A Fast Lexically Constrained Viterbi Algorithm for On-line Handwriting Recognition

Alain Lifchitz, Frederic Maire

A FAST LEXICALLY CONSTRAINED VITERBI ALGORITHM FOR ON-LINE HANDWRITING RECOGNITION

ALAIN LIFCHITZ

Laboratoire d'Informatique de Paris 6, Université P6 & CNRS (UMR 7606), Case 169, 4, place Jussieu F-75252 Paris Cedex 05, France
E-mail: alain.lifchitz@lip6.fr

FREDERIC MAIRE

School of Computing Science, Queensland University of Technology, 2 George Street, GPO Box 2434 Brisbane, Q4001 Australia
E-mail: f.maire@qut.edu.au

Abstract : Most on-line cursive handwriting recognition systems use a lexical constraint to help improve the recognition performance. Traditionally, the vocabulary lexicon is stored in a trie (automaton whose underlying graph is a tree). In this paper, we propose a solution based on a more compact data structure, the directed acyclic word graph (DAWG). We show that our solution is equivalent to the traditional system. Moreover, we propose a number of heuristics to reduce the size of the DAWG and present experimental results demonstrating a significant improvement.

1 Introduction

Since the pioneering work of Vintsyuk [17] on *Automatic Speech Recognition* (ASR) systems, it is well known that *Hidden Markov Models* (HMM) [13] and *Dynamic Programming* (DP) [3], [12], provide a theoretical framework and practical algorithms for temporal pattern recognition with lexical constraints (even for large vocabularies). The techniques initially developed for ASR are also applicable to *Handwriting Recognition* (HWR), which shares many features with ASR especially if autosegmentation (from word to letter) is used. Most on-line cursive handwriting recognition systems use a lexical constraint to help improve the recognition performance. Traditionally, the vocabulary lexicon is stored in a trie (automaton whose underlying graph is a tree). Here we extend this idea with a solution based on a more compact data structure, the *Directed Acyclic Word Graph* (DAWG). After recalling some basics in the next section, we show that our solution is equivalent to the traditional systems in terms of the word recognized. Then, we propose a number of heuristics to reduce the size of the DAWG and present experimental results demonstrating a significant improvement over the traditional architecture. Our notation is standard and follows [13].

2 Basic tools

In the context of on-line HWR of isolated words, the input of the recognition system is a temporal sequence of feature vectors called frames. The *a posteriori* probability of a word given a sequence of frames is computed using a HMM. A word-HMM is made of the concatenation of the letter-HMM's corresponding to each letter of the word. We can abstract each word-HMM as an automaton whose underlying graph is a chain. Each transition of the automata is labeled with a letter (or variant, namely *allograph*) of the word. That is each transition corresponds to a letter-HMM. At the letter scale, HMM states correspond to feature stationarity of frames (subunits of letter, namely *graphems*).

The objective is, given a sequence of frames and a lexicon, find the word with the largest *a posteriori* probability in this lexicon. The computation of this *a posteriori* probability of a word reduces to a matching of elastic patterns : In the framework of the so-called maximum approximation, an efficient DP algorithm, namely *Viterbi Algorithm* (VA) [18], [4], is used.

A lexical constraint significantly helps to obtain better performance : Practical experiments on HWR neuro-markovian software REMUS [6], [19], [21], shows that the recognition of words increases from 20% to 90%-98%, depending of vocabulary size, when a lexical constraint is applied in a two phases processing.

Practical applications use lexicons with sizes ranging from 10 (digits recognition) to some 10^6 words (*e.g.* postcode dictionary, vocal dictation) [8]. Exhaustive application of VA to each word of the lexicon is only tractable for small and medium size lexicon, as the computational cost grows approximately linearly with the number of letters in lexicon. This limitation of the traditional approach can be overcome in two different ways :

Two-phase approach : Most conventional recognition systems use an unconstrained recognition in a first phase, then only in a second phase determine the word in the lexicon (with respect to an edition distance) closest to the string identified in the first phase. The reduction in running time is at the cost of a certain decrease of recognition performances [20], [7]: A tradeoff has to be found.

