UNIVERSIDADE DE VIGO
Departamento de Informática

TESIS DOCTORAL

ANÁLISIS LÉXICO ROBUSTO

Autor: JUAN OTERO POMBO
Directores: MANUEL VILARES FERRO
JORGE GRAÑA GIL

Ourense, Junio 2009

Esta tesis opta a la mención de "Doctor Europeus"

Depósito Legal: OU80-2009

A María José e Iria

Deseo mostrar mi más sincero agradecimiento a los miembros del tribunal que han aceptado gustosos dedicar una parte de su valioso tiempo a la revisión y evaluación de este trabajo: Guillermo Rojo Sánchez (Universidade de Santiago de Compostela), José Gabriel Pereira Lopes (Universidade Nova de Lisboa, Portugal), Jean-Eric Pin (Centre National de la Recherche Scientifique (CNRS), Francia), Leo Wanner(Institució Catalana de Recerca i Estudis Avançats) y Víctor Manuel Darriba Bilbao (Universidade de Vigo). Es para mi un gran honor que mi trabajo sea juzgado por un tribunal formado por investigadores cuyo prestigio en el ámbito en que se enmarca esta tesis contribuirá sin duda a incrementar el impacto de la misma.

Del mismo modo, quiero agradecer también a Benoît Sagot (Institut National de Recherche en Informatique et Automatique, (INRIA), Francia) y a Jan Daciuk (Universidad Tecnológica de Gdańsk, Polonia) el que hayan aceptado actuar como revisores expertos europeos a los efectos de la obtención de la mención de *Doctor Europeus*.

Agradecementos

Se atopamos a alguén que nos debe agradecemento, deseguida o lembramos. ¡Cántas veces nos atopamos con alguén ao que lle debemos agradecemento e non pensamos nelo!

Goethe

Permítaseme a licenza de escribir estes agradecementos na miña lingua nai, xa que é nela na que flúen da miña mente os sentimentos máis sinceros, coa que me identifico e me identifican as persoas ás que adico estas liñas.

Tratarei de expresar aquí o meu agradecemento a todas as persoas que, dun xeito ou outro, fostes partícipes da traxectoria que me trouxo ata aquí. Todos os que sintades ledicia por min podedes considerarvos destinatarios desta mensaxe. Non busquedes o voso nome nos parágrafos seguintes, simplemente convertide o voso sentimento de ledicia no meu de agradecemento.

Papá e mamá, sei que nunca poderei pagar a débeda que teño con vós, aínda que sei tamén que endexamais ma reclamaredes. Débovos a vida, pero de nada me serviría sen todo o que me destes despois. Sodes vós os que me inculcastes os principios que fan de min o que son. Apostastes firme non só pola miña formación e educación, senon tamén pola de Isabel, María, Pili e Nieves. Tanto sacrificio merece alomenos a recompensa do recoñecemento.

Maria José e Iria, vós sodes as que agora me dades a seguridade e confianza que necesito. A tí María José, teño que agradecerche a túa paciencia, xenerosidade e comprensión. Desde que comezamos este noso proxecto todos os meus logros son tamén teus e éste é só un máis. Iria, a tí agradézoche cada sorriso e cada xesto, gustaríame poder expresar o que me fas sentir. Es a raíña da casa e só espero poder darche o mellor.

Manuel, tí ofrecéchesme a oportunidade, os recursos, os consellos, o apoio, e todo o necesario para alcanzar esta meta. Foches a miña mellor garantía de éxito. Sei que estaría condenado ao fracaso se non tivese contado co mellor director. Espero ter respondido ás expectativas que puxeches en min.

Jorge, os teus consellos e a túa capacidade de síntese para facer sinxelo o que para min era tan complexo servíronme para poder avanzar nos momentos de dúbida.

Víctor, Fran, Mario, Fernando, Moli, Mila, Sara, Erica, Nieves, Adrián e Vanesa, vós fostes os amigos e compañeiros do día a día, eses aos que lles vas roubando cada día uns minutos de atención e que a base de pequenas, desinteresadas e espontáneas aportacións

fostes condimentando este traballo. Suso, Miguel e Carlos, fostes o complemento perfecto, en todo momento sentín o voso apoio.

Gabriel e Guillermo, foi un auténtico luxo poder realizar as estancias das que desfrutei baixo a vosa tutela na Universidade Nova de Lisboa e na Universidade de Santiago.

Isabel, María, Pili, Nieves, Jose, Moncho, José Manuel, Fran, Laura, Juanjo, Dani e María Isabel, ademáis da miña familia sodes o meu contorno fora da Universidade. Con vós desfrutei de moitas das miñas horas de lecer.

Todos os amigos aos que non podo nomear aquí porque a lista sería moi longa e seguro que esquecería a algún.

A todos vós graciñas.

Resumen

El presente trabajo se encuadra en el marco del *Procesamiento del Lenguaje Natural*, área de la ciencia y la tecnología que se encarga del tratamiento automático del lenguaje natural o humano. En particular, incluye aquellas tareas relativas al análisis léxico, la corrección ortográfica, la etiquetación morfosintáctica y la aplicación de éstas a la *Recuperación de Información*.

El objetivo principal de esta tesis es el desarrollo de tecnología de base en estos aspectos concretos.

En este contexto, hemos centrado nuestros mayores esfuerzos en el desarrollo de un nuevo método regional de corrección ortográfica sobre *Autómatas Finitos* y su integración en una herramienta de etiquetación morfosintáctica [58], con el fin de sacar provecho de la información contextual embebida en un Modelo Oculto de Markov [98] subyacente. De este modo, hemos desarrollado una herramienta de análisis léxico robusto capaz de manejar los tres tipos de ambigüedades que pueden surgir en esta fase: La ambigüedad morfosintáctica, que surge cuando a una unidad léxica le pueden ser asignadas diferentes etiquetas morfosintácticas; la ambigüedad segmental, que aparece cuando es posible dividir el texto en unidades léxicas de más de un modo; y la ambigüedad léxica, que es la que introducen los métodos de corrección ortográfica cuando ofrecen varias alternativas de corrección.

Para estimar la viabilidad del método desarrollado se han realizado diversos experimentos. Inicialmente hemos contrastado los valores de precisión, cobertura y rendimiento obtenidos por nuestro método regional con aquellos facilitados por la propuesta de Savary [140], que consideramos una de las más eficientes, aún a nivel global. Estas primeras pruebas se realizaron sobre palabras aisladas, es decir, sin tener en cuenta el contexto en el que éstas aparecían. Los resultados obtenidos en cuanto a rendimiento fueron realmente satisfactorios ya que el método regional superaba con claridad al global. En lo que respecta a la cobertura, el método regional ofrece en término medio un menor número de alternativas, lo que provoca un ligero descenso en la precisión.

Nuestro siguiente paso, consistió en comprobar si la pérdida de precisión del método regional podía ser compensada en un entorno de corrección contextual, ya que el hecho de que éste devolviese un menor número de alternativas podría repercutir de forma positiva en la precisión del sistema global [161]. Nuestros experimentos no han corroborado esta hipótesis, pero han servido para evidenciar que el incremento del rendimiento del método regional en términos de espacio y tiempo respecto al global era aún mayor cuando

aplicábamos estas técnicas en un entorno de corrección contextual. Esto era debido a que, además de resultar más eficiente desde el punto de vista computacional, el algoritmo regional ofrece un menor número de alternativas de corrección. Esto nos anima a continuar en la búsqueda de técnicas y heurísticas que nos permitan determinar cuando es posible optar por una corrección regional.

Finalmente, hemos realizado pruebas con el fin de verificar la utilidad práctica de nuestra propuesta en un entorno de *Recuperación de Información* en el que las consultas presentan errores ortográficos. Para ello, hemos comparado tres métodos [118]. El primero, consiste en expandir las consultas con todas las alternativas de corrección. El segundo, aplica nuestro corrector contextual para determinar cuál de las alternativas obtenidas encaja mejor en el contexto de la palabra errónea. El tercero, evita la aplicación de métodos de corrección ortográfica al utilizar n-gramas tanto para la indexación como para la recuperación. El resultado de estas pruebas confirma que la aplicación de técnicas de corrección ortográfica mejora sustancialmente los resultados en presencia de consultas corruptas. Por otra parte, la utilización de n-gramas resulta ser una técnica muy robusta y que presenta la ventaja de que no requiere ningún recurso lingüístico extra.

En resumen, nuestro trabajo es una propuesta que abarca el diseño, la implementación y la evaluación en un entorno práctico de una técnica original de corrección automática de errores léxicos para lenguajes naturales. Los resultados son, a día de hoy, prometedores y marcan nuestra línea de trabajo futuro en este campo concreto.

Abstract

This work fits into the scope of *Natural Language Processing* (NLP), the field of science and technology that manages the automatic processing of natural or human language. In particular, it includes those tasks related to lexical analysis, spell checking, part-of-speech (POS)tagging and the application thereof to *Information Retrieval* (IR).

The main objective of this PhD thesis is the development of basic technology in these concrete points.

In this context, we have focused our best efforts on developing a new regional spelling correction method over *Finite Automata* (FA) and its integration into a part-of-speech tagging tool [58], in order to take advantage of contextual information embedded in the underlying Hidden Markov Model (HMM) [98]. Thus, we have developed a robust lexical analysis tool capable of handling all three types of ambiguities that can arise at this stage: the morphosyntactic ambiguity that arises when a lexical unit can be assigned different part-of-speech tags; segmental ambiguity, which appears when there is more than one way to split the text into lexical units, and lexical ambiguity, which is introduced by the spelling correction methods when they offer several alternatives for correction.

To estimate the feasibility of the developed method several experiments have been performed. Initially we checked the values of precision, recall and performance of our regional approach with those provided by the Savary's proposal [140], which we consider one of the most efficient. These first tests were conducted on isolated words, i.e. without taking into account the context in which they appeared. The results in terms of performance were highly positive since the regional method was clearly superior to the global one. In relation to recall, the regional approach offers a lower average number of alternatives, leading to a slight decrease in precision.

Our next step was to check whether the loss of precision of the regional method could be offset in a contextual spelling correction environment, given that the fact that it returned a smaller number of alternatives would have positive impacts on overall system precision [161]. Our experiments have not confirmed this hypothesis, but have served to highlight that the increase in performance of the regional method, in terms of space and time with respect to the global one, was even greater when we applied these techniques in a contextual correction environment. This was because, in addition to being more efficient from the computational point of view, the regional algorithm offers fewer alternatives for correction. This encourages us to continue searching for techniques and heuristics that allow us to determine when it is possible to choose a regional correction.

Finally, we have performed tests to verify the practical usefulness of our proposal in an IR environment in which queries have misspellings. To do this, we have compared three methods [118]. The first is to expand queries with all the options for correction. The second applies our contextual spelling correction method to determine which of the alternatives fits best in the context of the wrong word. The third avoids the application of spelling correction methods using n-grams for indexing and retrieval. The results of these tests have confirmed that the application of spelling correction techniques improves the results in the presence of degraded queries. Moreover, the use of n-grams appears to be a very robust technique and has the advantage of not requiring any extra linguistic resource.

In summary, our work is a proposal that covers the design, implementation and evaluation in a practical environment of an original technique for automatic correction of lexical errors in natural languages. The results to date are promising and serve to direct our future work in this concrete field.

Part-of-Speech tagging

Some languages, like Galician[1] or Spanish, show complex phenomena which have to be handled 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`, `P23` and `P33` even when they constitute only one term

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.

We are going to explain here a method to deal with 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

[1] The co-official language in Galicia, an autonomous community in the northwest of Spain.

(*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, it will be necessary 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, Graña *et al.* [58] proposed a method which extend the Viterbi [167] algorithm in order to evaluate streams of tokens of different lengths over the same structure.

Viterbi-L: the Viterbi Algorithm on Lattices

In the context of part-of-speech tagging with HMMs, the classic version of the Viterbi algorithm [167] 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 figure 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.

Figure 1: Trellises cannot represent ambiguous segmentations

The kind of ambiguous segmentations described above can be represented 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 figure 2, where we can see that it is even possible to represent overlapped dependencies.

This lattice contains 20 arcs, with which 234 possible paths can be built. 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).

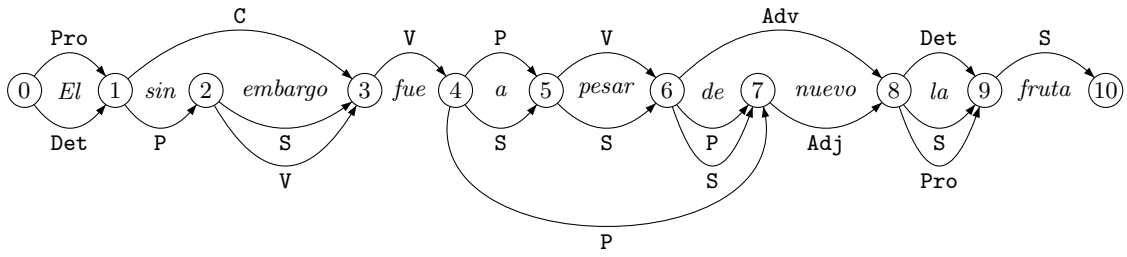

Figure 2: Ambiguous segmentations represented on a lattice

The Viterbi-L Equations

The equations of the Viterbi algorithm [167] can be adapted to process a language model operating on a lattice as follows [20]. Instead of the words, the gaps between the words are enumerated (see figure 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:

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

- 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 $(t'', t, q') \in \text{Lattice}$ 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, the goal is to estimate the parameters of the HMM from tagged texts, and use linear interpolation of uni-, bi-, and trigrams as smoothing technique [61], 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).

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 figure 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 figure 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.

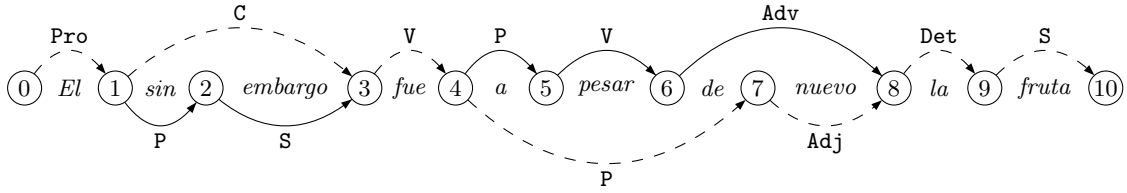

Figure 3: A lattice with conflictive paths

However, the number of different lattices to evaluate is not always the number of different lengths for the paths. For instance, as can be seen in figure 3 two paths of the same length[2] can produce another path with different length [3]. Therefore, more than one lattice could be needed to represent the paths of the same length without conflicts. In the case of figure 2, although there are 4 possible lengths for the paths[4], a total of 6 lattices are needed, which come from the mutual exclusion of the different alternatives of each ambiguous segmentation. The real space and time complexities of the algorithm are in

[2]The one that only uses the upper arcs of the lattice, and the one that uses lower arcs when available, both of 8 tokens.

[3]The one of 7 tokens marked with dashed arcs.

[4]7, 8, 9 or 10 tokens.

this case 82 accumulators and 119 operations.

Viterbi-N: the Viterbi Algorithm with Normalization

The goal 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, it would be necessary to keep as many accumulators as there are different lengths of all the paths reaching the point of origin of the arc. This information about lengths is incorporated 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 in the maximization operations to obtain the corresponding $\Delta_{t',t'',l+1}(q')$. When reaching the final instant, there will be 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.

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:

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

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

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

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

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).

Viterbi-N vs. Viterbi-L

By using intuition again, the space complexity of the Viterbi-N algorithm can be also considered 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 figure 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.

Spelling Correction

An ongoing question in natural language processing (NLP) is how to recover ungrammatical structures for processing text. Focusing on spelling correction tasks, there are few things more frustrating than spending a great deal of time debugging typing or other errors in order to ensure the accuracy of NLP tools over large amount of data. As a consequence, although it is one of the oldest applications to be considered in the field of NLP [87], there is an increased interest in devising new techniques in this area.

In this regard, previous proposals extend the repair region to the entire string, complemented with the consideration of thresholds on an editing distance [116, 140]. This global approach, which seems to be universally accepted, has probably been favored by the consideration of English, a non-concatenative language with a reduced variety of morphological associated processes [147], as running language. However, the application of this kind of techniques to highly inflectional languages such as Latin ones [29], or agglutinative languages such as Turkish [146], could fail to take advantage of the underlying grammatical structure, leading to a significant loss of efficiency.

In this context, we are interested in exploring regional repair techniques, introducing proper tasks for error location and repair region estimation. Our aim is to avoid examining the entire word, in contrast to global algorithms that expend equal effort on all parts of the word, including those containing no errors.

The operational model

Our aim is to parse a word $w_{1..n} = w_1 \dots w_n$ according to a regular grammar $\mathcal{G} = (N, \Sigma, P, S)$, where N is the set of non-terminals, Σ the set of terminal symbols, P the rules and S the start symbol. We denote by w_0 (resp. w_{n+1}) the position in the string, $w_{1..n}$, previous to w_1 (resp. following w_n). We generate from $\mathcal{G}$ a *numbered minimal acyclic finite state automaton* for the language $\mathcal{L}(\mathcal{G})$. In practice, we choose a device [94] generated using Galena [59]. A *finite automaton* (FA) is a 5-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input symbols, δ is a function of $\mathcal{Q} \times \Sigma$ into $2^{\mathcal{Q}}$ defining the transitions of the automaton, q_0 the initial state and $\mathcal{Q}_f$ the set of final states. We denote $\delta(q, a)$ by $q.a$, and we say that the FA is *deterministic* when, in any case, $\mid q.a \mid \leq 1$. The notation is transitive, so $q.w$ denotes the state reached by using the transitions labelled

by each letter $w_i,\ i \in \{1, ..., n\}$ of w. Therefore, w is *accepted* iff $q_0.w \in \mathcal{Q}_f$, that is, the *language accepted by* $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A}) = \{w,$ such that $q_0.w \in \mathcal{Q}_f\}$. An FA is *acyclic* when the underlying graph is. We talk about a *path in the* FA to refer to a sequence of states $\{q_1, \ldots, q_n\}$, such that $\forall i \in \{1, \ldots, n-1\},\ \exists a_i \in \Sigma,\ q_i.a_i = q_{i+1}$.

In order to reduce the memory requirements, we minimize the FA [35]. So, we say that two FAs are *equivalent* iff they recognize the same language. Two states, p and q, are *equivalent* iff the FA with p as initial state and the one that starts in q recognize the same language. An FA is *minimal* iff no pair in $\mathcal{Q}$ is equivalent.

It is important to note that although the standard recognition process is deterministic, the repair process could introduce non-determinism by exploring alternatives associated to possibly more than one recovery strategy. So, in order to get polynomial complexity, we avoid duplicating intermediate computations in the repair of $w_{1..n} \in \Sigma^+$, storing them in a table $\mathcal{I}$ of *items*, $\mathcal{I} = \{[q, i],\ q \in \mathcal{Q},\ i \in [1, n+1]\}$, where $[q, i]$ looks for the suffix $w_{i..n}$ to be analyzed from $q \in \mathcal{Q}$.

We describe our proposal using *parsing schemata* [143], a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{H} = \{[a, i],\ a = w_i\}$ an initial set of items called *hypothesis* that encodes the word to be recognized[5], and $\mathcal{D}$ a set of *deduction steps* that allow new items to be derived from already known items. Deduction steps are of the form $\{\eta_1, \ldots, \eta_k \vdash \xi \,/\, conds\}$, meaning that if all antecedents η_i are present and the conditions *conds* are satisfied, then the consequent ξ is generated. In our case, $\mathcal{D} = \mathcal{D}^{\text{Init}} \cup \mathcal{D}^{\text{Shift}}$, where:

$$\mathcal{D}^{\text{Init}} = \{\ \vdash [q_0, 1]\ \} \qquad \mathcal{D}^{\text{Shift}} = \{[p, i] \vdash [q, i+1] \,/\, \exists [a, i] \in \mathcal{H},\ q = p.a\}$$

The recognition associates a set of items S_p^w, called *itemset*, to each $p \in \mathcal{Q}$; and applies these deduction steps until no new application is possible. The word is recognized iff a *final item* $[q_f, n+1],\ q_f \in \mathcal{Q}_f$ has been generated. We can assume, without lost of generality, that $\mathcal{Q}_f = \{q_f\}$, and that exists an only transition from (resp. to) q_0 (resp. q_f). To get it, we augment the original FA with two states becoming the new initial and final states, and relied to the original ones through empty transitions, a concession to the minimality.

The edit distance

The *edit distance* [92] between two strings measures the minimum number of editing operations of insertion, deletion, replacement of a symbol, and transposition of adjacent symbols that are needed to convert one string into another. Let $x_{1..m}$ (resp. $y_{1..n}$) be the misspelled string (resp. a possible partial candidate string), the edit distance, $ed(x, y)$ is computed as follows:

[5] A word $w_{1...n} \in \Sigma^+,\ n \geq 1$ is represented by $\{[w_1, 1],\ [w_2, 2],\ \ldots\ ,\ [w_n, n]\}$.

$$
\begin{array}{lcl}
\mathrm{ed}(x_{i+1}, y_{j+1}) & = & \begin{cases}
\mathrm{ed}(x_i, y_j) & & \text{iff } x_{i+1} = y_{j+1} \\
 & & \text{(last characters are the same)} \\
1 + \min\{ & \mathrm{ed}(x_{i-1}, y_{j-1}), & \\
 & \mathrm{ed}(x_{i+1}, y_j), & \\
 & \mathrm{ed}(x_i, y_{j+1})\} & \text{iff } x_i = y_{j+1}, x_{i+1} = y_j \\
 & & \text{(last two characters are transposed)} \\
1 + \min\{ & \mathrm{ed}(x_i, y_j), & \\
 & \mathrm{ed}(x_{i+1}, y_j), & \\
 & \mathrm{ed}(x_i, y_{j+1})\} & \text{otherwise}
\end{cases} \\
\mathrm{ed}(x_0, y_j) & = & j \qquad\qquad 1 \leq j \leq n \\
\mathrm{ed}(x_i, y_0) & = & i \qquad\qquad 1 \leq i \leq m
\end{array}
$$

where x_0 (resp. y_0) is ε. We can now extend the concept of language accepted by an FA $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$, to define the *language accepted by an* FA $\mathcal{A}$ *with an error threshold* $\tau > 0$ as $\mathcal{L}_\tau(\mathcal{A}) = \{x, \text{ such that } ed(x, y) \leq \tau,\ y \in \mathcal{L}(\mathcal{A})\}$. We shall consider the edit distance as a common metrical basis in order to allow an objective comparison to be made between our proposal and previous ones.

Regional least-cost error repair

We talk about the *error* in a portion of the word to mean the difference between what was intended and what actually appears in the word. So, we can talk about the *point of error* as the point at which the difference occurs.

Definición 0.1. *Let* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *be an* FA*, and let* $w_{1..n}$ *be a word. We say that* w_i *is a* point of error *iff it verifies the following conditions:*

$$(1)\ \ q_0.w_{1..i-1} = q \qquad (2)\ \ q.w_i \notin \mathcal{Q}$$

□

The point of error is fixed by the recognizer and it provides the starting point for the repair, in which the following step consists in locating the origin of that error. We aim to limit the impact on the prefix already analyzed, focusing on the context close to the point of error and saving on computational effort. To do so, we first introduce a collection of topological properties that we illustrate in figure 4.

Definición 0.2. *Let* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *be an* FA*, and let* $p, q \in \mathcal{Q}$*. We say that* p is lesser than q *iff there exists a path* $\{p, \ldots, q\}$*. We denote that by* $p < q$*.* □

We have, in figure 4, that $q_i < q_{i+1},\ \forall i \in \{1, \ldots, 7\}$. Our order is induced by the transitional formalism, which results in a well defined relation since our FA is acyclic. In this sense, we can also give a direction to the paths.

Definición 0.3. *Let* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *be an* FA*, we say that* $q_s \in \mathcal{Q}$ *(resp.* q_d*) is a* source *(resp.* drain*)* state *for any path in* $\mathcal{A}$*,* $\{q_1, \ldots, q_m\}$*, iff* $\exists a \in \Sigma$*, such that* $q_1 = q_s.a$ *(resp.* $q_m.a = q_d$*).* □

Intuitively, we talk about source (resp. drain) states on out-coming (resp. incoming) transitions, which orientates the paths from sources to drains. So, in figure 4, q_1 (resp. q_8) is a source (resp. drain) for paths $\{q_9\}$, $\{q_2, q_{10}, q_6, q_7\}$, $\{q_2, q_3, q_{11}, q_5, q_6, q_7\}$ or $\{q_2, q_3, q_4, q_5, q_6, q_7\}$. We can now consider a coverage for FAs by introducing the concept of *region*.

Definición 0.4. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, we say that a pair (q_s, q_d), $q_s, q_d \in \mathcal{Q}$ is a* region *in $\mathcal{A}$, denoted by $\mathcal{R}_{q_s}^{q_d}(\mathcal{A})$, iff it verifies that*

(1) $q_s = q_0$ **and** $q_d = q_f$ *(the global* FA*)*
or
(2) $\{\forall \rho,\ source(\rho) = q_s\} \Rightarrow drain(\rho) = q_d$ **and** $\mid \{\forall \rho,\ source(\rho) = q_s\} \mid > 1$

which we write as $\mathcal{R}_{q_s}^{q_d}$ when the context is clear. We also denote $\text{paths}(\mathcal{R}_{q_s}^{q_d}) = \{\rho / source(\rho) = q_s,\ drain(\rho) = q_d\}$ *and, given $q \in \mathcal{Q}$, we say that $q \in \mathcal{R}_{q_s}^{q_d}$ iff $\exists \rho \in paths(\mathcal{R}_{q_s}^{q_d}),\ q \in \rho$.* □

This allows us to ensure that any state, with the exception of q_0 and q_f, is included in a region. Applied to figure 4, the regions are $\mathcal{A} = \mathcal{R}_{q_0}^{q_f}$, $\mathcal{R}_{q_1}^{q_8}$, $\mathcal{R}_{q_2}^{q_7}$ and $\mathcal{R}_{q_3}^{q_5}$, with $\{q_4, q_{11}, q_{12}\} \subset \mathcal{R}_{q_3}^{q_5} \not\ni q_3$ and $\mathcal{R}_{q_2}^{q_7} \ni q_9 \notin \mathcal{R}_{q_3}^{q_5}$. In a region, all prefixes computed before the source can be combined with any suffix from the drain through the paths between both. This provides a criterion to place around a state a zone for which any change in it has no effect on its context.

Definición 0.5. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, we say that a region $\mathcal{R}_{q_s}^{q_d}$ is the* minimal region *in $\mathcal{A}$ containing $p \in \mathcal{Q}$ iff it verifies that $q_s \geq p_s$ (resp. $q_d \leq p_d$), $\forall \mathcal{R}_{p_s}^{p_d} \ni p$. We denote it as $\mathcal{M}(\mathcal{A}, p)$, or simply $\mathcal{M}(p)$ when the context is clear.* □

In figure 4, $\mathcal{M}(q_4) = \mathcal{M}(q_{11}) = \mathcal{R}_{q_3}^{q_5}$ and $\mathcal{M}(q_3) = \mathcal{M}(q_9) = \mathcal{R}_{q_2}^{q_7}$. At this point, it is trivial to prove the following lemma, which guarantees the consistence of the previous concept based on the uniqueness of a minimal region.

Lema 0.1. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, then $p \in \mathcal{Q} \setminus \{q_0, q_f\} \Rightarrow \dot{\exists}\, \mathcal{M}(p)$.*

Proof. Trivial from definition 0.5. □

We can now formally introduce the concept of *point of detection*, the point at which the recognizer detects that there is an error and calls the repair algorithm.

Definición 0.6. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, and let w_j be a point of error in $w_{1..n} \in \Sigma^+$. We say that w_i is a* point of detection associated to w_j *iff:*

$$\exists q_d > q_0.w_{1..j},\ \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}_{q_0.w_{1..i}}^{q_d}$$

We denote this by $\text{detection}(w_j) = w_i$*, and we say that $\mathcal{M}(q_0.w_{1..j})$ is the* region defining the point of detection *w_i.* □

In our example in figure 4, if we assume w_j to be a point of error such that $q_{10} = q_0.w_{1..j}$, we conclude that $w_i = \text{detection}(w_j)$ if $q_2 = q_0.w_{1..i}$ since $\mathcal{M}(q_{10}) = \mathcal{R}^{q_7}_{q_2}$. So, the error is located in the immediate left recognition context, given by the closest source. However, we also need to locate it from an operational viewpoint, as an item in the computational process.

Definición 0.7. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, let w_j be a point of error in $w_{1..n} \in \Sigma^+$, and let w_i be a point of detection associated to w_j. We say that $[q, j] \in S^w_q$ is an* error item *iff $q_0.w_{j-1} = q$; and we say that $[p, i] \in S^w_p$ is a* detection item associated to w_j *iff $q_0.w_{i-1} = p$.* □

Following our running example in figure 4, $[q_2, i]$ is a detection item for the error item $[q_{10}, j]$. Intuitively, we talk about error and detection items when they represent states in the FA concerned with the recognition of points of error and detection, respectively. Once we have identified the beginning of the repair region from both the topological and the operational viewpoint, we can now apply the *modifications* intended to recover the recognition process from an error.

Definición 0.8. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, a* modification to $w_{1..n} \in \Sigma^+$ *is a series of edit operations, $\{E_i\}^n_{i=1}$, in which each E_i is applied to w_i and possibly consists of a sequence of insertions before w_i, replacement or deletion of w_i, or transposition with w_{i+1}. We denote it by $M(w)$.* □

We now use the topological structure to restrict the notion of modification, introducing the concept of *error repair*. Intuitively, we look for conditions that guarantee the ability to recover the standard recognition, at the same time as they allow us to isolate repair branches by using the concept of path in a region.

Definición 0.9. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA*, $x_{1..m}$ a prefix in $\mathcal{L}(\mathcal{A})$, and $w \in \Sigma^+$, such that xw is not a prefix in $\mathcal{L}(\mathcal{A})$. We define a* repair of w following x *as $M(w)$, so that:*

(1) $\mathcal{M}(q_0.x_{1..m}) = \mathcal{R}^{q_d}_{q_s}$ *(the minimal region including the point of error, $x_{1..m}$)*
(2) $\exists\{q_0.x_{1..i} = q_s.x_i, \ldots, q_s.x_{i..m}.M(w)\} \in \mathit{paths}(\mathcal{R}^{q_d}_{q_s})$

We denote it by $\text{repair}(x, w)$*, and $\mathcal{R}^{q_d}_{q_s}$ by* $\text{scope}(M)$. □

However, the notion of *repair*(x, w) is not sufficient for our purposes, since our aim is to extend the recovery process to consider all possible repairs associated to a given point of error, which implies simultaneously considering different prefixes.

Definición 0.10. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let $y_i \in y_{1..n}$ be a point of error, we define the* set of repairs for y_i*, as*

$$\text{repair}(y_i) = \{xM(w) \in \mathit{repair}(x, w)/w_1 = \mathit{detection}(y_i)\}$$

□

We now need a mechanism to filter out undesirable repair processes, in order to reduce the computational charges. To do so, we should introduce comparison criteria to select only those repairs with minimal cost.

Definición 0.11. *For each $a, b \in \Sigma$ we assume insert, $I(a)$; delete, $D(a)$, replace, $R(a,b)$, and transpose, $T(a,b)$, costs. The* cost of a modification $M(w_{1..n})$ *is given by* $\text{cost}(M(w_{1..n})) = \Sigma_{j \in J_\dashv} I(a_j) + \Sigma_{i=1}^{n}(\Sigma_{j \in J_i} I(a_j) + D(w_i) + R(w_i, b) + T(w_i, w_{i+1}))$, *where* $\{a_j,\ j \in J_i\}$ *is the set of insertions applied before* w_i; $w_{n+1} = \dashv$ *the end of the input and* $T_{w_n,\dashv} = 0$. □

In order to take edit distance as the error metric for measuring the quality of a repair, it is sufficient to consider discrete costs $I(a) = D(a) = 1,\ \forall a \in \Sigma$ and $R(a,b) = T(a,b) = 1,\ \forall a, b \in \Sigma,\ a \neq b$. On the other hand, when several repairs are available on different points of detection, we need a condition to ensure that only those with the same minimal cost are taken into account, looking for the best repair quality. However, this is not in contradiction with the consideration of error thresholds or alternative error metrics.

Definición 0.12. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let $y_i \in y_{1..n}$ be a point of error, we define the set of* regional repairs for y_i, *as follows:*

$$\textit{regional}(y_i) = \{xM(w) \in \textit{repair}(y_i) \Big/ \begin{array}{l} \textit{cost}(M) \leq \textit{cost}(M'),\ \forall M' \in \textit{repair}(x,w) \\ \textit{cost}(M) = \min_{L \in \textit{repair}(y_i)}\{\textit{cost}(L)\} \end{array} \}$$

□

It is also necessary to take into account the possibility of cascaded errors, that is, errors precipitated by a previous erroneous repair diagnosis. Prior to dealing with the problem, we need to establish the existing relationship between the regional repairs for a given point of error and future points of error.

