A Syntax–Directed Level Building Algorithm for Large Vocabulary Handwritten Word Recognition

Alessandro L. Koerich[1,2], Robert Sabourin[1,2], Ching Y. Suen[2], and Abdenaim El-Yacoubi[3]

[1] Lab. d'Imagerie, de Vision et d'Int. Artificielle, École de Technologie Supérieure
1100, rue Notre–Dame Ouest, H3C 1K3, Montreal, QC, Canada
{akoerich,sabourin}@livia.etsmtl.ca

[2] Centre for Pattern Recognition and Machine Intelligence, Concordia University
1455, Maisonneuve Blvd. West, H3G 1M8, Montreal, QC, Canada
suen@cenparmi.concordia.ca

[3] Pontifícia Universidade Católica do Paraná
R. Imaculada Conceição, 1155, 80215-901, Curitiba, PR, Brazil
yacoubi@ppgia.pucpr.br

Abstract. This paper describes a large vocabulary handwritten word recognition system based on a syntax–directed level building algorithm (SDLBA) that incorporates contextual information. The sequences of observations extracted from the input images are matched against the entries of a tree–structure lexicon where each node is represented by a 10–state character HMM. The search proceeds breadth—first and each node is decoded by the SDLBA. Contextual information about writing styles and case transitions is injected between the levels of the SDLBA. An implementation of the SDLBA together with a 36,100–entry lexicon is described. In terms of recognition speed, the results show that the SDLBA together with the tree–structured lexicon outperforms a baseline system that uses a Viterbi–flat–lexicon scheme while maintaining the same accuracy and consuming a reasonable amount of memory.

1 Introduction

Off–line recognition of handwritten words is a challenging task due to the high variability and uncertainty of human writing. Several proposals to solve this problem have been presented recently [1] [2] [3] [4] [5] [11]. The majority of the state–of–art systems have some constraints during the recognition task. One of the most common constraints is the limitation of the size of the lexicon that a system can deal with. Open vocabulary systems, that is, systems that do not rely on a lexicon during the recognition task are still far from reaching good recognition rates. Furthermore, these kinds of systems generally require a post–processing stage, where substitutions of characters can be performed in order to convert a non–valid word to a valid one. The limitation of the size of the

lexicon is directly related to the speed and accuracy of the systems since in a small lexicon the search space is equally small and the words are more likely to be dissimilar to each other. Thus, an important criterion in assessing system performance is the size of the lexicon used. The main goal of researchers has been accuracy. Nevertheless, practical applications also require the computation to be carried out in real–time within limited resources — CPU power and memory size — of commonly available computers. It has been relatively little work in this direction while preserving the accuracy of systems.

Basically, the word recognition problem can be viewed as a searching problem: searching for the most likely sequence of characters given a sequence of observations extracted from the input image. The problem has generally been tackled using Viterbi or Forward algorithm. Viterbi algorithm is a dynamic programming algorithm that searches the state space for the most likely state sequence that accounts for the input image. The state space is constructed by creating word HMMs from its constituent character HMMs, and all word HMMs are searched sequentially. The number of possible hypotheses grows exponentially as a function of the number of models, lexicon size and the form of linguistic constraints; and this imposes formidable computation and storage capability requirements for the implementation of search algorithms. The search space is huge for even medium–size lexicon applications.

The problem of recognizing unconstrained handwritten words in a large vocabulary is the focus of this paper. Along the next sections we present our approach based on a tree–structured lexicon and a syntax–directed level building algorithm (SDLBA) to deal with large vocabularies where the accuracy must be preserved while providing reasonable recognition speed and memory usage.

2 Language Syntax

The formal syntax of the language is described by a 36,100–word lexicon of French city names. The lexicon defines all possible words and sentences. The words range in length from 1 to 25 characters and the sentences range from 2 to 6 words. In average we have 12 characters per sentence.

There are two ways to organize the lexicon: flat–structured or tree–structured. Flat lexicons are easy to implement and integrate with an HMM recognizer. Furthermore, such kind of lexicon provides us good results for small to medium lexicons (10 to 1,000 words) in terms of recognition speed and considering a 10–state HMM topology. Over 1,000 words, both the recognition speed and the accuracy start to descend and affect the performance of the system [3]. While for a 10–entry lexicon we need to evaluate approximately 2 x 10 x 12 x 10 = 2,400 HMM states[1] for each word, for a 10,000–entry lexicon the number of states spreads out to 2,400,000 and the accuracy diminishes about 20%. Thus, it is necessary to find an alternative scheme to deal with large lexicons.

