Two Different Approaches for Cost-Efficient Viterbi Parsing with Error Correction*

Juan C. Amengual[1] and Enrique Vidal[2]

[1] Unidad Predepartamental de Informática
Universidad Jaume I, 12071 Castellón, SPAIN
[2] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, 46071 Valencia, SPAIN
e-mail: jcamen@inf.uji.es evidal@iti.upv.es

Abstract. The problem of Error-Correcting Parsing (ECP) using a complete error model and a Finite State Machine (FSM) is examined. This problem arises in many areas of Linguistic and Speech Processing, and is of paramount importance in Syntactical Pattern Recognition, where data is generally distorted or noisy. The Viterbi algorithm can be easily extended to perform ECP using a trellis diagram that has the same number of states as that of the FSM. However, the computational complexity of the ECP process could be prohibitive for real-time pattern recognition tasks. Two different approaches to perform an efficient implementation of such a parsing are suggested. The first one is an adaptation of an extension of the Viterbi algorithm proposed in the literature. In the second one, an algorithm based on a depth-first ("topological") sort of the states of the FSM, which leads to an efficient processing of the deletion transitions of the underlying error model, is proposed. Experiments are described with results assessing the relative merits of the different techniques.

1 Introduction

Error-Correcting Parsing (ECP) techniques have been widely used in the field of Syntactic Pattern Recognition [7] [9] [14] in tasks such as Automatic Speech Recognition [16] and OCR [12]. The success of these techniques lies on their *intrinsic* capacity to overcome the errors frequently produced in the representations of real-world patterns acquired through a noisy and/or distorted channel.

Under the scheme of ECP, we assume that we have a structural model, typically a (stochastic) Finite State Machine (FSM) associated to a Regular Grammar, and an (stochastic) error-correcting model, typically a functionally complete one, which takes insertions, substitutions and deletions of symbols belonging to some alphabet Σ into account. Σ stands for the set of primitives or features which uniquely characterice the given pattern we aim to recognize. Therefore, a string of symbols belonging to Σ represents the acquisition of a given object through some device. The FSM stands for the set of different strings

* Work partially supported by the Spanish CICYT under contract TIC93-0633-CO2-01.

which corresponds to the several ways the given object was acquired through this device. Finally, the error model accounts for the errors likely to be produced in the acquisition and/or feature extraction phases.

If no error-correcting model is given, the problem of recognition can be seen as a problem of simple parsing. Given an input string of symbols, we have to determine if this string belongs to the language generated by the FSM. In case it belongs, then we classify the object associated to this input string as an object modeled by the FSM. If the model is deterministic, this parsing is trivial. Otherwise, the Viterbi algorithm [6] is used to solve this problem. The same framework can be adopted for ECP, if an error-correcting model is provided, at the expense of a higher computational cost. Nevertheless, this higher cost could become prohibitive for real-time pattern recognition tasks.

In the next section we identify the computational problem posed by *deletion transitions*. In Sects. 3 and 4 it is shown how can this problem be efficiently solved through two different approaches. In Sect. 5 we adapt the well known Beam Search technique [11] to further accelerate the parsing process. Sect. 6 details the experiments that have been carried out to test the performance of both approaches. Finally, Sect. 7 establishes some conclusions from the observed results.

2 The Computational Problem Posed by Deletion Error Transitions

The problem of parsing with no error correction can be formulated as a search for the "minimum cost" path through a *trellis diagram* (see Fig. 1) associated to the FSM model and the given input string, a. The trellis diagram produced by the Viterbi algorithm is a directed *acyclic* multistage graph, where each node q_k^j corresponds to an state q_j in a given time interval (or stage) k. The stage k is associated with a symbol, a_k, in the string to be parsed and every arc, $t_k = (q_k^i, q_{k+1}^j)$, stands for a transition between the state q_i in stage k and the state q_j (it can be the same state) in stage $k+1$ (next time interval).