Definición 0.13. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let w_i, w_j be points of error in $w_{1..n} \in \Sigma^+$, $j > i$. We define the set of* viable repairs for w_i in w_j, *as*

$$\textit{viable}(w_i, w_j) = \{xM(y) \in \textit{regional}(w_i) / xM(y) \ldots w_j \textit{ prefix for } \mathcal{L}(\mathcal{A})\}$$

□

Intuitively, the repairs in $\textit{viable}(w_i, w_j)$ are the only ones capable of ensuring the continuity of the recognition in $w_{i..j}$ and, therefore, the only possible repairs at the origin of the phenomenon of cascaded errors.

Definición 0.14. *Let w_i be a point of error for $w_{1..n} \in \Sigma^+$, we say that a point of error w_k, $k > j$ is a* point of error precipitated by w_j *iff*

$$\forall xM(y) \in \textit{viable}(w_j, w_k),\ \exists \mathcal{R}^{q_d}_{q_0.w_{1..i}} \textit{ defining } w_i = \textit{detection}(w_j)$$

such that $\textit{scope}(M) \subset \mathcal{R}^{q_d}_{q_0.w_{1..i}}$. □

In practice, a point of error w_k is precipitated by the result of previous repairs on a point of error w_j, when the region defining the point of detection for w_k summarizes all viable repairs for w_j in w_k. This implies that the information compiled from those repair regions has not been sufficient to give continuity to a recognition process locating the new error in a region containing the preceding ones and, therefore, depending on them. That is, the underlying grammatical structure suggests that the origin of the current error could be a mistaken treatment of past errors. Otherwise, the location would be fixed in a zone not depending on these previous repairs.

The algorithm

We propose that the repair be obtained by searching the FA itself to find a suitable configuration to allow the recognition to continue, a classic approach in error repair. However, in the state of the art there is no theoretical size limit for the repair region, but only for the edit distance on corrections in it. So, in order to avoid distortions due to unsafe error location, the authors make use of global algorithms limiting the computations by a threshold on the edit distance. This allows them to restrict the section of the FA to be explored by pruning either all repair paths which are more distant from the input than the threshold [116], or those not maintaining a minimal distance no bigger than the threshold [140].

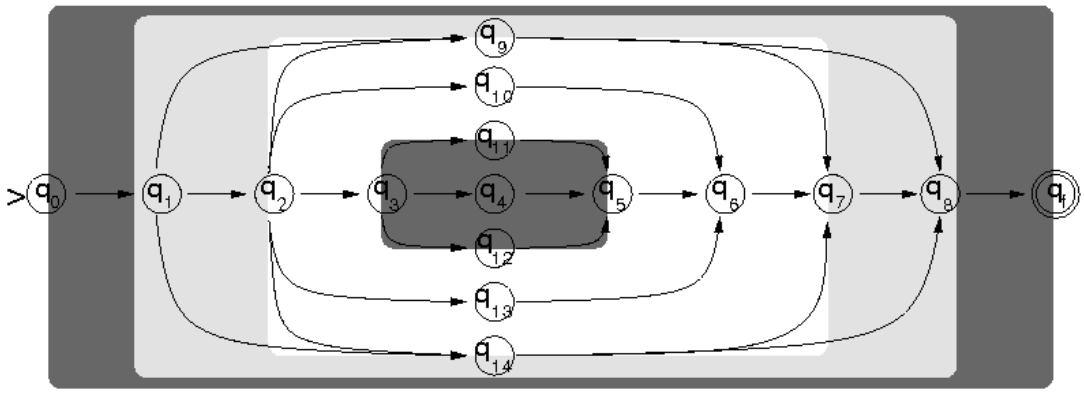

Figure 4: The concept of region applied to error repair

However, the fact that we are not profiting from the linguistic knowledge present in the FA to locate the error and to delimit its impact may lead to suboptimal computational costs or to precipitating new errors. We eliminate this problem by a construction where all repair phases are dynamically guided by the FA itself and, therefore, inspired by the underlying grammatical structure.

A simple case

We assume that we are dealing with the first error detected in a word $w_{1..n} \in \Sigma^+$. The major features of the algorithm involve beginning with the error item, whose error counter

is zero. So, we extend the item structure, $[p, i, e]$, where e is now the error counter accumulated in the recognition of w at position w_i in state p.

We refer again to figure 4. So, given an error item, $[q_{10} = q_0.w_{1..j}, j, e_j]$, the system locates the corresponding detection item, $[q_2 = q_0.w_{1..i}, i, e_i]$, by using a pointer on $\mathcal{M}(q_{10}) = \mathcal{R}^{q_7}_{q_2}$. We then apply all possible transitions in this region beginning at both, the point of error and the its associated point of detection, which corresponds to the following deduction steps in error mode, $\mathcal{D}_{\text{error}} = \mathcal{D}^{\text{Shift}}_{\text{error}} \cup \mathcal{D}^{\text{Insert}}_{\text{error}} \cup \mathcal{D}^{\text{Delete}}_{\text{error}} \cup \mathcal{D}^{\text{Replace}}_{\text{error}} \cup \mathcal{D}^{\text{Transpose}}_{\text{error}}$:

$$\mathcal{D}^{\text{Shift}}_{\text{error}} = \{[p, i, e] \vdash [q, i+1, e], \ \exists [a, i] \in \mathcal{H}, \ q = p.a\}$$
$$\mathcal{D}^{\text{Insert}}_{\text{error}} = \{[p, i, e] \vdash [p, i+1, e + I(a)], \ \nexists\, p.a\}$$
$$\mathcal{D}^{\text{Delete}}_{\text{error}} = \{[p, i, e] \vdash [q, i-1, e + D(w_i)] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}^{q_d}_{q_s} \\ p.w_i = q_d \in \mathcal{R}^{q_d}_{q_s} \textbf{or } q = q_d \end{array} \}$$
$$\mathcal{D}^{\text{Replace}}_{\text{error}} = \{[p, i, e] \vdash [q, i+1, e + R(w_i, a)], \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}^{q_d}_{q_s} \\ p.a = q \in \mathcal{R}^{q_d}_{q_s} \textbf{ or } q = q_d \end{array} \}$$
$$\mathcal{D}^{\text{Transpose}}_{\text{error}} = \{[p, i, e] \vdash [q, i+2, e + T(w_i, w_{i+1})] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}^{q_d}_{q_s} \\ p.w_i.w_{i+1} = q \in \mathcal{R}^{q_d}_{q_s} \textbf{ or } q = q_d \end{array} \}$$

where $w_{1..j}$ looks for the current point of error. Note that, in any case, the error hypotheses apply on transitions behind the repair region. The process continues until a repair covers the repair region, accepting a character in the remaining string. Returning to figure 4, the scope of repair for the error detected at $w_i \in \textit{detection}(w_j)$ is $\mathcal{M}(q_{10}) = \mathcal{R}^{q_7}_{q_2}$, the region defining the detection item $[q_2 = q_0.w_{1..i}, i, e_i]$. Once this has been performed on each recognition branch, we select the regional repairs and the process goes back to standard mode.

The general case

We now assume that the repair process is not the first one in the word and, therefore, can modify a previous one. This arises when we realize that we come back to a detection item for which some recognition branch includes a previous repair process. To illustrate such a case, we return to figure 4 assuming $[q_{10} = q_0.w_{1..k}, k, e_k]$ and $[q_8 = q_0.w_{1..l}, l, e_l]$ to be points of error. As a consequence, $[q_8 = q_0.w_{1..l}, l, e_l]$ would be precipitated by $[q_{10} = q_0.w_{1..k}, k, e_k]$ since $\mathcal{A} = \mathcal{R}^{q_f}_{q_0}$ defining $w_0 = \text{detection}(w_l)$ includes $\mathcal{R}^{q_7}_{q_2 = q_0.w_{1..j}}$, the scope of a previous repair.

To deal with precipitated errors, the algorithm re-takes the previous error counters, adding the cost of the new repair hypotheses to profit from the experience gained from previous recovery phases. At this point, regional repairs have two important properties. First, they are independent of the FA construction and secondly, there is no loss of efficiency in relation to global repair approaches.

Lema 0.2. *(The Expansion Lemma) Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let w_k, w_l be points of error in $w_{1..n} \in \Sigma^+$, such that w_l is precipitated by w_k, then:*

$$q_0.w_{1..i} < q_0.w_{1..j}, \ \mathcal{M}(q_0.w_l) = \mathcal{R}^{q_d}_{q_0.w_{1..i}}, \ w_j = y_1, \ xM(y) \in \textit{viable}(w_k, w_l)$$

Proof. Let $w_j \in \Sigma$, such that $w_j = y_1, \ xM(y) \in \textit{viable}(w_k, w_l)$ be a point of detection for w_k, for which some recognition branch derived from a repair in $\textit{regional}(w_k)$ has

successfully arrived at w_l. Let also w_l be a point of error precipitated by $xM(y) \in \mathit{viable}(w_k, w_l)$. By definition 0.14, we can affirm that

$$\text{scope}(M) \subset \mathcal{M}(q_0.w_l) = \mathcal{R}^{q_d}_{q_0.w_{1..i}}$$

Given that $\mathit{scope}(M)$ is the lowest region summarizing $q_0.w_{1..j}$, it follows that $q_0.w_{1..i} < q_0.w_{1..j}$. We conclude the proof by extending it to all repairs in $\mathit{viable}(w_k, w_l)$. □

Intuitively, we prove that the state associated to the point of detection in a cascaded error is lesser than the one associated to the source of the scope in the repairs precipitating it. As a consequence, the minimal possible scope of a repair for the cascaded error includes any scope of those previous repairs.

Corolario 0.1. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let w_k, w_l be points of error in $w_{1..n} \in \Sigma^+$, such that w_l is precipitated by w_k, then*

$$\mathit{max}\{\text{scope}(M),\ M \in \mathit{viable}(w_k, w_l)\} \subset \mathit{max}\{\text{scope}(\tilde{M}),\ \tilde{M} \in \mathit{regional}(w_l)\}$$

Proof. It immediately follows from lemma 0.2. □

This allows us to get an asymptotic behavior close to global repair methods. That is, the algorithm ensures a quality comparable to global strategies, but at the cost of a local one. This has profound implications for the efficiency, measured by time, the simplicity and the power of computing regional repairs.

Lema 0.3. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let w_i be a point of error in $w_{1..n} \in \Sigma^+$, the time bound for the regional repair is, in the worst case,*

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^\tau * \mathit{fan\text{-}out}^\tau_\mu)$$

where τ and fan-out$_\mu$ *are, respectively, the maximal error counter computed and the maximal fan-out of the automaton in the scope of the repairs considered.*

Proof. Here, the proof is a simple extrapolation of the estimation proposed for the Savary's algorithm [140]. In the worst case, there are at most $n!/(\tau! * (n-\tau)!)$ possible distributions of τ modifications over n word positions. For each distribution $(1 + 2 * \text{fan-out}_\mu)^\tau$ paths at most are followed, each path being of length $n + \tau$ at most. So, the worst case complexity is the one proposed. □

However, this lemma does not yet determine the relation with classic global approaches [116, 140], as is our aim, but only an average case estimation of our own time complexity. To reach this, we extend the repair region to the total FA.

Corolario 0.2. *Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ be an* FA *and let w_i be a point of error in $w_{1..n} \in \Sigma^+$, the time bound for the regional repair is, in the worst case, the same reached for a global approach.*

Proof. It immediately follows from the previous lemma 0.3 and corollary 0.1, as well as [140]. In effect, in the worst case, the scope of the repair is the global FA. □

Taking into account the kind of proof applied on lemma 0.3, this implies that our technique has the same time complexity claimed for Savary's global one [140], in the best of our knowledge the most efficient proposal on spelling correction.

Experimental results

We are interested in both computational and quality aspects. In this sense, we consider the concept of item previously defined in order to measure the computational effort. To take into account data related to the performance from both the user's and the system's viewpoint, we have introduced the following two measures, for a given word, w, containing an error:

$$performance(w) = \frac{useful\ items}{total\ items} \qquad recall(w) = \frac{proposed\ corrections}{total\ corrections}$$

that we complement with a global measure on the *precision* of the error repair approach in each case, that is, the rate reflecting when the algorithm provides the correction attended by the user. We use the term *useful items* to refer to the number of generated items that finally contribute to the obtaining of a repair, and *total items* to refer to the number of these structures generated during the process. We denote by *proposed corrections* the number of corrections provided by the algorithm, and by *total corrections* the number of possible ones, in absolute terms.

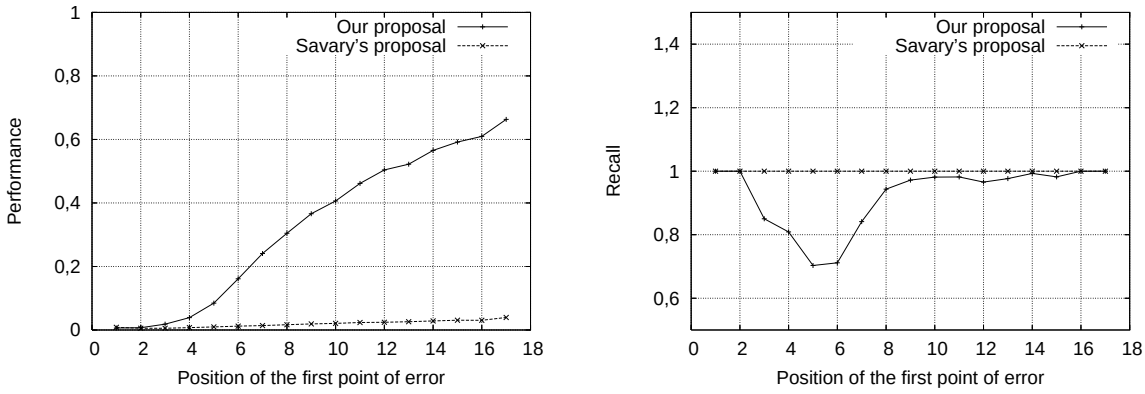

Figure 5: Performance and recall results.

The practical results shown in figure 5 appear to corroborate that not only the performance in our case is better than Savary's, but also that the difference existing between them increases with the location of the first point of error. With respect to the recall relation, Savary's algorithm shows a constant graph since the approach applied is global and, consequently, the set of corrections provided is always the entire one for a fixed error counter. In our proposal, the results prove that the recall is smaller than that for Savary's, which illustrates the gain in computational efficiency in comparison with the

global method. Finally, the precision of the regional (resp. the global) method is of 77% (resp. 81%). We must remember that here we are only taking into account morphological information, which has an impact on precision for a regional approach, but not for a global one, which always provides all possible repair alternatives. So, a precision measure represents a disadvantage for our proposal since we base efficiency on limitation of the search space. The future integration of linguistic information from both syntactic and semantic viewpoints should significantly reduce this gap in precision, which is less than 4%, or may even eliminate it.

Contextual spelling correction

A classic problem in spelling correction is how to rank the correction candidates provided by the spelling correction algorithm. Some systems avoid tackling this task by offering the user the possibility of choosing from a set of possible corrections for a given misspelt word. This is possible for interactive applications like word processors but, for most NLP tools, spelling correction is one of the first tasks to be performed and small and ranked sets of alternatives are welcome in order to minimize the overload in later phases.

In order to rank the repair alternatives offered by the spelling correction algorithm some additional information is needed. The stochastic part-of-speech tagger, previously introduced, allows us to take advantage of syntactic and lexical information embedded in probabilities of transition between tags and emission probabilities of words.

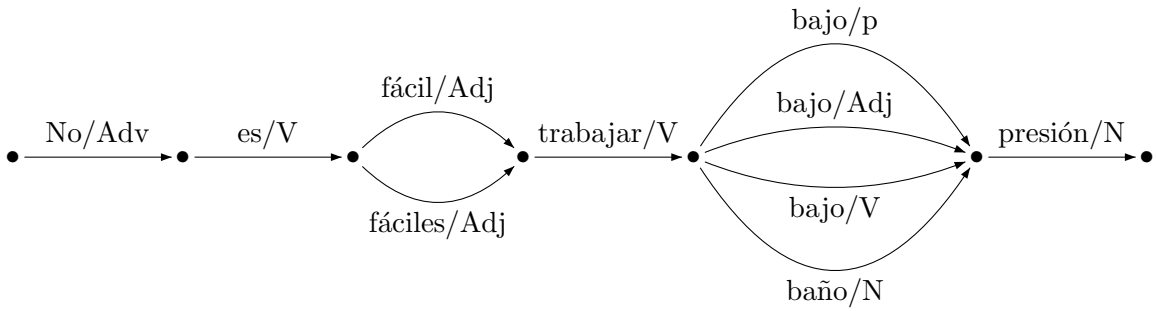

Figure 6: Spelling correction alternatives represented on a lattice

We can observe some similarities between segmentation disambiguation and contextual spelling correction. The first aims to choose the best segmentation alternative while the second aims to choose the best correction alternative and, in both cases, only contextual information extracted from adjacent tokens is taken into account. So lattices also constitute an excellent dynamic framework for the contextual spelling correction task.

In order to use lattices for contextual spelling correction, a slight modification must be made in relation to the lattice shown in figure 2, where arcs are only labeled with the POS tag corresponding to the words covered by it. In contextual spelling correction, arcs must be labelled with *word/tag* pairs because more than one correction alternative has to be represented in the same gap.

Let us consider e.g. the sentence `No es fácile* trabajar baio* presión` (*It is not easy to work under pressure*), where the words `fácile` and `baio` are misspellings. Let us assume that our spelling corrector provides `fácil/Adj`, and `fáciles/Adj` as possible corrections for `fácile*`, and `bajo/Adj`, `bajo/P`, `bajo/V` and `baño/N` for `baio*`. figure 6 shows a lattice representing this situation. An execution of the dynamic Viterbi's algorithm over this lattice provides us with the tags of the words, and also the most probable spelling corrections in the context of this concrete sentence. Depending on the application, a ranked list of correction candidates may be required. To provide this, computed probability for each path in the lattice may be used.

Experimental results

Our experiments have been performed on a set of sentences of the Spanish Clic-Talp corpus [141], in which we have introduced random *human-like* errors by means of the tool Misplel [15].

In a first set of tests, we introduce 1 mistake per sentence. In this case, the average number of paths that have to be evaluated by the tagger is 5 per sentence, and the results for the two spelling correction algorithms are: the correct word is present in the list of alternatives provided by the global spelling corrector in 96.3% of cases (95% for regional approach); in those situations, the correct word is selected by the tagger in 93.9% of cases; therefore, by combining these indicators, we obtain a global performance of 90.5% of the sentences completely recovered (89.6% for regional approach).

In a second set of tests, we introduce at most 3 mistakes per sentence. In this case, the average number of paths to be evaluated by the tagger increases to 84 paths per sentence, and the percentage of the sentences that are completely recovered decreases to a still respectable 80.9% (78.6% for the regional approach).

In relation to the spelling correction algorithm, we observe some interesting data. The global strategy obtains a slightly better global performance but if we center our attention on those cases where both algorithms return the correct word among the set of alternatives, the percentage of correct choices by the tagger is greater for the regional approach because the set of alternatives provided is smaller.

Information retrieval

Although formal models for *information retrieval* (ir) are designed for well-spelled corpora and queries, useful questioning should be robust against corruption phenomena and out-of-vocabulary words, in order to avoid increasing silence due to missing information, imprecision, inconsistency or uncertainty. So, tolerant retrieval becomes a priority in the design of ir applications, particularly when we are dealing with highly dynamic databases that continuously change over time, perhaps making machine-discovered knowledge inconsistent. More intuitively, we could say that such imperfection is a fact of life in this kind of systems, now largely popularized through web services.

A major factor at the root of these problems is the introduction of spelling errors by the user, either by accident, or because the term he is searching for has no unambiguous

spelling in the collection. It is therefore imperative to study this problem in query languages, since it can substantially hinder performance. In this sense, most authors directly apply error correction techniques on lexical forms in order to provide IR tools with a robust querying facility.

In this section, we propose and evaluate two different alternatives to deal with degraded queries on Spanish IR applications. The first one consists in using the spelling correction techniques introduced in the previous section. On the other hand, we propose a character n-gram-based strategy which has no dependence in extra linguistic resources. In order to study their validity, a testing framework has been formally designed and applied on both approaches. We are interested in a methodology that is simple and ready for use independently of the documentary database considered and the linguistic resources available

Our approach has initially been tested for Spanish. This language can be considered a representative example since it shows a great variety of morphological processes, making it a hard language for spelling correction [162].

Error Processing

The first phase of the evaluation process consists of introducing spelling errors in the test topic set. These errors were randomly introduced by an automatic error-generator according to a given error rate. Firstly, a *master error file* is generated as explained below. For each topic word with a length of more than 3 characters, one of the four edit errors described by Damerau [36] is introduced in a random position of the word. This way, introduced errors are similar to those that a human writer or an OCR device could make. At the same time, a random value between 0 and 100 is generated. Such a value represents the probability of not containing a spelling error. This way we obtain a master error file containing, for each word, its corresponding misspelled form, and a probability value.

All these data make it possible to easily generate different test sets for different error rates, allowing us to evaluate the impact of this variable on the output results. Such a procedure consists of reading the master error file and selecting, for each word, the original form in the event of its probability being higher than the fixed error rate, or than the misspelled form in the other case. So, given an error rate T, only $T\%$ of the words of the topics should contain an error. An interesting feature of this solution is that the errors are incremental, since the misspelled forms which are present for a given error rate continue to be present for a higher error rate, thereby avoiding any distortion in the results.

The next step consists of processing these misspelled topics and submitting them to the IR system. In the case of our n-gram-based approach no extra resources are needed, since the only processing consists of splitting them into n-grams. However, for correction-based approaches, a lexicon is needed, and in the particular case of our contextual corrector, a manually disambiguated training corpus is also needed for training the tagger. We have chosen to work with the MULTEX-JOC Spanish corpus and its associated lexicon. The MULTEX-JOC corpus [156] is a part of the corpus developed within the MULTEXT

project[6] financed by the *European Commission*. This part contains raw, tagged and aligned data from the *Written Questions and Answers* of the *Official Journal of the European Community*. The corpus contains approximately 1 million words per language: English, French, German, Italian and Spanish. About 200,000 words per language were grammatically tagged and manually checked for English, French, Italian and Spanish. Regarding the lexicon of the Spanish corpus, it contains 15,548 words that, once compiled, build an automaton of 55,579 states connected by 70,002 transitions.

The Evaluation Framework

The open-source TERRIER platform [120] has been employed as the retrieval engine of our system, using an InL2[7] ranking model [7]. With regard to the document collection used in the evaluation process, we have chosen to work with the Spanish corpus of the CLEF 2006 robust task [112],[8] which is formed by 454,045 news reports (1.06 GB). More in detail, the test set consists of the 60 training topics[9] established for that task. Topics are formed by three fields: a brief *title* statement, a one-sentence *description*, and a more complex *narrative* specifying the relevance assessment criteria. Nevertheless, only the *title* field has been used in order to simulate the case of short queries such as those used in commercial engines. Taking this document collection as input, two different indexes are then generated.

In order to test our correction-based proposal, a classical stemming-based approach is used for both indexing and retrieval. We have chosen to work with SNOWBALL,[10] from the Porter's algorithm [124], while the stop-word list used was that provided by the University of Neuchatel.[11] Both approaches are commonly used in IR research. Following Mittendorfer *et al.* [107, 108], a second list of so-named meta-stop-words has also been used for queries. Such stop-words correspond to meta-level content, i.e. those expressions corresponding to query formulation without giving any useful information for the search, as is the case of the query *"encuentre aquellos documentos que describan ..."* (`"find those documents describing ..."`).

On the other hand, for testing our n-gram-based approach, documents are lowercased, and punctuation marks, but not diacritics, are removed. The resulting text is split and indexed using 4-grams, as a compromise on the n-gram size after studying the previous results of the JHU/APL group [102]. No stop-word removal is applied in this case.

Experimental results

Our proposal has been tested for a wide range of error rates, T, in order to study the behavior of the system not only for low error densities, but also for high error rates

[6] `http://www.lpl.univ-aix.fr/projects/multext`

[7] Inverse Document Frequency model with Laplace after-effect and normalization 2.

[8] These experiments must be considered as unofficial experiments, since the results obtained have not been checked by the CLEF organization.

[9] C050–C059, C070–C079, C100–C109, C120–C129, C150–159, C180–189.

[10] `http://snowball.tartarus.org`

[11] `http://www.unine.ch/info/clef/`

existing in noisy and very noisy environments like those where the input are obtained from mobile devices or based on handwriting (i.e., tablet computing):

$$T \in \{0\%, 10\%, 20\%, 30\%, \ldots, 100\%\}$$

where T=0% means no extra errors have been introduced.

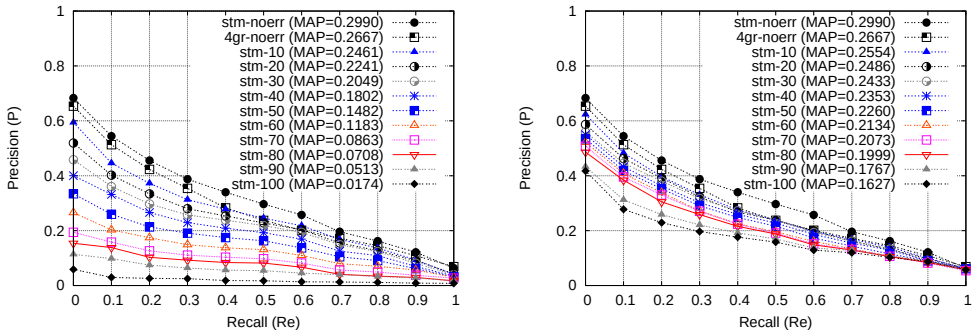

Figure 7: Precision vs. Recall for misspelled (non-corrected) topics: using stemming-based retrieval (left), using n-gram-based retrieval (right).

The first set of experiments performed were those using the misspelled (non-corrected) topics in the case of a classical stemming-based approach. The results obtained for each error rate T are shown in the graph of figure 7 (on the left) taking as baselines both the results for the original topics —i.e., for T=0%— (*stm-noerr*), and those obtained for such original topics but when using our n-gram based approach (*4gr-noerr*). Notice that *mean average precision* (MAP) values are also given. This first results show stemming to be sensitive to misspellings. As can be seen, even a low error rate such as T=10% has a significant impact on performance —MAP decreases by 18%—, which increases as the number of errors introduced grows: 25% loss for T=20%, 50% for T=50% (with 2 queries no longer retrieving documents) and 94% for T=100% (13 queries no longer retrieving documents), for example. All this is due to the fact that with the kind of queries like those we are using here —4 words on average—, each single word is of key importance, since the information lost when one term does not match because of a misspelling cannot be recovered from any other term.

Our second round of experiments tested the behavior of the system when using the first of the correction approaches considered in this work, that is, when submitting the misspelled topics once they have been processed by our simple spelling correction method. The correction module takes as input the misspelled topic, obtaining as output a corrected version where each misspelled word has been replaced by the closest term in the lexicon, according to its edit distance. In the event of a tie —more than one candidate word at the same closest edit distance—, the query is expanded with all corrections. For example, taking as input the sample sentence previously considered in figure 6, *"No es fácile trabajar baio presión"*, the output returned would be *"No es fácil fáciles trabajar bajo baño presión"*. On analysis, the results obtained, shown in figure 8 (on the left),

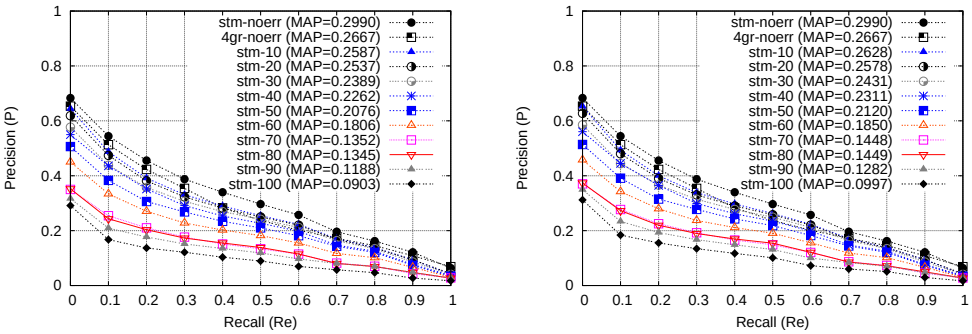

Figure 8: Precision vs. Recall for misspelled topics using stemming-based retrieval and spelling correction: expanding queries with all all corrections (left), using our contextual correction approach (right).

indicate that correction has a clear positive effect on performance, greatly diminishing —although not eliminating— the impact of misspellings, not only for low error rates (MAP losses diminish from 18% to 13% for T=10% and from 25% to 15% for T=20%), but even for high-very high error rates (from 50% to 31% for T=50% and from 94% to 70% for T=100%), as well as reducing the number of queries not retrieving documents (now only 1 for T=50% and 5 for T=100%). Data analyses also show that the relative effectiveness of correction increases at the same time as the error rate.

In order to try to remove noise introduced by ties when expanding queries with all the alternatives offered by the spellchecker, a third set of tests has been performed using our contextual spelling corrector. These results are shown in figure 8 (on the right) and, as expected, results consistently improve with respect to the original approach, although little improvement is attained through this extra processing (an extra 2% MAP loss recovery for 10%$\leq T \leq$60%), except for very-noisy environments (7–10% loss recovery for T¿60%).

Finally, we have tested our n-gram-based proposal. So, figure 8 shows the results when the misspelled (non-corrected) topics are submitted to our n-gram-based IR system. As can be seen, although stemming performs better than n-grams for the original queries, the opposite is the case in the presence of misspellings, the latter not only clearly outperforming stemming when no correction is applied, but also outperforming correction-based approaches —except for the very lowest error rates. Moreover, the robustness of this n-gram-based proposal in the presence of misspellings proves to be far superior to that of any of the previous stemming-based approaches. As an example, MAP losses for simple stemming —as stated before— were by 18% for T=10%, 25% for T=20%, 50% for T=50% and 94% for T=100%; for the same T values, the application of our contextual spelling corrector —which was slightly superior to Savary's proposal— reduced such losses to 12%, 14%, 29% and 67%, respectively; however, n-grams outperform both significantly, nearly halving these latter losses: 4%, 7%, 15% and 39%, respectively. Furthermore, there are no queries not retrieving documents, even for T=100%.

Tests have shown that classic stemming-based approaches are very sensitive to spelling errors, although the use of correction mechanisms allows the negative impact

of misspellings on system performance to be reduced. On the other hand, character n-grams have proved to be highly robust, clearly outperforming correction-based techniques, particularly at medium or higher error rates. Moreover, since it does not rely on language-specific processing, our n-gram-based approach can be used with languages of very different natures even when linguistic information and resources are scarce or unavailable.

Índice general

Índice de figuras

Índice de algoritmos

Índice de tablas

Lista de Abreviaturas

AF	Autómata Finito
AFD	Autómata Finito Determinista
AFDAN	Autómata Finito Determinista Acíclico Numerado
AFND	Autómata Finito no Determinista
AF-ε	Autómata Finito con transiciones nulas
BDS	Base de Datos Sintácticos del Español
CLEF	*Cross-Language Evaluation Forum*
COLE	COmpiladores y LEnguajes
CORGA	Corpus de Referencia do Galego Actual
CRATER	*Corpus Resources And Terminology ExtRaction*
CRPIH	Centro Ramón Piñeiro de Investigación en Humanidades
DARPA	*Defense Advanced Research Projects Agency*
FEDER	Fondos Europeos para el Desarrollo Regional
FPI	Formación de Personal Investigador
GB	GigaByte
HMM	Modelo de Markov Oculto (*Hidden Markov Model*)
IR	Recuperación de Información (*Information Retrieval*)
ITU	*International Telecommunications Union*
LyS	Lengua y Sociedad de la Información
JHU/APL	*Johns Hopkins University Applied Physics Lab*
MAP	*Mean Average Precision*
MMO	Modelo de Markov Oculto
NIST	*National Institute of Standards and Technology*
NLP	Procesamiento de Lenguaje Natural (*Natural Language Processing*)
OCR	Reconocimiento Óptico de Caracteres (*Optical Character Recognition*)
PDA	Asistente Personal Digital (*Personal Digital Assistant*)
PLN	Procesamiento de Lenguaje Natural
RAE	Real Academia de La Lengua Española
RI	Recuperación de Información
ROC	Reconocimiento Óptico de Caracteres
TREC	*Text REtrieval Conference*

Capítulo 1

Introducción

> "*El lenguaje es el medio que hace posible la formulación de preguntas y respuestas. La estructura del conocimiento es lingüística. La estructura de la conciencia es lingüística. La estructura del razonamiento es lingüística. La estructura del mundo, tal como lo concibe y utiliza el hombre, es lingüística. El lenguaje es el lugar de lo humano, en él vivimos, nos movemos y somos.*"
>
> José Manuel Briceño Guerrero
> *El origen del lenguaje*

En efecto, el lenguaje es la herramienta que hace posible la representación, organización, manejo y transmisión del conocimiento. Las civilizaciones conocidas más antiguas disponían de algún método que les permitiese transmitir su experiencia y sabiduría a las generaciones posteriores con el fin de garantizar una continua evolución.