[1] Considering that the average length of the words in the lexicon is 12 characters, 10 states for uppercase, and 10 states for lowercase character models [3].

Organizing the HMMs to be searched as a character tree instead of a flat structure of independent linear character HMM sequences for each word is probably the most often cited improvement in current search techniques. This structure is 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 character HMMs. Figure 1 illustrates the generation of a lexical tree, using some words representing city names extracted from the global lexicon.

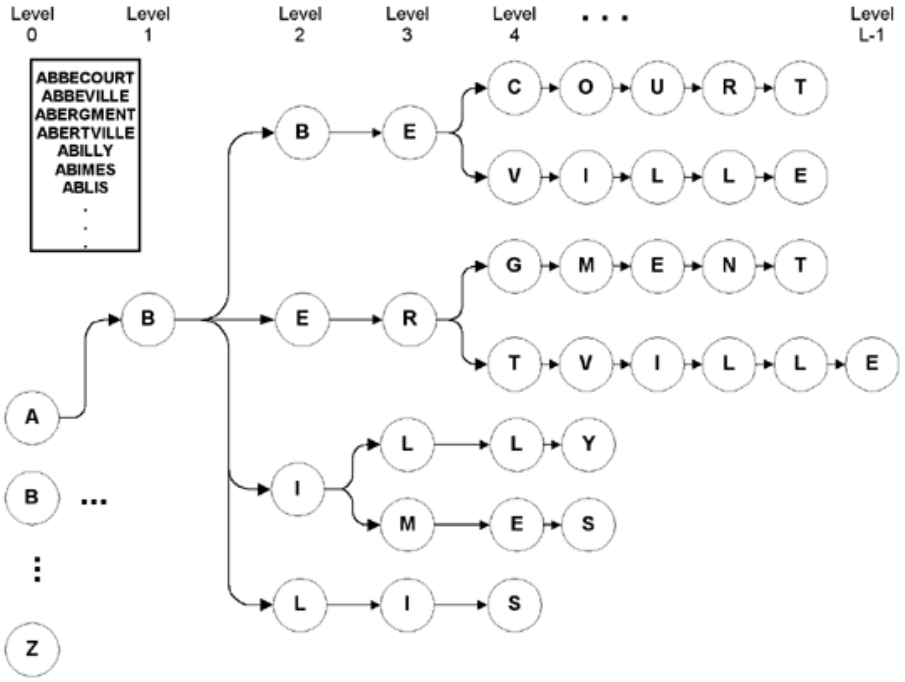

Fig. 1. Example of a lexical tree generated from some entries taken from the global lexicon.

3 Search Techniques

In handwritten word recognition, the search complexity of a full search algorithm based on dynamic programming increases linearly with the size of the vocabulary [2]. Therefore, how to manage the search complexities in a large vocabulary, especially in real–time applications, poses a serious challenge to the researchers.

Several techniques such as breadth–first, depth–first and best–first search have been employed to search and recover words from lexical trees [5] [6] [7]. Nevertheless, our problem is slightly more complex than just searching and recovering a word from the lexical tree. In fact, we need to traverse the whole tree

since we do not know in advance which word we are looking for because the answer is not a single node but a set of linked nodes making up a word. We need to recover all existing words, compute and rank their likelihoods, and select the potential word candidates. For each tree node, it is necessary to align a sequence of observations and compute the probability that such a node has generated the sequence of observations. Moreover, we must retain and use the information provided by the predecessor nodes to compute the likelihoods of the current node until all possible words of the branch are recovered. All these operations are complex and time–consuming, and they are strongly dependent on the lexicon size. Another problem is that our approach takes into account the writing style. Therefore, for each node, the likelihoods of uppercase and lowercase character HMMs must be computed.

3.1 Lexical Tree Decoding

There are several ways to transverse the lexical tree for recovering the words and computing their likelihoods. Since we need to traverse the entire tree, that is, we need to visit all nodes, there is no advantage in choosing one technique in particular. Therefore, the preponderant factor is the complexity of a practical implementation. It is worth mentioning that in the scope of this work we are not considered any pruning of the lexical tree or HMM states.