The trellis diagram can be straightforwardly extended to parse errors produced by changing a symbol for another symbol and errors produced by inserting a symbol before or after each symbol in the original string. In this way, an efficient error-correcting parser, taking both, *substitution* and *insertion* errors, into account can be implemented. This is due to the fact that such an extended trellis diagram still has the shape of a directed *acyclic* multistage graph, and the problem of finding a minimum cost path through this kind of graphs is essentially the problem solved by the Viterbi algorithm.

Now, we would like to extend this trellis diagram to parse also errors produced by *deletion* of one or more (consecutive) symbol(s) in the original string. Unfortunately, the resulting trellis diagram will no longer have the shape of a multistage graph since we have edges between the nodes belonging to the same stage k. Nevertheless, if the FSM model has no cycles, then this problem could be reduced to find a minimum cost path through a directed *acyclic* graph and,

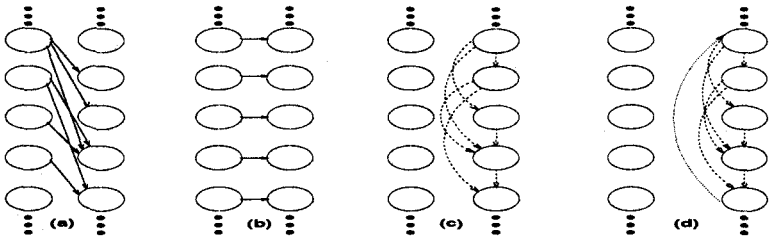

Fig. 1. Trellis with: a) Substitution error and proper FSM transitions b) Insertion error transitions. c) Deletion error transitions in an *acyclic* FSM. d) Deletion error transitions in a *cyclic* FSM

since the insertion and substitution transitions still fulfill the multistage graph conditions, an efficient algorithm can be implemented as an extension of the original Viterbi algorithm [2].

But if the model can be *any* FSM (associated to a *general* regular grammar) possibly with cycles, then the problem reduces to that of finding a minimum cost path through a general directed *cyclic* graph. In principle, finding a smallest cost path through such a kind of graphs would be significantly more expensive than the original Viterbi strategy.

3 Solving the Problem by Score Ordering

We present here an algorithm (EV2) based on a recurrence relation stated in [4]. In our notation this relation can be stated as follows:

$$C(q_{k+1}^j) = \min_{\substack{\forall i \in \delta'(q_l) \text{ in stage } k \\ \forall l \text{ in stage } k+1}} \left\{ C(q_k^i) + W(q_k^i, q_{k+1}^l) + \zeta(q_{k+1}^l, q_{k+1}^j) \right\} \quad (1)$$

where:

1. $C(q_k^i)$ stands for the cost of the minimum cost *path* from any of the initial states to state q_i in stage k.
2. δ' stands for the *inverse* of the transition function, δ, of the FSM.
3. $W(q_k^i, q_{k+1}^l)$ stands for the cost of the minimum cost *transition* that connects state q_i in stage k with state q_l in stage $k+1$.
4. $\zeta(q_{k+1}^l, q_{k+1}^j)$ stands for the cost of the minimum cost *path* that connects state q_l with state q_j, both in stage $k+1$.