Nuestra civilización no es una excepción y seguimos utilizando el lenguaje con el mismo fin. Sin embargo, la aparición de los ordenadores y la frenética y constante evolución de la informática ha dado lugar a lo que conocemos como "*Sociedad de la Información*". Hoy en día se generan y publican cantidades ingentes de información en formato electrónico que debe ser procesada y estructurada para permitir un acceso rápido y eficaz a la misma. Las empresas y organizaciones dedican buena parte de sus recursos a estructurar, almacenar y procesar la información necesaria para su funcionamiento cotidiano. Para ello utilizan bases de datos en las que la información debe ser introducida, usualmente, con excesiva rigidez respetando las restricciones formales y de integridad que este tipo de estructuras imponen. A pesar de ello, no pueden evitar que buena parte de la información relevante para su actividad permanezca almacenada de forma no estructurada, es decir, en formato texto.

Por otra parte, no podemos olvidarnos de la *web*, que ha venido a revolucionar el acceso a la información y en la que la mayor parte de los contenidos se encuentra con escaso o, simplemente, sin ningún tipo de estructura.

También debemos mencionar aquí el desafío recurrente de conseguir que las personas puedan comunicarse con las máquinas en lenguaje humano y que éstas sean capaces de extraer conocimiento a partir directamente del lenguaje natural.

Por tanto, se hace necesario contar con mecanismos eficaces desde el punto de vista computacional que permitan dar solución a los retos planteados y es así como surge el *Procesamiento del Lenguaje Natural* (PLN)[1] como una subdisciplina de la Inteligencia Artificial y la rama ingenieril de la lingüística computacional. De forma genérica, podemos estructurar el ámbito de trabajo del PLN en 4 niveles de análisis:

- **Léxico** : Se encarga de reconocer las unidades mínimas del lenguaje, que denominaremos *unidades o categorías léxicas* y que suelen coincidir con las palabras que componen las frases u oraciones.

- **Sintáctico** : Se encarga de reconocer unidades gramaticales formadas por varias unidades léxicas que se combinan entre sí, formando un árbol de análisis sintáctico que expresa las relaciones estructurales entre aquéllas.

- **Semántico** : Se encarga de capturar el significado de una frase por medio de técnicas de representación del conocimiento.

- **Pragmático** : Añade información al análisis del significado de la frase en función del contexto en que ésta aparece.

Aunque en los últimos años se han realizado importantes avances, los fundamentos teóricos del PLN se encuentran todavía en constante evolución. En este sentido, resulta de especial interés el desarrollo de tecnología de base, imprescindible para abordar tareas de mayor nivel como la traducción automática, elaboración automática de resúmenes, interfaces en lenguaje natural, *búsqueda de respuestas* y, en particular la *recuperación de información* (RI). En este sentido, y como primer paso en esta escala de problemas, centramos nuestros esfuerzos en el desarrollo y mejora de técnicas que permitan manejar el léxico. Más concretamente, nos centramos en el desarrollo de un marco común que nos permita representar y resolver los tres tipos de ambigüedades que podemos encontrarnos en este nivel de análisis y que son:

- *Ambigüedad segmental* : Hace referencia a aquellas situaciones en que la división de una frase en unidades léxicas puede realizarse de al menos dos formas distintas, bien sea porque cabe la posibilidad de que varias palabras consecutivas formen o no una única unidad léxica, como es el caso de la expresión "*a pesar de*" que podría ser una locución adverbial o una preposición seguida de un verbo y otra preposición; o porque una palabra pueda descomponerse en más de una unidad léxica, como ocurre por ejemplo con la palabra gallega "polo" que puede ser un nombre ("*pollo*") o puede descomponerse en "*por o*" ("*por el*") o "*pos o*" ("*lo pones*").

- *Ambigüedad morfosintáctica* : Ocurre cuando a una misma unidad léxica pueden serle asignadas más de una etiqueta morfosintáctica. Este tipo de ambigüedad suelen resolverla los etiquetadores morfosintácticos.

[1] Usualmente denominado por el término inglés *Natural Language Processing* (NLP).

- *Ambigüedad léxica* : Utilizaremos este término para referirnos a la situación que se produce cuando, al utilizar técnicas de corrección ortográfica, obtenemos varias alternativas de corrección para una palabra errónea determinada.

Para resolver estos casos de ambigüedad es necesario manejar, además de las propias unidades léxicas implicadas, aquéllas que conforman su contexto inmediato.

1.1 Motivación y objetivos de la tesis

El objetivo principal de esta tesis ha sido el desarrollo y evaluación de la tecnología de base necesaria para el PLN, más concretamente en el ámbito del análisis léxico, la corrección ortográfica y la etiquetación, continuando y complementando el trabajo que se ha venido realizando en este ámbito en el seno de los grupos de investigación *Compiladores y Lenguajes* (COLE) de la Universidad de Vigo y *Lengua y Sociedad de la Información* (LyS) de la Universidad de A Coruña.

Nuestros mayores esfuerzos se han centrado en el desarrollo de un nuevo método de corrección ortográfica regional sobre *autómatas finitos* (AF) como alternativa a los métodos de corrección global clásicos propuestos por Oflazer [116] y Savary [140], integrando las técnicas desarrolladas en la herramienta de etiquetación MrTagoo [57, 58, 60] con el fin de aprovechar la información morfosintáctica contextual embebida en el modelo estocástico que subyace en dicha herramienta, y determinar el grado de idoneidad de las alternativas de corrección obtenidas.

Por otra parte, la minimización del coste computacional, tanto desde el punto de vista espacial como temporal, ha sido prioritaria a lo largo de todo el proyecto mediante el uso de tecnología de estado finito y la integración de los métodos implementados sobre una estructura de datos común, aplicando técnicas de programación dinámica que redundan en un importante ahorro al evitar la repetición innecesaria de cálculos.

1.2 Ámbito de la tesis

El trabajo desarrollado en esta tesis doctoral se enmarca en el PLN cuyo objetivo fundamental es facilitar la comunicación entre las personas y las máquinas mediante el lenguaje humano. Asimismo, las técnicas aquí presentadas entran dentro de los campos de la Teoría de Autómatas y Lenguajes Formales, por constituir los AFs el núcleo sobre el que éstas han sido desarrolladas.

En lo que respecta al contexto en el que se ha desarrollado este proyecto de tesis, el trabajo de investigación presentado aquí es el fruto de becas, proyectos y estancias de investigación que se relacionan a continuación:

Becas y contratos de investigación

- **Universidade de Vigo**. Ayudas para la Investigación *Beca de Tercer Ciclo*. (01/01/2002 – 31/12/2002).

- **Xunta de Galicia**. Bolsas de colaboración en Proyectos de Investigación del Centro Ramón Piñeiro para la Investigación en Humanidades (CRPIH)[2]. *Beca predoctoral de investigación*. (15/07/2003 – 30/09/2003).

- **Xunta de Galicia**. Programa de Recursos Humanos del Plan Gallego de Investigación, Desarrollo e Innovación Tecnológica. *Beca predoctoral*, del 1/10/2003 al 31/12/2005.

- **Ministerio de Educación y Ciencia**. Programa Nacional de Potenciación de Recursos Humanos del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica. *Beca predoctoral de Formación de Personal Investigador (*FPI*)*, (TIN2004-07246-C03-01). (01/01/2006 – 14/09/2007).

- **Ministerio de Educación y Ciencia**. Programa Nacional de Potenciación de Recursos Humanos del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica. *Contrato en prácticas de Formación de Personal Investigador (*FPI*)*, (TIN2004-07246-C03-01). (15/09/2007 – 14/09/2009).

Contratos de I+D con empresas

- Programas Sectoriales de Recursos Naturales, Tecnologías para la Innovación y Servicios al Ciudadano del Plan Gallego de Investigación, Desarrollo e Innovación Tecnológica. Generación, extracción y estructuración de información legislativa mediante técnicas de inteligencia artificial (GENEEIA). (PGIDIT05SIN044E). (01/10/2005 – 30/09/2006).

- Programas Sectoriales de Investigación Aplicada, PYME I+D e I+D Suma del Plan Gallego de Investigación, Desarrollo e Innovación Tecnológica. Modelo para la vinculación y el almacenamiento de objetivos sobre documentos XML de legislación. (PGIDIT07SIN028E). (2007-2008).

Proyectos de I+D de ámbito nacional

- Extracción de información económica multilingüe. (TIN2004-07246-C03-01 y PGIDIT05PXIC30501PN). (13/12/2004 – 12/12/2007).

- Búsqueda de respuestas empleando metagramáticas (HUM2007-66607-C04-02). (2007 – 2010)

Proyectos de I+D de ámbito autonómico

- Evaluación interactiva de la relevancia en entornos de recuperación automática de información (PGIDIT-02PXIB30501PR). (01/01/2002 – 31/12/2005).

[2] http://www.cirp.es/

- Mejora en la recuperación de noticias y en el acceso a información financiera: recuperación de textos sobre bases documentales de agencias de noticias (07SIN005206PR). (2007 – 2010).

Proyectos de I+D de ámbito local y de la universidad

- Aplicación de técnicas de procesamiento del lenguaje natural en sistemas de búsqueda de respuestas sobre documentos técnicos. (27/07/2005 – 26/07/2006).

- Herramientas para el análisis y clasificación automática de documentos legislativos(2005-INOU-09). (01/03/2006 – 30/11/2006).

Redes temáticas

- Red Temática REDEGRID: Red Gallega de Computación Paralela, Distribuida e Tecnologías GRID (PGIDIT-PR462A-02/4). (23/07/2002 – 22/07/2003)

- Red Gallega de Procesamiento del Lenguaje y recuperación de información (2006/23). (01/01/2006 – 31/12/2008).

Estancias de Investigación

- *Facultade de Ciências e Tecnología de la Universidade Nova de Lisboa, Portugal*. Se ha realizado una estancia de seis meses de duración en el Departamento de Informática que dirige *D. Luís Moniz Pereira*, Dr. *honoris causa* por la Technology University Dresden de Alemania, bajo la tutela del *Dr. Gabriel Pereira Lopes*, investigador de reconocido prestigio en el campo de PLN. El tema de esta estancia ha sido la aplicación de información contextual a la corrección ortográfica.

- *Facultad de Informática de la Universidad de A Coruña*. Esta estancia, de seis meses de duración, ha transcurrido en el Departamento de Computación de la Universidad de A Coruña bajo la tutela del *Dr. Jorge Graña Gil*, codirector de este trabajo.

- *Facultade de Filología de la Universidad de Santiago de Compostela*. En esta estancia de tres meses de duración se ha trabajado en desambiguación segmental y aplicación práctica del algoritmo de Viterbi sobre retículos en la segmentación y la etiquetación. El contexto en el que ha transcurrido es el Departamento de Lengua Española, bajo la tutela del *Dr. Guillermo Rojo Sánchez*, catedrático de lengua española, ocupa el sillón "N" de la *Real Academia de La Lengua Española* (RAE) de la que fue secretario, especialista en teoría sintáctica del español y experto en Lingüística Informática.

1.3 Estructura de la memoria

Esta memoria se estructura en tres partes. En la primera se realiza una extensa introducción a la etiquetación del lenguaje natural profundizando en los etiquetadores

estocásticos y en los *Modelos de Markov Ocultos* (MMOs)[3] [98]. En la segunda se introducen algunos conceptos relacionados con el ámbito de la RI, necesarios para la correcta comprensión de los experimentos realizados en este campo. La tercera parte constituye el verdadero núcleo de este proyecto. En ella se realiza una introducción a la corrección ortográfica sobre AFs y se detalla el desarrollo de un nuevo método de corrección regional integrándolo con las técnicas de etiquetación y desambiguación segmental presentadas en la primera parte, construyendo así un analizador léxico robusto. Finalmente se presentan algunos resultados experimentales que corroboran la utilidad de los algoritmos desarrollados, y se recogen las conclusiones y las líneas de trabajo futuro.

A continuación se muestra un breve resumen de cada uno de los capítulos que conforman esta memoria.

1.3.1 Parte I: Etiquetación del lenguaje natural

Capítulo 2. Introduce una serie de nociones y notaciones relacionados con los lenguajes regulares y los AFs, imprescindibles para seguir la explicación de los algoritmos que se presentarán a lo largo de la presente memoria.

Capítulo 3. En este capítulo se realiza una breve introducción a la etiquetación del lenguaje natural, describiendo el problema y resumiendo la historia y evolución de los etiquetadores para finalizar enumerando algunas de las aplicaciones clásicas de la etiquetación.

Capítulo 4. En este capítulo nos centraremos en las técnicas utilizadas a lo largo de este trabajo para realizar el análisis léxico, describiendo en profundidad el modelo utilizado para la implementación del lexicón, así como los métodos que permiten un acceso eficiente al mismo [57].

Capítulo 5. En este capítulo estudiaremos los (MMOs), un método estocástico comúnmente utilizado en sistemas de etiquetación. Se enunciarán los tres problemas canónicos asociados a éstos, explicando en profundidad el algoritmo de Viterbi [167] como solución a uno de ellos. Finalmente, se describirá el problema de la desambiguación segmental y se propondrá, como solución al mismo, una variante del algoritmo de Viterbi sobre retículos [58], en lugar de los enrejados clásicos.

1.3.2 Parte II: Recuperación de información

Capítulo 6. Con el objetivo de facilitar la plena comprensión de los experimentos realizados, se incluye este capítulo de introducción y familiarización con los conceptos básicos relacionados con la RI. Tras delimitar el ámbito de aplicación de este tipo de sistemas, se describe el funcionamiento de uno de ellos: los paradigmas de representación interna y comparación de documentos y consultas, así como el proceso de generación de

[3] Usualmente conocidos por su nombre en inglés *Hidden Markov Models* (HMMs).

dichas representaciones internas y su implementación. Esto nos sirve de base para describir la indexación y la recuperación basadas en n-gramas de caracteres superpuestos.

Capítulo 7. En este capítulo nos centraremos en la evaluación de la efectividad de los sistemas de RI, diferenciando dos aspectos: por una parte las medidas de evaluación empleadas y, por otra, las colecciones de referencia necesarias para dicha evaluación.

1.3.3 Parte III: Corrección ortográfica

Capítulo 8. En este capítulo nos remontaremos a los orígenes de la corrección ortográfica automática. Presentaremos los diferentes métodos utilizados a lo largo de los últimos cuarenta años, indicando las novedades que éstos aportaban respecto a los anteriores. Comenzaremos describiendo las técnicas de corrección basadas en n-gramas en contraposición a aquéllas que hacen uso de diccionarios, para explicar a continuación las técnicas de corrección basadas en la distancia de edición, y terminar haciendo mención a técnicas basadas en similaridad de claves, en reglas, en n-gramas, en redes neuronales y técnicas probabilísticas.

Capítulo 9. Nos centraremos aquí en aquellas estrategias de corrección basadas en diccionarios. En este caso, adquiere una especial importancia la modelización del diccionario ya que de ella dependerá la eficiencia del algoritmo. Los AFs se convierten por tanto en una importante herramienta a la hora de modelar el diccionario como veremos en el capítulo 4. Estudiaremos aquí dos métodos de corrección global sobre AFs: El algoritmo de Oflazer [115, 116] y el algoritmo de Savary [140].

Capítulo 10. En este capítulo presentamos nuestro algoritmo de corrección ortográfica regional [162] que aplica heurísticas para reducir el coste computacional de los algoritmos globales y tratar, al mismo tiempo, de mejorar la calidad de la respuesta obtenida, reduciendo para ello el número de correcciones candidatas.

Capítulo 11. Describiremos algunas técnicas de corrección ortográfica contextual propuestas por otros autores para posteriormente detallar cómo el algoritmo de Viterbi [167] presentado en el capítulo 5 puede ser adaptado para conseguir una solución íntegra que nos permita manejar los tres tipos de ambigüedades que se describen a nivel léxico [117].

Capítulo 12. Mostraremos y analizaremos los resultados de los experimentos realizados durante la realización de este trabajo de investigación. Se describen aquí los recursos lingüísticos utilizados y se realiza una exhaustiva comparación entre el método regional desarrollado en el capítulo 10 y el método global propuesto por Savary [140]. También se realizan experimentos para verificar el comportamiento de nuestra técnica de corrección contextual descrita en el capítulo 11 y, finalmente, se detalla una metodología de generación de errores ortográficos que nos permite comparar la aplicación de corrección ortográfica contextual, en sistemas de RI basados en extracción de raíces, con la indexación y la

recuperación basadas en n-gramas de caracteres superpuestos, definiendo para ello un marco de evaluación adecuado.

Capítulo 13. Finalmente, se recogen en este capítulo las conclusiones y aportaciones realizadas en este trabajo de tesis, así como las líneas de investigación a explorar en el futuro.

1.4 Difusión de resultados

Durante el desarrollo de este proyecto de tesis se han ido realizando publicaciones en revistas y congresos tanto de ámbito nacional como internacional correspondientes a resultados parciales que se han ido obteniendo. Estas publicaciones se detallan a continuación:

Publicaciones en revistas de ámbito internacional

1. Manuel Vilares, Juan Otero, Fco. Mario Barcala y Eva Domínguez. **Automatic Spelling Correction in Galician**. In J. Luís Vicedo, Patricio Martínez-Barco, Rafael Muñoz y Maximiliano Saiz Noeda, editores, *Advances in Natural Language Processing*, volumen 3230 de *Lecture Notes in Artificial Intelligence*, páginas 45–57. Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2004.

2. Manuel Vilares, Juan Otero y Jorge Graña. **On asymptotic finite-state error repair**. In Alberto Apostolico y Massimo Melucci, editores, *String Processing and Information Retrieval*, volumen 3246, de *Lecture Notes in Computer Science*, páginas 271–272, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2004.

3. Manuel Vilares, Juan Otero y Jorge Graña. **Regional versus Global Finite-State Error Repair**. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volumen 3406 de *Lecture Notes in Computer Science*, páginas 120–131, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2005.

4. Manuel Vilares, Juan Otero y Jorge Graña. **Regional Finite-State Error Repair**. In Michael Domaratzki, Alexander Okhotin, Kai Salomaa y Sheng Yu, editores, *Implementation and Application of Automata*, volumen 3317 de *Lecture Notes in Computer Science*, páginas 269–280, Springer-Verlag, ISSN 0302-9743, Berlín-Heidelberg, 2005.

5. Manuel Vilares, Juan Otero y Jorge Graña. **Spelling Correction on Technical Documents**. In Roberto Moreno-Díaz, Franz Pichler y Alexis Quesada-Arencibia, editores, *Computer Aided Systems Theory – EUROCAST 2005*, volumen 3643 de *Lecture Notes in Computer Science*, páginas 131–139, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2005.

6. Manuel Vilares, Juan Otero y Jesus Vilares. **Robust Spelling Correction**. In Jacques Farré, Igor Litovsky y Sylvain Schmitz, editores, *Implementation and*

Application of Automata, volumen 3845 de *Lecture Notes in Computer Science*, páginas 319–328, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2006.

7. Manuel Vilares, Juan Otero y Victor M. Darriba. **Regional versus Global Robust Spelling Correction**. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volumen 3878 de *Lecture Notes in Computer Science*, páginas 575–585, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2006.

8. Juan Otero, Jorge Graña y Manuel Vilares. **Contextual Spelling Correction**. In Roberto Moreno-Díaz, Franz Pichler y Alexis Quesada-Arencibia, editores, *Computer Aided Systems Theory – EUROCAST 2007*, volumen 4739 de *Lecture Notes in Computer Science*, páginas 290–296, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2007.

9. Juan Otero, Jesús Vilares y Manuel Vilares. **Dealing with Corrupted Queries in Text Retrieval**. In Hector Geffner, Rui Prada, Isabel Machado Alexandre y Nuno David, editores, *Advances in Artificial Intelligence – IBERAMIA 2008*, volumen 5290 de *Lecture Notes in Computer Science*, páginas 362–371, Springer-Verlag, Berlín-Heidelberg, ISSN 0302-9743, 2008.

Contribuciones a congresos de ámbito internacional

1. Manuel Vilares, Juan Otero y Jorge Graña. **Spelling Correction on Technical Documents**. In Alexis Quesada-Arencibia, Roberto Moreno-Díaz jr. y José-Carlos Rodríguez, editores, *Cast and Tools for Robotics, Vehicular and Communication Systems*, *Proceedings of the tenth International Workshop on Computer Aided Systems Theory – EUROCAST 2005*, páginas 73–76, Las Palmas de Gran Canaria, ISBN 84-689- 0432-5, Febrero 2005.

2. Juan Otero, Jorge Graña y Manuel Vilares. **Contextual Spelling Correction**. In In Alexis Quesada-Arencibia, José-Carlos Rodríguez, Roberto Moreno-Díaz y Roberto Moreno-Díaz jr., editores, *Computer Aided Systems Theory*, *Proceedings of the tenth International Workshop on Computer Aided Systems Theory – EUROCAST 2007*, páginas 97–98, Las Palmas de Gran Canaria, ISBN 978-84-690-3603-7, Febrero 2007.

3. Miguel A. Molinero, F. Mario Barcala, Juan Otero and Jorge Graña. **Practical application of one-pass Viterbi algorithm in tokenization and part-of-speech tagging**. In Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nicolov y Nikolai Nicolov, editores, *Proceedings of the International Conference Recent Advances in Natural Language Processing*, páginas 35-40, Borovets, Bulgaria, ISBN 978-954-91743-7-3, septiembre 2007.

4. Juan Otero, Jesús Vilares y Manuel Vilares. **Corrupted Queries in Spanish Text Retrieval: Error Correction vs. N-Grams**. In Fotis Lazarinis, Efthimis N. Efthimiadis, Jesús Vilares y John Tait, editores, *Proceedings of ACM CIKM 2008*

Workshop on Improving Non-English Web Searching (iNEWS08), páginas 39-46, Napa Valley, California, USA, ISBN 978-1-60558-253-5, 2008.

Publicaciones en revistas de ámbito nacional

1. Jesús Vilares, Fco. Mario Barcala, Santiago Fernández, y Juan Otero. **Manejando la variación morfológica y léxica en la Recuperación de Información Textual**. *Procesamiento del Lenguaje Natural*, 30:99-106, ISSN 1135-5948, 2003.

1.5 Comunicación con el autor

Los comentarios y sugerencias acerca de esta memoria y del trabajo en ella reflejado son bienvenidos. Se puede contactar con el autor en la dirección

Juan Otero Pombo
Departamento de Informática
Escola Superior de Enxeñaría Informática
Campus de As Lagoas s/n
32.004 Ourense (España)

jop@uvigo.es
http://ccia.ei.uvigo.es/profesores/jop/

Existe información adicional referente a esta tesis y a trabajos relacionados en la página web del grupo COLE (*COmpiladores y LEnguajes*) en la url:

http://www.grupocole.org.

Parte I

Etiquetación del lenguaje natural

Capítulo 2

Definiciones y notaciones

En este capítulo introducimos una serie de nociones y notaciones relacionadas con los lenguajes regulares y los AFs, imprescindibles para seguir la explicación de los algoritmos que se presentarán posteriormente.

2.1 Lenguajes regulares

Para llegar a la definición de lenguaje regular debemos introducir primero una serie de conceptos previos. Comenzamos con el más simple y a la vez uno de los más importantes en la teoría de la computación, ya que a partir del mismo se definen y construyen buena parte de los demás.

Definición 2.1. *Un* alfabeto Σ *es un conjunto finito de elementos llamados* símbolos*.*

□

Como no puede ser de otra manera, la definición de alfabeto no difiere de la concepción general que todos tenemos. Tampoco lo hace la definición de cadena o palabra de un lenguaje que será una agrupación de símbolos del alfabeto.

Definición 2.2. *Una* cadena w *sobre un alfabeto* Σ *es una secuencia de cero o más símbolos del alfabeto. Para referirnos al símbolo que ocupa la posición* i *en la cadena* w *escribiremos* w_i*. La cadena que no contiene símbolos se denomina* cadena vacía *y se representa como* ε*. El conjunto de todas las cadenas definidas sobre* Σ*, incluida* ε*, se designa como* Σ^**; su cierre reflexivo-transitivo.*

□

Ejemplo 2.1. *Tomemos un alfabeto sencillo como el de los números binarios,* $\Sigma = \{0, 1\}$*. De este modo, cualquier número binario es una cadena formada por símbolos de* Σ*:*

$$01010, 0, 1, 0111, 1111, 00, \varepsilon \in \Sigma^*$$

O tomemos $\Sigma = \{a, b\}$*. Tendríamos cadenas como:*

$$aab, bac, a, bc, \varepsilon \in \Sigma^*$$

■

Una vez definidas las cadenas, resulta interesante señalar algunas de sus propiedades y operaciones antes de definir el concepto de lenguaje.

Definición 2.3. *Sea Σ un alfabeto y sea $w \in \Sigma^*$. Se define la* longitud *de w, denotada por $|w|$, como el número de símbolos de que consta. Para hacer constar la longitud de una cadena en su denominación podremos denotar en algunos casos la cadena w / $|w| = n$ como $w_{1..n}$.*

□

Ejemplo 2.2. *Con $\Sigma = \{0, 1\}$, tendríamos que:*

$$|\varepsilon| = 0 \qquad |0| = 1 \qquad |1| = 1 \qquad |010| = 3 \qquad |000000| = 6$$

■

Haciendo uso del concepto anterior, podemos realizar una definición recursiva del operador más común sobre cadenas, que será utilizado extensamente a lo largo de este texto: la concatenación.

Definición 2.4. *Sea Σ un alfabeto y sean $u, v \in \Sigma$. Se define la* concatenación *de* u *y* v*, denotada por uv, de la forma siguiente:*

1. *Si $|v| = 0$, $uv = u$*
2. *Si $|v| > 0$, tenemos que $v = wa$, con $a \in \Sigma$ y $|w| = n - 1$. Entonces, $uv = (uw)a$.*

□

Ejemplo 2.3. *Sea $\Sigma = \{a, b\}$. Si $u = ababab$ y $v = bbbbbb$, tenemos que $uv = abababbbbbb$.*

■

Es decir, uv es la cadena formada por los símbolos de u seguidos de los símbolos de v. La cadena vacía es el elemento neutro de este operador. En efecto, para cualquier cadena $u \in \Sigma^*$, tenemos que $u = u\varepsilon = \varepsilon u$. En base a este operador de concatenación podemos definir otros conceptos de interés en el ámbito de las cadenas.

Definición 2.5. *Sea Σ un alfabeto y sean $u, v \in \Sigma^*$. Se dice que* u *es una* subcadena *de* v *sii* u *está contenida en* v*, es decir si $\exists x, y \in \Sigma^*$ / $v = xuy$. Denotamos con $v_{n:m}$ la subcadena que va desde la posición* n *a la posición* m *de* v*, ambas incluidas.*

□

Definición 2.6. *Sea Σ un alfabeto y sean $u, v \in \Sigma^*$. Se dice que u es un* prefijo *de v sii $\exists w \in \Sigma^*$ / $v = uw$. Se dice que u es un* sufijo *de v sii $\exists w \in \Sigma^*$ / $v = wu$.*

□

Ejemplo 2.4. *Sea $\Sigma = \{a, b\}$, si $v =$* aaabbbaaa *tenemos que la cadena $u =$* abb *es una subcadena de u, dado que con $x =$* aa *e $y = baaa$ tenemos que $v = xuy$ y podríamos decir en este caso que $u = v_{3:5}$. De igual forma, si $v =$* abbaa*, $u =$* ab *sería un prefijo de v, ya que con $w =$* baa *tenemos que $v =$* vw*. Mientras que si $u =$* a*, tenemos que es un sufijo de v, dado que con $w =$* abba *se verifica que $v =$* wu*. La cadena vacía ε es prefijo y sufijo de cualquier cadena u, dado que $u = \varepsilon u = u\varepsilon$.*

■

De esta manera hemos definido satisfactoriamente cuales son los constituyentes básicos de un lenguaje, los símbolos y cadenas que lo forman, así como algunas de sus propiedades. Podemos, pues, centrarnos en el concepto mismo de lenguaje.

Definición 2.7. *Sea Σ un alfabeto, definimos un* lenguaje sobre Σ *como un subconjunto, finito o infinito, de Σ^*.*

□

Ejemplo 2.5. *Con el alfabeto binario $\Sigma = \{0, 1\}$, tenemos que Σ^* es el conjunto de cadenas formadas por símbolos binarios. Un posible lenguaje sobre Σ^* será el de los números binarios pares, formado por aquellas cadenas binarias terminadas en 0.*

Para $\Sigma = \{a, b\}$, el conjunto de cadenas formadas por un número de símbolos "a" seguido de idéntico número de símbolos "b", lo que se denota como $\{a^n b^n \ / \ n \in \mathbb{N}\}$, será un subconjunto de Σ^* y, por tanto, un lenguaje sobre Σ.

■

Dado que los lenguajes son conjuntos, todas las operaciones de conjuntos les son aplicables también. Así, la intersección, unión y diferencia de dos lenguajes sobre un alfabeto Σ dan como resultado lenguajes sobre Σ. El *complementario de un lenguaje L* sobre un alfabeto Σ es $\Sigma^* - L$, y también es un lenguaje sobre Σ.

Definición 2.8. *Sean L_1 y L_2 dos lenguajes, definimos la* concatenación de L_1 y L_2 *como $L_1L_2 = \{uv \ / \ u \in L_1 \text{ y } v \in L_2\}$. Esto es, L_1L_2 es el lenguaje por las cadenas obtenidas mediante concatenación de cada una de las cadenas de L_1 con cada una de las cadenas de L_2.*

□

Ejemplo 2.6. *Sea $L_1 = \{ab, ba\}$ y $L_2 = \{a, b\}$, entonces la concatenación de L_1 con L_2 será:*

$$L_1L_2 = \{aba, abb, baa, bab\}$$

■

Tal como se desprende de la definición, la concatenación de lenguajes, así como la de cadenas, no tiene la propiedad conmutativa, es decir $L_1L_2 \neq L_2L_1$. A partir de la concatenación de lenguajes definimos la *potencia k-ésima* de un lenguaje como la concatenación consigo mismo, k veces.

Definición 2.9. *Sea L un lenguaje, definimos la* potencia L^k, $k \geq 0$ *como sigue:*

1. $L^0 = \{\varepsilon\}$
2. $L^{k+1} = L^k L$

□

Ejemplo 2.7. *Sea* $L = \{a, b\}$*, entonces:*

$$L^2 = \{aa, ab, ba, bb\}\,,$$

$$L^3 = \{aaa, aba, baa, bba, aab, abb, bab, bbb\} \cdots$$

■

A partir de las operaciones que hemos visto podemos construir lenguajes tan complejos como deseemos a partir de otros más sencillos.

Estamos, pues, en posesión de una definición de lenguaje general. Ahora bien, una cosa es saber lo que un lenguaje es y otra obtener una representación particular realmente manejable. La forma más simple de lograr este objetivo es enumerar sus cadenas constituyentes. Pero este procedimiento no resulta muy práctico cuando el lenguaje consta de más de unas pocas cadenas o pretendemos definir propiedades o clasificaciones entre los lenguajes. De ahí que surja la necesidad de establecer algún mecanismo para generar lenguajes con una notación finita. Estos generadores son las *gramáticas*, representaciones formales adaptadas al tratamiento computacional.

Definición 2.10. *Una* gramática *es una 4-tupla* $\mathcal{G} = (N, \Sigma, P, S)$ *donde:*

- Σ *es el alfabeto finito de la gramática o conjunto finito de* símbolos terminales*, o* palabras*, o* categorías léxicas*.*
- N *es un conjunto finito de* símbolos no terminales*, o* variables*, o* categorías sintácticas*,* $N \cap \Sigma = \emptyset$*.*
- P *es un subconjunto finito de* $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ *a cuyos elementos denominaremos* producciones*,* reglas*, o* reglas de producción*.*
- $S \in N$ *es el* símbolo inicial*, o* axioma *de la gramática.*

□

En lo sucesivo, para unificar criterios y facilitar la lectura, utilizaremos las siguientes convenciones en la representación de los elementos de una gramática:

- $V = N \cup \Sigma$, es el conjunto total de símbolos.
- $a, b, c, \cdots \in \Sigma$, son símbolos terminales.
- $A, B, C, \cdots \in N$, son símbolos no terminales.
- $X, Y, Z, \cdots \in V$, son símbolos arbitrarios, terminales y no terminales.
- $u, v, w, \cdots \in \Sigma^*$, son cadenas de terminales.
- $u_{1..n}$, es una cadena $u \in \Sigma^*$, de longitud n.

- $u_{i:j}$, es la subcadena de $u_{1..n} \in \Sigma^*$ que va del carácter en la posición i al carácter en la posición j, ambos incluidos.
- u_i, es el carácter de $u \in \Sigma^*$ en la posición i.
- $\alpha, \beta, \gamma, \cdots \in V^*$, son cadenas arbitrarias de símbolos terminales y no terminales.
- ε, es la *cadena vacía*.
- Las producciones $(\alpha, \beta) \in P$ se escriben como $\alpha \to \beta \in P$

Ejemplo 2.8. *Una de las posibles definiciones de la gramática que genera el lenguaje de los números binarios pares del ejemplo 2.5, sería:*

$$\mathcal{G} = (\{S\}, \{0,1\}, \{S \to A0, A \to 0A, A \to 1A, A \to \varepsilon\}, S)$$

La gramática que genera el lenguaje del ejemplo 2.5 de las cadenas con cualquier número de símbolos "a" seguido del mismo número de símbolos "b", podría definirse como:

$$\mathcal{G} = (\{S\}, \{a,b\}, \{S \to aSb, S \to \varepsilon\}, S)$$

■

En un gramática, el símbolo inicial S es un símbolo distinguido de entre el conjunto de símbolos no terminales. Las producciones se pueden interpretar de dos maneras:

- Lectura descendente: α se deriva en β.
- Lectura ascendente: α se reduce a partir de β.

