A Common Solution for Tokenization and Part-of-Speech Tagging: One-Pass Viterbi Algorithm vs. Iterative Approaches*

Jorge Graña, Miguel A. Alonso, and Manuel Vilares

Departamento de Computación
Universidad de La Coruña
Campus de Elviña s/n
15071 - La Coruña, Spain
{grana, alonso, vilares}@dc.fi.udc.es

Abstract. Current taggers assume that input texts are already tokenized, i.e. correctly segmented in *tokens* or high level information units that identify each individual component of the texts. This working hypothesis is unrealistic, due to the heterogeneous nature of the application texts and their sources. The greatest troubles arise when this segmentation is ambiguous. The choice of the correct segmentation alternative depends on the context, which is precisely what taggers study.
In this work, we develop a tagger able not only to decide the tag to be assigned to every token, but also to decide whether some of them form or not the same term, according to different segmentation alternatives. For this task, we design an extension of the Viterbi algorithm able to evaluate streams of tokens of different lengths over the same structure. We also compare its time and space complexities with those of the classic and iterative versions of the algorithm.

1 Introduction

Some languages, like Galician or Spanish, show complex phenomena that we have to handle before tagging. Among other tasks, the segmentation process is responsible for identifying information units such as sentences or words. In the case of words, for instance, the problem is that the spelling of a word does not always coincide with the linguistic concept. Therefore, we have two options:

1. The simpler approaches just consider "spelled words" and extend the tags in order to represent relevant phenomena. For instance, the Spanish word `reconocerse` (*to recognize oneself*) could be tagged as `V+Pro` even when it is formed by a verb and an enclitic pronoun, and the words of the Spanish expression `a pesar de` (*in spite of*) would be respectively tagged as `P13`,

* This work has been partially supported by the Spanish Government (under projects TIC2000-0370-C02-01 and HP2001-0044), and by the Galician Government (under project PGIDT01PXI10506PN).

P23 and P33 even when they constitute only one term[1]. However, this approach is not valid for Galician because its great morphological complexity would produce an excessive growth of the tag set, and the process of building reference texts containing all those tags in a significantly representative number of situations would be an extremely hard task.

2. Another solution is not to extend the basic tag set. As an advantage, the complexity of the tagging process is not affected by a high number of tags. As a drawback, this approach makes the tasks of the tokenizer more complex. Now, it not only has to identify "spelled words", but often also has either to split one word into several words, or join several words in only one.

In our work, we have chosen the second option. The greatest troubles arise when this segmentation is ambiguous. For instance, the words in the Spanish expression `sin embargo` will normally be tagged together as a conjunction (*however*), but in some context they could be a sequence of a preposition and a noun (*without seizure*). In the same way, the Galician word `polo` can be a noun (*chicken*), or the contraction of the preposition `por` and the article `o` (*by the*), or even the verbal form `pos` with the enclitic pronoun `o` (*put it*).

In this way, the preprocessor should only perform the detection and pretagging of alternatives. The choice of the correct one depends on the context, which is precisely what is studied by the tagger. In consequence, the aim of the present work is to develop a tagger able not only to decide the tag to be assigned to every token, but also to decide whether some of them form or not the same term, and assign the appropriate number of tags on the basis of the alternatives provided by the preprocessor. For this task, we extend the Viterbi algorithm in order to evaluate streams of tokens of different lengths over the same structure. Finally, we will perform an intuitive study of performances by comparing time and space complexities of the new algorithm with those of the classic and iterative versions.

2 Viterbi-L: the Viterbi Algorithm on Lattices

2.1 Why Lattices?

In the context of part-of-speech tagging with Hidden Markov Models (HMMs), the classic version of the Viterbi algorithm [4] is applied on trellises, where the first row contains the words of the sentence to be tagged, and the possible tags appear in columns below the words. However, let us consider, for instance, a sentence in which the expression `sin embargo` appears. As we can see in Fig. 1, problems arise when we try to situate the tag `C` because, in both options, the tag should apply to the whole expression, and hence the paths marked with dashed lines should not be allowed, and the ones marked with thick lines should be allowed.

