IJDAR (2003) 6: 126–144
Digital Object Identifier (DOI) 10.1007/s10032-003-0113-0

International Journal on Document Analysis and Recognition IJDAR

Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models

Alessandro L. Koerich[1], **Robert Sabourin**[2], **Ching Y. Suen**[3]

[1] PPGIA – CCET – PUCPR, Pontifical Catholic University of Paraná, Rua Imaculada Conceição, 1155, 80.215-901, Curitiba (PR) Brazil
[2] Département de Génie de la Production Automatisée (GPA), École de Technologie Supérieure (ETS), 1100, rue Notre-Dame Ouest, H3C 1K3, Montreal, QC, Canada
[3] Centre for Pattern Recognition and Machine Intelligence (CENPARMI), Concordia University, 1455, Maisonneuve Blvd. West, Suite GM606, H3G 1M8, Montreal, QC, Canada

Received: 8 July 2002 / Accepted: 1 July 2003
Published online: September 12, 2003 – © Springer-Verlag 2003

Abstract. This paper presents a handwriting recognition system that deals with unconstrained handwriting and large vocabularies. The system is based on the segmentation-recognition paradigm where words are first loosely segmented into characters or pseudocharacters and the final segmentation is obtained during the recognition process, which is carried out with a lexicon. Characters are modeled by multiple hidden Markov models (HMMs), which are concatenated to build up word models. The lexicon is organized as a tree structure, and during the decoding words with similar prefixes share the same computation steps. To avoid an explosion of the search space due to the presence of multiple character models, a lexicon-driven level building algorithm (LDLBA) is used to decode the lexical tree and to choose at each level the more likely models. Bigram probabilities related to the variation of writing styles within the words are inserted between the levels of the LDLBA to improve the recognition accuracy. To further speed up the recognition process, some constraints are added to limit the search efforts to the more likely parts of the search space. Experimental results on a dataset of 4674 unconstrained words show that the proposed recognition system achieves recognition rates from 98% for a 10-word vocabulary to 71% for a 30,000-word vocabulary and recognition times from 9 ms to 18.4 s, respectively.

Keywords: Handwriting recognition – Large vocabulary – Level building algorithm – Hidden Markov models – Search strategies

Correspondence to: Alessandro L. Koerich
(e-mail: alekoe@ppgia.pucpr.br)

1 Introduction

Offline recognition of handwritten words is a challenging task due to the high variability and uncertainty of handwriting. Several proposals to solve this problem have been presented recently [2,5,6,13,17,23,25]. However, handwriting recognition has success in only very constrained contexts. 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 (or writer-dependent) recognition

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 and recognition of words in small lexicons. 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.

Despite this, 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 to 1000 words [5,6, 13,24,35,40]. Such a restricted vocabulary is usually integrated into the recognition engine to discard as soon as possible the unlikely word hypotheses. The use of a restricted vocabulary, also called lexicon, reduces some problems related to the segmentation of words into characters [17,25,41] and also helps to disambiguate single characters by looking at the entire context.

The recognition process can be viewed as a search problem: given a sequence of features extracted from the input image, find a word from the lexicon that best matches such a sequence. Generally, this search prob-

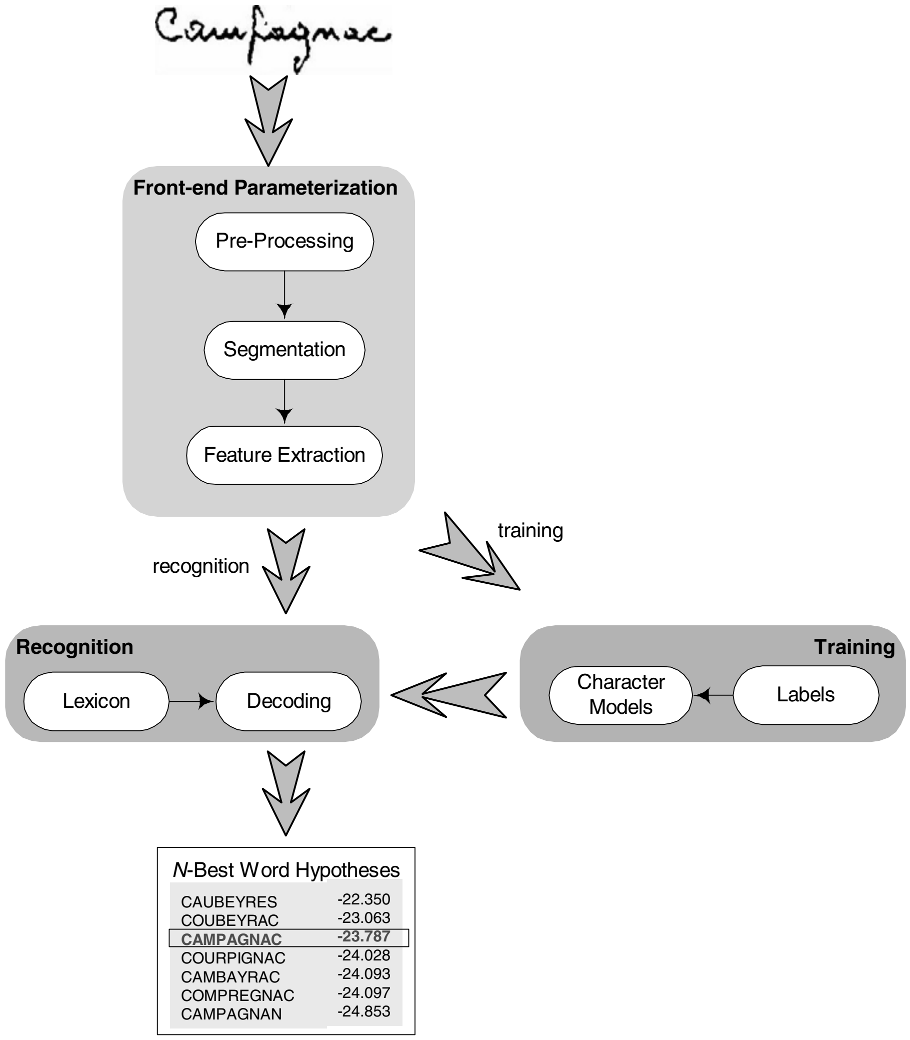


Fig. 1. An overview of the handwriting recognition system

lem has been tackled using dynamic programming techniques [6,13,40], but as the number of words in the lexicon grows, recognition becomes more and more difficult due to two main reasons: it is more probable to have similar words; more words need to be matched against the sequence of features [40]. The former results in more confusions to the recognition engine, so the recognition accuracy decreases as the number of words in the lexicon increases. The latter results in higher recognition times since more word hypotheses have to be decoded. For these reasons, there are relatively few studies with vocabularies comprising more than 1000 words. A very common approach is to use pruning methods prior to recognition to reduce the lexicon size to tens or hundreds of words [16,18,20,22,30,39,43]. As a result, the recognition time aspect can be negligible. These pruning methods are based on the characteristics of the application environment, such as the ZIP code in postal envelopes [13], the courtesy amount in bankchecks [19], on characteristics of the word to be recognized, such as structural elements extracted from the words, e.g., upstrokes and downstrokes [30,39], key characters [20,43], estimation of word length [22,43], or on some measure of similarity of the words in the lexicon [16,18]. Nevertheless, the problem is that all of these pruning methods rely on heuristics, and the number of word hypotheses is reduced at the expense of accuracy. Moreover, most of these pruning methods are not capable of dealing with unconstrained handwriting but only with certain types of writing styles: cursive [30,39,43] or handprinted [20]. Some methods also involve the extraction of other features from the input image rather than those used during the recognition process [20,30,43] and may introduce delays and increase the computational complexity of the overall recognition process. Besides these pruning methods, some search techniques that constrain the search space such as A*, beam search, and multipass have also been used in handwriting recognition [2,9,21,38]. A survey on these search methods used in large vocabulary handwriting recognition is presented in [28].

Another very common constraint is the writing style. The large number of writing styles and the variability among them makes the problem of handwriting recognition even more challenging. If different ways of writing a character can be identified, then it is often possible to increase the recognition accuracy by modeling each subclass separately, creating multiple models per character class [8,12]. On the other hand, while multiple character models may improve the recognition accuracy, they may also increase the computational complexity of the recognition process.

In this paper, the problem of handwriting recognition given a large vocabulary and unconstrained handwriting is addressed. The basic problem we have to solve is this: Having multiple character class models to cope with unconstrained handwriting, how does one make use of these models and accommodate them into a lexicon-driven recognition approach where a large vocabulary is employed? In this paper, we focus our attention on the problem of matching a sequence of observations generated from high-level features extracted from words and statistical models of characters (HMMs) in an efficient manner.

The organization of this paper is as follows. Section 2 presents an overview of the handwriting recognition system. Section 3 presents an overview of the word recognition problem in a probabilistic framework. Section 4 shows some ways of organizing a lexicon for single and multiple character class models. Section 5 presents the search strategy based on the level-building algorithm, the use of bigram probabilities to improve the recognition accuracy, and the use of contextual information to constrain the search space. Experimental results are presented in Sect. 6. An analysis of the achieved results and a discussion are presented in the last section.

2 An overview of the handwriting recognition system

A wide variety of techniques are used to perform handwriting recognition. Figure 1 highlights the many components of the handwriting recognition system that was designed to deal with unconstrained handwriting (handprinted, cursive, and mixed styles), multiple writers (writer-independent), and dynamically generated lexicons. For these reasons it uses a segmentation-recognition strategy where handwritten words are loosely segmented (oversegmented) into subword units (characters or pseudocharacters). These subword units are modeled in a probabilistic framework by elementary HMMs. The Markovian modeling 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 a "super-HMM" created by concatenating character HMMs. The decoding algorithm finds the N-best word hypotheses that have the highest likelihood given the observation sequence.

In the following sections, we provide a brief description of the main components of the handwriting recognition system.