Factorization of the DP calculation : If two words have a common prefix then the DP computations of the *a posteriori* probabilities can be factorized. Hence, a speed-up and reduction in storage can be obtained simply by using a *trie* (a tree-like data structure known) [5]. Each node in the trie corresponds to a letter. As an exhaustive DP calculation is performed for all the words of the lexicon (unlike in the two-phase approach), the results are more accurate, and thanks to the sharing of intermediate results, the running time is also improved compared to the traditional approach. In HWR, this trick seems to have been used first by Manke *et al.* [10] for their Npen^{++} system. It is still the state-of-art in HWR and in ASR.

Using a trie has many advantages : a) Elimination of redundant computation/storage for common prefixes present in natural languages, b) easy implementation, c) it has also the nice property that the words of the lexicon and the

paths from the root to a terminal node are in bijection. The trie structure is a good tradeoff between simplicity and efficiency, and is widely used in practice.

Unfortunately we were disappointed by the poor compression ratio, from 1.5 to 4.2, dependent on languages (English/French) and vocabularies size (10^3 - 10^5 words), we got experimentally (second column of **Table 3**). Since practical applications, with large vocabulary, require very efficient processing, both in term of speed and storage, it is important to go further and extend the use of VA to more compact and complex lexicon structures, like DAWG. That is, use both prefix and suffix commonality [2], [15]. Lacouture *et al.* [9] and more recently Mohri *et al.* [11] have worked on similar problems with *Finite State Automata* (FSA) for ASR.

The *Fast Lexically Constrained Viterbi Algorithm* (FLCVA) is our answer to this challenge. Next we show how to construct a lexicon-HMM equivalent to a set of word-HMM's.

3 Conditions for equivalence of lexicon-HMM structures with respect to Viterbi Algorithm

To determine the word with the highest *a posteriori* probability among all the words belonging to a lexicon, we can organize the word-HMM's of the lexicon in different manners. We can either keep the word-HMM's separated (flat lexicon), or we can try to factorize all the common bits of the different word-HMM's

3.1 Parallel-HMM approach

In this first approach, we apply VA in parallel on each word-HMM of the lexicon, to compute $\delta_T(f,\lambda_{word_i}) = \max_{q_1,q_2,...,q_{T-1}} P(q_1,q_2,..,q_{T-1},q_T = e_f,O_1,O_2,..,O_T \mid \lambda_{word_i})$ where T is the number of frames and e_f is the final state of the i[th] word-HMM λ_{word_i}. The recognition system returns the word $word_i$ for which $\delta_T(f,\lambda_{word_i})$ is maximum.

3.2 Lexicon-HMM approach

In this second approach, we use only one big HMM. This lexicon-HMM is obtained from a FSA recognizing the lexicon, by replacing each transition arc of the automaton with the corresponding letter-HMM. When VA terminates, we obtain an optimal assignment of frames to states in the lexicon-HMM. This assignment is optimal in the sense that it maximizes the joint probability $P(q_1,q_2,..,q_{T-1},q_T = e_f,O_1,O_2,..,O_T)$ where e_f is the terminal state of the lexicon-HMM. The sequence $(q_1,q_2,...,q_T)$ of states induces a path in the associated automaton. This path defines the word the recognition system returns.

3.3 Equivalence of the two approaches

3.3.1 Conditions for equivalence

These two approaches will recognized the same word provided the two following conditions are satisfied : The first condition is that the letter-HMM's used for the same letter at different positions, in different words, are identical. In fact, we can consider a more general hypothesis, where instead of using the same single letter-HMM per letter, a same set of parallel letter-HMM's is used, to accommodate all the allographs that a writer can use for a same letter[1]. The second condition is that the transition probability from the final state of a letter-HMM to the start state of the following letter-HMM is the same for each letter-HMM and each word-HMM (recall that a word-HMM is simply the concatenation of letter-HMM's). Let p_u denote this universal transition probability. With these two simplifying assumptions, we can state that

Theorem
The lexicon-HMM approach and the parallel-HMM approach return the same word.