This relation states that the minimum cost path to state j in stage $k+1$, $C(q_{k+1}^j)$, is the *minimum cost* of all the paths which involve a state q_i (predecessor of q_l) in stage k and a state q_l in stage $k+1$. The properness of this relation lies on the function ζ, since its computation yields, for all pair of states

```
INPUT     FSM ; (Q, Σ, δ, I, F), Q (set of states), Σ (input alphabet with λ the null symbol).
          δ (transition function), I (set of initial states) and F (set of final states).
          T = array_{Σ×Σ}: ℜ      (cost of insertions, substitutions and deletions)
          x = string to be parsed      (noisy string).
OUTPUT    final_score : ℜ
VAR       C: array_Q of ℜ;             (minimum cost path to each state in stage k)
          C´: array_Q of ℜ;            (minimum cost path to each state in stage k+1)
          a: symbol;                   (symbol belonging to the considered transition)
          a: ℜ;  α, PMIN : ℜ;          (BEAM SEARCH)

BEGIN EV2
 C:= [∞]; C´:= [∞];
 ∀ q ∈ I do C_q := 0; endV
 for each symbol x_i in the string x do
  PMIN := ∞;
  ∀ q ∈ Q do
   a := C_q + T_{λ,a};
   If (a < min(C´q , PMIN + α) then C´q := a; PMIN := min(a , PMIN); endif      (INSERTIONS)
   ∀ q´ ∈ δ(q) do            (SUBSTITUTIONS)
    a := C_q + W(q, q´) + T_{a,a}; If (a < min(C´q´ , PMIN + α) then C´q´ := a; PMIN := min(a , PMIN); endif
   endV
  endV
  ∀ q ∈ Q do
   q := Find_minimum(C´);        (Apply Dijkstra)
   ∀ q´ ∈ δ(q) do            (DELETIONS)
    a := C´q + W(q, q´) + T_{a,λ};
    If (a < min(C´q´ , PMIN + α) then C´q´ := a; PMIN := min(a , PMIN); endif
   endV
  endV
  C := C´; C´:= [∞];
 endfor
 final_score = min_{∀q ∈ I} (C_q);
END EV2
```

Fig. 2. The Algorithm EV2 developed from the recurrence relation in [4]

in the FSM, the cost of the minimum cost path that connects both of them. A solution for computing the function ζ consists in directly applying the Dijkstra algorithm [1]. Given that there is no deletion transition with a negative cost, we can discard all transitions from a state to itself. Similarly, we can discard transitions which reach an state that is actually part of the minimum cost path. This is due to the fact that the minimum cost path between two states cannot pass through another state twice. So, proceeding like Dijkstra we have to choose as the next state in the minimum cost path, a state having the minimum accumulated cost (*score*). Therefore, the states in the arrays (see Fig. 2) have to be ordered by this score.

Fig. 2 shows the algorithmic strategy *we developed* from the recurrence relation in [4]. The computational cost of *Find_minimum* is $|Q|$, in the worst case. But if the arrays are implemented as *priority queues* [1], then a $\log|Q|$ average cost can be expected. It is necessary to dynamically change the score (therefore the position) of the states in the *heap* to carry out this implementation. This is not really a problem if we take care of storing the *pointer* to each state in the heap, so we are able to *heapify* the arrays from the position of the state whose score has changed. So, the worst-case temporal complexity of this algorithm is $\bigcirc(|Q|^2.|x|)$ [3], expected $\bigcirc(|Q|.\max(\log|Q|, B).|x|)$ in the average case, if the

implementation of the arrays C and C' as priority queues is performed ($|Q|$ is the number of states in the FSM, B is the maximum branching factor and $|x|$ is the number of symbols in the input string). Since this algorithm is intended for classification purposes it is only needed the *final score* for the input string. As a consequence, the spatial complexity is $O(|Q|)$.

4 Solving the Problem by Depth-First Ordering

In this section, we propose an algorithm (EV1) based on a recurrence relation which is inspired in previous ideas of [15] and [13] for ECP with *acyclic* FSM's. This relation can be stated as follows:

$$C(q_{k+1}^j) = \min_{\substack{\forall i \in \delta'(q_l) \text{ in stage } k \\ \forall l \in \delta'_T(q_j) \text{ in stage } k+1}} \left\{ C(q_k^i) + W(q_k^i, q_{k+1}^l) + W_T(q_{k+1}^l, q_{k+1}^j) \right\} \quad (2)$$

where:

1. $C(q_k^i)$, $W(q_k^i, q_{k+1}^l)$ and δ' are as in (1).
2. δ'_T stands for a *generalization* of function δ' which returns, for a given state q, the whole set of states that are his *"topological predecessors"* in the FSM.
3. $W_T(q_{k+1}^l, q_{k+1}^j)$ stands for the minimum cost *path* that connects state q_l with state q_j, both in stage $k+1$, and *only* involves states that are topological predecessors of state q_j in the FSM.
   Observe that a "topological order" of the states in a FSM can *only* be properly defined for acyclic FSM's (as those considered in [15], [13], using ECGI grammars). Therefore, the concept of "topological sort" has to be somewhat *adapted* to cyclic FSM's as we subsequently see in this section.