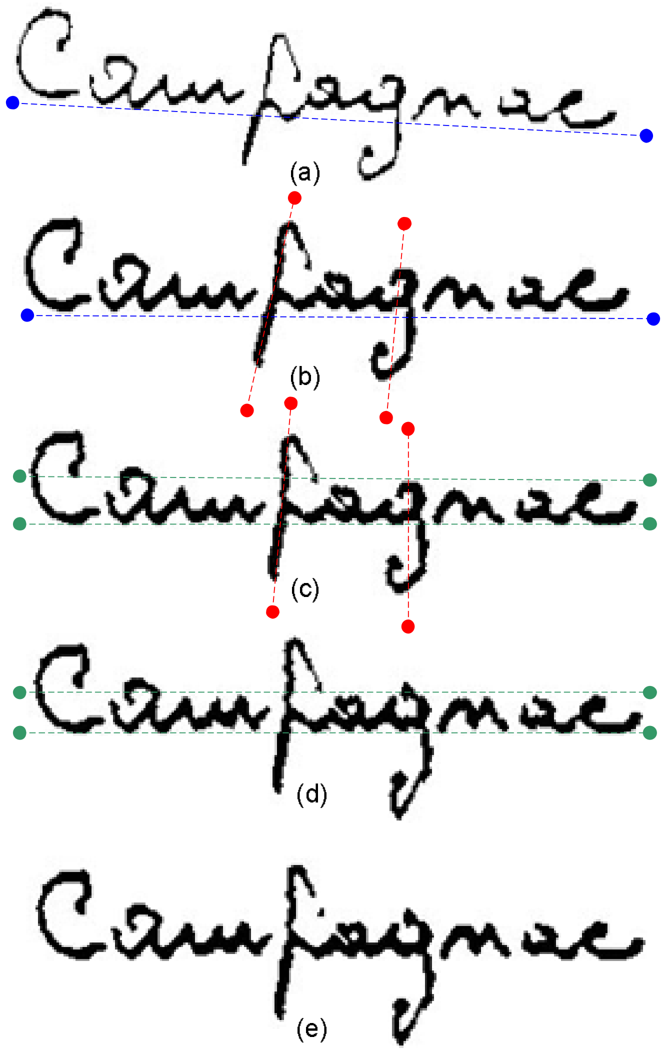


Fig. 2a–e. Preprocessing steps applied to a word image. **a** Original image. **b** Baseline slant normalization. **c** Character skew correction. **d** Lowercase character area normalization. **e** Final image after smoothing

2.1 Preprocessing

Preprocessing attempts to eliminate some variability related to the writing process that is not very significant from the point of view of recognition, such as variability due to the writing environment, writing style, acquisition, digitizing of images, etc. In addition, the preprocessing steps are also useful for holding the assumption of the Markovian modeling, which states that the observations in the sequence should be statistically independent once

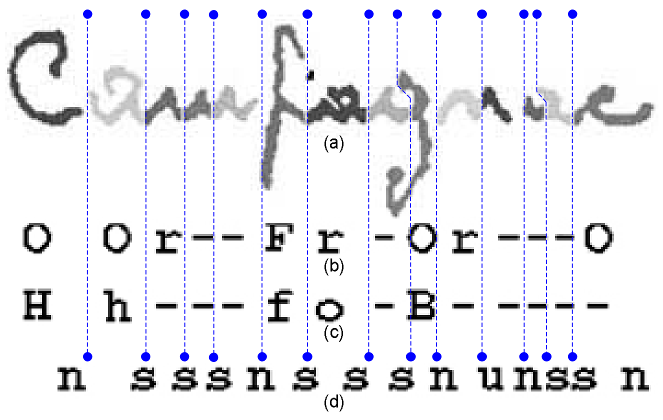

(a)
O O r - - F r - O r - - - O
(b)
H h - - - f o - B - - - - -
(c)
n s s s n s s s n u n s s n
(d)

Fig. 3. **a** Segmentation of a word into characters or pseudocharacters. **b** Sequence of global features. **c** Sequence of bidimensional contour transition histogram features. **d** Sequence of segmentation features

the underlying hidden state is known. Figure 2 illustrates the preprocessing steps, which include baseline slant normalization, lowercase character area normalization (for cursive words), character skew correction, and smoothing. More details about the preprocessing steps can be found in [11,13].

2.2 Segmentation of words into characters

Segmentation of words into basic units is necessary when dealing with dynamically generated lexicons of moderate size. The segmentation method performs an explicit segmentation 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. The algorithm mainly 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 3a illustrates the segmentation of a word into characters or pseudocharacters.

2.3 Feature extraction

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 pseudocharacter level because other information, such as context, word length, etc., are available and permit high discrimination of words. Thus, features at the grapheme level are considered with the aim of clustering letters into classes. A grapheme may consist of a full character, a fragment of a character, or more than one 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 (Fig. 3b). Each combination of these features within a segment is encoded by a distinct symbol, leading in this way to an alphabet of 27 symbols [11]. The second feature set is based on the analysis of the bidimensional contour transition histogram of each segment in the horizontal and vertical directions (Fig. 3c), which leads to a set of 14 symbols [11]. There are also five segmentation features that try to reflect the way segments are linked together (Fig. 3d). 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 the average segment width. 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.

2.4 Character and word models

Several HMM architectures can be considered for handwriting recognition. This stems from the fact that handwriting is certainly not a Markovian process and, even if it was 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.

In some applications, it is more convenient to produce observations by transitions rather than by states. The character models use discrete HMMs where observations are produced by transitions and transitions with no output are also incorporated into the model. In this case, the conventional definition of an HMM is modified and now the compact notation is $\lambda = \{A, A', \Pi\}$, where $A = \{a_{ij}^{z_g}\}$ is the probability distribution associated with transitions from state s_i to state s_j and at the same time producing observation symbol z_g, $A' = \{a_{ij}'^{\Phi}\}$ is the probability distribution associated with null transitions from state s_i to state s_j and at the same time producing null observation symbol Φ, and $\Pi = \{\pi_i\}$ is the initial state distribution.

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 ten-state left-right HMM as shown in Fig. 4a with three paths to take into account these configurations is adopted. Transition t_{1-10} emitting the null symbol models the letter omission case. Null transition t_{1-2} models

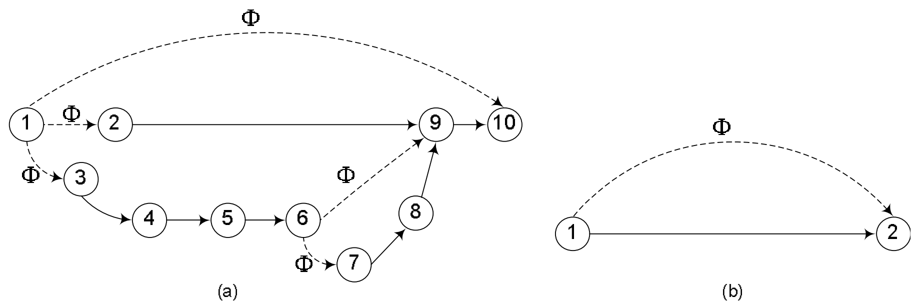


Fig. 4. a A left–right character HMM with 10 states and 12 transitions where 5 of them are null (*dotted line*). **b** The interword space model with 2 states and 2 transitions, where 1 is null [13]

the case of a correctly segmented character, while transitions 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 transitions t_{4-5}, t_{7-8}, and t_{9-10} model the nature of the segmentation points that gave rise to this oversegmentation. In addition, there is a special model for interword space in the case where the input image contains more than one word (compound word). This model simply consists of two states linked by two transitions, modeling a space or no space between a pair of words (Fig. 4b).

Considering a discrete symbol observation with G symbols, a character HMM represented by its compact notation $\lambda = \{A, A', \Pi\}$ and a recognition vocabulary represented by a lexicon $\mathcal{R}$ that contains V words in which the average length is L characters, word models, denoted as $\hat{\lambda}$, regarded as a "super-HMM" can be built by the concatenation of L subword HMMs, i.e.:

$$\hat{\lambda} = \lambda_1 \oplus \cdots \oplus \lambda_L \tag{1}$$

2.5 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' for all models 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 (Eq. 1), where L is the number of characters 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.

A variant of the Baum–Welch algorithm is used for training in which the segments produced by the segmentation algorithm need not be manually labeled [13, 14]. Since a sufficient learning 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.

While the preprocessing, segmentation, feature extraction, modeling, and training of characters are very important aspects in any recognition system, they have already been presented elsewhere [11,13]. Therefore, in the remainder of this paper we will focus our attention on the original aspects of the handwriting recognition system: the recognition approach that deals with multiple character models and a large vocabulary.

3 Recognition of unconstrained handwritten words

The basic problem in large vocabulary handwriting recognition is: given a handwritten word represented by a sequence of observations denoted as $O = (o_1 \dots o_t \dots o_T)$ in which T is the number of observations in the sequence and o_t represents the *t-th* symbol, and a recognition vocabulary represented by a set of words $\mathcal{R} = \{w_1, \dots, w_v, \dots, w_V\}$ in which V is the number of words and w_v is the *v-th* word that is formed by the concatenation of characters such that $w_v = (c_1 \dots c_l \dots c_L)$ in which L is the total number of characters that form a word and c_l represents the l-th character, find the word hypothesis that best represents the sequence of observations.

A 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.,

$$\hat{w} \ni P(\hat{w}|O) = \max_{w \in \mathcal{R}} P(w|O) \tag{2}$$

where $\hat{w}$ is the word with maximum a posteriori probability. 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 approximated by:

$$\hat{w} \approx \arg\max_{w \in \mathcal{R}} P(O|w) \tag{3}$$

While it may be possible to train full-word models for a small vocabulary of words, this is not feasible for larger vocabularies (> 100 words) due to the amount of data required to train each model. The way we compute

$P(O|w)$ for large vocabularies is to build statistical models for subword units (characters) in an HMM framework, build up word models from these subword models using a lexicon to describe the spelling of words, and then evaluate the model probabilities via standard concatenation methods.