Sketch of Proof Recall that to build a lexicon-HMM, we begin with a finite automaton whose language is the lexicon. Then we substitute to each transition labeled with a letter x, a letter-HMM for the letter x. For each arc labeled y incident to the arc labeled x in the automaton, we insert a transition of probability p_u from the terminal state of the letter HMM for the letter x to the start state of the letter-HMM y in the lexicon-HMM.

The terminology “lexicon-HMM” is abusive as the probabilities of the transition leaving a terminal state of a letter HMM do not necessarily add up to one anymore in the new construct. But this does not pose any problem with respect to the determination of the optimal sequence of states by VA. First note, that if a sequence of states is present in the parallel-HMM, then an equivalent sequence is also present in the lexicon-HMM by construction. Reciprocally, each sequence of states in the lexicon-HMM can be viewed as the concatenation of sequences of states of the letter-HMM's (to which we can associate a sequence of states of a word-HMM of the parallel-HMM).

The equivalence of the two approaches results from the fact that DP, like VA, always computes the optimal assignment of frames to states that maximizes the overall score of a sequence of states. By construction of the lexicon-HMM from the parallel-HMM, the set of sequences of states of the lexicon-HMM and the set of

[1] Due to the reduction of the lexicon-HMM it is equivalent to label each allograph with different letter names. The same procedure can be used to cope with different context of the same allograph. This means that in fact the alphabet (set of unique letters) size increases.

sequences of states of the parallel-HMM are isomorphic. In the sense, that given a sequence of states in one HMM (and the unique word associated to this sequence), we can find a corresponding sequence of states in the other HMM such that this sequence has the same score and is associated to the same word. Thanks to this property, the maximum score for a sequence of states is the same in the two different HMM's (the lexicon-HMM and the parallel-HMM). ➤

Note that we have not made any hypothesis on the topology of the HMM. VA will compute the most likely sequence of states to yield a given sequence of observations whatever the probability matrices. In particular, VA will work with a word-HMM.

4 Lexicon word-graph reduction for the FLCVA

4.1 Goal of the heuristic

The goal differs from the classical problem of automata minimization [1], whose solution was improved by Revuz [14] for the special case of tries : the unique *Minimal Deterministic Finite Automaton* (MDFA) minimizes the number of states.

For our application the most interesting automaton is not this MDFA, but an automaton, not necessarily deterministic and unique, that has a minimal number of transition arcs : Here we want to minimize the DP computation of letter-HMMs, *i.e* their numbers, and we do not care about the number of states of the automaton (although the numbers of transitions and states are related). Moreover, the deterministic constraint is completely relaxed. It is why we have to design an *adhoc* reduction heuristic keeping of course invariant the recognized language (FSA terminology) or generated vocabulary. We use the term *reduction* to avoid confusion with *minimization*.

This reduction process will be applied directly on a different representation of the lexicon automaton, the DAWG. Each node is a letter. Edges are left-right admissible transitions between letter nodes.

4.2 The reduction heuristic

The reduction heuristic is based on an iterative use of two fusion modes of nodes, applied from right (the higher level) to left (the lower level) : grouping-fusion (**Figure 1**) and fission-fusion (**Figure 2**).

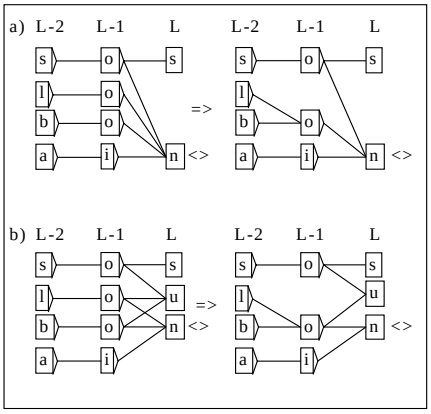

Figure 1. The grouping-fusion mode : a) most frequent case, b) general case.

