics. The zones provide information on the character contours and the direction of the strokes that form them.

We have carried out an empirical evaluation of the several grid dimensions ($2 \times 2 \dots 7 \times 7$), where the criterion to determine the best zoning was the recognition rate achieved. In such experiments, best results were achieved by 3×3 and 3×2 zoning.

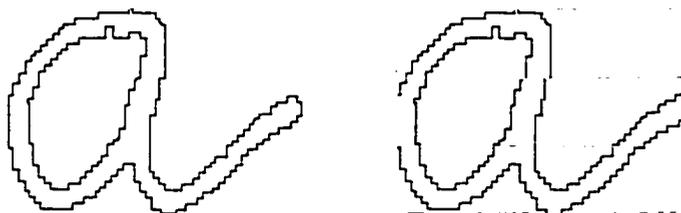


Figure 45 (a) An example of the contour extracted from the letter "a". (b) the contour split in 6 parts corresponding to the 6 zones

4.2.4.4 Selection of Characteristics

As we have mentioned before, besides these 3 different features types, we also have developed some others as shown in Table XVIII, which are defined in [65, 165]. In-

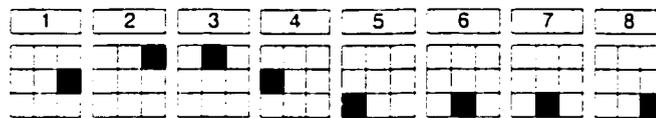


Figure 46 The eight directions used in the directional-contour histogram

stead of using more complicated and time-costly methods to select features, such as float search or genetic algorithms [69,92], we have carried out some exploratory experiments to determine which combination of features achieves the best recognition rates on the NIST database. For such an aim, we have tested seven combinations as shown in Table XVIII as well as the dimension of the resulting feature vector. The recognition rates achieved by each of these combinations are presented in Table XIX. However, the recognition rate was not the only criterion, we have also taken into account the dimension of the resulting feature vector where smaller is better.

Table XVIII

Several combination of features to determine the best feature vector for isolated handwritten character recognition

Feature Type	Dimension	Experiment						
		1	2	3	4	5	6	7
Surface	1	Yes	Yes	—	—	—	—	—
Extrema	16	Yes	Yes	—	—	—	—	—
Orientation	9	Yes	Yes	Yes	Yes	—	—	—
Excentricity	9	Yes	Yes	Yes	Yes	—	—	—
Projections	20	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Profiles	40	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Contours	20	Yes	Yes	Yes	Yes	—	—	—
Direction (9 zones)	72	Yes	Yes	Yes	Yes	Yes	—	—
Ratio H/W	1	—	Yes	Yes	—	—	—	Yes
Surfaces	6 or 9	—	Yes	Yes	—	—	—	Yes
Direction (6 zones)	48	—	—	—	—	—	Yes	Yes
Dimension	—	187	197	180	170	132	108	118

Table XIX

Character recognition rates achieved by different combinations of features on the NIST database, considering 26 metaclasses

Dataset	Character Recognition Rate (%)						
	Experiment						
	1	2	3	4	5	6	7
Training	93.77	94.03	94.63	94.50	94.36	94.71	94.40
Validation	88.80	88.94	89.36	88.91	89.43	89.98	89.56

The best results were obtained in experiment 6 by using the combination of projections, profiles and contour-directional histograms (with 6 zones). So, the resulting feature vector that is used in the rest of this thesis for isolated character recognition has 108 elements, as shown in Figure 47.

Horizontal and Vertical Projections (10 + 10)	Top, Bottom, Right and Left Profiles (10 + 10 + 10 + 10)	Directional Contour Histogram (6 zones) (8 + 8 + 8 + 8 + 8 + 8)
--	---	--

Figure 47 Resulting 108-dimensional feature vector for isolated handwritten character recognition

4.2.5 NN Classifier

We have designed a simple unconstrained character recognizer based on a multilayer perceptron (MLP). Among various neural network models, MLP is the most widely used, especially in the problems of pattern classification [177].

The choice of such a classifier to perform the character recognition task was determined by several constraints such as: recognition speed, capacity of dealing with unbalanced distribution of samples per class, and mainly because, if properly configured, an MLP classifier estimates Bayesian *a posteriori* probabilities [144]. The

last characteristic is very important, because later we want to combine the output of the character recognizer with that of the HMM classifier. If both classifiers estimate Bayesian *a posteriori* probabilities at the output, they can be combined in a probabilistic framework [174]. Furthermore, Wang et al. [168] have presented a comparison of statistical (Support Vector Machines, HMM, and k -NN) and neural classifiers (MLP) both in terms of recognition accuracy and recognition speed. They have found that the statistical classifiers are 5–15 times slower than MLP when only the *TOP* 1 hypothesis is required, and 4.5–81 times slower than MLP when the *TOP* 5 hypotheses are required.

Many neural network classifiers provide outputs which estimate Bayesian *a posteriori* probabilities. When the estimation is accurate, network outputs can be treated as probabilities and sum to one [8, 144]. However, estimation accuracy depends on the network complexity, the amount of training data, and the degree of which training data reflects true likelihood distributions and *a priori* class probabilities. Interpretation of network outputs as Bayesian probabilities allows outputs from multiple networks to be combined for higher level decision making, simplifies creation of rejection thresholds, makes it possible to compensate for differences between pattern class probabilities in training and test data, allows outputs to be used to minimize risk functions, and suggests alternative measures of network performance [8, 144].

4.2.5.1 Network Architecture

The perceptron is the simplest form of a neural network used for the classification of a special type of patterns said to be linearly separable [62]. The multilayer perceptron combines several perceptrons to form a weighted sum of the components of the input vector and adds a bias value. The model of each neuron includes a smooth nonlinearity at the output end. The most common used form of nonlinearity

is a sigmoidal nonlinearity defined by the logistic function:

$$y_i = \frac{1}{1 + e^{-\beta_i}} \quad (4.34)$$

where β_i is the net internal activity level of neuron i and y_i is the output of the neuron.

The network consists of an input layer, an output layer, and one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons overcome single perceptron's limitations of linear decision and enable the network to learn complex tasks. Neurons in adjacent layers are connected through links whose associated weights determine the contribution of neurons on one end to the overall activation of neurons on the other end.

To build an MLP classifier basically we have to determine the number of layers and the number of neurons in each layer. In theory, for an m -class classification problem in which the union of the m distinct classes forms the entire input space, we need a total of m outputs to represent all possible classification decisions. Many different strategies could be used to recognize isolated handwritten characters. The most straightforward ones are:

- A 52-class classification problem: uppercase and lowercase representations of a single character are considered different classes (e.g. "A" and "a" are two distinct classes);
- A 26-class classification problem: uppercase and lowercase representations of a single character are merged into a unique class called *metaclass* (e.g. "A" and "a" form the metaclass "Aa").

Recalling that one of the characteristics of the output of the baseline SRTP recog-

nition system is the weakness to distinguish between uppercase and lowercase characters, the natural choice is to consider a 26-class classification problem and merge them into a unique class, that is, 26 metaclasses. Figure 48 confirms that the baseline recognition system is not reliable to distinguish between uppercase and lowercase characters since it recognizes approximately 45% of the cases correctly. Besides that, some exploratory experiments have been carried out with the NIST database to investigate the influence of the number of classes on the recognition rate [103] which has also indicated that is preferable to merge uppercase and lowercase characters into a single character class. Table XX shows the recognition rates obtained by using different strategies: two independent neural networks for uppercase and lowercase character; a unique classifier where uppercase and lowercase characters are merged into a single class (*mixed*); and the combination of the results of the two neural network classifiers by a *max* rule. These results also confirm that it is preferable to deal with 26-class classification problem.

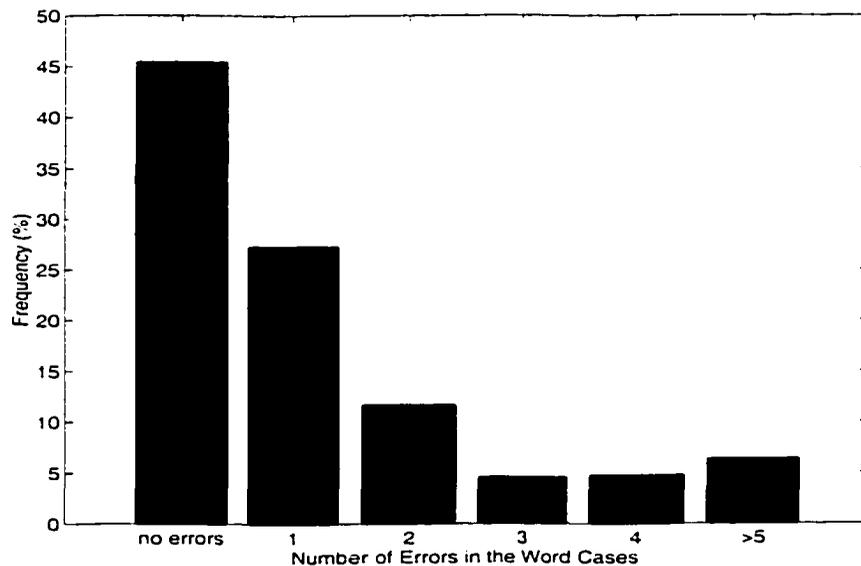


Figure 48 Number of errors in the correct case recognition at the output of the word recognizer for the validation dataset

Table XX

Results for isolated character recognition on NIST database considering uppercase and lowercase characters separately, considering uppercase and lowercase as a single class (Mixed) and considering the combination of two specialized classifiers

Dataset	Character Recognition Rate (%)			
	Uppercase 26 classes	Lowercase 26 classes	Mixed 26 metaclasses	Combination Upper+Lower 52 classes
Training	97.26	95.01	94.71	91.27
Validation	93.09	88.88	89.98	86.33
Test	92.37	84.69	88.10	85.51

Figure 49 shows the architectural graph of the multilayer perceptron with one hidden layer. The network is fully connected, which means that each neuron in any layer of the network is connected to all the nodes/neurons in the previous layer. The network takes a 108-dimensional feature vector as input, and it has 100 units in the hidden layer and 26 outputs, one for each character class. The number of hidden neurons was determined by a rule of thumb and some exploratory experiments where the error on the training and validation sets were used as criteria.

Networks output should sum to one for each input value if outputs accurately estimate Bayesian probabilities. For the MLP network, the value of each output necessary remains between zero and one because of the sigmoidal functions used.

4.2.5.2 Frequency Balancing

A simple method for alleviating difficulty with unequal prior class probabilities is to adjust the number of patterns in each class, either by subsampling (removing patterns from higher frequency classes), or by duplication of patterns in lower frequency classes.

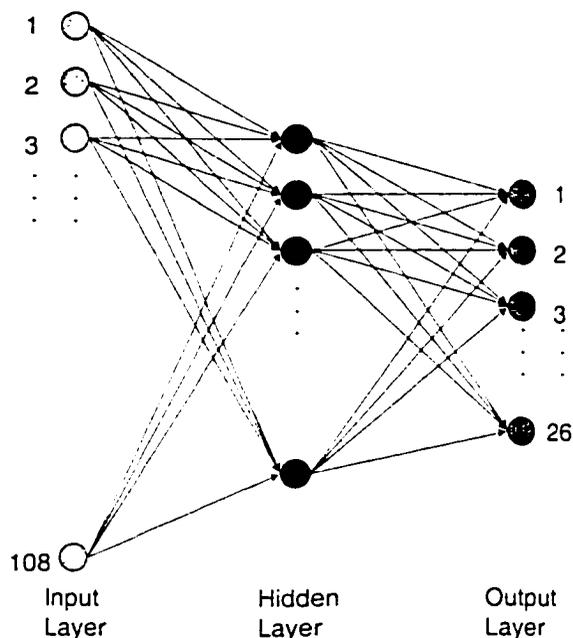


Figure 49 Architectural graph of the multilayer perceptron with 108 inputs, 90 neurons in the hidden layer and 26 outputs

The training data derived from the city names contained in the SRTP database exhibit very non-uniform priors for the various character classes, and neural networks readily model these priors. However, reducing the effects of these priors on the network, in a controlled way, forces the network to allocate more of its resources to low-frequency, low-probability classes [170]. This is of significant benefit to the overall word recognition process [170].

To this end, the frequency of the character class during training is explicitly balanced by probabilistically skipping and repeating patterns, based on a pre-computed repetition factor, as suggested by Yaeger et al. [170]. Each presentation of a repeated pattern is “warped” randomly as discussed later.

First, for a given character class c , a normalized frequency of samples in that class is computed as:

$$F_c = \frac{S_c}{\hat{S}} \quad (4.35)$$

where S_c is the number of samples in character class c , and \hat{S} is the average number of samples over all character classes, computed as:

$$\hat{S} = \frac{1}{C} \sum_{c=1}^C S_c \quad (4.36)$$

with C being the number of character classes. The repetition factor is defined to be:

$$R_c = \left(\frac{1}{F_c}\right) \quad (4.37)$$

In the case of the SRTP database, for the training dataset we have $\hat{S} = 1.630$ and $C = 26$.

4.2.5.3 Stroke Warping

During training random variations in character stroke consisting of small changes in size, rotation, and horizontal and vertical scalings. This produces alternate character forms that are consistent with stylistic variations within and between writers. The amounts of each distortion is chosen randomly. A small set of such variations is shown in Figures 50 and 51.

This stroke warping scheme is somewhat related to the one proposed by Yaeger et al. [170], however, we did not attempt to optimize the amount of distortion needed to yield optimum generalization based on cross-validation experiments. The algorithm

for stroke warping is presented in Appendix 2.

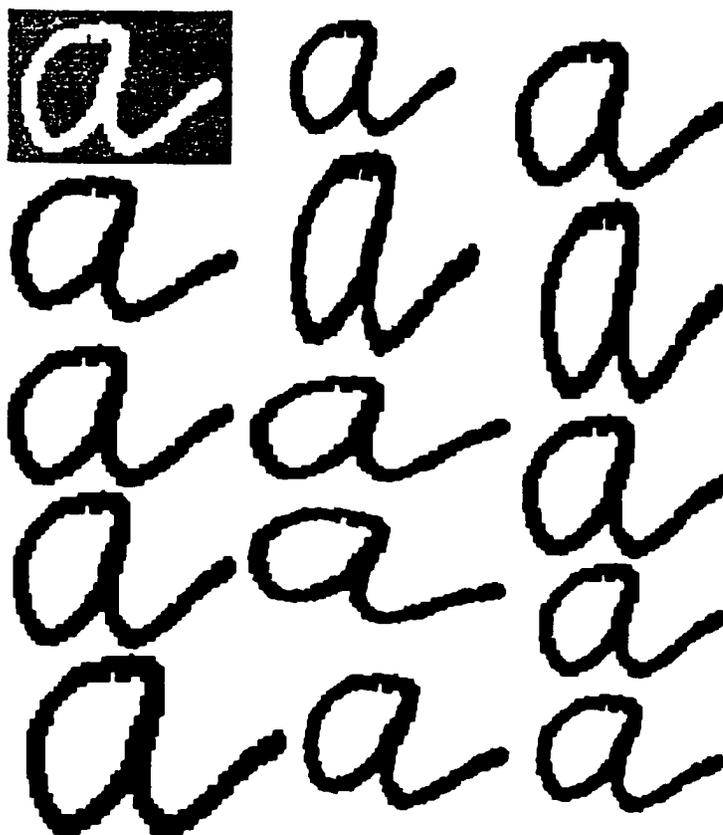


Figure 50 A few random stroke warpings of the same original “a” character (inside the gray box)

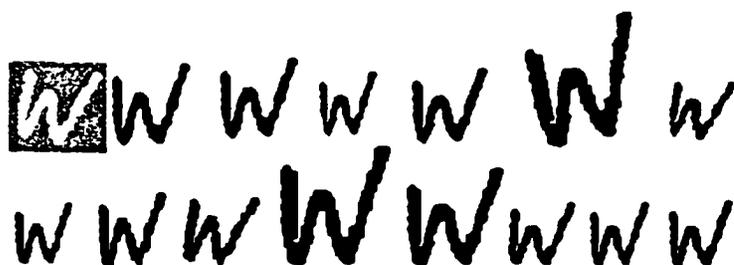


Figure 51 A few random stroke warpings of the same original “W” character (inside the gray box)

4.2.5.4 Training the Neural Network Classifier

Conventionally, neural networks have to be trained to accomplish an specific recognition task. The training consists in the adjustment of the weights of the links to get the desired behaviour. This modification is very often based on the Hebbian rule which states that a link between two units is strengthened if both units are active at the same time.

The weights of the network are trained to associate a "desired" output vector with an input vector. This is achieved via the *Error Back-Propagation* algorithm [8, 43, 62] that uses a steepest descent procedure to iteratively minimize the error. The squared-error cost function has been used more frequently than any alternative. Its use yields good performance with large databases on real-world problems; and it can be used for classification problems [144]. The relationship between minimizing a squared-error cost function and estimating Bayesian probabilities was established for multiclass case [144]. For a 1 of C problem, d_i equals one if the input X belongs to class c_i and zero otherwise. Therefore, the conditional expectations are the following:

$$E\{d_i|X\} = \sum_{j=1}^C d_i P(c_j|X) = P(c_j|X) \quad (4.38)$$

which are the Bayesian probabilities. Therefore, for a 1 of C problem, when network parameters are chosen to minimize a squared-error cost function, the outputs estimate the Bayesian probabilities so as to minimize the mean-square estimation error.

The training remains for a certain number of training cycles until it reaches a predefined error on the training set. The backpropagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is

sufficiently small. However, to ensure a good generalization and to avoid overfitting, after each iteration, the network is tested for its generalization performance and the learning process is stopped when the validation set error is minimum. At this point, the net generalizes best.

The 26-metaclass MLP classifier was trained with 80,600 characters from NIST database (3,100 samples per metaclass, where 1,660 are uppercase and 1,440 are lowercase samples) using the backpropagation algorithm. A validation set with 23,670 characters was also used during the training. Figure 52 shows the training and validation curves for the NIST datasets. The NIST dataset the training error decreases to values as low as 0.10 while the validation error decreases to values as low as 0.20. For instance, for the NIST dataset, the training is stopped at a minimum of the error on the validation set ($MSE \approx 0.16$) that occurs close to 200 epochs.

The same classifier was trained with 84,811 characters from SRTP database (3,260 samples per metaclass, where 1,630 are uppercase and 1,630 are lowercase samples) using the backpropagation algorithm. A validation set with 27,282 characters was also used during the training. Figure 53 shows the training and validation curves for the SRTP datasets. Compared to the NIST dataset, the training of MLP with the SRTP dataset converges to relatively higher errors as shown in Figure 53. The training error decreases to values as low as 0.30 while the validation error decreases to values as low as 0.50. The training is stopped at a minimum error on the validation set ($MSE \approx 0.47$) that occurs close to 120 epochs.

It is worth to mention that during training many techniques were used, such as some noise is added to all links in the network, the patterns are presented in different order in the different cycles, etc. These techniques often improve the performance of the network, since they help to avoid local minima.

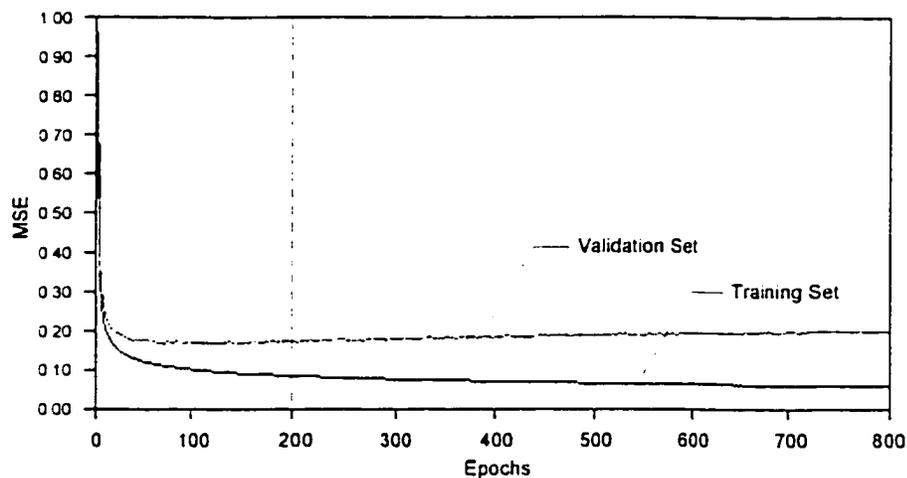


Figure 52 Training and validation curves for the NIST database. The dashed line indicates where the training procedure was stopped

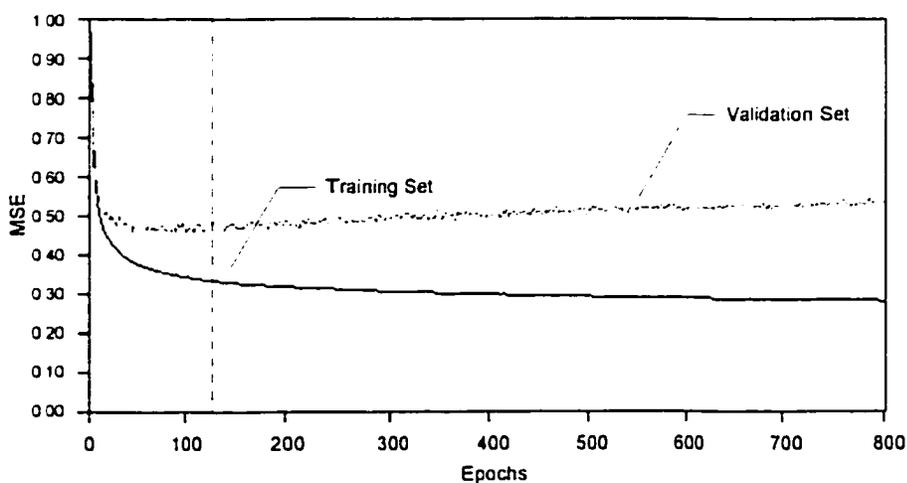


Figure 53 Training and validation curves for the SRTP database. The dashed line indicates where the training procedure was stopped

4.2.5.5 Compensating for Varying *a priori* Class Probabilities

Networks with outputs that estimate Bayesian probabilities do not explicitly estimate the three terms on the right of Equation 4.39 separately.

$$P(c_i|X) = \frac{P(X|c_i)P(c_i)}{P(X)} \quad (4.39)$$

However, the output of the network, denoted as $y_i(X)$ is implicitly the corresponding *a priori* class probability $P(c_i)$ times the class probability $P(X|c_i)$ divided by the unconditional input probability $P(X)$.

During the network training, *a priori* class probabilities $P(c_i)$ were modified due to the frequency balancing. As a result, the output of the network has to be adjusted to compensate for training data with class probabilities that are not representative of the real class distributions.

Correct class probabilities can be used by first dividing network outputs by training-data class probabilities and then multiplying by the correct class probabilities, as:

$$\hat{y}_i(X) = \hat{P}(c_i|X) = \frac{y_i(X)}{P_{train}(c_i)} P_{real}(c_i) \quad (4.40)$$

where $\hat{y}_i(X)$ denotes the corrected network output, $P_{train}(c_i)$ denotes the *a priori* class probability of the frequency-balanced training set, and $P_{real}(c_i)$ denotes the real *a priori* class probability of the training set.

4.2.6 Combination of the Character Scores

Having the scores of each character segmented from a word hypothesis, we can combine such scores to obtain the score for the word hypothesis. So, the output of the character recognizer Ψ_{NN} can be represented by:

$$\Psi_{\text{NN}} = P(L_n|S_n) = \prod_{h=1}^H \frac{P(c_h|x_h)}{P(c_h)} P(L_n) \quad (4.41)$$

where $P(L_n|S_n)$ is the *a posteriori* probability of the word hypothesis L_n , $P(c_h|x_h)$ is the *a posteriori* probability estimated by the NN to each segment x_h , $P(c_h)$ is the *a priori* probability of the character class and it can be estimated from the words in the recognition vocabulary, and $P(L_n)$ is the probability of the word hypothesis n . $P(L_n)$ can be neglected since the words in the lexicon have the same *a priori* probability. In this study we have considered meta-classes of characters for the estimation of $P(c_h|x_h)$ and $P(c_h)$.

4.2.6.1 Problems of Character Omission and Undersegmentation

The HMMs in the baseline recognition systems includes null transitions that model undersegmentation or the absence of a character within a word (usually due to the misspelling of the word by the writer). Figure 5-4 shows an example of a word with undersegmented characters. In this case, no feature is associated with the character “E” and it is not possible to return to the word image and capture the information relative to such a character for verification purposes. So, our problem is to estimate an *a posteriori* probability of a non-existent character.

We have analyzed different ways to overcome such a problem: ignore the character during the concatenation of character or assign a “dummy” probability to such a character. The alternative which has produced the best results is to use an average

a posteriori character class obtained by using the MLP as a standard classifier. For each character class we compute an “average probability” when the class is correctly recognized by the neural classifier. These “dummy” probabilities are used when there is no segment associated to a character within the word labels due to undersegmentation problems.

Figure 55 shows a similar problem where the character “t” at the prefixes “St” is eliminated during the pre-processing steps because its size and position. In all these case, the baseline recognition system will also associate a null transition, so it will not be possible to return to the word image and capture the information relative to such a character for verification purposes. So, here the “dummy” probabilities are also used.

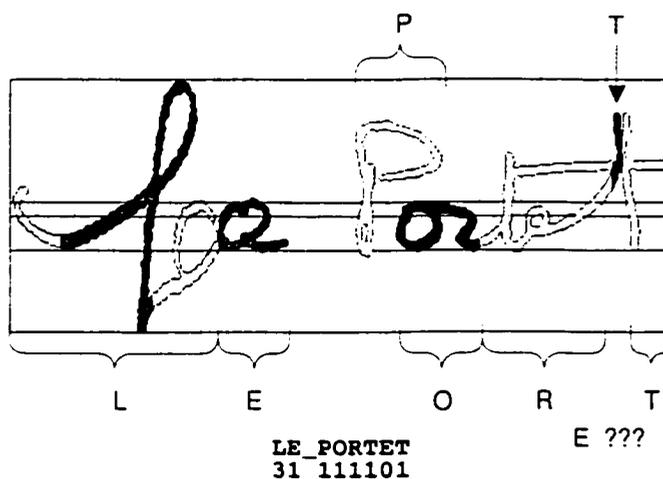


Figure 54 An example of undersegmentation where the characters “or” and “te” are linked within the word “Le Portet”, and during the recognition they are recognized as “O” and “R” respectively. The character “E” is omitted

4.2.6.2 Duration Modeling

Because of the normalization of the features in the feature extraction which transforms characters of any length into a fixed-length representation, the probability

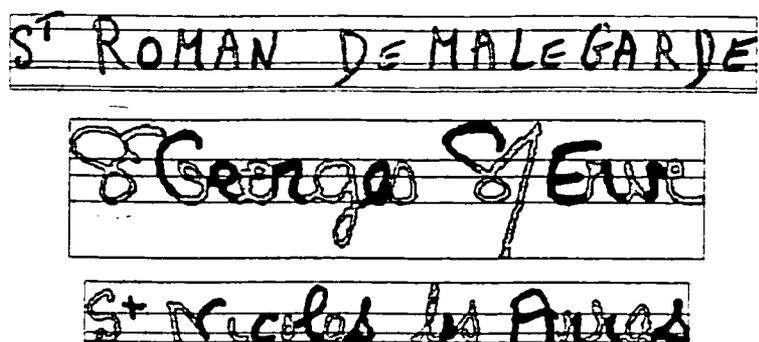


Figure 55 Some examples of character omission where the character “t” is omitted in the prefix “St” for the city names “ST ROMAN DE MALE-GARDE”, “St Georges S Erve”, and “St Nicolas les Arras”

score for a character estimated by the MLP classifier does not depend on the duration of the character.

For this reason and due to the multiplication of probability scores to obtain word probability scores, short words are more likely to have high probability scores, while for long words the probability scores drops. One simple manner to overcome this problem it to associate duration scores to each character class.

In order to provide duration information to each segment scored by the MLP classifier, we constructed a simple durational model. For each character class, we have the probability of splitting a character into 0, 1, 2, 3 or more segments as shown in Table XII in Section 4.1.6.3. Although they do not represent character durations, we can assume these probabilities as duration probability scores and combine them with the scores assigned by the MLP to each character. However, since the neural network classifier deals with metaclasses of characters, the duration probability scores must refer to the duration of metaclasses which is the mean probability between the uppercase and the lowercase probability classes.

In such a way we substitute the segment x_h by the pair $(x_{h,f}, d)$ with $x_{h,f}$ the fixed-

length segment representation and d the duration of the original segment. Hence, instead of $P(c_h|x_h)$ we have $P(c_h|x_{h,f}, d)$, but:

$$P(c_h|x_{h,f}, d) = \frac{P(d|c_h, x_{h,f})P(c_h, x_{h,f})}{P(x_{h,f}, d)} = \frac{P(d|c_h)P(c_h|x_{h,f})}{P(d)} \quad (4.42)$$

since d and $x_{h,f}$ are independent. Hence, to get the overall segment score, we multiply the MLP score $P(c_h|x_{h,f})$ by the duration score $P(d|c_h)$ and divide by the duration probability $P(d)$. So, the Equation 4.41 can be rewritten as:

$$\Psi_{NN} = P(L_n|S_n) = \prod_{h=1}^H \frac{P(c_h|x_h)P(d|c_h)}{P(c_h)P(d)} P(L_n) \quad (4.43)$$

Notice that this is a very rough approximation to the duration model. Therefore, we do not expect significant improvements in the performance of the neural network classifier in terms of word recognition rate.

4.3 Combination of the Word Classifiers Decision

In this section we are particularly interested in combining the outputs of both word classifiers with the aim of compensating the weakness of each individual classifier. This combination is expected to result in better performance (in terms of recognition accuracy) relative to the baseline recognition system alone.

Combination of multiple classifier decisions is a powerful method for increasing classification rates in difficult pattern recognition problems, as is the case of large vocabulary handwriting recognition. Researchers have found that in many situations it is better to combine multiple relatively simple classifiers, generally involving different features and classification algorithms, than try to build a single sophisticated

classifier to achieve better recognition rates [48].

This is partially true in our case, since the baseline recognition system is regarded as a sophisticated classifier that executes a huge task of assigning a class R_v to the input pattern w . On the other hand, the isolated character classifier is simple and it use quite different features and classification strategy. Such a classifier also operated in another decision space and in fact it does not perform a classification because it does not assign a class to the input pattern, but it estimates the *a posteriori* probability of a segment x to belong to a class c . However, by concatenating the probability of the multiple segments x_1, x_2, \dots, x_L it is possible to execute a similar task as the baseline recognition system.