[1] To simplify, in this work, we use `Adj` for adjective, `Adv` for adverb, `C` for conjunction, `Det` for determiner, `P` for preposition, `Pro` for pronoun, `S` for substantive, and `V` for verb. The tags that appear in real life come from projects GALENA (*Generation of Natural Language Analyzers*) and CORGA (*Reference Corpus of Current Galician*). See `http://coleweb.dc.fi.udc.es` for more information on both projects.

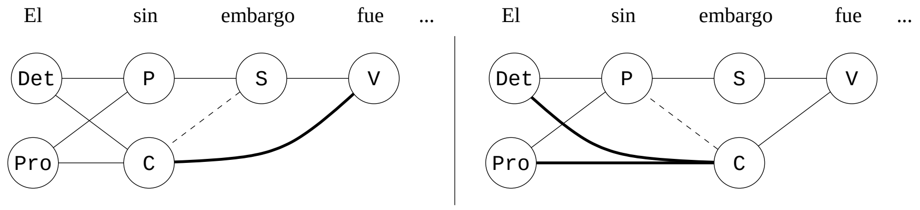

Fig. 1. Trellises cannot represent ambiguous segmentations

We can represent the kind of ambiguous segmentations described above more comfortably by using lattices. In these structures, the arcs that conform the paths have their origin and target points in the gaps between words. The labels of these arcs contain the tags, as is shown in Fig. 2, where we can see that it is even possible to represent overlapped dependencies.

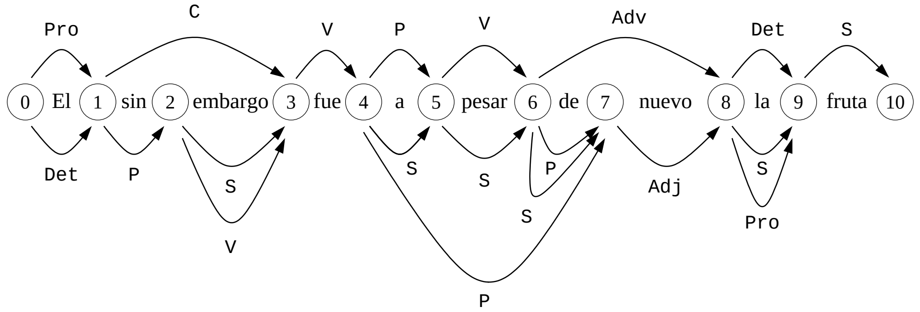

Fig. 2. Ambiguous segmentations represented on a lattice

For the purposes of a later discussion, we indicate that this lattice contains 20 arcs, with which we can build 234 possible paths. The lengths of these paths are 7, 8, 9 or 10 tokens. The correct tagging is the one formed by the 8 arcs drawn in the upper part of the lattice, and corresponds to the following sense: *He however went to weigh again the fruit* (literal translation).

2.2 The Viterbi-L Equations

The equations of the Viterbi algorithm can be adapted to process a language model operating on a lattice as follows [1]. Instead of the words, the gaps between the words are enumerated (see Fig. 2), and an arc between two gaps can span one or more words, such that an arc is represented by a triple (t, t', q), starting at time t, ending at time t' and representing state q. We introduce accumulators $\Delta_{t,t'}(q)$ that collect the maximum probability of state q covering words from position t to t'. We use $\delta_{i,j}(q)$ to denote the probability of the derivation emitted by state q having a terminal yield that spans positions i to j.

- Initialization: $\Delta_{0,t}(q) = P(q|q_s)\,\delta_{0,t}(q)$
- Recursion:

$$\Delta_{t,t'}(q) = \max_{(t'',t,q')\in\text{Lattice}} \Delta_{t'',t}(q')\,P(q|q')\,\delta_{t,t'}(q) \quad \text{for} \quad 1 \leq t < T \qquad (1)$$

- Termination: $\max_{Q\in\mathcal{Q}^*} P(Q,\text{Lattice}) = \max_{(t,T,q)\in\text{Lattice}} \Delta_{t,T}(q)\,P(q_e|q)$

where q_s and q_e are the initial and ending states, respectively. Additionally, it is necessary to keep track of the elements in the lattice that maximized each $\Delta_{t,t'}(q)$. When reaching time T, we get the best last element in the lattice

$$(t_1^m, T, q_1^m) = \arg\max_{(t,T,q)\in\text{Lattice}} \Delta_{t,T}(q)\,P(q_e|q)$$

Setting $t_0^m = T$, we collect the arguments $(t'', t, q') \in$ Lattice that maximized equation (1) by going backwards in time:

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \arg\max_{(t'',t_i^m,q')\in\text{Lattice}} \Delta_{t'',t_i^m}(q')\,P(q_i^m|q')\,\delta_{t_i^m,t_{i-1}^m}(q_i^m)$$

for $i \geq 1$, until we reach $t_k^m = 0$. Now, $q_1^m \ldots q_k^m$ is the best sequence of phrase hypothesis (read backwards). We will call this process the Viterbi-L algorithm.