Al primer miembro α de una regla de producción "$\alpha \to \beta$" se le suele llamar *parte izquierda* de la regla de producción, mientras que el segundo miembro β recibe el nombre de *parte derecha* de la regla. A las reglas cuya parte derecha es la cadena vacía ε se les llama *reglas-ε* o *producciones-ε*. Cuando dos producciones "$\alpha \to \beta$" y "$\alpha \to \gamma$" tienen la misma parte izquierda, se pueden escribir abreviadamente como $\alpha \to \beta \mid \gamma$.

Por su parte, Σ es el alfabeto sobre el que se definen las cadenas del lenguaje que pueden ser generadas por la gramática. Para ello, en una lectura descendente, se parte del símbolo inicial de la gramática hasta llegar a una cadena de símbolos terminales, siendo las producciones las encargadas de describir las transformaciones necesarias para ello. En el caso ascendente, partiremos desde la cadena hasta obtener, mediante reducciones consecutivas, el axioma. En cualquier caso, a través de dichas producciones podemos generar secuencias de símbolos terminales y no terminales. Dependiendo de la forma de las reglas de producción, podremos obtener lenguajes más o menos complejos. De este modo, podemos clasificar los lenguajes en función de las gramáticas que los generan y, más concretamente, en función de la forma de dichas reglas de producción. Así, Chomsky [32] propone una jerarquía con cuatro clases fundamentales de gramáticas: regulares, independientes del contexto, dependientes del contexto y gramáticas con estructura de frase, siendo las regulares las más simples.

Definición 2.11. *Una gramática* $\mathcal{G} = (N, \Sigma, P, S)$ *se dice* lineal por la derecha *(resp.* por la izquierda*) sii* $\forall p \in P$ *es de la forma:*

$$1.\ A \to aB \ (resp.\ A \to Ba) \qquad\qquad 2.\ A \to a$$

□

Definición 2.12. *Una gramática* $\mathcal{G} = (N, \Sigma, P, S)$ *se dice* regular*, según la jerarquía de Chomsky [32], sii es lineal por la izquierda o por la derecha. Los lenguajes generados por este tipo de gramáticas se denominan* lenguajes regulares*.*

□

Ejemplo 2.9. *Podemos considerar el lenguaje formado por palabras con símbolos "a" y "b" alternos, comenzando por una "a". Una posible gramática para generar este lenguaje sería:*

$$\mathcal{G} = (\{S\}, \{a, b\}, \{S \to aA, A \to bS, A \to b\}, S)$$

Podemos ver como, en efecto, cumple con la definición de gramática regular. Las partes derechas de sus reglas de producción están formadas por un símbolo terminal, como en el caso de "$A \to b$*", o por un símbolo terminal seguido de un no terminal, como "*$S \to aA$*" ó "*$A \to bS$*".*

■

2.2 Autómatas finitos

Un AF es un modelo matemático de un sistema, con entradas y salidas discretas [122]. El sistema puede estar en cualquiera de las configuraciones internas finitas o estados. El estado del sistema dependerá de las entradas pasadas y determinará el comportamiento del sistema ante entradas futuras.

Definición 2.13. *Definimos formalmente un* autómata finito *(*AF*) como una 5-tupla* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$*, donde:*

- $\mathcal{Q}$ *es un conjunto finito de* estados*,*
- Σ *es un conjunto finito de símbolos de entrada, o* alfabeto de entrada*,*
- δ *es la* función de transición *que define las* transiciones *del autómata y es del tipo* $\mathcal{Q} \times \Sigma \longrightarrow \mathcal{P}(\mathcal{Q})$*,*
- $q_0 \in \mathcal{Q}$ *es el* estado inicial *del autómata, y*
- $\mathcal{Q}_f \subseteq \mathcal{Q}$ *es el conjunto de* estados finales *o de* aceptación*.*

□

Ejemplo 2.10. *Con el fin de ilustrar los conceptos y definiciones referentes a* AF*s que se realizarán a continuación, describiremos ahora cada uno de los componentes de un autómata de ejemplo:*

- $\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$,
- $\Sigma = \{0, 1\}$,
- *la función de transición* δ *se describe en el cuadro 2.1*
- *El estado inicial es* q_0, *y*
- $\mathcal{Q}_f = \{q_1\}$

δ	0	1
q_0	$\{q_1\}$	$\{q_1, q_2\}$
q_1	$\emptyset$	$\emptyset$
q_2	$\{q_1\}$	$\{q_3\}$
q_3	$\{q_1\}$	$\emptyset$

Tabla 2.1: Función de transición de un AF.

■

Es posible representar un AF como un grafo dirigido en el que los estados serán los nodos y las transiciones desde un estado p a otro q mediante el símbolo de entrada a se representarán mediante un arco dirigido desde el nodo que representa al estado p hacia el nodo que representa al estado q etiquetada con el símbolo a. Para distinguir los estados finales, éstos se representarán con doble círculo mientras que el estado inicial[1] se marcará con una punta de flecha. La representación gráfica del autómata del ejemplo 2.10 puede verse en la figura 2.1

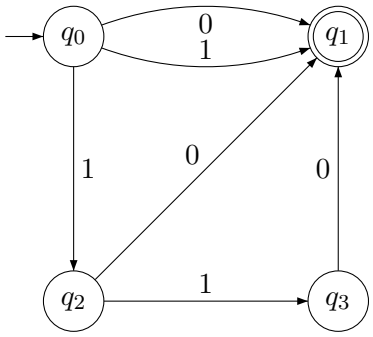

Figura 2.1: Representación gráfica de un AF.

Podemos ver un AF como un dispositivo dotado de un control, capaz de encontrarse en un momento t en un conjunto de estados $\Delta_t \in \mathcal{P}(\mathcal{Q})$, leyendo una palabra w o secuencia

[1] Normalmente será el que esté más a la izquierda, pero no necesariamente

de símbolos $w_{1..n} \in \Sigma^*$ escritos en una cinta, como se muestra en la figura 2.2. En un movimiento el AF que se encuentra en el conjunto de estados Δ_t y leyendo el símbolo w_t en la cinta, pasa al conjunto de estados $\Delta_{t+1} = \{q \mid \exists\, r \in \Delta_t,\ q \in \delta(r, w_t)\}$ y desplaza la cabeza lectora una posición hacia la derecha. Si $\exists\, q \in \Delta_{t+1} \mid q \in \mathcal{Q}_f$, se dice entonces que el autómata acepta o reconoce la cadena escrita en la cinta de entrada hasta la posición $t+1$, excluyendo ésta.

El conjunto de estados que se alcanza mediante la transición etiquetada con el símbolo a desde el conjunto de estados Δ se denota como $\Delta.a = \bigcup_{q \in \Delta} \delta(q,\ a)$. Esta notación se puede ampliar de forma natural por transitividad a palabras, es decir, si w es una palabra de longitud n, entonces $\Delta.w$ denota el conjunto de estados alcanzado desde Δ aplicando de forma sucesiva las transiciones etiquetadas con cada una de las letras de w que será $\Delta.w_1.w_2...w_n$. En el ejemplo 2.10, $\{q_0\}.1$ sería $\{q_1, q_2\}$ ya que desde q_0 existen dos transiciones etiquetadas con el símbolo de entrada 1, una que nos lleva a q_1 y la otra a q_2.

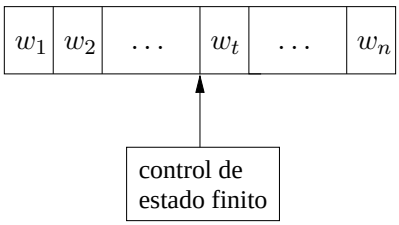

Figura 2.2: Esquema de funcionamiento de un AF.

Un AF es por tanto un dispositivo capaz de reconocer un conjunto finito de palabras y rechazar todas aquéllas que no pertenezcan a ese conjunto. Así, si partiendo del estado inicial del autómata y utilizando de forma sucesiva cada uno de los caracteres de una palabra determinada para transitar a través de sus estados, es posible alcanzar un estado final, entonces podremos decir que la palabra es reconocida, mientras que si no es así diremos que es rechazada.

Definición 2.14. *Decimos que una palabra w es una* palabra aceptada o reconocida por el AF $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *sii* $\exists\, q \in \{q_0\}.w \mid q \in \mathcal{Q}_f$.

□

Ejemplo 2.11. *Para el AF del ejemplo 2.10, tendríamos que $w = 10$ sería una palabra aceptada ya que $\{q_0\}.1 = \{q_1, q_2\}$ y $\{q_1, q_2\}.0 = \{q_1\}$ y por tanto, q_1 que es el estado final del AF pertenece a $\{q_0\}.w$. Sin embargo, la cadena $w = 11$ no sería aceptada ya que $\{q_0\}.w = \{q_3\}$ y q_3 no es un estado final.* ■

Definición 2.15. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un AF, definimos $\mathcal{L}(\mathcal{A})$, el* lenguaje reconocido por $\mathcal{A}$*, como el conjunto de todas las palabras w tales que $q_0.w \in \mathcal{Q}_f$.* □

Ejemplo 2.12. *El lenguaje reconocido por el AF del ejemplo 2.10 estaría formado por las palabras $0, 1, 10$ y 110, ya que son las únicas que nos llevan desde q_0, que es el estado inicial, a q_1, que es el estado final.* ■

Un AF no es más que un conjunto de estados interconectados y transitables por lo que podemos definir intuitivamente el concepto de camino en un AF como una secuencia ordenada de estados que cumplen que desde cada uno de ellos es posible transitar al siguiente.

Definición 2.16. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un* AF*, definimos un* camino *en $\mathcal{A}$ como una secuencia de estados $\{p_1, \ldots, p_n\}$, tal que $\forall i \in \{1, \ldots, n-1\}$, $\exists a_i \in \Sigma, p_i.a_i = p_{i+1}$.* □

Ejemplo 2.13. *Para el autómata del ejemplo 2.10, las secuencias $\{q_0, q_2, q_1\}$ y $\{q_0, q_2, q_3, q_1\}$ serían caminos, mientras que $\{q_0, q_3, q_2\}$ no lo sería. Nótese que el orden es importante.* ■

Durante el proceso de reconocimiento de una palabra sobre un AF recorreremos una serie de estados que comenzará en el estado inicial y, si la palabra pertenece al lenguaje reconocido por el autómata, acabará en un estado final. Al camino recorrido durante el reconocimiento de una palabra w sobre un AF $\mathcal{A}$ lo denominaremos *traza* de w en $\mathcal{A}$.

Definición 2.17. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un autómata y $w = w_1, w_2, \ldots, w_n$ una palabra de $\mathcal{L}(\mathcal{A})$, definimos la* traza *de la palabra w en $\mathcal{A}$ y la denotamos por $\mathcal{T}_{\mathcal{A}}(w)$, o $\mathcal{T}(w)$ cuando el contexto esté suficientemente claro, a la secuencia de estados $\{p_1, \ldots, p_{n+1}\}$ tal que $\forall i \in \{1, \ldots, n\}, p_i.w_i = p_{i+1}$ y $p_1 = q_0$* □

Ejemplo 2.14. *Para el autómata del ejemplo 2.10, la traza de la palabra* `110` *sería*

$$\mathcal{T}(110) = \{q_0, q_2, q_3, q_1\}$$

■

Podemos decir entonces que el proceso de reconocimiento de una palabra consiste en encontrar su traza, de forma que si ésta termina en un estado final, la palabra es reconocida y si no es rechazada. Este proceso puede resultar más o menos sencillo en función de si cabe o no la posibilidad de que en alguno de los estados del AF nos encontremos con que el siguiente carácter de la cadena de entrada nos permite alcanzar más de un estado. En caso de que en algún momento sea posible transitar a más de un estado con el mismo carácter entonces diremos que el AF no es determinista.

Definición 2.18. *Decimos que un* AF *$\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ es un* autómata finito determinista *(*AFD*) sii la función de transición δ es del tipo $\mathcal{Q} \times \Sigma \longrightarrow \mathcal{Q}$, o lo que es lo mismo, $\mid q.a \mid \leq 1, \forall\ q \in \mathcal{Q}, a \in \Sigma$.* □

En realidad, en Hopcroft y Ullman [71] se demuestra que para cualquier *autómata finito no determinista* (AFND) existe un AFD que acepta el mismo lenguaje, pero el concepto de determinismo y no determinismo resulta de especial importancia en la teoría de lenguajes y en la teoría de la computación. El AF de la figura 2.1 no es determinista ya que desde q_0 con un 1 podemos ir a q_1 o a q_2. Un AFD equivalente podría ser el que se representa en la figura 2.3 para el que dados un estado y un símbolo cualesquiera sólo existe una única transición posible.

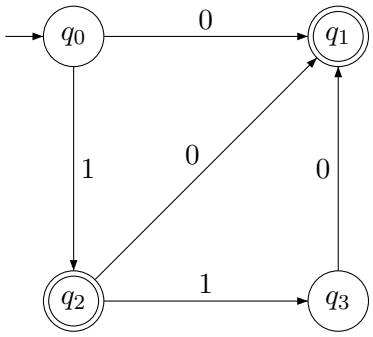

Figura 2.3: AFD equivalente al de la figura 2.1.

Podemos ampliar nuestro modelo de AFND para permitir transiciones etiquetadas con la cadena vacía ε que llamaremos *transiciones nulas*. El funcionamiento no variará demasiado, las transiciones etiquetadas con ε podrán ser utilizadas para cambiar de estado sin desplazar la cabeza lectora en la cadena de entrada de la figura 2.2.

Definición 2.19. *Definimos un* AF con transiciones nulas *(AF-ε) como un* AF *en el que la función de transición δ es del tipo $\mathcal{Q} \times (\Sigma \cup \{\varepsilon\}) \longrightarrow \mathcal{P}(\mathcal{Q})$, siendo $\mathcal{P}(\mathcal{Q})$ el conjunto de las partes de $\mathcal{Q}$* □

En Hopcroft y Ullman [71] se demuestra también que para cualquier AF-ε existe un AFND equivalente, es decir, que acepta exactamente el mismo lenguaje. En la figura 2.4 se muestra un AF-ε que es equivalente al AFND de la figura 2.1.

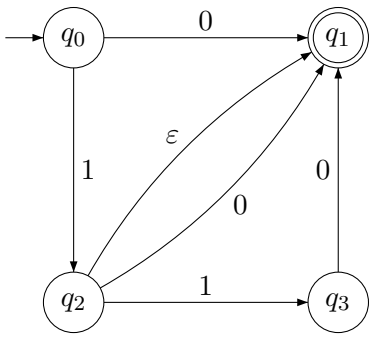

Figura 2.4: Representación gráfica de un AF-ε.

2.3 Autómatas finitos acíclicos deterministas numerados

En la sección anterior hemos tratado los AF de un modo generalista, pero para este trabajo nos interesa su aplicación a la implementación de analizadores léxicos, en concreto para representar diccionarios que en muchos casos alcanzan a contener cientos de miles de palabras.

La primera condición que impondremos al AF es que sea determinista, lo cual no es una restricción real porque sabemos que para cualquier AFND hay un AFD equivalente.

La segunda condición impuesta es que el AF ha de ser acíclico. El cumplimiento de esta condición garantizará que el número de palabras del diccionario será finito y también su longitud, característica que es propia de los lenguajes naturales.

Definición 2.20. *Un* AF *es* acíclico *(*AFA*) cuando su grafo subyacente lo es también.* □

Un diccionario no es más que un conjunto de palabras, por lo que para representarlo y poder luego recorrerlo de forma eficiente podemos pensar en un árbol de letras, que es un AFDA en el que el estado inicial es la raíz del árbol, siendo las hojas los estados finales. En la figura 2.5 se muestra un ejemplo de árbol de letras que representa las palabras del español *chorizo, cohabitante, coherente y cooperase*.

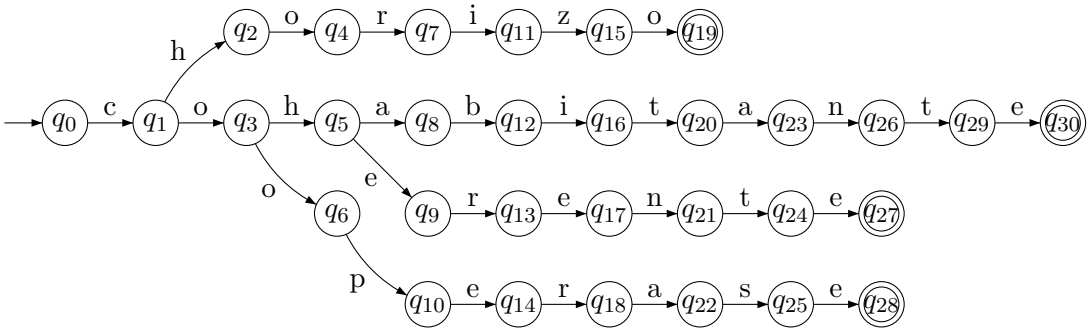

Figura 2.5: Árbol de letras.

El principal problema de este tipo de estructura son los requerimientos de memoria, ya que las posibilidades de compartición de letras entre palabras se reducen únicamente a los prefijos. Es por esto que resulta aconsejable someter esta primera aproximación a un proceso de minimización de AFs [71], que siempre es posible y además garantiza que el AF resultante es equivalente al de partida, o lo que es lo mismo reconocerá el mismo conjunto de palabras.

Por otra parte, ya que como hemos indicado la construcción del árbol de letras puede resultar imposible debido a las restricciones de memoria, se sugiere una construcción incremental del mismo alternando operaciones de inserción de palabras con operaciones

de minimización con el fin de que el AF almacenado en memoria sea en todo momento mínimo, como se describe en Daciuk [35].

Definición 2.21. *Decimos que dos* AFs $\mathcal{A}$ *y* $\mathcal{M}$ *son* equivalentes *si* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{M})$. *Se dice también que dos estados p y q de un autómata dado son* equivalentes *si y sólo si el sub-autómata que comienza con p como estado inicial y el que comienza en q son equivalentes. O lo que es lo mismo, si para toda palabra w tal que p.w es un estado final, entonces q.w también lo es y viceversa.*

□

La existencia de estados equivalentes en un autómata implica redundancia o duplicidad, aspecto poco recomendable a la hora de almacenarlo en memoria.

Definición 2.22. *Por el contrario, decimos que dos estados p y q son* distinguibles*, o* no equivalentes*, si y sólo si existe al menos una palabra tal que p.w es un estado final y q.w no lo es, o viceversa.*

□

Ejemplo 2.15. *En la figura 2.5 podemos observar que los estados* q_{17} *y* q_{23} *son equivalentes ya que los sub-autómatas que parten de ellos también lo son, ambos reconocen el sufijo "*nte*". También son equivalentes los estados* q_{25} *y* q_{29} *cuyos sub-autómatas reconocen el sufijo "*e*" y* q_{19} *y* q_{30} *que reconocen la cadena vacía. Por el contrario, podemos decir que los estados* q_{11} *y* q_{26} *son distinguibles ya que el sub-autómata que parte del primero reconoce el sufijo "*zo*" mientras que el que parte del segundo reconoce el sufijo "*te*".*

■

Los estados equivalentes pueden ser colapsados en uno solo, eliminando los demás junto con el sub-autómata que parte de ellos y cambiando las transiciones que apuntaban a los estados eliminados para que apunten al que queda, consiguiendo así reducir el número de estados y transiciones del autómata.

Definición 2.23. *Dado un autómata* $\mathcal{A}$*, existe un único autómata* $\mathcal{M}$*, mínimo en el número de estados, que cumpla que* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{M})$. *Un* autómata mínimo *es aquél para el que no existe ningún par de estados equivalentes.*

□

Definición 2.24. *Definimos un* AFD acíclico numerado *(*AFDAN*) como aquel autómata finito determinista acíclico que incorpora para cada estado un valor entero, que denominaremos* peso*, y que indicará el número de palabras que se pueden aceptar mediante el sub-autómata que comienza en dicho estado.*

□

En la figura 2.6 se muestra el AFDAN mínimo equivalente al de la figura 2.5. Nótese que los estados que eran equivalentes se han colapsado.

Los AFDANs resultan una excelente herramienta para la modelización de diccionarios ya que son la estructura más compacta que se puede diseñar para el reconocimiento de un conjunto finito de palabras en la que el tiempo de reconocimiento es lineal respecto

a la longitud de la palabra a analizar y no depende ni del tamaño del diccionario ni del autómata. Además, al estar numerado, se puede obtener para cada palabra del diccionario un índice que la identifica unívocamente utilizando para ello una sencilla función presentada por Graña [57], tal y como se explica en la sección 4.1.4.

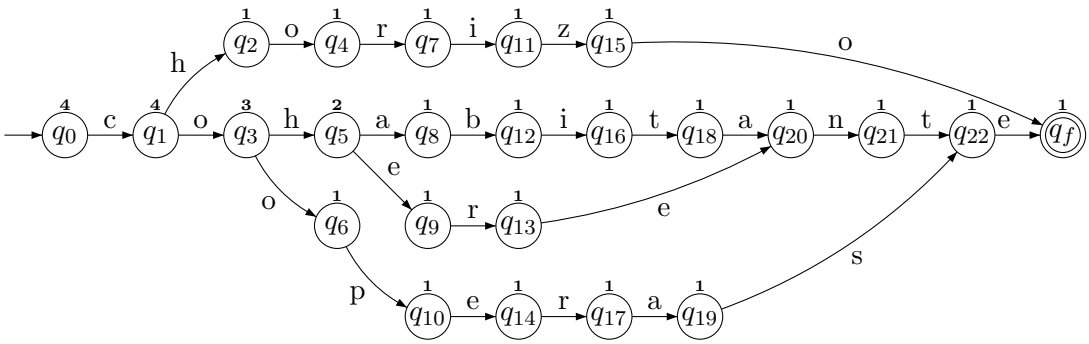

Figura 2.6: AFDAN mínimo equivalente al de la figura 2.5.

Capítulo 3

Introducción a la etiquetación del lenguaje natural

La etiquetación del lenguaje natural es un proceso que consiste en marcar las palabras de un texto, asignando a cada una de ellas una categoría sintáctica basándose tanto en su definición como en su relación con las palabras adyacentes relacionadas en la frase. Una forma simplificada de etiquetación es la que se enseña a los niños en edades tempranas que consiste en identificar las palabras de una frase como nombre, verbo, adjetivo, determinante, etc.

Hoy en día existen algoritmos en el contexto de la lingüística computacional que permiten realizar la etiquetación del lenguaje natural de forma automática y eficiente.

El estado del arte en este campo ha estado íntimamente ligado a la *lingüística de corpus*. El primer *corpus* de un tamaño considerable que aconsejaba su análisis por ordenador fue el Corpus Brown desarrollado por Henry Kucera y Nelson Francis [84], a mediados de los sesenta. Este consistía en una serie de textos en prosa de alrededor de un millón de palabras, extraídos de quinientos ejemplos de publicaciones en inglés elegidos al azar. Su etiquetación se prolongó durante varios años y resultó muy laboriosa. La primera aproximación fue realizada con un programa que etiquetaba el texto basándose en una gran lista de reglas obtenidas de forma manual que indicaban qué combinaciones de etiquetas podían ocurrir y cuáles no. Los resultados obtenidos por este programa alcanzaban un modesto 70 % de acierto y fueron revisados manualmente de forma reiterada de modo que a finales de los setenta la etiquetación del *corpus* era casi perfecta, teniendo en cuenta que en algunos casos ni siquiera los expertos se ponían de acuerdo. Utilizado para innumerables estudios sobre frecuencias de palabras y categorías sintácticas, sirvió para inspirar otros similares en diferentes lenguas, sin embargo, hoy en día ha sido ampliamente superado por *corpora* de dimensiones mucho mayores como el British National Corpus que cuenta con cien millones de palabras.

La etiquetación del lenguaje natural resulta sensiblemente más compleja que manejar un diccionario de palabras con su correspondiente etiqueta, ya que algunos términos pueden pertenecer a diferentes categorías sintácticas dependiendo del papel que juegan

en una frase concreta. Si preguntásemos a alguien acerca de la categoría sintáctica de la palabra *"sobre"*, es muy probable que la respuesta fuese que depende del contexto y como ejemplo ilustrativo podríamos analizar la frase

"Pon el sobre que sobre sobre la mesa"

en la que la palabra *"sobre"* aparece tres veces en un corto intervalo desempeñando tres funciones sintácticas distintas: sustantivo común masculino singular, verbo en tercera persona de singular del presente de subjuntivo y preposición. La elección de la categoría sintáctica correcta en casos como el del ejemplo sólo es posible a partir del contexto de la palabra que presenta la ambigüedad.

En el proceso de etiquetación deben ser resueltas este tipo de ambigüedades determinando cuál de las alternativas resulta ser la que mejor encajaría en el contexto en que ésta aparece. Por otra parte, disponer de un diccionario que contenga todas las palabras de una lengua con todas sus posibles derivaciones resulta muy complicado por lo que se hace necesario poder asignar etiquetas correctas a todas las palabras que aparezcan en una frase, aún cuando éstas no se encuentren en el diccionario.

A pesar de la complejidad que pueda introducir el hecho de que existan ambigüedades que deben ser resueltas y palabras desconocidas para las que no cabe otra técnica que la de la adivinación, la etiquetación es un problema de ámbito limitado. Es por esto que resulta más simple que el análisis sintáctico y mucho más que el semántico ya que en el caso de la etiquetación se establecen solamente las categorías de las palabras dejando de lado la identificación de sintagmas, la relación que entre éstos pueda existir y el significado o conocimiento que de una frase podamos extraer.

Los etiquetadores suelen moverse en unos índices de acierto que oscilan entre el 95 y el 97 %. Cabe aclarar que estos índices de acierto tan cercanos al 100 % pueden resultar engañosos ya que en determinados ámbitos, como el de los artículos periodísticos, el número medio de palabras por frase es de 20 ó 25 por lo que un índice de error de tan sólo el 5 %, supondría una media de una o dos palabras erróneamente etiquetadas en cada frase. Esto puede provocar errores en cascada en etapas posteriores de análisis alterando de forma significativa el significado final de la frase. Ello evidencia el impacto y la importancia del etiquetado en el contexto del desarrollo de herramientas en procesamiento del lenguaje natural.

3.1 Información útil para la etiquetación

Sabemos que a una misma palabra pueden serle asignadas diferentes etiquetas dependiendo del contexto en que ésta aparezca, y que el objetivo final de la etiquetación es que cada palabra tenga una etiqueta que indique la categoría sintáctica a la que pertenece. Estamos pues, identificando un subproblema de la etiquetación que consiste en averiguar cúal de las posibles etiquetas de una palabra es la correcta en un contexto sintáctico determinado. En general podemos decir que existen dos fuentes de información básicas para dar solución al problema:

1. En primer lugar, podemos fijarnos en las etiquetas de las palabras que rodean a aquélla que presenta ambigüedad. Es posible que éstas sean también ambiguas, pero observando las posibles combinaciones podríamos ver qué secuencias de etiquetas son comunes y cuáles no. Por ejemplo, en español resulta muy común la secuencia *artículo-sustantivo-adjetivo*, mientras que otras como *artículo-verbo-adjetivo* o *artículo-preposición-adjetivo* son mucho menos frecuentes o casi imposibles. De este modo, si la palabra "*sobre*" apareciese en el contexto "*el sobre blanco*" optaríamos por etiquetarla como sustantivo.

   Esta técnica, por sí sola, no resulta demasiado exitosa ya que los primeros etiquetadores para el inglés [62], basados en reglas deterministas que hacían uso de este tipo de patrones sintagmáticos, etiquetaban correctamente sólo el 77 % de las palabras. Una de las razones de este rendimiento tan bajo es que en inglés existen procesos productivos, como el que permite a determinados sustantivos que podamos tener en nuestro diccionario transformarse y funcionar como verbos, que hacen que el número de palabras que presentan ambigüedad sea muy elevado. De esta forma, se pierde información restrictiva necesaria para poder realizar la desambiguación.

2. Por otra parte, podemos fijarnos en la frecuencia con que una palabra ambigua suele aparecer para cada una de sus posibles etiquetas. Esta información puede resultar providencial a la hora de seleccionar la alternativa correcta, tal y como puso de manifiesto Charniak [30] al descubrir que un etiquetador que asigne simplemente a cada palabra su etiqueta más probable puede alcanzar un índice de acierto del 90 %. De hecho, el rendimiento de ese etiquetador fue utilizado como referencia en todos los trabajos posteriores.

3.2 Breve historia de los etiquetadores

En síntesis, los primeros métodos desarrollados en relación con el proceso de etiquetación consistían simplemente en realizar la búsqueda de las palabras en un diccionario con el objetivo de obtener su categoría.

3.2.1 Etiquetadores contextuales

Uno de los sistemas pioneros en el uso del contexto sintagmático de las palabras para asignar etiquetas fue el CGC (*Computational Grammatical Coder*) presentado por Klein y Simmons [81] en 1963. Éste se basaba en un pequeño diccionario de no más de 2000 palabras no ambiguas y un conjunto de tablas, laboriosamente construidas, mediante las que trataba de sacar provecho de la bien conocida organización sintáctica del inglés. Las tablas contenían reglas que permitían predecir todas aquellas etiquetas que podían ocurrir en un determinado contexto sintáctico en términos de restricciones conocidas o asumidas. El procedimiento consistía entonces en buscar primero las palabras contenidas en el diccionario asignándoles su etiqueta, que resultaba ser única, lo que a su vez proveía de información contextual para aplicar las reglas sintácticas sobre las palabras restantes e inferir de este modo qué etiquetas podían serles aplicadas.

Este sistema tenía que lidiar con las limitaciones de memoria existentes en los ordenadores de la época que obligaban a almacenar los diccionarios extremadamente grandes en soportes físicos externos, poco eficientes desde el punto de vista computacional, tales como cintas. Como curiosidad, para ilustrar estas limitaciones, cabe destacar de este sistema que era capaz de etiquetar 1250 palabras por minuto en un IBM 7090 cuya memoria principal era de tan solo 32K, de la que únicamente utilizaba la mitad para almacenar un pequeño diccionario de unas 2000 palabras, las reglas gramaticales y el código del propio programa.

3.2.2 Etiquetadores estocásticos

El primer etiquetador estocástico conocido fue desarrollado por Stolz, Tannenbaum y Cartensen [149] y recibió el nombre de WISSYN. Al igual que el CGC, este etiquetador también hacía uso de un diccionario reducido que contenía las palabras más comunes para luego estimar las etiquetas de las demás palabras en el *corpus*. La principal diferencia con CGC es que mientras que éste generaba las etiquetas que podrían encajar en un contexto determinado, WISSYN predecía la más probable. Resulta evidente que las probabilidades son de utilidad para adivinar la etiqueta de una palabra en la medida en que una mayor probabilidad condicional explica un mayor número de apariciones de la etiqueta en ese contexto. Sin embargo en situaciones en que varias etiquetas resulten casi equiprobables el sistema está irremediablemente abocado a incurrir en numerosos errores.

El etiquetador WISSYN se ejecutaba en cuatro fases. En las tres primeras identificaba las palabras con mayor frecuencia de aparición mientras que en la última realizaba la predicción para las restantes. La primera fase consistía en una simple búsqueda en un pequeño diccionario que contenía las palabras más frecuentes con sus respectivas etiquetas. La segunda no se centraba en la palabra en sí misma sino en determinadas características morfológicas que servían para identificar ciertas etiquetas. En estas dos primeras fases se conservaba algún grado de ambigüedad ya que no siempre se asignaba una única etiqueta a las palabras encontradas en estos diccionarios. En la tercera fase se trataba de eliminar las ambigüedades identificadas aplicando algunas reglas estructurales que hacían uso del contexto ya identificado de forma no ambigua. En este punto del proceso aún quedaban un buen número de palabras sin tratar, cuyas etiquetas se adivinaban en la cuarta y última fase que usaba probabilidades condicionadas calculadas empíricamente para predecir las etiquetas a partir del contexto sintáctico de las mismas.

3.2.3 Etiquetadores basados en Modelos de Markov Ocultos

Uno de los primeros etiquetadores basado en MMOs fue creado en la Universidad de Lancaster, como parte del proyecto de etiquetación del *corpus* LOB [51, 99]. El punto central de este etiquetador era el manejo de probabilidades para las secuencias de bigramas de etiquetas, con uso limitado de un contexto de mayor orden, y las probabilidades de asignación de una palabra a sus diferentes etiquetas eran gestionadas mediante factores de descuento diseñados a medida. Los etiquetadores basados en modelos de Markov que utilizan ambos tipos de información, esto es, las probabilidades de las palabras

y las probabilidades de transición entre etiquetas, fueron introducidos por Church y DeRose [33, 40].

A pesar de que los trabajos de Church y DeRose fueron la clave del resurgir de los métodos estadísticos en lingüística computacional, la aplicación de los MMOs al proceso de etiquetación había comenzado en realidad mucho antes en los centros de investigación de IBM en Nueva York y París por parte de Jelinek [77] y Derouault y Merialdo [41]. Otras referencias de los primeros trabajos sobre etiquetación probabilística corresponden a Bahl y Mercer [10], Baker [11] y Foster [47].