This situation is particularly interesting and unusual because the verifier is tied to the output of the baseline recognition system. So, it is possible to establish a hierarchy among the output of both classifiers. The definition of a verification system is related to such a concept, that is, verifier is a specialized type of classifier devoted to ascertain in a dependable manner whether an input pattern belongs to a given category.

It seems that the most suitable combination rule for such a problem is to use a weighted sum of the logarithm of the scores provided by both classifiers to obtain a single score for each word hypothesis. Changing the weights allows us to adjust the influences of the input recognition scores on the final score. Given the confidence scores of both classifiers, we can obtain a composite score denoted as P_{COMB} by a linear combination of the outputs of both classifiers:

$$P_{\text{COMB}} = \alpha \log(P_{\text{HMM}}) + \beta \log(P_{\text{NN}}) \quad (4.44)$$

where α and β are the weights associated to the baseline recognition system and to

the verification module respectively, and their sum results in:

$$\alpha + \beta = 1 \quad (4.45)$$

In practice, for each word hypothesis of the baseline recognition system, denoted as \hat{L}_n , and for each word hypothesis of the verification module, denoted as L_n , the corresponding confidence scores \hat{P}_n and P_n , are summed to obtain a composite score P_n^* . In order to keep the notation uniform, we define L_n^* as the label of the word hypothesis corresponding to the composite score P_n^* , where $L^* = \hat{L} = L$.

So, we end up with a rescored N -best list that can be reordered based on the composite scores P_n^* . It is expected that the truth hypothesis be shifted up to the top of the list. Since both classifiers generate different confidence scores, we have normalized the output scores between 0 and 1 before the combination as

$$P'_{\text{HMM}} = \frac{P_{\text{HMM}}}{\sum_{n=1}^V P_{\text{HMM}}^n} \quad (4.46)$$

$$P'_{\text{NN}} = \frac{P_{\text{NN}}}{\sum_{n=1}^N P_{\text{NN}}^n} \quad (4.47)$$

where P'_{HMM} denotes the normalized word score provided by the word classifier based on HMMs, P'_{NN} denotes the normalized word score provided by the verification module based on NNs and V and N are the vocabulary size and the number of best word hypotheses respectively. However, the attempts to use normalized scores did not bring any improvement to the final results. In fact, the better results were

obtained by using the raw scores provided by both classifiers.

Figure 56 shows the variation on the word recognition rate of the recognition-verification scheme according to the variation of the weights (α and β) of the combination for the SRTP validation dataset. Optimum performance is achieved for $\alpha = 0.85$ and $\beta = 0.15$.

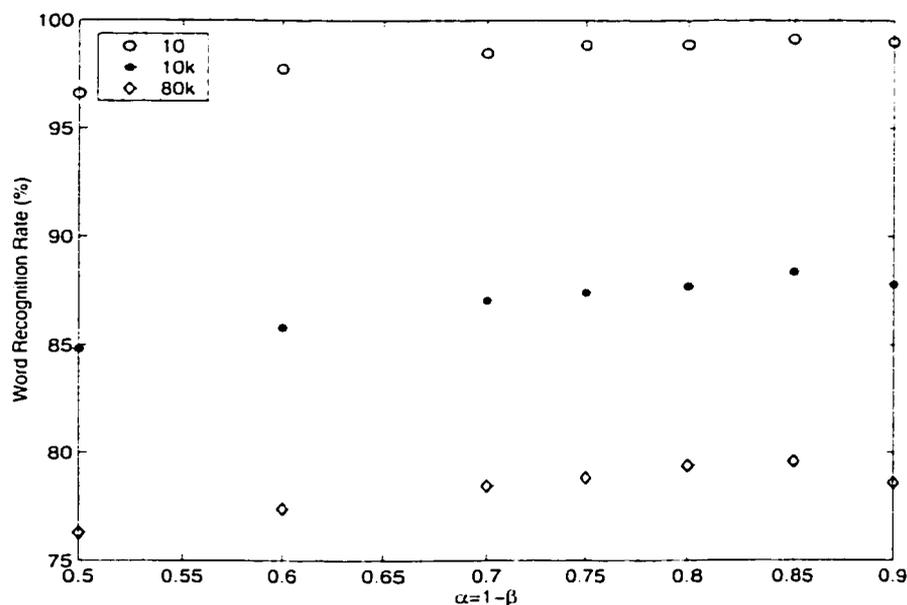


Figure 56 The variation of the word recognition rate of the combination of the HMM-MLP classifiers as a function of the combination weights for the SRTP validation dataset and 3 different sizes of dynamically generated lexicons: 10, 10,000 and 80,000 words

Figure 57 illustrates the complete recognition-verification scheme. Decision on whether to accept or reject the classification of the input pattern is postponed to a last stage where rejection takes place.

Figure 58 shows the 10-best word hypotheses generated by the baseline recognition system with the confidence scores and segmentation information for the input handwritten word "CAMPAGNAC" (Figure 58b). Figure 58c shows the loose segmentation of the word into pseudo-characters produced by the segmentation module

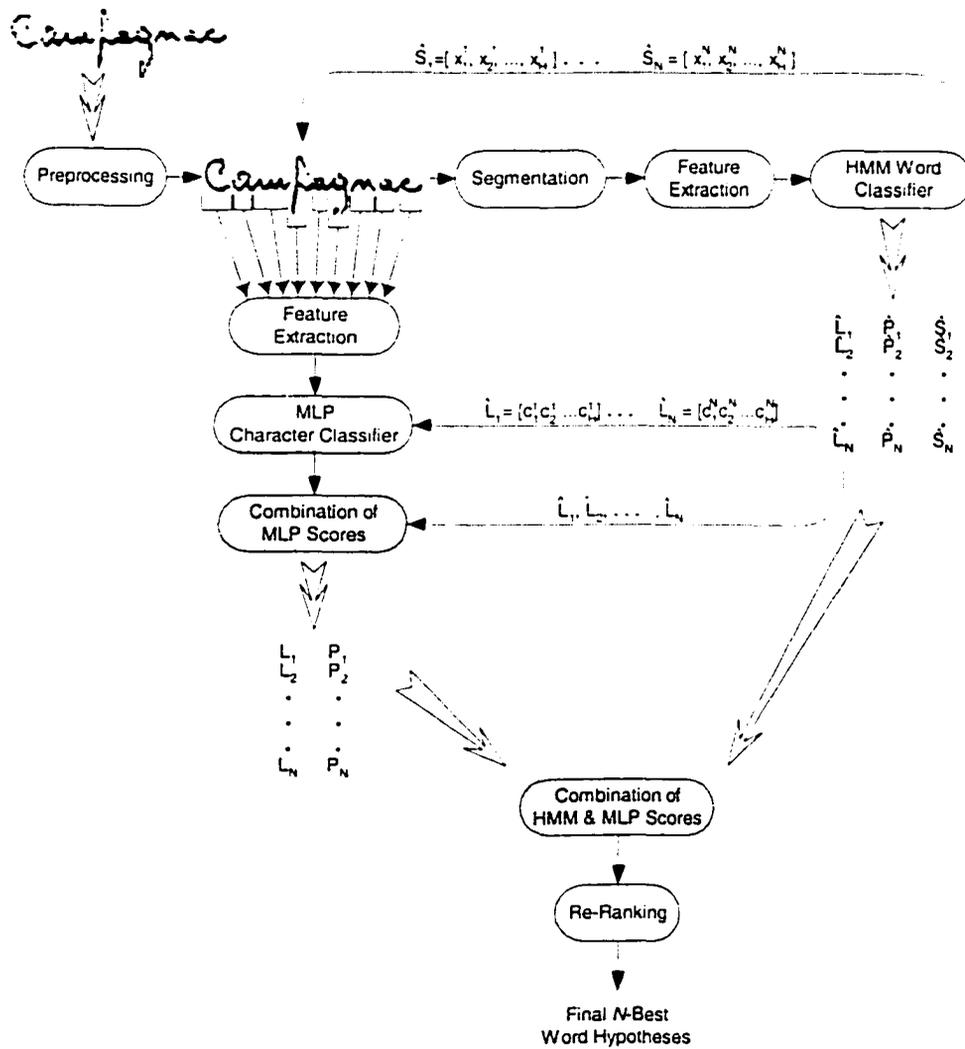


Figure 57 An overview of the integration of the baseline recognition system with the verification module to form a hybrid HMM/NN handwriting recognition system

of the baseline recognition system and 58c shows the final segmentation of the word into characters obtained by using the information provided at the output of the baseline recognition system. The scores assigned to each segment by the MLP classifier of the verification module and the score obtained by summing the logarithm of each segment score is show in Figure 58d. The scores of all 10-best word hypotheses computed by the verification module are shown in Figure 58e.

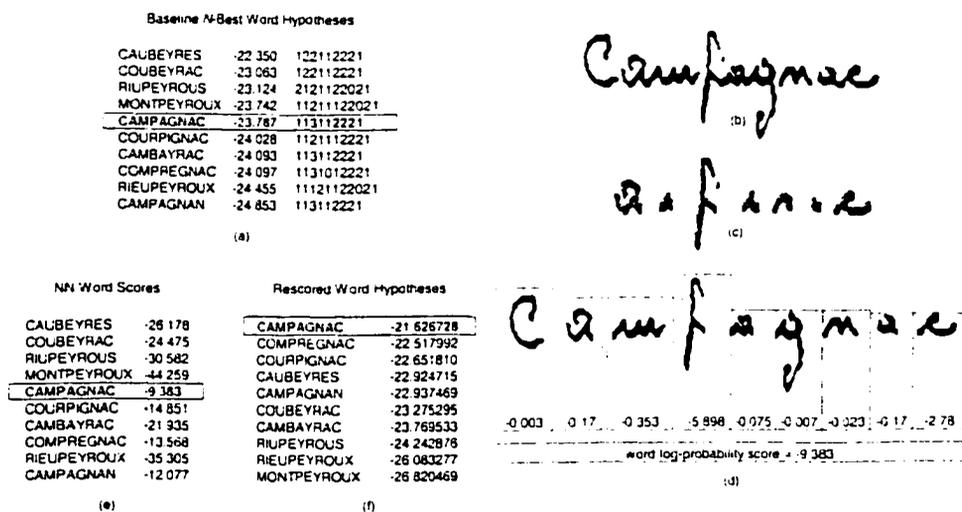


Figure 58 (a) *N*-best list of word hypotheses generated by the baseline recognition system. (b) input handwritten word, (c) loose segmentation of the word into pseudo-characters, (d) final segmentation of the word into characters, scores assigned by the MLP classifier to each segment, and the composite score obtained by summing the scores of each segment. (e) *N*-best list of word hypotheses scored by the MLP classifier, (f) rescored *N*-best word hypotheses

4.3.1 Outline of the Verification Scheme

The integration of the verification module implemented to rescore word hypothesis with the baseline recognition system can be summarized as follows:

- Take one word hypothesis (e.g. $\hat{L}_1, \hat{P}_1, \hat{S}_1$) from the *N*-best word hypothesis list generated by the baseline recognition system;

- For each pair (x_h, c_h) go to the word image and extract features from such a segment:
- Use the MLP classifier to “score” the segment x_h and take the output of the MLP that corresponds to the class c_h :
- At the end, when all segments corresponding to a word hypothesis have been scored, combine their scores to obtain a composite score for the word \hat{L}_1 , denoted as P_1 :
- Repeat the same procedure for all other word hypotheses in the N -best word hypothesis list:
- At the end, for each word hypothesis, combine the scores given by the baseline recognition system (\hat{P}_n) and the scores given by the verification module (P_n):
- Based on the combined scores, denoted as P_n^* , rerank the N -best word hypothesis list:
- End.

4.3.2 Summary of the Verification of Handwritten Words

We have presented a verification scheme that rescores the N -best word hypothesis list provided by the baseline recognition system in an attempt to improve the performance in terms of recognition accuracy. The N -best word hypothesis list provides the word hypotheses, the corresponding likelihoods and segmentation into characters. The first step of the verification is to assign a Bayesian *a posteriori* probability to each segment given their classes. Next, by taking the probabilities assigned to all segments that form each word hypothesis, a new probability score to such a hypothesis is obtained. At the second step, the scores generated by the verification module are combined with the scores generated by the baseline recognition system. Based on such composite scores, the N -best word hypothesis list is re-ranked.

The recognition/verification scheme is efficient because it combines two different classification strategies and operated in two representation spaces (word and character) that are regarded to be complementary. Furthermore, the verification scheme is very fast since it employs a neural network classifier. So, it does not introduce significant delay into the overall recognition process (see Section 5.4.7).

One drawback of this verification scheme is that it depends on the output of the baseline recognition system. If the truth word hypothesis is not present among the N -best word hypotheses, the verification becomes useless. At least, it is not able to improve the recognition accuracy. However, this problem can be alleviated by using a great number of word hypotheses instead of only the ten best word hypotheses.

In the next chapter the performance of the verification module is presented and its effectiveness to improve the recognition accuracy is demonstrated in practice.

4.4 Rejection Mechanism

Conventionally, classifiers employed in handwriting recognition perform imperfect recognition, that is, they produce an output that does not correspond to the input data. So, the question is not only to find a solution, but most importantly to find out if the word hypothesis provided by the recognizer is trustworthy. That is why the many word hypotheses are provided and they may indicate a certain degree of incorrectness in the output of the recognizer.

Conventionally, classifiers employed in handwritten word recognition do not produce a unique output, but a list with the best hypotheses. The reason for that is that the likelihoods or confidence scores of the $TOP.N$ word hypotheses may be similar or not. This information is used as a guide to establish a rejection criteria for example, or even to serve as the input of a second level of decision.

4.4.1 Principles of the Rejection Mechanism

Statistical theory suggests rejecting an input if all probability scores for that input are less than a threshold. Such a rejection rule can be directly implemented by using classifier outputs as probability scores and rejecting an input if all outputs are below a threshold. Figure 59 shows an overview of the proposed rejection scheme which is based on the confidence scores of the *TOP* 1 and *TOP* 2 word hypotheses.

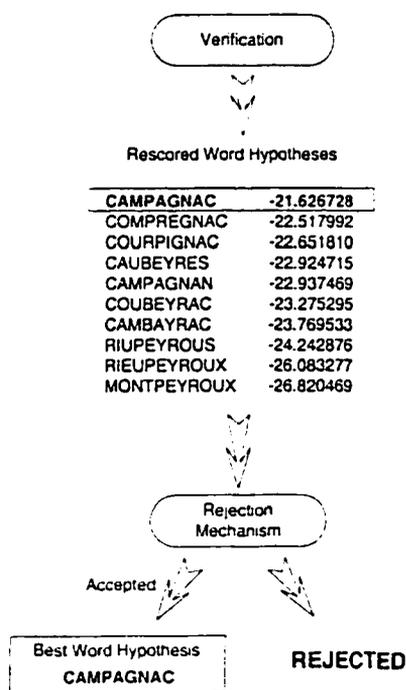


Figure 59 An overview of the rejection mechanism and the final recognition result either accepting or rejecting an input image

The main task of the rejection mechanism shown in Figure 59 is to decide whether the word hypothesis at *TOP* 1 position can be accepted or not. The output of the recognition-verification system, denoted as Ψ_{COMB} can be represented by:

$$\Psi_{\text{COMB}} = \{L_n^*, P_n^*\} \quad (4.48)$$

where L_n^* is the label for the n -th word hypothesis and P_n^* is the composite confidence score assigned to the n -th word hypothesis.

We have adopted a very simple rejection criterion based on the confidence scores estimated by the recognition-verification system and a single threshold denoted as R_{th} , which is defined as:

$$R_{th} = P_1^*(1 + \kappa) \quad (4.49)$$

where $\kappa \in [0, 1]$ is a threshold that indicates the amount of variation of the score of the *TOP* 1 word hypothesis and P_1^* is the score of the *TOP* 1 word hypothesis.

Then rejection can be established by requiring:

$$P_2^* \leq R_{th} \quad (4.50)$$

where P_2^* is the score of the *TOP* 2 word hypothesis.

By varying the threshold κ , the proportion of word hypotheses rejected can be controlled. Figure 60 shows the word recognition rate (*TOP* 1) as a function of rejection rate for the baseline recognition system alone and the combination of the baseline recognition system with the verification module on the SRTP validation dataset for 10k-word dynamically generated lexicon. Note that higher recognition rates are obtained by the recognition-verification system, confirming our hypothesis that the use of a verification module also collaborates to improve the reliability of the recognition process.

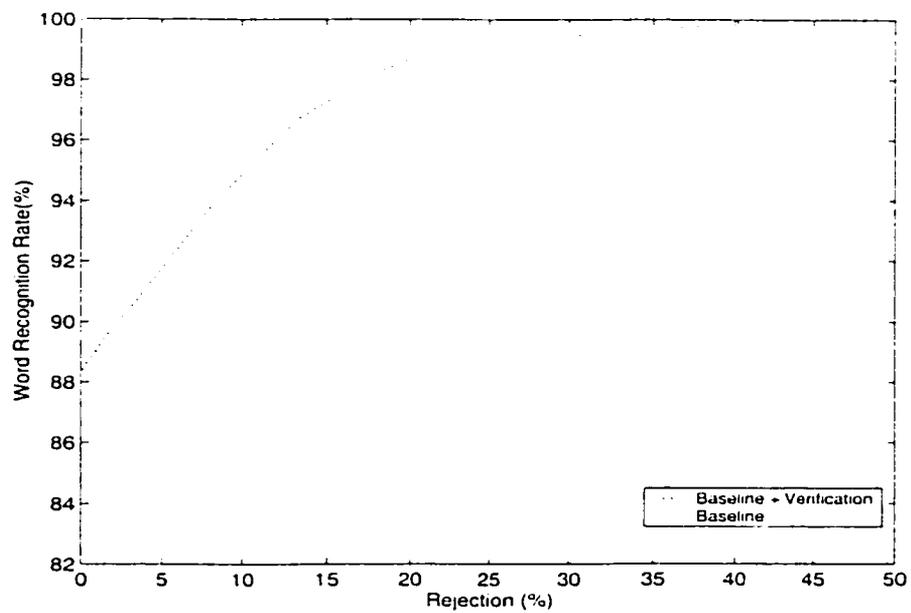


Figure 60 Word recognition rate (TOP 1) as a function of rejection rate for the baseline recognition system alone and the baseline + verification on the SRTP validation dataset and 10k-word dynamically generated lexicon

4.4.2 Summary of the Rejection Mechanism

Rejection is a powerful method for reducing error rate. However, if the rejection mechanism uses the probability scores resulting from the combination of the baseline recognition system with the verification module, the results can be significantly improved.

For example, by rejecting 20% of the word hypotheses with the lowest difference in the confidence scores between the *TOP* 1 and *TOP* 2 word hypotheses, the *TOP* 1 recognition rate is increased from about 82% to about 93% considering only the baseline recognition system and from about 88% to about 99% considering the combination of the baseline recognition system and verification module (Figure 60).

In the next chapter the performance of the rejection mechanism is presented and its effectiveness in improving the recognition reliability is demonstrated in practice.

CHAPTER 5

PERFORMANCE ANALYSIS

This chapter presents the experiments undertaken during the development of the algorithms and methods presented in Chapter 4 and the results obtained by applying the strategies and methods proposed to the problem of large vocabulary handwriting recognition.

First we describe the performance measurements used to evaluate the algorithms and the databases used for such an aim. Next we evaluate each strategy proposed in Chapter 4 to deal with the problem of recognition speed. We evaluate the performance of the baseline recognition system (BLN), and the implementations based on the strategies that we have proposed: lexicon-driven level building algorithm (LDLBA), the constrained level building algorithm (CLBA), the class dependent constrained level building algorithm (CDCLBA), the fast two-level HMM decoding (FS), and the distributed recognition scheme (DS). A summary of the speed improvements obtained by the proposed methods and a comparison with the results obtained by the baseline recognition system is presented.

Following the improvements in recognition speed, the strategy to improve the accuracy in recognizing handwritten words based on the recognition of isolated characters is presented. First, the results of three methods to recognize isolated handwritten characters, say, k -nearest prototypes (k -NP), k -nearest neighbors (k -NN), and neural networks (NN) are presented and compared. Next we evaluate the use of the NN classifier in the recognition of handwritten words and its integration with the handwritten word recognition system based on HMMs. Rejection is introduced and the performance of the hybrid HMM/NN recognition system is assessed under different rejection levels.

5.1 Measuring Performance

This thesis presents the results of a number of experiments aimed at testing the usefulness of methods described. Since there is usually no direct, objective measure of effectiveness of one method compared with another, methods are compared by training once the character HMMs and testing on an unseen test set. The final results obtained are word recognition rates showing the percentage of words in the test set correctly classified by the whole system as well as the time lapse required to accomplish the recognition.

It is important that the data used to verify a concept or hypothesis has not been used in any form while building the recognizer. Therefore, we have used three sets of data used to build a recognition system: training dataset, validation dataset, and test dataset.

- The *training dataset* is used to build the recognizer and adjust the parameters. The amount of required training data depends on the recognition approach. For most recognition systems, more training data will yield better recognition results [105].
- The *validation dataset* is used to evaluate new algorithms during the development phase of the recognizer. Since many decisions are made based on this data, it becomes *contaminated*; the decisions may not be independent of the validation data, and the resulting performance may be higher on the validation data than on completely unseen data;
- The *test dataset* is used for the final assessment of the system should be therefore be unseen. This means none of the system's parameters has been adjusted to this data, and no decision was ever made based on recognition results on this data.

Although we have mainly focused on large vocabulary handwriting recognition in

this thesis, it is interesting to consider how the techniques developed here extend to smaller vocabulary tasks. For this reason, the experimental results presented through this chapter comprise different sizes of vocabulary.

5.1.1 Recognition Accuracy

Because the recognition process is found to be dependent on the words presented in the lexicons, results are subject to a certain amount of variation. A statistically correct approach would suggest to conduct many experiments and estimate means and standard deviations. However, it would be very computationally expensive to conduct multiple experiments for each strategy and lexicon size presented in this thesis. In experiments where only one run has been carried out, standard deviations estimated from multiple runs under similar conditions are quoted. In fact, we have adopted exactly the same conditions for all experiments, that is, the same trained models and dynamic lexicons.

The criteria to evaluate the recognition accuracy is the recognition rate which is given as:

$$\text{Recognition Rate} = 100 \cdot \frac{\text{number of correctly recognized words}}{\text{total number of words}} \quad (5.1)$$

and the Error Rate is defined as 100-Recognition Rate.

5.1.2 Recognition Time

The recognition time is defined as the time in seconds required to recognize one word and it is measured in CPU-seconds, which is the time the recognition process has exclusive use of the central processing unit of a computer with a multitasking operating system such as UNIX. However, the recognition time depends on the

computer and compiler used for the experiments. In this chapter, the recognition time is always given in CPU-seconds and it covers only the recognition process, excluding pre-processing, segmentation and feature extraction steps.

The criterion that we have used to evaluate the recognition time of each strategy is the running time of the software implementation. All results are averaged over the test set of 4,674 words and ten runs. Along this research several different machines were used:

- SUN Ultra1, 167MHz-UltraSparc processor, 128MB of RAM memory;
- SUN Enterprise 6000, SMP architecture, 14×167MHz-UltraSparc processors, 1.75GB of RAM memory;
- PC AMD Athlon 1.1GHz, 512MB of RAM memory.

It is very important to notice that the recognition time also depends on the software implementation, for example, if some data is kept in disk instead of in RAM, the access time is slower, and consequently, the recognition speed is affected. The implementations of all search algorithms were kept as similar as possible, so they share many parts of code, including disk operations. We have tried to modify only the parts concerning the search mechanisms.

5.2 Databases

A database is an essential requirement in pattern recognition for the development, evaluation and comparison of different techniques. A database must provide a enough number of sample to allow the training and testing of pattern recognition systems with some statistical confidence. During the developing of our research, we have used the NIST and the SRTP databases which are described as follows.

5.2.1 NIST Database

The NIST Special Database 19 (SD19) contains 3,699 handwritten filled binary images (HSF's) and 814,255 samples of isolated alphanumeric characters. Usually, the sets *hsf0*, *hsf1*, *hsf2*, and *hsf3* are used for training, the *hsf7* for validation and the set *hsf4* for testing. The set *hsf4* is also called *TDI*. Considering all sets, we have 178,793 lowercase characters and 208,066 uppercase characters.

The NIST database is not representative of the context (words), since the data was gathered in different conditions. However, performance on a standard dataset does give an indication of the usefulness of the features and provides performance figures that can be compared with the results obtained by other research groups. The NIST database has been used in the early development of the isolated character recognition strategies.

The distribution of digits and characters among the 52 valid classes (a-z, A-Z) and among the three datasets (learning, validation and testing) is given in Tables XXI-XXIII.

5.2.2 SRTP Database

The SRTP database is composed of more than 40,000 binary images of real postal envelopes digitized at 200dpi (dots per inch) of resolution as shown in Figure 14 (Chapter 2). From these images, three datasets were generated as shown in Table XXIV which contain city names manually located on the envelopes. Table XXIV also presents other statistics of the datasets. It is worthwhile to mention the presence of compound words in the datasets (e.g. "*Chire en Montreuil*").

The first problem that we have to deal with in developing the verification module is to build a database of isolated characters. The idea is to use the same datasets that has been used to the development of the baseline recognition system. However,

Table XXI

Training dataset of NIST database (184,033 uppercase + 155,215 lowercase characters)

Character Class	Number Samples						
A	6,085	N	8,192	a	10,205	n	11,943
B	3,191	O	27,737	b	4,642	o	1,800
C	10,221	P	8,340	c	1,794	p	1,589
D	4,033	Q	1,664	d	10,497	q	2,405
E	4,652	R	4,528	e	27,408	r	15,050
F	9,325	S	22,898	f	1,608	s	1,776
G	1,707	T	9,974	g	3,050	t	19,861
H	2,425	U	13,224	h	8,755	u	1,826
I	11,847	V	3,945	i	1,838	v	1,777
J	3,063	W	4,084	j	1,446	w	1,765
K	1,662	X	1,785	k	1,740	x	1,861
L	4,341	Y	4,206	l	15,420	y	1,614
M	9,216	Z	1,778	m	1,731	z	1,814

Table XXII

Test dataset (TD1 dataset) of NIST database (11,941 uppercase + 12,000 lowercase characters)

Character Class	Number Samples						
A	459	N	439	a	481	n	460
B	435	O	459	b	461	o	454
C	518	P	467	c	494	p	415
D	396	Q	452	d	439	q	384
E	365	R	446	e	424	r	491
F	419	S	445	f	468	s	438
G	389	T	469	g	437	t	434
H	402	U	458	h	504	u	475
I	815	V	482	i	364	v	524
J	426	W	475	j	293	w	465
K	377	X	472	k	395	x	472
L	496	Y	453	l	916	y	387
M	460	Z	467	m	475	z	450

Table XXIII

Validation dataset of NIST database (12,092 uppercase + 11,578 lowercase characters)

Character Class	Number Samples						
A	449	N	471	a	490	n	449
B	439	O	472	b	454	o	483
C	525	P	465	c	486	p	410
D	466	Q	450	d	460	q	351
E	390	R	452	e	457	r	437
F	428	S	467	f	433	s	465
G	425	T	457	g	326	t	451
H	421	U	472	h	483	u	493
I	740	V	492	i	406	v	534
J	429	W	467	j	205	w	466
K	411	X	471	k	404	x	479
L	500	Y	429	l	697	y	367
M	450	Z	454	m	450	z	442

Table XXIV

The number of samples (city name images) in each dataset of the SRTP database

Dataset	Number of Samples	Word Average Length	Number of Different Words
Training	12,023	10.69	4,814
Validation	3,475	11.64	1,392
Test	4,674	11.14	2,540

in the database provided by the SRTP–France only the fields of the envelopes are segmented and labeled, that is words, not individual characters. Segmenting and labeling data is expensive, time-consuming and error prone. But unfortunately, we have to consider the contextual information embedded in the characters within the words, so, building the database is essential to the development of the verification module.

5.2.2.1 Construction of an Isolated Character Database from Handwritten Words

In the SRTP database, the words and sentences are already labeled, but the information related to the segmentation is not available. However, we can obtain the information about the segmentation from the output of the baseline recognition system with some degree of reliability by using bootstrapping as illustrated in Figure 61. Generally, segmentation of word in characters is done by hand, but we can adopt a procedure similar to that adopted by Kimura et al. [81]. They have used a semi-automatic interactive procedure utilizing a lexicon of word to segment the characters from handwritten words. In our case, with bootstrapping, we can use the baseline recognition systems to label and segment the words into characters and build an isolated character database in an automatic fashion. By doing so, we have obtained more than 100,000 isolated characters from the handwritten words. Tables XXV–XXVII show the distribution of characters among the 52 valid classes (a-z, A-Z) and among the three datasets (training, validation and test).

However, simply using the word recognizer to segment words in characters and to assign a class/label for each segment has shown to be a very inefficient strategy, since doing that, we do not take into account that some of the segments may be related to undersegmented or oversegmented characters. In such cases, we wrongly attributed a label to the segments.

