COVER SHEET

Reduction of Non Deterministic Automata for Hidden Markov Model Based Pattern Recognition Applications

Frederic Maire[1], Frank Wathne[1] and Alain Lifchitz[2]

[1] Smart Devices Laboratory, Faculty of Information Technology, Queensland University of Technology,
2 George Street, GPO Box 2434, Brisbane Q4001 Australia
f.maire@qut.edu.au, frankwathne@hotmail.com
[2] Laboratoire d'Informatique de Paris 6, Université P6 & CNRS (UMR 7606),
8, rue du Capitaine Scott, F-75015 Paris France
alain.lifchitz@lip6.fr

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 a previous paper, we showed that non-deterministic automata were computationally more efficient than tries. In this paper, we propose a new method for constructing incrementally small non-deterministic automata from lexicons. We present experimental results demonstrating a significant reduction in the number of labels in the automata. This reduction yields a proportional speed-up in HMM based lexically constrained pattern recognition systems.

1 Introduction

Since the pioneering work of Vintsyuk [16] on automatic speech recognition systems, Hidden Markov Models (HMM) [12] and Dynamic Programming (DP) [3], [11], have provided a theoretical framework and practical algorithms for temporal pattern recognition with lexical constraints (even for large vocabularies). The techniques initially developed for speech recognition are also applicable to on-line handwriting recognition (especially if auto-segmentation 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). We have previously extended this idea with a solution based on a more compact data structure, the Directed Acyclic Word Graph (DAWG) [9]. In this paper, we propose a new construction algorithm that allows an incremental building of small non-deterministic automaton. Moreover, this new automaton is more compact than previously proposed automata. After recalling briefly the basics of lexically constrained pattern recognition problems in Section 2, we will describe taxonomy of automata in Section 3. In Sections 4 and 5, we review standard reduction techniques for automata. In Section 6, we propose new heuristics to reduce node-automata. Experimental results demonstrating significant improvements are presented in Section 7. Our notation is standard and follows [12].

2 Lexically Constrained Pattern Recognition

A number of pattern recognition problems like hand gesture recognition, on-line hand writing recognition and speech recognition can be solved by performing an elastic matching between an input pattern and a set of prototype patterns. In all these applications, an a posteriori probability of a word given a sequence of frames (feature vectors) is computed using a HMM.

2.1 Word-HMM, Letter-HMM and Viterbi Algorithm

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 automaton is labelled 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 graphemes). The objective of the lexically constrained pattern recognition problem 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 [17], [4], is used. A lexical constraint significantly helps to obtain better performance; practical experiments on a neuro-Markovian pattern recognition software called REMUS [6], [18], [19], shows that the recognition of words increases from 20% to 90%-98%, depending on the size of the vocabulary, when a lexical constraint is applied. Practical applications use lexicons with sizes ranging from 10 (digits recognition) to some 10^6 words (e.g. postcode dictionary, vocal dictation) [7]. Exhaustive application of Viterbi Algorithm 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.

2.2 Factorization of HMMs into Non-Deterministic Automata

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 well known tree-like data structure) [5]. Each edge/node in the trie corresponds to a letter. Thanks to the sharing of intermediate results, the running time has to improve dramatically compared to the trivial approach consisting in running Viterbi Algorithm independently on each word-HMM.

A trie eliminates the redundant computation/storage for common prefixes present in natural languages and is easy to implement. The trie structure is a good trade-off between simplicity and efficiency, and is widely used in practice. Unfortunately we were disappointed [9] 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. 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 Viterbi Algorithm to more compact and complex lexicon structures, like DAWG. That is, use both prefix and suffix commonality [2], [14]. Lacouture et al. [8], and more recently Mohri et al. [9], have worked on similar problems with Finite State Automata (FSA) for Automatic Speech Recognition.

The automata that we build are not traditional deterministic automata. This choice is motivated by the following observations; traditional automata are graphs whose arcs have labels. Each arc is labelled with a letter. The nodes/states of the automata are not labelled. The nodes correspond to languages. It is natural to wonder whether putting the labels in the nodes instead of in the arcs would improve the compactness of the automata. The main computational cost of running Viterbi algorithm on a graph is a function of the number of labels in the graph. Hence the importance of finding a representation that minimizes the number of labels. Moreover, Viterbi algorithm does not require a deterministic automaton. We call *node-automaton* a directed graph whose nodes can be labelled with letters. The arcs of a node-automaton are transitions with no label. A transition of a node-automaton is just a routing device. Node-automata are better for HMM factorization because in a node-automaton the processing is done in the node and the routing is done with the arcs. Whereas with traditional automata (that we call *edge-automata*), these two tasks are not separated. Experimental results demonstrate a clear superiority of node-automata over edge-automata with respect to the computational cost of running Viterbi Algorithm on a whole lexicon (see Section 7).