Since our main goal is to align the sequence of observations and compute the likelihoods once for each node, a linked list seems to be the best solution to traverse in breadth the lexical tree and at the same time, compute and retain the likelihoods for each observation frame without knowing a priori to which word such node belongs [10]. Processing consists of pushing the node to the end of the linked list, initializing it with the accumulated likelihoods at the last node of the list, aligning the whole sequence of observations with such a node and computing the likelihoods for all frames. The same procedure is systematically used until all nodes are pushed into the list. In spite of using more memory than a stack decoder, such a scheme is easier to implement and it requires less operations. Consequently, it is faster than a stack decoder [10].

4 Syntax–Directed Level Building Algorithm (SDLBA)

Until now, we have not mentioned how to compute the likelihoods for each node. As we have previously pointed out, for each node we need to align the sequence of observations and compute the probability that such a node has generated the sequence or at least part of that sequence. Two algorithms are commonly used in this task: Viterbi [3] and Level Building algorithm [9].

Given a set of individual character models $C = \{c_0, \overline{c}_1, c'_2, \ldots, c_{K-1}\}$, and a sequence of observations $O = \{o_0, o_1, \ldots, o_{T-1}\}$, recognition means decoding O into the sequence of models C. 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, the LBA is used to match an observation sequence to a number of models [9]. 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. Since for each node we want to compute the likelihoods of both uppercase and lowercase character HMMs, the LBA seems to be more adequate than Viterbi.

However, it is necessary to adapt this algorithm to take into account some particular characteristics of our character model since all characters are modeled by a left–right–10–state–arc–based HMM [3]. Moreover, we also take into account some contextual information that is given by the prior knowledge about the probability to start a word with an uppercase character (c_k), a lowercase ($\overline{c}_k$) character, or a special character or digit (c'_k) and the probabilities of changing or maintaining the same class of character along the word.

The LBA is an algorithm having no constraints, that is, any model can follow any other model. Since the lexical tree guides the recognition process, the algorithm needs to incorporate some constraints to handle the language syntax provided by the lexical tree as well as the contextual information related to the class transition probabilities [3]. Different from an open vocabulary where all character HMMs are permitted at all levels of the LBA, here, the character HMMs that will be tested at each level, depends on the sequence of nodes of the lexical tree. Furthermore, it will be necessary to compute the likelihoods of only 2 character models at each level of the LBA: one corresponding to the uppercase and other corresponding to the lowercase character. On the other hand, for digits and special characters, only one model is computed. Now, we present the LBA considering that the observations are emitted along transitions (arc model), the presence of null transitions, and the logarithm of the probabilities (known as *likelihoods*).

1) *Initialization*: For l=0, t=0 and j=0 we have:

$$\delta_t(l, j) = 0.0 \tag{1}$$

where $\delta_t(l, j)$ accumulates the likelihoods for each frame t, state j, and position l of the model in the sentence (or level of the LBA). However, the null transitions ij_{Φ} must be initialized also for l=0, t=0, and j=1,2,...,N-1 as:

$$\delta_t(l, j) = \max_{0 \leq i < j} \left[\delta_t(l, j) + a^c_{ij\Phi}(l) \right] \tag{2}$$

where $a^c_{ij\Phi}$ 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, given that the HMM models the character c 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 predecessor level $(l-1)$. At levels $(l > 0)$ we must pick up the likelihood at the most suitable observation frame from the previous level $(l-1)$. For l=1,2,...,L-1, t=0,1,...,T-1 and j=0, we have:

$$\delta_t(l,j) = \delta_t(l-1, N-1) \tag{3}$$

where L is the number of concatenated characters that makes up a word or a sentence. It also corresponds to the number of levels of a specific branch of the lexical tree. T is the length of the observation sequence, or, the number of observations in a sequence.