3.3 Evaluación de los etiquetadores

En la actualidad, la mayoría de los entornos conocidos se basan en la obtención de un modelo a partir de grandes cantidades de texto etiquetado mediante alguna técnica de entrenamiento. Estos etiquetadores ofrecen un rendimiento que oscila entre el 95 % y el 97 % de acierto global, esto es, calculado sobre el total de las palabras de un texto[1]. En cualquier caso, el rendimiento de este tipo de etiquetadores depende en gran medida de una serie de factores como los que se enumeran a continuación:

- La cantidad de texto de entrenamiento disponible. En general, cuanto mayor sea ésta, mejor será el rendimiento obtenido.

- El *juego de etiquetas*[2]. Normalmente, cuanto mayor sea, mayor será también el número de combinaciones posibles entre las mismas lo que hará más complejo aprender patrones de etiquetas que nos permitan resolver ambigüedades. El impacto del tamaño del juego de etiquetas se puede mitigar con un mayor tamaño y calidad del conjunto de textos de entrenamiento que garantice varias apariciones de cada una de las etiquetas, en cada uno de los contextos en que éstas son susceptibles de aparecer.

- La diferencia entre el ámbito de los textos de entrenamiento y los textos a etiquetar. El léxico y las expresiones que podemos encontrar en textos de distintos ámbitos pueden diferir sensiblemente. No sólo podemos encontrar diferencias entre textos de distinto ámbito sino también entre textos de distintas épocas, o incluso entre textos de distintos autores. Así, si los textos que se pretende etiquetar pertenecen al mismo ámbito que el texto con que se ha realizado el entrenamiento, entonces el rendimiento será mayor ya que el modelo aprendido durante la fase de entrenamiento se ajustará mejor a ellos. Los resultados que los investigadores proporcionan sobre sus etiquetadores suelen provenir de situaciones como ésta, pero resulta evidente que si los textos de aplicación provinieran de una época distinta, una fuente distinta, un género o estilo distinto el rendimiento obtenido sería menor.

[1] Algunos autores ofrecen este dato teniendo en cuenta sólo las palabras ambiguas, en cuyo caso las cifras son sensiblemente inferiores.

[2] Usualmente conocido por el término en inglés *tag set* o *tagset*.

- Las palabras desconocidas. La cobertura del diccionario resulta tener un importante impacto sobre el rendimiento de los etiquetadores. El número de palabras desconocidas es a veces un problema derivado del presentado en el apartado anterior ya que cuando el conjunto de textos de entrenamiento pertenece a distinto ámbito que los textos de aplicación, el número de palabras desconocidas suele ser mayor. Un ejemplo típico en el que a menudo aparecen gran cantidad de palabras desconocidas es cuando se intenta etiquetar textos procedentes de algún dominio técnico, debido a la gran cantidad de palabras exclusivas utilizadas.

Los cambios en los cuatro aspectos anteriores pueden producir un impacto considerable en el rendimiento de los etiquetadores. Si, por ejemplo, el conjunto de entrenamiento es pequeño, el de etiquetas grande y el ámbito del *corpus de entrenamiento* es distinto del ámbito de los textos a etiquetar, o si el número de palabras desconocidas es elevado, es probable que el rendimiento obtenido por cualquier etiquetador esté muy por debajo del rango de acierto global citado anteriormente.

3.4 Aplicaciones de la etiquetación

Podemos considerar la etiquetación una de las tareas de PLN de más bajo nivel por lo que muchas tareas de nivel superior pueden beneficiarse en mayor o menor medida de que los textos de entrada estén correctamente etiquetados. Enumeramos a continuación una serie de aplicaciones en las que la etiquetación puede jugar un papel importante:

- El *análisis sintáctico superficial*[3]: los analizadores superficiales más simples se limitan a buscar estructuras sintácticas simples en una oración mientras que otros más sofisticados asignan a esas estructuras una función gramatical (sujeto, complemento directo, complemento indirecto, complemento circunstancial, etc). Una presentación conjunta del análisis sintáctico superficial y la etiquetación puede verse en [1].

- *Adquisición automática de información léxica*[4]: El objetivo básico de este proceso es desarrollar algoritmos y técnicas estadísticas que permitan rellenar los huecos de información sintáctica y semántica existentes en los diccionarios electrónicos, a través del estudio de patrones que se pueden observar en grandes *corpora* de texto.

- *Extracción de información*[5]: El objetivo fundamental de la extracción de información es encontrar valores para un conjunto predeterminado de ranuras de información de una plantilla, o lo que es lo mismo, rellenar los campos de un formulario predeterminado a partir de documentos que contienen la información necesaria para ello. La etiquetación y el análisis sintáctico superficial facilitan la identificación de las entidades que sirven para rellenar los campos y la relación entre ellas. En cierto

[3] Usualmente conocido por el término en inglés *shallow parsing*.

[4] Usualmente conocido por el término en inglés *lexical acquisition*.

[5] Usualmente conocida por el término en inglés *information extraction*, también referenciado a veces como *message understanding* (entendimiento del mensaje) o *data mining* (minería de datos).

modo, podríamos ver la extracción de información como un proceso de etiquetación en el que las categorías o etiquetas son semánticas en lugar de morfo-sintácticas. Sin embargo, los métodos utilizados en la práctica son bastante diferentes [26].

- *Recuperación de información*[6]: Esta tarea permite obtener un conjunto de documentos que satisfacen potencialmente una necesidad de información que un usuario expresa en forma de consulta. El rendimiento de la recuperación de información puede verse mejorado en ciertos casos si las asociaciones consulta-documento se realizan apoyándose en frases nominales en lugar de en palabras aisladas [45, 145, 150]. En este campo la etiquetación y el análisis sintáctico superficial pueden ayudar a encontrar los términos adecuados de indexación.

- Sistemas de *respuesta a preguntas*[7]: Estos sistemas pretenden responder a preguntas concretas del usuario con frases en lenguaje natural en lugar de devolverle una lista de documentos en los que podría encontrar lo que busca [88, 25]. En este ámbito la etiquetación y el análisis sintáctico superficial puede ayudar a deducir qué tipo de entidad busca el usuario y cómo está relacionada con las demás.

A pesar de la utilidad aparente de la etiquetación, existen más trabajos y referencias sobre ésta de forma aislada que sobre la aplicación de la misma a tareas de interés inmediato. Esto puede ser debido a que los analizadores sintácticos probabilísticos mejor lexicalizados son hoy en día lo suficientemente buenos como para trabajar con texto no etiquetado y realizar la etiquetación internamente en lugar de utilizar un etiquetador como preprocesador [31].

[6] Usualmente conocida por el término en inglés *information retrieval*.
[7] Usualmente conocidos por el término en inglés *question answering systems*.

Capítulo 4

Análisis léxico

Típicamente, uno de los primeros pasos en cualquier aplicación PLN consistirá en transformar el flujo de caracteres de entrada en un flujo de unidades léxicas de más alto nivel[1]. Estas unidades suelen coincidir con las palabras del texto y el proceso de identificación de las mismas y asignación de las etiquetas candidatas a cada una de ellas es lo que denominamos *análisis léxico*[2]. Para realizar esta tarea, resulta imprescindible conocer, en la medida de lo posible, el vocabulario utilizado en el ámbito al que pertenece el texto que nos permitirá reconocer las palabras en él contenidas y acceder de forma rápida a las etiquetas posibles de cada una de ellas. Así, denominaremos *lexicón* a una lista ordenada de unidades léxicas con sus rasgos distintivos que conforman el vocabulario susceptible de ser utilizado en un determinado ámbito.

En este capítulo nos centraremos en la técnica utilizada a lo largo de este trabajo para realizar el análisis léxico, describiendo en profundidad el modelo utilizado para la implementación del lexicón, así como los métodos que permiten un acceso eficiente al mismo [57].

4.1 Modelización de un lexicón

Un lexicón no es más que una lista de palabras de un lenguaje, las cuales pueden tener asociada información. El modelo que de forma automática nos viene a todos a la cabeza para representar tal estructura es el de una tabla en la que las filas se corresponden con las palabras y las columnas con los rasgos conocidos para cada una de ellas. En este sentido, podemos pensar en utilizar una base de datos relacional como modelo para nuestro lexicón.

Las bases de datos relacionales son una potente herramienta que nos facilita la gestión y mantenimiento de tablas de datos ofreciéndonos un amplio abanico de operaciones relacionales muy potentes que nos permiten realizar de forma eficiente operaciones de inserción, actualización, eliminación o consulta; garantizando la integridad del conjunto.

[1] Denominados normalmente por el término en inglés *tokens*.
[2] También denominado por el término en inglés *scanning*.

Sin embargo, no debemos perder de vista una característica que suele presentar cualquier lexicón, que es su tamaño. Baste con mencionar que, a menudo, una herramienta de este tipo para el español puede alcanzar a tener varios cientos de miles de entradas, o incluso millones, lo que supone un importante reto de almacenamiento y recuperación.

Así pues, un analizador léxico necesitará que los accesos al lexicón sean lo más eficientes posible en términos de tiempo. También es importante, dado el número de entradas que puede alcanzar a tener el lexicón, que no se derroche espacio de memoria. Un aspecto menos importante y que podríamos colocar en un segundo plano sería la eficiencia de las inserciones de nuevas palabras, ya que el número de búsquedas a realizar supera con creces al de inserciones. Por último, los procesos menos frecuentes para un analizador léxico serían el borrado y la modificación de entradas en el lexicón, aunque podrían ocurrir. Por tanto, las bases de datos tradicionales no son el mecanismo más adecuado para almacenar un lexicón.

Teniendo en cuenta que partimos de una lista de palabras con sus rasgos y que la inserción de nuevas palabras es poco frecuente, podemos asumir que esta lista estará ordenada *a priori*, lo que no representa un coste significativo ya que esta ordenación se realizará una sola vez. Esta suposición nos permitirá reducir de forma considerable el coste de las búsquedas que es nuestro principal objetivo.

La visión más simple del lexicón que se puede ofrecer al usuario consiste en un fichero de texto en el que en cada línea tendríamos la información de una palabra. En este trabajo utilizaremos cuatro rasgos por palabra que son:

- **Forma**: Es la grafía de la palabra en sí misma.
- **Etiqueta**: Es la etiqueta morfosintáctica que corresponde a la palabra.
- **Lema**: Es la forma canónica de la palabra. Todas las palabras derivadas de una misma tendrán el mismo lema aunque distinta forma. Por ejemplo, el lema de todos los tiempos verbales de un verbo será la forma en infinitivo del mismo.
- **Probabilidad de emisión**: Es la probabilidad de que la etiqueta de la palabra aparezca asociada a ella. Se obtiene a partir de un *corpus de entrenamiento* como se explicará más adelante.

Las palabras ambiguas, es decir, aquéllas que presentan la misma forma, pero distintas etiquetas, aparecerán repetidas en una nueva línea tantas veces como etiquetas distintas presenten. Así, la palabra *sobre* aparecería en el fichero de texto de la siguiente manera:

```
...
sobre   P   sobre   0.113229
sobre   Scms   sobre   0.001263
sobre   Vysps0   sobrar   0.011776
...
```

Este modelo de lexicón puede servir como interfaz con el usuario, para que éste pueda consultarlo y manipularlo a su antojo con la simple ayuda de un editor de texto. Sin

embargo, no resulta operativo si lo que se pretende es mantenerlo en la memoria principal del ordenador para de este modo poder manipularlo y, sobre todo, acceder a la información almacenada para una palabra determinada de forma eficiente.

Necesitamos, por tanto, un modelo que nos permita almacenar las palabras de nuestro lexicón, junto con su información asociada, con la intención de recuperarlas luego de la forma más rápida posible. La cuestión fue abordada por Knuth en [82], donde desarrollaba una serie de técnicas de búsqueda de las cuales analizamos y comparamos a continuación algunas que pueden resultar de interés.

4.1.1 Búsqueda binaria

La búsqueda binaria se basa en almacenar las palabras de nuestro lexicón en una tabla ordenada. El procedimiento a seguir a la hora de realizar una búsqueda consistiría en ir directamente a la entrada que se encuentre en la mitad de la tabla y compararla con la cadena que estamos buscando. Si son iguales, la búsqueda termina. Por el contrario, si la palabra que buscamos es mayor, tendremos que aplicar el proceso de forma recursiva en la mitad derecha de la tabla y si es menor en la mitad izquierda. De esta forma en cada paso se reduce el espacio de búsqueda a la mitad lo que hace que la complejidad, en el peor de los casos, sea $\mathcal{O}(log_2(n))$ siendo n el número de entradas del diccionario.

Este modelo resulta interesante porque las búsquedas son muy eficientes. Sirva como ejemplo que para encontrar una palabra en un diccionario de 500.000 entradas necesitaríamos en el peor de los casos unos 19 pasos. Esta eficiencia se obtiene a costa de penalizar las inserciones y los borrados ya que la tabla debe permanecer ordenada, pero como ya hemos indicado, estas operaciones tienen menor importancia para nosotros.

4.1.2 Búsqueda en un árbol binario

Esta técnica de búsqueda se realiza sobre un árbol binario que debe estar ordenado de forma que para cualquier nodo, todos los que desciendan de él por el hijo izquierdo serán menores y los que lo hagan por el derecho han de ser mayores. La primera palabra que se inserta en el árbol pasa a ser la raíz del mismo y a partir de ésta se van insertando las demás a izquierda o derecha según corresponda. En realidad, podríamos imaginar un árbol binario implícito en el procedimiento de búsqueda binaria del apartado anterior. La diferencia entre ambas técnicas está en la raíz del árbol. En la búsqueda binaria se garantiza que la raíz del árbol implícito estará centrada, es decir, tendrá el mismo número de palabras a la izquierda que a la derecha, como se muestra en la figura 4.1. Sin embargo, en la búsqueda basada en un árbol binario el equilibrio del mismo dependerá del orden de llegada de las palabras pudiendo, en el peor de los casos, degenerar en una lista si las palabras se insertan en un orden determinado. Se puede intuir, por ejemplo, lo que ocurriría con el árbol de la figura 4.1 si las palabras llegasen en orden ascendente, descendente o en el orden siguiente: *cerdo, vaca, conejo, perro, gallina, oveja, gato.*

Esta técnica tendrá una complejidad entre $\mathcal{O}(log_2(n))$, en el mejor de los casos en que el árbol está perfectamente equilibrado y $\mathcal{O}(n)$ en el peor de los casos en que está completamente degenerado. Sin embargo, ha de tenerse en cuenta que los árboles equilibrados son más frecuentes que los degenerados y además existen técnicas para

mantener el árbol equilibrado que penalizarían las inserciones. Dado que, como ya se ha indicado, las inserciones y borrados en nuestro diccionario se consideran ínfimas en comparación con las búsquedas, podemos suponer que nuestro árbol estará perfectamente equilibrado por lo que la búsqueda binaria sería equivalente a la búsqueda en un árbol binario en términos de complejidad.

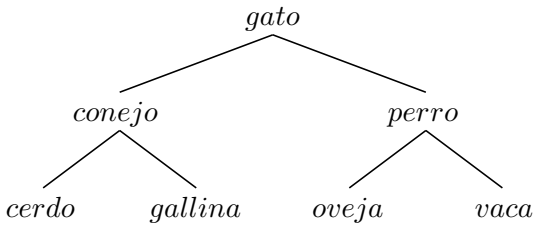

Figura 4.1: Árbol binario equilibrado.

4.1.3 Tablas asociativas

Las tablas *asociativas*[3] constituyen una alternativa a las dos técnicas anteriores ya que no se basan en la comparación del término buscado con una serie de entradas del diccionario. En lugar de esto, a la cadena buscada se le aplicará la llamada *función asociativa*[4] que nos devolverá el índice de la tabla en que podremos encontrarla. Esta función ha de dispersar convenientemente las cadenas a almacenar en el diccionario, produciendo el menor número de colisiones posibles, es decir, dadas dos palabras w_1 y w_2 cualesquiera es deseable, siempre que sea posible que $f(w_1) \neq f(w_2)$. Parece evidente que el factor que determinará la complejidad de esta técnica de búsqueda será la *función asociativa* ya que una vez aplicada a la palabra a buscar, el acceso a la misma será inmediato.

4.1.4 Los AFDAN como función asociativa

En [57], Graña describe la representación más compacta que se puede diseñar para albergar toda la información presente en un lexicón. En la figura 4.2 se muestra un esquema de esta representación. Para ilustrar el funcionamiento de dicha estructura, veamos como se realizaría el acceso a la palabra "*sobre*":

- En primer lugar, por medio de la función PALABRA_A_INDICE, cuyo pseudocódigo puede verse en el algoritmo 1, obtenemos una posición de la tabla de índices.
- En la posición de la tabla de índices obtenida en el punto anterior encontramos una nueva posición, en este caso referente a la tabla de datos. Esta posición nos indica la fila a partir de la cual se encuentran los datos referentes a la palabra en cuestión. En este caso, dado que la palabra "*sobre*" es ambigua y presenta tres etiquetas diferentes, tendremos una fila para cada una de ellas. Para saber el número etiquetas de una

[3] Resulta frecuente el uso del término en inglés *hash table*.
[4] Resulta frecuente el uso del término en inglés *hash function*.

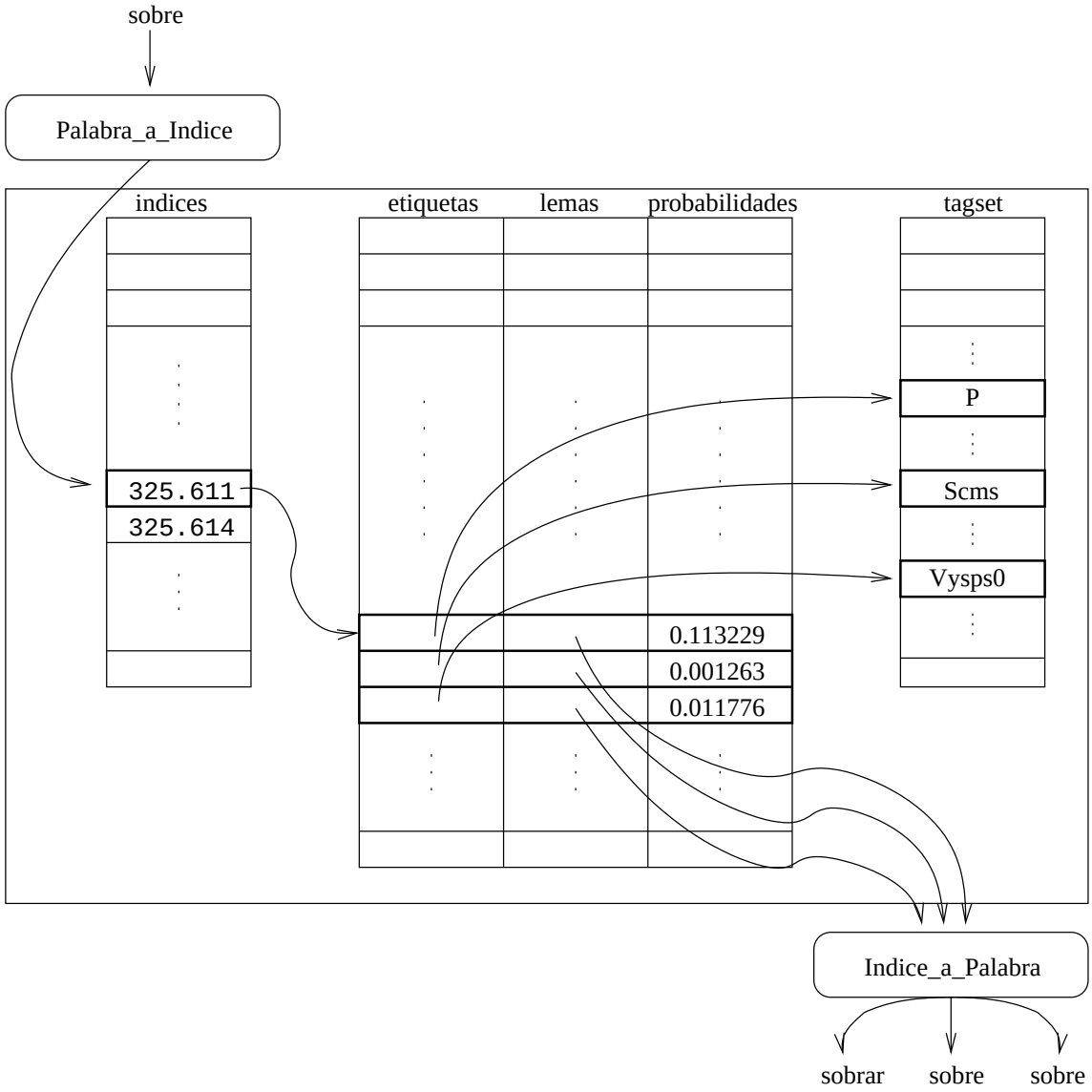

Figura 4.2: Modelización compacta de un diccionario [57].

palabra basta con consultar la siguiente posición de la tabla de índices y restarle la posición actual.

- En cada una de las filas de la tabla de datos encontraremos tres valores numéricos. El primero de ellos nos indica la posición de la tabla de *tagset*[5] en la que se encuentra la etiqueta de la palabra. El segundo nos servirá para obtener el lema de la palabra a través de la función INDICE_A_PALABRA, cuyo pseudocódigo puede verse en el algoritmo 2, y el tercero nos dará la probabilidad de que esa etiqueta se asocie a la palabra.

Algoritmo 1 Función PALABRA_A_INDICE

```
1:  procedimiento PALABRA_A_INDICE(w_{1..n}, A)
2:      Indice ← 1
3:      q ← q_0
4:      para i ← 1 hasta n hacer
5:          si ∃q' | q' = q.w_i entonces
6:              para cada c ∈ Σ, p ∈ Q | c < w_i y q.c = p hacer
7:                  Indice ← Indice + Numero(p)
8:              fin para
9:              q ← q'
10:         sino
11:             retornar 0
12:         fin si
13:     fin para
14:     si q ∈ Q_f entonces
15:         retornar Indice
16:     sino
17:         retornar 0
18:     fin si
19: fin procedimiento
```

Esta representación, propuesta por Graña [57], es en realidad una tabla para la que el autor propone la siguiente función asociativa basada en los AFDAN introducidos en la sección 2.3:

$$f(w) = 1 + \sum_{i=1}^{n} \sum_{a<w_i} peso(q_0.w_{1:i}.a)$$

donde $peso\,(q_i.a)$ es el peso del estado $q_i.a$, y lo que hace la función es recorrer la traza de w y en cada estado q_i de la misma calcular la suma de los pesos correspondientes a los estados accesibles mediante letras lexicográficamente menores[6] que w_i. El valor final

[5] Se utiliza en la figura el término en inglés como identificador por resultar más corto que el término correspondiente en español que sería *conjunto de etiquetas*.

[6] Según el orden definido por la tabla de codificación de caracteres.

de la función se obtiene totalizando las sumas parciales de todos los estados en $\mathcal{T}(w)$ y sumándole 1.

Ejemplo 4.1. *Retomando el autómata de la figura 2.6 tendríamos que* $f(coherente) = 3$. *Veamos cómo se ha realizado el cálculo:*

1. *Calculamos la traza de la palabra,*

$$\mathcal{T}(coherente) = \{q_0, q_1, q_3, q_5, q_9, q_{13}, q_{20}, q_{21}, q_{22}, q_{23}\}$$

2. *Para cada estado de la traza hemos de sumar los pesos asociados a todos aquéllos accesibles con una letra menor que la de la palabra que nos lleva al siguiente estado de la traza. En este caso, los únicos estados por los que pasamos y en los que hay transición para más de una letra son el* q_1, q_3 *y* q_5, *por lo que sólo nos fijaremos en esos tres.*

   - *En* q_1 *tenemos que la letra de la palabra que nos lleva al siguiente estado de la traza es la 'o' y hay una transición desde ese estado etiquetada con una letra menor que ella que es la 'h', y que nos lleva a* q_5*; cuyo peso es* 1. *Si hubiese más transiciones etiquetadas con letras menores, se sumarían los pesos de los estados a los que éstas nos llevasen.*
   - *En* q_3 *la letra de la palabra que nos lleva al siguiente estado de la traza es la 'h' y sólo hay otra transición que parte desde dicho estado, pero en este caso está etiquetada con la 'o', que no es menor que la 'h', por lo que no se tiene en cuenta.*
   - *En* q_5 *la letra de la palabra que nos lleva al siguiente estado de la traza es la 'e', y hay una transición etiquetada con la 'a' que nos lleva a* q_8, *cuyo peso es el* 1.

3. *Finalmente totalizamos todos los sumatorios calculados en cada estado de la traza y le sumamos 1, por lo que tendríamos* $1 + 1 + 1 = 3$.

■

Puede comprobarse fácilmente que con esta función los índices se asignan a las palabras por orden alfabético, además, lo que es más importante, no permite colisiones y dispersa de forma eficiente los datos en la tabla. La complejidad no depende del número de palabras del diccionario, sino que depende únicamente de la longitud de las mismas, lo cual la convierte en la función asociativa más eficiente que conocemos.

La función INDICE_A_PALABRA, nos permitirá representar, y por tanto almacenar, de un modo compacto las palabras del lexicón. Esto puede resultar especialmente útil para aplicaciones PLN que deban ejecutarse en dispositivos en que el espacio almacenamiento resulta escaso, como teléfonos móviles o PDA's.

Algoritmo 2 Función INDICE_A_PALABRA

```
1: procedimiento INDICE_A_PALABRA(indice, A)
2:     q ← q_0
3:     w ← ε
4:     resto ← indice
5:     i ← 1
6:     repetir
7:         a ← Primero(Σ)
8:         Letra_Encontrada ← falso
9:         repetir
10:            si ∃p ∈ Q | q.a = p entonces
11:                si resto > Numero(p) entonces
12:                    resto ← resto − Numero(p)
13:                sino
14:                    w(i) ← a
15:                    i ← i + 1
16:                    q ← p
17:                    si q ∈ Q_f entonces
18:                        resto ← resto − 1
19:                    fin si
20:                    Letra_Encontrada ← verdadero
21:                fin si
22:            fin si
23:        hasta a ≠ Ultimo(Σ) o resto = 0 o Letra_Encontrada
24:    hasta resto = 0
25:    retornar w
26: fin procedimiento
```

Capítulo 5

Modelos de Markov

Los primeros sistemas de etiquetación morfosintáctica se basaban en reglas diseñadas manualmente que trataban de modelar el comportamiento del lenguaje humano. Sin embargo, en la medida en que el número de textos electrónicos disponibles iba creciendo, las aproximaciones estocásticas adquirieron una importancia mayor dado que la necesidad de grandes *corpora* de entrenamiento dejaba de ser un problema. Los etiquetadores estocásticos permiten olvidarse del tedioso trabajo de construcción de reglas ya que los rasgos del modelo se obtienen de forma automática, lo que permite además reparar en rasgos que un humano difícilmente observaría. Sin embargo, en general, los sistemas basados en reglas presentan una ventaja frente a los basados en aprendizaje automático. Los primeros permiten obtener una traza de las reglas aplicadas para la obtención de una solución, lo que puede servir al usuario para depurar el sistema o simplemente obtener una explicación de por qué el sistema ofrece una determinada salida mientras que para los sistemas de aprendizaje automático, la lógica del motor de inferencia se difumina en una serie de valores de probabilidad que hacen imposible ofrecer al usuario una explicación comprensible.

En este capítulo estudiaremos los *Modelos de Markov Ocultos* (MMOs) un método estocástico comúnmente utilizado en sistemas de etiquetación. Se enunciarán los tres problemas canónicos asociados a éstos, explicando en profundidad el algoritmo de Viterbi [167] como solución a uno de ellos. Finalmente, se describirá el problema de la desambiguación segmental y se propondrá, como solución al mismo, una variante del algoritmo de Viterbi sobre retículos [58], en lugar de los enrejados clásicos.

5.1 Procesos de Markov de tiempo discreto

Podemos ver un proceso de Markov como la modelización de un sistema que se encuentra en un determinado estado en un instante concreto y que a intervalos de tiempo discretos puede cambiar de estado, de acuerdo con unas probabilidades de transición. Utilizaremos $\mathcal{Q}$ para denotar el conjunto de estados del sistema. Dado que el conjunto de estados es finito, podemos definir una correspondencia unívoca entre $\mathcal{Q}$ y el conjunto de los números

naturales, de forma que podamos identificar cada estado con un número entre 1 y N, donde $N = |\mathcal{Q}|$. Además utilizaremos $t = 1, 2, \ldots, T$ para denotar el instante de tiempo asociado a cada cambio de estado y q_t para denotar el estado en el que se encuentra el sistema en el instante de tiempo t. En general, una descripción probabilística de un sistema de este tipo requeriría la especificación del estado actual y de todos los estados precedentes. Sin embargo, hay dos características importantes de las cadenas de Markov que nos permiten simplificar el modelo:

- **Propiedad del horizonte limitado**: Permite considerar que el estado actual no depende desde un punto de vista probabilístico de todos los estados previos, sino únicamente de un subconjunto finito de ellos. Así, diremos que una cadena de Markov es de orden n cuando son necesarios n estados previos para poder predecir el siguiente. Por ejemplo, para las cadenas de Markov de tiempo discreto de primer orden, tendríamos:

$$P(q_t = j \mid q_{t-1} = i, q_{t-2} = k, \ldots) = P(q_t = j \mid q_{t-1} = i) \tag{5.1}$$

- **Propiedad del tiempo estacionario**: Permite considerar que la probabilidad de transición entre dos estados i y j no depende del instante de tiempo sino únicamente de los propios estados, esto es, la probabilidad de pasar del estado i al estado j es la misma independientemente del instante de tiempo en que se encuentre el sistema. De este modo podemos definir la función de probabilidad de transición entre estados mediante la matriz $A = \{a_{ij}\}$, como sigue:

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) = P(j \mid i), \quad 1 \leq i, j \leq N \tag{5.2}$$

  que como función de probabilidad cumplirá con la siguiente restricción:

$$\sum_{j=1}^{N} a_{ij} = 1, \quad \forall i$$

  Además, es necesario especificar también la probabilidad que tiene cada uno de los estados de ser el estado inicial. Para esto definimos el vector $\pi = \{\pi_i\}$:

$$\pi_i = P(q_1 = i), \quad \pi_i \geq 0, \quad 1 \leq i \leq N, \quad \sum_{i=1}^{N} \pi_i = 1$$

A un proceso estocástico de estas características se le puede llamar un *modelo de Markov observable* porque su salida es el conjunto de estados por los que pasa en cada instante de tiempo, y cada uno de estos estados se corresponde con un suceso observable.

La probabilidad de observar una secuencia de estados determinada, es decir, la probabilidad de que ocurran una secuencia finita de observaciones, $o_1, o_2, \ldots, O_T$, con todos los $o_i \in \{1, 2, \ldots, N\}$, se obtiene de forma simple mediante el producto de las probabilidades que aparecen en la matriz de probabilidades de transiciones:

$$P(o_1, o_2, \ldots, o_T) = \pi_{o_1} \prod_{t=1}^{T-1} a_{o_t o_{t+1}}$$

Ejemplo 5.1. *Para ilustrar los conceptos relativos a los modelos de Markov, supongamos que nos dedicamos a observar la meteorología de nuestra ciudad durante un determinado intervalo considerando los estados* lluvioso *y* soleado. *Al cabo de un tiempo podremos construir nuestro propio modelo de Markov que nos permita, entre otras cosas, predecir el tiempo que hará al día siguiente observando únicamente el que ha hecho en los días previos. Este modelo podría ser similar al que se muestra en la figura 5.1 que se describe a continuación:*

$$\mathcal{Q} = \{lluvioso,\ soleado\}$$

$$A = \begin{array}{c} \\ lluvioso \\ soleado \end{array} \begin{array}{c} \begin{array}{cc} lluvioso & soleado \end{array} \\ \begin{pmatrix} 0{,}7 & 0{,}4 \\ 0{,}3 & 0{,}6 \end{pmatrix} \end{array}$$

$$\pi = \begin{array}{c} \\ lluvioso \\ soleado \end{array} \begin{pmatrix} 0{,}6 \\ 0{,}4 \end{pmatrix}$$

■

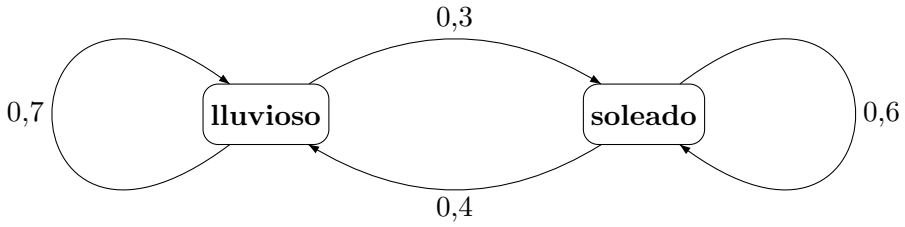

Figura 5.1: Un modelo de Markov para la evolución del clima.

5.2 Modelos de Markov Ocultos

En la sección anterior hemos visto modelos de Markov en los que cada estado se corresponde de manera unívoca con un único suceso observable. Es decir, cuando el sistema se encuentra en un estado su salida no será aleatoria sino que será siempre la misma. Sin embargo, estos modelos tan simples no nos servirán para modelar el problema de la etiquetación, por lo que debemos extenderlos para incluir aquellos sistemas en que la salida en cada estado no sea fija sino una función probabilística del estado. El modelo resultante es doblemente estocástico y se denomina *Modelo de Markov Oculto* (MMO) ya que uno de los procesos aleatorios no es directamente observable, es decir, está oculto.