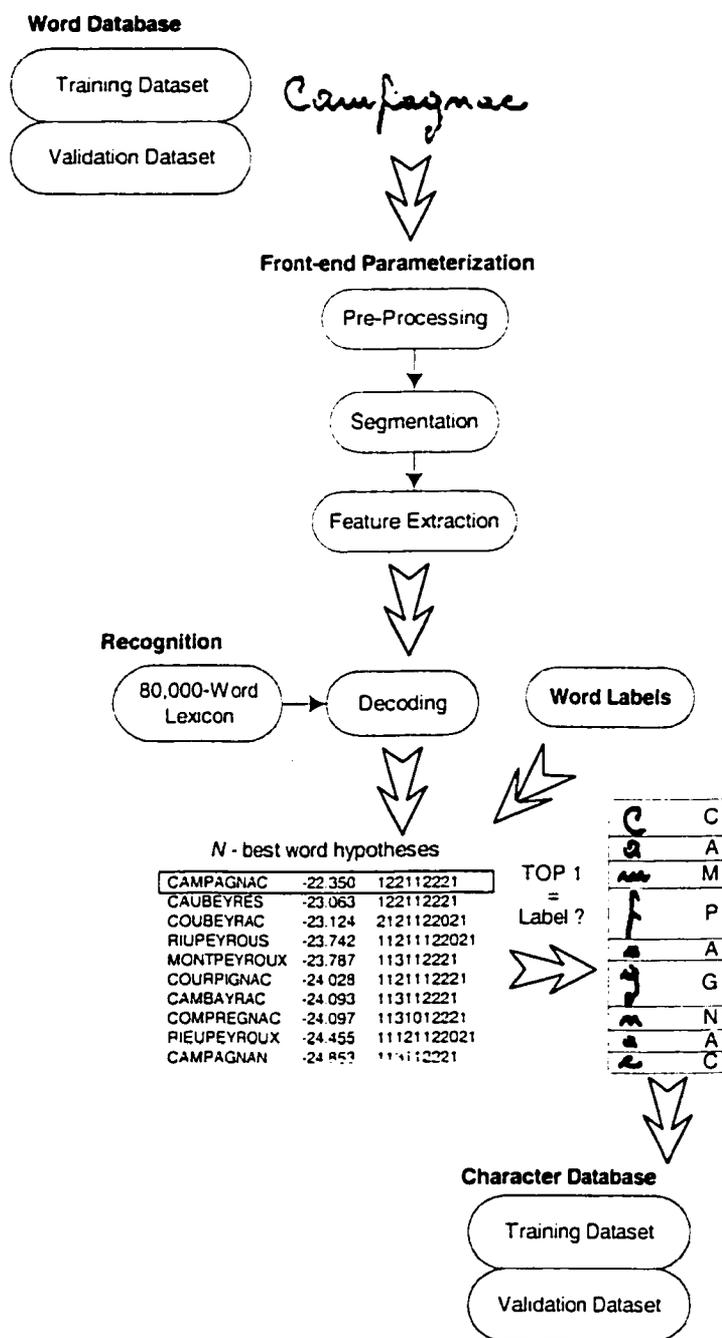


Figure 61 The bootstrapping process to automatically generate a character database from unconstrained handwritten words of the SRTP database

Table XXV

Training dataset of the SRTP database (115.088 characters from 12.022 words)

Character Class	Number Samples						
A	4.679	N	4.499	a	4.494	n	4.760
B	1.938	O	3.184	b	667	o	3.609
C	4.701	P	1.163	c	1.542	p	360
D	1.578	Q	152	d	2.660	q	90
E	7.239	R	4.459	e	11.354	r	4.033
F	524	S	4.676	f	147	s	3.402
G	1.167	T	2.242	g	870	t	2.159
H	890	U	2.136	h	814	u	2.842
I	2.875	V	1.351	i	3.580	v	640
J	259	W	35	j	92	w	7
K	17	X	1.035	k	15	x	2.317
L	1.317	Y	773	l	3.519	y	835
M	2.178	Z	261	m	744	z	217

So, to increase the quality of the samples in our character database, we decided to consider only the words that are correctly recognized in a very selective task where a 80.000-word vocabulary was used. By doing so, we successfully eliminated most of the segmentation problems, and we ended up with a relatively cleaner dataset.

The segmentation provided by the baseline recognition system is somewhat reliable, but not perfect, even if the word is correctly recognized (as *TOP 1*), it does not mean that the alignment is also correct and each segment corresponds to a whole character. Due to the characteristics of the baseline recognition system that rely on the context (whole word) during the recognition process, it occurs that fragments of characters (oversegmented characters), characters joined with fragments of neighbor characters, or even two joined characters (undersegmented characters) be considered as a single character. If we want to use this dataset to build a classifier it is necessary to eliminate as much as possible all these elements that do not correspond to isolated

Table XXVI

Validation dataset of the SRTP database (36,170 characters from 3,475 words)

Character Class	Number Samples						
A	1,191	N	1,060	a	1,270	n	1,240
B	570	O	941	b	348	o	1,133
C	1,813	P	230	c	558	p	97
D	564	Q	90	d	1,284	q	118
E	2,163	R	1,325	e	4,213	r	885
F	96	S	974	f	19	s	853
G	287	T	518	g	234	t	405
H	215	U	732	h	173	u	967
I	979	V	320	i	1,230	v	185
J	27	W	64	j	11	w	2
K	34	X	600	k	20	x	1,418
L	1,683	Y	193	l	1,181	y	273
M	516	Z	69	m	187	z	105

characters because they may cause misleading and confusions.

To carry out a visual inspection of all images to check if they really correspond to the label assigned by the baseline recognition system would be a very time-consuming task, since approximately 200,000 images have to be inspected. To avoid such a tedious task, we have used two methods to eliminate some of the garbage:

- The size of the image files give a good evidence: images with small sizes generally refer to fragments of characters; images with large sizes generally refer to undersegmented characters (at least a whole character and a fragment of another neighbor characters;
- Use an isolated character classifier to pre-classify the images and give us a hint about possible segmentation problems; To accomplish such a task we have used a classifier based on a MLP trained on the NIST database [103]. The reason

Table XXVII

Test dataset of the SRTP database (46,700 characters from 4,674 words)

Character Class	Number Samples						
A	1,733	N	1,725	a	1,820	n	1,821
B	751	O	1,210	b	255	o	1,414
C	2,096	P	465	c	672	p	143
D	676	Q	52	d	1,315	q	34
E	2,962	R	1,740	e	5,108	r	1,561
F	193	S	1,702	f	66	s	1,510
G	483	T	817	g	363	t	871
H	287	U	774	h	278	u	1,159
I	1,062	V	507	i	1,487	v	290
J	95	W	23	j	29	w	4
K	10	X	500	k	6	x	1,221
L	1,644	Y	300	l	1,433	y	369
M	819	Z	69	m	248	z	99

for having used the NIST database is to have a certain level of independence between the cleaning process and the development of the classifiers. Just as an illustration, Table XXVIII shows the recognition results of the samples of the SRTP training, validation, and test datasets by a neural classifier based on MLP.

However, the number of words was reduced from 12,022 to 8,043 and as a consequence, the number of samples for certain character classes is as low as 5 while for others is as high as 8,800. Table XXIX shows the number of samples per class for the cleaned training dataset. Note that this procedure to cleaning the database was applied only on the training dataset. Validation and test datasets were not cleaned. Figure 62 shows some examples of bad characters shapes that were eliminated from the database during the cleaning process.

Table XXVIII

Character recognition rates for isolated characters of SRTP database by using a NN classifier trained on the training dataset of NIST database

Dataset	Character Recognition Rate (%)
Training	49.4
Validation	48.3
Test	50.1

Table XXIX

Cleaned training dataset of the SRTP database (43.365 uppercase + 41.446 lowercase characters = 84.811 characters from 8.043 words)

Character Class	Number Samples						
A	3,384	N	3,321	a	3,217	n	3,347
B	1,372	O	2,390	b	499	o	2,586
C	3,814	P	794	c	1,166	p	269
D	1,224	Q	112	d	2,317	q	65
E	5,482	R	3,268	e	8,824	r	2,870
F	404	S	3,365	f	119	s	2,313
G	904	T	1,644	g	629	t	1,588
H	671	U	1,599	h	609	u	2,120
I	2,138	V	992	i	2,537	v	435
J	186	W	27	j	64	w	5
K	13	X	893	k	13	x	2,027
L	3,084	Y	558	l	2,574	y	589
M	1,537	Z	189	m	523	z	141

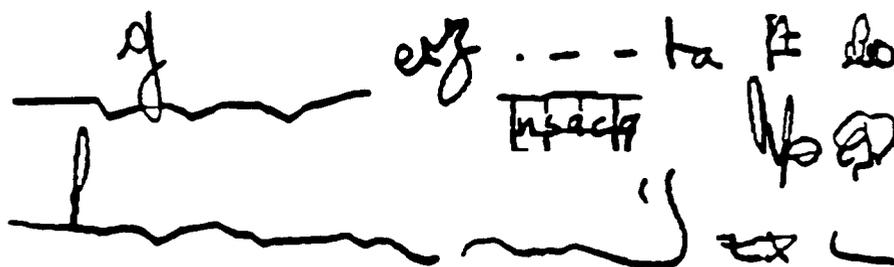


Figure 62 Some examples of bad characters shapes that were eliminated from the database during the cleaning process

5.3 Improving Recognition Speed

In this section we present the experiments undertaken during the development of the fast recognition strategies presented in Chapter 4. First we describe the performance measurements used to evaluate the algorithms and the datasets used for such an aim. Next we evaluate each strategy proposed in Chapter 4 to deal with the problem of recognition speed. We evaluate the performance of the baseline recognition system adapted to the following recognition strategies: lexicon-driven level building algorithm (LDLBA) presented in Section 4.1.5, the constrained level building algorithm (CLBA) presented in Section 4.1.6, the class dependent constrained level building algorithm (CDCLBA) presented in Section 4.1.6.3, the fast two-level HMM decoding (FS) presented in Section 4.1.7, and the distributed recognition scheme (DS) presented in Section 4.1.8. A summary of the speed improvements and a comparison of results obtained by the many methods with the performance of the baseline SRTP recognition system is presented.

5.3.1 Testing Conditions

All experiments presented in this section were performed under the following conditions:

- Seventy-two HMMs corresponding to 26 uppercase letters (A-Z), 26 lowercase letters (a-z), 10 digits (0-9) and 8 special symbols were trained with a dataset of 12,092 words. A validation set of 3,769 words was also used during the training procedure which is describe in Chapter 3. Note that the HMMs were trained once and they were used with all algorithms and search methods presented through this thesis;
- After training, parameters required to optimize the algorithms' performance were always adjusted using the validation dataset;
- A test set with 4,674 words was used for evaluation and comparison of search strategies. All the results presented in the tables through this section refer to such a test set, except when mentioned otherwise in context;
- A lexicon with 36,016 French city names was used in all recognition experiments and dynamic lexicons with different sizes were randomly generated from this lexicon. The only exception is in the Section 5.3.7 where a larger lexicon with 85,092 entries was used;
- All experiments were repeated 10 times for ten different dynamic lexicons and the recognition rates and recognition times in tables are the average values over the 10 runs, except when mentioned otherwise in context. The standard deviations are not presented in tables because they are non significative (lower than 0.5% for the recognition rate; lower than 5% for the recognition time);
- All experiment were conducted on the same machine, a SUN Ultra1, 167MHz, 128MB RAM except when mentioned otherwise in context.

5.3.2 Recalling the Baseline Recognition System Performance (BLN)

The performance of the baseline recognition system was already presented in Chapter 3. However, since in this section we will frequently refer to it to compare with the performance of other recognition approaches, it is reproduced in Table XXX.

Table XXX

Word recognition rate and recognition time for the baseline recognition system (BLN) on the SRTP test dataset

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)
	TOP 1	TOP 5	TOP 10	
10	98.93	99.93	100	0.222
100	95.89	98.99	99.40	1.989
1k	89.79	95.97	97.30	19.50
10k	79.50	89.53	91.89	182.5
30k	73.70	85.17	88.15	493.1

5.3.3 Lexicon-Driven Level Building Algorithm (LDLBA)

The first attempt to improve the performance of the baseline recognition system is by representing the lexicon as trie and by pursuing the search only by taking the character with the best partial likelihood, as presented in Section 4.1.5. Table XXXI summarizes the results for the recognition accuracy and recognition time for the recognition system based on the LDLBA, considering the same five dynamic lexicons (10, 100, 1k, 10k, and 30k entries) used to evaluate the performance of the baseline recognition system in Chapter 3. Table XXXI also lists the speedup obtained over the baseline SRTP recognition system (BLN).

Clearly the LDLBA decoding is several times faster than the baseline recognition system (BLN) with speedup factors between 5.5 and 7.7 for 10-word and 30k-word lexicons respectively. The relative reduction in recognition rates, compared to the baseline recognition system, is not very significant as shown in Table XXXII. The average loss of accuracy is 1.068%, 0.568%, and 0.496% for the *TOP 1*, *TOP 5*, and *TOP 10* best choices respectively. However, it can be argued that the speedup afforded by the LDLBA is well worth the increase in error rate.

Table XXXI

Word recognition rate and recognition time of the system based on the level building algorithm and a lexical tree (LDLBA) and speedup over the Viterbi flat lexicon of baseline recognition system (BLN)

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	Speedup (\times BLN)
	TOP 1	TOP 5	TOP 10		
10	98.76	99.93	100	0.040	5.5
100	95.46	98.82	99.40	0.354	5.6
1k	89.00	95.49	96.81	3.091	6.3
10k	78.22	88.49	90.99	24.75	7.4
30k	71.03	84.02	87.06	64.16	7.7

Table XXXII

Difference in the word recognition rates between the system based on the level building algorithm and a lexical tree (LDLBA) and the baseline recognition system (BLN)

Lexicon Size	Word Recognition Rate (%)		
	(BLN-LDLBA)		
	TOP 1	TOP 5	TOP 10
10	0.17	0	0
100	0.43	0.17	0
1k	0.79	0.48	0.49
10k	1.28	1.04	0.90
30k	2.67	1.15	1.09
Average	1.063	0.568	0.496

We attribute the reduction in recognition rate to the the level reduction of the LBA that selects only the best character model at each word position and pursue the search considering only such a best model. On the other hand, the Viterbi decoding of the baseline recognition system with a flat lexical structure decode all hypotheses. This would also be possible with the tree structure, however the number of hypotheses to decoded will make the advantages of using a lexical tree not meaningful.

The problems in the search due to the level reduction is pronounced specially on short words. Considering that the average length of the words in the test set is 11.14 characters, 83% of the search errors account for words with less than 6 characters.

More important, we shall see in the subsequent sections of this chapter that the loss in accuracy can be completely recovered (and even improved) by efficiently postprocessing the list of *TOPN* word hypotheses. Another important remark is that in principle, there are no parameters to adjust in the LDLBA to either improve its accuracy or speed. Its performance depends only on the quality of the character models.

5.3.4 Time and Length Constraints (CLBA)

The inclusion of some constraints to the LDLBA yields the constrained level building algorithm (CLBA) presented in Section 4.1.6. It is an attempt to further reduce the processing time while preserving the recognition accuracy. These constraints are particularly interesting to establish a trade-off between recognition accuracy and recognition time.

We have chosen as goal to obtain the maximum speedup of the recognition process while reducing the accuracy of the LDLBA no more than 0.5%. We have used a statistical experimental design method to determine the values of the control

factors s^* , e^* , and Lv^* while D_{max} was kept constant and equal six since this is the maximum number of observations that the character HMMs can emit [86] (due to the HMM architecture). To this aim we have used three regression models where the independent variables are the three control factors, and the dependent variables are the responses of the recognition system: recognition rate and recognition time. Afterwards, an L_9 orthogonal array was employed to gain information on the control factors and to determine the coefficients of the regression models. Based on these regression models, the optimal values of the control factors that jointly optimize both the accuracy and the speed were determined [86]. Several experimental runs were conducted, corresponding to the different values of the control factors and both recognition rate and recognition time were measured. In these experiments, we have used the validation set. Table XXXIII shows the values of the constraints for different lexicon sizes resulting from the optimization step. With the three control factors set up as in Table XXXIII, the performance of the CLBA was evaluated over the test set and the results are shown in Table XXXIV.

Table XXXIII

Values of the constraints of the CLBA determined by the statistical experimental design technique

Lexicon Size	Constraints		
	Lv^*	e^*	s^*
10	0.18	0.5	-0.5
100	0.68	0.5	-0.5
1k	0.87	0.5	-0.5
10k	0.85	0.5	-0.5
30k	0.89	0.5	-0.5

Table XXXV shows the reduction in recognition rates relatively to the baseline recognition system. The constraints added to the LBA brings about an average loss in accuracy of 0.394%, 0.392%, and 0.342% for the *TOP 1*, *TOP 5*, and *TOP 10*

Table XXXIV

Word Recognition rate and recognition time for the system based on the constrained level building algorithm and a lexical tree (CLBA)

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	Speedup (\times BLN)
	TOP 1	TOP 5	TOP 10		
10	98.50	99.85	100	0.028	7.8
100	94.95	98.59	99.19	0.243	8.2
1k	88.42	95.04	96.41	2.059	9.5
10k	77.60	87.89	90.37	16.32	11.1
30k	71.03	83.42	86.58	41.81	11.8

best choices respectively when compared with the performance of the LDLBA. On the other hand, if we compare the speedup shown in Table XXXIV, the CLBA is faster than the LDLBA with speedup factors between 1.42 and 1.53. The average loss in accuracy is 1.462%, 0.96%, and 0.838% for the *TOP 1*, *TOP 5*, and *TOP 10* best choices respectively when compared with the performance of the baseline recognition system. Comparing the performance of the CLBA with the baseline recognition system in terms of recognition time, we have speedup factors between 7.8 and 11.8. The speedup afforded by the CLBA is well worth the increase in error rate.

The errors introduced by the CLBA are due to the constraints. We have investigated the effects of each constraint on the recognition accuracy and recognition time separately. We have found that the constraint Lv^* does not introduce search errors for any lexicon size. The speedup obtained due to this constraint is not very expressive for small lexicons ($\leq 1,000$ words) but it becomes very interesting for larger lexicons. On the other hand the constraints s^* and e^* have a strong influence on the recognition accuracy and recognition time independently of the lexicon size. Table XXXVI shows in an abbreviate manner the individual contributions of s^* and

Table XXXV

Difference in the word recognition rates between the system based on the constrained level building algorithm and the baseline recognition system (BLN)

Lexicon Size	Word Recognition Rate (%) (BLN-CLBA)		
	TOP 1	TOP 5	TOP 10
10	0.43	0.08	0
100	0.94	0.40	0.21
1k	1.37	0.93	0.89
10k	1.90	1.64	1.52
30k	2.67	1.75	1.57
Average	1.462	0.96	0.838

Lv^* on the recognition accuracy and speed of the LDLBA.

Table XXXVI

Individual influence of the control factors s^* and Lv^* on the recognition rate ($TOP 1$) and on the recognition time of the LDLBA

Lexicon Size	Word Recognition Rate (%)		Recognition Time (sec/word)	
	s^*	Lv^*	s^*	Lv^*
10	98.48	98.76	0.030	0.039
100	94.95	95.46	0.258	0.343
1k	88.42	89.00	2.237	2.942
10k	77.60	78.21	17.17	21.71

5.3.5 Class Dependent Constraints (CDCLBA)

In this section we summarize the improvement in recognition accuracy and in recognition time obtained by applying character class dependent constraints to the LDLBA.

Table XXXVII shows the recognition rate and recognition time achieved by the CD-CLBA.

Table XXXVII

Word recognition rate and recognition time using character class dependent constraints incorporated to the level building algorithm

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	Speedup (\times BLN)
	TOP 1	TOP 5	TOP 10		
10	98.74	99.90	100	0.026	8.54
100	95.42	98.80	99.37	0.227	8.76
1k	88.98	95.47	96.79	1.885	10.34
10k	78.22	88.49	90.99	15.27	11.95
30k	71.03	84.02	87.06	38.90	12.68

As we can see from Table XXXVII, we succeeded in recovering some accuracy that was lost with the CLBA. Table XXXVIII shows the reduction in recognition rates relatively to the baseline recognition system. Now, the average loss in accuracy was reduced to 1.084%, 0.582%, and 0.506% for the *TOP 1*, *TOP 5*, and *TOP 10* best choices respectively when compared with the performance of the baseline recognition system. At the same time, recognition time was also slightly improved, and now we have speedup factors between 8.5 and 12.7. The performance is slightly better due to the adaptation of the constraints to the particularities of the character classes.

5.3.6 Fast Two-Level HMM Decoding (FS)

This section reports the improvements on the performance of the baseline recognition system by breaking up the computation of word likelihoods into two steps: character HMM decoding and word HMM decoding. Tables XXXIX summarizes the word recognition rates obtained by using the fast two-level HMM decoding presented in Section 4.1.7. The bottom line is that there is no difference in accuracy

Table XXXVIII

Difference in the word recognition rates between the system based on the class dependent constrained level building algorithm and the baseline recognition system (BLN)

Lexicon Size	Word Recognition Rate (%) (BLN-CDCLBA)		
	TOP 1	TOP 5	TOP 10
10	0.19	0.03	0.00
100	0.46	0.19	0.03
1k	0.81	0.50	0.51
10k	1.28	1.04	0.90
30k	2.67	1.15	1.09
Average	1.083	0.581	0.505

between the FS and the baseline recognition system. However, these results were already expected because the fast-two level decoder maintains the optimality of the conventional Viterbi algorithm implemented in the baseline recognition system. Note that we have used the same global recognition model shown in Figure 24 for the baseline recognition system but with the lexical tree as shown in Figure 30b with one lowercase and one uppercase model for each character class.

Table XXXIX also shows the recognition times for the 5 dynamically generated lexicons. We note that our implementation has not been optimized or tuned up. Hence, the costs shown in the table are outstanding. Speedup factor between 11 and 17. Moreover, even for small and medium-size lexicons, the fast search strategy is advantageous.

The results shown in Table XL are even better. There is a significant improvement in recognition time relative to the baseline recognition system while preserving exactly the same word recognition rates. Speedup factors between 5 and 26 were obtained.

To ensure that the corresponding likelihood scores found by the optimal fast search agree exactly with those found by the standard Viterbi algorithm implemented in the baseline recognition system in all cases, we have compared the likelihoods of all words in the test set. So, the optimality of the search strategy was confirmed.

Table XXXIX

Word recognition rate and recognition time for the system based on the fast two-level HMM decoding and a flat lexicon (FSFlat) 36k1 task

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	Speedup (\times BLN)
	TOP 1	TOP 5	TOP 10		
10	98.93	99.93	100	0.020	11.10
100	95.89	98.99	99.40	0.120	16.57
1k	89.79	95.97	97.30	1.130	17.26
10k	79.50	89.53	91.89	11.15	16.36
30k	73.70	85.17	88.15	33.50	14.72

Table XL

Word recognition rate and recognition time for the system based on the fast two-level HMM decoding and a lexical tree (FSTree) 36k1 task

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	Speedup (\times BLN)
	TOP 1	TOP 5	TOP 10		
10	98.93	99.93	100	0.041	5.41
100	95.89	98.99	99.40	0.122	16.30
1k	89.79	95.97	97.30	0.841	23.19
10k	79.50	89.53	91.89	6.987	26.12
30k	73.70	85.17	88.15	20.17	24.45

5.3.7 Experiments with a Very-Large Vocabulary

The results reported in the previous section motivated us to investigate the performance of the fast two-level HMM decoding with even larger vocabularies. To such an aim, we have built a very-large vocabulary by adding to the 36.1k-entry lexicon of French city names more words corresponding to US city names (29.100 words), Italian city names (13.800 words), Brazilian city names (5.300 words), and Quebec city names (1.700 words). After eliminating replicated words, we ended up with a vocabulary of 85.092 city names (85.1k) where the words have an average length of 11.20 characters.

Table XLI shows the word recognition rates and processing times resulting from the fast two-level on 5 different dynamically generated lexicons with size 10, 1.000, 10.000, 40.000 and 80.000. Note that these dynamically generated lexicons are quite different from the lexicons used in the evaluations presented so far since they were generated from a 85.1k-word vocabulary. This also explains the difference in the word recognition rates relative to the previous evaluations. On the other hand the other testing conditions remain the same.

Table XLI

Word recognition rate and recognition time for the system based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task on a Sun Ultra1

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	
	TOP 1	TOP 5	TOP 10	FSTree	FSFlat
10	98.84	99.96	100	0.030	0.027
1k	91.01	96.32	97.71	0.990	1.129
10k	81.06	90.58	92.36	8.510	10.27
40k	73.23	84.64	87.91	32.30	42.45
80k	68.65	81.32	85.10	63.24	85.59

To have an idea on how the fast two-level HMM decoding improves over the baseline SRTP recognition system it would be necessary to evaluate its performance on the same test conditions. However, for such a vocabulary size it would be impractical to run the conventional Viterbi search over the entire test set of 4.674 words, since such an experiment could easily take more than one month to be completed. So, a palliative solution to evaluate the performance of the baseline recognition system with very-large vocabularies was to run a reduced experiment over only a subset of the test database composed by 1,000 words randomly selected. The performance of the fast two-level search was also re-evaluated using the same limited dataset. Figure 63 shows that the performance of the baseline recognition system is completely flawed on very-large vocabulary tasks since it needs more than 16 minutes to recognize a single word. On the other hand, the performance of the FS strategy is much better, however, it is still impractical since it needs approximately 60 seconds to recognize a single word.

In practice, the computation complexity of the fast two-level search strategy could be further reduce by using some programming techniques and consuming a little bit more of memory. However, since our criteria to compare the improvements is based on the recognition time of the computational implementation of the algorithm, to ensure a fair comparison none code optimization was used.

5.3.8 Distributed Recognition Scheme (DS)

We have implemented a distributed version of the fast two-level HMM decoding (FSTree like in Section 5.3.6) using the multithreaded programming interface for the Solaris 5.7 system to verify the effectiveness of the distributed recognition scheme in improving the recognition time. Both the non-distributed and the distributed tasks were run on a SUN Enterprise 6000 under the same test conditions drawn in Section 5.3.1 except that the test were carried out only for the 30k-entry dynami-

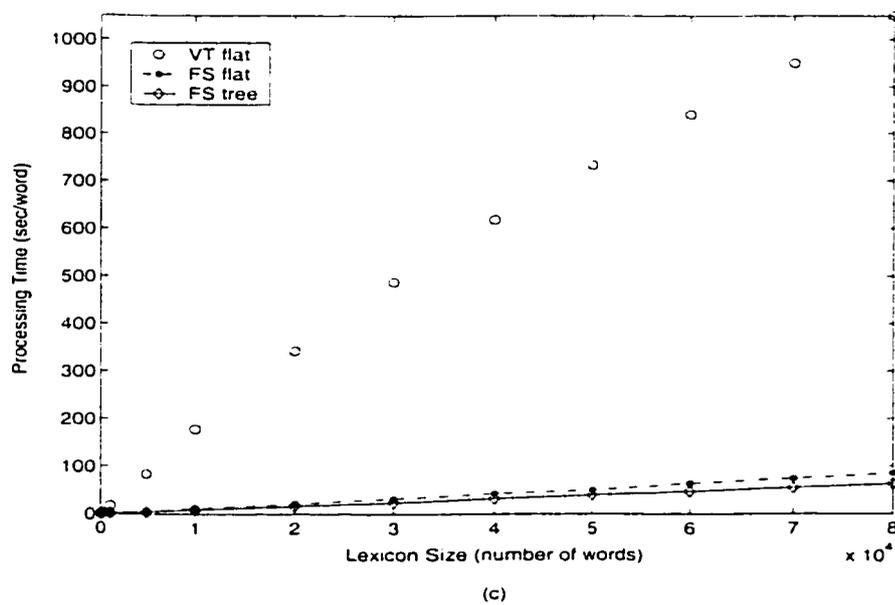


Figure 63 Comparison of average processing times achieved by the baseline SRTP recognition system implemented with a conventional Viterbi search (VT flat) and the implementation based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task (85.1k) over a limited test set of 1,000 words

cally generated lexicon since it does not seem very useful to apply the distributed recognition on small and medium vocabulary tasks.

Figure 6-4 shows the results obtained by the conventional recognition scheme (non-distributed) ($N_P = 1$) and other configurations that use from 2 to 10 processors. The 30k-entry dynamically generated lexicon was split into equal parts according to the number of processors used in the test. The tests were repeated 10 times to avoid distortions due to swapping or other processes running at the same time. The presented recognition times reflect the CPU time reported by the *C* routine *times*.

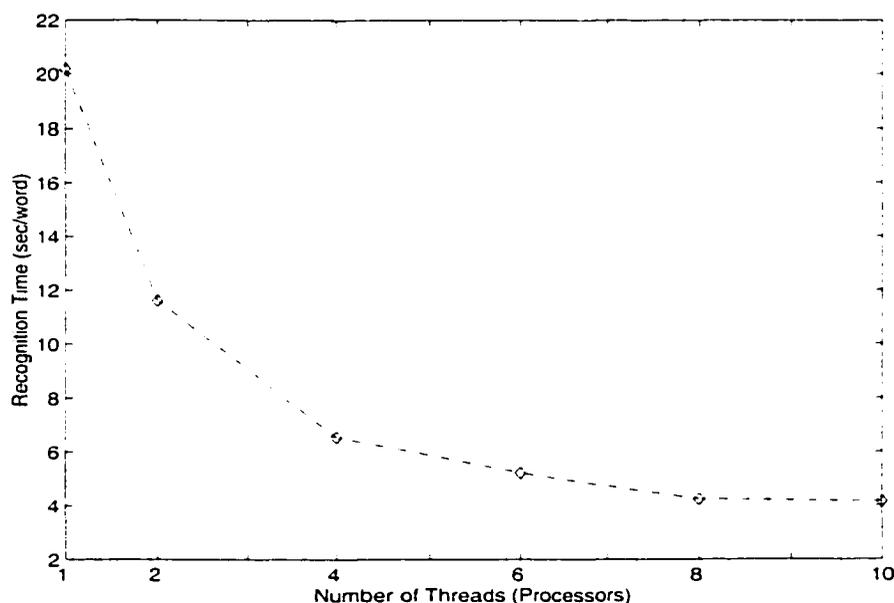


Figure 6-4 Average processing time for the distributed recognition scheme according to the number of threads (processors) for a 30k-entry dynamically generated lexicon

As we can see from Figure 6-4 there is a significant reduction of over 4 times in processing time relative to the conventional recognition scheme (non-distributed). This improvement is unquestionably significant since the recognition accuracy is preserved. Besides the speedup achieved, it is also instructive to evaluate the efficiency of the distributed recognition scheme relative to the conventional recognition

Table XLII

Figure of performance for the distributed recognition scheme (DS-FSTree) for a different number of processors

Number of Threads (N_P)	Lexicon Size (w/ N_P)	Recognition Time (sec/word)	Speedup (S_p)	Efficiency (E_f)
1	30,000	20.17	—	—
2	15,000	11.61	1.79	0.89
4	7,500	6.54	3.08	0.77
6	5,000	5.21	3.87	0.64
8	3,750	4.24	4.75	0.59
10	3,000	4.15	4.85	0.48

schemes [93]. The efficiency, denoted as E_F , represents the effective utilization of computing resources by the distributed recognition scheme and it is the ratio of the speedup achieved, denoted as S_p to the number of processing elements used, denoted as N_P , with respect to a single processing element. Table XLII shows the processing time, speedup and efficiency obtained by the different number of processors.

The relative reduction in processing time, compared to the conventional recognition scheme based on the fast two-level search strategy, is unquestionably significant, even if the speedup of distributed scheme on N_P processors is less than N_P . Only an ideal distributed system can deliver a speedup equal to N_P . In practice, the processor cannot devote 100% of its time to the computation of the algorithm. Furthermore, the distributed scheme incurs overhead from several sources such as communication overhead, idle time due to load imbalance, and contention for shared data structures. For instance, we have found a difference of 7 to 12% in the processing time due to the load imbalance, that is, the sub-lexicons have the same number of words, but they do not have the same number of characters.