We must introduce a new back pointer array (α) to record the observation frame (t) at the previous level ($l-1$) in which the character ended. For l=1,2,...,L-1, t=0 and j=0 we have:

$$\alpha_t(l,j) = 0 \tag{4}$$

For all other observation frames (t=1,2,...,T-1) we have:

$$\alpha_t(l,j) = t \tag{5}$$

2) *Recursion*: For l=0, t=1,2,...,T-1, j=0,1,...,N-1 and considering the presence of null transitions, we have:

$$\delta_t(l,j) = \max\left[\max_{0\le i<j}\left[\delta_{t-1}(l,i) + a^c_{ijO_{t-1}}(l)\right]; \max_{0\le i<j}\left[\delta_t(l,i) + a^c_{ij\Phi}(l)\right]\right] \tag{6}$$

where $a^c_{ijO_{t-1}}$ 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 $O_{t-1} = v_m$, given that the HMM models the character c and $v_m \in V = \{v_0, v_1, v_2, \ldots, v_{M-1}\}$ is the discrete set of possible observation symbols. M is the number of distinct observation symbols.

During the recursion the back pointer $\alpha_t(l,j)$ is updated for l=0,1,...,L-1 t=1,2,...,T-1 and j=1,2,...,N-1 as:

$$\alpha_t(l,j) = \begin{cases} \alpha_t\left[l, \arg\max\limits_{0\le i<j}\left[\delta_t(l,i) + a^c_{ij\Phi}(l)\right]\right] & \text{if } ij \text{ is null} \\ \alpha_{t-1}\left[l, \arg\max\limits_{0\le i<j}\left[\delta_{t-1}(l,i) + a^c_{ijO_{t-1}}(l)\right]\right] & \text{otherwise} \end{cases} \tag{7}$$

For higher levels ($l > 0$), $\alpha_t(l,j)$ is also computed by equation (7). However, $\delta_t(l,j)$ is computed by using equation (6) with a slight difference: now, only the states greater than 0 must be considered (j=1,2,...,N-1) since for $j = 0$, $\delta_t(l,j)$ was already computed by equation (3).

3) *Termination*: For l=0, t=0,1,...,T-1 and a given character model $c \in C = \{c_0, \overline{c_1}, c'_2, \ldots, c_{K-1}\}$, we have:

$$\begin{aligned} P_t(l,c) &= \delta_t(l, N-1) \\ B_t(l,c) &= 0 \end{aligned} \tag{8}$$

For higher levels (l=1,2,...,L-1), $P_t(l,c)$ is also computed by equation (8), but $B_t(l,c)$ now is given by:

$$B_t(l,c) = \alpha_t(l, N-1) \tag{9}$$

At the output of the level we store the result 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 $l = 0$ and at higher levels ($l > 0$) we cycle both uppercase (c_k) and lowercase ($\overline{c}_k$) characters in the manner described above, or once if it is a special character or digit (c'_k).

4) *Level Reduction*: At the end of each level when the special characters or both uppercase and lowercase character models have been used, we level reduce to form the array P_t^*. For all l and t we have:

$$P_t^*(l) = \max_c [P_t(l,c)] \tag{10}$$

where P_t^* is the best level output probability.

The above equation searches the model c at level l that best fits (gives the highest likelihood) at each observation t. The level output back pointer for each all l and all t is given by:

$$B_t^*(l) = B_t\left[l, \arg\max_c [P_t(l,c)]\right] \tag{11}$$

where B_t^* is the level output back pointer.

Finally, it is also necessary to keep the output character model c for each level (l) that gives the highest likelihood for each level and observation frame:

$$W_t^*(l) = \arg\max_c [P_t(l,c)] \tag{12}$$

where W_t^* is the level output character indicator.

Now, it is necessary to adapt this algorithm to take into account some contextual information given by the class transition probabilities.

4.1 Incorporation of Contextual Information into LBA

From the learning database it is possible to estimate the probability of a word which starts with an uppercase character ($c = c_k$), a lowercase character ($c = \overline{c}_k$), or a special character ($c = c'_k$) as well as the probability of changing or maintaining the same class along the word. This contextual information contributes to improve the recognition rate of the system [3]. To incorporate such knowledge to the LBA, equation (2) must be rewritten.

For l=0, t=0,1,...,T-1, and j=1,2,...,N-1 we have:

$$\delta_t(l,j) = \max_{0 \le i < j} \left[\delta_t(l,i) + a^c_{ij\Phi}(l) + ctr(l,c,i)\right] \tag{13}$$

where $ctr(l,c,i)$ is the probability of changing or maintaining the case along the word, and for our character model, it is defined as:

$$ctr(l,c,i) = \begin{cases} \tau\,(0, c(l)) & \text{if } l = 0 \text{ and } i = 0 \\ \tau\,(c(l-1), c(l)) & \text{if } l > 0 \text{ and } i = 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where τ is the character class transition probability.