In practice, our intention is to estimate the parameters of our HMM from tagged texts, and use linear interpolation of uni-, bi-, and trigrams as our smoothing technique [2], i.e. our operating model will be a second order HMM. This is not a problem because the Viterbi-L algorithm described above can work with the second order hypothesis simply by considering pairs of tags (or states) as labels of the arcs, instead of only one tag (or state).

2.3 Complexity of the Viterbi-L Algorithm

Intuitively, we can consider the space complexity as the number of probability accumulators that we have to store during execution. In this version, we have one accumulator per arc. For time complexity, we consider the number of operations that we have to perform. This is, for a given arc, the number of arcs reaching the origin point of the arc under consideration. For instance, in order to pass Viterbi-L on the lattice of Fig. 2, we need 20 accumulators and 36 operations.

However, we have to make the following reflection. With this simple version of the algorithm, the shortest paths have priority because they involve a smaller number of multiplications and hence they obtain a better cumulative probability. This is a problem, since the shortest paths do not always correspond to correct interpretations.

To avoid this problem, we could consider the individual evaluation of lattices with paths of the same length, and their subsequent comparison. It would therefore also be necessary to define an objective criterion for that comparison. If the tagging paradigm used is the framework of the HMMs, as is our case, a consistent criterion is the comparison of the normalization of the cumulative probabilities.

Let us call p_i the cumulative probability of the best path in a lattice with paths of length i tokens. In the case of Fig. 2, we would have p_7, p_8, p_9 and p_{10}. These values are not directly comparable, but if we use logarithmic probabilities, we can obtain normalized values by dividing them by the number of tokens. In this case, $p_7/7$, $p_8/8$, $p_9/9$ and $p_{10}/10$ are now comparable, and we can select the best path from the best lattice as the most probable interpretation. One reason to support the use of HMMs is that in other tagging paradigms the criteria for comparison may not be so easy to identify.

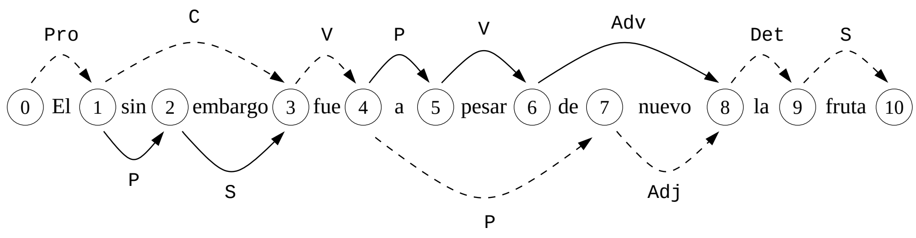

Fig. 3. A lattice with conflictive paths

However, the number of different lattices to evaluate is not always the number of different lengths for the paths. For instance, we can see in Fig. 3 how two paths of the same length (the one that only uses the upper arcs of the lattice, and the one that uses lower arcs when available, both of 8 tokens) can produce another path with different length (the one of 7 tokens marked with dashed arcs). Therefore, we could need more than one lattice to represent the paths of the same length without conflicts. In the case of Fig. 2, although we have 4 possible lengths for the paths (7, 8, 9 or 10 tokens), we need a total of 6 lattices, which come from the mutual exclusion of the different alternatives of each ambiguous segmentation. This analysis is shown in the decision tree of Fig. 4. To save space, we have not drawn the corresponding lattices, but for each of them we indicate: the number of possible paths, the length of those paths in tokens, the number of arcs in the lattice, and the number of operations needed to pass Viterbi-L on it. Therefore, by looking at the total numbers involved in this set of lattices, we can deduce that the real space and time complexities of the algorithm are 82 accumulators and 119 operations.