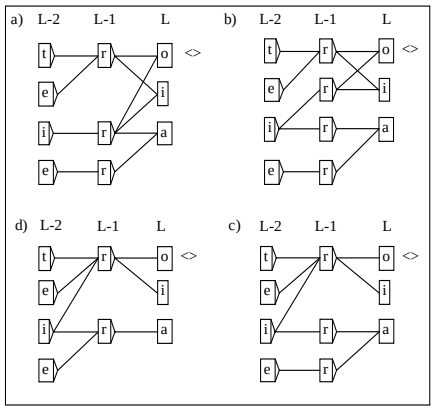

Figure 2. The fission-fusion mode : a) initial lattice, b) fission, c) first fusion, d) second fusion with one node less.

4.2.1 Grouping-fusion mode

This mode happens to be very similar to the one used in Revuz algorithm [14]. After a complete scanning of the current level the total number of nodes/edges is guaranteed to be less (if reduction occurs) or equal than before, leading to a stable convergent process. The algorithm begin by the highest level, scanning every node, like the current node "n" **Figure 1**a. Predecessors of the current node are sorted by label. Merging of predecessors is done if it does not change the local vocabulary (here only the two central nodes labeled "o"). Global link update is done before change of current node. **Figure 1**b exhibit a less frequent, more general and complex case : it illustrates how intra-level node coupling has to be managed.

4.2.2 Fission-fusion mode

This fusion mode, illustrated in **Figure 2**, is more complex than the previous one. It transforms the lattice in an equivalent lattice, trying in fact to prepare a favorable situation for the grouping-fusion mode. From the initial sub-lattice a) it produces a new virtual equivalent sub-lattice b), by fission of central node "r", predecessor of current node "o". Then compulsory merging of one new "r" nodes occurs leading to the actual sub-lattice c), with a constant net number of nodes. The only benefit is a much 'simpler' sub-lattice : reduction of the number of predecessors for the current node "o", reduction of the total numbers of successors links for the above level. It is, in general, a better context for a reduction by the grouping-fusion mode : This reduction effectively occurs in d), although it is not guarantee in general. For this mode, nothing can be said, in general, about the net total number of edges.

4.2.3 Reduction of the lexicon lattice

The consecutive six reduction steps of the lexicon lattice are : Flat, Trie, Leaves, Nodes, Inter-Level Leaves, Inter-Level Nodes. The **Table 1** summarizes the actual reduction algorithm, based on iterative application of both previous fusion modes for "Nodes" and "Inter-Level Nodes" (ILN) steps. Notice the mention "until no change" of the loops in the "Nodes" and "ILN" steps : It means that saturation in reduction is achieved during each subpart of the algorithm. "Nodes" and " ILN" steps are very similar, except that for the former some caution must be taken in the selection of node *n*, to avoid multiprocessing of the same nodes.

Figure 4 exemplifies the result of the algorithm at the maximum reduction on a toy flat lexicon of 7 words given in **Figure 3**. There is a reduction both in the number of nodes (from 57 to 36) and in the number of edges (from 50 to 38).

Flat	Initial flat lexicon
Trie	Make a trie from the flat lexicon
Leaves	For each level *L* do Merge leaves, with identical labels, of the current level.
Nodes	Loop until no change For each level *L* do (beginning with the highest) For each node *n* do (beginning with the leaves) Loop until no change Grouping-fusion; Fission-fusion
Inter-Level Leaves	Merge leaves, with identical labels, of all levels.
Inter-Level Nodes	Loop until no change For each level *L* do (beginning with the highest) For each node *n* do (beginning with the leaves) Loop until no change Grouping-fusion; Fission-fusion

Table 1. The six reduction steps of the lexicon lattice.

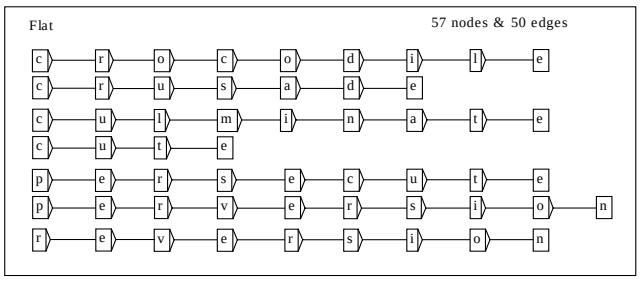

Figure 3. A flat toy lexicon of seven words.