Distributed scheme involves the classical communication versus computation trade-off, thus, the search overhead is greater than one, implying that the distributed scheme does more work than the sequential scheme. As we saw in Figure 64, the speedup tends to saturate. In other words, the efficiency drops with an increasing number of processors. This phenomenon is true for all distributed systems, and is often referred to as *Amdahl's law* [93]. The efficiency is gradually reduced since the communication overhead (C_o) defined as $C_o = t_{comm} \cdot N_p$, where t_{comm} denotes the communication time between the processors, is also a multiple of the number of processors.

5.3.9 Summary of Speed Improvements

The contents of this section can be summarized by comparing the performances of various approaches along two axes: word recognition rate *versus* recognition time¹. Figures 65, 66, and 67 captures this information succinctly for a small, medium and large vocabulary respectively.

The figures show that we can attain the recognition rate of the baseline SRTP recognition system while speeding up the system by factors between 8 and 119 for a large vocabulary (30,000 words) and by factors between 6 and 23 for a medium vocabulary (1,000 words). However, it should be noticed that for small lexicons (< 100 words) the performance of the fast-two level search with a flat lexicon is slightly better than with a lexical-tree. This occurs because there are very few shared prefixes in the words in a small lexicon relatively to a large lexicon.

Figure 68 compares the word recognition rates considering the *TOP* 1 choice and recognition times achieved by the search strategies for a 30,000-word vocabulary task. As we can see, the fast-two level decoding algorithm has a superior performance relative to the baseline recognition system.

¹The recognition time was evaluated on a SUN Ultra 1.

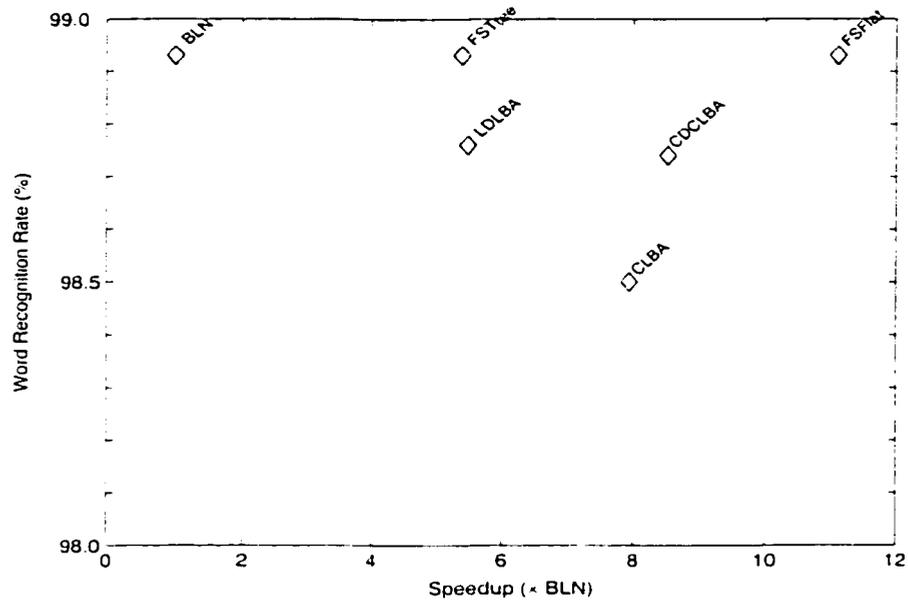


Figure 65 Word recognition rate *versus* speedup of various systems for a 10-word vocabulary

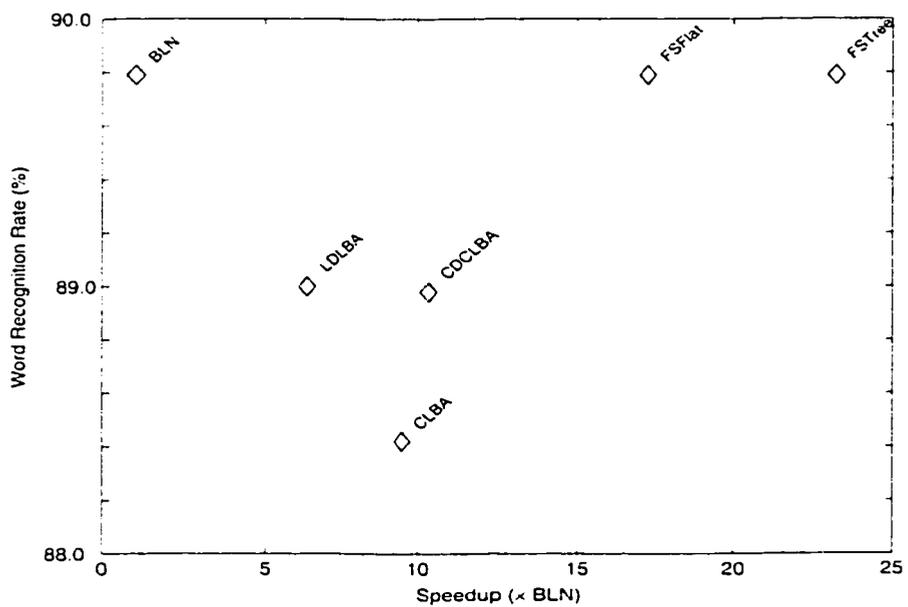


Figure 66 Word recognition rate *versus* speedup of various systems for a 1k-word vocabulary

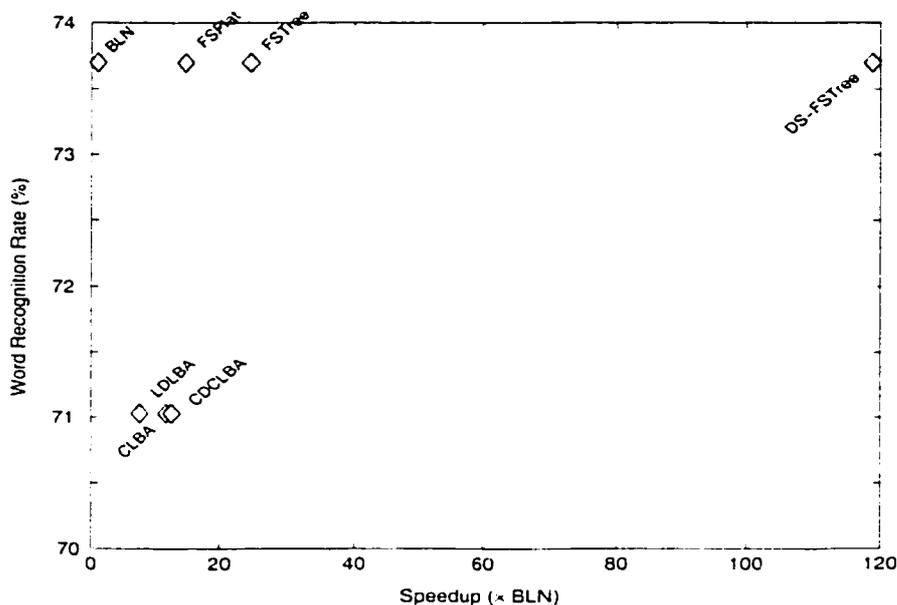


Figure 67 Word recognition rate *versus* speedup of various systems for a 30k-word vocabulary

Note that the distributed recognition scheme was implemented on a different platform (SUN 6000), while the other search techniques were implemented on a SUN Ultra1. However, both machines have exactly the same processor (UltraSparc1 167MHz), so their performances can be approximately compared as is the case in Figures 67 and 68. In fact, we have evaluated the performance of the other search techniques on the SUN 6000 and the results in terms of recognition time did not vary significantly from those obtained on the SUN Ultra1 and reported along this section.

5.4 Improving Recognition Accuracy

In this section we present the experiments undertaken during the development of the strategies to improve the recognition accuracy presented in Chapter 4. First we describe the testing conditions and the datasets used to evaluate the recognition strategies. We introduce two other classifiers, one based on k -nearest prototypes

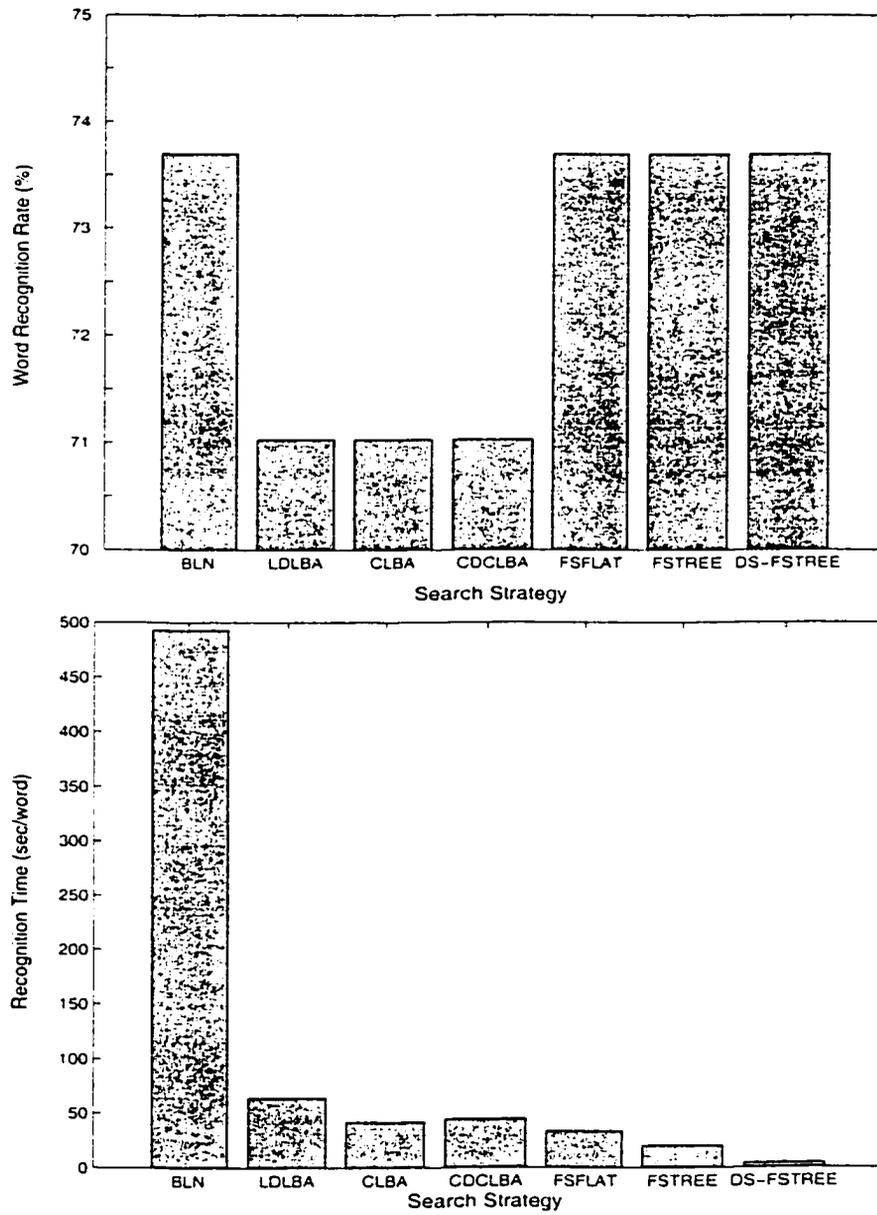


Figure 68 Comparison of the search strategies for a 30k-word vocabulary task: (a) recognition accuracy for the *TOP* 1 choice, (b) recognition time

(k -NP) and other based on k -nearest neighbors (k -NN). The performance of such classifiers to recognize handwritten isolated characters is presented in Sections 5.4.3.2 and 5.4.3.1. The performance of the neural network classifier (NN) presented in Section 4.2.5 is presented in Section 5.4.3.3. In Section 5.4.3.4 the performance of these three classifiers to recognize isolated handwritten characters from both NIST and SRTP databases is compared. In Section 5.4.3.3 we evaluate the use of the NN classifier in the recognition of handwritten words to further integrate it with the baseline recognition system in an attempt to improve recognition accuracy.

5.4.1 Testing Conditions

In this section we have three different types of experiments: recognition of isolated handwritten characters, recognition of handwritten words and combination of different classifiers to optimize the recognition accuracy in handwritten words. For each type of experiment, the testing conditions vary slightly.

The experiments related to the recognition of isolated handwritten characters described along Section 5.4.3 were conducted as follows:

- From the training dataset of the NIST database were taken 1,660 and 1,440 samples per character class for uppercase and lowercase characters respectively. For each sample a feature vector composed by the three feature types described in Section 4.2.4 was generated. These feature vectors corresponding to 80,600 isolated characters were used to train the classifiers;
- From the validation and test datasets of the NIST database, the feature vectors were generated by the same manner. The validation feature vectors correspond to 23,670 characters (uppercase + lowercase) and the test feature vectors correspond to 23,941 characters (uppercase + lowercase);
- In all experiments with isolated character recognition we have considered 26 *metaclasses* of characters which are formed by the union of the uppercase and

lowercase representations of characters (e.g. "A" + "a" = metaclass "A"):

- All experiment were conducted on the same machine, a PC AMD Athlon 1.1GHz with 512MB of RAM memory, except when mentioned otherwise in context.

However, for the SRTP database, the procedure is slightly different since we have more limited datasets where the main problem is the unbalanced distribution of samples between the character classes. To build the feature vectors we have relied on the frequency balancing method presented in Section 4.2.5.2.

- From the cleaned training dataset of the SRTP database (Table XXIX) were taken 1.630 and 1.630 samples per character class for uppercase and lowercase characters. The number of samples per character class was determined by Equation 4.36. For those character classes with few samples, synthetic samples were generated by stroke warping technique presented in Section 4.2.5.3;
- For each character sample a feature vector composed by the three feature types described in Section 4.2.4 was generated. These feature vectors corresponding to 84.811 isolated characters were used to train the classifiers;
- From the validation and test datasets of the SRTP database, the feature vectors were generated in a similar manner, however, no frequency balancing was carried out. The validation feature vectors correspond to 36,170 characters (uppercase + lowercase) and the test feature vectors correspond to 46,700 characters (uppercase + lowercase).

For the recognition of handwritten words, only the SRTP database was used. The datasets for training and validation for isolated character recognition are exactly the same described above. However, here we have many additional validation and test sets that are generated from the N -best word hypothesis lists that are resulting from

the process of word recognition. Due to practical limitations, we have restrained the experiments in this section to 5 different dynamic lexicon sizes with 10, 1.000, 10.000, 40.000 and 80.000 words.

The testing conditions for the verification of handwritten words are given as follows:

- For each dynamic lexicon, an N -best word hypothesis list is generated by running a complete word recognition experiment as described in Section 5.3:
- The lists were generated for the words in the validation and test datasets:
- For each word in the validation and test dataset, a list with the *TOP* 10 best word hypotheses containing the ASCII transcriptions was produced together with the segmentation hypothesis of this word into characters and the probability score of each hypothesis:
- Having the segmentation of each word hypothesis we return to the word image and extract the features as described in Section 4.2.4. These feature vectors corresponding to segmented characters are used during the recognition of the isolated characters.

5.4.2 Reevaluation of the Word Recognition Performance

During the development of this thesis, more powerful hardware platforms became available. So, part of the algorithms developed so far were migrated to this new platform. To illustrate the performance of the fast two-level HMM decoding on this new platform, the experiments with the very-large vocabulary presented in Section 5.3.7 were carried out on a PC Athlon 1.1GHz with 512MB of RAM memory. Table XLIII shows that the recognition process runs faster on such a machine.

So, henceforth, the algorithms and methods will be developed and the performance evaluated on this new hardware platform, as well as the results in terms of recognition time and accuracy will be compared with the results presented in Table XLIII.

Table XLIII

Word recognition rate and recognition time for the system based on the fast two-level HMM decoding (FSFlat and FSTree) for a very-large vocabulary task on an Athlon 1.1GHz

Lexicon Size	Word Recognition Rate (%)			Recognition Time (sec/word)	
	TOP 1	TOP 5	TOP 10	FSTree	FSFlat
10	98.84	99.96	100	0.010	0.009
1k	91.01	96.32	97.71	0.273	0.321
10k	81.06	90.58	92.36	1.992	2.576
40k	73.23	84.64	87.91	7.516	9.741
80k	68.65	81.32	85.10	14.46	19.52

5.4.3 Recognition of Isolated Handwritten Characters

In this section we analyze the performance of the neural network classifier proposed in Section 4.2.5 to recognize isolated handwritten characters from two different databases: NIST and SRTP. The NIST database is a standard database that has been widely used in handwriting recognition. For this reason, by using this database it is easy to evaluate the performance of the proposed neural network classifier and compare it with others. On the other hand, the isolated characters generated from the SRTP database are particular for our work, so we do not have any reference to assess and compare the performance of the proposed neural classifier. The manner we have found to circumvent this problem was to implement two other classifiers, one based on k -NN and other based on k -NP.

In Section 5.4.3.1 we present briefly the characteristics of the k -nearest neighbor (k -NN) as well as its performance in terms of recognition accuracy and recognition time. In Section 5.4.3.2 we present briefly the characteristics of the k -nearest prototype (k -NP) classifier and its performance in terms of recognition accuracy and recognition

time is evaluated as well. Finally, in Section 5.4.3.3 we evaluate the performance of the neural network classifier.

5.4.3.1 k Nearest Neighbor Classifier (k -NN)

One of the simplest classifier to implement is the k -nearest neighbor (k -NN). The idea underlying this classifier is to compare each pattern of unknown class with all reference patterns (samples in the training dataset) at the feature space and using the Euclidian distance as measurement. The character class of the closest neighbor is assigned to the unknown pattern. For all the experiments we have considered 26 metaclasses of characters.

Table XLIV shows the recognition rates on the validation and test datasets of NIST database for $k = 1, 3, 5, 7$, and 9. Table XLV shows the recognition time of the k -NN for the same values of k . Table XLVI shows the recognition rates on the validation and test datasets of SRTP database for $k = 1, 3$, and 5. The recognition time of the k -NN for the same values of k is shown in Table XLVII.

Table XLIV

Results for isolated character recognition on the NIST database using different number of neighbors (k) for a k -NN classifier

Dataset	Character Recognition Rate (%)				
	1-NN	3-NN	5-NN	7-NN	9-NN
Validation	86.91	87.48	87.69	87.61	87.70
Test	84.21	84.91	85.28	85.19	85.05

The character recognition rates of the k -NN has a tendency to increase with an increasing number of neighbors. The accuracy of the classifier is in line with the results obtained by other researchers presented in Table VI that do not make dis-

Table XLV

Recognition time for isolated character recognition on the NIST database using different number of neighbors (k) for a k -NN classifier

Classifier	Recognition Time (sec./character)
1-NN	0.28
3-NN	0.29
5-NN	0.30
7-NN	0.31
9-NN	0.31

inction between uppercase and lowercase characters. The difference between the recognition rates achieved on the validation and test datasets is explained by the fact that many samples in the training and validation datasets are from the same person, while the samples in the test dataset were written by different people. So, the test dataset is a more difficult dataset. The recognition time depicted in Table XLV highlights one of the main problems of such a kind of classifier. Although there are several techniques to speedup k -NN classifiers [116].

Table XLVI

Results for isolated character recognition on the SRTP database using different number of neighbors (k) for a k -NN classifier

Dataset	Character Recognition Rate (%)		
	1-NN	3-NN	5-NN
Validation	64.01	68.04	70.22
Test	64.19	67.86	70.25

The character recognition rates reported in Table XLVI are significantly inferior to

Table XLVII

Recognition time for isolated character recognition on the SRTP database using different number of neighbors (k) for a k -NN classifier

Classifier	Recognition Time (sec/character)
1-NN	0.29
3-NN	0.30
5-NN	0.31

those obtained on the NIST database. Different from the NIST database where the characters samples were written isolated on a favorable environment, the isolated characters on the SRTP datasets were automatically generated from handwritten words using the baseline recognition system to segment and label each sample. So, the SRTP datasets may contain many corrupted samples that actually are not whole characters but fragments, joined fragments from different characters or even joined characters.

We have compared the confusion matrices of both NIST and SRTP validation datasets and many differences come up. For instance, the number of confusions between the characters "T" and "I" is significantly higher for the SRTP dataset. This is due to pre-processing and segmentation steps of the baseline recognition system that eliminates the horizontal bar of the "T"'s when the horizontal and vertical bars are disconnected.

5.4.3.2 k -Nearest Prototype Classifier (k -NP)

Considering the slowness of the k -NN classifier, to speed up the recognition process, instead of comparing each pattern of unknown class with all samples in the training dataset, the classification can be carried out by comparing the pattern with class

prototypes generated from the training dataset. This classification method is known as k -Nearest Prototypes (k -NP) and it consists in generating class prototypes. A straightforward manner of create class prototypes is to average the samples in the training dataset that correspond to a given character class. A conventional algorithm that carries out such an average process is the k -means algorithm.

The k -means algorithm can be described in 4 steps as:

1. For each class, k prototypes are created;
2. The samples in the training dataset are classified in the k sub-classes according to the Euclidian distance between them and the class prototypes;
3. Each class prototype is replaced by the mean of samples in the associated sub-class;
4. Repeat from Step 2 until the results become stable;
5. End.

Since uppercase and lowercase characters are merged to form metaclasses of characters there are several manners to create metaclass prototypes, but for simplicity we consider only the following:

- np class prototypes for each of 26 metaclasses, without making distinction between uppercase and lowercase characters to generate the metaclass prototypes (denoted as $1M$);
- np class prototypes for each of 26 metaclasses, making distinction between uppercase and lowercase characters to generate the metaclass prototypes, where $np/2$ class prototypes are generated from uppercase characters and $np/2$ class prototypes are generated from lowercase characters (denoted as $2C$);

Figure 69 shows the recognition rates achieved by taking different number of neighbors (k) in the decision rule and different configurations of the class prototypes on the NIST validation dataset. The first remark is that an increasing number of

neighbors in the decision rule did not have improved the recognition rates, but instead, the accuracy was reduced. So, the further studies with the k -NP classifier are pursued based on $k = 1$.

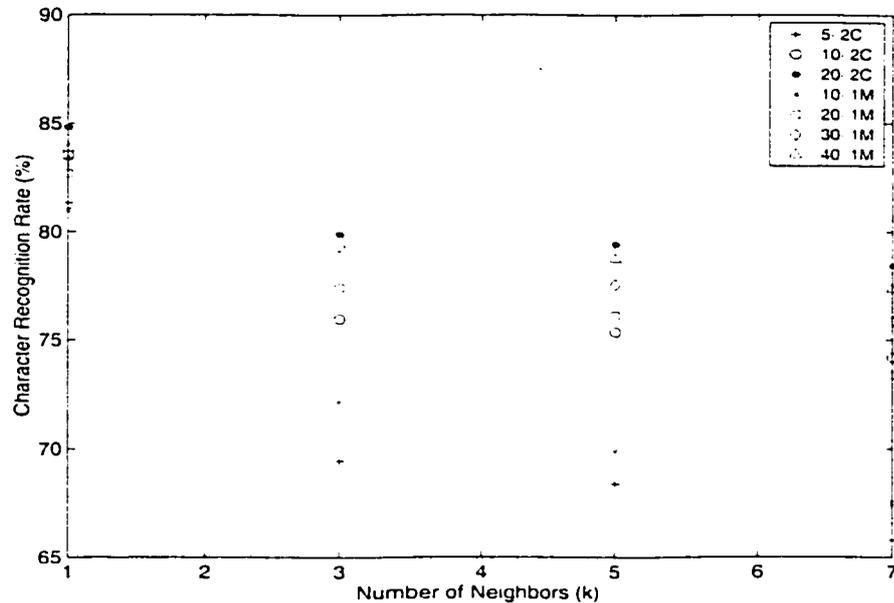


Figure 69 Character recognition rates of the k -NP on the NIST validation dataset for different configurations of prototypes and number of neighbors

Next, we have carried out some experiments to find out which is the best way to group the samples to compute the class prototypes, that is, for a given metaclass, create a equal number of prototypes from the corresponding uppercase and lowercase character samples ($npU = npL = np/2$) or leave the k -means algorithm to decide it ($npU + npL = np$). Figure 70 shows the recognition rates that have been obtained by different number of prototypes per character class. It seems that it is better to create a large and equal number of prototypes for both uppercase and lowercase character samples than to create metaclass prototypes from different number of uppercase and lowercase character samples.

Table XLVIII shows the character recognition figures on the training, validation and

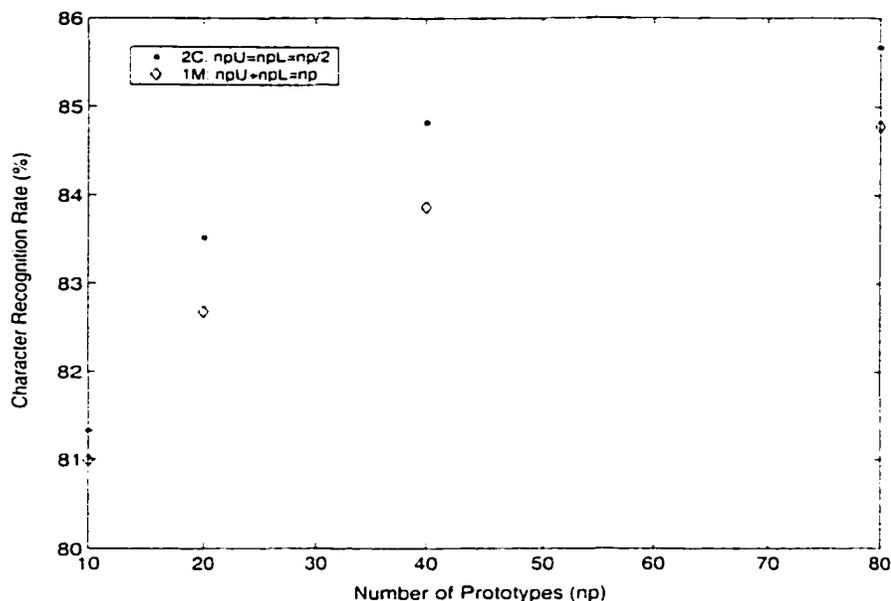


Figure 70 Number and configuration of class prototypes for the 1-NP classifier on the NIST validation dataset

test sets of the NIST database resulting from 3 different configurations for the class prototypes. As we can see, there is a significant improvement of over 6% in accuracy on the training dataset by increasing the number of prototypes. On the other hand, the improvements are not so significant on the validation and test datasets ($\approx 1\%$). However, the results have to be analyzed taking into account also the recognition times shown in Table XLIX. The recognition time increase approximately linearly with the number of prototypes. In practical terms, the recognition times are too high to be used in the proposed verification scheme since the time required to recognize a 10-character word will be more than 200ms.

The same experiments were repeated to the SRTP database. Table L shows the character recognition figures on the training, validation and test sets of the SRTP database resulting from 3 different configurations for the class prototypes and the recognition times are shown in Table LI. As expected, there is a significant decrease in recognition rates compared to the NIST database. We attribute the

Table XLVIII

Character recognition rates on the NIST database using different prototype configurations for a 1-NP classifier

Dataset	Character Recognition Rate (%)		
	1-NP (100 · 2C)	1-NP (500 · 2C)	1-NP (800 · 2C)
Training	93.62	98.52	99.49
Validation	86.13	87.05	87.00
Test	82.95	83.69	84.06

Table XLIX

Character recognition times on the NIST database for different prototype configurations for a 1-NP classifier

Classifier	Recognition Time (sec/character)
1-NN (100 · 2C)	22m
1-NN (500 · 2C)	81m
1-NN (800 · 2C)	129m

decrease in character recognition rate to poorer quality of the samples presented in the SRTP database that were produced by automatic segmentation and labeling. Furthermore, the characters samples in the NIST database were written in isolation and not within words.

Table L

Character recognition rate on the SRTP database using different number of neighbors (k) for a k -NP classifier

Dataset	Character Recognition Rate (%)		
	1-NP (100 · 2C)	1-NP (500 · 2C)	1-NP (800 · 2C)
Training	70.57	85.64	92.69
Validation	62.59	63.49	63.81
Test	62.40	63.48	63.61

Table LI

Character recognition time on the SRTP database using different number of neighbors (k) for a k -NP classifier

Classifier	Recognition Time (sec/character)
k-NN (100 · 2C)	28m
k-NN (500 · 2C)	89m
k-NN (800 · 2C)	135m

5.4.3.3 Neural Network Classifier (NN)

The neural network was implemented according to the architecture described in Section 4.2.5. The MLP classifier was trained with 80,600 characters from NIST

database (3,100 samples per class) using the backpropagation algorithm. A validation set with 23,670 characters was also used during the training to watch over the generalization and stop the training at the minimum of the error. Table LII shows the character recognition rates and character recognition time achieved on the three datasets of NIST database.

Table LII

Character recognition rates and character recognition time on the NIST database for the MLP classifier

Dataset	Character Recognition Rate (%)	Recognition Time (sec/character)
Training	94.71	210 μ
Validation	89.98	210 μ
Test	88.10	210 μ

The same experiments were repeated on the SRTP database. Table LIII shows the character recognition figures on the training, validation and test sets of the SRTP. The MLP classifier was trained with a balanced feature vector generated from 84,760 characters from SRTP database (3,260 samples per character class) using the backpropagation algorithm. For some character classes with few samples (e.g. "K", "Q", "W", etc.), the stroke warping technique presented in Section 4.2.5.3 was used to generate new samples. The class probabilities at the output of the MLP were corrected to compensate the changes in the *a priori* class probabilities introduced by the frequency balancing.

5.4.3.4 Comparison of the Classifier Results

Figures 71 and 72 compare the character recognition rate and character recognition time achieved on the NIST and SRTP databases. For both databases, it is evident

Table LIII

Character recognition rate and character recognition time on the SRTP database for the MLP classifier

Dataset	Character Recognition Rate (%)	Recognition Time (sec/character)
Training	76.48	210 μ
Validation	73.54	210 μ
Test	73.51	210 μ

the supremacy of the MLP classifier over the k -NN and k -NP classifiers both in terms of accuracy and speed. The neural network based on MLP has a generalization power that is superior to the k -NN and the k -NP. Moreover, once trained, the NN classifier carries out the recognition task much faster than other classifiers.

These results endorse the choice of the MLP classifier as the core of the verification module which relies on isolated character recognition to post-process the N -best word hypotheses provided by the the baseline recognition system. In the next section, the MLP classifier is used in the recognition of handwritten words.

5.4.4 Recognition of Handwritten Words by NN Classifier

In this section we analyze the performance of the MLP classifier to recognize handwritten words from the SRTP database. In fact, the same MLP classifier evaluated in the preceding section is used for such an aim, where the probability scores obtained for each isolated character are combined to made up the word probability score as described in Section 4.2.6. But now, the datasets used to analyze the performance are slightly different, since they are generated from the N -best word hypothesis list as described in Section 5.4.1.