Here we can make 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 (trellis or lattice) through which the best time alignment path has to be found. Recalling that characters are modeled by HMMs and word models are built by the concatenation of such models, the decoding rule of Eq. 3 can be rewritten as:

$$\hat{w} \approx \arg\max_{w \in \mathcal{R}} \{ \max_{Q} P(o_1 \ldots o_T, q_1 \ldots q_T | \lambda_1 \oplus \cdots \oplus \lambda_L) \} \tag{4}$$

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

Conventionally, given an observation sequence, all words in the lexicon are represented by "super-HMMs" and the Viterbi algorithm is used to decode each word separately, finding the best state sequence as well as the corresponding likelihood for each of them. When all words have been decoded, the words with the highest likelihoods are chosen as the best word hypotheses.

3.1 Complexity of handwriting recognition

One of the most important aspects of large vocabulary handwriting recognition is the computational complexity of the recognition process. This aspect is usually overlooked in applications that deal with smaller vocabularies. The complexity depends on the representation used for each of the elements as well as on the algorithm used for decoding. In describing the computational complexity of the recognition process, we are interested in the number of basic mathematical operations, denoted as $\mathcal{O}$. The computational complexity, denoted as $\mathcal{C}$, for a generic recognition process considering character HMMs and a conventional Viterbi algorithm can be approximated by:

$$\mathcal{C} = \mathcal{O}(TN^2LV) \tag{5}$$

This is a rough approximation considering that each character class is modeled by only one HMM. However, if handwriting is unconstrained, more than one HMM per character class is usually necessary because a single model is not enough to account for the high variability and ambiguity of handwriting [13].

Assuming now that each character class is modeled by two HMMs, one for uppercase and one for lowercase, and that each word is either entirely uppercase or lowercase, the computational complexity is now

$$\mathcal{C} = \mathcal{O}(2TN^2LV) \tag{6}$$

However, if we assume that words can be made up by the combination of uppercase and lowercase characters (unconstrained words), the computational complexity blows up exponentially as:

$$\mathcal{C} = \mathcal{O}(2^LTN^2V) \tag{7}$$

If we generalize for multiple models per character class, the computational complexity is still higher:

$$\mathcal{C} = \mathcal{O}(H^LTN^2V) \tag{8}$$

where H denotes the number of HMMs for each character class.

Therefore, recognition of unconstrained handwritten words given a large vocabulary and multiple character models to account for unconstrained writing styles is a challenging and computationally demanding problem that is clearly impractical to be carried out on today's computing platforms. The most common approaches attempt to limit either one or more of the variables involved in the recognition process or even to impose restrictions on the writing styles. A survey on the methods used in large vocabulary handwriting recognition is presented in [28].

4 Organization of the search space with multiple character class models

A single source of computational efficiency in performing searches is in organizing the HMMs to be searched as a character tree (Fig. 5b) instead of as a flat structure (Fig. 5a). A flat-structured lexicon is easy to implement and to integrate into an HMM framework, and it provides a good tradeoff in terms of computational complexity for small to medium lexicons [13]. 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 1200.[1] The extension to a 10,000-word lexicon increases the number of states linearly, and as a consequence the recognition time grows about 1000 times [26].

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 (Fig. 5b). Now, for the same 10-word lexicon approximately 900 HMM states have to be searched during the decoding.[2]

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 (> 1000 words)

[1] LNV, where L=12 is the average number of characters in the words of the lexicon, N=10 is the number of states of the character HMMs, and V=10 is the lexicon size.

[2] $(L - L_s)NV$, where L_s=3 is the average number of characters shared by the words of the lexicon.

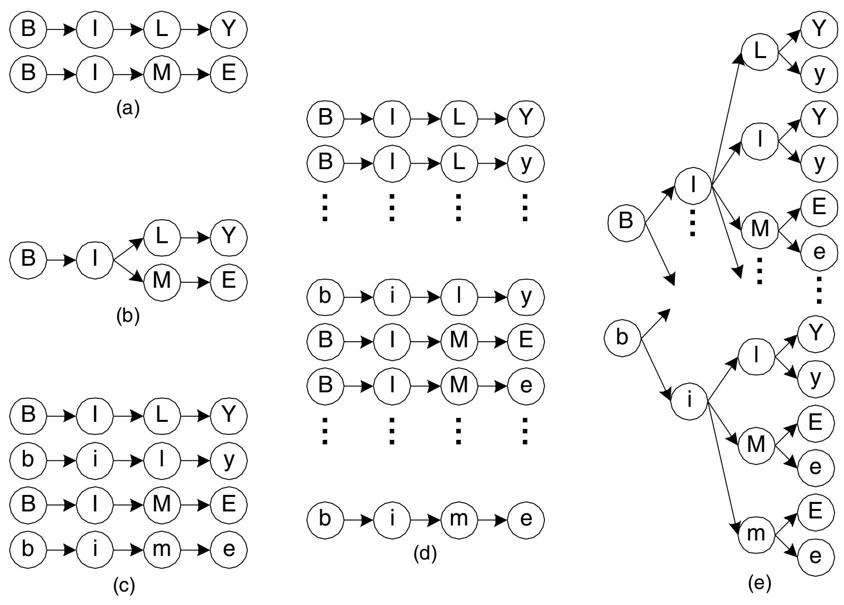


Fig. 5a–e. Two four-character words (*BILY* and *BIME*) represented as: **a** a flat structure, **b** a tree structure where the prefix *BI* is shared by both words, **c** a flat structure where each word is entirely uppercase or lowercase, **d** a flat structure where each character class (e.g., "M") has two models (an uppercase and a lowercase model) and all combinations of models within the words are allowed, **e** a tree representation of the two words considering all combinations of uppercase and lowercase models

Table 1. 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	Number of characters		Reduction
size	Flat lexicon	Lexical tree	ratio
10	119	113	1.05
100	1,198	987	1.21
1,000	11,998	8,361	1.43
10,000	120,035	66,558	1.80
30,000	360,012	173,631	2.07

it is more likely to have words with similar prefixes, so it is very interesting to use a tree-structured lexicon to avoid repeated computation of such prefixes. But notice that the number of shared characters depends strongly on the nature of the lexicon. Table 1 shows the reduction in the number of characters by representing lexicons of different sizes as tree structures. The expected speedup in the recognition process by representing the lexicon as a lexical tree is proportional to the *reduction ratio* shown in Table 1, which is not very significant. However the tree representation preserves the recognition accuracy.

4.1 Multiple character class models

In practice, it is not important to distinguish writing styles at the output of the recognition system, e.g., it does not matter if the word in Fig. 2 is recognized as "campagnac", "Campagnac", or "CAMPAGNAC" because all three representations of the word have the same meaning.[3] On the other hand, it would be very interesting to cope with different writing styles during the recognition process. However, identification of writing styles prior to recognition is also a difficult task that may introduce errors into the recognition process. One alternative has been to build recognizers capable of recognizing any writing style. In this case, the most straightforward approach is to have different models for uppercase and lowercase characters [13].

So far 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 for a single model to capture the high variability and ambiguity of a large number of writing styles and writers. 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 because it provides more accurate representation of the variants of handwriting [8,13,37]. On the other hand, while multiple character models may improve the recognition accuracy, they also may increase the computational complexity.

Assuming that each character class is now modeled by multiple HMMs (e.g., the character "M" is modeled by λ^m, $\lambda^{m'}$, λ^M, $\lambda^{M'}$, etc.) that may represent an uppercase letter model, a lowercase letter model, a letter model at the beginning or at the end of a word, etc. Usually we do not know which model we have to use during the recognition since the writing style is not know a priori. So we have to account for all possible combinations of uppercase and lowercase models during the decoding. Figure 5c shows the words of Fig. 5a considering that the writing style does not change within the word, while Fig. 5d shows the same words but now considering all combinations of models within the words. For the sake of simplicity, we assume that for each character class we have only two different models, one that models uppercase characters and another that models lowercase

[3] Some applications that carry out postprocessing of the recognition output may require correct identification of the writing style.

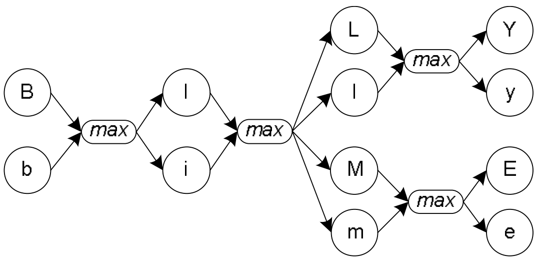


Fig. 6. A simplified representation of the lexical tree of Fig. 5d by using the maximum approximation

characters (e.g., the character "A" is modeled by λ^A and λ^a). Now, for a 10-word lexicon, approximately 4,096,000 HMM states[4] have to be decoded and the extension to a 10,000-word vocabulary blows up the number of HMMs to be searched. Even if we represent the lexicon by a tree structure as shown in Fig. 5e, 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 yet.

4.2 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 6 shows an approximated way to represent the lexical tree of Fig. 5e. In such a case, the number of HMM states to be searched can be approximated by $H(L-L_s+1)NV$, in which H is the number of different models for each character class. 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.

While the growth of the number of HMM states can be controlled by using the maximum approximation, it can cause pruning errors during the search and the accuracy may not be preserved. This occurs because we are making 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 may not be 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 in which to have a precise search is to evaluate each combination of character models independently, and that is computationally very expensive (Figs. 5d and 5e).

The criteria to select the best model at each tree level is based on the likelihood of each model. Thus, in the next section 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] $H^L NV$, where $H = 2$ is the number of different models for each character class, $L = 12$ is the average number of characters within the words in the lexicon, $N = 10$ is the number of states of the character HMMs, and $V = 10$ is the lexicon size.

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 as a consequence the advantages of avoiding repeated computation of common prefixes may become insignificant. The main problem is that each tree node can have multiple models associated with it, which results in a new tree branch for each additional model, that is, H^L search hypotheses for each word.