Para definir formalmente un MMO recurrimos a enumerar los elementos que lo componen.

Definición 5.1. *Definimos un* MMO *como una 5-tupla* $(\mathcal{Q}, V, \pi, A, B)$*, donde:*

1. $\mathcal{Q}$ *es el* conjunto de estados del modelo*. Aunque los estados permanecen ocultos, para la mayoría de las aplicaciones se conocen* a priori*.*

2. V *es el* conjunto de los sucesos observables*. Denotaremos a su vez a cada uno de los sucesos observables como* v_i *de forma que* $V = \{v_1, v_2, \ldots, v_M\}$*, siendo* M *el número de sucesos observables del modelo.*

3. $\pi = \{\pi_i\}$*, es la* distribución de probabilidad del estado inicial*. Por tanto:*

$$\pi_i = P(q_1 = i), \qquad \pi_i \geq 0, \qquad 1 \leq i \leq N, \qquad \sum_{i=1}^{N} \pi_i = 1$$

4. $A = \{a_{ij}\}$ *es la* distribución de probabilidad de las transiciones entre estados*, es decir:*

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) = P(j \mid i), \qquad 1 \leq i, j \leq N, \qquad 1 \leq t \leq T$$

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad \forall i$$

5. $B = \{b_j(v_k)\}$*, es la* distribución de probabilidad de los sucesos observables*, también conocido como* conjunto de probabilidades de emisión*, es decir:*

$$b_j(v_k) = P(o_t = v_k \mid q_t = j) = P(v_k \mid j), \qquad 1 \leq j \leq N, 1 \leq k \leq M, 1 \leq t \leq T$$

$$b_j(v_k) \geq 0$$

$$\sum_{k=1}^{M} b_j(v_k) = 1, \forall j$$

□

Ejemplo 5.2. *En el ejemplo 5.1 planteábamos un problema en el que el estado del tiempo en nuestra ciudad se comportaba como una cadena de Markov discreta con dos estados, "lluvioso" y "soleado". Supongamos ahora que nos mudamos de ciudad a otra muy lejana y que mantenemos contacto con un amigo que nos cuenta lo que ha hecho cada día, pero nunca nos dice nada acerca del tiempo. Sabemos que, dependiendo exclusivamente del estado del tiempo en ese día, existe una determinada probabilidad de que nuestro amigo realice alguna de estas tres actividades: caminar, comprar o limpiar. Dado que nuestro amigo nos cuenta sus actividades del día, esas son las observaciones, mientras que el estado del tiempo serían los estados que permanecen ocultos para nosotros. El sistema completo es un* MMO *como el que se muestra gráficamente en la figura 5.2 para el que* $\mathcal{Q}$*,* π *y* A *coinciden con los del ejemplo 5.1 mientras* V *y* B *se describen a continuación:*

$$V = \{caminar, comprar, limpiar\}$$

$$B = \begin{array}{c} \begin{array}{ccc} caminar & comprar & limpiar \end{array} \\ \begin{pmatrix} 0{,}1 & 0{,}4 & 0{,}5 \\ 0{,}6 & 0{,}3 & 0{,}1 \end{pmatrix} \end{array} \begin{array}{l} lluvioso \\ soleado \end{array}$$

Con un modelo como éste podríamos adivinar el estado del tiempo a partir de una secuencia de acciones de nuestro amigo, es decir, si nuestro amigo nos dijese que el lunes y el martes fue a caminar, el miércoles a comprar y el jueves y el viernes los dedicó a limpiar, podríamos adivinar el tiempo más probable que ha hecho cada día de la semana. ■

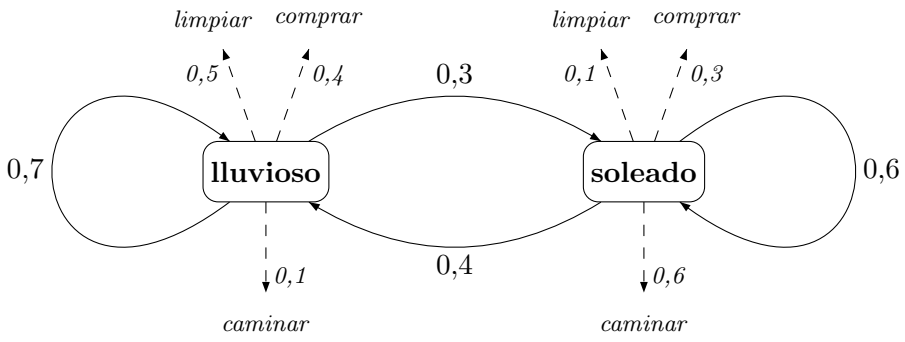

Figura 5.2: Un MMO para la evolución del clima.

Los MMOs representan una herramienta de gran utilidad para muchas aplicaciones reales. Sin embargo, en nuestro caso, queremos modelar el problema de la etiquetación mediante un MMO por lo que debemos centrarnos en identificar cada uno de los componentes del modelo:

1. El conjunto de estados $\mathcal{Q}$ se correspondería con el conjunto de etiquetas.
2. El conjunto de los distintos sucesos observables V serían las palabras del lexicón.
3. La distribución de probabilidad del estado inicial π vendría determinada por la probabilidad que cada etiqueta tenga de ser la primera de una frase, por ejemplo, la probabilidad de que `Det` sea la primera etiqueta de una frase.
4. La distribución de probabilidad de transiciones entre estados determinará la probabilidad de que después de una etiqueta determinada venga otra, por ejemplo, la probabilidad de que después de una etiqueta `Det` venga una etiqueta `S`.
5. La distribución de probabilidad de los sucesos observables indicará la probabilidad de que de una determinada etiqueta se emita una palabra del lexicón, por ejemplo, la probabilidad de que a partir de la etiqueta `V` se emita la palabra "*sobre*".

En lo que respecta a $\mathcal{Q}$ y V, éstas vienen determinadas por el conjunto de etiquetas con el que decidamos trabajar y el lexicón de nuestro lenguaje. No así los tres parámetros restantes del modelo que representan probabilidades y por lo tanto deben ser estimadas.

Existen tres problemas canónicos asociados con los MMOs:

1. ¿Cómo calculamos la probabilidad de una secuencia $O = (o_1, o_2, \ldots, o_T)$ de observaciones dado un modelo $\mu = (\mathcal{Q}, V, \pi, A, B)$?.

   La resolución de este problema nos proporciona la probabilidad de que la secuencia haya sido generada por el modelo. Existen dos algoritmos que resuelven este problema de forma eficiente que se denominan *procedimiento hacia adelante* y el *procedimiento hacia atrás* [57].

2. ¿Cómo elegimos la secuencia de etiquetas $S = (q_1, q_2, \ldots, q_T)$ que mejor explica una secuencia de palabras observadas $O = (o_1, o_2, \ldots, o_T)$ dado el modelo $\mu = (\mathcal{Q}, V, \pi, A, B)$?.

   Resolver esta pregunta supone descubrir la parte oculta del modelo, es decir, adivinar qué camino ha seguido la cadena de Markov. En el problema de la etiquetación, este camino nos ofrece la secuencia de etiquetas más probable para las palabras del texto. Para dar respuesta a esta pregunta se utiliza el algoritmo de Viterbi [167] que explicaremos en la siguiente sección.

3. ¿Cómo estimamos los parámetros π, A y B del modelo $\mu = (\mathcal{Q}, V, \pi, A, B)$?.

   Lo más normal es que no se conozcan *a priori* los parámetros del modelo por lo que deben ser estimados a partir de los datos observados. La secuencia de observaciones utilizada para ajustar el modelo se denomina *secuencia de entrenamiento* y en el caso de la etiquetación consiste en grandes colecciones de texto etiquetado, normalmente de forma manual. Para entrenar el modelo se suele utilizar el *algoritmo de Baum-Welch* [13], aunque existen otros métodos [38].

5.3 El algoritmo de Viterbi

El algoritmo de Viterbi [167] da solución a uno de los tres problemas canónicos asociados a los MMOs. Dado un modelo $\mu = (\mathcal{Q}, V, \pi, A, B)$ se trata de encontrar la secuencia $S = (q_1, q_2, \ldots, q_T)$ más probable a partir de una observación $O = (o_1, o_2, \ldots, o_T)$, es decir, la secuencia de estados óptima que mejor explica la secuencia de observaciones.

Consideremos la variable $\delta_t(i)$ que se define como

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_{t-1}, q_t = i, o_1, o_2, \ldots, o_t \mid \mu)$$

es decir, $\delta_t(i)$ almacena la probabilidad del mejor camino que termina en el estado i, teniendo en cuenta las t primeras observaciones. Se demuestra fácilmente que

$$\delta_{t+1}(j) = \left[\max_{1\leq i\leq N} \delta_t(i)a_{ij}\right] b_j(o_{t+1}) \tag{5.3}$$

Puesto que el objetivo es obtener la secuencia de estados más probable, a la vez que calculamos todos los $\delta_t(i)$ para todos los estados y todos los instantes de tiempo, debemos almacenar una traza que nos permita recordar cuál fue el argumento que maximizó la ecuación 5.3. Para esto utilizaremos la variable $\psi_t(j)$. A continuación se describen los 4 pasos del algoritmo de Viterbi [167]:

1. Inicialización:
$$\delta_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$$

2. Recurrencia:
$$\delta_{t+1}(j) = \left[\max_{1\leq i\leq N} \delta_t(i)a_{ij}\right] b_j(o_{t+1}), \quad t = 1, 2, \ldots, T-1, \quad 1 \leq j \leq N \tag{5.4}$$
$$\psi_{t+1}(j) = \arg\max_{1\leq i\leq N} \delta_t(i)a_{ij}, \quad t = 1, 2, \ldots, T-1, \quad 1 \leq j \leq N$$

3. Terminación:
$$q_T^* = \arg\max_{1\leq i\leq N} \delta_T(i)$$

4. Construcción hacia atrás de la secuencia de estados:
$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \ldots, 1$$

donde La complejidad de este algoritmo es de $\mathcal{O}(N^2T)$.

Durante los cálculos del algoritmo de Viterbi [167], en el cálculo de $\max_{1\leq i\leq N} \delta_t(i)a_{ij}$, se pueden producir empates. Si esto ocurre, se elige un camino de entre los empatados de forma aleatoria. Por otra parte, es posible considerar una implementación del algoritmo de Viterbi [167] que devuelva las n mejores secuencias, ya que existen aplicaciones prácticas para las que puede que no sólo interese la mejor.

Ejemplo 5.3. *Retomemos el ejemplo 5.2 en el que disponíamos de un* MMO *que nos permitía adivinar el estado del tiempo en la ciudad de un amigo tan solo con que éste nos revelase lo que había hecho durante la semana. Recordamos que los parámetros de nuestro modelo*[1] *eran:*

$$\mathcal{Q} = \{llu, sol\} \quad V = \{cam, com, lim\}$$

$$\pi = \begin{pmatrix} 0{,}6 \\ 0{,}4 \end{pmatrix} \begin{matrix} llu \\ sol \end{matrix} \qquad A = \begin{matrix} \\ llu \\ sol \end{matrix} \begin{matrix} \begin{matrix} llu & sol \end{matrix} \\ \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}4 & 0{,}6 \end{pmatrix} \end{matrix}$$

[1] Codificamos los estados y las observaciones con sus tres primeras letras para mejorar la legibilidad.

$$B = \begin{array}{c} \begin{array}{ccc} cam & com & lim \end{array} \\ \begin{pmatrix} 0{,}1 & 0{,}4 & 0{,}5 \\ 0{,}6 & 0{,}3 & 0{,}1 \end{pmatrix} \end{array} \begin{array}{c} llu \\ sol \end{array}$$

Supongamos que nuestro amigo nos ha revelado las siguientes actividades:

Lunes	*Martes*	*Miércoles*	*Jueves*	*Viernes*
caminar	*comprar*	*comprar*	*limpiar*	*caminar*

Para saber qué tiempo ha hecho cada uno de los días de la semana aplicamos el algoritmo de Viterbi [167] sobre la secuencia de observaciones

$$O = (cam, com, com, lim, cam)$$

de longitud $T = 5$ *realizando los siguientes cálculos:*

$$\delta_1(llu) = \pi_{llu} b_{llu}(cam) = 0{,}06$$
$$\delta_1(sol) = \pi_{sol} b_{sol}(cam) = 0{,}24$$

$$\delta_2(llu) = max[\delta_1(llu)a_{llu,llu}, \underline{\delta_1(sol)a_{sol,llu}}]b_{llu}(com) = 0{,}0384 \quad \psi_2(llu) = sol$$
$$\delta_2(sol) = max[\delta_1(llu)a_{llu,sol}, \underline{\delta_1(sol)a_{sol,sol}}]b_{sol}(com) = 0{,}0432 \quad \psi_2(sol) = sol$$

$$\delta_3(llu) = max[\underline{\delta_2(llu)a_{llu,llu}}, \delta_2(sol)a_{sol,llu}]b_{llu}(com) = 0{,}0107520 \quad \psi_3(llu) = llu$$
$$\delta_3(sol) = max[\delta_2(llu)a_{llu,sol}, \underline{\delta_2(sol)a_{sol,sol}}]b_{sol}(com) = 0{,}0077760 \quad \psi_3(sol) = sol$$

$$\delta_4(llu) = max[\underline{\delta_3(llu)a_{llu,llu}}, \delta_3(sol)a_{sol,llu}]b_{llu}(lim) = 0{,}0037632 \quad \psi_4(llu) = llu$$
$$\delta_4(sol) = max[\delta_3(llu)a_{llu,sol}, \underline{\delta_3(sol)a_{sol,sol}}]b_{sol}(lim) = 0{,}0004665 \quad \psi_4(sol) = sol$$

$$\delta_5(llu) = max[\underline{\delta_4(llu)a_{llu,llu}}, \delta_4(sol)a_{sol,llu}]b_{llu}(cam) = 0{,}0010536 \quad \psi_5(llu) = llu$$
$$\delta_5(sol) = max[\delta_4(llu)a_{llu,sol}, \underline{\delta_4(sol)a_{sol,sol}}]b_{sol}(cam) = 0{,}0000839 \quad \psi_5(sol) = sol$$

En cada paso aparecen subrayados los términos máximos. El estado del que proviene ese término máximo es el que se asigna a cada $\psi_t(j)$*. La probabilidad máxima para la secuencia completa de observaciones se alcanza en* $\delta_5(llu)$*, lo que implica que* $q_5^* = llu$*, y al reconstruir hacia atrás la secuencia de estados obtenemos*

$$S = (sol, llu, llu, llu, llu)$$

tal y como se puede observar también en el enrejado de la figura 5.3. En dicho enrejado, se han representado con líneas discontinuas todos los caminos posibles. Con línea continua se han marcado las flechas que unen cada estado j *con el estado del instante anterior indicado por* $\psi_t(j)$*. Por ejemplo, el estado* sol *del instante de tiempo 2 aparece unido con una flecha continua al estado* sol *del instante de tiempo 3, ya que* $\psi_3(sol) = sol$*. Intuitivamente, esto quiere decir que aún no sabemos si el camino óptimo pasará por el estado* sol *del instante*

de tiempo 3, pero en caso de que pase lo hará también por el estado sol *del instante de tiempo 2. Cuando vemos que la probabilidad máxima en el instante 5 corresponde al estado* llu*, es fácil seguir las flechas con línea continua en sentido inverso para obtener la secuencia de estados más probable. En el enrejado, se indican con una línea continua más gruesa las flechas que forman parte del camino más probable.*

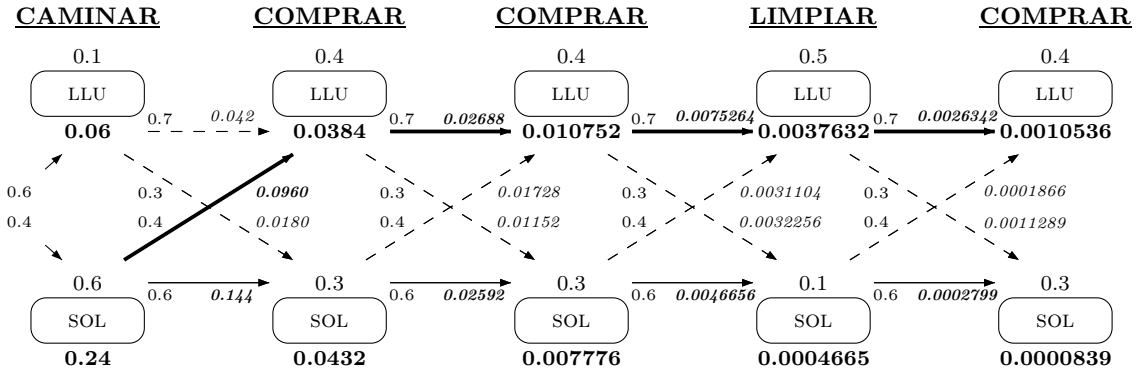

Figura 5.3: Algoritmo de Viterbi [167] sobre un enrejado.

Respecto a los valores que aparecen en el enrejado de la figura 5.3, en la parte superior de cada estado, se muestra la probabilidad de que desde él se emita la observación correspondiente a ese instante de tiempo, mientras que en la parte inferior, se muestra en negrita el valor de δ. Por su parte, en cada flecha se muestran dos valores. Al inicio de la misma, se muestra la probabilidad de transición entre los estados que une, mientras que al final, con distinta tipografía, se muestra el resultado de multiplicar el valor de δ en el estado origen por la probabilidad de transición hacia el estado destino. De este modo, calculamos el valor de δ en cada estado, multiplicando la probabilidad de que desde ese estado se emita la observación correspondiente a ese instante de tiempo por el valor máximo de entre los que aparecen al final de las flechas que llegan a ese estado, y que en la figura se muestra en negrita.

De este modo hemos podido adivinar que el lunes estuvo soleado y desde el martes al viernes no ha dejado de llover.

■

Una vez comprendido el funcionamiento del algoritmo de Viterbi [167] para el caso general, su aplicación al problema de la etiquetación resulta simple. Se trataría de realizar los cálculos sobre enrejados simplificados como el de la figura 5.4, en el que en cada posición no se considera el conjunto de etiquetas completo, sino solamente aquéllas que aparezcan en el lexicón para cada palabra concreta.

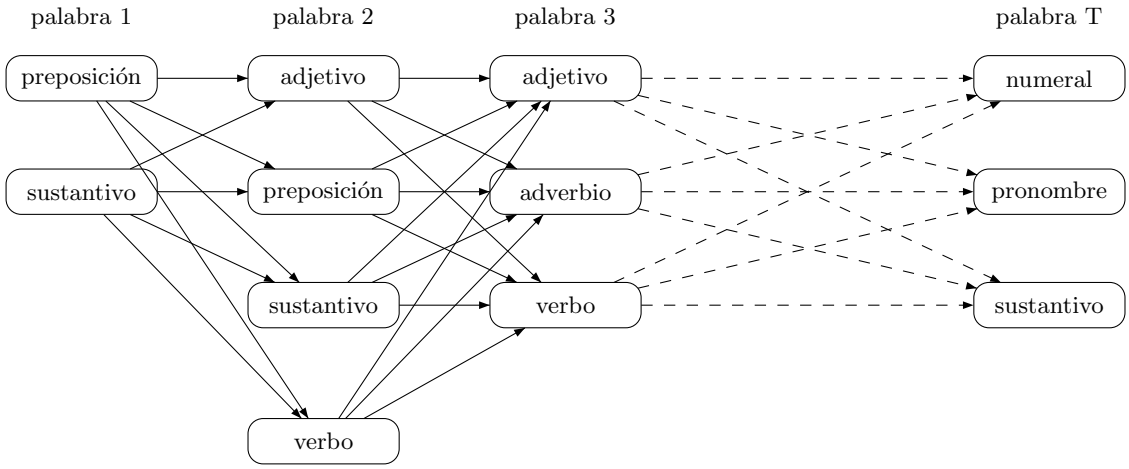

Figura 5.4: Enrejado simplificado para la etiquetación de una frase de T palabras.

5.4 Eliminación de ambigüedades de segmentación

Generalmente los etiquetadores asumen que el texto de entrada se presenta correctamente segmentado en unidades léxicas que a menudo coincidirán con las palabras y que identifican cada componente individual del texto. Sin embargo, esta hipótesis de trabajo no es realista debido a la naturaleza heterogénea de los textos y a las diversas fuentes de las que éste puede proceder. El mayor problema aparece aquí cuando un texto puede ser segmentado de varias formas distintas. En esa situación necesitaremos información contextual que nos permita decidir cuál de las posibles segmentaciones es la más probable.

En esta sección, introducimos un etiquetador que además de asignar una etiqueta a cada *unidad léxica*, será capaz de decidir cuándo algunas de estas unidades deben unirse formando un mismo término o incluso cuándo deben ser separadas en términos distintos. Para ello, utilizaremos una extensión del algoritmo de Viterbi [167] capaz de evaluar secuencias de unidades léxicas de diferentes longitudes sobre una misma estructura [58]. También compararemos su complejidad en tiempo y espacio con la versión clásica del algoritmo.

5.4.1 Segmentación ambigua

A la hora de diseñar un etiquetador morfosintáctico, una de las primeras preguntas que debemos plantearnos hará referencia al formato en que éste recibirá el texto de entrada. La mayoría de los etiquetadores suponen que el texto ha sido sometido a una fase de preprocesado que se ha encargado de la tarea de segmentación en frases y de cada una de esas frases en unidades léxicas. Por ejemplo, para texto escrito, la tarea de preprocesado más simple que podemos realizar consistiría en considerar el carácter de salto de línea como separador de frases y el espacio como separador de palabras, añadiendo una serie de reglas para identificar los signos de puntuación, las fechas o, en general, cualquier otra unidad léxica susceptible de un tratamiento especial. Sin embargo, para lenguas con una

rica morfología, como el español o el gallego [109], existen una serie de casos en que la tarea de segmentación resulta sensiblemente más compleja. Por una parte, es bastante común el uso de contracciones y formas verbales con pronombres enclíticos, lo que provoca que una misma palabra contenga información de dos o más componentes lingüísticos que deben ser separados en unidades léxicas individuales. Además, existen locuciones para las que varias palabras actúan juntas debiendo ser consideradas como una unidad léxica compuesta.

En este contexto, las ambigüedades de segmentación aparecen cuando una o varias palabras pueden ser segmentadas en unidades léxicas de, al menos, dos formas distintas. En la figura 5.5 se muestran dos ejemplos que ilustran los dos tipos de ambigüedades que nos podemos encontrar. Para el caso del gallego, la palabra *polo* puede ser segmentada y etiquetada como sustantivo (*pollo*), como contracción de la preposición '*por*' y el determinante '*o*' (*por el*) o como la forma verbal '*pos*' y el pronombre enclítico '*o*' (*pones el*). Por su parte, la expresión en español '*sin embargo*' puede ser considerada como una unidad léxica compuesta en cuyo caso se etiquetaría como conjunción o bien como dos unidades léxicas que se etiquetarían como preposición y nombre.

$$\text{polo} \left\{ \begin{array}{ll} \text{polo} & \text{N} \\ \left\{ \begin{array}{ll} \text{por} & \text{P} \\ \text{o} & \text{Det} \end{array} \right. & \\ \left\{ \begin{array}{ll} \text{pos} & \text{V} \\ \text{o} & \text{Pro} \end{array} \right. & \end{array} \right. \qquad \text{sin embargo} \left\{ \begin{array}{ll} \text{sin embargo} & \text{C} \\ \left\{ \begin{array}{ll} \text{sin} & \text{P} \\ \text{embargo} & \text{N} \end{array} \right. & \end{array} \right.$$

Figura 5.5: Ejemplo de alternativas ambiguas de segmentación.

Para manejar estas ambigüedades, se han venido utilizando por parte de diferentes autores [110, 166] etiquetas artificiales que son asignadas a unidades léxicas compuestas o etiquetas compuestas que son asignadas a palabras que aglutinan más de una unidad sintáctica. De este modo, se elude el tratamiento de las ambigüedades de segmentación simplemente dejando su resolución para fases del PLN posteriores.

En este sentido, podemos considerar dos opciones para tratar las ambigüedades de segmentación:

1. Extender el conjunto de etiquetas con el fin de representar fenómenos relevantes como contracciones o pronombres enclíticos. Por ejemplo, el verbo *reconocerse* podría ser etiquetado como `V+Pro`[2] haciendo notar de este modo que se trata de una palabra formada por dos unidades léxicas. Por otra parte, en lo que respecta a las locuciones podría asignársele la etiqueta correspondiente a la función léxica que juegan en cada caso indicando por medio de subíndices que se trata de un componente de una *unidad léxica compuesta*. De este modo la locución "*a pesar de*" podría ser etiquetada como

[2]Siendo `V` la etiqueta de verbo y `Pro` la etiqueta de pronombre.

P13, P23 y P33 indicando que se trata de una *unidad léxica compuesta* de tres elementos que juega el papel de preposición.

Esta aproximación resulta más simple ya que evita las decisiones de segmentación dejándolas para fases posteriores en las que, presumiblemente, dispondremos de más información que nos facilitará la elección de la alternativa de segmentación adecuada. Sin embargo, la aplicación de este método al caso del gallego provocaría un aumento excesivo del tamaño del conjunto de etiquetas debido a la rica morfología de esta lengua, lo que a su vez provocaría que el proceso de construcción de textos de referencia que contuviesen todas esas etiquetas en un número significativamente representativo, se convirtiese en una tarea extremadamente compleja.

2. No extender el conjunto de etiquetas e identificar las *unidades léxicas compuestas*, así como aquéllas que han de ser divididas en dos. Esta solución tiene la ventaja de que la complejidad del proceso de etiquetación no se ve afectada por un elevado número de etiquetas. Como contrapartida la complejidad de la tarea de segmentación es mayor ya que se deberá decidir cuándo una palabra es susceptible de ser dividida en varias unidades léxicas o cuándo varias palabras han de ser consideradas como una sola. En este sentido, el preprocesador debería realizar únicamente la detección y preetiquetado de las diferentes alternativas de segmentación. Cuando existan varias alternativas de segmentación será necesario elegir la más adecuada según el contexto en que éstas aparezcan.

Centrándonos en la opción de no aumentar el conjunto de etiquetas, Graña, Alonso y Vilares [58] proponen una extensión del algoritmo de Viterbi [167] capaz de evaluar secuencias de unidades léxicas de diferentes longitudes sobre una misma estructura.

5.4.2 Viterbi-L: el algoritmo de Viterbi sobre retículos

En el ámbito de la etiquetación morfosintáctica con MMOs, la versión clásica del algoritmo de Viterbi [167] se aplica sobre enrejados en los que la primera fila contiene las palabras a ser etiquetadas y las posibles etiquetas aparecen en columnas bajo ellas. Sin embargo, esta estructura resulta demasiado rígida para representar ambigüedades de segmentación. Consideremos, por ejemplo, una frase en la que aparezca la expresión "*sin embargo*". Tal y como se muestra en la figura 5.6, los problemas surgen cuando intentamos colocar la etiqueta `C` debido a que, en ambos casos, la etiqueta debería ser aplicada a la expresión completa, y por lo tanto los caminos marcados con líneas discontinuas no deberían ser permitidos mientras que los marcados con líneas punteadas sí deberían serlo.

Se hace pues necesaria una nueva estructura que nos permita representar de forma cómoda e intuitiva las ambigüedades de segmentación. El problema de los enrejados radica en que los nodos representan los componentes léxicos de la frase por lo que una variación en la longitud de una de estas unidades provocaría una variación en los nodos del enrejado. La utilización de *retículos*[3] en los que las etiquetas se colocan en los arcos que conforman los caminos y tienen su origen y destino en los huecos entre componentes léxicos permiten

[3] Usualmente denominado por el término en inglés *lattice*.

Figura 5.6: Los enrejados no permiten representar segmentaciones ambiguas.

representar de forma simple unidades léxicas compuestas. En la figura 5.7 se muestra el retículo correspondiente a la frase "*El sin embargo fue a pesar de nuevo la fruta*" que resulta muy ilustrativa por la cantidad de ambigüedades de segmentación que presenta y en la que se puede observar cómo se representan de forma homogénea y sobre una misma estructura tanto las ambigüedades de etiquetación como las de segmentación. Combinando sus 20 arcos, se obtienen 234 rutas posibles de las que la correcta es la que se muestra en la parte superior.

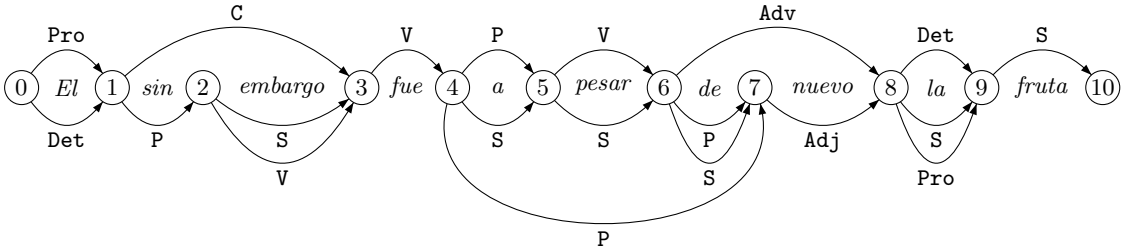

Figura 5.7: Segmentaciones ambiguas representadas en un retículo.

Para poder aplicar el algoritmo de Viterbi sobre retículos necesitamos realizar una ligera adaptación [20]. En lugar de palabras, se enumeran los huecos entre las mismas, como se muestra en la figura 5.7, de modo que un arco que una dos huecos pueda abarcar una o más palabras, de tal forma que un arco se representa mediante una tripla $(t, t\prime, q)$, denotando que empieza en el instante t, finaliza en el instante $t\prime$ y representa el estado q. Para poder realizar los cálculos, introducimos acumuladores $\Delta_{t,t\prime}(q)$ que recogerán la máxima probabilidad del estado q abarcando las palabras desde la posición t a la posición $t\prime$. Usaremos $\delta_{i,j}(q)$ para denotar la probabilidad de que desde el estado q se emitan los terminales abarcados entre la posición i y j:

- Inicialización:

$$\Delta_{0,t}(q) = P(q \mid q_s)\delta_{0,t}(q) \tag{5.5}$$

- Recursión:

$$\Delta_{t,t\prime}(q) = \max_{(t\prime\prime,t,q\prime)\in Reticulo} \Delta_{t\prime\prime,t}(q\prime)P(q \mid q\prime)\delta_{t,t\prime}(q) \text{ para } 1 \leq t < T \tag{5.6}$$

- Terminación:

$$\max_{Q\in\mathcal{Q}^*} P(Q, Reticulo) = \max_{(t,T,q)\in Reticulo} \Delta_{t,T}(q)P(q_e \mid q) \tag{5.7}$$

donde q_s y q_e son el estado inicial y el final, respectivamente. Además, es necesario mantener la traza de los elementos en el enrejado que maximizaron cada $\Delta_{t,t\prime}(q)$. Cuando alcanzamos el instante T, obtenemos el mejor elemento de entre los últimos en el retículo.

$$(t_1^m, T, q_1^m) = \arg\max_{(t,T,q)\in Reticulo} \Delta_{t,T}(q)P(q_e \mid q) \tag{5.8}$$

Estableciendo $t_0^m = T$, recuperamos los argumentos $(t\prime\prime, t, q\prime) \in Reticulo$ que maximizan la ecuación 5.6 retrocediendo en el tiempo:

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \arg\max_{(t\prime\prime,t_i^m,q\prime)\in Reticulo} \Delta_{t\prime\prime,t_i^m}(q\prime)P(q_i^m \mid q\prime)\delta_{t_i^m,t_{i-1}^m}(q_i^m) \tag{5.9}$$

para $i \geq 1$, hasta que alcancemos $t_k^m = 0$. De este modo, $q_1^m, \ldots q_k^m$ es la mejor secuencia de hipótesis de la frase (en orden inverso). A este proceso lo llamaremos el *algoritmo Viterbi-L* [58].

Complejidad del algoritmo Viterbi-L

Intuitivamente, podemos considerar la complejidad espacial del algoritmo Viterbi-L [58] como el número de acumuladores de probabilidad que necesitaremos almacenar durante su ejecución. En esta versión, tendremos un acumulador por cada arco. En cuanto a la complejidad temporal, consideraremos el número de operaciones a realizar. Esto es, para un arco dado, el número de arcos que alcanzan su punto de origen. Por ejemplo, para ejecutar el algoritmo Viterbi-L sobre el retículo de la figura 5.7, necesitaremos 20 acumuladores y 36 operaciones.

Sin embargo, esta versión del algoritmo presenta una anomalía que consiste en que las rutas del retículo con menor longitud tendrán prioridad sobre las más largas ya que cada componente extra en la ruta implica una multiplicación más por una probabilidad que necesariamente será menor que 1, lo que provocará una probabilidad acumulada menor. Esto es un problema ya que las rutas cortas no siempre serán las más adecuadas.

Para evitar este problema, podríamos considerar la evaluación individual de retículos con caminos de la misma longitud, comparándolos posteriormente. En este contexto, sería necesario definir también un criterio para esta comparación. Dado que el paradigma de etiquetación utilizado en este caso se basa en MMOs, la normalización de las probabilidades acumuladas resulta un criterio consistente.