It can be observed that the value for $W_T(q_{k+1}^l, q_{k+1}^j)$ will be 0, if q_l and q_j are the same and there is no *self-loop* (a transition from a state to itself) for state q_j. In this case, this value will be greater than 0 when considering *self-loops* but, then, the minimum cost for $C(q_{k+1}^j)$ will be given for the cost of $C(q_k^i) + W(q_k^i, q_{k+1}^l)$. Similarly, we can discard, as the minimum cost path, the set of all the *complete paths* (which begin in state q_i in stage k and end in state q_j in stage $k+1$) which pass through some state twice. Note that this relation is similar to the one proposed in [4]. Nevertheless, there are significant differences among them.

The relation (2) states that the minimum cost path to state j in stage $k+1$, $C(q_{k+1}^j)$, is the minimum cost of all the paths involving a state q_i in stage k, a *transition* from this state to some state q_l in stage $k+1$ (with q_l belonging to $\delta(q_k^i)$, so q_i is a *direct* predecessor of q_l) and one or more (consecutive) deletion transitions (between states in stage $k+1$) from each state q_l which is a *topological predecessor* of q_j to this state (q_j). Observe that, proceeding like this, one or more consecutive deletion errors can be parsed for every time interval or stage k.

```
INPUT     same as EV2 algorithm.
OUTPUT    same as EV2 algorithm.
VAR       TopologicalQ: list of state;          (list of states topologically sorted)
          C: arrayQ of ℜ; C': arrayQ of ℜ; a:symbol;   (same as EV2 algorithm)
          a: ℜ; back: boolean; bs: state;       (to parse cycles in the FSM)

BEGIN EV1
 TopologicalQ := Preprocess(Q); C:= [∞]; C':= [∞];
 ∀ q ∈ I do Cq := 0; endV
 for each symbol xi in the string x do
  ∀ q ∈ Q do
   a := Cq + Tλ,xi; if (a < C'q) then C'q := a endif          (INSERTIONS)
   ∀ q' ∈ δ(q) do          (SUBSTITUTIONS)
    a := Cq + W(q, q') + Tσ,xi; if (a < C'q') then C'q' := a; endif
   endV
  endV
  q := First(TopologicalQ);
  while q ≠ Last(TopologicalQ) do
   back := FALSE; bs := Last(TopologicalQ) ;
   ∀ q' ∈ δ(q) do          (DELETIONS)
    a := C'q + W(q, q') + Tσ,λ;
    if (a < C'q') then C'q' := a;
     if (q' < q) then       (a cycle has been found)
      back := TRUE;
      if (q' < bs) then bs:= q'; endif
     endif
    endif
   endV
   if (back) then q := bs else q := Next(TopologicalQ); endif
  endwhile
  C := C'; C':= [∞];
 endfor
 final_score = min_{∀q ∈ F} (Cq);
END EV1
```

Fig. 3. The Algorithm EV1 developed from the ideas of [15], [13]

Therefore, the same results achieved by proceeding like Dijkstra can be achieved by following a topological order of the states in the FSM to compute the values for W_T.

It should be argued that it is not possible to define any topological order for the states in a cyclic FSM. But, even for cyclic FSM's it is possible for us to define a kind of topological order which can serve to our purposes: the so-called *Depth-first topological sort* [1]. Following this ordering we are able to detect the so-called *back-arcs* [1]. These are the transitions which produce cycles in the FSM. This kind of topological order is *compatible* with the parsing of deletion errors such as it is established in (2). Once the states of the FSM have been sorted, the only thing to worry about is to determine when a back-arc is parsed. But, with a proper management of the list of states topologically sorted (implemented as a hash table, for instance) the backtracks produced by these specific transitions can be efficiently and adequately parsed, leading to an *expected average case* temporal complexity $\bigcirc(|Q|.B.|x|)$ [3] (see Fig. 3).