For higher levels, the initialization differs slightly since we must take into account the information provided by the predecessor level $(l-1)$ as well as the contextual information related to the character class transitions. Such contextual information is incorporated in the initialization of higher levels. For l=1,2,...,L-1, t=0,1,...,T-1 and j=1,2,...,N-1 the null transitions (ij_Φ), are also initialized by equation (13).

4.2 SDLBA and Lexical Tree: A Unified Structure

Having presented the lexical tree, the search technique and the SDLBA, the last question that remains is: how to put these three elements together to recognize a handwritten word? First of all we need to formalize the relation among characters, nodes, character HMMs and levels of the LBA, even if it is evident that each character of a word/sentence is represented by a node in the lexical tree and each node represents a level of the LBA. Considering that a word is defined by the concatenation of L characters given by $W = c_0 c_1 \ldots c_{L-1}$, in the lexical tree such a word is represented by a sequence of linked nodes $N = n_0 n_1 \ldots n_{L-1}$. Deliberately the variable L was used to represent the number of levels of the LBA, the number of character in a word, and the depth of a branch in the lexical tree. Figure 2 illustrates the integration among a word from the lexicon, the lexical tree, and the LBA. The experimental results concerning such a unified structure will be presented in the next section.

4.3 Differences Between the SDLBA and the Baseline System

In spite of the idea that the Viterbi algorithm employed in the baseline system [3] is a particular case of LBA for testing only one model, there are some details that make them different if we consider the injection of contextual information between the character HMMs when making up a word HMM. The sole difference is that in the LBA we have a level reduction between the levels as given by equations (10)–(12). This means that at the end of each level we take a decision between the uppercase and lowercase models, choosing that one that gives the best likelihood for each observation frame. As a consequence, for the subsequent levels $(l+1)$, only the character model (c_k or $\overline{c}_k$) that gives the best likelihood at level (l) will be considered. On the other hand, with Viterbi, the decision is taken only after the likelihood of the last character model of the word is computed. Therefore, in the SDLBA we have a local decision while in the Viterbi scheme we have a global decision [8]. Moreover, the baseline system requires the double

of memory space since it keeps the likelihoods of both uppercase and lowercase characters until the final decision is taken. Figure 3 shows the behavior of both algorithms when dealing with contextual information.