3 Automata Taxonomy

We have experimented with two types of acyclic automata. They differ only in that the edge automata labels are stored in the edges, whereas in node-automata the labels are stored in the nodes. In order to describe the reduction algorithms we recall here the standard definitions we need throughout the rest of the paper.

A *finite state automaton* is a quintuple (Q, E, Σ, I, T) where Q is a set of states (nodes), Σ is an alphabet (a set of symbols), E is a set of *transitions* (*directed edges* or *arcs*), I is a set of *initial states* and T is a set of *terminal* (*accepting*) states. The automata discussed in this paper will have a single initial state called the *root* node. For some of the algorithms presented here, there will be a single terminal node referred to as the *sink* node. A quintuple $(Q, E, \Sigma, \{r\}, \{s\})$ will denote an automaton with a single root r and a single terminal node s. A path of length k is a sequence of nodes $(n_0, n_1, \ldots, n_k)$ where each successive pair of nodes in the sequence is connected by a transition. If (n_i, n_j) is a transition, then we say that n_i is a predecessor of n_j, and n_j a successor of n_i. We will denote the set of successors of n_i by $\mathrm{succ}(n_i)$. Similarly, $\mathrm{pred}(n_i)$ will denote the set of predecessors of n_i. The node

n_j is *reachable* from the node n_i, if there exists a path from n_i to n_j. In this case, we also say that n_j is a *descendant* of n_i.

Each ordered pair of nodes (n_i, n_j) implicitly defines a *language* (a set of words) denoted by $L_Q(n_i, n_j)$. Each sequence of labels encountered along a path from n_i to n_j makes up a word. $L_Q(n_i, n_j)$ is the language generated by all possible paths between n_i to n_j. More generally, $L_Q(A, B)$ will denote the language defined by $L_Q(A,B) = \bigcup_{(n_i,n_j)\in A\times B} L_Q(n_i, n_j)$. The language recognized by the automaton is $L_Q(I,T)$.

4 Automata Reduction

A key concept for automata minimization is the *contraction* (or *merging*) of equivalent nodes. We will first characterize useful equivalence relations, and then explain how they allow the merging of nodes.

4.1 Equivalence Relation and Node Contraction

Many equivalence relations can be defined on Q. Recall that an equivalence relation R on Q can be viewed as a partition of Q. Two nodes are equivalent with respect to R if and only if they belong to the same part of the partition of Q. Obviously, to be of any interest, the reduction operation must preserve the language of the automaton. That is, the reduced automaton should generate the same language as the original automaton. A sufficient condition for this to happen is that any two equivalent nodes with respect to R generate the same language. If an equivalence relation R satisfies this sufficient condition, we will say that R is *admissible*.

Formally, the contraction of two equivalent nodes n_i and n_j with respect to R will preserve the language of the automaton provided that

$$n_i R n_j \Rightarrow L_Q(n_i, T) = L_Q(n_j, T).$$

Whenever two nodes satisfy $L_Q(n_i, T) = L_Q(n_j, T)$, we will use the standard terminology, and say that the nodes n_i and n_j are *indistinguishable*. Our NFA minimization algorithm computes an equivalence relation between nodes that entails indistinguishability.

The contraction of a set of nodes A is a new node a' obtained by merging all nodes of A. The new automaton (after contraction of the set of nodes A) is obtained by removing all nodes belonging to A, then inserting a new node a', and connecting a' to all successors and predecessors of the nodes that were in A. Formally, $\text{succ}(a') = \bigcup_{n_i \in A} \text{succ}(n_i)$ and $\text{pred}(a') = \bigcup_{n_i \in A} \text{pred}(n_i)$. The merging of two nodes n_i and n_j is just a special case of set of nodes contraction where $A = \{n_i, n_j\}$.

We can extend the equivalence relation to sets of nodes. Two sets of nodes A and B are said to be equivalent if the two nodes a' and b' are equivalent.

4.2 Other Relevant Relations defined on the Nodes

The type of equivalence relations we have defined so far is also referred to as *down-equivalence* because they only consider the descendants. We may also define *up-equivalence* and *up-indistinguishability* in a similar way by considering the reversed automaton. The reversed automaton is obtained by first swapping the set of initial states and the set of terminal states, then reversing all the transitions. That is, (n_i, n_j) is a transition in the reversed automaton if and only if (n_j, n_i) is a transition in the original automaton. The reversed automaton generates the mirror language of the original automaton.