Assuming that the states in the FSM have been (depth-first) topologically sorted in a preprocessing stage, full ECP can thus be implemented as a direct generalization of the original Viterbi algorithm, as detailed in Fig. 3. The temporal complexity of such a preprocessing stage is $\bigcirc(|Q|.B)$ [1] [2], which could

be clearly negligible in many cases. Note that the worst-case time complexity is still $\bigcirc(|Q|^2.|x|)$ and the spatial complexity is $\bigcirc(|Q|)$, as in EV2.

5 A Suboptimal Search Strategy

Despite the adequate temporal and spatial complexity of the original Viterbi algorithm (both linear with $|Q|$), the size of the FSM models could be an important bottleneck for some real-time Pattern Recognition tasks. To this end, a search technique – Beam Search [11] – could be easily integrated with the parsing strategy of the Viterbi algorithm at the risk of obtaining suboptimal solutions. Nevertheless some empirical results has shown the goodness of such an approach [15] [16], in the sense of obtaining approximately optimal or the optimal solutions with a drastical decrease in the temporal complexity of the original algorithm. The temporal complexity becomes *sublinear* with $|Q|$ depending upon a given parameter which measures the tradeoff between the efficiency and the accuracy of the final maximum likelihood classification result: the *beam width* (this parameter is named α in Fig. 2). The more this parameter decreases, the less the accuracy of the search and the more the temporal complexity decreases and vice versa.

Although the extension of the algorithm developed in Sect. 3 to perform Beam Search is straightforward (see bold font in Fig. 2 – observe that, for the sake of conciseness, the management of the set of states Q as a *list of visited states* is omitted in this figure –), this is not the case for the algorithm detailed in Sect. 4. This is due to the fact that, using Beam Search, the list of states topologically sorted is not *fixed* beforehand (it could change for each stage of the parsing process). Therefore, we would have to dynamically (topologically) sort the *visited* states before each deletion transitions parsing stage, thus leading to a temporal complexity similar to that of the algorithm EV2 in the worst case. It is worth exploring possible solutions for this problem, given the good performance achieved by the algorithm EV1 (see next section for result details).

Although neither mentioned in [4] nor in Sect. 3, a preprocessing stage (using the original Dijkstra algorithm) can be used in EV2 to compute and store the values of ζ, for all pair of states in the FSM. The temporal and spatial complexity of such a stage is $\bigcirc(|Q|^2)$. Although this spatial complexity may be *prohibitive in many applications*, this preprocessing stage would lead to a time-efficient algorithm for ECP, with a worst-time complexity linear with $|Q|$, which might be very useful in some cases. There are basically two reasons for not implementing this preprocessing in EV2. The first one is related with the fact that a quadratic spatial complexity could be very inadequate for *real-time* tasks. The second one is that this preprocessing is clearly incompatible with the Beam Search strategy. Think about the fact that the minimum cost path which connects state q_l with state q_j, both in stage $k+1$, could pass through another state, namely q_r, which *could have not been visited* in this stage, so forcing to compute this preprocessing before each deletion transitions parsing stage. Obviously this method is clearly unapproachable for most of the tasks.

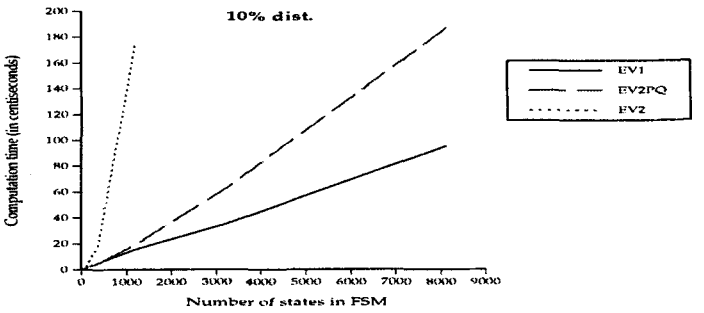

Fig. 4. *Average* computing times (in centiseconds) measured without Beam Search (EV2PQ is the implementation of EV2 with priority queues)