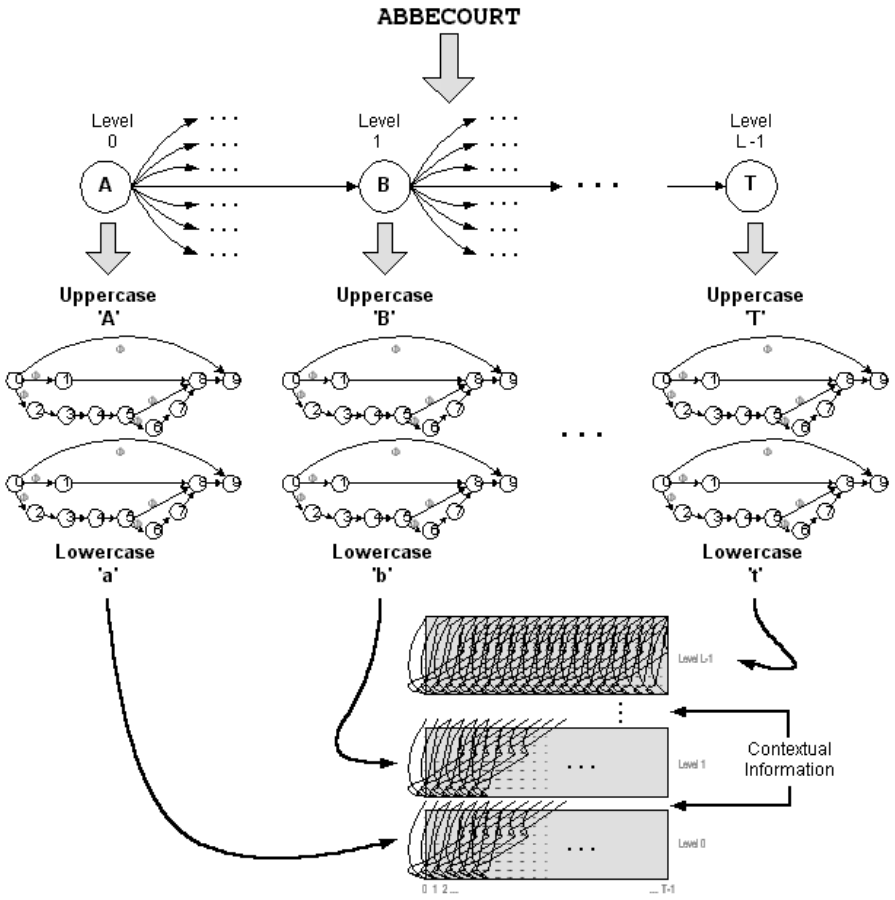

Fig. 2. A simplified overview of the unified structure.

5 Experimental Results

The performance of the recognition scheme based on the SDLBA and lexical tree was evaluated using the same database employed to evaluate the performance of the baseline system. Table 1 summarizes the results for recognition accuracy and recognition speed considering a 36,100–word global lexicon. Seven different sizes of dynamic lexicon were tested: 10, 100, 1,000, 5,000, 10,000, 20,000 and 30,000 entries.

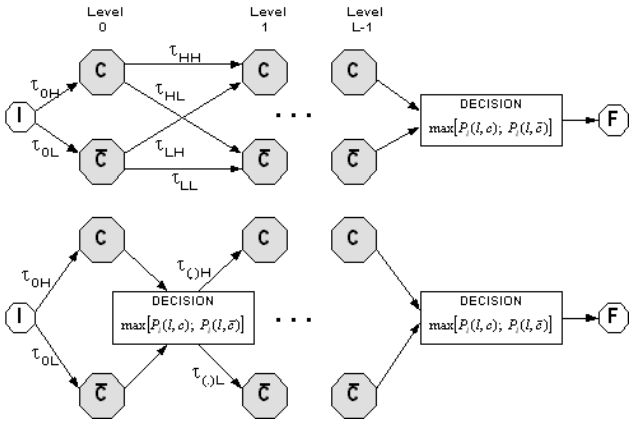

Fig. 3. Difference between Viterbi algorithm and LBA.

The first object of our comparison is the accuracy of both systems. Table 1 compares the results of the SDLBA system with those provided by the baseline system [3]. The baseline system and the proposed scheme differ only in the recognition scheme. The sole difference is that the baseline system uses a Viterbi algorithm together with a flat lexicon.

As we can see, the results for dynamic lexicons with 10, 100, 1,000 and 5,000 entries are approximately the same. Therefore, we can conclude that the local decision imposed by the SDLBA when compared with the global decision of the Viterbi, does not affect the global recognition accuracy.

Table 1. Recognition rates and speed for the SDLBA and baseline system (BLN).

Lexicon Entries	Recognition Rate (%)						Recognition Speed (w/m)	
	TOP1		TOP5		TOP10			
	SDLBA	BLN	SDLBA	BLN	SDLBA	BLN	SDLBA	BLN
10	98.76	99.08	99.93	99.93	100.00	100.00	218.21	208.13
100	95.46	95.74	98.82	99.01	99.40	99.44	24.34	22.54
1,000	89.00	89.37	95.48	96.21	96.81	97.45	2.80	2.28
5,000	82.26	81.38	91.10	-	93.54	-	0.66	0.47
10,000	78.22	76.31	88.49	-	90.99	-	0.36	0.24
20,000	74.54	-	85.69	-	88.94	-	0.20	0.12
30,000	71.61	-	84.02	-	87.06	-	0.15	0.09

If we compare the recognition speed of both systems, a significant improvement was obtained by the SDLBA system. As expected, the scheme based on the SDLBA and lexical tree structure is faster than the scheme based on Viterbi and flat lexicon, especially for large lexicons (>1,000 entries). Finally, to complete

our analysis, Figure 4 shows the behavior of the new scheme in terms of the reduction in the number of HMM states to be evaluated and the speed of the system. We can verify that both curves have similar behavior with a predicted gain in terms of speed according to the reduction of the search space.