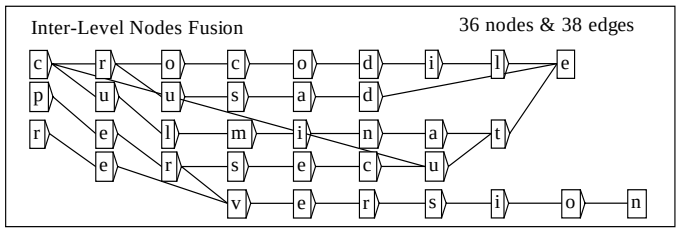

Figure 4. The most reduced (ILN) lexicon.

4.2.4 Experimental results and conclusion

For the testing of reduction heuristic, we have used five lexicons that can be downloaded from *http://www-poleia.lip6.fr/~lifchitz/HWR/flcva/lexs/*. The three first lexicons are lists of English words found in */usr/dict/words* on Unix systems (older release) and the remaining two are lists of French words extracted from the French lexicon coming with Unix spellchecker *ispell*. **Table 2** gives more details.

Lexicon	Nb of words	Nb of letters	Alphabet	$length_{min}$	$length_{ave}$	$length_{max}$
1 English	1000	6966	26	3	7.0	13
2 English	10645	78197	26	1	7.3	21
3 English	20233	149129	26	1	7.4	22
4 French	65536	631422	26	2	9.6	25
5 French	130499	1256938	28	2	9.6	25

Table 2. Brief descriptions of the five lexicons used for experiments.

The reduction times were measured on 233 MHz P. / 64 Mo PC under Windows 95.

Lexicon	Nb of nodes (trie) Nb of letters/*	Nb of nodes (ILN) Nb of letters/*	Nb of edges (ILN) Reduction time (seconds)
1	4753	2097	3050
	1.47	3.32	1.3
2	40683	10567	21151
	1.92	7.40	5.9
3	66308	13773	33847
	2.25	10.83	11.1
4	193582	18147	70290
	3.26	34.79	124.4
5	297701	17908	93574
	4.22	70.19	564.1

Table 3. Experimental reduction results for the five lexicons (**Table 2**). The number of edges of the trie structure is equal of its number of nodes.

Although the complexity of the minimization of the number of transitions of a non deterministic automaton is still an open problem, the heuristic proposed leads to a significant improvement for real-world vocabularies. Personal Digital Assistants (PDA) and other smart handheld devices have too modest resources (a relatively small storage capacity and slow CPU) to allow features like advanced user interfaces (natural interactivity) : Nevertheless efficient use of these limited resources will permit sophisticated HWR. A much more interesting reduction ratio occurs for ILN (70.19) than for the trie (4.22) in case of 130K words French vocabulary. Interestingly, the reduction ratio seems to increase with the lexicon size. Some recognition systems, especially for mobile computers, need the functionality of incremental updating of vocabulary (add/remove words). Our DAWG structure allows such adaptive update avoiding the computation from scratch of the reduction of the slightly modified lexicon.

The implementation of VA with DAWG requires special attention, but lack of space prevents us to discuss this issue here (how backtracking and path history management are performed will be explained in a forthcoming paper).

5 Acknowledgements

We thank the authors of graph Viewer/Editor FSGraph [16] for making available for free their software, Dr. Zsolt Wimmer for stimulating discussions and providing the first version of the REMUS software, Prof. Patrick Gallinari for introducing one of us (AL) to HWR and the French Government for sponsoring the other (FM) with a "Visiting Scientist Fellowship" during his visit of the LIP6. Online copy of this paper at *http://webia.lip6.fr/~lifchitz/FLCVA/*.