The difficulties can be overcome by using a level building algorithm (LBA) to find at each level the best model and to reduce the amount of computation at the subsequent levels. The LBA was introduced by Myers and Rabiner to recognize connected words from a small vocabulary and a highly constrained word syntax [32,33]. Since the vocabulary was small, they used whole words as the basic speech-recognition units. The LBA has also been used for the recognition of printed words [15], cursively handwritten words [34], and numerical strings [4]. However, the LBA has not been used to integrate multiple character class models, lexical tree search, and contextual information.

Given a recognition vocabulary $\mathcal{R} = \{R_1, \ldots, R_v, \ldots, R_V\}$ in which R_v is a reference word (Fig. 7a), a set of character models $\mathcal{M} = \{\lambda_1, \ldots, \lambda_m, \ldots, \lambda_M\}$ in which M is the number of models and λ_m is the m-th model that models character classes (letters, digits, and symbols) (Fig. 7c), and a sequence of observations $O = (o_1 \ldots o_t \ldots o_T)$ (Fig. 7e), word recognition means decoding O into a concatenation of L models $\lambda_1 \oplus \cdots \oplus \lambda_L$ (Fig. 7d). Since the concatenation of the characters is driven by the lexicon, we call this recognition scheme *lexicon-driven level building algorithm (LDLBA)*. The LDLBA matches the observation sequence to the multiple character models at each level, determining the maximum likelihood state sequence of the models. It jointly optimizes the segmentation of the sequence into subsequences produced by different models and the matching of the subsequences to particular models.

Although the LBA has been used with statistical models like HMMs [36], 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 trivial. Thus the complete procedure for finding the maximum likelihood state sequence of a word that is built up by the concatenation of character HMMs is presented as follows. We consider a lexicon-driven strategy where the lexicon is represented as a tree structure, bigram probabilities between the levels of the LBA, character HMMs with null transitions, observations emitted along transitions, and the logarithm of the model parameters (known as *likelihoods*). The parameters used to

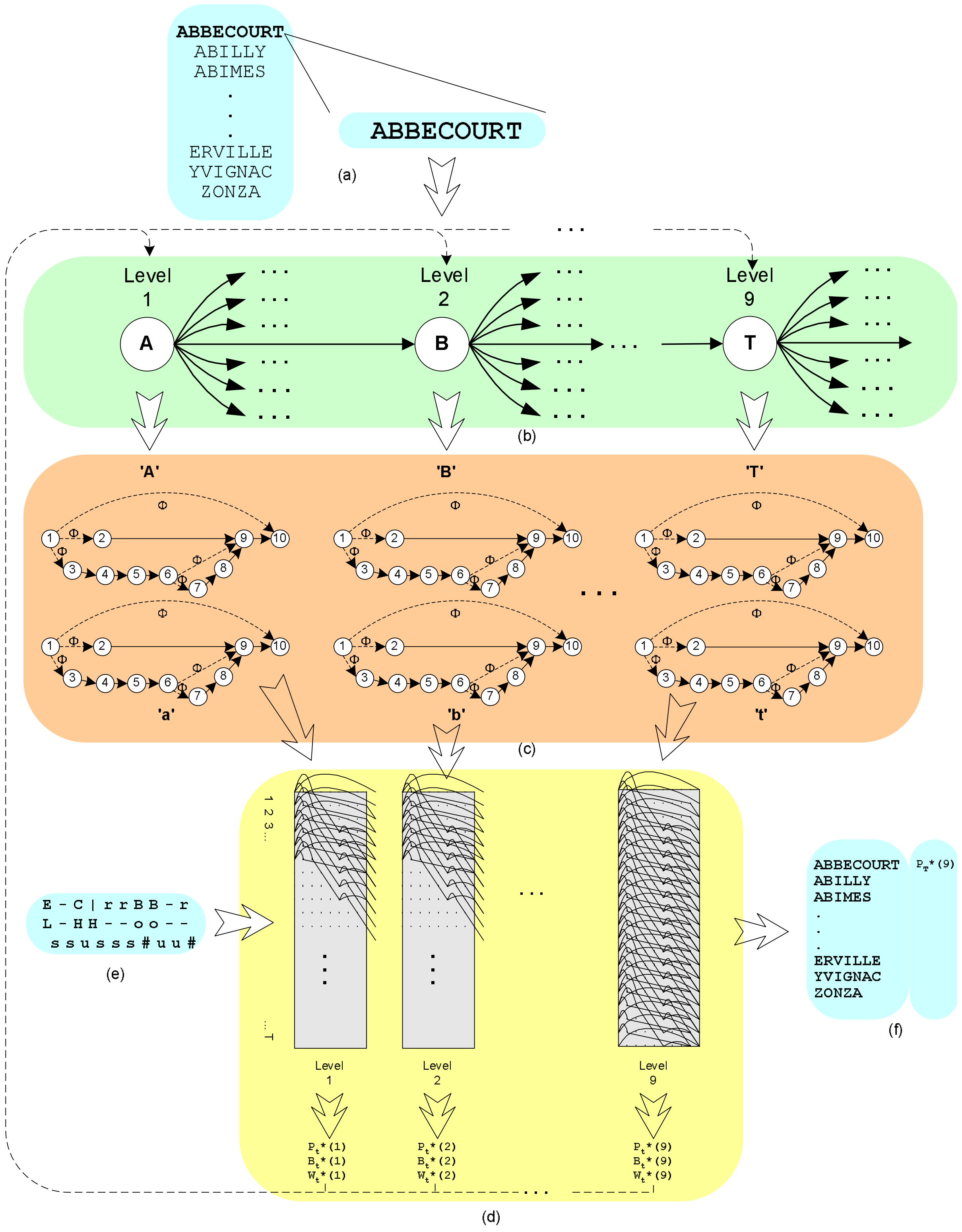


Fig. 7a–f. A simplified overview of the word decoding process. **a** A reference word taken from the lexicon. **b** The lexical tree formed by all reference words. **c** Character models concatenated according to the spelling of the reference word. **d** The LDLBA that matches the observation sequence and the word models generated by the concatenation of character models. **e** The input observation sequence. **f** The output of the word decoding process

Table 2. Definition of the elements of an HMM and other terms used in the LDLBA

Term	Definition
$A = \{a_{ij}\}$	State transition probability distribution in which a_{ij} denotes the probability of going from state s_i at time $t-1$ to state s_j at time t and producing an observation symbol $O_{t-1} = z_g$
$A' = \{a_{ij}\}$	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 and producing a null observation symbol Φ
B	Level backtrack pointer
G	Number of possible observation symbols
H	Number of models per character class
i,j	State index
l	Character (or level) index
m	Model index
N	Number of states in the model
P	Level output probability
q_t	State of the process at time t
$S = \{s_1, \ldots, s_N\}$	Set of possible states of the model
t	Observation index
T	Length of the observation sequence $O = (o_1 \ldots o_T)$
W	Level output model
$\mathcal{Z} = \{z_1, \ldots, z_G\}$	Discrete set of possible observation symbols
$\alpha_t(l, j)$	Observation frame in which a character ends
$\delta_t(l, j)$	Probability for each frame t, position l, and state j
λ	HMM model
τ	Bigram probability related to the writing style within a word

define an HMM as well as other terms used in the algorithm are shown in Table 2.

(1) *Initialization*: For a given model λ_m, $l = 1$, $t = 1$, and $j = 1$, we have:

$$\delta_t(l, j) = \tau(0, \lambda_m(l)) \tag{9}$$

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 LDLBA) and τ is the bigram probability related to the writing style (probability of starting a word with a given writing style) and is estimated on the training dataset according to the position of the characters within the words. However, the null transitions must be initialized also for $l = 1$, $t = 1$, and all states ($j = 2, \ldots, N$) as:

$$\delta_t(l, j) = \max_{1 \leq i < j} \left[\delta_t(l, i) + a'^{\Phi}_{ij} + \tau(0, \lambda_m(l))\right] \tag{10}$$

where a'^{Φ}_{ij} is the probability of passing from a state i at frame t to a state j at frame t and producing a null observation symbol Φ, and N is the number of states in the model.

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 $l = 2, \ldots, L$, $t = 1, \ldots, T$, and $j = 1$, we have:

$$\delta_t(l, j) = \delta_t(l-1, N) + \tau(\lambda_m(l-1), \lambda_m(l)) \tag{11}$$

where T is the length of the observation sequence, L is the number of concatenated characters that form a word, and τ is the bigram probability of keeping or changing the writing style within the word.

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 $l = 2, \ldots, L$, $t = 1$, and $j = 1$ we have:

$$\alpha_t(l, j) = 0 \ . \tag{12}$$

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

$$\alpha_t(l, j) = t \ . \tag{13}$$

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

$$\begin{aligned} \delta_t(l, j) = \max \Big\{ &\max_{1 \leq i < j} \left[\delta_{t-1}(l, i) + a^{O_{t-1}}_{ij}\right] ; \\ &\max_{1 \leq i < j} \left[\delta_t(l, i) + a'^{\Phi}_{ij}\right] \Big\} \end{aligned} \tag{14}$$

where $a^{O_{t-1}}_{ij}$ is the state transition probability distribution for which a_{ij} is the probability of passing from a state i at frame $t-1$ to a state j at frame t and producing an observation symbol $O_{t-1} = z_g$, where $z_g \in \mathcal{Z} = \{z_1, \ldots, z_g, \ldots, z_G\}$ 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 $l = 1, \ldots, L$, $t = 2, \ldots, T$, and $j = 2, \ldots, N$

as:

$$\alpha_t(l,j) = \begin{cases} \alpha_t\left(l, \arg\max\limits_{1\le i<j}\left[\delta_t(l,i) + a'^{\Phi}_{ij}\right]\right) & \text{if } ij \text{ is null} \\ \alpha_{t-1}\left(l, \arg\max\limits_{1\le i<j}\left[\delta_{t-1}(l,i) + a^{O_{t-1}}_{ij}\right]\right) & \text{otherwise} \end{cases} \tag{15}$$

For higher levels ($l > 1$), $\alpha_t(l,j)$ is also computed by Eq. 15. However, $\delta_t(l,j)$ is computed by Eq. 14 with a slight difference: now, only the states greater than 1 must be considered ($j = 2, \dots, N$) since for $j = 1$, $\delta_t(l,j)$ was already computed by Eq. 11.