Llamemos p_i a la probabilidad acumulada de la mejor ruta del retículo con rutas de i unidades léxicas de longitud. En la figura 5.7, tendríamos p_7, p_8, p_9 y p_{10} que no serían

directamente comparables, pero si usamos probabilidades logarítmicas[4], podemos obtener valores normalizados simplemente dividiendo cada probabilidad por el número de unidades léxicas. En este caso, $p_7/7, p_8/8, p_9/9$ y $p_{10}/10$ ya serían comparables, lo que nos permitiría seleccionar la mejor ruta de cualquiera de los retículos como la interpretación más probable. Uno de los motivos de la utilización de MMOs es que en otros paradigmas de etiquetación el criterio de comparación no sería tan fácil de identificar.

Sin embargo, el número de retículos diferentes a evaluar no siempre coincidirá con el número de longitudes diferentes de caminos. Por ejemplo, en la figura 5.8, podemos ver cómo al tratar de aislar dos caminos de longitud ocho[5] separamos también, sin pretenderlo, un camino de longitud siete, que se destaca con arcos punteados. Por tanto, podríamos necesitar más de un retículo para representar los caminos de la misma longitud sin conflictos. Para el caso de la figura 5.7, necesitaríamos un total de 6 retículos aún cuando sólo existirían cuatro longitudes de camino diferentes (7, 8, 9 y 10 unidades léxicas). Estos seis retículos vendrían de la exclusión mutua de las diferentes alternativas de cada segmentación ambigua y pueden verse en la figura 5.9. A la vista de este conjunto de retículos, podemos deducir que la complejidad espacial y temporal real del algoritmo sería de 82 acumuladores y 119 operaciones. Sin embargo, al eliminar los conflictos provocados por la segmentación ambigua, los retículos resultantes podrían ser perfectamente representados como enrejados, por lo que no sería necesario modificar el algoritmo de Viterbi clásico [167].

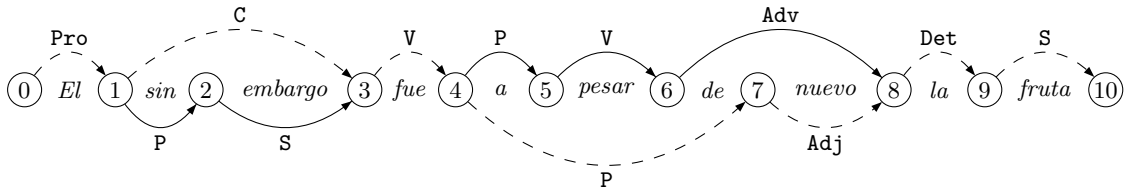

Figura 5.8: Un retículo con caminos conflictivos.

5.4.3 Viterbi-N: el algoritmo de Viterbi con normalización

Para poder sacar ventaja del uso de retículos es necesario un método que permita obtener los valores para cada una de las rutas del retículo de la figura 5.7 en una sola pasada, es necesario almacenar más de un acumulador por arco. Exactamente, se necesitan tantos acumuladores como longitudes diferentes tengan los caminos que llegan al punto de origen del arco. Es necesario, por tanto, incorporar esta información relativa a las longitudes en una tercera componente del índice de cada acumulador: $\Delta_{t,t\prime,l}(q)$. De este modo, en las operaciones de maximización para obtener el correspondiente $\Delta_{t\prime,t\prime\prime,l+1}(q\prime)$ sólo son comparados los acumuladores con la misma longitud l. Al alcanzar el instante final, habremos obtenido tantos acumuladores como diferentes longitudes de camino existan,

[4] Al utilizar probabilidades logarítmicas, los productos se convierten en sumas por lo que para normalizar basta con dividir por el número de sumandos.

[5] El que usa sólo los arcos superiores del retículo y aquél que usa los arcos inferiores cuando es posible.

lo que nos permitirá normalizar los mejores caminos correspondientes a cada acumulador de acuerdo con su longitud para de este modo poder elegir la interpretación más probable. Este proceso recibe el nombre de *algoritmo Viterbi-N* [58].

El uso de probabilidades logarítmicas nos permite realizar los cálculos de forma más eficiente al sustituir productos por sumas y, además, se evitan los problemas de precisión que se producen al multiplicar varias veces por un factor menor que 1. Para utilizar probabilidades logarítmicas y acumuladores por cada longitud, es necesario reemplazar los productos por sumas y adaptar las ecuaciones como sigue:

- Inicialización:

$$\Delta_{0,t,1}(q) = P(q \mid q_s)\delta_{0,t}(q) \tag{5.10}$$

- Recursión:

$$\Delta_{t,t\prime,l}(q) = \underset{(t\prime\prime,t,q\prime)\in Reticulo}{max} \Delta_{t\prime\prime,t,l-1}(q\prime)P(q \mid q\prime)\delta_{t,t\prime}(q) \text{ para } 1 \leq t < T \tag{5.11}$$

- Terminación:

$$\underset{Q\in\mathcal{Q}^*}{max} P(Q, Reticulo) = \max_l \frac{\underset{(t,T,q)\in Reticulo}{max} \Delta_{t,T,l}(q)P(q_e \mid q)}{l} \tag{5.12}$$

Adicionalmente, también es necesario guardar la traza de los elementos del retículo que maximizan cada $\Delta_{t,t\prime,l}(q)$. Al alcanzar el instante T, utilizamos la ecuación 5.13 para obtener la longitud del mejor camino y a continuación obtenemos el mejor elemento final para todos los caminos de longitud L en el retículo aplicando la ecuación 5.14.

$$L = \arg\max_l \frac{\underset{(t,T,q)\in Reticulo}{max} \Delta_{t,T,l}(q)P(q_e \mid q)}{l} \tag{5.13}$$

$$(t_1^m, T, q_1^m) = \underset{(t,T,q)\in Reticulo}{\arg\max} \Delta_{t,T,L}(q)P(q_e \mid q) \tag{5.14}$$

Posteriormente, estableciendo $t_0^m = T$, recuperamos los argumentos $(t\prime\prime, t, q\prime) \in Reticulo$ que maximizan la ecuación 5.11 retrocediendo en el tiempo:

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \underset{(t\prime\prime,t_i^m,q\prime)\in Reticulo}{\arg\max} \Delta_{t\prime\prime,t_i^m,L-i}(q\prime)P(q_i^m \mid q\prime)\delta_{t_i^m,t_{i-1}^m}(q_i^m) \tag{5.15}$$

para $i \geq 1$, hasta que alcancemos $t_k^m = 0$. De este modo, $q_1^m, \ldots q_k^m$ es la mejor secuencia de hipótesis de la frase (en orden inverso).

Aplicando nuevamente la intuición, podemos considerar la complejidad espacial del algoritmo Viterbi-N como el número de acumuladores, y la complejidad temporal como el número de operaciones a realizar. En este caso, tal y como se ha explicado, para la complejidad temporal podemos tener más de un acumulador por arco. Para la complejidad temporal, para cada arco, es necesario calcular la suma del total de acumuladores de los

arcos que llegan a su punto de origen. De este modo, la ejecución del algoritmo Viterbi-N sobre el retículo de la figura 5.7, necesitaría 44 acumuladores (frente a los 82 que necesitaba el Viterbi-L) y 73 operaciones (en lugar de 119). Además, el algoritmo Viterbi-N no necesita obtener los retículos sin conflictos, lo que permite ahorrar el proceso de división del retículo conflictivo en varios que no lo sean.

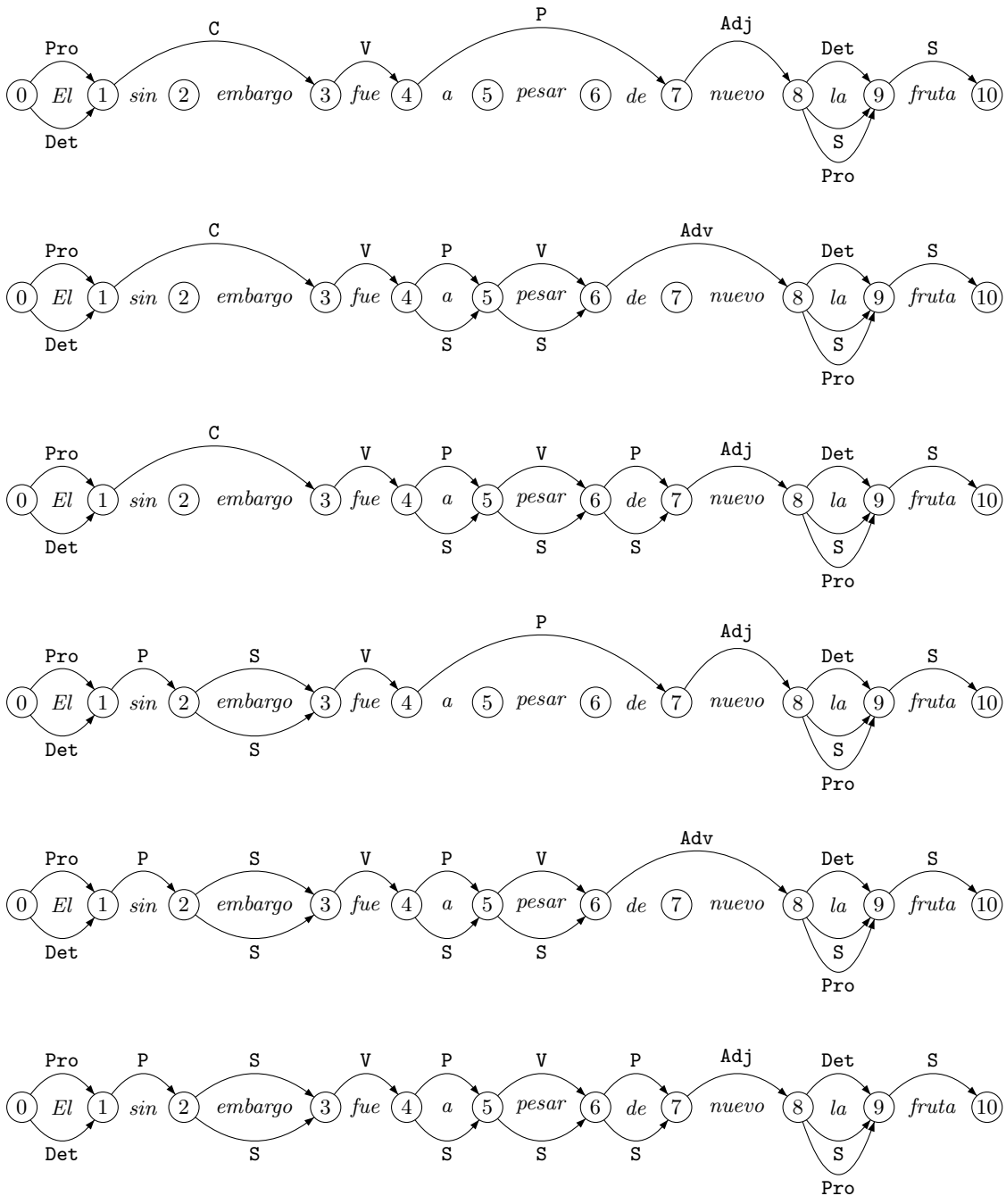

Figura 5.9: Retículos sin caminos conflictivos equivalentes al de la figura 5.8.

Parte II

Recuperación de información

Capítulo 6

Recuperación de información

La *Recuperación de Información* (RI)[1] es el área de la inteligencia artificial que trata de la adquisición, representación, almacenamiento, organización y acceso a elementos de información [9]. Desde un punto de vista práctico, un proceso de RI permite obtener un conjunto de documentos que satisfacen potencialmente una necesidad de información del usuario. Esta puntualización es importante ya que un sistema de RI no devuelve al usuario la información que éste desea sino que únicamente le indicará cuáles son los documentos que pueden resultar relevantes para la necesidad de información que aquél declara [89]. Existen, por otra parte, sistemas de RI popularmente conocidos como son los motores de búsqueda en internet[2].

En este sentido, una biblioteca tradicional, con sus fichas en papel, sus estanterías llenas de libros y su bibliotecario, es un sistema de RI ya que cumple escrupulosamente con la definición que hemos dado en el párrafo anterior. El bibliotecario adquiere nuevos volúmenes, crea sus fichas y los almacena en las estanterías de forma ordenada con la intención de recuperarlos posteriormente y satisfacer así las necesidades de información de los usuarios. Sin embargo, no es a esta clase de sistemas de RI a los que nos referimos en este capítulo sino a aquéllos en los que se utilizan herramientas informáticas para realizar todos o alguno de los procesos del sistema.

Con el objetivo de facilitar la plena comprensión de los experimentos realizados se incluye este capítulo de introducción y familiarización con los conceptos básicos relacionados con la RI. Tras delimitar el ámbito de aplicación de este tipo de sistemas, se describe el funcionamiento de uno de ellos: los paradigmas de representación interna y comparación de documentos y consultas, así como el proceso de generación de dichas representaciones internas y su implementación. Esto nos sirve de base para describir la indexación y la recuperación basadas en n-gramas de caracteres superpuestos.

[1] En inglés *Information Retrieval* (IR).
[2] Tales como Google, Altavista o Yahoo.

6.1 Tareas de recuperación de información

Existen diferentes tipos de sistemas de RI, dependiendo de la naturaleza de la tarea a realizar, entre las que podemos distinguir las siguientes[159]:

- **Recuperación *ad hoc***: Es en la que se basan los buscadores *web*. En ella el conjunto de documentos permanece estable[3], mientras que las nuevas consultas procedentes de los usuarios llegan al sistema de forma continua y dinámica.
- **Categorización o clasificación de documentos**: Consiste en la asignación de un documento a una o más clases fijadas con anterioridad en función de su contenido. A diferencia de la recuperación *ad hoc* en la que las necesidades de información son puntuales y específicas, en el caso de la categorización las necesidades permanecen estables en el tiempo, con escasas modificaciones, y su carácter es menos específico ya que recoge simultáneamente diversas necesidades potenciales de información del usuario [75]. Las necesidades de información son, por tanto, de naturaleza estática en este caso.
- **Clustering de documentos**: En contraste con la clasificación de documentos, en la que se asume la preexistencia de una serie de clases o grupos de documentos, el objetivo de la tarea de *clustering* es el de generar una serie de clases o *clusters* a partir de un conjunto dado de documentos de forma que se maximice la similaridad intra-clúster y se minimice la similaridad inter-clúster [79].
- **Segmentación de documentos**: Consiste en la división automática de un documento en subpartes semánticamente coherentes. Es decir, un documento mayor se particiona en secciones que traten subtemas diferentes, ya sea para procesar cada una de esas partes de forma separada, o para mostrar al usuario únicamente las partes relevantes del documento devuelto [68].

6.2 Modelos de representación interna

6.2.1 El paradigma *bag-of-terms*

La representación elegida para documentos y consultas resulta de especial importancia [151]. Generalmente, los documentos han sido representados como conjuntos de términos denominados *términos índice* o *palabras clave*, que son generados de forma manual o automáticamente a partir del contenido del documento. Este tipo de representación interna de los documentos se denomina *bag-of-terms* [9], y se basa en una interpretación extrema del *principio de composicionalidad* [79], según el cual la semántica de un documento viene determinada únicamente por los términos que lo forman. Este tipo de estructura no permite una representación completa y adecuada de la semántica del documento ya que no tiene en cuenta, por ejemplo, las relaciones entre términos o la existencia de diferentes sentidos para una misma palabra. Sin embargo, su sencillez

[3] Esta condición se relaja en el caso de la *web*.

conceptual y de implementación unido a que su rendimiento en la práctica resultase bastante satisfactorio han hecho que este paradigma haya venido dominando durante décadas el campo de la RI.

6.2.2 Peso asociado a un término

Si bien la semántica de un documento viene determinada por el conjunto de términos índice que lo representan, parece lógico pensar que no todos los términos tendrán la misma influencia en dicha semántica. Esta influencia se representa mediante la asignación de un valor numérico que denominaremos *peso*, de tal forma que un mayor peso de un término en un documento denotará una mayor relevancia del mismo en ese texto [9, 83].

Existen dos factores que adquieren especial relevancia a la hora de calcular el peso de un término: su frecuencia de aparición dentro de un documento, y su distribución dentro de la colección. De este modo, asumimos que cuantas más veces aparezca un término en un documento, más representativo será en su semántica, por lo que se le debería asignar un peso mayor. En esta línea, si en un documento aparece la palabra *coche* de forma iterativa, es lógico pensar que ese documento hable sobre coches [95]. En contrapartida, cuantos más documentos de una colección hagan referencia a un término, menor será la capacidad de discriminación del mismo por lo que menor debería ser también el peso que éste reciba. En efecto, si la colección de documentos pertenece a un taller de reparaciones, es muy probable que la palabra coche aparezca en una gran cantidad de estos textos por lo que no resultará de gran utilidad.

Para realizar el cálculo de pesos, es frecuente considerar también un tercer factor que tenga en cuenta el tamaño del documento [134], ya que a mayor longitud, mayor será la probabilidad de que se produzcan correspondencias, de modo que los documentos de mayor longitud se verían favorecidos.

6.2.3 Modelos de recuperación

A la hora de diseñar un sistema de RI se ha de establecer previamente un modelo de representación para los documentos y las necesidades de información del usuario, y especificar cómo han de compararse ambas representaciones. Es preciso, pues, definir el *modelo de recuperación* sobre el que ha de desarrollarse el sistema.

A continuación, describimos brevemente los modelos clásicos más representativos: los modelos booleano, vectorial y probabilístico.

El modelo booleano

El modelo booleano es el más simple de los tres aquí descritos, y se basa en la teoría de conjuntos y el álgebra de Boole [9]. En este modelo el usuario especifica en su consulta una expresión booleana formada por una serie de términos ligados mediante operadores booleanos[4]. El sistema devolverá aquellos documentos que satisfacen la expresión lógica de la consulta. De esta forma, cualquier documento de la colección podrá ser relevante o

[4]Comúnmente AND, OR y NOT.

no relevante para una consulta, pero no se establece gradación alguna que nos permita realizar una ordenación de los mismos dependiendo de su relevancia.

La dificultad para el usuario a la hora de expresar sus consultas en forma de expresión booleana, unido a la no existencia de los conceptos de correspondencia parcial y gradación de relevancia [174], han hecho que el modelo booleano se encuentre relegado a ciertos ámbitos en que se requieren correspondencias exactas, como en algunos sistemas de información legislativa [75].

El modelo vectorial

Para dar solución a las carencias del modelo booleano, el modelo vectorial [134, 135] plantea un marco formal diferente en el que es posible tanto la correspondencia parcial, como la gradación de relevancia según los pesos de los términos en consultas y documentos.

La idea básica de este modelo reside en la construcción de una matriz $\mathcal{M}$ en la que las filas se corresponderían con los documentos de la colección y las columnas con los términos que aparecen en los mismos. De este modo, en $\mathcal{M}(i,j)$ se almacenaría el peso del término j en el documento i.

Las consultas, al igual que los documentos, deben ser expresadas también en forma de vector de forma que el cálculo de la distancia geométrica entre el vector consulta y los vectores de cada uno de los documentos de la colección nos proporciona un valor que nos permite ordenar los documentos según su relevancia.

Existen varias fórmulas para el cálculo de la distancia entre vectores aunque la más conocida es la *función coseno*, que consiste en calcular el producto escalar de los dos vectores y dividirlo por el producto de sus normas, esto es, el producto de las raíces cuadradas del sumatorio de sus componentes al cuadrado,

$$sim(d_j,q) \;=\; cos\,(\Theta) \;=\; \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \;\times\; |\overrightarrow{q}|} \;=\; \frac{\sum_{i=1}^{t} w_{ij} \;\times\; w_{iq}}{\sqrt{\sum_{i=1}^{t} {w^2}_{ij}} \;\times\; \sqrt{\sum_{i=1}^{t} {w^2}_{iq}}}$$

De este modo obtenemos un valor de similitud que estará entre cero y uno. Si no hay coincidencia alguna entre los vectores el producto escalar será cero, lo que indicará que la similitud entre consulta y documento es nula. De igual modo, el grado máximo de similitud se dará cuando consulta y documento incluyan los mismos términos con idéntico peso.

Existen otras funciones de similitud distintas de la del coseno y que además no son difíciles de calcular, sino más bien de interpretar y que, por tanto, son menos aplicadas en RI.

La simplicidad y los buenos resultados unido a su capacidad para permitir correspondencias parciales y gradación de relevancia, además de aceptar consultas en lenguaje natural, son las características principales que han hecho que el modelo vectorial se convirtiese en una de las bases sobre la que se sustentan gran parte de los desarrollos y experimentos en el ámbito de la RI [65, 131, 136, 137], convirtiéndose además en el modelo de referencia con el que comparar nuevos modelos [9].

El modelo probabilístico

Si el modelo vectorial se basaba en la teoría de conjuntos y el modelo vectorial era de carácter algebraico, el modelo probabilístico se basa en la teoría de probabilidades. Las bases de este modelo fueron establecidas por Robertson y Sparck Jones en [129]. El objetivo de este modelo es el de calcular la probabilidad de que un documento sea relevante para la consulta en base a las propiedades de éste [130], o lo que es lo mismo, en base a los términos índice que contiene el documento. Según el *principio de orden por probabilidades* [128], el rendimiento óptimo de un sistema se obtiene al ordenar los documentos de acuerdo con sus probabilidades de relevancia.

El modelo parte de las siguientes hipótesis:

1. Todo documento es, bien relevante, bien no relevante para la consulta.
2. El hecho de que un documento sea considerado como relevante para una consulta no aporta información alguna respecto a la relevancia de otros documentos.

En base a estas suposiciones, dada una consulta, el modelo asigna a cada documento una medida de similaridad que se calcula como el cociente entre la probabilidad de que el documento sea relevante para la consulta y la probabilidad de que no lo sea. Los documentos devueltos al usuario se ordenan según esta medida de similaridad.

Numerosos experimentos demuestran que los procedimientos del modelo de recuperación probabilístico obtienen buenos resultados. En todo caso, los resultados no son mucho mejores que los obtenidos en el modelo booleano y en el vectorial. Posiblemente en el nuevo contexto de la recuperación a texto completo de bases de datos heterogéneas en Internet, se complique lo suficiente la recuperación como para que las técnicas de recuperación probabilística aporten mayor utilidad.

6.3 Normalización e indexación de documentos

Una vez descritos los modelos de representación interna, introducimos ahora algunas técnicas que nos permitirán convertir los documentos y consultas a algún tipo de forma canónica. A este proceso lo denominaremos *normalización* y consistirá en la aplicación sucesiva de una serie de transformaciones previas a la selección de términos índice.

6.3.1 Obtención de términos índice

La indexación de los documentos de la colección es un proceso clave de la RI. La extracción de los términos asociados a los documentos se realiza mediante la concatenación de una serie de transformaciones denominadas genéricamente *operaciones de texto* [9]. Esta serie de transformaciones convierte los términos del texto a algún tipo de forma canónica que facilita el establecimiento de correspondencias durante el proceso de búsqueda.

Estas operaciones suelen consistir en el análisis léxico del texto, la eliminación de las denominadas *palabras vacías*[5], la eliminación de mayúsculas y signos de puntuación,

[5] Usualmente denominadas por el término en inglés *stopwords*.

la *extracción de raíces*[6], y la selección de los *términos índice* que serán utilizados posteriormente por el sistema de búsqueda.

Análisis léxico

Generalmente, los documentos introducidos en un sistema de RI estarán en formato texto, es decir, un flujo de palabras, números, fechas, signos de puntuación, siglas, abreviaturas, fórmulas matemáticas, y todo ello separado normalmente mediante espacios. El proceso de análisis léxico se encarga de identificar todos estos ítems en el texto resolviendo las posibles ambigüedades[7].

La identificación de los distintos caracteres y su tratamiento variará según el idioma, aunque el nivel de complejidad de los analizadores léxicos no suele ser muy elevado, resultando ser herramientas sencillas y eficientes.

Eliminación de palabras vacías

Denominamos *palabras vacías* [9, 83] a aquéllas cuya capacidad de discriminación es ínfima debido a su excesiva frecuencia de aparición, por ejemplo, formas verbales de *ser* o *estar*, o a que su contenido semántico es escaso; por ejemplo, artículos, preposiciones y conjunciones. Dada su poca utilidad, estas palabras son descartadas, lo que puede suponer un ahorro de recursos de entre un 25 % y un 40 % [9, 174].

Del mismo modo, en el caso de las consultas es conveniente eliminar, además de las palabras vacías, el contenido de su *metanivel* [107, 108], es decir, aquellas expresiones que no aportan información a la búsqueda sino que forman parte de la formulación de la pregunta como es el caso, por ejemplo, de "*obtener los documentos que traten sobre ...*". Para ello sería necesario emplear una segunda lista de palabras vacías que denominaremos *meta-palabras vacías*[8].

Eliminación de mayúsculas y signos ortográficos

En los sistemas de RI es habitual pasar los términos en mayúsculas a minúsculas con el fin de evitar, por ejemplo, la falta de correspondencia de términos que, por ejemplo, estén en mayúscula simplemente por el hecho de aparecer al principio de una oración [83]. Por la misma razón, suelen ser eliminados también los signos ortográficos tales como tildes o diéresis.

Extracción de raíces

La variación léxica es una característica del lenguaje humano [76] que complica la tarea de comparación de documentos y consultas. Desde el punto de vista de un sistema de RI que no maneja la variación léxica, los términos *coche* y *coches* son diferentes, aunque es

[6] Usualmente denominada por el término en inglés *stemming*.
[7] Este proceso suele denominarse con el término en inglés *tokenization*.
[8] Usualmente denominadas por el término en inglés *meta-stopwords*.

casi seguro que un usuario que incluya en su consulta el término *coche* estará interesado también en los documentos que contengan el término *coches*.

La *extracción de raíces* consiste en la reducción de un término a su raíz[9] bajo la suposición de que ésta contiene la semántica básica del mismo [9, 83]. Entre los algoritmos de extracción de raíces destacan el de Porter [124], quizás el más popular, y el algoritmo de Lovins [93]. La aplicación de estos algoritmos permite reducir de forma significativa la cantidad de recursos necesarios a la vez que facilita la correspondencia de términos que, difiriendo en la forma, coinciden en la semántica. Sin embargo, también presentan errores por exceso y por defecto. Por una parte, puede ocurrir que dos palabras con escasa o nula relación semántica sean reducidas a una misma raíz y por otra puede ocurrir todo lo contrario, esto es, que dos términos muy próximos semánticamente no sean reducidos a raíces idénticas. En el primer caso se producirían correspondencias indeseadas mientras que en el segundo, correspondencias esperadas no tendrían lugar.

Selección de términos índice

Después de las transformaciones previas sólo queda seleccionar aquellos términos que servirán para indexar cada documento. Lo habitual es utilizar todos los términos, lo que se denomina *representación a texto completo* [9]. Sin embargo, existe también la posibilidad de seleccionar un subconjunto de los mismos y que sean éstos los que se utilicen como términos índice. Esta selección puede ser realizada de forma manual, por expertos, o de forma automática.

6.3.2 Indexación

A la hora de plantear una búsqueda en una colección de documentos, existen dos estrategias distintas. La primera consiste en realizar una búsqueda secuencial, como es el caso de la herramienta `grep` de Unix [52] o de algunos sistemas de RI como el *seft* [39]. Este tipo de búsqueda resulta adecuada para colecciones pequeñas en las que se producen amplias y continuas modificaciones. Sin embargo, para colecciones grandes de carácter estático o semi-estático, como es el caso de las bases de datos documentales, las hemerotecas o la propia *web*, es más común el uso de *índices* que permitan acelerar la búsqueda. De este modo, se añadirían al sistema dos nuevos componentes: los índices, conteniendo las estructuras asociadas a la colección, y el *motor de indexación*; que será el software encargado de la generación, mantenimiento y consulta de dichas estructuras.

La estructura de datos más ampliamente utilizada en RI es el *índice invertido* o *fichero invertido* [9, 83] que consiste en una lista de todos los términos índice existentes en la colección y, para cada uno de ellos, una lista de los documentos en los que aparece dicho término. En el caso de modelos que utilizan pesos, es posible almacenar también, bien los datos necesarios para el cálculo de los mismos o bien directamente los propios pesos.

En lo que respecta a la estructura de datos a utilizar para implementar el índice, existen diversas posibilidades [82]: mediante matrices asociativas, árboles binarios de búsqueda, etc.

[9] Esta raíz no tiene que coincidir necesariamente con el lexema de la palabra.

6.4 El proceso de búsqueda

El objetivo final de un sistema de RI es el de satisfacer las necesidades de información de los usuarios ofreciéndoles una lista de documentos, típicamente ordenada, en los que se espera que se encuentre aquello que el usuario busca. Para ello el usuario ha de indicar al sistema cuál es su necesidad concreta de información por medio de una consulta. El sistema se encargará de transformar la consulta a un formato que dependerá del modelo de recuperación empleado y que permitirá compararla con los documentos que conforman la colección.

Las consultas constituyen por tanto el mecanismo que permitirá al usuario acotar y recuperar aquellos documentos de la colección que contienen la información en la que éste está interesado. Esto representa un desafío para los sistemas de RI debido a que los usuarios de los mismos no suelen ser especialmente hábiles a la hora de formular sus consultas. Además, a menudo los términos utilizados por los usuarios en las interrogaciones, y por los autores en la elaboración de los documentos, difieren substancialmente, lo que puede dificultar el establecimiento de correspondencias [175].

Con el objetivo de aumentar la efectividad de las consultas, los sistemas de RI suelen incluir mecanismos de reformulación, añadiendo nuevos términos relacionados con los especificados en la consulta inicial.

6.4.1 Expansión de consultas mediante tesauros

El término *expansión de consultas* [16, 27, 74, 97][10] engloba a toda una serie de procesos automáticos o semiautomáticos para la reformulación o refinamiento de la consulta original, con el fin de mejorar la efectividad del proceso de recuperación, mediante la adición de nuevos términos, ya sea relacionados con los ya introducidos por el usuario o bien con los asociados a documentos que se saben o se suponen relevantes.

Podemos clasificar las técnicas de expansión de consultas más comúnmente utilizadas en las basadas en *tesauros* y las basadas en *realimentación*.

Las técnicas de expansión de consultas consiguen por lo general mejorar el rendimiento de éstas. Sin embargo, también es posible que entre los términos introducidos se incluyan por error algunos que no guarden relación con los originales, lo que perjudicaría el rendimiento del sistema al aumentar el número de documentos recuperados que en realidad no son relevantes para el usuario [75].

Un *tesauro* es una base de datos lexicográfica que almacena una representación jerárquica de un lexicón de acuerdo con las relaciones semánticas entre sus palabras [100].

El empleo de tesauros permite reformular la consulta inicial de forma automática [158, 169] o bien de forma manual [83], ofreciendo al usuario la posibilidad de navegar por su estructura y elegir los términos deseados.

Una de las herramientas más utilizadas en este tipo de expansión es *WordNet* [14, 46, 63, 105, 106], una base de datos lexicográfica del inglés en la que sustantivos, verbos, adjetivos y adverbios se organizan en conjuntos de sinónimos denominados *synsets*, de

[10] Usualmente denominada por el término en inglés *query expansion*.

modo que cada uno de ellos se asocia a un sentido determinado. *WordNet* establece la siguientes relaciones semánticas:

- ***Sinonimia* y *antonimia***: Existe una relación de sinonimia entre dos o más términos cuando tienen el mismo significado. Por el contrario, la relación de antonimia denota significados opuestos o contrarios; por ejemplo, *fragmento*, *pedazo* y *trozo* son sinónimos mientras que *grande* y *pequeño* son antónimos.

- ***Hiperonimia* e *hiponimia***: Se dice que un término es hiperónimo de otro, denominado hipónimo, cuando el significado del primero incluye al del segundo; por ejemplo *pájaro* es hiperónimo de *jilguero* e hipónimo de *ave*.

- ***Meronimia* y *holonimia***: Se denomina merónimo al término cuyo significado constituye una parte del significado total de otro, denominado holónimo; por ejemplo, *mano* es holónimo de *dedo* y merónimo de *brazo*.

Existe también una base de datos paralela para lenguas europeas denominada *EuroWordNet* [170].

Las aproximaciones basadas en tesauros no consiguen, por sí solas, mejorar los resultados obtenidos debido a que una misma palabra puede tener asociados distintos sentidos [169]. Para resolver este problema sería necesaria la utilización de técnicas de *desambiguación del sentido de las palabras* [44, 104, 148][11]. Sin embargo, la aplicación de este tipo de técnicas no siempre es posible ya que las consultas de los usuarios a menudo constan de unas pocas palabras y no presentan necesariamente una estructura gramatical correcta[12].

6.4.2 Expansión de consultas mediante realimentación

Las técnicas de expansión de consultas basados en *realimentación por relevancia*[13] son las más utilizadas en la actualidad debido a su buen comportamiento general [9].

Estas estrategias basan su funcionamiento en el empleo de la información acerca de la relevancia de un subconjunto de los documentos obtenidos a partir de la consulta inicial del usuario con el fin de:

- Expandir la consulta en sentido estricto, mediante la adición de nuevos términos pertenecientes a los documentos considerados relevantes.

- Optimizar el rendimiento de la consulta mediante la modificación de los pesos de los términos que la componen.

[11] Usualmente denominadas por el término en inglés *word-sense disambiguation*.