3 Viterbi-N: the Viterbi Algorithm with Normalization

Our proposal is to use only one lattice, and perform only one pass of the Viterbi algorithm. In order to do so, it is necessary to store more than one accumulator per arc. More exactly, we keep as many accumulators as there are different lengths of all the paths reaching the point of origin of the arc. We incorporate this information about lengths in a third component of the index of each accumulator: $\Delta_{t,t',l}(q)$. In this way, only accumulators with the same length l are compared

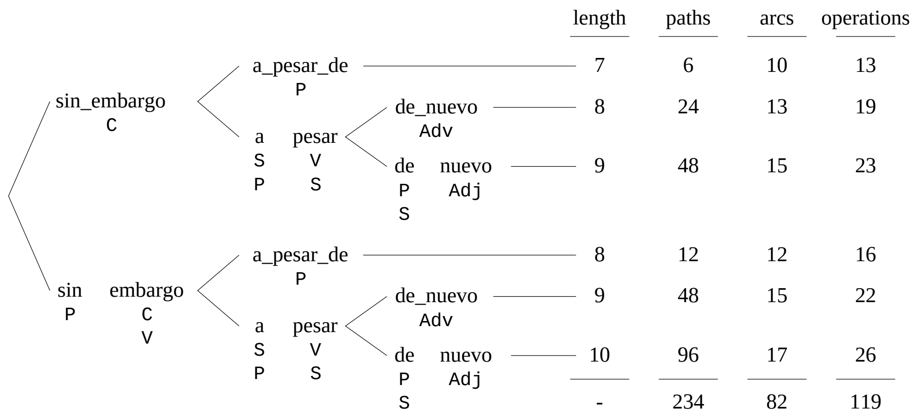

Fig. 4. Decision tree to avoid conflictive paths

in the maximization operations to obtain the corresponding $\Delta_{t',t'',l+1}(q')$. When reaching the final instant, we will have obtained as many accumulators as there are different lengths, allowing us to normalize their corresponding best paths according to those lengths before selecting the most probable interpretation. We will call this process the Viterbi-N algorithm.

3.1 The Viterbi-N Equations

Assuming the use of logarithmic probabilities to speed up the calculations and avoid problems of precision that arise in products with factors less than 1, we replace those products by sums and adapt the equations as follows:

- Initialization: $\Delta_{0,t,1}(q) = P(q|q_s) + \delta_{0,t}(q)$
- Recursion:

$$\Delta_{t,t',l}(q) = \max_{(t'',t,q')\in\text{Lattice}} \Delta_{t'',t,l-1}(q') + P(q|q') + \delta_{t,t'}(q) \quad \text{for} \quad 1 \leq t < T \tag{2}$$

- Termination: $\max_{Q\in\mathcal{Q}^*} P(Q, \text{Lattice}) = \max_{l} \dfrac{\max_{(t,T,q)\in\text{Lattice}} \Delta_{t,T,l}(q) + P(q_e|q)}{l}$

Additionally, it is also necessary to keep track of the elements in the lattice that maximized each $\Delta_{t,t',l}(q)$. When reaching time T, we get the length of the best path in the lattice

$$L = \arg\max_{l} \frac{\max_{(t,T,q)\in\text{Lattice}} \Delta_{t,T,l}(q) + P(q_e|q)}{l}$$

Next, we get the best last element of all paths of length L in the lattice

$$(t_1^m, T, q_1^m) = \arg\max_{(t,T,q)\in\text{Lattice}} \Delta_{t,T,L}(q) + P(q_e|q)$$

Setting $t_0^m = T$, we collect the arguments $(t'', t, q') \in$ Lattice that maximized equation (2) by going backwards in time:

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \arg\max_{(t'', t_i^m, q') \in \text{Lattice}} \Delta_{t'', t_i^m, L-i}(q') + P(q_i^m | q') + \delta_{t_i^m, t_{i-1}^m}(q_i^m)$$

for $i \geq 1$, until we reach $t_k^m = 0$. Now, $q_1^m \ldots q_k^m$ is the best sequence of phrase hypothesis (read backwards).