(3) *Level termination*: For $l = 1$, $t = 1, \dots, T$ and a given model λ_m, we have:

$$\begin{aligned} P_t(l,\lambda_m) &= \delta_t(l,N) \\ B_t(l,\lambda_m) &= 0 \end{aligned} \tag{16}$$

For higher levels ($l = 2, \dots, L$), $P_t(l,\lambda_m)$ is also computed by Eq. 16, but $B_t(l,\lambda_m)$ now is given by:

$$B_t(l,\lambda_m) = \alpha_t(l,N) \tag{17}$$

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 ($m = 1, \dots, H$)(uppercase, lowercase, context-dependent, etc.) in the manner described above, that is, steps 1, 2, and 3 are repeated for each character class model.

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

$$P_t^*(l) = \max_m \left[P_t(l,\lambda_m)\right] \tag{18}$$

where P_t^* is the best level output probability.

The above equation searches the model λ_m at the 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:

$$B_t^*(l) = B_t\left(l, \arg\max_m \left[P_t(l,\lambda_m)\right]\right) \tag{19}$$

where B_t^* is the level output back pointer.

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

$$W_t^*(l) = \arg\max_m \left[P_t(l,\lambda_m)\right] \tag{20}$$

where W_t^* is the level output character indicator.

(5) *Word termination*: Since the words have a known length, at the end of the last level (L) the probability score for the word hypothesis is given as:

$$\hat{P}_T(L) = \max_m \left[P_T(L,\lambda_m)\right] \tag{21}$$

The level output back pointer for the word hypothesis is given by:

$$\hat{B}_T(L) = B_T\left(L, \arg\max_m \left[P_T(L,\lambda_m)\right]\right) \tag{22}$$

The output model λ_m for L that gives the highest likelihood score is given as:

$$\hat{W}_T(L) = \arg\max_m \left[P_T(L,\lambda_m)\right] \tag{23}$$

This is the decoding procedure for only one word from the lexicon, given an observation sequence. The same procedure is repeated for all other words in the lexicon. However, since the lexicon is organized as a tree structure, the computation of steps 1 to 4 can be avoided for words that have similar prefixes. Instead of steps 1 to 4, the results stored in the tree nodes (the arrays P_t^*, B_t^*, and W_t^*) are used and the complete decoding procedure is applied only to the remainder of the word.

5.1 Summary of the decoding procedure

The relation among words, characters, tree nodes, character HMMs, and levels of the LDLBA can be simply stated: each word is composed of a sequence of L characters that are further represented by a sequence of L linked nodes in the lexical tree. During the decoding each tree node is "replaced" by an HMM, which together with the observation sequence forms a level of the LDLBA (trellis or lattice). At each level, the multiple models for the character class are cycled. At the end of each level, the arrays P_t^*, B_t^*, and W_t^* are obtained. These arrays are integrated into the lexical tree to avoid repeated computation of the prefixes of similar words. Figure 7 illustrates the decoding of a word from the lexicon for a given observation sequence O.

The result of the recognition process is a list containing multiple word hypotheses with the respective likelihoods (Fig. 7f). Additionally, the state lattice that contains all characters that form the word hypotheses (which is readily converted into an ASCII label) and the segmentation of the word hypotheses into characters can also be provided.

5.2 Analysis 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 without a significant increase in the size of the search space. Figure 8 shows an example of multiple models for the character "M". If the level reduction is not carried out as shown in Fig. 8a, each model would generate a tree branch to be decoded during the recognition (Fig. 8b) and that would be very time-consuming. On the other hand, the main drawback of the LDLBA is that it provides a sub-optimal solution due to the level reduction and as

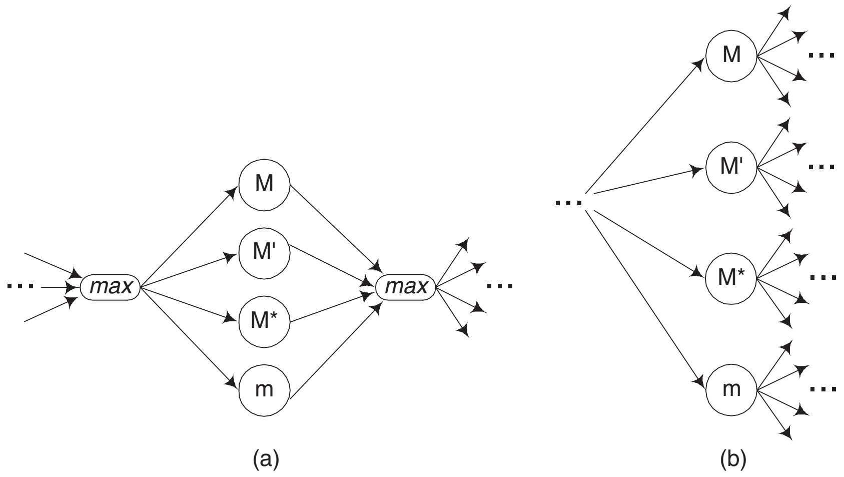


Fig. 8a,b. The inclusion of several models for a unique character class in parallel. **a** With level reduction only the model with the highest likelihood is expanded. **b** Without level reduction all models are expanded

a consequence, the recognition accuracy is expected to be inferior relative to a recognition approach that accounts for the whole search space. The sub-optimal solution arises from the level reduction, that is, at the end of each level we choose one among the multiple models (e.g. uppercase or lowercase 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 decoding of the last character model of the word.

A close analysis of the errors of the LDLBA indicates that the majority of them ($\approx 83\%$) occurs on relatively short words ($\leq$ 4 characters). For long words, even if wrong local decisions are taken, that is, the best model is wrongly selected at the level reduction, the final likelihood of the words is not severely affected since the errors are minimized by the context during decoding, while for short words wrong local decisions are more meaningful.

5.3 Incorporation of time and length constraints to the LDLBA

Even though the computational complexity for the level building approach seems to be significantly less than an exhaustive search through the whole search space, there are several ways of further improving its performance in terms of speed. To this end, we can rely on the particular characteristics of our application:

- We are decoding words of known length from a lexicon. Thus the number of levels l of the LDLBA is known a priori (before the decoding).
- The words are formed by the concatenation of character HMMs whose 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 (no more than 6).

Given these three remarks, it is clear that the conditions for the recursion of the LDLBA (Eq. 14) are not realistic since it is carried out for almost the entire sequence of observations ($t = 2, \ldots, T$) at almost all levels ($l = 2, \ldots, L$). In other words, this means that the whole observation sequence that represents a whole word is always matched against individual character models. This is done due to the difficulty of segmenting words into characters because the boundaries of characters, that is, which part of the sequence of observations corresponds to each character, are difficult to determine in the case of unconstrained handwritten words. 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 optimized. Another point is that short observation sequences are more likely to be generated from short words. Therefore, there is not much sense in decoding long words from the lexicon given a short observation sequence. Nevertheless, if we consider the architecture of the character model shown in Fig. 4 in which the maximum and minimum number of observations that it can emit are well known, new limits to optimize the search can be easily established.

Therefore, the two hypotheses to further improve the performance of the LDLBA are: (1) by constraining the number of levels according to the length of the observation sequences (*length*) and (2) by limiting the number of observations aligned at each level (*time*).

5.3.1 Time constraint. The time constraint concerns the limitation of the number of observations aligned at each level of the 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. Figure 9 shows how these two constraints are incorporated into the levels of the LDLBA.

Table 3. A summary of the approximate computational complexity and storage requirements for several search strategies

Search strategy	Computational complexity	Storage requirements
Exhaustive FL	TN^2H^LV	$2TH^L$
Exhaustive LT	$TN^2H^{L'}V$	$2TH^{L'} + 2TH^{(L-L')}V$
LDLBA FL	TN^2HLV	$3TL$
LDLBA LT	$TN^2HL'V$	$3TL + 3T(L-L')V$
CLDLBA LT	$T'N^2HL''V$	$3TL + 3T(L-L')V$

To incorporate these two constraints into the LDLBA, only the limits of the observation index t are modified as follows.

$$t = s(l), s(l)+1, \ldots, e(l)-1, e(l) \tag{24}$$

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 Eqs. 25 and 26, respectively:

$$s(l) = \begin{cases} 1 & \text{if } l = 1 \\ \lfloor l - s^* \rfloor & \text{if } l > 1,\ s^* < l \end{cases} \tag{25}$$

$$e(l) = \begin{cases} \lfloor D_{max} \cdot [l + e^*] \rfloor & \text{if } D_{max} \cdot [l + e^*] < T \\ T & \text{otherwise} \end{cases} \tag{26}$$

where s^* and e^* are the two control factors to be determined and D_{max} is the maximum number of observations that can be emitted, which is associated with the HMM architecture.

5.3.2 Length constraint. The length constraint concerns the limitation of the number of levels of the LDLBA. We introduce the variable Lv, which denotes the maximum number of levels of the LDLBA given an observation sequence with length T (Fig. 9). To incorporate this constraint into the LDLBA, 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:

$$l = 1, 2, \ldots, Lv, \qquad 1 < Lv \leq L \tag{27}$$

where Lv is an integer given by Eq. 28 and its lower and upper limits are given by the shortest and the longest words in the lexicon, respectively.

$$Lv = TLv^*, \qquad 0 < Lv^* \leq 1 \tag{28}$$

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 tradeoff between recognition accuracy and recognition speed. 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. Several approaches exist for optimization such as response surface techniques, hill climbing algorithms, and genetic algorithms. 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 that studies the effects of the multiple variables simultaneously and optimizes the performance of the constrained recognition system [1]. The application of the statistical experimental design technique to determine the optimal values of the control factors (s^*, e^*, and Lv^*) that optimize the performance of the recognition system is described in [27].