6 Experiments and Results

Some experiments have been carried out in order to test the performance of the previously presented algorithms. A set of *six* stochastic FSM's were used in these experiments. These FSM's were automatically learned from 50,000 sentences of a Language Learning and Understanding task recently proposed by Feldman et al [5] called "Miniature Language Acquisition" (MLA). The learning was performed by the k-TSI Grammatical Inference algorithm proposed in [8], for increasing values of k from 2 to 7. This algorithm infers a (stochastic) FSM that accepts the smallest k-Testable Language in the strict sense (k-TS language) that contains the training set of sentences. Stochastic k-TS languages are equivalent to the languages modeled by the well known N-GRAMS, with $N = k$.

The size of the inferred FSM's ranges from 26 to 8,133 states. The test set consists of 1,000 sentences not previously used in the training phase. All of these sentences do belong to each language respectively generated by each FSM. However the test set was *distorted* using a distortion model proposed in [10] in order to simulate the noise produced in the acquisition and/or feature extraction phases, generally resulting in sentences no longer belonging to the different FSM languages. Two different percentages of *global distortion* – evenly distributed between each parameter of the distortion model – were used: 5% and 10%. The error correction model was specified according to the distortion parameters previously used.

Results, in terms of the observed average parsing time for each input string, are shown in Figs. 4 and 5. Fig. 4 shows performance results *without* Beam Search for EV1, EV2 and the implementation of EV2 with priority queues. The results are shown only for a 10% of distortion; the results were *exactly the same* for a 5% of distortion. Fig. 5 shows the observed results for the experiments performed with Beam Search (for beam widths of 10, 20 and 40). It should be reminded that *only* the two different implementations of EV2 algorithm incorporate the Beam Search strategy (see Sect. 5). Fig. 6 shows the percentage of states visited with this suboptimal search strategy (the percentage achieved for

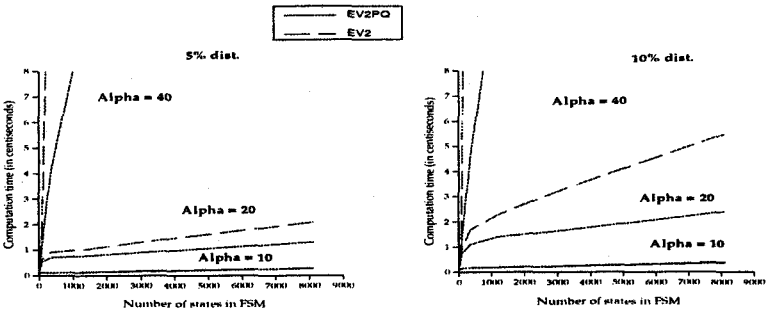

Fig. 5. *Average* computing times (in centiseconds) measured for Beam widths of 10, 20 and 40 (for the two different implementations of EV2 *only*)

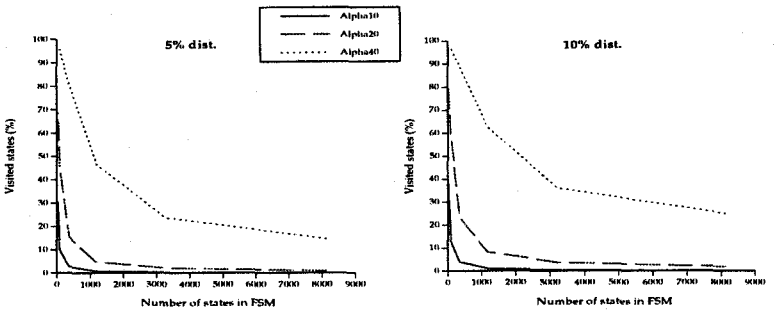

Fig. 6. Rate of visited states for Beam widths of 10, 20 and 40

the two different implementations of EV2 was exactly the same, as it could be expected).