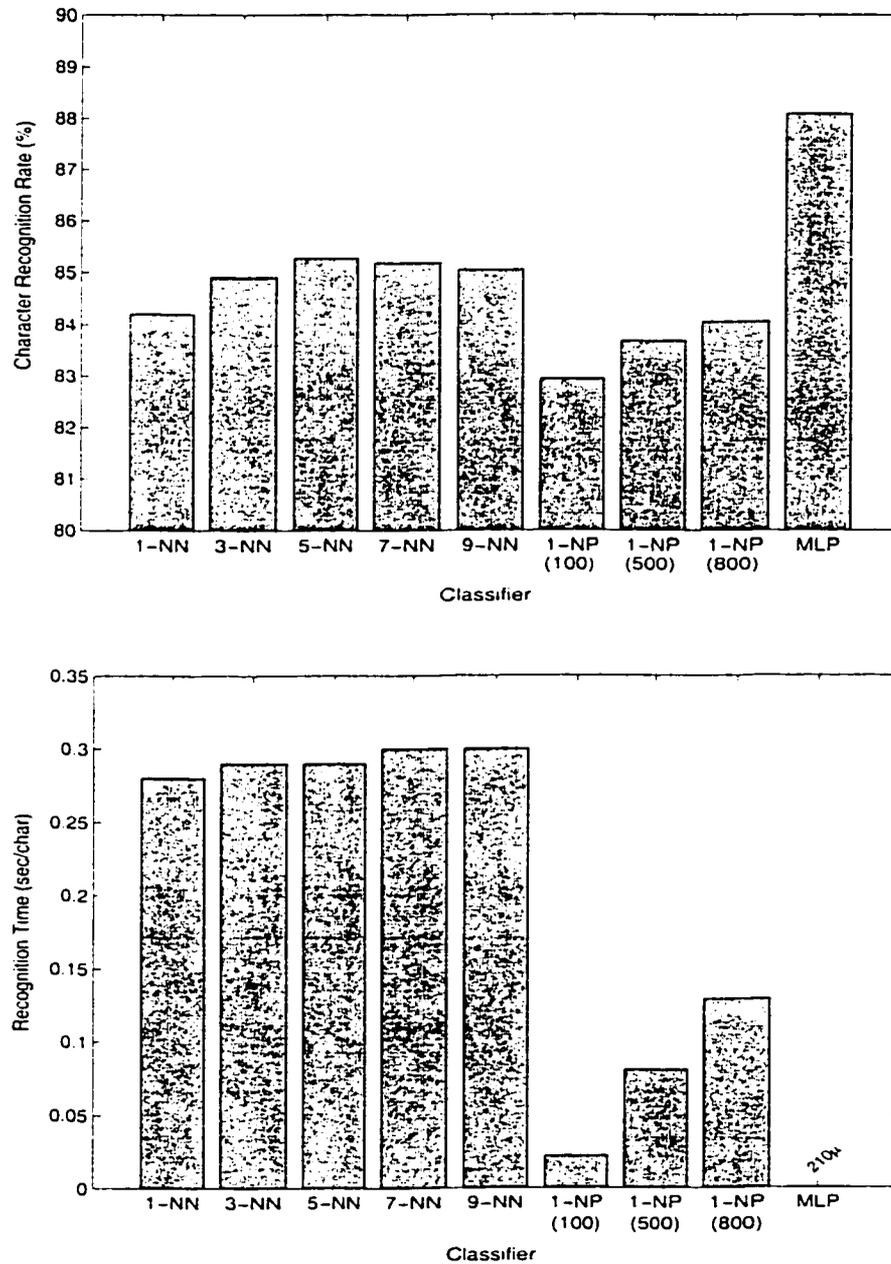


Figure 71 Comparison of character recognition rate and character recognition time on the NIST test dataset using k -NN, k -NP and MLP classifiers

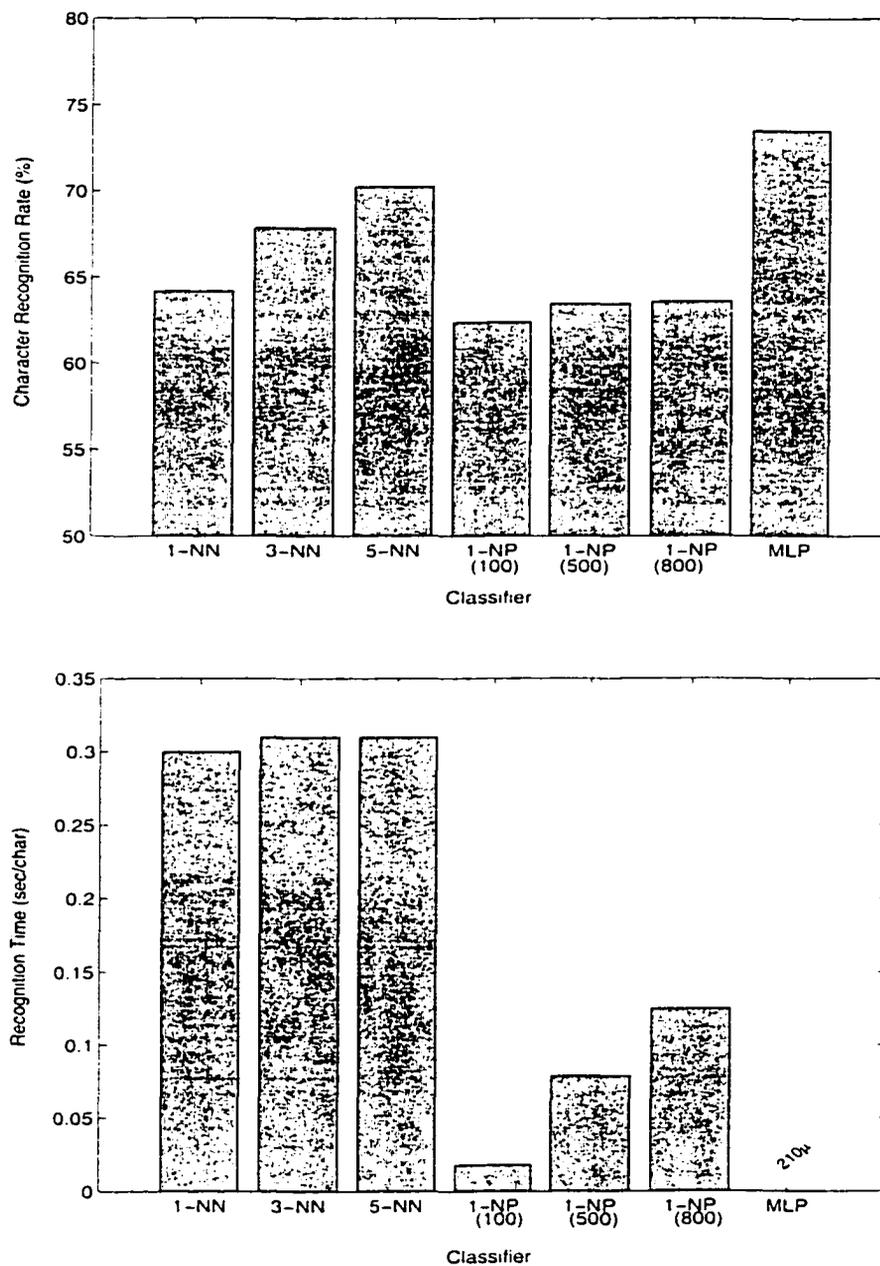


Figure 72 Comparison of character recognition rate and character recognition time on the SRTP test dataset using k -NN, k -NP and MLP classifiers

Table LIV

Word recognition rates using the NN classifier alone for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words

Lexicon Size	Word Recognition Rate (%)			
	TOP 1	TOP 2	TOP 5	TOP 10
10	89.90	94.30	98.07	100
1k	86.07	90.54	95.00	97.71
10k	80.82	85.96	90.27	92.36
40k	74.19	80.28	85.12	87.91
80k	71.41	77.73	82.49	85.10

Table LIV shows the word recognition rates resulting from the combination of the characters scored by MLP classifier for 5 different lexicons with 10, 1,000, 10,000, 40,000 and 80,000 words. Figure 73 provides a comparison with the baseline recognition system based on the FSTree search strategy.

The relative increase on recognition errors, compared to the baseline recognition system, is unquestionably significant for vocabularies with less than 40,000 words. On the other hand, for vocabularies with 40,000 and 80,000 words, the MLP classifier performs better.

We attribute the reduction in word recognition rate to the different words with very different lengths that are more likely to be present in small lexicons. The summation of log probabilities favors short words. On the other hand, for larger vocabularies, the word hypotheses in the *TOP* 10 list are more likely to have similar lengths. To revert this problem it is necessary to take into account the duration of each character when performing the summation of character log-probability scores. Table LV shows the word recognition rates resulting from the combination of the characters scored by

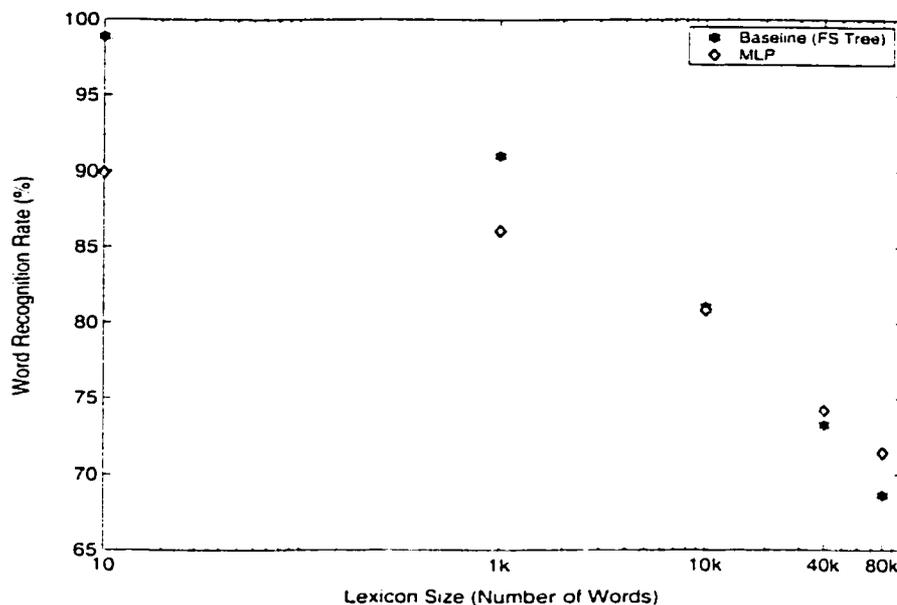


Figure 73 Word recognition rates (*TOP 1*) for the MLP classifier and for the baseline recognition system based on the FSTree search strategy for different lexicon sizes (log scale)

the MLP classifier with character duration modeling as presented in Section 4.2.6.2.

Table LV

Word recognition rates using the the MLP with character duration modeling for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words

Lexicon Size	Word Recognition Rate (%)			
	TOP 1	TOP 2	TOP 5	TOP 10
10	92.75	96.65	98.69	100
1k	87.78	91.57	95.14	97.71
10k	82.39	86.82	90.41	92.36
40k	75.95	81.38	85.43	87.91
80k	73.71	78.65	82.69	85.10

Figure 74 provides a comparison of the MLP with duration modeling with the con-

ventional MLP and with the baseline recognition system based on the FSTree search strategy. In spite of an improvement on the word recognition rates, there still is a significant difference between the baseline recognition system and the MLP with duration modeling. We attribute this inferior performance to the quality of the duration modeling used. Recalling Section 4.2.6.2, in fact we did not have used a “real” duration estimation, but we have adapted it from the behavior of the segmentation module of the baseline recognition system. Probably a “real” duration model based on the shape of characters and on “real” cases instead of metaclasses would improve better the word recognition rate.

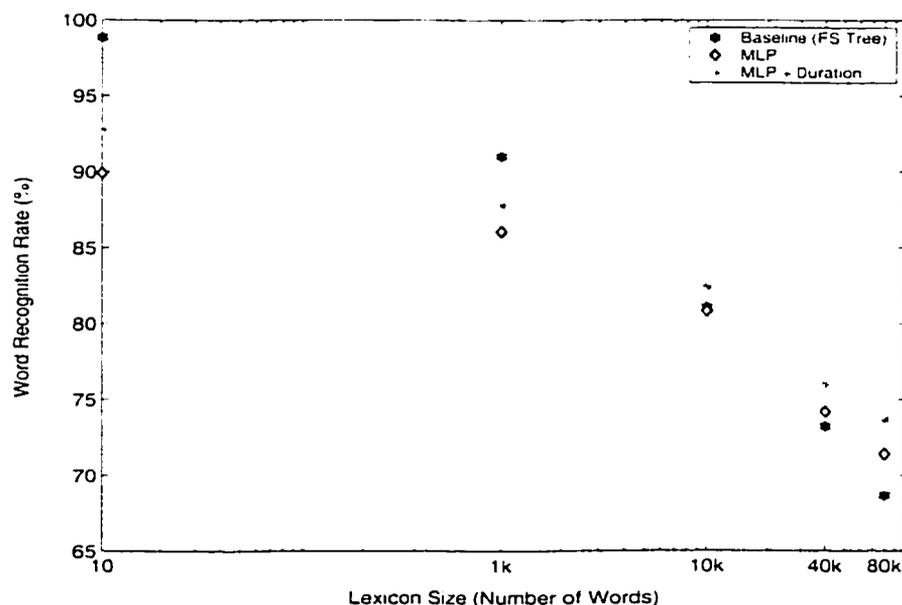


Figure 74 Word recognition rates (*TOP 1*) for the MLP classifier with and without duration model and for the baseline recognition system based on the FSTree search strategy for different lexicon sizes (log scale)

5.4.5 Verification of Unconstrained Handwritten Words

In this section we summarize the improvement in recognition accuracy obtained by combining the recognition of handwritten words by NN classifier, presented in Section 5.4.4 with the recognition of handwritten words by the baseline recognition

system based on the fast two-level HMM decoding (FSTree), presented in Section 5.3.6.

Table LVI shows the recognition rates resulting from using the baseline recognition system alone, the MLP classifier alone, and the combination of the both by a non-weighted ($\alpha = 0.5$ and $\beta = 0.5$) and a weighted rule ($\alpha = 0.85$ and $\beta = 0.15$). The choice of the values for the weight factors α and β was done empirically on the SRTP validation dataset as we have seen in Section 4.3.

Figure 75 illustrates the improvement in recognition accuracy obtained by the combination of the baseline recognition system with the verification module based on the MLP classifier. As we can see from Table LVI and Figure 75, there is a significant improvement of over 9% in accuracy relatively to the baseline recognition system alone for a 80,000-word vocabulary. The effects of the verification module are gradually reduced as the size of the vocabulary decreases, but it is still able to boost in 0.5% the accuracy for a 10-word vocabulary. So, the relative increase in word recognition rate, compared to the baseline recognition system alone, is unquestionably significant, especially for large vocabularies.

However, the use of the duration model did not bring any advantage when combining the scores with the baseline recognition system. As we have pointed out before, this is probably due to the quality of the duration model that we have used that does not really reflects the duration of characters, but how they are segmented.

5.4.6 Error Analysis

In spite of the improvement in recognition accuracy, there is still a difference of 7.05% between the *TOP* 1 and the *TOP* 10 best choices (for the 80,000-word lexicon). To understand better the role of the verification module in the recognition of unconstrained handwritten words, we have analyzed the situations where the

Table LVI

Word recognition rates from the baseline recognition system alone, the MLP classifier alone and the combination of both classifiers by two different rules for 5 sizes of lexicons: 10, 1k, 10k, 40k, and 80k words

Lexicon Size	Classifier	Words Recognition Rate (%)			
		TOP 1	TOP 2	TOP 5	TOP 10
10	Baseline FSTree	98.84	99.74	99.96	100.00
	MLP Classifier	89.90	94.30	98.07	—
	FSTree + MLP	96.66	98.01	99.50	—
	FSTree + MLP Weighted	99.40	99.78	99.98	—
	FSTree + MLP Weighted + Dur	99.25	99.66	99.89	—
1k	Baseline FSTree	91.01	94.20	96.32	97.71
	MLP Classifier	86.07	90.54	95.00	—
	FSTree + MLP	90.22	93.39	95.78	—
	FSTree + MLP Weighted	94.18	95.91	97.24	—
	FSTree + MLP Weighted + Dur	94.14	95.93	97.17	—
10k	Baseline FSTree	81.06	85.83	90.58	92.36
	MLP Classifier	80.82	85.96	90.27	—
	FSTree + MLP	84.47	88.01	90.86	—
	FSTree + MLP Weighted	87.76	90.09	91.98	—
	FSTree + MLP Weighted + Dur	87.82	89.96	91.84	—
40k	Baseline FSTree	73.23	79.48	84.64	87.91
	MLP Classifier	74.19	80.28	85.12	—
	FSTree + MLP	78.21	82.43	85.64	—
	FSTree + MLP Weighted	81.30	84.85	86.54	—
	FSTree + MLP Weighted + Dur	81.32	84.93	86.52	—
80k	Baseline FSTree	68.65	75.50	81.32	85.10
	MLP Classifier	71.41	77.73	82.49	—
	FSTree + MLP	74.81	79.57	83.03	—
	FSTree + MLP Weighted	78.05	82.03	84.04	—
	FSTree + MLP Weighted + Dur	77.90	82.07	83.99	—

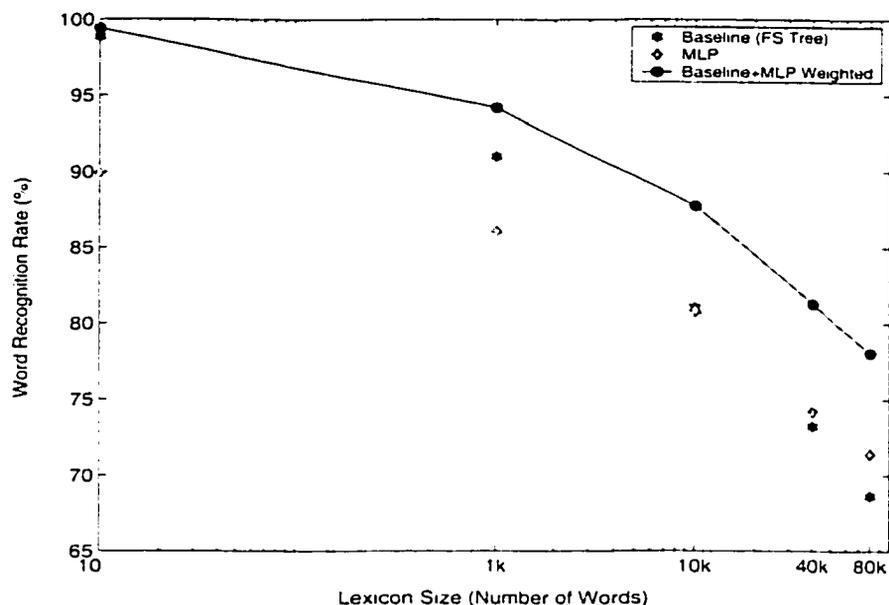


Figure 75 Word recognition rates (*TOP 1*) of the recognition-verification system compared with the baseline recognition system and MLP classifier (log scale)

verifier succeeds to re-score and re-rank the truth word hypothesis (shifting up to the top of the N -best list). The error analysis for the case of 80,000-word lexicon is presented as follows:

- 1,136 words out of 4,674 were re-ranked (24.31%), where 508 words were correctly re-ranked (10.88%), that is, the truth word hypothesis is shifted up to the *TOP 1* position (Figure 76) and for the 628 words (13.43%) the truth word hypothesis was not shift up to the *TOP 1* position (Figure 77);
- However, for 445 words out of 628 (9.52%), the truth word hypothesis was not among the *TOP 10* word hypotheses;
- For the 183 remaining words out of 628 (3.91%), 69 words were correctly recognized by the baseline recognition system alone (1.48%), but after re-scoring they were shifted down from the *TOP 1* position (Figure 78);

- For the 114 remaining words out of 183 (2.44%) the verification module was not able to shift up the truth word hypothesis to the *TOP 1* position (Figure 77), but at least, it did not mess up the results provided by the baseline recognition system alone.

In summary, the verification module was able to re-rank correctly 10.88% of the words hypothesis, shifting them up to the *TOP 1* position, but on the other hand it also re-rank wrongly 1.48% of the words hypothesis, shifting them down from the *TOP 1* position. This represent an overall improvement of 9.4% in the word recognition rate for an 80,000-word vocabulary.

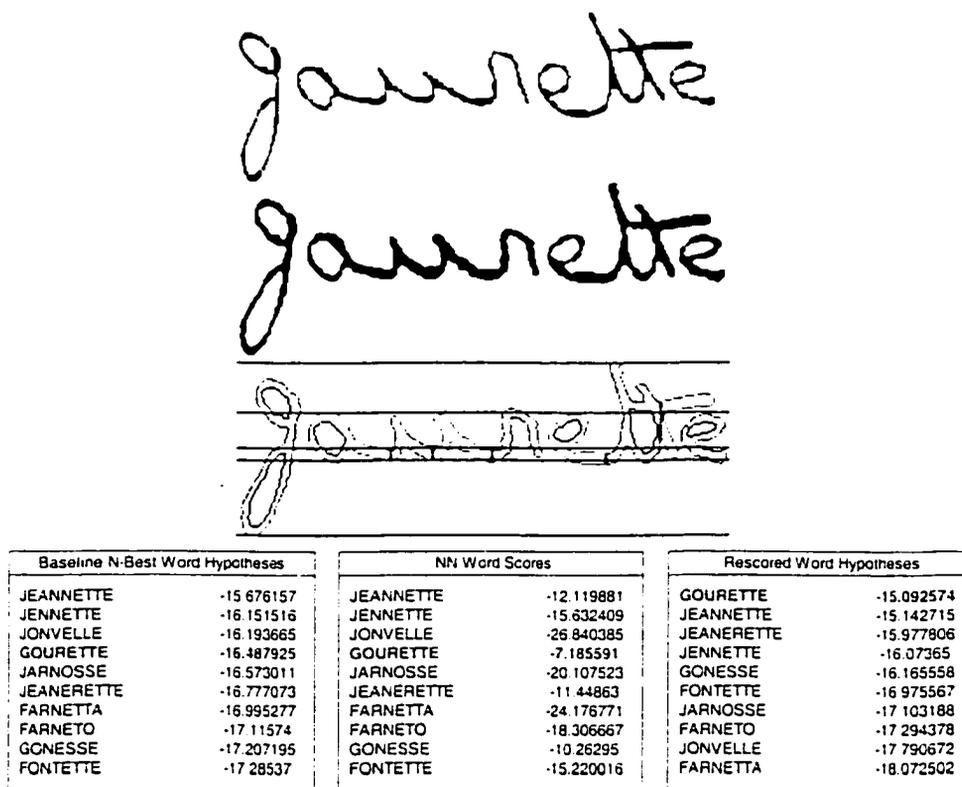
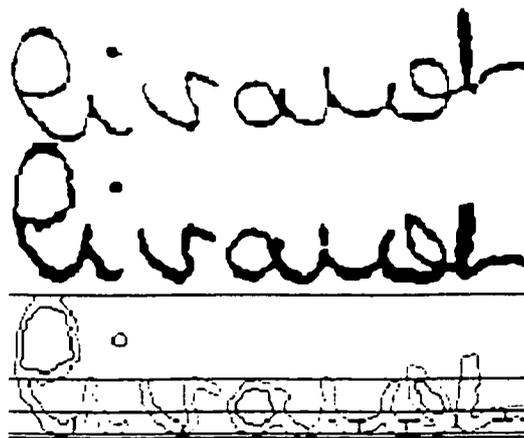


Figure 76 An example of a recognition error corrected by the verification module. The truth word hypothesis (*gourette*) was shifted up from *TOP 4* position to the *TOP 1* position after the verification



Baseline N-Best Word Hypotheses		NN Word Scores		Rescored Word Hypotheses	
BIZANET	-14.022193	BIZANET	-26.516455	CIVAUX	-13.679352
RIGAUD	-14.209303	RIGAUD	-19.803072	LIVAROT	-13.724647
THICOURT	-14.279164	THICOURT	-37.365616	BUANES	-14.312346
BUANES	-14.351032	BUANES	-14.093126	BUOUX	-14.50661
LIVAROT	-14.550066	LIVAROT	-9.04727	HECOURT	-14.797771
HECOURT	-14.758919	HECOURT	-15.017935	RIGAUD	-15.048368
BUICOURT	-14.759047	BUICOURT	-23.214258	LYAUD	-15.560814
BUOUX	-14.852474	BUOUX	-12.54671	BIZANET	-15.896332
CIVAUX	-14.95283	CIVAUX	-6.462975	BUICOURT	-15.027328
LYAUD	-15.018403	LYAUD	-18.634478	THICOURT	-17.742132

Figure 77 An example of a recognition error not corrected by the verification module. The truth word hypothesis (*livarot*) was shifted up from *TOP 5* position to the *TOP 2* position after the verification

Baseline N-Best Word Hypotheses		NN Word Scores		Rescored Word Hypotheses	
SOUES	-5.088444	SOUES	-5.592467	SOUCE	-5.138668
SOUCE	-5.561076	SOUCE	-2.745023	SOUES	-5.164048
SOUEL	-6.777806	SOUEL	-4.381108	SOUEL	-6.418301
POUZE	-6.852237	POUZE	-29.38862	SOUGE	-6.91887
SOUGE	-6.867385	SOUGE	-7.210623	SOURS	-7.451002
SOURS	-6.907815	SOURS	-10.529063	LOUZE	-8.517439
ISQUES	-7.001887	ISQUES	-30.632927	JONES	-8.860287
JONES	-7.031264	JONES	-19.224749	FOLIES	-10.065301
LOUZE	-7.067467	LOUZE	-16.733948	POUZE	-10.232605
FOLIES	-7.125498	FOLIES	-26.724184	ISQUES	-10.546543

Figure 78 An example of a recognition error caused by the verification module. The truth word hypothesis (*SOUES*) was shifted down from *TOP 1* position to the *TOP 2* position after the verification

5.4.7 Computation Breakup

At the first part of our work (Section 5.3), a lot of effort was devoted to improving the performance of the baseline recognition system in terms of recognition time. So, at this point it is worthwhile to ascertain the impact of the verification module in the whole recognition process, that is, in both the recognition accuracy and the recognition time, and not only in improving the accuracy. Table LVII shows the computation breakup for the verification module, where the average time spent on feature extraction, recognition and combination of output scores is presented.

However, to drawn any conclusion, it is also necessary to know the time spend by the baseline recognition system to recognize a word. Table LVIII shows the computation breakup for the baseline recognition system, where the average time spend in feature extraction, recognition and combination of output scores is presented.

Table LVII

Computation breakup for the verification of handwritten words for a *TOP* 10 word hypothesis list and an 80,000-word vocabulary measured on an Athlon 1.1GHz

Module	Recognition Time (sec/word)
Feature Extraction	<100m
Recognition	<5m
Combination	<5m
Total	<110m

Table LVIII

Computation breakup for the recognition of handwritten words for a *TOP* 10 word hypothesis list and an 80,000-word vocabulary on an Athlon 1.1GHz

Module	Recognition Time (sec/word)
Pre-processing	<500m
Segmentation	<200m
Feature Extraction	<50m
Recognition	<15
Total	<16

The time required by the verification process corresponds to less than 1% of the time required by baseline recognition system to generate the *TOP* 10 word hypotheses. However, it can be argued that the nearly 10% rise in the word recognition rate afforded by the verification module is well worth.

However, the time required for verification does not depend on the lexicon size, but only on the number of *N*-best hypothesis. On the other hand, the time spend in recognition by the baseline system depends strongly on the lexicon size. For a 10-word lexicon, the overall recognition time is approximately 1 sec. So, the verification process now corresponds to about 13% of the time required by baseline recognition system to generate the *TOP* 10 word hypotheses. It can be argued that the nearly 0.5% rise in the word recognition rate afforded by the verification module is still worth.

5.4.8 Summary of Accuracy Improvements

Performing a verification of the word hypotheses generated by the baseline recognition system is unquestionably advantageous mainly in case of large and very-large vocabulary tasks. The resulting accuracy is significantly better than that of the baseline recognition system and it is achieved at almost negligible delay in the overall recognition process. Nevertheless, the improvement in recognition accuracy is still very advantageous for medium and small recognition tasks.

The quality of the data used for training the MLP classifier is very bad and that was reflected in the character recognition rate achieved by the MLP classifier relatively to the character recognition rate achieved for the same classifier but trained on the NIST database. However, even with this serious problem, the use of the MLP classifier to score the segments provided by the baseline recognition system succeeded very well and it was possible to improve the word recognition rates significantly.

On the other hand, while the quality of the SRTP character database is not good since it was generated automatically by bootstrapping with no human intervention. This aspect is very relevant since gathering, segmenting and labeling character by hand would be very tedious and time-consuming, and it would not be a guarantee of obtaining better results than those reported in this chapter. The reason is that the database generated by bootstrapping reproduces the same kind of data that is expected during the recognition process.

5.5 Rejection of Word Hypotheses

In this section we present the experiments undertaken during the development of the rejection mechanism to improve the reliability of the word recognition presented in Chapter 4.

Using the simple rejection criterion presented in Section 4.4, our main goal here is to show that the recognition-verification scheme proposed in Chapter 4 is capable not only to improve the recognition accuracy but also the recognition reliability. For such an aim, we have applied the rejection criteria at three different levels as shown in Figure 79.

- At the scores generated at the output of the baseline recognition system alone (RC1);
- At the scores generated at the output of the verification module alone (RC2);
- At the composite scores produced at the output of the recognition-verification system (RC3).

In the first case (RC1) the decision on whether rejecting or accepting a word hypothesis is based on the scores of the *TOP* 1 and *TOP* 2 word hypotheses generated by the baseline recognition system alone. If the word hypothesis is accepted, the word recognition rate is taken at the output of the recognition-verification system.

In the second case (RC2) the decision on whether rejecting or accepting a word hypothesis is based on the scores of the *TOP* 1 and *TOP* 2 word hypotheses generated by the verification module alone. If the word hypothesis is accepted, the word recognition rate is also taken at output of the recognition-verification system.

In the third case (RC1) the decision on whether rejecting or accepting a word hypothesis is based on the composite scores of the *TOP* 1 and *TOP* 2 word hypotheses generated by the combination of the baseline recognition system and the verification module. If the word hypothesis is accepted, the word recognition rate is also taken at output of the recognition-verification system.

Figure 80 shows the word recognition rate (*TOP* 1) as a function of rejection rate for the baseline recognition system alone (RC1), the verification module alone (RC2) and the baseline+verification scheme (RC3) on the SRTP test dataset for an 80k-word lexicon.