5.4 Computational complexity and storage requirements

It is worthwhile comparing the computational complexity and storage requirements of the different search strategies. The computational complexity of decoding an elementary N-state HMM is TN^2, which is common for all search strategies. The recognition process involves the decoding of V words that are made up by the concatenation of L characters that are modeled by HMMs where each character class may have H models. Table 3 shows the computational complexity of decoding the whole lexicon considering several search strategies and the combination of the elements involved in the recognition process. In this table, we consider both a flat-structured lexicon (FL) and a lexical tree (LT). In the case of the lexical tree, the variable L is replaced by L', which is defined as the L/RR, where RR is the reduction rate in the number of characters afforded by representing the lexicon as a tree structure.

The storage requirements refer to the arrays used during the decoding of the whole lexicon, that is, those that keep the probabilities and the best state sequences. The search strategies that use flat-structured lexicons (denoted as FL in Table 3), require only the storage of the variables used during the decoding of each word because no information is shared between words. On the other hand, the search strategies that use tree-structured lexicons (denoted as LT in Table 3) require the storage of the variables used during the decoding of each word as well as some additional variables to store the information that is shared by words with common prefixes. Furthermore, the search strategies based on the LDLBA also require the storage of the best models at each level.

The computational complexity of the constrained lexicon-driven level building algorithm (CLDLBA) is difficult to establish exactly since it depends on heuristics to reduce some of the elements involved in the number of computations, say T and L, where $T' < T$ denotes the reduced number of operations due to the time constraint and $L'' < L'$ denotes the reduced number of operations due to the length constraint.

Table 3 also includes the computational complexity and the storage requirements for the exhaustive search

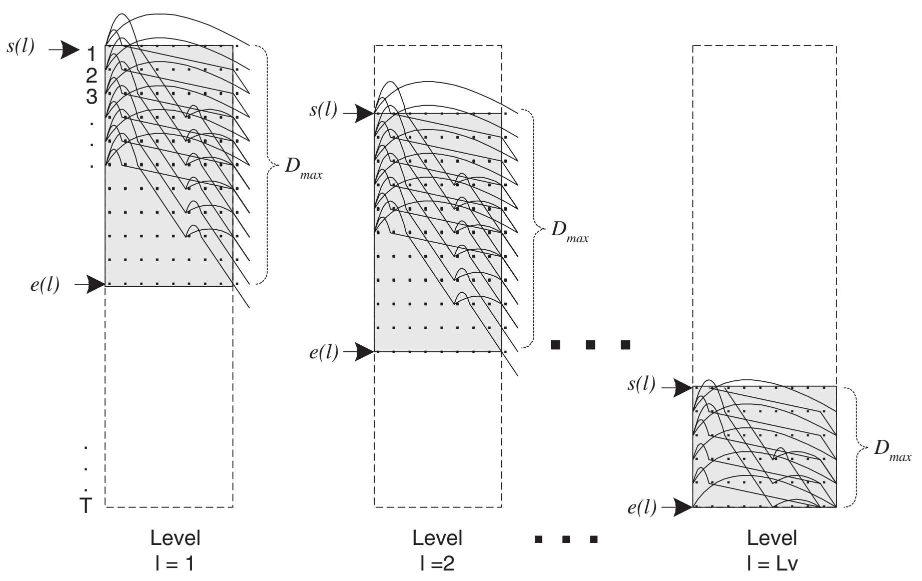


Fig. 9. Levels of the LDLBA incorporating the time constraints $s(l)$ and $e(l)$, which limit the number of observations aligned at each level, and the length constraint Lv, which limits the number of levels of the LDLBA

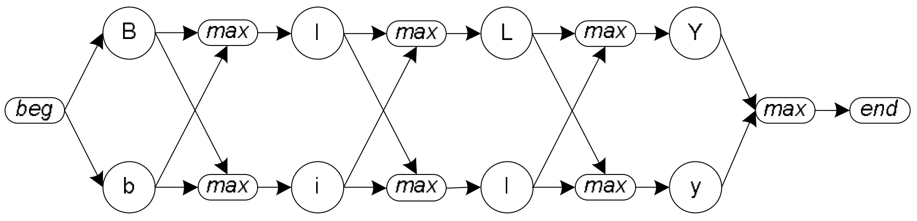


Fig. 10. The decoding of an unconstrained handwritten word with the conventional Viterbi algorithm and a flat-structured lexicon

Table 4. Typical computational requirements for the case T=30, T'=25, N=10, H=2, L=10, L'=8, L''=7, and V=30,000

Search strategy	Computational complexity ($\times 10^9$)	Storage requirements
Exhaustive FL	92	61,440
Exhaustive LT	23	7,261,440
LDLBA FL	1.8	900
LDLBA LT	1.44	5,400,900
CLDLBA LT	1.05	5,400,900

of the whole search space using the conventional Viterbi algorithm with a flat lexicon (FL) and with a lexical tree (LT). The exhaustive search (Fig. 5d) implies the use of two arrays for each tree branch to keep the best states and likelihoods at each t. This highlights another problem: the number of possible hypotheses grows exponentially as a function of the number of character class models, and this imposes formidable storage capability requirements. To get a feeling of the computational complexity and storage requirements of each recognition strategy in Table 4, we used typical values of the parameters.

6 Experimental results

Experiments have been carried out using a database containing unconstrained handwritten words extracted from postal envelopes consisting of 12,012 and 3,475 handwritten words for training and validation, respectively. The test set used in the experiments consists of another 4,674 handwritten words. Character models (HMMs) were trained once and used with all search strategies. Parameters and models were developed exclusively on the training and validation sets. The criteria that we have used to evaluate the recognition approaches are:

– *Recognition speed*: performance in terms of the running time of the software implementation (on an AMD Athlon 1.1 GHZ) – all figures refer to the recognition time and do not include the computation time needed for preprocessing and feature extraction.
– *Recognition accuracy*: performance in terms of the recognition rate, which is given as the number of correctly recognized words by the total number of words.

All results are averaged over the test set of 4,674 words and 10 runs. At each run, a lexicon of fixed size is randomly generated. Furthermore, to assess the performance of the recognition system for both small and large vocabularies, all the experiments were carried out considering five different sizes of lexicons (10, 100, 1,000, 10,000, and 30,000 entries), which were dynamically generated from a global lexicon with 36,116 entries.

6.1 Performance of the baseline recognition system (BRS)

For comparison purposes, we first evaluated the performance of a baseline recognition system (BRS) given

Table 5. Word recognition rate and recognition time for the baseline recognition system (BRS) on the SRTP test dataset

Lexicon	Recognition rate (%)			Recognition
size	TOP 1	TOP 5	TOP 10	time (sec/word)
10	98.93	99.93	100	0.074
100	95.89	98.99	99.40	0.690
1,000	89.79	95.97	97.30	6.902
10,000	79.50	89.53	91.89	75.39
30,000	73.70	85.17	88.15	216.7

Table 6. Word recognition rate and recognition time of the LDLBA and speedup over the baseline recognition system (BRS)

Lexicon	Recognition rate (%)			Recognition	Speedup
size	TOP 1	TOP 5	TOP 10	time (sec/word)	(×BRS)
10	98.76	99.93	100	0.013	5.5
100	95.46	98.82	99.40	0.123	5.6
1,000	89.00	95.49	96.81	1.095	6.3
10,000	78.22	88.49	90.99	10.18	7.4
30,000	71.03	84.02	87.06	28.14	7.7

the experimental conditions stated earlier. This baseline recognition system was already presented elsewhere and was implemented using the paradigm shown in Fig. 10 with a flat-structured lexicon and Viterbi search to decode unconstrained words [13]. The BRS uses two models per character, one for uppercase and another for lowercase. Table 5 shows the performance of the BRS in terms of both recognition accuracy and recognition time.

As expected, the recognition rate decreases with an increasing number of words in the lexicon. However, the foremost concern is the recognition time that is as low as 74 ms for a small lexicon and as high as 3.6 min in the case of a large lexicon. Thus this recognition approach is clearly impractical for use in real-life applications dealing with large vocabularies.

6.2 Performance of the lexicon-driven level building algorithm (LDLBA)

The first attempt to improve the performance of the BRS is by representing the lexicon as a tree and pursuing the search only by considering the character models with the best partial likelihood. Table 6 summarizes the results for the recognition accuracy and recognition time for the recognition system based on the LDLBA. Table 6 also lists the speedup obtained over the BRS.

Clearly the LDLBA decoding is several times faster than the BRS with speedup factors between 5.5 and 7.7 for 10-word and 30,000-word lexicons, respectively. The relative reduction in recognition rates, compared to the BRS, is not very significant, as shown in Table 7. The average loss of accuracy is 1.068%, 0.568%, and 0.496% for

Table 7. Difference in the word recognition rates between the LDLBA and the BRS

Lexicon	Recognition rate (%)		
size	TOP 1	TOP 5	TOP 10
10	0.17	0	0
100	0.43	0.17	0
1,000	0.79	0.48	0.49
10,000	1.28	1.04	0.90
30,000	2.67	1.15	1.09
Average	1.068	0.568	0.496

the *TOP* 1, *TOP* 5, and *TOP* 10 best choices, respectively, and increases with the lexicon size. However, it can be argued that the speedup afforded by the LDLBA is well worth the increase in error rate.

The errors are due to the search algorithm approximation and/or pruning of the correct transcription. We attribute the reduction in recognition rate to the level reduction of the LDLBA 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 BRS with a flat lexical structure decodes all hypotheses. This would also be possible with the tree structure; however, the number of hypotheses to be decoded would make the advantages of using a lexical tree not meaningful.

The problems in the search due to the level reduction is more pronounced on short words especially. Considering that the average length of the words in the test set is 11.14 characters, 83% of the search errors occur with words less than 4 characters long.