When two nodes have exactly the same successors (which implies that they are indistinguishable), we say that these nodes are *similar*. *Subsumption* is a relationship more general than similarity. The node n_i subsumes the node n_j if and only if $\text{succ}(n_i) \supseteq \text{succ}(n_j)$. That is, every successor of n_j is also a successor of n_i. For node-automata we add the requirement that n_i and n_j have the same label. Two nodes are said to be *comparable* if one subsumes the other.

Automata minimization algorithms need sometimes to consider the *height* and the *depth* of a node. The height of a node is the length of the longest path from the node down to a leaf node. The depth of a node is the length of the longest path from the root to the node. A *level* is the set of all nodes of a given height. Levels are important because indistinguishable nodes must have the same height.

4.3 Contraction and Split Operators

In order to reduce the automata, we use mainly two node operators that work on pairs of nodes. The contraction operator merges two nodes and was defined in Section 4.1.

The *split* operator is used for splitting a node into two nodes to prepare the automaton for further reduction. The new automaton obtained by splitting a node n is derived from the original automaton by replacing n with two nodes n' and n'' such that $\text{pred}(n) = \text{pred}(n') \cup \text{pred}(n'')$ and $\text{succ}(n') = \text{succ}(n'') = \text{succ}(n)$.

It is easy to see that

- Split operations do not change the language of the automaton,
- Contraction operations do not change the language of the automaton provided they are applied on equivalent nodes with respect to an admissible equivalence relation,
- After a split operation the new automaton will always be non deterministic.

5 Previous Works

We review in this section the main algorithms to minimize deterministic and non-deterministic automata.

5.1 Minimal Deterministic Automata

The classical algorithm for deterministic automata minimization [1] first computes the indistinguishable equivalence classes, and then contracts separately all these equivalence classes. In the special case of acyclic automata defined by a lexicon, a faster algorithm to build the minimal deterministic automaton was proposed by Revuz [13].

The algorithm of Revuz starts with the construction of the trie of the lexicon. Then the trie is traversed either level-by-level, starting with the leaves, or in a post order fashion. During the traversal the similarity classes are determined: the node currently considered is either equivalent to some other already visited node, or will become the representative of a new equivalence class. In the former case the current node is merged with the representative of that class. The bottom-up traversal ensures that all successor nodes of the current node have already been reduced. Therefore during the execution of the algorithm, two nodes on a same level will be indistinguishable if and only if they are similar (that is have the successors in the current automaton).

We have implemented and tested this algorithm for the classical edge-automata as well as for the node-automata.

5.2 Compact Non-Deterministic Automata

The algorithm proposed by Sgarbas, Fakotakis and Kokkinakis [15] incrementally build a small non-deterministic automaton. We will describe the main ideas of this algorithm. The original algorithm was created for edge-automata, but we have adapted it to node-automata. The automaton returned by the algorithm is called a

compact NFA (*CNFA*). The main feature of a CNFA is that it does not contain similar nodes.

The algorithm expects as input a NFA with a single root and a single sink node. Words are inserted incrementally in the automaton. A word insertion is made in two phases. In the first phase, a chain of new nodes corresponding to the new word is added to the automaton. The chain starts at the root node and finishes at the sink node. All the intermediate nodes are new. In the second phase, similar nodes are detected then merged in order to keep the automata compact. This is done by first traversing the automaton starting from the sink node to identify nodes that are down-similar with nodes of the newly inserted word. When no more such nodes can be found the procedure is repeated from the root, this time looking for up-similar nodes. When no more up-similar nodes are found, we continue looking from the bottom again. This procedure is repeated while we find either down-similar or up-similar nodes. The resulting automaton does not contain any similar nodes.

The CNFA algorithm provides good compression results, but further reduction can be obtained with the heuristics that we will describe in the next section.

6 Compressed Non-Deterministic Automata

We propose a new heuristic, generalization of the one of [9], relying on the split and contraction operators, to further reduce NFAs. The equivalence relation R_k that we use for the contraction operator is original. For deterministic automata, we have seen that we can determine easily whether two nodes are indistinguishable by comparing their successors. For non-deterministic automata, the situation is more complicated. Let $\mathrm{succ}(n,x)$ denote the successors of node n with the label x. $\mathrm{succ}(n,x)$ can be a large set whereas it contains at most one node for deterministic automata. Next, we define an admissible equivalence relation with a given look-ahead depth. Then, we will outline our NFA compression algorithm.