Figure 80 highlights that the performance of the rejection based on the baseline recognition system or on the verification module separately is not particularly good. The use of the composite scores in the rejection criterion allows the performance to be significantly improved. So, rejection is a powerful method for reducing the error rate. For example, by rejecting 30% of the word hypotheses (RC3), the *TOP* 1 word recognition rate is increased from 78.05% to almost 95%.

Figures 81, 82 and 83 show the word recognition rate (*TOP* 1) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for the lexicon sizes 10, 10,000 and 80,000 words respectively.

Notice that in all cases (Figures 81, 82 and 83), the rejection criterion proposed in Section 4.4 based on the difference in the scores between the *TOP* 1 and *TOP*

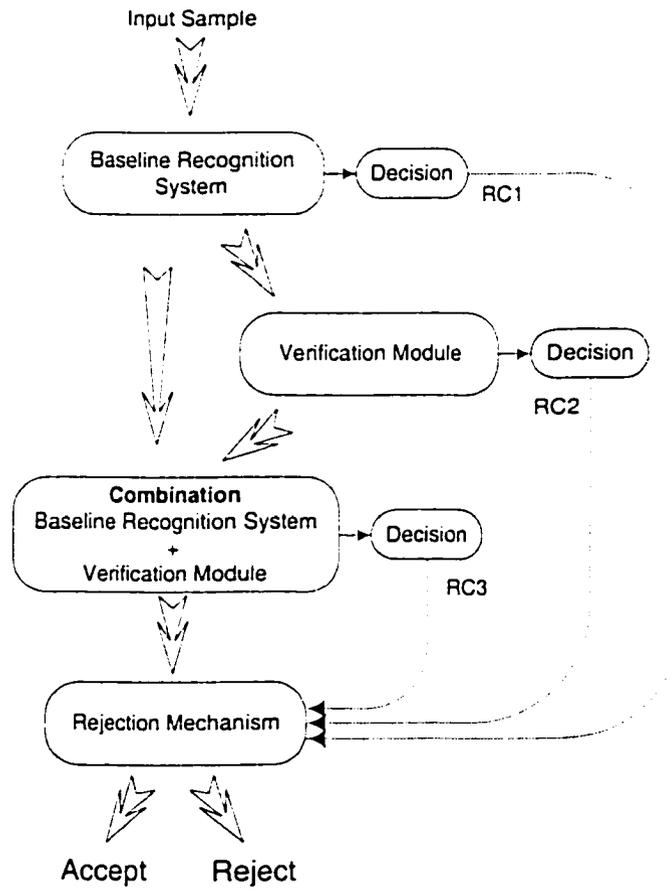


Figure 79 The rejection criterion applied at three different levels: output scores of the baseline recognition system alone, output scores of the verification module alone and output scores of the the combination of the baseline recognition system with the verification

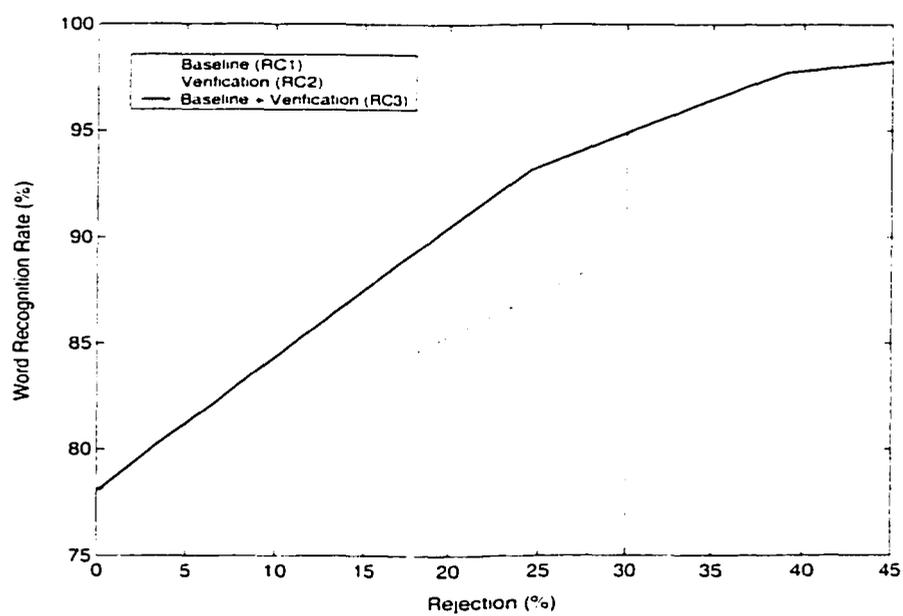


Figure 80 Word recognition rate (*TOP 1*) as a function of rejection rate for the baseline recognition system alone, the verification module alone and the baseline+verification scheme on the SRTP test dataset for an 80k-word lexicon

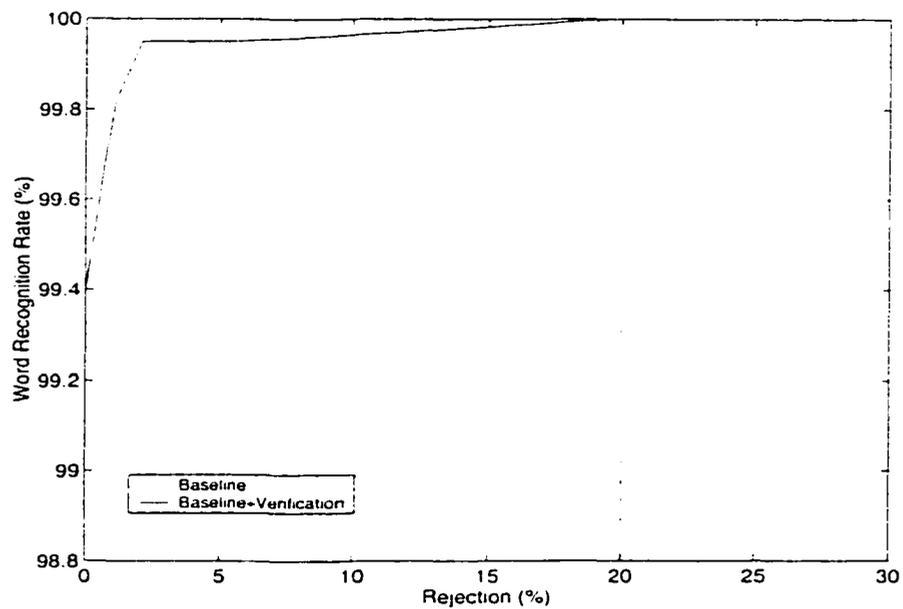


Figure 81 Word recognition rate (*TOP 1*) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for a 10-word lexicon

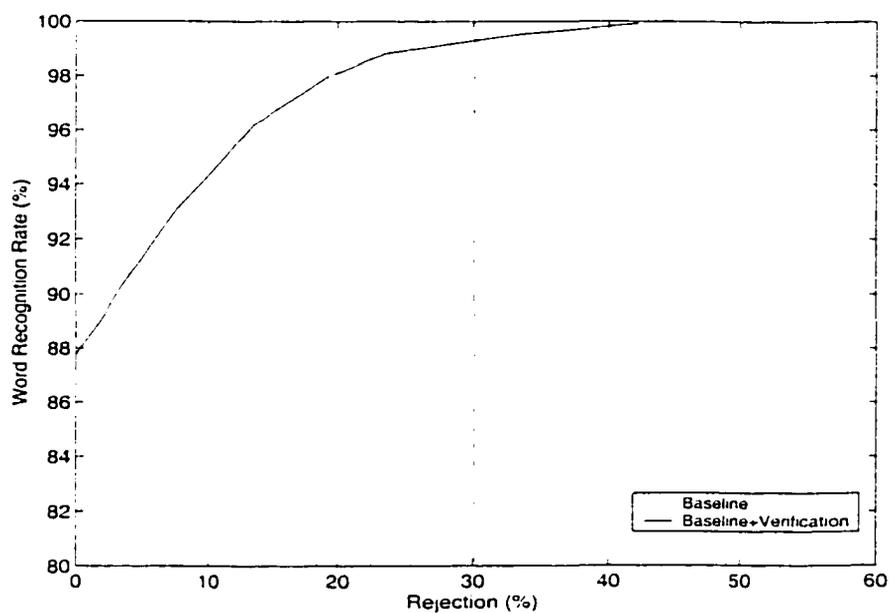


Figure 82 Word recognition rate (*TOP 1*) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for a 10k-word lexicon

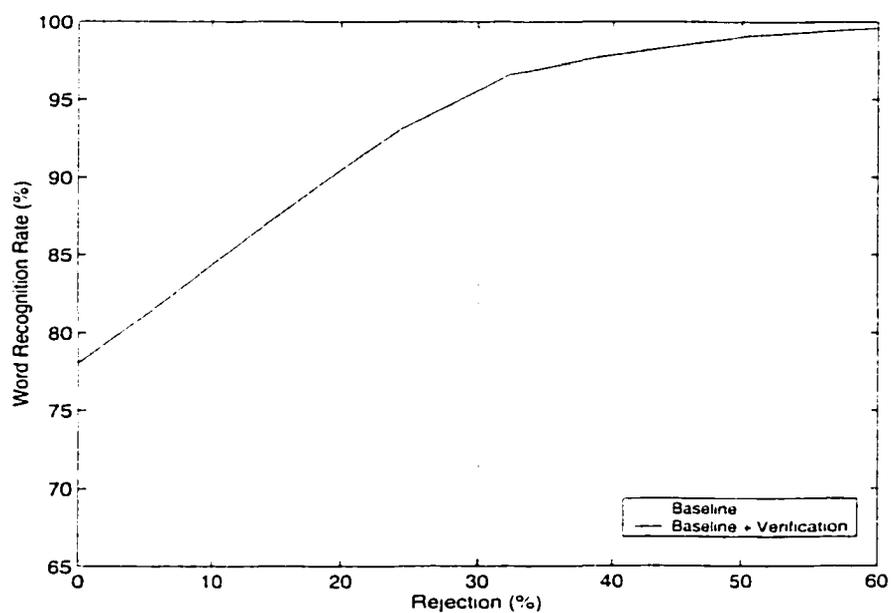


Figure 83 Word recognition rate (*TOP 1*) as a function of rejection rate for the baseline recognition system alone, and the combination of the baseline recognition system with the verification module on the SRTP test dataset for an 80k-word lexicon

2 best word hypotheses is very efficient to improve the recognition rate. However, the improvements are more significant on large vocabularies. It is also unquestionably the advantages of using the scores resulting from the combination recognition-verification relative to the use of the scores of the baseline recognition system alone.

CONCLUSION

This thesis has focused on the problems relating to the computational efficiency and recognition accuracy of large vocabulary handwriting recognition. The main concern addressed in this thesis is that it is possible to improve two mutual conflicting aspects of performance to build up an efficient large vocabulary handwriting recognition system. It is relatively easy to improve recognition speed while trading away some accuracy, for example by pruning the lexicon prior to the recognition, but it is much harder to improve both the recognition speed and the accuracy.

This thesis has described many improvements to a baseline recognition system which has been implemented and tested on a database of unconstrained handwritten city names. The results show that it is now possible to recognize writer-independent, unconstrained handwritten words in reasonable time, using a very-large vocabulary with relative accuracy. This is only possible because of the methods and strategies that have been developed in this thesis.

Improvements in recognition speed are achieved by using the fast two-level HMM decoding strategy that breaks up the computation of word probabilities into two levels: state level and character level. This enables the reuse of the character probabilities to decode all words in the vocabulary, avoiding repeated computation of state sequences. The effects of such a decoding strategy were boosted by incorporation distributed computing concepts that partitions the recognition task among several processors.

Improvements in recognition accuracy are achieved by using a recognition and verification strategy that deals with the N -best word hypotheses generated by the baseline recognition system. Isolated character recognition by a neural network classifier is introduced to overcome some of the limitations of the HMMs and to

reduce the ambiguity between similar character shapes. The recognition accuracy is improved by combining the results of the baseline recognition system with the results of the verification module.

The main results achieved by using the novel recognition strategies presented in this thesis are a 120 speedup factor and about 10% improvement in accuracy of the baseline SRTP recognition system. We have used the system provided by SRTP-France as a benchmark for comparison purposes. On an 80,000-word vocabulary task the baseline recognition system requires tens of minutes to perform the recognition of a single word with recognition rates of about 68%. Using the strategies that have been developed to improve both the recognition speed and recognition accuracy, it is possible to achieve recognition rates up to 78% with recognition times lower than 15 seconds in recognizing unconstrained handwriting word in a very-large vocabulary of 80,000 words. It is also one of the premier system in terms of vocabulary size, recognition speed, and recognition accuracy of its kind.

The recognition accuracy and the recognition times obtained in this thesis may not meet the throughput requirements of many real-life applications, however, they are very encouraging and hopefully, this work will stimulate other researchers to pursue interesting research into this subject since many applications require large vocabularies.

Summary of Results

The results in this thesis are based on the test carried out on the SRTP database. It consists of unconstrained handwritten French city names (handprinted, cursive, and mixed). The experiments were conducted using five different dynamic lexicon sizes, 10, 1,000, 10,000, 40,000, and 80,000 words. We now summarize the main results from this thesis below.

- It is possible to improve the recognition speed by using a tree-structured lexicon with no effect on recognition accuracy. Organizing the lexicon as a tree-structure reduces the number of characters to be decoded in approximately 2 times relatively to the baseline recognition SRTP system that uses a flat lexicon;
- It is possible to reduce the complexity of the search by selecting only the best character models at each tree level with no significant effects on recognition accuracy ($\approx 1\%$);
- The lexicon-driven level building algorithms which integrates the lexical tree and the selection of best character models is about 5.5 to 7.7 faster than the baseline recognition system for a small and large vocabulary tasks respectively. The average loss of accuracy is inferior to 1.1%;
- It is possible to further improve upon the recognition speed by limiting the number of observations aligned at each level as well as by limiting the number of levels of the level building algorithm. The constrained level building algorithm is about 7.8 to 11.8 faster than the baseline recognition system with an increase of about 1.5% in the word recognition rate;
- The contextual dependent constrained level building algorithm further improves upon the recognition speed by adjusting the constraints according to the character models. Incorporating this contextual dependent constraints to the LBA leads to a further speedup in the recognition by factors of about 8.54 to 12.68 for small and large vocabularies respectively, while the loss in accuracy is minimized;
- The peak performance is obtained with the novel fast two-level HMM decoding algorithm that is about 6 to 26 times faster than the baseline recognition system with equal recognition accuracy;
- It is still possible to speedup the recognition by partitioning the recognition task and distributing the recognition process among several processors. This

speeds up the computation by a factor of 120 for large vocabulary while not affecting the recognition rate:

- Besides the improvements in recognition speed, the accuracy can also be improved by postprocessing the N -best word hypothesis list. The verification module based used to rescore the *TOP* 10 word hypotheses achieves a word recognition rate of about 71% for a very-large recognition task. This word recognition rate is 3% higher than the accuracy of the baseline recognition system:
- The combination of the baseline recognition system with the verification module boosts the word recognition rate to 78.05% with less than 1% relative increase in overall recognition time (for a 80,000-word vocabulary):
- The rejection mechanism over the combination of the baseline recognition system with the verification module improves significantly the word recognition rate to about 95% while rejecting 30% of the word hypotheses (for a 80,000-word vocabulary).

In summary, we have constructed a very-large vocabulary handwriting recognition system that can recognizes unconstrained handwritten words with an accuracy of about 78% and a processing time of about 15 seconds on a conventional personal computer ². It is important to notice that we did not attempt to optimize the performance of the components of the verification module, say, segmentation, feature extraction, recognition and combination, neither on recognition accuracy nor on recognition speed.

Contributions

One of the main contributions of this thesis is in extending the limits of off-line handwriting recognition to very-large vocabularies. So far, most of the research

²AMD Athlon 1.1GHz with 512MB of RAM

has been concentrated on improving the recognition accuracy of small and medium vocabulary system. Many of the current handwriting recognition systems focus mainly on accuracy and require a large amount of computation power.

We have brought to attention the importance of the computational complexity aspect in building handwriting recognition systems that deal with large and very-large vocabulary. While for small and medium vocabulary tasks, recognition speed is not a issue, in the case of large vocabulary tasks it becomes an issue. This work suggests that concerns of accuracy and efficiency cannot be detached.

We have demonstrated that by using the methods and strategies used in this thesis it is possible to build a very-large vocabulary handwriting recognition system and achieve high recognition speeds and reasonable recognition rates. Particularly the combination of a recognition and verification modules seems to be a very promising strategy to improve not only the recognition accuracy of large vocabulary systems, but also the accuracy of small and medium vocabulary system.

We have started with a baseline recognition system which performance was limited to medium lexicon of no more than 1,000 words, and even for such vocabulary, the performance was not very good. We ended up with a 80,000-word handwriting recognition system that delivers high recognition speed with speed up factors up to 120 without losing any accuracy when using the fast two-level search strategy in a distributed recognition scheme and with about 10% of improvement in the recognition accuracy at zero-level rejection.

Future Work

During the development of this thesis, we did not have the opportunity to address some problems, or due to time constraints some important aspects have been overlooked. We believe that the performance of the proposed recognition-verification

strategy may be further improved by developing a number of points, such as:

- *Verification of undersegmented and oversegmented characters:* we have assumed that the segmentation of word into characters carried out by the baseline recognition system is reliable. However, about 20% of the words (2,001 out of 10,006 word on the training dataset of the SRTP database) are not correctly segmented. A post-processing of the segmentation points at the verification levels could be useful to overcome some of such segmentation problems;
- *Optimization of the isolated character classifier based on the recognition rate over the word recognition:* The isolated character classifier was developed on the NIST database and further used with the SRTP database. However we believe that better results may be achieved if the design of the character recognition is based on the environment where it will be used;
- *Automatic selection of the N -best word hypotheses:* throughout our work, we have limited the number of N -best word hypotheses to be verified to 10. However, as we have seen, if more word hypotheses are taken, the recognition rates may be improved. Automatic selection of the N -best word hypotheses based on the probability scores may help to improve the recognition rate by selecting more word hypotheses when necessary;
- *Better model duration for isolated characters based on the character shape:* the model duration used in this thesis was not able to improve the accuracy in recognizing handwritten words and characters. However, we believe that a better model based on the shape of the characters will be very useful and will be able to improve further the recognition accuracy of the NN classifier as well as of the combination with the baseline recognition system.

Furthermore, another important aspect that may be further investigated is the *extension of the fast two-level HMM decoding* to other similar problems that involves large vocabularies, such as speech recognition.

APPENDIX 1

Determination of the Control Factors

This appendix gives an overview of the statistical method to determine the values of the control factors s^* , e^* , and Lv^* that optimizes the performance of the constrained level-building algorithm presented in Section 4.1.6. The method is based on statistical experimental design techniques [10].

The method consists of a two-step approach where the first step consists of exploration of the performance of the constrained recognition system over the entire range of acceptable settings of the three control factors. During this step, possible interactions between the control factors are ignored. The exploratory experiment identifies the control factors which have statistically significant effects on the response and yields an approximation to the optimal setting of the control factors. The second step consists of a further exploration of the performance in a small region centered on the optimal setting of the control factors estimated by the exploratory experiment. At this step we take into account the interaction between the control factors. The steps of the experimental design are presented as follows.

- Determine the objectives and identify the factors and levels;
- Determine the number of experiments to run;
- Determine the influence of the factors and the regression model;
- Optimize the control factors to obtain the best results.

Objectives and Factors

We assume that the performance of the recognition system is characterized by two responses, the recognition rate and the recognition time, denoted as y_{RR} and y_{RS} respectively. The responses are assumed to be governed by the three control factors s^* , e^* , and Lv^* .

Optimization involves estimating the relationship between the responses and the control factors. Once the form of this relationship is known approximately, the

control factors may be set so as to optimize the response.

In our recognition system an optimal response means maximizing the recognition rate y_{RR} and minimizing the recognition time y_{RS} . Therefore, we need to determine the combination of experimental factors that simultaneously optimize both response variables.

Base Design

Since we have three experimental factors, we can use a complete factorial plan 3^3 that allows for estimation of the effects of three control factors each at 3 levels as detailed in Table LIX. The appropriate numerical values for the control factors shown in Table LIX were determined at the first step.

Table LIX

The level of the control factors.

Control Factor	Levels		
	1	2	3
s^*	-0.5	0.75	2.0
e^*	0.5	1.25	2.0
L_c^*	-2.0	0.0	2.0

Fifty-four experimental runs were conducted corresponding to the 27 combinations of the three control factors shown in Table LIX. However, to allow the estimation of second order effects the experiment was replicated, that is, another 27 experimental runs corresponding to one replication were conducted on a different dynamically generated lexicon. Table LX shows both responses, that is, y_{RR} and y_{RS} for each trial, considering two 100-word dynamically generated lexicons, one for the first 27 trials, and another for the last 27 trials. The results were obtained on the validation dataset of the SRTP database.

Table LX

The full factorial 3³ array for control factors with one replication.

Trial No.	Levels of the Control Factor	Recognition Rate (%)	Recognition Time (sec/word)
1	-0.5	94.53	0.1166
2	0.75	94.76	0.1211
3	2	94.82	0.1254
4	-0.5	95.08	0.1217
5	0.75	95.25	0.1254
6	2	95.31	0.1301
7	-0.5	95.08	0.1257
8	0.75	95.25	0.1300
9	2	95.31	0.1337
10	-0.5	94.53	0.1169
11	0.75	94.76	0.1215
12	2	94.82	0.1257
13	-0.5	95.08	0.1211
14	0.75	95.25	0.1256
15	2	95.31	0.1298
16	-0.5	95.08	0.1261
17	0.75	95.25	0.1301
18	2	95.31	0.1350
19	-0.5	88.92	0.1048
20	0.75	89.06	0.1083
21	2	89.06	0.1115
22	-0.5	89.47	0.1094
23	0.75	89.55	0.1131
24	2	89.55	0.1163
25	-0.5	89.47	0.1134
26	0.75	89.55	0.1174
27	2	89.55	0.1208
28	-0.5	94.94	0.1171
29	0.75	95.17	0.1211
30	2	95.25	0.1252
31	-0.5	95.28	0.1219
32	0.75	95.54	0.1256
33	2	95.63	0.1294
34	-0.5	95.28	0.1255
35	0.75	95.54	0.1302
36	2	95.63	0.1341
37	-0.5	94.94	0.1166
38	0.75	95.17	0.1214
39	2	95.25	0.1255
40	-0.5	95.28	0.1216
41	0.75	95.54	0.1250
42	2	95.63	0.1304
43	-0.5	95.28	0.1258
44	0.75	95.54	0.1304
45	2	95.63	0.1338
46	-0.5	89.30	0.1047
47	0.75	89.44	0.1085
48	2	89.44	0.1115
49	-0.5	89.64	0.1090
50	0.75	89.78	0.1128
51	2	89.78	0.1159
52	-0.5	89.64	0.1138
53	0.75	89.78	0.1173
54	2	89.78	0.1204

Figures 84 and 85 show the effects of the control factors on both responses for a 100-entry lexicon. Figure 84 shows that the control factor Lv^* has the most pronounced effect on y_{RR} . The effect of this control factor is approximately quadratic. The other control factors have less effect and they seem to be approximately linear. Figure 85 shows that the control factor Lv^* has the most pronounced effect on y_{RS} , but the effect due to the control factors s^* and e^* are also pronounced. However, the effects of both factors are approximately linear.

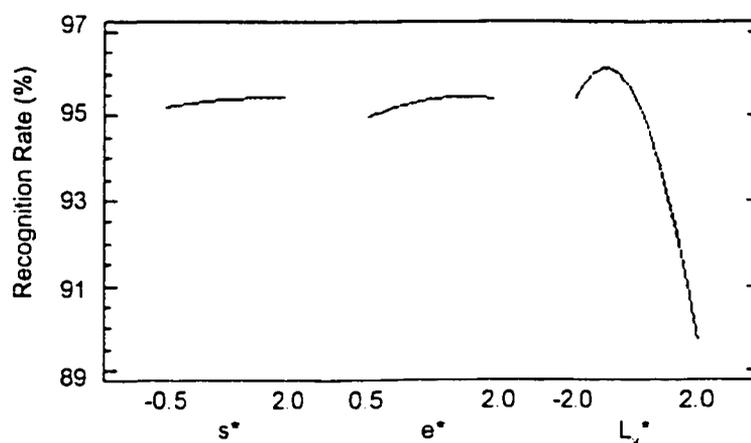


Figure 84 Main effects of the control factors on the recognition rate for a 100-word vocabulary.

Analysis of Variance

Tables LXI and LXII list the standard analysis of variance for the experimental runs. Table LXI shows that the control factors s^* , e^* , and Lv^* have a statistically significant effect (at the 5% level) on y_{RR} . Moreover, the quadratic effects of the control factors e^* and Lv^* are significant. These results confirm what we have seen in Figure 84. The effects of the interaction of the control factors are not significant and they can be neglected.

Table LXII shows that the control factors s^* , e^* , and Lv^* have a statistically significant effect (at the 5% level) on y_{RS} . Moreover, the quadratic effect of the control

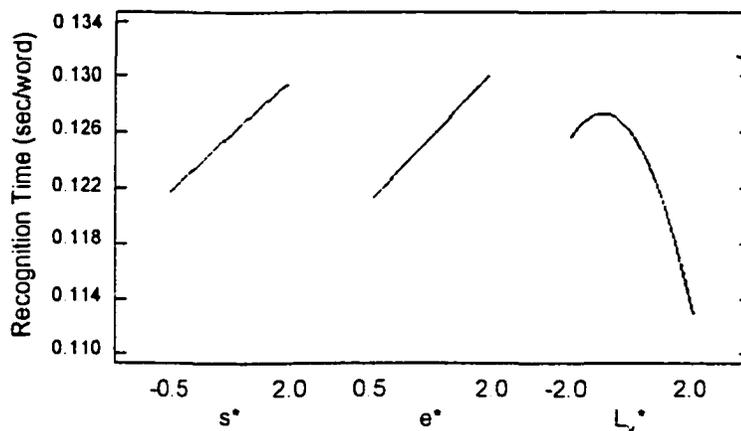


Figure 85 Main effects of the control factors on the recognition speed for a 100-word vocabulary.

factor Lv^* is significant. These results confirm what we have seen in Figure 85. The effects of the interaction of the control factors e^* and Lv^* is also significant. For the rest, the interactions are not significant and they can be neglected.

Table LXI

Analysis of Variance for Recognition Rate (y_{RR}).

Factor	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
s^*	0.503855	1	0.503855	16.81	0.0002*
e^*	1.68938	1	1.68938	56.36	0*
Lv^*	293.831	1	293.831	9801.77	0*
s^*s^*	0.0593774	1	0.0593774	1.98	0.1663
s^*e^*	0.000552509	1	0.000552509	0.02	0.8926
s^*Lv^*	0.0422682	1	0.0422682	1.41	0.2414
e^*e^*	0.563126	1	0.563126	18.79	0.0001*
e^*Lv^*	0.000138106	1	0.000138106	0	0.9462
Lv^*Lv^*	97.9435	1	97.9435	3267.26	0*
Error	1.319	44	0.0299773		
Total	395.952	53			

*Denotes significance at 5% level.

Table LXII

Analysis of Variance for Recognition Time (y_{RS}).

Factor	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
s^*	0.000566408	1	0.000566408	3903.83	0*
e^*	0.000712178	1	0.000712178	4908.51	0*
Lv^*	0.00148093	1	0.00148093	10206.93	0*
s^*s^*	1.14E-07	1	1.14E-07	0.79	0.3801
s^*e^*	1.11E-08	1	1.11E-08	0.08	0.7839
s^*Lv^*	2.73375E-06	1	2.73375E-06	18.84	0.0001*
e^*e^*	1.86E-08	1	1.86E-08	0.13	0.7223
e^*Lv^*	2.71E-08	1	2.71E-08	0.19	0.6679
Lv^*Lv^*	0.000515184	1	0.000515184	3550.78	0*
Error	6.38398E-06	44	1.45E-07		
Total	0.00328399	53			

*Denotes significance at 5% level.

Models for the Experiment

The form of the experiment and the Figures 84 and 85 suggest fitting a quadratic model for both y_{RR} and y_{RS} . So, we derive two regression models where the independent variables are s^* , e^* and Lv^* , and the dependent variables are the responses of the recognition system, that is, the recognition rate y_{RR} and recognition time y_{RS} . The regression models have the following form:

$$y_{RR} = \mu_{RR} + a_1s^* + a_2e^* + a_3Lv^* + a_4s^{*2} + a_5s^*e^* + a_6s^*Lv^* + a_7e^{*2} + a_8e^*Lv^* + a_9Lv^{*2} \quad (1.1)$$

$$y_{RS} = \mu_{RS} + b_1s^* + b_2e^* + b_3Lv^* + b_4s^{*2} + b_5s^*e^* + b_6s^*Lv^* + b_7e^{*2} + b_8e^*Lv^* + b_9Lv^{*2} \quad (1.2)$$

where μ_{RR} and μ_{RS} are the mean recognition rate and recognition time respectively. a_1, \dots, a_9 and b_1, \dots, b_9 are the coefficients to be determined and s^* , e^* and Lv^* represent the levels of the corresponding control factors.

Fitting the models of Equations 1.1 and 1.2 to the data in Table LX explains 99.66% and 99.80% of the variability in y_{RR} and y_{RS} respectively. The coefficients of the regression models fitted to the data by a least square method are shown in Tables LXIII and LXIV. The fitted regression models are used to obtain the final estimate of the optimal setting of s^* , e^* and Lv^* .

Table LXIII

Coefficients of fitted regression models for recognition rate (y_{RR}).

Coefficient	Estimate
μ_{RR}	94.3297
a_1	0.16857
a_2	1.25546
a_3	-1.41387
a_4	-0.0450194
a_5	-0.00511791
a_6	-0.0167865
a_7	-0.385114
a_8	-0.00159922
a_9	-0.714229

Multiple Correlation Coefficient,
 $R^2=0.9966$.

Table LXIV

Coefficients of fitted regression models for recognition time (y_{RS}).

Coefficient	Estimate
μ_{RS}	0.11582
b_1	0.00329546
b_2	0.00612235
b_3	-0.00313364
b_4	-0.0000624
b_5	-0.0000228889
b_6	-0.000135
b_7	-0.0000699259
b_8	0.0000223889
b_9	-0.00163806

Multiple Correlation Coefficient,
 $R^2=0.9980$.

Multiple Response Optimization

The desirability function approach is one of the most widely used methods in industry for the optimization of multiple response processes. It is based on the idea that the "quality" of a product or process that has multiple quality characteristics, with one of them outside of some "desired" limits, is completely unacceptable. The method finds operating conditions, the values of s^* , e^* and Lv^* in our case, that provide the "most desirable" response values.