6.3 Performance of the constrained LDLBA (CLDLBA)

The inclusion of some constraints in the LDLBA yields the constrained lexicon-driven level building algorithm (CLDLBA) presented in the previous section. It is an attempt to further reduce processing time while preserving recognition accuracy. These constraints are particularly interesting for establishing a tradeoff between recognition accuracy and recognition speed.

We have set as our 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 to six since this is the maximum number of observations that the character HMMs can emit (due to the HMM architecture). To this end, we 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 speed. 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

Table 8. Values of the constraints of the CLDLBA determined by the statistical experimental design technique [27]

Lexicon	Constraints		
size	Lv^*	e^*	s^*
10	0.18	0.5	−0.5
100	0.68	0.5	−0.5
1,000	0.87	0.5	−0.5
10,000	0.85	0.5	−0.5
30,000	0.89	0.5	−0.5

Table 9. Word recognition rate and recognition time for the system based on the CLDLBA

Lexicon	Recognition rate (%)			Recognition	Speedup
size	TOP 1	TOP 5	TOP 10	time (sec/word)	(×BRS)
10	98.50	99.85	100	0.009	7.8
100	94.95	98.59	99.19	0.084	8.2
1,000	88.42	95.04	96.41	0.726	9.5
10,000	77.60	87.89	90.37	6.793	11.1
30,000	71.03	83.42	86.58	18.36	11.8

optimal values of the control factors that jointly optimize both the accuracy and the speed were determined [27]. Several experimental runs were conducted, corresponding to the different values of the control factors, and both recognition rate and recognition speed were measured. In these experiments, we used the validation set. Table 8 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 8, the performance of the CLDLBA was evaluated over the test set and the results are shown in Table 9.

Table 8 shows that the time constraints (s^*, e^*) do not depend on the lexicon size. On the other hand, the length constraint (Lv^*) does depend on it. Table 10 shows the reduction in recognition rates relative to the BRS. 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* ten best choices, respectively, when compared with the performance of the LDLBA. On the other hand, if we compare the speedup shown in Table 9, the CLDLBA 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* ten best choices, respectively, when compared with the performance of the BRS. Comparing the performance of the CLDLBA with the BRS in terms of recognition speed, we have speedup factors between 7.8 and 11.8. The speedup afforded by the CLDLBA is well worth the increase in error rate.

The errors introduced by the CLDLBA are due to the constraints. We have investigated the effects of each constraint on the recognition accuracy and on the recognition speed separately and 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 (≤ 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 speed independently of the lexicon size. This is attributed to the architecture of the character models that include null transitions. These two constraints are particularly sensitive to the presence of null transitions. Table 11 shows in an abbreviated manner the individual contributions of s^* and Lv^* to the accuracy and speed of the LDLBA.

Table 10. Difference in the word recognition rates between the CLDLBA and the BRS

Lexicon	Recognition rate (%)		
size	TOP 1	TOP 5	TOP 10
10	0.43	0.08	0
100	0.94	0.40	0.21
1,000	1.37	0.93	0.89
10,000	1.90	1.64	1.52
30,000	2.67	1.75	1.57
Average	1.462	0.96	0.838

Table 11. Individual influence of the control factors s^* and Lv^* on the recognition rate (*TOP* 1) and on the recognition speed of the LDLBA

Lexicon	Recognition rate (%)		Recognition time (sec/word)	
size	s^*	Lv^*	s^*	Lv^*
10	98.48	98.76	0.030	0.039
100	94.95	95.46	0.258	0.343
1,000	88.42	89.00	2.237	2.942
10,000	77.60	78.21	17.17	21.71

Table 12. Recognition rate (TOP 1) for the LDLBA with and without bigram probabilities

Lexicon	Recognition rate (%)	
Size	Without bigram	With bigram
10	98.31	98.76
100	94.43	95.46
1,000	86.90	89.00
10,000	75.54	78.22
30,000	68.88	71.03

6.4 Influence of the contextual information between levels

The results presented so far have taken into account the bigram probabilities between the word characters. However, we have also analyzed the influence of the bigram probabilities related to the writing style transition within the words. Table 12 shows the recognition accuracy of the LDLBA with and without the bigram probabilities. Note that the use of such contextual information improves sig-

Table 13. Recent results on large vocabulary handwriting recognition

Author	Method	Lexicon size	Accuracy (%)	Test set (#)	Comments
Marti et al. [31]	HMM	7,719	60.05	44,019	Unconstrained, omniwriter, 250 writers
Cho et al. [7]	HMM	10,000	67.09	700	Cursive, omniwriter
Wimmer et al. [42]	NN/HMM	20,200	73.13	1,500	Cursive, writer-dependent, 9 writers
Brakensiek et al. [3]	HMM	30,000	89.2	800	Cursive, writer-dependent, 4 writers
Dzuba et al. [10]	DP	40,000	60.7	3,000	Cursive, omniwriter

nificantly the accuracy of the LDLBA (1.8% on average), especially for large lexicons.

7 Discussion and conclusion

This paper has focused on the problems of unconstrained handwriting recognition given a large vocabulary and multiple character models. The foremost concern addressed in this paper is computational complexity of the recognition process. We have proposed a tree-organized lexicon and a decode algorithm based on the level building algorithm to deal efficiently with multiple character models avoiding an exponential explosion of the search space, with a moderate reduction in recognition accuracy.

The use of the LDLBA to decode unconstrained handwritten words facilitates the inclusion of multiple models for each individual character (e.g., context-dependent models) with an insignificant increase in the size of the search space and computational complexity. However, due to the local decision, the recognition accuracy is slightly inferior to that of an equivalent Viterbi algorithm that accounts for the whole search space.

We have also introduced some constraints to the decoding algorithm to further reduce the search effort. We have seen that limiting the number of observations according to the level of the LDLBA, as well as limiting the number of levels of the LDLBA by taking into account the length of the observation sequences, leads to an improvement of 24.4%–30.3% in the recognition speed with a slight reduction of 0.28%–0.77% in the recognition rate for lexicons with 10 to 30,000 entries, respectively, when compared with the conventional LDLBA. If we compare these results with those obtained by the BRS using the Viterbi algorithm with a flat lexicon, the improvement in speed is more impressive (7.8–11.8 speedup factors) with a reasonable reduction in the recognition rate (0.45%–1.8%). Nevertheless, the recognition times for large vocabularies are still far from meeting the throughput requirements of many real-life applications; however, the achieved results are very promising, that is, a recognition rate of about 71% and a decoding time of about 18 s on a 30,000-word vocabulary.

Table 13 shows some recent results from the literature for the problem of large vocabulary handwriting recognition. It should be stressed that these studies have used different datasets and experimental conditions, which makes a direct comparison very difficult. However, they are very useful to illustrate how difficult is the problem we are dealing with as well as to highlight the promising results that we have obtained by applying the proposed recognition scheme.

We also have investigated the use of Viterbi beam search; however, contrary to what is reported [9,21], we did not obtain good results. We have attributed this to the characteristics of our HMMs that have null transitions. In such a situation, it is difficult to find a pruning threshold since we can reach the last state of the last character within a word with very low likelihoods (given a sum of logs). Furthermore, the search strategy that we have adopted is time asynchronous, and the beam search is usually associated with time-synchronous search.

In all experiments presented in Sect. 6, two models per character class were used. Nevertheless, we believe that the loss in recognition accuracy of the proposed search method may be compensated by using a higher number of models per character class such as context-dependent character models that depend on the immediate left or right neighboring characters, models for the first characters of words, models for the most common prefixes, etc. [12,29]. For instance, the benefits of using multiple character models in the baseline recognition system is presented in [11,12]. As mentioned earlier, these models can be easily integrated into the LDLBA with little effect on recognition time.

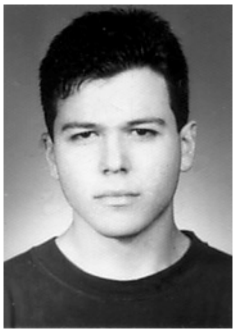
Acknowledgements. The authors would like to acknowledge the National Research Council of Brazil (CNPq) (grant 200276-98/0), the Ministère de l'Éducation du Québec, and the NSERC, Canada for supporting this research, and the Service Technique de la Poste (SRTP-France) for providing us the database and the baseline recognition system.

References

1. Box GEP, Hunter WG, Hunter JS (1978) Statistics for experimenters: an introduction to design, data analysis, and model building. Wiley, New York
2. Bozinovic RM, Srihari SN (1989) Off-line cursive script word recognition. IEEE Trans Patt Anal Mach Intell 22(1):63–84
3. Brakensiek A, Willett D, Rigoll G (2000) Unlimited vocabulary script recognition using character n-grams. In: Proceedings of the 22nd DAGM symposium, Tagungsband Springer-Verlag, Kiel, Germany, 13–15 September 2000, pp 436–443