6.1 The equivalence relation R_k

We define recursively the following equivalence relations R_k:

- R_0 is the similarity relation between nodes (their successors are required to be the same),
- Node n_i and node n_j are in relation with respect to R_k if and only if all the following conditions are satisfied
  - The nodes n_i and n_j have the same height,
  - The nodes n_i and n_j are both either terminal or non terminal.

– The nodes n_i and n_j have the same set of labels for their outgoing transitions (but they are allowed to have a different number of transitions for a given label),
– For each common transition label x, $\mathrm{succ}(n_i, x)$ is equivalent to $\mathrm{succ}(n_j, x)$ with respect to R_{k-1}.

The last condition concerns the equivalence between two sets of nodes. This was defined in Section 4.1.

6.2 The NFA compression heuristic

Our heuristic shuttles through the graph level-by-level starting at the deepest one. The skeleton of the heuristic is;

```
Loop until no change in the automaton
   Compute the heights of all nodes
   For each level
      Contract all equivalent nodes
      Separate all comparable nodes
   End for
End loop
```

In the deterministic case, the above algorithm will produce the unique minimal DFA. For non-deterministic automata, we know that there may be several minimal automata. Moreover the order in which we contract the nodes does matter. Blindly removing equivalent nodes in a NFA may lead to a sub-optimum automaton. The more reduced is level $(k+1)$, the more difficult it will be to reduce level k. To increase the likelihood of further contractions, we use the split operator to separate comparable nodes (see Section 4.2). Although splitting nodes increases the number of nodes on the current level, it makes more likely the creation of equivalent nodes in the predecessor level. Overall, there is a significant reduction in the number of nodes as demonstrated with the results of next section.

7 Experimental Results

We have run the different reduction algorithms on two families of artificial lexicons that have a known minimal NFA and DFA. Another purpose of these lexicons was to validate the implementation of the algorithms. We have also tested the algorithms on edge and node automata with different size English and French lexicons. Table 1 gives some details on the lexicons used.

Lexicons

Name	# words	# letters	Alphabet size	Mean length	Max length	Type
DNA_4_8_1.txt	87380	669924	4	7.5	8	Artificial
HWR_4_8_1.txt	13120	98416	4	7.7	8	Artificial
Lex1000.txt	1000	6966	26	7	13	English
Lex10645.txt	10645	78197	26	7.3	21	English
Lex20233.txt	20233	149129	26	7.4	22	English
Lex65536f.txt	65536	631422	26	9.6	25	French
Lex130499.txt	130499	1256938	28	9.6	25	French

Table 1 Features of the lexicons

DNA_4_8_1 is the language of all words from 1 to 8 letters, on a alphabet of 4 letters, consecutive letters being required to be distinct. The HWR language differs from the DNA language by relaxing the constraint on consecutive letters.

	Edge Automata					
	Compact NFA (Edge Automata)		Compressed NFA (Edge Automata)		Minimal DFA (Edge Automata)	
Lexicon	Nodes	Transitions	Nodes	Transitions	Nodes	Transitions
DNA_4_8_1.txt	30	128	30	128	30	88
HWR_4_8_1.txt	9	60	9	60	9	32
lex1000.txt	1433	2433	1427	2424	1462	2459
lex10645.txt	7220	17864	7675	17119	9076	18294
lex20233.txt	9463	29523	10460	26867	13685	28741
lex65536f.txt	14102	68740	17157	60479	20949	44980
lex130499.txt	15331	99808	17237	68783	20928	47732

Table 2 Comparison of edge-automata

Table 2 shows that for edge-automata, compressed NFA have fewer labels (transitions) than compact NFA. But for large lexicons minimal DFA contains fewer labels.

	Node Automata					
	Compact NFA (Node Automata)		Compressed NFA (Node Automata)		Minimal DFA (Node Automata)	
Lexicon	Nodes	Transitions	Nodes	Transitions	Nodes	Transitions
DNA_4_8_1.txt	33	127	33	127	33	88
HWR_4_8_1.txt	33	162	33	162	33	116
lex1000.txt	2100	3077	2099	3076	2105	3081
lex10645.txt	10693	21309	10569	21174	11974	21248
lex20233.txt	14142	34229	13781	33825	17282	32579
lex65536f.txt	20559	72733	18251	65605	25075	50439
lex130499.txt	21932	102276	17476	81304	24968	53255

Table 3 Comparison of node-automata

Table 3 shows that for node-automata, compressed NFA have significantly fewer labels (nodes) than the other types of automata. With respect to the complexity of the compression algorithm, it is clear that the process of finding all indistinguishable states may sound computationally costly as the algorithm recursively checks nodes down the automaton. However, we discovered that if you are scanning an automaton with few indistinguishable states, the recursion rarely goes deeper than 2 or 3 levels. In fact, a practical solution consists in first building a compact NFA, then further reducing the NFA with our compression algorithm.