References

1. Aho A. V., Hopcroft J. E. and Ullman J. D., "The Design and Analysis of Computer Algorithms", *Addison-Wesley*, Reading M.A., 1974.
2. Appel A. W. and Jacobson G. J., "The world's fastest scrabble program", *Communications of the ACM*, Vol. **31**, No. 5, pp. 572-578 & 585, May 1988.
3. Bellman R., "Dynamic Programming", *Princeton University Press*, 1957.
4. Forney Jr D. G., "The Viterbi Algorithm", *Proceedings of the IEEE*, Vol. **61**, No 3, pp. 268-278, March 1973.
5. Fredkin E., "Trie Memory", *Communications of the ACM*, Vol. **3**, No 9, pp. 490-499, September 1960.
6. Garcia-Salicetti S., "Une approche neuronale prédictive pour la reconnaissance en-ligne de l'écriture cursive", *Thèse de Doctorat Paris 6, Spécialité: Informatique*, 17 décembre 1996.

7. Gilloux M., "Réduction dynamique du lexique par la méthode Tabou", *1er Colloque International Francophone sur l'Ecrit et le Document (CIFED'98)*, Québec (Canada), pp. 24-31, 11-13 mai 1998.
8. Kosmala A., Willett D. and Rigoll G., "Advanced State Clustering for Very Large Vocabulary HMM-based On-Line Handwriting Recognition", *ICDAR'99*, Bangalore (India), pp. 442-445, 20-22 September 1999.
9. Lacouture R. and De Mori R., "Lexical Tree Compression", *Eurospeech'91*, Genova (Italy), pp. 581-584, September 1991.
10. Manke S., Finke M. and Waibel A., "A Fast Search Technique for Large Vocabulary On-Line Handwriting Recognition", *IWFHR5*, Colchester (England), pp. 183-188, 2-5 September 1996.
11. Mohri M. and Riley M., "Network optimizations for large-vocabulary speech recognition", *Speech Communication*, Vol. **28**, pp. 1-12, 1999.
12. Ney H., "Dynamic programming as a technique for pattern recognition", *ICPR'82*, Munich (Germany), pp. 1119-1125, October 1982.
13. Rabiner L. R., and Juang B.-H., "*Fundamentals of Speech Recognition"*, Ed. Prentice Halls, pp. 321-389, 1993.
14. Revuz D., "Minimization of acyclic deterministic automata in linear time"., *Theoretical Computer Science*, Vol **92**, pp. 181-189, 1992.
15. Ristov S. and Laporte E., "Ziv Lempel Compression of Huge Natural Language Data Tries Using Suffix Arrays", *Proceedings of Combinatorial Pattern Matching, 10th Annual Symposium*, Warwick University, UK, M.Crochemore and M.Paterson (editors), Berlin: Springer, pp. 196-211, July 1999.
16. Silberztein M., "INTEX: a Finite State Transducer toolbox", *Proceedings of the 2nd International Workshop on Implementing Automata (WIA'97)*, Springer Verlag, 1998. *http://www-igm.univ-mlv.fr/~unitex/download.html*
17. Vintsyuk T. K., "Recognition of words in spoken speech by dynamic-programming methods", *Kibernetika*, Vol. **4**, No 1, pp. 81-88, January 1968.
18. Viterbi A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Theory*, IT-**13**, pp. 260-269, April 1967.
19. Wimmer Z., "Contribution à la lecture de documents papiers manuscrits: reconnaissance des champs numériques et cursifs par méthodes neuronales markoviennes", *Thèse de Docteur-Ingénieur ENST de Paris, Spécialité: Informatique*, 28 septembre 1998.
20. Wimmer Z., Dorizzi B., Gallinari P., "Dictionary Preselection In a Neuro-Markovian Word Recognition System", *ICDAR'99*, Bangalore (India), pp. 539-542, 20-22 September 1999b.
21. Wimmer Z., Garcia-Salicetti S., Lifchitz A., Dorizzi B., Gallinari P., Artières T., "REMUS", *http://webia.lip6.fr/~lifchitz/HWR/*, January 1999a.