4. Britto AS, Sabourin R, Bortolozzi F, Suen CY (2001) A two-stage hmm-based system for recognizing handwritten numeral strings. In: Proceedings of the 6th international conference on document analysis and recognition, Seattle, 10–13 September 2001, pp 396–400
5. Bunke H, Roth M, Schukat-Talamazzini EG (1995) Off-line cursive handwriting recognition using hidden markov models. Patt Recog 28(9):1399–1413
6. Chen MY, Kundu A, Zhou J (1994) Off-line handwritten word recognition using a hidden markov model type stochastic network. IEEE Trans Patt Anal Mach Intell 16(5):481–496
7. Cho W, Kim JH (1994) Off-line recognition of cursive words with network of hidden Markov models. In: Proceedings of the 4th international workshop on the frontiers of handwriting recognition, Taipei, Taiwan, 7–9 December 1994, pp 410–417
8. Connell S (2000) Online handwriting recognition using multiple pattern class models. PhD thesis, Michigan State University, East Lansing
9. Dolfing JGA (1998) Handwriting recognition and verification – a hidden Markov approach. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands
10. Dzuba G, Filatov A, Gershuny D, Kill I (1998) Handwritten word recognition – the approach proved by practice. In: Proceedings of the 6th international workshop on frontiers in handwriting recognition, Taejon, Korea, 12–14 August 1998, pp 99–111
11. El-Yacoubi A (1996) Modélisation Markovienne de l'écriture manuscrite application à la reconnaissance des adresses postales. PhD thesis, Université de Rennes 1, Rennes, France
12. El-Yacoubi A, Sabourin R, Suen CY, Gilloux M (1998) Improved model architecture and training phase in an off-line hmm-based word recognition system. In: Proceedings of the 14th international conference on pattern recognition, Brisbaine, Australia, 16–20 August 1998, pp 17–20
13. El-Yacoubi A, Gilloux M, Sabourin R, Suen CY (1999a) Unconstrained handwritten word recognition using hidden markov models. IEEE Trans Patt Anal Mach Intell 21(8):752–760
14. El-Yacoubi A, Sabourin R, Gilloux M, Suen C (1999b) Off-line handwritten word recognition using hidden markov models. In: Jain LC, Lazzerini B (eds) Knowledge-based intelligent techniques in character recognition, CRC Press, Boca Raton, FL, pp 191–229
15. Elms AJ, Procter S, Illingworth J (1999) The advantage of using an hmm-based approach for faxed word recognition. Int J Doc Anal Recog 1:18–36
16. Farouz C (1999) Reconnaissance de mots manuscrits hors-ligne dans un vocabulaire ouvert par modélisation Markovienne. PhD thesis, Université de Nantes, Nantes, France
17. Gader PD, Mohamed MA, Chiang JH (1994) Handwritten word recognition with character and inter-character neural networks. IEEE Trans Sys Man Cybern Part B 27:158–164
18. Gilloux M (1998) Réduction dynamique du lexique par la méthode tabou. In: Proceedings of the colloque international Francophone sur l'ecrit et le document, Quebec, 11–13 May 1998, pp 24–31
19. Guillevic D, Suen CY (1995) Cursive script recognition applied to the processing of bank cheques. In: Proceedings of the 3rd international conference on document analysis and recognition, Montreal, 14–16 August 1995, pp 11–14
20. Guillevic D, Nishiwaki D, Yamada K (2000) Word lexicon reduction by character spotting. In: Proceedings of the 7th international workshop on frontiers in handwriting recognition, Amsterdam, 11–13 September 2000, pp 373–382
21. Jaeger S, Manke S, Reichert J, Waibel A (2001) Online handwriting recognition: the npen++ recognizer. Int J Doc Anal Recog 3:169–180
22. Kaufmann G, Bunke H, Hadorn M (1997) Lexicon reduction in an hmm-framework based on quantized feature vectors. In: Proceedings of the 4th international conference on document analysis and recognition, Ulm, Germany, 18–20 August 1997, pp 1097–1101
23. Kim G, Govindaraju V (1997a) Bankcheck recognition using cross validation between legal and courtesy amounts. In: Impedovo S, Wang PSP, Bunke H (eds) Int J Patt Recog Artif Intell pp 657–673. World Scientific, Singapore
24. Kim G, Govindaraju V (1997b) A lexicon driven approach to handwritten word recognition for real-time applications. IEEE Trans Patt Anal Mach Intell 19(4):366–379
25. Kimura F, Shridhar M, Chen Z (1993) Improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words. In: Proceedings of the 2nd international conference on document analysis and recognition, Tsukuba, Japan, 20–22 October 1993, pp 18–22
26. Koerich AL, Sabourin R, Suen CY, El-Yacoubi A (2000) A syntax-directed level building algorithm for large vocabulary handwritten word recognition. In: Proceedings of the 4th international workshop on document analysis systems, Rio de Janeiro, 10–13 December 2000, pp 255–266
27. Koerich AL, Sabourin R, Suen CY (2001) A time-length constrained level building algorithm for large vocabulary handwritten word recognition. In: Proceedings of the 2nd international conference on advances in pattern recognition, Rio de Janeiro, 11–14 March 2001, pp 127–136
28. Koerich AL, Sabourin R, Suen CY (2002) Large vocabulary off-line handwriting recognition: a survey. Patt Anal Applicat 6(2):97–121
29. Lee KF (1990) Context-dependent phonetic hidden markov models for speaker-independent continuous speech recognition. IEEE Trans Acoust Speech Signal Process 39(4):599–609
30. Madhvanath S, Krpasundar V, Govindaraju V (2001) Syntatic methodology of pruning large lexicons in cursive script recognition. Patt Recog 34:37–46
31. Marti U, Bunke H (2000) Handwritten sentence recognition. In: Proceedings of the 15th international conference on pattern recognition, Barcelona, 3–8 September 2000, pp 467–470
32. Myers CS, Rabiner LR (1981a) Connected digit recognition using a level building dtw algorithm. IEEE Trans Acoust Speech Signal Process 29(3):351–363
33. Myers CS, Rabiner LR (1981b) A level building dynamic time warping algorithm for connected word recognition. IEEE Trans Acoust Speech Signal Process 29(2):284–297

34. Procter S, Illingworth J (1999) Handwriting recognition using hmms and a conservative level building algorithm. In: Proceedings of the 7th international conference on image processing and its applications, Manchester, UK, 12–15 July 1999, pp 736–739
35. Procter S, Illingworth J, Mokhtarian F (2000) Cursive handwriting recognition using hidden markov models and a lexicon-driven level building algorithm. IEE Proc Vision Image Signal Process 147(4):332–339
36. Rabiner LR, Levinson SE (1985) A speaker-independent, syntax-directed, connected word recognition system based on hidden markov models and level building. IEEE Trans Acoust Speech Signal Process 33(3):561–573
37. Rabiner L, Lee CH, Juang BH, Wilpon JG (1989) Hmm clustering for connected word recognition. In: Proceedings of the international conference on acoustics, speech and signal processing, Glasgow, 23–26 May 1989, pp 405–408
38. Ratzlaff EH, Nathan KS, Maruyama H (1996) Search issues in the IBM large vocabulary unconstrained handwriting recognizer. In: Proceedings of the 5th international workshop on frontiers in handwriting recognition, Essex, UK, 2–5 September 1996, pp 177–182
39. Seni G, Srihari RK, Nasrabadi N (1996) Large vocabulary recognition of on-line handwritten cursive words. IEEE Trans Patt Anal Mach Intell 18(7):757–762
40. Senior AW, Robinson AJ (1998) An off-line cursive handwriting recognition system. IEEE Trans Patt Anal Mach Intell 20(3):309–321
41. Shridhar M, Houle G, Kimura F (1997) Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. In: Proceedings of the 4th international conference on document analysis and recognition, Ulm, Germany, 18–20 August 1997, pp 861–865
42. Wimmer Z, Salicetti SG, Dorizzi B, Gallinari P (1997) Off-line cursive word recognition with a hybrid neural-hmm system. In: Proceedings of the 1st Brazilian symposium on document image analysis, Curitiba, Brazil, 2–5 November 1997, pp 249–260
43. Zimmermann M, Mao J (1999) Lexicon reduction using key characters in cursive handwritten words. Patt Recog Lett 20:1297–1304



Alessandro L. Koerich received the B.Sc. degree in electrical engineering from the Federal University of Santa Catarina (UFSC), Brazil, in 1995, the M.Sc. degree in electrical engineering from the University of Campinas (UNICAMP), Brazil, in 1997, and the Ph.D. degree in automated manufacturing engineering from the École de Technologie Supérieure (ETS), Université du Québec, Montréal, Canada, in 2002. From 1997 to 1998 he was a lecturer at the Federal Center for Technological Education (CEFETPR). From 1998 to 2002 he was a visiting scientist at the Centre for Pattern Recognition and Machine Intelligence (CENPARMI). In 2003, he joined the Pontifical Catholic University of Paraná (PUCPR), Curitiba, PR, Brazil where he is currently an associate professor of computer science. His research interests include machine learning, pattern recognition, and multimedia.

Robert Sabourin received the B.ing., M.Sc.A. and Ph.D. degrees in electrical engineering from the École Polytechnique de Montréal in 1977, 1980, and 1991, respectively. In 1977, he joined the physics department of the Université de Montréal where he was responsible for the design and development of scientific instrumentation for the Observatoire du Mont Mégantic. In 1983, he joined the staff of the École de Technologie Supérieure, Université du Québec, Montréal, P.Q., Canada, where he is currently a professeur titulaire in the Département de Génie de la Production Automatisée. In 1995, he joined the Computer Science Department of the Pontifícia Universidade Católica do Paraná (PUCPR, Curitiba, Brazil) where since 1998 he has been coresponsible for the implementation of a Ph.D. program in applied informatics. Since 1996, he has been a senior member of the Centre for Pattern Recognition and Machine Intelligence (CENPARMI). His research interests are in the areas of handwriting recognition and signature verification for banking and postal applications.

Ching Y. Suen received an M.Sc.(Eng.) from the University of Hong Kong and a Ph.D. from the University of British Columbia, Canada. In 1972, he joined the Department of Computer Science of Concordia University, where he became professor in 1979 and served as chairman from 1980 to 1984 and as associate dean for research of the Faculty of Engineering and Computer Science from 1993 to 1997. Currently he is a distinguished research chair in AI & PR and the director of CENPARMI, the Centre for Pattern Recognition and Machine Intelligence. Professor Suen is the author/editor of 11 books and more than 300 papers on subjects ranging from computer vision and handwriting recognition to expert systems and computational linguistics. He is the founder and editor-in-chief of a journal and an associate editor of several journals related to pattern recognition. A Fellow of the IEEE, IAPR, and the Academy of Sciences of the Royal Society of Canada, he has served several professional societies as president, vice-president, or governor. He is also the founder and chair of several conference series including ICDAR, IWFHR, and VI and served as the general chair of the International Conference on Pattern Recognition in Quebec City in 2002. Dr. Suen is the recipient of several awards, including the ITAC/NSERC award in 1992 and the Concordia "Research Fellow" award in 1998.