8 Conclusions

Figure 1 (below) illustrates clearly the benefit of using node-automata instead of edge-automata; the number of labels of the node-automata is a fraction of the number of labels of the edge-automata. Although our minimization method does not claim to return the optimal non-deterministic automaton, the heuristics proposed in this paper 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 speech or hand writing recognition. Some recognition systems, especially for mobile computers, need the functionality of incremental updating of vocabulary (add/remove words). Our NFA construction allows such adaptive update, avoiding the recomputation from scratch of the minimized NFA of the slightly modified lexicon (so far, we have only implemented the addition of new words).

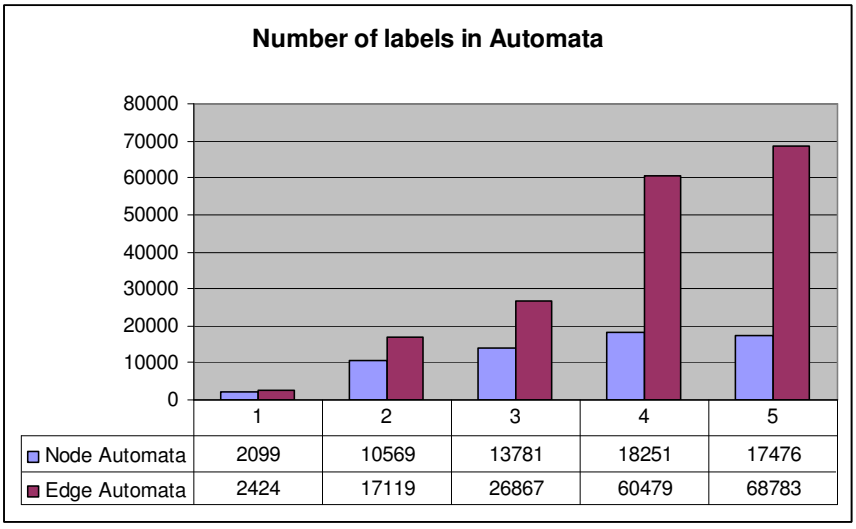

Figure 1 Comparison of the numbers of labels for compressed real world lexicons

9 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. 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.
8. Lacouture R. and De Mori R., "Lexical Tree Compression", Eurospeech'91, Genova (Italy), pp. 581-584, September 1991.
9. Lifchitz, A. and Maire F., "A Fast Lexically Constrained Viterbi Algorithm For Online Handwriting Recognition", In: L.R.B. Schomaker and L.G. Vuurpijl (Eds.), Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, ISBN 90-76942-01-3, Nijmegen: International Unipen Foundation, pp 313-322, September 11-13 2000.
10. Mohri M. and Riley M., "Network optimizations for large-vocabulary speech recognition", Speech Communication, Vol. 28, pp. 1-12, 1999.
11. Ney H., "Dynamic programming as a technique for pattern recognition", ICPR'82, Munich (Germany), pp. 1119-1125, October 1982.
12. Rabiner L. R. and Juang B.-H., "Fundamentals of Speech Recognition", Ed.Prentice Halls, pp. 321-389, 1993.
13. Revuz D., "Minimization of acyclic deterministic automata in linear time"., Theoretical Computer Science, Vol 92, pp. 181-189, 1992.
14. 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.
15. Sgarbas K. N., Fakotakis N. D. and Kokkinakis G. K., "Incremental Construction of Compact Acyclic NFAs". Proceedings ACL-2001, 39th Annual Meeting of the Association for Computational Linguistics, Toulouse (France), pp. 474-481, July 6-11 2001.
16. Vintsyuk T. K., "Recognition of words in spoken speech by dynamicprogramming methods", Kibernetika, Vol. 4, No 1, pp. 81-88, January 1968.
17. 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.
18. 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.
19. Wimmer Z., Garcia-Salicetti S., Lifchitz A., Dorizzi B., Gallinari P., Artières T.,"REMUS", January 1999a
    http://www-poleia.lip6.fr/CONNEX/HWR/