For each response y_i , a desirability function $d_i(y_i)$ assigns numbers between 0 and 1 to the possible values of y_i , with $d_i(y_i) = 0$ representing a completely undesirable value of y_i and $d_i(y_i) = 1$ representing a completely desirable or ideal response value. The individual desirabilities are then combined using the geometric mean, which gives the overall desirability D , which is defined as:

$$D = \sqrt{(d_{RR}y_{RR} \times d_{RS}y_{RS})} \quad (1.3)$$

In our system an optimal response means maximizing y_{RR} and minimizing y_{RS} . Therefore, we need to determine the combination of experimental factors that simultaneously optimize both response variables. We do so by maximizing Equation 1.3. Optimization of D with respect to the three control factors s^* , e^* and Lv^* was carried out using the Statgraphics software.

Figure 86 shows the estimated response surface while Table LXV shows the combination of factor levels which maximize the desirability function over the indicated region. Table LXVI shows the responses for the optimum values of the control factors.

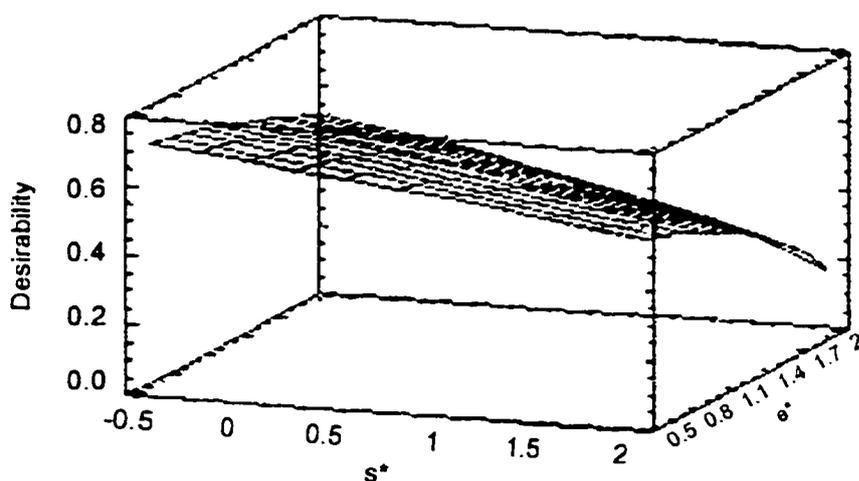


Figure 86 Estimated response surface for $Lv^* = 0.0$.

Notice that the optimal control factors were determined for a 100-word lexicon. Since the responses y_{RR} and y_{RS} depend on the lexicon size, the complete procedure must be repeated to determine the optimal setup for other lexicon sizes.

Table LXV

Combination of factor levels that maximize the desirability function.

Factor	Low	High	Optimum
s^*	-0.5	2.0	-0.5
e^*	0.5	2.0	0.5
Lv^*	-2.0	2.0	-0.68

Table LXVI

Optimal responses for the optimum setup of the control factors.

Response	Optimum
y_{RR}	95.3519%
y_{RS}	0.118 sec/word

APPENDIX 2

Stroke Warping Algorithm

This appendix presents the stroke warping algorithm used to create variations in character stroke consisting of small changes in size, rotation, and horizontal and vertical scalings to produce alternate character forms.

The following variables are used in the stroke warping algorithm:

- I : A character image of size $m \times n$;
- I' : Resized character image;
- I'' : Re-scaled character image;
- I''' : Rotated character image;
- m, m', m'' : Horizontal dimension of an image;
- n, n', n'' : Vertical dimension of an image;
- Tr : Threshold for image resizing which ranges between 0.8 and 1.2;
- Tsx : Threshold for horizontal image re-scaling which ranges between 0.6 and 1.4;
- Tsy : Threshold for vertical image re-scaling which ranges between 0.6 and 1.4;
- Tt : Threshold for image rotating which ranges between -15 and +15 degrees;

The ranges of the thresholds Tr , Ts , and Tt have been determined empirically. The values of the thresholds $T_{(.)}$ are uniformly distributed between $T_{(.)min}$ and $T_{(.)max}$ ($T_{(.)min} \leq T_{(.)} \leq T_{(.)max}$). The complete procedure for generating alternate character forms for a given character class C is described as follows:

1. Select randomly one sample image I from the character class C ;
2. Resize the character image I of size $m \times n$ by Tr using nearest neighbor interpolation. If Tr is between 0 and 1.0, the resulting image I' is smaller than I . If Tr is greater than 1.0, I' is larger than I . The aspect ratio is maintained;
3. Re-scale the (resized) character image I' of size $m' \times n'$ by Tsx and Tsy using nearest neighbor interpolation. The aspect ratio is not maintained;

4. Rotate the (resized and re-scaled) character image I'' of size $m'' \times n''$ by Tt degrees using the nearest neighbor interpolation;
5. Repeat from Step 1 if more samples are required;
6. End.

Figure 87 shows the character images resulting from the steps of the stroke warping algorithm.

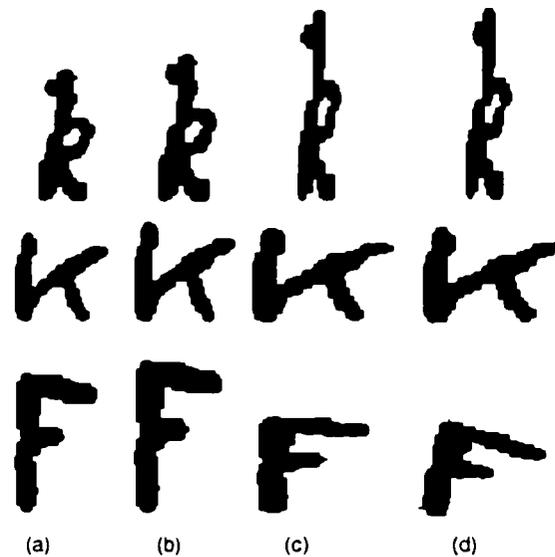


Figure 87 The character image resulting from each step of the stroke warping algorithm: (a) original character image, (b) resized character image, (c) re-scaled character image, (d) final character image after rotation.

APPENDIX 3

Hidden Markov Models

This appendix gives an overview of the HMM modeling considering the conventional definition of HMMs in which outputs are generated by states (state model) and the alternative definition in which outputs are generated by transitions into states (transition model).

The Hidden Markov Model Concept

A hidden Markov model is a doubly stochastic process with an underlying stochastic process that is not observable, but can be observed through another stochastic process that produces the sequence of observations. The hidden process consists of a set of states connected to each other by transitions with probabilities, while the observed process consists of a set of outputs or observations, each of which may be emitted by each state (or transition) according to some output probability density function. In the context of this thesis we consider discrete observations drawn from an alphabet.

In some applications it is more convenient to produce observations by transitions rather than by states. Furthermore, it is sometimes useful to allow transitions with no output, in order to model the absence of an event in a given stochastic process. The following parameters are used to defined an HMM:

- T : Length of the observation sequence $O = (o_1 o_2 \dots o_T)$;
- N : Number of states in the model;
- M : Number of possible observation symbols;
- $S = \{s_1, s_2, \dots, s_N\}$: Set of possible states of the model;
- q_t : State of the process at time t ;
- $V = \{v_1, v_2, \dots, v_M\}$: Discrete set of possible observation symbols;

- $A = \{a_{ij}\}$: State transition probability distribution, in which a_{ij} denotes the probability of going from state s_i at time t to state s_j at time $t + 1$:
- $B = \{b_j(k)\}$: Observation symbol probability distribution. in which $b_j(k)$ denotes the output symbol probability in state s_j of producing a real observation symbol $o_t = v_k$:
- $\Pi = \{\pi_i\}$: The initial state distribution. in which π_i denotes the probability of being in state i at time $t = 1$.

Given a model, to be represented by the compact notation $\lambda = \{A, B, \Pi\}$, three basic problems of interest must be solved for the model:

1. Given an observation sequence $O = (o_1 o_2 \dots o_T)$ and a model λ , how do we compute $P(O|\lambda)$, the probability of the observation sequence given the model? This is the evaluation problem.
2. Given an observation sequence $O = (o_1 o_2 \dots o_T)$ and a model λ , how do we find the optimal state sequence $Q = (q_1 q_2 \dots q_T)$ in λ that has generated O ? This is the recognition problem:
3. Given a set of observation sequences and an initial model λ , how can we re-estimate the model parameters so as to increase the likelihood of generating this set of sequences? This is the training problem.

In the next section we present the algorithms devoted to solve these three problems for both state and transition models. We assume a predefined initial state $s = s_1$ and a predefined final state $s = s_N$ that do not change over time. In this case, $\pi_1 = 1$ and $\pi_i = 0$ for $i = 2, 3, \dots, N$.

The Evaluation Problem

To compute $P(O|\lambda)$ the *forward-backward* procedure is used. We define the forward probability $\alpha_t(i)$ and the backward probability $\beta_t(i)$ as:

$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = s_i | \lambda) \quad (3.1)$$

$$\beta_t(i) = P(o_{t+1} o_{t+2} \dots o_T | q_t = s_i, \lambda) \quad (3.2)$$

The forward and the backward procedures for the state model are presented elsewhere [139]. On the other hand, there is a lack for the transition model. So, in this section we present the forward and the backward procedures for the transition model and considering the presence of null transitions.

The Forward and Backward Algorithm for Transition Model

The forward and the backward procedures [139] are modified to take into account the assumption that symbols are emitted along transitions.

The Forward Procedure

1. Initialization:

$$\alpha_1(1) = 1 \quad (3.3)$$

2. Induction ($2 \leq t \leq T + 1$, $1 \leq j \leq N$):

$$\alpha_t(j) = \sum_{i=1}^N [\alpha_{t-1}(i) a_{ij}^o] \quad (3.4)$$

3. Termination:

$$P = \alpha_{T-1}(N) \quad (3.5)$$

The Backward Procedure

1. Initialization:

$$\beta_{T+1}(N) = 1 \quad (3.6)$$

2. Induction ($1 \leq t \leq T$, $1 \leq i \leq N$):

$$\beta_t(i) = \sum_{j=1}^N [\beta_{t+1}(j) a_{ij}^{\alpha_{t+1}}] \quad (3.7)$$

3. Termination:

$$P = \beta_1(1) \quad (3.8)$$

The Forward and Backward Algorithm for Transition Model with Presence of Null Transitions

The forward and the backward procedures [139] are modified to take into account the assumption that symbols are emitted along transitions and that may exist null transitions.

The Forward Procedure

1. Initialization ($1 \leq j \leq N$):

$$\begin{aligned} \alpha_1(1) &= 1 \\ \alpha_1(j) &= \sum_{i=1}^N [\alpha_1(i) a_{ij}^{\phi}] \end{aligned} \quad (3.9)$$

2. Induction ($2 \leq t \leq T + 1$, $1 \leq j \leq N$):

$$\alpha_t(j) = \sum_{i=1}^N [\alpha_{t-1}(i)a_{ij}^{o_t} + \alpha_t(i)a_{ij}^{\prime\phi}] \quad (3.10)$$

3. Termination:

$$P = \alpha_{T+1}(N) \quad (3.11)$$

The Backward Procedure

1. Initialization ($1 \leq j \leq N$):

$$\begin{aligned} \beta_{T+1}(N) &= 1 \\ \beta_{T+1}(i) &= \sum_{j=1}^N [\beta_{T+1}(j)a_{ij}^{\prime\phi}] \end{aligned} \quad (3.12)$$

2. Induction ($1 \leq t \leq T$, $1 \leq i \leq N$):

$$\beta_t(i) = \sum_{j=1}^N [\beta_{t+1}(j)a_{ij}^{o_{t+1}} + \beta_t(j)a_{ij}^{\prime\phi}] \quad (3.13)$$

3. Termination:

$$P = \beta_1(1) \quad (3.14)$$

The Training Problem

The main advantage of HMM based approaches is the existence of a procedure called *Baum-Welch* algorithm [139] that adjusts iteratively and automatically HMM parameters given a training set of observation sequences. This algorithm guarantees that the model converges to a local maximum of the probability of observation of the training set according to the maximum likelihood estimation criterion. The local maximum depends strongly on the initial HMM parameters.

To describe the procedure for reestimation (iterative update and improvement) of the HMM parameters, we first define $\xi_t(i, j)$ the probability of being at state s_i at time t and state s_j at time $t + 1$, producing a real observation o_t , given the model and the observation sequence, and $\xi'_t(i, j)$, the probability of being at state s_i at time t and state s_j at time t , producing a null observation symbol Φ , given the model and the observation sequence as:

$$\xi_t(i, j) = P(q_t = s_i, q_{t+1} = s_j | O, \lambda) \quad (3.15)$$

$$\xi'_t(i, j) = P(q_t = s_i, q_t = s_j | O, \lambda) \quad (3.16)$$

We also define $\gamma_t(i)$ as the probability of being at state s_i at time t , given the entire observation sequence and the model as:

$$\gamma_t(i) = P(q_t = s_i | O, \lambda) \quad (3.17)$$

Again, in this section we present the *Baum-Welch* algorithm for the transition model and considering the presence of null transitions. The *Baum-Welch* algorithm for the state model is presented elsewhere [139].

Algorithm for Transition Model

From the definitions of the forward and backward variables, we can rewrite $\xi_t(i, j)$ in the form:

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}^{o_t} \mathcal{Z}_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}^{o_t} \mathcal{Z}_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i)a_{ij}^{o_t} \mathcal{Z}_{t+1}(j)} \quad (3.18)$$

We can relate $\gamma_t(i)$ and $\xi_t(i, j)$ by summing over j , giving:

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j) \quad (3.19)$$

The reestimation of the HMM parameters $\{a_{ij}^k\}$ for A is:

$$a_{ij}^k = \frac{\text{expected number of transitions from } s_i \text{ to } s_j \text{ and observing } v_k}{\text{expected number of being in } s_i} \quad (3.20)$$

Given the definition of $\gamma_t(i)$ and $\xi_t(i, j)$, we have:

$$a_{ij}^k = \frac{\sum_{t=1}^T \varrho(o_t, v_k) \xi_t(i, j)}{\sum_{t=1}^T \gamma_t(i)} = \frac{\sum_{t=1}^T \varrho(o_t, v_k) \alpha_t(i) a_{ij}^{o_t} \mathcal{Z}_{t+1}(j)}{\sum_{t=1}^T \alpha_t(i) \mathcal{Z}_t(i)} \quad (3.21)$$

where we have defined $\varrho(o_t, v_k)$ as:

$$\varrho(o_t, v_k) = \begin{cases} 1 & \text{if } o_t = v_k \\ 0 & \text{otherwise} \end{cases} \quad (3.22)$$

If we define the current model as $\lambda = \{A\}$ and use that to compute the right-hand side of Equation 3.20, and we define the reestimated model as $\bar{\lambda} = \{\bar{A}\}$, as determined by the left-hand side of Equation 3.20, then it has been proven by Baum

and his colleagues that the initial model λ defines a critical point of the likelihood function. in which case $\bar{\lambda} = \lambda$: or model $\bar{\lambda}$ is more likely than model λ in the sense that $P(O|\bar{\lambda}) > P(O|\lambda)$. that is, we have found a new model $\bar{\lambda}$ from which the observation sequence is more likely to have been produced.

Based on the above procedure, if we iteratively use $\bar{\lambda}$ in place of λ and repeat the reestimation calculation, we then can improve the probability of O being observed from the model until some limiting point is reached.

Algorithm for Transition Model with Presence of Null Transitions

If we consider the presence of null transitions, we have to reestimate $a'_{ij}{}^{\Phi}$ and $a''_{ij}{}^{\kappa}$. From the definitions of the forward and backward variables, we can rewrite $\xi'_t(i, j)$ in the form:

$$\xi'_t(i, j) = \frac{\alpha_t(i)a'_{ij}{}^{\Phi}\beta_t(j)}{P(O|\lambda)} \quad (3.23)$$

And now, $\gamma_t(i)$ is defined as:

$$\gamma_t(i) = \sum_{j=1}^N [\xi_t(i, j) + \xi'_t(i, j)] = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \quad (3.24)$$

The reestimation of the HMM parameters $\{a'_{ij}{}^{\Phi}\}$ for A' is:

$$a'_{ij}{}^{\Phi} = \frac{\text{expected number of transitions from } s_i \text{ to } s_j \text{ and producing } \Phi}{\text{expected number of being in } s_i} \quad (3.25)$$

Given the definition of $\gamma_t(i)$ and $\xi'_t(i, j)$, we have:

$$a_{ij}^{k\phi} = \frac{\sum_{t=1}^T \xi'_t(i, j)}{\sum_{t=1}^T \gamma_t(i)} = \frac{\sum_{t=1}^T \alpha_t(i) a_{ij}^{k\phi} \beta_t(j)}{\sum_{t=1}^T \alpha_t(i) \beta_t(i)} \quad (3.26)$$

On the other hand, the reestimation of a_{ij}^k does not change and it is still given by Equation 3.21.

The Recognition Problem

The recognition problem is usually solved using a near-optimal procedure, the Viterbi algorithm, by looking for the best state sequence $Q = (q_1 q_2 \dots q_T)$ for the given observation sequence $O = (o_1 o_2 \dots o_T)$. To do so, we first define the best score (highest probability) along a single path denoted as $\delta_t(i)$, which accounts for the first t observations and ends in state i as:

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1 q_2 \dots q_{t-1}, q_t = i, o_1 o_2 \dots o_t | \lambda) \quad (3.27)$$

We also define the quantity $\omega_t(i)$ which goal is to recover the best state sequence by a procedure called backtracking. The quantities $\delta_t(i)$ and $\omega_t(i)$ can be computed recursively. In the next sections we present the conventional Viterbi algorithm for HMM outputs attached to states (state model), HMM outputs attached to transitions (transition model) and HMM outputs attached to transitions with the presence of null transitions.

The Viterbi Algorithm for State Model

Considering that the state s_1 is the only possible initial state and that the state s_N is the only possible terminal state, the complete procedure for finding the best state sequence can be stated as follows:

1. Initialization:

$$\begin{aligned}\delta_1(1) &= 1 \\ \omega_1(1) &= 0\end{aligned}\tag{3.28}$$

2. Recursion ($2 \leq t \leq T$, $1 \leq j \leq N$):

$$\begin{aligned}\delta_t(j) &= \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t) \\ \omega_t(j) &= \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}]\end{aligned}\tag{3.29}$$

3. Termination:

$$\begin{aligned}P &= \delta_T(N) \\ q_T &= N\end{aligned}\tag{3.30}$$

4. Path Backtracking ($T - 1 \leq t \leq 1$):

$$q_t = \omega_{t+1}(q_{t+1})\tag{3.31}$$

The Viterbi Algorithm for Transition Model

Here, we need to redefine $A = \{a_{ij}^k\}$, in which a_{ij}^k is the probability of going from state s_i at time t to state s_j at time $t + 1$, and at the same time producing a real observation symbol $o_t = v_k$:

Considering that the state s_1 is the only possible initial state and that the state s_N is the only possible terminal state, the complete procedure for finding the best state sequence can be stated as follows:

1. Initialization:

$$\begin{aligned}\delta_1(1) &= 1 \\ \omega_1(1) &= 0\end{aligned}\tag{3.32}$$

2. Recursion ($2 \leq t \leq T + 1$, $1 \leq j \leq N$):

$$\begin{aligned}\delta_t(j) &= \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{o_{t-1}}] \\ \omega_t(j) &= \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{o_{t-1}}]\end{aligned}\tag{3.33}$$

3. Termination:

$$\begin{aligned}P^* &= \delta_{T+1}(N) \\ q_{T+1}^* &= N\end{aligned}\tag{3.34}$$

4. Path Backtracking ($T \leq t \leq 1$):

$$q_t^* = \omega_{t+1}(q_{t+1}^*)\tag{3.35}$$

The Viterbi Algorithm for Transition Model with Presence of Null Transitions

In many applications it is useful to introduce the possibility of null transitions that change state but results in no output. We must now make the appropriate changes in the formulas to accommodate the presence of null transitions.

Let $A' = \{a'_{ij}{}^\Phi\}$ be the probability of null transition from state s_i at time t to state s_j at time t , producing a null observation symbol denoted as Φ . We must have:

$$\sum_{j=1}^N [a'_{ij}{}^\Phi + \sum_{k=1}^M a_{ij}^k] = 1\tag{3.36}$$

So, the complete procedure for finding the best state sequence can now be stated as follows:

1. Initialization ($1 \leq j \leq N$):

$$\begin{aligned}\delta_1(1) &= 1 \\ \omega_1(1) &= 0 \\ \delta_1(j) &= \max_{1 \leq i \leq N} [\delta_1(i)a_{ij}^{\prime\phi}] \\ \omega_1(j) &= \arg \max_{1 \leq i \leq N} [\delta_1(i)a_{ij}^{\prime\phi}]\end{aligned}\tag{3.37}$$

2. Recursion ($2 \leq t \leq T + 1$, $1 \leq j \leq N$):

$$\begin{aligned}\delta_t(j) &= \max \left\{ \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}^{\prime\phi}]; \max_{1 \leq i \leq N} [\delta_t(i)a_{ij}^{\prime\phi}] \right\} \\ \omega_t(j) &= \begin{cases} \arg \max_{1 \leq i \leq N} [\delta_t(i)a_{ij}^{\prime\phi}] & \text{if } ij \text{ is null} \\ \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}^{\prime\phi}] & \text{otherwise} \end{cases}\end{aligned}\tag{3.38}$$

$$\zeta_t(j) = \begin{cases} 1 & \text{if } ij \text{ is null} \\ 0 & \text{otherwise} \end{cases}$$

3. Termination:

$$\begin{aligned}P^* &= \delta_{T+1}(N) \\ q_{T+1}^* &= N\end{aligned}\tag{3.39}$$

4. Path Backtracking ($T \leq t \leq 1$):

$$q_t^* = \omega_{t+1}(q_{t+1}^*)\tag{3.40}$$

However, a new variable that accounts for the presence of null transitions, denoted as q'_t is introduced. After computing q_t^* we have to check if the variable ζ indicates the presence of a null transition, that is, $\zeta_t(q_t^*) = 1$. In such a case, without decreasing the time index t , q'_t is computed as:

$$q'_t = \omega_t(q_t^*) \quad \text{if } \zeta_t(q_t^*) = 1 \quad (3.41)$$

Following the presence of a null transition during the backtracking procedure, that is $\zeta_{t+1}(q_{t+1}^*) = 1$, the computation of q_t^* is modified slightly as follows (now with a decrease in the time index t):

$$q_t^* = \omega_{t+1}(q'_{t+1}) \quad \text{if } \zeta_{t+1}(q_{t+1}^*) = 1 \quad (3.42)$$

So, the best state sequence denoted as Q^* will be given by:

$$Q^* = (q_T^* q'_T q_{T-1}^* q'_{T-1} \cdots q_2^* q'_2 q_1^* q'_1) \quad (3.43)$$

The Fast Two-Level Decoding Algorithm for Transition Model

In Section 4.1.7 we have introduced the *Fast Two-Level Decoding Strategy*. However, we have presented the algorithm in terms of state model due to its novel aspect. In this section we present the *Fast Two-Level Decoding Algorithm* considering the transition model.

First Level: Character Model Decoding

1. Initialization: for $1 \leq s \leq T$, $i = 1$

$$\begin{aligned} \delta_s(1) &= 1 \\ \omega_s(1) &= 0 \end{aligned} \quad (3.44)$$

2. Recursion: for $s < t \leq T$, $1 \leq j \leq N$

$$\begin{aligned} \delta_t(j) &= \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{o_{t-1}}] , \\ \omega_t(j) &= \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{o_{t-1}}] \end{aligned} \quad (3.45)$$

3. Termination: if $j = N$

$$\begin{aligned} e &= t \\ \chi(s, e) &= \delta_t(N) \\ q_e^* &= N \end{aligned} \quad (3.46)$$

4. Backtracking: for $t = e - 1, e - 2, \dots, s$, q_t^* is determined recursively by:

$$\begin{aligned} q_t^* &= \omega_{t+1}(q_{t+1}^*) \\ \epsilon(s, e) &= (q_e^* q_{e-1}^* q_{e-2}^* \dots q_s^*) \end{aligned} \quad (3.47)$$

Second Level: Word Model Decoding

1. Initialization: for $1 \leq l \leq L$, $1 \leq t \leq T$:

$$\hat{\delta}_t(l) = \begin{cases} \chi(1, t) & \text{if } l = 1 \\ \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1) \chi(s, t)] & \text{if } 2 \leq l \leq L \end{cases} \quad (3.48)$$

$$\hat{\omega}_t(l) = \begin{cases} 1 & \text{if } l = 1 \\ \arg \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1) \chi(s, t)] & \text{if } 2 \leq l \leq L \end{cases}$$

2. Termination: for $l = L$, $t = T$:

$$\begin{aligned} \hat{P}^* &= \hat{\delta}_T(L) \\ \hat{q}^*(L) &= \hat{\omega}_T(L) \end{aligned} \quad (3.49)$$

3. Character Backtracking: for $L - 1, L - 2, \dots, 1$:

$$\hat{q}^*(l) = \hat{\omega}_{\hat{q}^*(l+1)}(l) \quad (3.50)$$

4. State Backtracking: for $L - 1, L - 2, \dots, 1$:

$$\begin{aligned}\hat{\epsilon}_L &= \epsilon[T, \hat{q}^*(L)] \\ \hat{\epsilon}_l &= \epsilon[\hat{q}^*(l+1) - 1, \hat{q}^*(l)]\end{aligned}\tag{3.51}$$

The Fast Two-Level Decoding Algorithm for Transition Model with Presence of Null Transitions

In this section we present the *Fast Two-Level Decoding Algorithm* considering the transition model and the presence of null transitions. This algorithm was used in the experiments presented in Chapter 5.

First Level: Character Model Decoding

1. Initialization: for $1 \leq s \leq T + 1$, $i = 1$

$$\begin{aligned}\delta_s(1) &= 1 \\ \omega_s(1) &= 0 \\ \delta_s(j) &= \max_{1 \leq i \leq N} [\delta_s(i) a_{ij}^{\text{nb}}] \\ \omega_s(j) &= \arg \max_{1 \leq i \leq N} [\delta_s(i) a_{ij}^{\text{nb}}]\end{aligned}\tag{3.52}$$

2. Recursion ($s \leq t \leq T + 1$, $1 \leq j \leq N$):

$$\begin{aligned}\delta_t(j) &= \max \left\{ \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{\text{ot}-1}] ; \max_{1 \leq i \leq N} [\delta_t(i) a_{ij}^{\text{nb}}] \right\} \\ \omega_t(j) &= \begin{cases} \arg \max_{1 \leq i \leq N} [\delta_t(i) a_{ij}^{\text{nb}}] & \text{if } ij \text{ is null} \\ \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}^{\text{ot}-1}] & \text{otherwise} \end{cases} \\ \zeta_t(j) &= \begin{cases} 1 & \text{if } ij \text{ is null} \\ 0 & \text{otherwise} \end{cases}\end{aligned}\tag{3.53}$$

3. Termination: if $j = N$:

$$\begin{aligned} e &= t \\ \chi(s, e) &= \delta_t(-N) \\ q_e^* &= N \end{aligned} \tag{3.54}$$

4. Path Backtracking: for $t = e - 1, e - 2, \dots, s$, q_t^* is determined recursively by:

$$q_t^* = \omega_{t-1}(q_{t+1}^*) \tag{3.55}$$

However, a new variable that accounts for the presence of null transitions, denoted as q'_t is introduced. After computing q_t^* we have to check if the variable ζ indicates the presence of a null transition, that is, $\zeta_t(q_t^*) = 1$. In such a case, without decreasing the time index t , q'_t is computed as:

$$q'_t = \omega_t(q_t^*) \quad \text{if} \quad \zeta_t(q_t^*) = 1 \tag{3.56}$$

Following the presence of a null transition during the backtracking procedure, that is $\zeta_{t+1}(q_{t+1}^*) = 1$, the computation of q_t^* is modified slightly as follows (now with a decrease in the time index t):

$$q_t^* = \omega_{t+1}(q'_{t+1}) \quad \text{if} \quad \zeta_{t+1}(q_{t+1}^*) = 1 \tag{3.57}$$

So, the best state sequence denoted as $\varepsilon(s, e)$ will be given by:

$$\varepsilon(s, e) = (q_e^* q'_e q_{e-1}^* q'_{e-1} \dots q_{s+1}^* q'_{s+1} q_s^* q'_s) \tag{3.58}$$

Second Level: Word Model Decoding

1. Initialization: for $1 \leq l \leq L$, $1 \leq t \leq T$:

$$\hat{\delta}_t(l) = \begin{cases} \chi(1, t) & \text{if } l = 1 \\ \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1)\chi(s, t)] & \text{if } 2 \leq l \leq L \end{cases} \quad (3.59)$$

$$\hat{\omega}_t(l) = \begin{cases} 1 & \text{if } l = 1 \\ \arg \max_{1 \leq s \leq T} [\hat{\delta}_s(l-1)\chi(s, t)] & \text{if } 2 \leq l \leq L \end{cases}$$

2. Termination: for $l = L$, $t = T$:

$$\begin{aligned} \hat{P}^* &= \hat{\delta}_T(L) \\ \hat{q}^*(L) &= \hat{\omega}_T(L) \end{aligned} \quad (3.60)$$

3. Character Backtracking: for $L-1, L-2, \dots, 1$:

$$\hat{q}^*(l) = \hat{\omega}_{\hat{q}^*(l+1)}(l) \quad (3.61)$$

4. State Backtracking: for $L-1, L-2, \dots, 1$:

$$\begin{aligned} \hat{\epsilon}_L &= \epsilon[T, \hat{q}^*(L)] \\ \hat{\epsilon}_l &= \epsilon[\hat{q}^*(l+1) - 1, \hat{q}^*(l)] \end{aligned} \quad (3.62)$$

BIBLIOGRAPHY

- [1] N. Arica and F. Yarman-Vural. An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*. 31(2):216-233, 2001.
- [2] E. Augustin, O. Baret, D. Price, and S. Knerr. Legal amount recognition on french bank checks using a neural network-hidden markov model hybrid. In *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pages 45-54, Taejon, Korea, 1998.
- [3] S. Austin, G. Zavaliagkos, J. Makhoul, and R. Schwartz. Speech recognition using segmental neural nets. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 625-628, San Francisco, USA, 1992.
- [4] I. Bazzi, R. Schwartz, and J. Makhoul. An omnifont open-vocabulary ocr system for english and arabic. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6):495-504, 1999.
- [5] B. Belkacem. Une application industrielle de reconnaissance d'adresses. In *Proc. 4eme Colloque National Sur l'Ecrit et le Document*, pages 93-100, Nantes, France, 1996.
- [6] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, USA, 1957.
- [7] R. Bippus and V. Margner. Script recognition using inhomogeneous p2dhmm and hierarchical search space reduction. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 773-776, Bangalore, India, 1999.
- [8] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford Univ. Press, Oxford, U.K., 1995.
- [9] M. Blumenstein and B. Verma. Neural-based solutions for the segmentation and recognition of difficult handwritten words from a benchmark database. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 281-284, Bangalore, India, 1999.
- [10] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. John Wiley & Sons, 1978.
- [11] R. M. Bozinovic and S. N. Srihari. Off-line cursive script word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63-84, 1989.