The final results of *maximum likelihood scoring* for each sentence and for each experiment were exactly the same for each algorithm. The 100% of the distorted test sentences were recognized as *belonging to the language* generated by each FSM (even using the lowest Beam width). The observed computing times for the preprocessing stage in the algorithm EV1 were negligible with regard to the parsing process (they ranged from less than a centisecond to 3 centiseconds). All the experiments were carried out in an HP9000 Unix Workstation (Model 735) performing 121 MIPS.

7 Discussion and Conclusions

Several techniques have been proposed to accelerate the process of Finite-State Error-Correcting Parsing. This constitutes a core process in many applications using Syntactic Pattern Recognition techniques. A significant improvement in parsing speed is achieved by the proposed EV1 algorithm with regard to previously proposed techniques (see Fig. 4). Furthermore, a dramatic acceleration is achieved by applying suboptimal techniques based on Beam-Search strategies

to the proposed algorithms (see Figs. 5 and 6). Next step is to study adequate ways to apply Beam-Search to the new algorithm EV1.

Acknowledgement

The authors wish to thank to the anonymous referees their careful reading and valuable comments.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley Publishing Company (1983)
2. Amengual, J.C., Vidal, E.: Una extensión del Algoritmo de Viterbi para el Análisis Sintáctico Corrector de Errores (ASCE) sobre Gramáticas ECGI mediante Búsqueda en Haz. Technical Report, DSIC-II/3/94. Depto. de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia Spain (1994)
3. Amengual, J.C., Vidal, E.: Fast Viterbi Decoding with Error Correction. Preprints of the VI Spanish Symposium on Pattern Recognition and Image Analysis. Edited by A. Calvo and R. Medina. Cordoba, Spain (3–7 April 1995) 218–226
4. Bouloutas, A., Hart, G.W., Schwartz, M.: Two Extensions of the Viterbi Algorithm. IEEE Trans. on Information Theory, Vol. **37** no. **2** (March 1991) 430–436
5. Feldman, J.A., Lakoff, G., Stolcke, A., Weber, S.H.: Miniature Language Acquisition: A touchstone for cognitive science. Technical Report, TR-90-009. International Computer Science Institute. Berkeley California (April 1990)
6. Forney, G.D.: The Viterbi algorithm. Proc. IEEE, vol. **61** (1973) 268–278
7. Fu, K.S.: Syntactic Pattern Recognition and Applications. Prentice Hall (1982)
8. García ,P., Vidal, E.: Inference of k-testable languages in the strict sense and application to Syntactic Pattern Recognition. IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-12 no. **9** (September 1990) 920–925
9. Gonzalez, R.C., Thomason, M.G.: Syntactic Pattern Recognition. An Introduction. Addison-Wesley Pub. Co., Advanced Book Program Reading Massachusetts (1978)
10. Hunt, M.J.: Evaluating the performance of connected-word speech recognition systems. Proceedings of the ICASSP (1988) 457–460
11. Lowerre, B.T.: The Harpy Speech Recognition System. Internal Report. Carnegie-Mellon University (1976)
12. Lucas, S., Vidal, E., Amiri, A., Hanlon, S., Amengual, J.C.: A Comparison of Syntactic and Statistical Techniques for Off-Line OCR. In Grammatical Inference and Applications. R.C. Carrasco and J. Oncina (eds.). LNCS **862**. Springer-Verlag (1994) 168–179
13. H. Rulot, H.: ECGI. Un algoritmo de Inferencia Gramatical mediante Corrección de Errores. Phd Dissertation. Universidad de Valencia (1992)
14. Thomason, M.G.: Errors in regular languages. IEEE Trans. Comput., vol. **C-23** no. **6** (June 1974) 597–602
15. Torró, F.: Estudio de alternativas en la reducción de la complejidad del Algoritmo de Reconocimiento basado en el método ECGI. Proyecto Fin de Carrera. Facultad de Informática UPV. Valencia (1989)
16. Torró, F., Vidal, E., Rulot, H.: Fast and Accurate Speaker Independent Speech Recognition using structural models learnt by the ECGI Algorithm. Signal Processing V Theories and Applications. Elsevier Science Publishers (1990)