3.2 Viterbi-N vs. Viterbi-L

By using intuition again, we can also consider the space complexity of the Viterbi-N algorithm as the number of accumulators, and the time complexity as the number of operations to perform. In this case, for space complexity, we can have more than one accumulator per arc, as has been explained above. And for time complexity, we calculate, for each arc, the sum of the number of accumulators of the arcs reaching its point of origin. In order to pass Viterbi-N on the lattice of Fig. 2, we need only 44 accumulators (instead of the 82 needed by Viterbi-L) and 73 operations (instead of 119). Furthermore, we also avoid the analysis of conflictive paths and their distribution in several lattices.

4 Viterbi-I: the Iterative Viterbi Algorithm

To find the best normalized path in a lattice is a problem that has already been studied [3]. In that work, the authors prove the correctness of the following iterative algorithm.

4.1 The Viterbi-I Equations

We have the following available elements: L, a set of possible sentences in a lattice, where any sentence S has $|S|$ tokens; a cost function C that associates a real number $C(S)$ to any sentence S; and an algorithm $\mathcal{A}(L, C, \beta)$ that permits us to find $\arg\max_{S \in L} C(S) - \beta|S|$ for any real number β.

The algorithm $\mathcal{I}(\mathcal{A}, L, C)$ extracts the solution of $\arg\max_{S \in L} \bar{C}(S)$, where $\bar{C}(S) = C(S)/|S|$:

- Initialization: $\bar{C}(S_{-1}) = 0$.
- Recursion: We calculate $S_i = \arg\max_{S \in L} C(S) - \bar{C}(S_{i-1})|S|$ by using the algorithm $\mathcal{A}(L, C, \bar{C}(S_{i-1}))$.
- Termination: We stop iterations when $|S_i| = |S_{i-1}|$. Now, S_i is the solution.

4.2 Viterbi-I vs. Viterbi-N

The cost function C involves the usual transitions and emission probabilities, and can be applied in the frame of algorithm $\mathcal{A}$ by a technique similar to the Viterbi process. Hence the name Viterbi-I algorithm. This means that to every iteration of the algorithm we can attach the smallest space and time complexities that we found for lattices, i.e. 20 accumulators and 36 operations in the case of Fig. 2. However, at least two iterations are needed to reach a solution (which implies multiplying the cited time complexity at least by 2), and it is not possible to guarantee that two iterations will be enough in all cases.

The average number of iterations in the context of part-of-speech tagging should be investigated further. But even though this number is close to 2, the Viterbi-N algorithm will always provide a solution with only one pass over the corresponding lattice, with a space complexity approximately the double of the number of arcs, which is not critical in practice (44 vs. 20 accumulators, in the case of Fig. 2), and with approximately the same time complexity as the minimum needed by Viterbi-I (73 vs. $36 \times 2 = 72$ operations).

5 Conclusion and Future Work

We have presented an extension of the Viterbi algorithm that allows us to analyze ambiguous segmentations in only one pass, and a discussion about intuitive complexities. It would of course be necessary to perform a formal study of theoretical complexities. This implies generalizing the number of nodes and arcs in the lattice. By doing this, we could see that the space complexity of our one-pass algorithm presents a cubic growth, while in the iterative case we would only appreciate an increment in the number of iterations. However, this is not a practical problem, since lattices with millions of interpretations do not correspond to real sentences. In fact, the guide example used in this work is somewhat artificial. It is possible to assume that real sentences will not present more than one or two ambiguous segmentations, the one-pass approach therefore being more suitable for part-of-speech tagging. Be that as it may, the most important future task is to apply these algorithms on large real data. We expect this general approach to work well, especially for Galician, where it is more frequent to find ambiguous segmentations.

References

1. Brants, T. (1999). Cascaded Markov models. In *Proc. of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL-99)*, Bergen, Norway.
2. Graña, J.; Chappelier, J.-C.; Vilares, M. (2001). Integrating external dictionaries into part-of-speech taggers. In *Proc. of the Euroconference on Recent Advances in Natural Language Processing (RANLP-2001)*, Tzigov Chark, Bulgaria.
3. Rozenknop, A.; Silaghi, M. (2001). Algorithme de décodage de treillis selon le critère du coût moyen pour la reconnaissance de la parole. In *Actes de la 8ème conférence sur le Traitement Automatique des Langues Naturelles (TALN-2001)*, Tours, France.
4. Viterbi, A.J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory*, vol. IT-13 (April).