- [12] A. Brakensiek, J. Rottland, A. Kosmala, and G. Rigoll. Off-line handwriting recognition using various hybrid modeling techniques and character N-grams. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 343–352. Amsterdam, Netherlands, 2000.
- [13] A. Brakensiek, D. Willett, and G. Rigoll. Unlimited vocabulary script recognition using character n-grams. In *Proc. 22. DAGM-Symposium, Tagungsband Springer-Verlag, Kiel, Germany*, 2000.
- [14] A. S. Britto, R. Sabourin, F. Bortolozzi, and C. Y. Suen. A two-stage hmm-based system for recognizing handwritten numeral strings. In *Proc. 6rd International Conference on Document Analysis and Recognition*, pages 396–400, Seattle, USA, 2001.
- [15] H. Bunke, M. Roth, and E. G. Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden markov models. *Pattern Recognition*, 28(9):1399–1413, 1995.
- [16] C. J. C. Burges, J. I. Ben, J. S. Denker, Y. Lecun, and C. R. Nohl. Off-line recognition of handwritten postal words using neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):689–704, 1993.
- [17] J. Cai and Z. Q. Liu. Off-line unconstrained handwritten word recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(3):259–280, 1993.
- [18] F. Camastra and A. Vinciarelli. Cursive character recognition by learning vector quantization. *Pattern Recognition Letters*, 22:625–629, 2001.
- [19] R. G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.
- [20] D. Y. Chen, J. Mao, and K. Mohiuddin. An efficient algorithm for matching a lexicon with a segmentation graph. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 543–547, Bangalore, India, 1999.
- [21] J. K. Chen and F. K. Soong. An n-best candidates-based discriminante training for speech-recognition applications. *IEEE Transactions on Speech and Audio Processing*, 2(1):206–216, 1994.
- [22] M. Y. Chen, A. Kundu, and S. N. Srihari. Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE Transactions on Image Processing*, 4(12):1675–1688, 1995.

- [23] M. Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition using a hidden markov model type stochastic network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):481–496, 1994.
- [24] J. H. Chiang. A hybrid neural network model in handwritten word recognition. *Neural Networks*, 11:337–346, 1998.
- [25] Y. C. Chim, A. A. Kassim, and Y. Ibrahim. Dual classifier system for hand-printed alphanumeric character recognition. *Pattern Analysis and Applications*, 1:155–162, 1998.
- [26] S. J. Cho, J. Kim, and J. H. Kim. Verification of graphemes using neural networks in an hmm-based on-line korean handwriting recognition system. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 219–228, Amsterdam, Netherlands, 2000.
- [27] W. Cho and J. H. Kim. Off-line recognition of cursive words with network of hidden markov models. In *Proc. 4th International Workshop on the Frontiers of Handwriting Recognition*, pages 410–417, Taipei, Taiwan, 1994.
- [28] S. Connell. *Online Handwriting Recognition Using Multiple Pattern Class Models*. PhD thesis, Michigan State University, East Lansing, USA, May 2000.
- [29] J. G. A. Dolfin. *Handwriting Recognition and Verification - A Hidden Markov Approach*. PhD thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 1998.
- [30] J. Dong, A. Krzyzak, and C. Y. Suen. Local learning framework for recognition of lowercase handwritten characters. In *Proc. International Workshop on Machine Learning and Data Mining in Pattern Recognition*, page to appear, Leipzig, Germany, 2001.
- [31] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Machine Intelligence*, 7(4):705–719, 1993.
- [32] H. Duplantier. Interfaces de visualisation pour la reconnaissance d'Écriture. Technical report, École de Technologie Supérieure, Montreal, Canada, 1998. 28 pages.
- [33] G. Dzuba, A. Filatov, D. Gershuny, and I. Kill. Handwritten word recognition – the approach proved by practice. In *Proc. 6th International Workshop on Frontiers in Handwriting Recognition*, pages 99–111, Taejon, Korea, 1998.

- [34] A. El-Yacoubi. *Modélisation Markovienne de L'écriture Manuscrite Application à la Reconnaissance des Adresses Postales*. PhD thesis, Université de Rennes I, Rennes, France, 1996.
- [35] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Y. Suen. Unconstrained handwritten word recognition using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):752-760, 1999.
- [36] A. El-Yacoubi, R. Sabourin, M. Gilloux, and C. Suen. Off-line handwritten word recognition using hidden markov models. In L. C. Jain and B. Lazzerini, editors, *Knowledge-Based Intelligent Techniques in Character Recognition*, pages 191-229. CRC Press, 1999.
- [37] A. J. Elms, S. Procter, and J. Illingworth. The advantage of using an hmm-based approach for faxed word recognition. *International Journal on Document Analysis and Recognition*, 1:18-36, 1998.
- [38] C. Farouz. *Reconnaissance de Mots Manuscrits Hors-Ligne dans un Vocabulaire Ouvert par Modélisation Markovienne*. PhD thesis, Université de Nantes, Nantes, France, August 1999.
- [39] C. Farouz, M. Gilloux, and J. M. Bertille. Handwritten word recognition with contextual hidden markov models. In *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pages 133-142, Taejon, Korea, 1998.
- [40] J. T. Favata. Offline general handwritten word recognition using an approximate beam matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):1009-1021, 2001.
- [41] D. M. Ford and C. A. Higgins. A tree-based dictionary search technique and comparison with n-gram letter graph reduction. In R. Plamondon and G. Leedham, editors, *Computer Processing of Handwriting*, pages 291-312. World Scientific, 1990.
- [42] G. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268-278, 1973.
- [43] J. Freeman and D. Skapura. *Neural Networks - Algorithms, Applications and Programming Techniques*. Addison-Wesley, 1992.
- [44] H. C. Fu and Y. Y. Xu. Multilinguistic handwritten character recognition by bayesian decision-based neural networks. *IEEE Transactions on Signal Processing*, 46(10):2781-2789, 1998.

- [45] T. Fujisaki, K. Nathan, W. Cho, and H. Beigi. On-line unconstrained handwriting recognition by a probabilistic method. In *Proc. 3rd International Workshop on Frontiers in Handwriting Recognition*, pages 235–241. Buffalo, USA, 1993.
- [46] P. Gader, M. Whalen, M. Ganzberger, and D. Hepp. Handprinted word recognition on a NIST data set. *Machine Vision and Applications*, 8:31–41, 1995.
- [47] P. D. Gader, M. A. Mohamed, and J. H. Chiang. Handwritten word recognition with character and inter-character neural networks. *IEEE Transactions on Systems, Man and Cybernetics – Part B*, 27:158–164, 1994.
- [48] P. D. Gader, M. A. Mohamed, and J. M. Keller. Fusion of handwritten word classifiers. *Pattern Recognition Letters*, 17:577–584, 1996.
- [49] M. Gilloux. Hidden markov models in handwriting recognition. In *Fundamentals in Handwriting Recognition*, volume 126 of *NATO ASI Series*. pages 264–288. Springer Verlag, France, 1994.
- [50] M. Gilloux. Real-time handwritten word recognition within large lexicons. In *Proc. 5th International Workshop on Frontiers in Handwriting Recognition*, pages 301–304, Essex, UK, 1996.
- [51] M. Gilloux. Réduction dynamique du lexique par la méthode tabou. In *Proc. Colloque International Francophone sur l’Ecrit et le Document*, pages 24–31, Quebec, Canada, 1998.
- [52] N. D. Gorsky. Experiments with handwriting recognition using holographic representation of line images. *Pattern Recognition Letters*, 15(9):853–859, 1994.
- [53] V. Govindaraju and R. K. Krishnamurthy. Holistic handwritten word recognition using temporal features derived from off-line images. *Pattern Recognition Letters*, 17(5):537–540, 1996.
- [54] F. Grandidier. Analyse de l’ensemble des confusions d’un système. Technical report, École de Technologie Supérieure, Montreal, 2000.
- [55] F. Grandidier, R. Sabourin, C. Y. Suen, and M. Gilloux. A new strategy for improving feature sets in a discrete hmm-based handwriting recognition system. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 113–122, Amsterdam, Netherlands, 2000.
- [56] D. Guillevic, D. Nishiwaki, and K. Yamada. Word lexicon reduction by character spotting. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 373–382, Amsterdam, Netherlands, 2000.

- [57] D. Guillevic and C. Suen. HMM-kNN word recognition engine for bank cheque processing. In *Proc. International Conference on Pattern Recognition*, pages 1526–1529, Brisbane, Australia, 1998.
- [58] D. Guillevic and C. Y. Suen. Cursive script recognition: A sentence level recognition scheme. In *Proc. 4th International Workshop on the Frontiers of Handwriting Recognition*, pages 216–223, Taipei, Taiwan, 1994.
- [59] D. Guillevic and C. Y. Suen. Cursive script recognition applied to the processing of bank cheques. In *Proc. 3rd International Conference on Document Analysis and Recognition*, pages 11–14, Montreal, Canada, 1995.
- [60] T. M. Ha and H. Bunke. Off-line handwritten numeral recognition by perturbation method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):535–539, 1997.
- [61] K. Hanazawa, Y. Minami, and S. Furui. An efficient search method for large-vocabulary continuous-speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 1787–1790, Munich, Germany, 1997.
- [62] S. Haykin. *Neural Networks: A Comprehensive Foundation*. IEEE Computer Society Press, 1994.
- [63] A. Hennig and N. Sherkat. Cursive script recognition using wildcards and multiple experts. *Pattern Analysis and Applications*, 4(1):51–60, 2001.
- [64] A. Hennig, N. Sherkat, and R. J. Whitrow. Recognising letters in on-line handwriting with hierarchical fuzzy inference. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 936–940, Ulm, Germany, 1997.
- [65] L. Heutte, T. Paquet, J. V. Moreau, Y. Lecourtier, and C. Olivier. A structural/statistical feature based vector for handwritten character recognition. *Pattern Recognition Letters*, 19:629–641, 1998.
- [66] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press, 1990.
- [67] S. Jaeger, S. Manke, J. Reichert, and A. Waibel. Online handwriting recognition: The npen++ recognizer. *International Journal on Document Analysis and Recognition*, 3:169–180, 2001.
- [68] S. Jaeger, S. Manke, and A. Waibel. Npen++: An on-line handwriting recognition system. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 249–260, Amsterdam, Netherlands, 2000.

- [69] A. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 19(2):153–158, 1997.
- [70] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 22(1):4–38, 2000.
- [71] A. Kaltenmeier, T. Caesar, J. M. Gloger, and E. Mandler. Sophisticated topology of hidden markov models for cursive script recognition. In *Proc. International Conference on Document Analysis and Recognition*, pages 139–142, Tsukuba, Japan, 1993.
- [72] N. Kato, M. Suzuki, S. Omachi, H. Aso, and Y. Nemoto. Handwritten character recognition system using directional element feature and asymmetric mahalanobis distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 21(3):258–262, 1999.
- [73] G. Kaufmann, H. Bunke, and M. Hadorn. Lexicon reduction in an hmm-framework based on quantized feature vectors. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 1097–1101, Ulm, Germany, 1997.
- [74] P. Kenny, R. Hollan, V. N. Gupta, M. Lennig, P. Mermelstein, and D. O’Shaughnessy. A* admissible heuristics for rapid lexical access. *IEEE Transactions on Speech and Audio Processing*, 1(1):49–58, 1993.
- [75] G. Kim and V. Govindaraju. Bankcheck recognition using cross validation between legal and courtesy amounts. In S. Impedovo, P. S. P. Wang, and H. Bunke, editors, *International Journal of Pattern Recognition and Artificial Intelligence*, pages 657–673. World Scientific, 1997.
- [76] G. Kim and V. Govindaraju. A lexicon driven approach to handwritten word recognition for real-time applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):366–379, 1997.
- [77] G. Kim, V. Govindaraju, and S. N. Srihari. An architecture for handwriting text recognition systems. *International Journal on Document Analysis and Recognition*, 2:37–44, 1999.
- [78] G. Kim and S. Kim. Feature selection using genetic algorithms for handwritten character recognition. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 103–112. Amsterdam, Netherlands, 2000.

- [79] J. H. Kim, K. K. Kim, and C. Y. Suen. Hybrid schemes of homogeneous and heterogeneous classifiers for cursive word recognition. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 433–442. Amsterdam, Netherlands, 2000.
- [80] W. S. Kim and E. H. Park. Off-line recognition of handwritten korean and alphanumeric characters using hidden markov models. *Pattern Recognition*, 29(5):845–858, 1996.
- [81] F. Kimura, N. Kayahara, Y. Miyake, and M. Shridhar. Machine and human recognition of segmented characters from handwritten words. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 866–869. Ulm, Germany, 1997.
- [82] F. Kimura, M. Shridhar, and Z. Chen. Improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words. In *Proc. International Conference on Document Analysis and Recognition*, pages 18–22. Tsukuba, Japan, 1993.
- [83] S. Knerr and E. Augustin. A neural network-hidden markov model hybrid for cursive word recognition. In *Proc. 14th International Conference on Pattern Recognition*, pages 1518–1520, Brisbane, Australia, 1998.
- [84] A. L. Koerich, Y. Leydier, R. Sabourin, and C. Y. Suen. A hybrid large vocabulary handwritten word recognition system using neural networks with hidden markov models. In *Proc. 8th International Workshop on Frontiers in Handwriting Recognition*, pages 99–104. Niagara-on-the-Lake, Canada, 2002.
- [85] A. L. Koerich, R. Sabourin, and C. Y. Suen. A distributed scheme for lexicon-driven handwritten word recognition and its application to large vocabulary problems. In *Proc. 6th International Conference on Document Analysis and Recognition*, pages 660–664, Seattle, USA, 2001.
- [86] A. L. Koerich, R. Sabourin, and C. Y. Suen. A time-length constrained level building algorithm for large vocabulary handwritten word recognition. In *Proc. 2nd International Conference on Advances in Pattern Recognition*, pages 127–136, Rio de Janeiro, Brazil, 2001.
- [87] A. L. Koerich, R. Sabourin, and C. Y. Suen. Fast two-level viterbi search algorithm for unconstrained handwriting recognition. In *Proc. 27th International Conference on Acoustics, Speech, and Signal Processing*, pages 3537–3540, Orlando, USA, 2002.
- [88] A. L. Koerich, R. Sabourin, and C. Y. Suen. Large vocabulary off-line handwriting recognition: A survey. *Pattern Analysis and Applications*, 2002. In press. 24 pages.

- [89] A. L. Koerich, R. Sabourin, and C. Y. Suen. Lexicon-driven hmm decoding for large vocabulary handwriting recognition with multiple character models. *International Journal on Document Analysis and Recognition*, 2002. Submitted. 40 pages.
- [90] A. L. Koerich, R. Sabourin, C. Y. Suen, and A. El-Yacoubi. A syntax-directed level building algorithm for large vocabulary handwritten word recognition. In *Proc. 4th International Workshop on Document Analysis Systems*, pages 255–266. Rio de Janeiro, Brazil, 2000.
- [91] A. Kornai. An experimental hmm-based postal ocr system. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 3177–3180. Munich, Germany, 1997.
- [92] M. Kudo and J. Sklansky. Comparison of algorithms that select features for pattern recognition. *Pattern Recognition*, 33:25–41, 2000.
- [93] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Algorithm Design and Analysis*. Benjamin Cummings, Redwood City - USA, 1994.
- [94] T. Y. Kwok and M. P. Perrone. Adaptive n-best list handwritten word recognition. In *Proc. 6th International Conference on Document Analysis and Recognition*, pages 168–172. Seattle, USA, 2001.
- [95] B. Lazzerini and F. Marcelloni. A linguistic fuzzy recogniser of off-line handwritten characters. *Pattern Recognition Letters*, 21:319–327, 2000.
- [96] E. Lecolinet and O. Baret. Cursive word recognition: Methods and strategies. In *Fundamentals in Handwriting Recognition*, volume 126 of *NATO ASI Series*, pages 235–263. Springer Verlag, France, 1994.
- [97] Y. LeCun, O. Matan, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten zip code recognition with multilayer networks. In *Proc. 10th International Conference on Pattern Recognition*, pages 35–40, Atlantic City, USA, 1990.
- [98] C. K. Lee and G. Leedham. Rapid analytical verification of handwritten alphanumeric address fields. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 571–576. Amsterdam, Netherlands, 2000.
- [99] L. Lee, M. Lizarraga, N. Gomes, and A. Koerich. A prototype for Brazilian bankcheck recognition. In S. Impedovo, P. S. P. Wang, and H. Bunke, editors, *Automatic Bankcheck Processing*, pages 549–569. World Scientific, 1997.

- [100] S. W. Lee. Off-line recognition of totally unconstrained handwritten numeral strings using multilayer neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):648–652, 1996.
- [101] A. Leroy. Lexicon reduction based on global features for on-line recognition. In *Proc. 4th International Workshop on Frontiers in Handwriting Recognition*, pages 431–440. Taipei, ROC, 1994.
- [102] A. Leroy. Progressive lexicon reduction for on-line handwriting. In *Proc. 5th International Workshop on Frontiers in Handwriting Recognition*, pages 399–404. Essex, UK, 1996.
- [103] Y. Leydier. Reconnaissance de caracteres manuscrits grace a des reseaux neuronaux. Technical report, École de Technologie Supérieure, Montreal, Canada. August 2001. 20 pages.
- [104] A. Lifchitz and F. Maire. A fast lexically constrained viterbi algorithm for on-line handwriting recognition. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 313–322. Amsterdam, Netherlands, 2000.
- [105] C. L. Liu, H. Sako, and H. Fujisawa. Performance evaluation of pattern classifiers for handwritten character recognition. *International Journal on Document Analysis and Recognition*, 4:191–204, 2002.
- [106] Y. Lu and M. Shridhar. Character segmentation in handwritten words - an overview. *Pattern Recognition*, 29(1):77–96, 1996.
- [107] S. Madhvanath and V. Govindaraju. Holistic lexicon reduction. In *Proc. 3th International Workshop on Frontiers in Handwriting Recognition*, pages 71–78. Buffalo, USA, 1993.
- [108] S. Madhvanath and V. Govindaraju. Holistic lexicon reduction for handwritten word recognition. In *Proc. SPIE - Document Recognition III*, pages 224–234. San Jose, USA, 1996.
- [109] S. Madhvanath and V. Govindaraju. The role of holistic paradigms in handwritten word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):149–164, 2001.
- [110] S. Madhvanath, E. Kleinberg, and V. Govindaraju. Holistic verification of handwritten phrases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1344–1356, 1999.
- [111] S. Madhvanath, E. Kleinberg, V. Govindaraju, and S. N. Srihari. The hover system for rapid holistic verification of off-line handwritten phrases. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 855–859. Ulm, Germany, 1997.

- [112] S. Madhvanath and V. Krpasundar. Pruning large lexicons using generalized word shape descriptors. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 552–555, Ulm, Germany, 1997.
- [113] S. Madhvanath, V. Krpasundar, and V. Govindaraju. Syntactic methodology of pruning large lexicons in cursive script recognition. *Pattern Recognition*, 34:37–46, 2001.
- [114] S. Madhvanath and S. N. Srihari. Effective reduction of large lexicons for recognition of offline cursive script. In *Proc. 5th International Workshop on Frontiers in Handwriting Recognition*, pages 189–194, Essex, UK, 1996.
- [115] U. Mahadevan and S. N. Srihari. Parsing and recognition of city, state, and zip codes in handwritten addresses. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 325–328, Bangalore, India, 1999.
- [116] D. Mangalagiu. *Accélération de l'algorithme des K Plus Proches Voisins par Réorganisation*. PhD thesis, École Polytechnique, Paris, France, 1999.
- [117] S. Manke, M. Finke, and A. Waibel. A fast search technique for large vocabulary on-line handwriting recognition. In A. C. Downton and S. Impedovo, editors, *Progress in Handwriting Recognition*, pages 437–444, World Scientific, Singapore, 1996.
- [118] U. Marti and H. Bunke. Towards general cursive script recognition. In *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pages 379–388, Taejon, Korea, 1998.
- [119] U. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 705–708, Bangalore, India, 1999.
- [120] U. Marti and H. Bunke. Handwritten sentence recognition. In *Proc. 15th International Conference on Pattern Recognition*, pages 467–470, Barcelona, Spain, 2000.
- [121] T. Matsui, T. Tsutsumida, and S. N. Srihari. Combination of stroke/background structure and contour-direction features and handprinted alphanumeric recognition. In *Proc. 4th International Workshop on Frontiers in Handwriting Recognition*, pages 87–96, Taipei, Taiwan, 1994.
- [122] G. Menier and G. Lorette. Lexical analyzer based on a self-organizing feature map. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 1067–1071, Ulm, Germany, 1997.

- [123] M. A. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden markov modeling and segmentation-based dynamic programming technique. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):548-554, 1996.
- [124] M. A. Mohamed and P. Gader. Generalized hidden markov models – part ii: Application to handwritten word recognition. *IEEE Transactions on Fuzzy Systems*, 8:82-94, 2000.
- [125] C. S. Myers and L. R. Rabiner. Connected digit recognition using a level building dtw algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29(3):351-363, June 1981.
- [126] C. S. Myers and L. R. Rabiner. A level building dynamic time warping algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29(2):284-297, April 1981.
- [127] H. Ney. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(3):263-271, 1984.
- [128] H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16(5):64-83, 1999.
- [129] I. S. Oh and C. Y. Suen. Distance features for neural network-based recognition of handwritten characters. *International Journal on Document Analysis and Recognition*, 1(2):73-88, 1998.
- [130] L. S. Oliveira, N. Benahmed, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Feature subset selection using genetic algorithms for handwritten digit recognition. In *Proc. 14th Brazilian Symposium on Computer Graphics and Image Processing*, pages 362-369, Florianopolis, Brazil, 2001.
- [131] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. A modular system to recognize numerical amounts on brazilian bank cheques. In *Proc. 6th International Conference on Document Analysis and Recognition*, pages 389-394, Seattle, USA, 2001.
- [132] D. Ollivier, M. Weinfeld, and R. Guegan. Combining different classifiers and level of knowledge: A first step towards an adaptive recognition system. In *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pages 89-98, Taejon, Korea, 1998.
- [133] H. S. Park and S. W. Lee. Off-line recognition of large-set handwritten characters with multiple hidden markov models. *Pattern Recognition*, 29(2):231-244, 1996.

- [134] P. Pedrazzi and A. M. Colla. Simple feature extraction for handwritten character recognition. In *Proc. International Conference on Image Processing*, pages 320–323, Washington, USA, 1995.
- [135] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68–89, 2000.
- [136] R. K. Powalka, N. Sherkat, and R. J. Whitrow. Word shape analysis for a hybrid recognition system. *Pattern Recognition*, 30(3):412–445, 1997.
- [137] S. Procter and J. Illingworth. Handwriting recognition using hmms and a conservative level building algorithm. In *Proc. 7th International Conference on Image Processing and its Applications*, pages 736–739, Manchester, UK, 1999.
- [138] S. Procter, J. Illingworth, and F. Mokhtarian. Cursive handwriting recognition using hidden markov models and a lexicon-driven level building algorithm. *IEE Proceedings on Vision, Image and Signal Processing*, 147(4):332–339, 2000.
- [139] L. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.
- [140] L. Rabiner, C. H. Lee, B. H. Juang, and J. G. Wilpon. Hmm clustering for connected word recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 405–408, Glasgow, England, 1989.
- [141] L. R. Rabiner and S. E. Levinson. A speaker-independent, syntax-directed, connected word recognition system based on hidden markov models and level building. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(3):561–573, 1985.
- [142] E. H. Ratzlaff, K. S. Nathan, and H. Maruyama. Search issues in the IBM large vocabulary unconstrained handwriting recognizer. In *Proc. 5th International Workshop on Frontiers in Handwriting Recognition*, pages 177–182, Essex, UK, 1996.
- [143] S. Renals and M. M. Hochberg. Start-synchronous search for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 7(5):542–553, 1999.
- [144] M. D. Richard and R. P. Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

- [145] T. Robinson and J. Christie. Time-first search for large vocabulary speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 829–832. Seattle, USA, 1998.
- [146] H. Sakoe. Two-level dp-matching — a dynamic programming-based pattern matching algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 27(6):588–595, 1979.
- [147] G. Saon. *Modeles Markoviens Uni et Bidimensionnels pour la Reconnaissance de l'Écriture Manuscrite Hors-Ligne*. PhD thesis, Université Henri Poincaré, Nancy, France, November 1997.
- [148] G. Saon. Cursive word recognition using a random field based hidden markov model. *International Journal on Document Analysis and Recognition*, 1:199–208, 1999.
- [149] C. Scagliola and G. Nicchiotti. Enhancing cursive word recognition performance by the integration of all the available information. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 363–372. Amsterdam, Netherlands, 2000.
- [150] H. Schwenk. *Amélioration des classifieurs neuronaux par incorporation de connaissances explicites: Application à la reconnaissance de caractères manuscrits*. PhD thesis, Université Pierre et Marie Curie, Paris, France, July 1996.
- [151] G. Seni and R. K. Srihari. A hierarchical approach to on-line script recognition using a large vocabulary. In *Proc. 4th International Workshop on Frontiers in Handwriting Recognition*, pages 472–479, Taipei, ROC, 1994.
- [152] G. Seni, R. K. Srihari, and N. Nasrabadi. Large vocabulary recognition of on-line handwritten cursive words. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):757–762, 1996.
- [153] A. Senior. *Off-Line Cursive Handwriting Recognition using Recurrent Neural Networks*. PhD thesis, University of Cambridge, Cambridge, England, September 1994.
- [154] A. W. Senior and A. J. Robinson. An off-line cursive handwriting recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):309–321, 1998.
- [155] M. Shridhar, G. Houle, and F. Kimura. Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 861–865, Ulm, Germany, 1997.

- [156] R. K. Srihari. Use of lexical and syntactic techniques in recognizing handwritten text. In *Proc. of ARPA Workshop on Human Language Technology*, pages 403–407. Princeton, USA, 1994.
- [157] S. Srihari. A survey of sequential combination of word recognizers in handwritten phrase recognition at cedar. In *Proc. 1st International Workshop on Multiple Classifier Systems*, pages 45–51. Cagliari, Italy, 2000.
- [158] S. Srihari, V. Govindaraju, and R. Srihari. Handwritten text recognition. In *Proc. 4th International Workshop on Frontiers in Handwriting Recognition*, pages 265–274. Taipei, Taiwan, 1994.
- [159] S. N. Srihari. Recognition of handwritten and machine–printed text for postal address interpretations. *Pattern Recognition Letters*, 14:291–302, 1993.
- [160] S. N. Srihari, Y.-C. Shin, V. Ramanaprasad, and D. S. Lee. A system to read names and addresses on tax forms. *Proceedings of IEEE*, 84(7):1038–1049, 1996.
- [161] T. Steinherz, E. Rivlin, and N. Intrator. Offline cursive script word recognition – a survey. *International Journal on Document Analysis and Recognition*, 2:90–110, 1999.
- [162] C. Y. Suen. N–gram statistics for natural language understanding and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):15–21, 1979.
- [163] H. Takahashi and T. D. Griffin. Recognition enhancement by linear tournament verification. In *Proc. International Conference on Document Analysis and Recognition*, pages 585–588, Tsukuba, Japan, 1993.
- [164] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of art in on–line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.
- [165] O. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition. *Pattern Recognition*, 29(4):641–662, 1996.
- [166] V. Vuori, J. Laaksonen, E. Oja, and J. Kangas. Speeding up on–line recognition of handwritten characters by pruning the prototype set. In *Proc. 6rd International Conference on Document Analysis and Recognition*, Seattle, USA, 2001.
- [167] L. Vuurpijl and L. Schomaker. Two–stage character classification: A combined approach of clustering and support vector classifiers. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 423–432, Amsterdam, Netherlands, 2000.

- [168] F. Wang, L. Vuurpijl, and L. Schomaker. Support vector machines for the classification of western handwritten capitals. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, pages 167–176, Amsterdam, Netherlands, 2000.
- [169] Z. Wimmer, B. Dorizzi, and P. Gallinari. Dictionary preselection in a neuro-markovian word recognition system. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 539–542, Bangalore, India, 1999.
- [170] L. Yaeger, R. Lyon, and B. Webb. Effective training of a neural network character classifier for word recognition. In *Proc. Advances in Neural Information Processing Systems*, pages 807–813, Seattle, USA, 1997.
- [171] L. S. Yaeger, B. J. Webb, and R. F. Lyon. Combining neural networks and context-driven search for on-line, printed handwriting recognition in the newton. *AI Magazine*, pages 73–89, 1998.
- [172] H. Yamada and Y. Nakano. Cursive handwritten word recognition using multiple segmentation determined by contour analysis. *IEICE Transactions on Information Systems*, 79(5):464–470, 1996.
- [173] S. Young. Large vocabulary continuous speech recognition. *IEEE Signal Processing Magazine*, 13(5):47–55, 1996.
- [174] G. Zavaliagkos, Y. Zhao, R. Schwartz, and J. Makhoul. A hybrid segmental neural net/hidden markov model system for continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 2(1):151–160, 1994.
- [175] S. D. Zenzo, M. del Bueono, M. Meucci, and A. Spirito. Optical recognition of handprinted characters of any size, position, and orientation. *IBM Journal of Research and Development*, 36(3):487–501, 1992.
- [176] B. Zhang, M. Fu, H. Yan, and M. A. Jabri. Handwritten digit recognition by adaptive-subspace self-organizing map (assom). *IEEE Transactions on Neural Networks*, 10(4):939–945, 1999.
- [177] G. P. Zhang. Neural networks for classification: A survey. *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 30(4):451–462, 2000.
- [178] J. Zhou. *Recognition and Verification of Unconstrained Handwritten Numeral*. PhD thesis, Concordia University, Montreal, Canada, November 1999.
- [179] M. Zimmermann and J. Mao. Lexicon reduction using key characters in cursive handwritten words. *Pattern Recognition Letters*, 20:1297–1304, 1999.