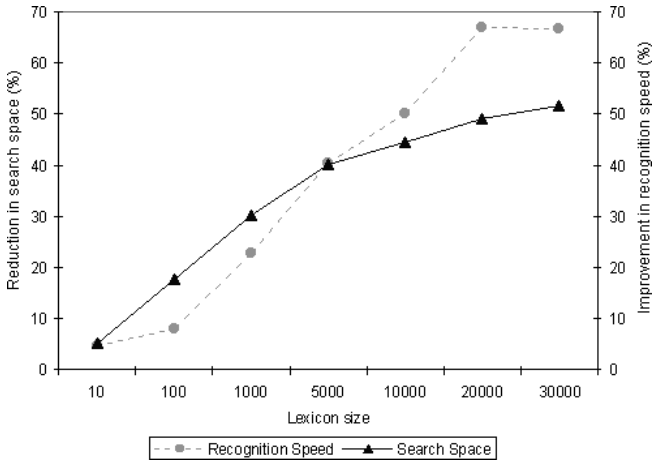

Fig. 4. Relation between the reduction in the search space and the recognition speed

6 Conclusion and Future Work

This paper has focused on the problems related to the computational efficiency of a handwritten word recognition system. The foremost concern addressed in this paper is the performance evaluation of the system considering different sizes of lexicons. We have verified that for medium and large lexicons (1,000 entries and more) such a system requires tens of seconds to recognize a handwritten word while its accuracy is also reduced gradually. In order to translate the gains made by the system in recognition accuracy into practical use, it is necessary to improve the computational efficiency. It is relatively easy to improve recognition speed while trading away some accuracy. Nevertheless, it is much harder to improve the recognition speed and preserve the original accuracy.

The second issue addressed in this paper is a comparison between two different approaches, one based on the LBA and other based on the Viterbi algorithm. In particular, a tree–based lexicon structure combined with a breadth–first search and a SDLBA have been employed successfully to optimize the search space and increase the recognition speed while preserving the recognition accuracy. However, searching even this reduced space still requires several tens of times.

The results motivate further investigations of other mechanisms to further reduce the search space and the computational load. Additionally, the structure of the recognition module facilitates the inclusion of contextual–dependent

models such as specialized models for the first characters of words, for the most common prefixes and suffixes, etc, as a means to improve the recognition rate. Such ideas will be the subject of our future research.

7 Acknowledgements

The authors wish to thank the Brazilian Council for Scientific Development and Technology (CNPq, Brazil) and the Ministére de Éducation du Québec (MEQ, Canada) which have supported this work and also the Service de Recherche Technique de la Poste (SRTP, France) for providing us the baseline system and the database.

References

1. R. M. Bozinovic and S. H. Srihari. Off–Line Cursive Script Word Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11 (1): 68–83, January 1989.
2. 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, May 1994.
3. A. El-Yacoubi, M. Gilloux, R. Sabourin and C. Y. Suen. An HMM Based Approach for Off–line Unconstrained Handwritten Word Modeling and Recognition. IEEE Transactions on PAMI, Vol. 21, No. 8, pp.752–760, 1999.
4. D. Guillevic and C. Y. Suen HMM–KNN Word Recognition Engine for Bank Cheque Processing. In 14th International Conference on Pattern Recognition, pages 1526–1529, Brisbaine, Australia, August 1998.
5. A. Kaltenmeier, T. Caesar, J. M. Gloger, and E. Mandler. Sophisticated Topology of Hidden Markov Models for Cursive Script Recognition. In 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japan, October 1993.
6. M. Koga, R. Mine, H. Sako, and H. Fujisawa. Lexical Search Approach for Character–String Recognition. In 4th Workshop on Document Analysis Systems, pages 237–251, Nagano, Japan, November 1998.
7. S. Manke, M. Finke and A. Waibel. A Fast Search Technique for Large Vocabulary On–line Handwriting Recognition. In Progress in Handwriting Recognition, pages 437–444. World Scientific, September 1996.
8. Hermann Ney. The Use of a One–Stage Dynamic Programming Algorithm for Connected Word Recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, 32 (2): 263–271, April 1984.
9. 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, June 1985.
10. T. Robinson and J. Christie. Time–First Search for Large Vocabulary Speech Recognition. In International Conference on Acoustics, Speech and Signal Processing, pages 829–832, Seattle, USA, May 1998.
11. 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, March 1998.