[12] Además sería necesario aplicar un proceso de desambiguación similar en los documentos, lo que resultaría demasiado costoso.

[13] Usualmente denominada por el término en inglés *relevance feedback*.

La información acerca de la relevancia de los documentos devueltos en primera instancia puede ser obtenida mediante interacción con el usuario, o de forma automática mediante *expansión por pseudo-relevancia* o *expansión ciega* [83], que consiste en suponer que los n primeros documentos obtenidos inicialmente son relevantes. En este último caso, existe el riesgo de tomar como relevantes documentos que en realidad no lo eran, lo que afectará negativamente al rendimiento del sistema. A pesar de esto, los resultados obtenidos acostumbran a ser suficientemente satisfactorios [75].

Por otro lado, las técnicas de expansión de consultas mediante realimentación presentan una serie de ventajas respecto a las basadas en tesauros:

- Aíslan al usuario de los detalles del proceso de reformulación, solicitándole únicamente, en el caso de que no se trate de una expansión ciega, que indique cuáles de los documentos devueltos inicialmente considera relevantes.

- Permiten dividir el proceso completo de búsqueda en una serie de pasos más pequeños y fáciles de manejar.

Este tipo de expansión fue puesta en práctica inicialmente en el contexto del modelo vectorial [131], aunque su aplicación se extendería posteriormente al modelo probabilístico [129].

6.5 Indexación y recuperación basada en n-gramas

Los sistemas de RI tradicionales utilizan la técnica de extracción de raíces para llevar a cabo tanto la tarea de indexación de los documentos como el procesado de las consultas. Sin embargo, uno de los mayores inconvenientes a la hora de utilizar este tipo de técnicas es su fuerte dependencia del lenguaje. Esta dependencia hace que acercamientos que funcionan bien para una o dos lenguas, posiblemente debido a que han sido cuidadosamente adaptadas a ellas, no funcionen en absoluto para otras, si no es mediante una costosa adaptación que incluso afecta a veces a algoritmos, con la consiguiente recodificación de *software*. La complejidad de estas tareas de indexación y procesado de consultas es aún mayor cuando tratamos de hacer RI en un entorno multilingüe en el que los documentos y las consultas pueden encontrarse en decenas de lenguas diferentes, por ejemplo, el repositorio de documentos de los gobiernos de la Unión Europea o el de una compañía multinacional. La complejidad de los sistemas de RI sólo puede ser reducida en estos casos minimizando la cantidad de información lingüística involucrada sin que el deterioro en la precisión y la eficiencia sea significativo. Para dar solución a esta problemática, el *Johns Hopkins University Applied Physics Lab* (JHU/APL) [102, 103] ha propuesto la utilización de n-gramas de caracteres superpuestos para la indexación de documentos.

Formalmente un n-grama de caracteres no es más que una subsecuencia de longitud n de una secuencia de caracteres dada. En este punto, por ejemplo, podemos dividir la palabra `"patata"` es los 3-gramas de caracteres superpuestos `-pat-`, `-ata-`, `-tat-` y `-ata-`.

Centrándonos en RI monolingüe, la adaptación resulta sencilla ya que tanto las consultas como los documentos son simplemente desmenuzados en n-gramas superpuestos,

en lugar de en palabras. Los n-gramas resultantes son entonces procesados como lo haría cualquier motor de recuperación. Su interés viene dado por las posibilidades que ofrecen, especialmente en lengua no inglesa, al facilitar un modo alternativo para la *normalización* de formas de palabras y permitir tratar lenguas muy diferentes sin procesamiento específico al idioma y aún cuando los recursos lingüísticos disponibles son escasos o inexistentes.

Capítulo 7

Evaluación de sistemas de RI

En este capítulo nos centraremos en la evaluación de la efectividad de los sistemas de RI, diferenciando dos aspectos: por una parte las medidas de evaluación empleadas y, por otra, las colecciones de referencia necesarias para dicha evaluación.

7.1 Medidas de evaluación

A la hora de evaluar un sistema de RI, podemos fijarnos en diferentes aspectos [9]: su eficiencia en términos de coste espacio-temporales; su efectividad a la hora de devolver el mayor número de documentos relevantes, minimizando al mismo tiempo el número de no relevantes devueltos [155]; el esfuerzo requerido para realizar o modificar una consulta; y la amigabilidad de la interfaz.

En esta sección introducimos las medidas de evaluación más comúnmente empleadas para medir la efectividad de los sistemas de RI.

Las medidas de evaluación aquí descritas pueden ser calculadas bien como medidas puntuales relativas a una consulta concreta, o bien como medidas globales relativas a un conjunto de consultas. En este último caso las medidas se calculan promediando los valores obtenidos para cada consulta individual respecto al número de consultas empleado, excepto en el caso de la precisión media de documento, cuyo caso concreto se trata más adelante.

7.1.1 Precisión y cobertura

Ante una necesidad de información de un usuario, podemos clasificar la colección de documentos en aquéllos que son relevantes y los que no lo son. En este sentido, las medidas básicas de evaluación de un sistema de RI tratarán de significar la idoneidad del conjunto de documentos recuperados atendiendo a la cantidad de textos relevantes y no relevantes recuperados. Un sistema de RI será mejor cuantos más documentos relevantes incluya en la respuesta a una consulta, a la vez que minimiza el número de documentos no relevantes devueltos. Así, la *precisión* nos indica la proporción de documentos que son relevantes de

entre todos los recuperados mientras que la *cobertura*[1] nos dirá qué proporción de todos los documentos relevantes que había en la colección hemos recuperados.

Definimos formalmente las medidas de *precisión* y *cobertura* para una consulta dada como:

$$Precisión = \frac{|R \cap A|}{|A|} \tag{7.1}$$

$$Cobertura = \frac{|R \cap A|}{|R|} \tag{7.2}$$

donde A, de tamaño $|A|$, es el conjunto de documentos devueltos por el sistema; R, de tamaño $|R|$, es el conjunto de documentos relevantes existentes en la colección; y $R \cap A$, de tamaño $|R \cap A|$, es el conjunto de documentos relevantes devueltos por el sistema.

7.1.2 Precisión a los 11 niveles estándar de cobertura

Tanto la precisión como la cobertura evalúan la idoneidad del conjunto de documentos recuperado sin tener en cuenta el orden en que son devueltos y requiriendo además un examen de todo el conjunto. Sin embargo, el usuario suele recibir los documentos ordenados de mayor a menor grado de relevancia. De este modo, los valores de precisión y relevancia irán variando a medida que el usuario va examinando, ordenadamente, los documentos recuperados. En consecuencia, podemos calcular los valores de precisión para el conjunto de documentos examinados a medida que se vayan alcanzando unos determinados niveles de cobertura, esto es, cuando se hayan examinado un porcentaje dado de los documentos relevantes. Generalmente se muestran los niveles de precisión al nivel 0.0 de cobertura, al nivel 0.1, al 0.2, y así sucesivamente en incrementos de 0.1 hasta alcanzar el nivel 1. A dichos niveles se les denomina los *11 niveles estándar* de cobertura.

Dado que no siempre es posible calcular la precisión a un nivel de cobertura concreto, las precisiones son interpoladas mediante la expresión

$$Pr(r_j) = max_{r_j \leq r \leq rj+1} Pr(r)$$

donde r_j es el j-ésimo nivel de cobertura estándar, es decir, la precisión interpolada a un nivel de cobertura estándar determinado es la precisión máxima conocida en cualquier valor entre dicho nivel de cobertura y el siguiente.

A modo de ejemplo, supongamos una colección de 20 documentos de los que 4 son relevantes para una consulta dada. Supongamos que nuestro sistema, ante dicha consulta, ha devuelto dichos documentos en las posiciones 1ª, 3ª, 8ª y 12ª. Los niveles de precisión y cobertura para cada documento relevante encontrado son:

[1] Usualmente denominada por el término en inglés *recall*.

	Precisión		Cobertura	
1º documento relevante recuperado	1	(1/1)	0,25	(1/4)
2º documento relevante recuperado	0,667	(2/3)	0,50	(2/4)
3º documento relevante recuperado	0,375	(3/8)	0,75	(3/4)
4º documento relevante recuperado	0,333	(4/12)	1	(4/4)

Por lo tanto, de acuerdo con la regla de interpolación previamente introducida, la precisión a los niveles de cobertura del 0 al 2 es 1, del 3 al 5 es 0,667, del 5 al 7 es 0,375, y del 8 al 10 es 0,333.

7.1.3 Precisión a los n documentos devueltos

Otra posibilidad, similar a la anterior, de cara a comparar dos conjuntos de resultado ordenados, es mostrar la precisión obtenida con los n primeros documentos recuperados, en lugar de a determinados niveles de cobertura.

7.1.4 Precisión media no interpolada

Como ya hemos comentado, para un sistema de RI no sólo es importante maximizar el número de documentos relevantes recuperados al tiempo que se minimiza el de no relevantes, sino que además se espera que los primeros aparezcan en las posiciones más altas de la lista de documentos devueltos. La precisión media no interpolada es una medida que tiene en cuenta el orden en que los documentos relevantes son devueltos. Se calcula como la media de las precisiones obtenidas después de que cada documento relevante es recuperado.

La precisión media interpolada para el ejemplo introducido para la precisión a los 11 niveles estándar de cobertura sería:

$$\textit{Pr. no int.} = \frac{1 + 0{,}667 + 0{,}375 + 0{,}333}{4} = 0{,}594$$

7.1.5 Precisión media de documento

A pesar de que todas las medidas presentadas en los apartados anteriores se calculan para una única consulta, es fácil extenderlas a un conjunto de ellas calculando la *media ponderada*, es decir, dividiendo la suma de las medidas obtenidas para cada consulta por el número de consultas.

Sin embargo, la precisión media de documento no tiene sentido a nivel de una única consulta, sino que es siempre calculada para un conjunto de ellas. Para ello se calcula la suma de todas las precisiones obtenidas, para cada consulta realizada, después de cada documento relevante recuperado, y se divide finalmente por el número global de documentos relevantes, en lugar de dividir por el número total de consultas.

Supongamos que, además de la consulta que venimos utilizando como ejemplo en apartados anteriores, tenemos otra consulta que cuenta con 2 documentos relevantes que

son devueltos en las posiciones 1ª y 3ª. Las precisiones para cada documento devuelto por esta nueva consulta serían:

	Precisión		Cobertura	
1º documento relevante recuperado	1	(1/1)	0,5	(1/2)
2º documento relevante recuperado	0,667	(2/3)	1	(2/2)

y la precisión media de documento se calcularía, a nivel global del conjunto de las dos consultas, como:

$$Pr.\ doc.\ =\ \frac{\big(1+0{,}667+0{,}375+0{,}333\big)\ +\ \big(1+0{,}667\big)}{4\ +\ 2}\ =\ 0{,}674$$

7.1.6 Precisión-R

Es la precisión obtenida a los R documentos recuperados, donde R es el número de documentos relevantes para esa consulta.

Retomando de nuevo el ejemplo utilizado para la precisión a los 11 niveles estándar de cobertura, y dado que existían 4 documentos relevantes para la consulta en cuestión, la precisión-R será la precisión a los 4 documentos recuperados, esto es, 0,5.

7.2 Colecciones de Referencia

Uno de los mayores problemas en el ámbito de la Recuperación de Información ha sido, hasta hace poco, la falta de colecciones de evaluación con suficiente entidad y de libre acceso, que permitiesen una evaluación de los sistemas lo más completa posible y facilitasen la comparación de resultados.

Una colección de evaluación está compuesta por tres elementos: los documentos, las consultas, y una lista de los documentos de la colección que son relevantes para cada consulta.

La elección de una colección adecuada resulta ser de suma importancia a la hora de evaluar un sistema de RI, ya que es esta elección la que nos proporcionará la convicción de que los resultados obtenidos son fiables y representativos. La calidad de una colección viene dada por diversos aspectos:

- *Su disponibilidad para la comunidad científica.* El libre acceso a una colección promueve su utilización por otros investigadores, facilitando la comparación de resultados.
- *El tamaño de la colección.* Cuanto mayor sea el repositorio de documentos y el número de consultas a utilizar, más se ajustarán los resultados obtenidos al comportamiento real del sistema [72].
- *La calidad de las consultas.* Dicha calidad depende de su variedad, de la diversidad de construcciones empleadas, y de su correspondencia o no con necesidades de información realistas.

- *La calidad de los documentos.* Viene dada por la variedad de los mismos y por su realismo en cuanto a que no hayan sido sometidos a ningún tipo de tratamiento especial.

Colecciones como *Cranfield* [78], o CACM e ISI [48], por ejemplo, se mostraban inadecuadas a las necesidades para una evaluación fiable del sistema debido a su pequeño tamaño (número de documentos: 1.400, 3.204 y 1.460, respectivamente; número de consultas: 22, 52, 35+41). Esta situación ha cambiado recientemente gracias al trabajo de la *Text REtrieval Conference (*TREC*)* y del *Cross-Language Evaluation Forum (*CLEF*)*.

7.2.1 *Text REtrieval Conference (*TREC*)*

La situación inicial de falta de colecciones de evaluación estándar dio un giro tras la celebración, en 1992, del primer *Text REtrieval Conference (*TREC[2]*)*, congreso de carácter anual organizado por el *National Institute of Standards and Technology (*NIST*)* y la *Information Technology Office* de la *Defense Advanced Research Projects Agency (*DARPA*)*. El objetivo perseguido por la organización del TREC es el de apoyar la investigación en el campo de la RI. Para ello:

- Facilita la infraestructura, herramientas y metodologías necesarias para la evaluación a gran escala de sistemas de RI.
- Promueve la comunicación entre industria, gobiernos e investigadores, facilitando de este modo la transferencia tecnológica.

Las colecciones de documentos empleadas en el TREC son del orden de decenas, o incluso centenares, de miles de documentos. En su mayor parte se trata de artículos periodísticos procedentes de periódicos o agencias de noticias. Tal es el caso, por ejemplo, de la colección de *Los Angeles Times*, formada por artículos publicados en dicho periódico durante el año 1994 y que suponen 131.896 artículos (475 MB) con una longitud media de 527 palabras por documento.

Junto con las colecciones de documentos, TREC suministra una serie de requerimientos o necesidades de información, denominados *temas*[3], en torno a 50 por edición. El proceso de convertir dichos *temas* en *consultas* efectivas debe ser llevado a cabo por el propio sistema. Los participantes deben enviar a la organización, dentro de un plazo dado, los resultados devueltos por sus sistemas para dichos *temas*.

El mayor problema surge a la hora de identificar los documentos relevantes para cada *tema* debido al gran número de documentos y consultas existentes. Para esta tarea se emplea una técnica denominada *pooling* [168], consistente en tomar, para cada *tema*, los K primeros documentos devueltos[4] por cada uno de los sistemas participantes. Dichos documentos son luego revisados por especialistas, que son quienes establecen la relevancia o no de cada uno de ellos. En lo que se refiere al concepto de *relevancia*, la organización de TREC optó por el siguiente criterio[5]:

[2] `http://trec.nist.gov` (visitada en marzo de 2009).
[3] Del término en inglés *topics*.
[4] Generalmente se toma K=100.
[5] Véase `http://trec.nist.gov/data/reljudge_eng.html` (visitada en marzo 2009).

> *"Si estuviera redactando un informe sobre la materia del* tema *en cuestión y pudiese usar para dicho informe la información contenida en el documento examinado, entonces dicho documento será considerado relevante"*

7.2.2 *Cross-Language Evaluation Forum (*CLEF)

A pesar de su inestimable contribución, las diferentes ediciones de TREC se han centrado en el inglés, salvo contadas excepciones, por lo que la investigación en sistemas de RI en otros idiomas seguía encontrándose con los mismos problemas. Tal era el caso del español, ya que las colecciones empleadas en su sección dedicada al español de las ediciones TREC-4 y TREC-5, y formada por artículos de noticias escritos en español (de México), no están ya disponibles.

La incorporación en la segunda edición del *Cross-Language Evaluation Forum (*CLEF-*2001*[6]*)* de una colección para el español peninsular, cambió esta situación. El CLEF es un congreso de corte similar al TREC, pero dedicado a las lenguas europeas, tanto en tareas monolingües como multilingües. Si en su primera edición, en el año 2000, los idiomas disponibles eran inglés, francés, alemán e italiano, las colecciones disponibles se han ido ampliando a español, portugués, finlandés, ruso, búlgaro y húngaro.

[6] Véase `http://www.clef-campaign.org` (visitada en marzo 2009).

Parte III

Corrección ortográfica

Capítulo 8

Estado del arte

En este capítulo nos remontaremos a los orígenes de la corrección ortográfica automática. Presentaremos los diferentes métodos utilizados a lo largo de los últimos cuarenta años, indicando las novedades aportadas en cada caso [87]. Comenzaremos describiendo las técnicas de corrección basadas en n-gramas, en contraposición a aquéllas que hacen uso de diccionarios, para explicar a continuación las técnicas de corrección basadas en la distancia de edición, y terminar haciendo mención a acercamientos basados en similaridad de claves, en reglas, en n-gramas, en redes neuronales y en estrategias probabilísticas.

8.1 Detección de errores

En un período inicial, los esfuerzos se dirigían a la *detección de errores* mediante técnicas eficientes de emparejamiento de patrones y comparación de cadenas. Se trataba, *groso modo*, de comprobar si una determinada cadena estaba o no en una lista de palabras predefinida.

Desde muy temprano se diferenciaban dos técnicas para la detección de errores, una basada en n-gramas [64, 73, 111, 179] y la otra basada en la búsqueda en uno [5, 142, 154] o varios [121] diccionarios. Mientras que la primera era más eficiente, la segunda era más precisa, aunque las búsquedas cuando el diccionario era demasiado grande solían penalizar el factor tiempo.

Históricamente, los sistemas de reconocimiento de texto han hecho uso de las técnicas de n-gramas mientras que para los correctores ortográficos se han preferido las basadas en diccionario.

8.1.1 Técnicas basadas en el análisis de n-gramas

Los *n-gramas* son sub-secuencias de n letras en palabras o cadenas donde n acostumbra a ser uno, dos o tres. Los n-gramas de una letra reciben el nombre *unigramas* o *monogramas*, de los de dos *digramas* o *bigramas* y los de tres *trigramas*.

En general, las técnicas de detección de errores basadas en esta aproximación, consisten en buscar cada una de las sub-secuencias de una palabra en una tabla precompilada de n-gramas con el fin de saber si existen, o cuál es la frecuencia con la que aparecen.

	a	**c**	**l**	**e**	**s**	**t**	**u**	**z**
a					1			1
c	1			1				
l	1			1				
e			1		1			
s	1					1		
t				1				
u			1					
z							1	

Tabla 8.1: Tabla binaria de bigramas.

Aquéllas que contienen n-gramas poco frecuentes o inexistentes, son identificadas como desconocidas.

Para la obtención de la tabla de n-gramas es necesario disponer de un diccionario o de un gran *corpus* de texto. Estas tablas pueden adquirir una gran variedad de formas. Las más sencillas son las *tablas binarias de bigramas* que consisten en una tabla bidimensional indexada por los caracteres del alfabeto, representando así todos los bigramas posibles. Cada elemento de la tabla es puesto a 1 si ese bigrama aparece en al menos una palabra, y a 0 si no aparece en ninguna.

Ejemplo 8.1. *Ilustraremos esta técnica con un ejemplo sencillo. Supongamos que queremos obtener la tabla binaria de bigramas para las palabras de la siguiente frase:*

la casa es azul celeste

La tabla binaria de bigramas estaría indexada por los caracteres del alfabeto. En nuestro ejemplo $\Sigma = \{a, c, l, e, s, t, u, z\}$. *De esta forma el primer carácter del bigrama identificará la fila y el segundo la columna. Así, para indicar que el bigrama* "la" *está presente en al menos una palabra de nuestro lexicón, pondremos un 1 en la fila* l*, columna* a*. En este caso, el conjunto de bigramas es* $B = \{la, ca, as, sa, es, az, zu, ul, ce, el, le, es, st, te\}$. *La tabla 8.1 se corresponde con la tabla binaria de bigramas de nuestro ejemplo en la que se han omitido los ceros en las casillas de bigramas no existentes.*

■

Se puede extender la idea a las *tablas binarias de trigramas* sólo con aumentar a tres, de forma natural, las dimensiones de la tabla. Podemos hablar también de *tablas de n-gramas posicionales* y *no posicionales*. En las no posicionales no importa la posición en que aparezca el n-grama mientras que en las posicionales sí. En consecuencia, para el caso de tablas de n-gramas posicionales necesitaremos $(l - n) + 1$ tablas siendo l la longitud máxima de las palabras del diccionario y n la longitud de los n-gramas. Si volvemos sobre el ejemplo 8.1, la palabra más larga es *"celeste"* por lo que el número de tablas posicionales que necesitaríamos sería 6, y los bigramas que habría en cada una de ellas serían los que se muestran a continuación:

$$B_1 = \{la, ca, es, az, ce\} \quad B_4 = \{es\}$$
$$B_2 = \{as, zu, el\} \quad B_5 = \{st\}$$
$$B_3 = \{sa, ul, le\} \quad B_6 = \{te\}$$

Las técnicas de detección de errores basadas en n-gramas son especialmente útiles para detectar aquéllos producidos por dispositivos de *reconocimiento óptico de caracteres* (ROC)[1] ya que estos errores suelen consistir en la confusión entre dos caracteres con rasgos similares (como la *D* y la *O*, la *S* y el *5* ó la *t* y la *f*) que con frecuencia resultan en n-gramas improbables. En este sentido, Harmon [66], continuando con el trabajo de Sitar [144], indicó que en grandes cantidades de texto genérico publicado en inglés, el 42 % de todas las combinaciones posibles de bigramas nunca ocurren y que además, si se substituye una letra por otra elegida aleatoriamente de entre todas las del alfabeto, se obtiene al menos un bigrama con probabilidad cero en el 70 % de los casos.

8.1.2 Técnicas basadas en diccionario

Las técnicas basadas en diccionario consisten simplemente en comprobar si una palabra o cadena aparece en una lista de palabras aceptables que es el diccionario.

El principal problema surge cuando el diccionario alcanza un tamaño considerable, el tiempo de respuesta para una búsqueda secuencial, crece exponencialmente. En PLN, el número de entradas del diccionario va desde unos cuantos miles hasta millones, por lo que disponer de técnicas que permitan un rápido acceso a cualquier palabra adquiere una especial relevancia. Esta cuestión ha sido abordada desde tres puntos de vista diferentes: vía algoritmos eficientes de búsqueda y encaje de patrones, mediante esquemas de partición de diccionarios, y aplicando técnicas de procesamiento morfológico.

La técnica más comúnmente utilizada para acelerar el acceso al diccionario es el uso de *tablas asociativas*[2], aunque se han utilizado otras como los *árboles binarios de búsqueda* [82]. La principal ventaja de estas tablas radica en su naturaleza de acceso aleatorio que permite eliminar un gran número de comparaciones necesarias en la búsqueda secuencial o incluso en la de árboles. Sin embargo, presentan también una desventaja como es la necesidad de encontrar una buena *función asociativa*[3] que permita colisiones sin necesidad de que la tabla sea de gran tamaño.

Para reducir el tiempo de búsqueda se han utilizado otras técnicas estándar, como los *árboles binarios de búsqueda* [82], o la técnica de reconocimiento de patrones sobre expresiones regulares propuesta por Aho y Corasick [5], que consistía en un método para crear un AF para representar un pequeño conjunto de términos (palabras clave) que debían ser encajados en un *corpus*.

Además, otro factor a tener en cuenta es el espacio de almacenamiento necesario para el diccionario por lo que ya en 1960, Blair [17] publicaba un método que consistía en asignar pesos a las letras con el fin de construir abreviaturas de 4 ó 5 caracteres para todas las palabras del diccionario. Cuando dos abreviaturas coincidían, se asumía que las palabras coincidían también. Esto permitía reducir el tamaño del diccionario, así como el número

[1] Resulta frecuente el uso del término procedente de las siglas en inglés OCR.
[2] Resulta frecuente el uso del término en inglés *hash table*.
[3] Resulta frecuente el uso del término en inglés *hash function*.

de comparaciones a realizar para cada palabra, ya que el tamaño de las mismas disminuía a 4 ó 5 letras.

Una forma bastante peculiar de construir abreviaturas fue la usada por Davidson [37] basándose en una propuesta de Odell y Russell [114] para aplicaciones de corrección de ortografía fonética. Este método tenía su base en unas clases de equivalencia formadas por letras que podrían ser fácilmente intercambiables. A cada una de estas clases se las identificaba mediante un dígito entre 0 y n, siendo $n+1$ el número de clases de equivalencia diferentes. La técnica consistía en representar cada palabra mediante su primera letra seguida por los dígitos identificadores de las clases de equivalencia a las que pertenecían cada una de las demás letras de la palabra. A continuación se eliminaban los ceros y los caracteres contiguos idénticos se agrupaban. Esta técnica resulta muy rudimentaria teniendo en cuenta la granularidad de la misma y el hecho de que no ofrece una función que permita graduar los candidatos, sino que simplemente se le muestran éstos al usuario.

Otra forma de economizar espacio consistía en almacenar únicamente la raíz de cada palabra en lugar de todas sus variantes morfológicas (plurales, tiempos verbales, ...). Elliot [43], en 1988, describía un componente de procesamiento morfológico mejorado para el programa *spell* de Unix. El autor era consciente de que utilizar métodos simples que únicamente se limitasen a truncar afijos conducía, en muchos casos, a aceptar palabras inexistentes en el lenguaje, por lo que ideó una solución que se basaba en la creación de clases de equivalencia de afijos a partir de similaridades ortográficas.

Por su parte, Peterson [121] proponía dividir el diccionario en tres niveles. Un primer nivel que se almacenaría en memoria caché con unos cientos de palabras que representasen el 50 % de los accesos al diccionario. El segundo nivel se almacenaría en memoria principal y estaría formado por la palabras que supusiesen otro 45 % de accesos, quedando el 5 % restante para un tercer nivel que se almacenaría en memoria secundaria. Este método trataba de alcanzar un compromiso entre la rapidez de acceso que ofrecen los tres niveles de memoria y la capacidad de almacenamiento de cada una de ellas, teniendo en cuenta la frecuencia de aparición de las palabras en los textos.

Con los avances tecnológicos, que progresivamente permitían incrementar de forma considerable tanto la velocidad como la capacidad de los dispositivos de almacenamiento, la utilización de diccionarios empezó a ser cada vez más común, aunque esto no eliminaba la necesidad de incluir procesamiento morfológico en las aplicaciones de corrección automática de errores debido a la gran riqueza y variedad morfológica de no pocos lenguajes.

8.2 Corrección ortográfica

Para algunas aplicaciones, la detección de errores es suficiente, pero para otras no basta y es necesario ofrecer también mecanismos de corrección que además de advertir de la presencia de un error ofrezcan una lista de correcciones candidatas clasificadas de algún modo.

La investigación en corrección de errores abarca un amplio período de tiempo que podemos aproximar entre principios de los sesenta y la actualidad. En este intervalo de

tiempo se han desarrollado diversas técnicas de corrección, tanto de propósito general como específico.

El problema general se puede descomponer en tres tareas: detección del error, generación de las correcciones candidatas y por último la clasificación de las mismas.

Se han desarrollado un buen número de técnicas diferentes. De entre ellas, las más estudiadas son las que se basan en el cálculo de la *distancia de edición* [171, 36, 92] entre la palabra a comprobar y una entrada del diccionario que supuestamente contiene las posibles correcciones.

8.2.1 Distancia de edición mínima

En 1964, Damerau [36] presentaba un algoritmo de corrección ortográfica en el que tenía en cuenta el resultado de un estudio del que dedujo que el 80 % de los errores ortográficos generados por humanos consistían en uno de los siguientes: inserción, borrado o sustitución de una letra o bien transposición de dos letras contiguas. Levenshtein [92], casi al mismo tiempo, desarrollaba un algoritmo para corregir eliminaciones, inserciones y transposiciones. A partir de esta clasificación, Wagner [171] introdujo el concepto de *distancia de edición mínima*, que consistía en el número mínimo de operaciones de edición descritas por Damerau necesarias para transformar una cadena de texto en otra.

$$
\begin{array}{lcl}
\mathrm{ed}(x_{i+1}, y_{j+1}) & = & \begin{cases}
\mathrm{ed}(x_i, y_j) & \text{si } x_{i+1} = y_{j+1} \\
1 + \min\{ \ \mathrm{ed}(x_{i-1}, y_{j-1}), & \\
\qquad \mathrm{ed}(x_{i+1}, y_j), & \\
\qquad \mathrm{ed}(x_i, y_{j+1})\} & \text{si } x_i = y_{j+1}, x_{i+1} = y_j \\
1 + \min\{ \ \mathrm{ed}(x_i, y_j), & \\
\qquad \mathrm{ed}(x_{i+1}, y_j), & \\
\qquad \mathrm{ed}(x_i, y_{j+1})\} & \text{en otro caso}
\end{cases} \\
\mathrm{ed}(x_0, y_j) & = & j \qquad 1 \leq j \leq n \\
\mathrm{ed}(x_i, y_0) & = & i \qquad 1 \leq i \leq m
\end{array}
\tag{8.1}
$$

La mayoría de los algoritmos basados en la *distancia de edición mínima* [36, 92, 172, 173] calculaban ésta a partir de valores de distancia enteros. Sin embargo, algunos iban más allá asignando valores de distancia no enteros en base a alguna particularidad. Así, era frecuente realizar una escala graduada de distancias teniendo en cuenta la proximidad de las teclas en el teclado o bien, para el caso de lenguajes como el inglés, la similaridad fonética. En 1988, Veronis [157] desarrollaba un algoritmo de programación dinámica que ponderaba las distancias de edición basándose en este tipo de similaridad y lo justificaba con la gran desviación de la ortografía correcta que ocasionan los errores de origen fonético.

Otros trabajos inspirados en distancias introducían la *distancia de edición mínima inversa* [56] que consistía en generar primero, a partir de una palabra con errores ortográficos, todas las permutaciones posibles de distancia 1, comprobando luego cada una de ellas contra el diccionario en busca de correcciones candidatas. Esto suponía que para una cadena errónea de longitud n y un alfabeto de 26 símbolos, el número de cadenas

a comprobar contra el diccionario serían $26 * (n + 1)$ inserciones más n eliminaciones, más $25 * n$ substituciones, más $n - 1$ transposiciones; o lo que es lo mismo $53 * n + 25$ cadenas, y esto suponiendo un solo error en la palabra inicial.

Wagner [171] fue pionero en la aplicación de técnicas de programación dinámica al problema de la corrección automática de errores para incrementar la eficiencia computacional.

Algoritmo 3 Cálculo de la distancia de edición utilizando programación dinámica.

```
1:  procedimiento DISTANCIA-EDICION(v_{1..n}, w_{1..m})
2:      para i ← 0, n hacer
3:          d_{i,0} ← i                          ▷ inicialización de la fila 0 de la matriz d
4:      fin para

5:      para j ← 0, m hacer
6:          d_{0,j} ← i                          ▷ inicialización de la columna 0 de la matriz d
7:      fin para

8:      para i ← 0, n hacer
9:          para j ← 0, m hacer

10:             si v_i = w_j entonces
11:                 cost ← 0
12:             sino
13:                 cost ← 1
14:             fin si

15:             d_{i,j} ← MINIMO(d_{i-1,j} + 1, d_{i,j-1} + 1, d_{i-1,j-1} + cost)
                                                 ▷ Borrado, inserción y substitución
16:             si i > 1 y j > 1 y v_i = w_{j-1} y v_{i-1} = w_j entonces
17:                 d_{i,j} ← MINIMO(d_{i,j}, d_{i-2,j-2} + cost)        ▷ Trasposición
18:             fin si

19:         fin para
20:     fin para

21:     retornar d_{n,m}

22: fin procedimiento
```

Con el fin de mejorar el tiempo de búsqueda en los algoritmos de *distancia de edición mínima*, se buscó una alternativa a la programación dinámica en lo que Fredkin [49] había denominado *tries* por formar parte del vocablo inglés *retrieval*[4]. Un *trie* es en esencia un árbol m-ario, cuyos nodos son vectores de m componentes correspondientes

[4]La traducción de *retrieval* al español sería *recuperación.*

a dígitos o caracteres. Cada nodo del nivel l representa el conjunto de todas las palabras que empiezan por la secuencia de l caracteres que nos conducen a ese nodo. Se desarrollaron técnicas [42, 19] en las que el diccionario se almacenaba en un árbol de caracteres pseudo-binario. La búsqueda implicaba examinar un subconjunto de las ramas del árbol a la vez que se hacían suposiciones de errores de inserción, borrado, sustitución y transposición cuando eran necesarias hasta alcanzar las correcciones candidatas en las hojas.

Las técnicas basadas en la *distancia de edición mínima* se pueden aplicar a cualquier tarea de corrección ortográfica, incluyendo edición de texto, interfaces en lenguaje de comandos, interfaces en lenguaje natural, etc.

8.2.2 Otras técnicas

Aunque, como ya se ha comentado, las técnicas de corrección de errores más utilizadas han sido las basadas en la distancia de edición, no parece aconsejable obviar el trabajo realizado por otros autores en base a otro tipo de estrategias. En particular, debemos hacer mención a técnicas basadas en similaridad de claves, en reglas, en n-gramas, en redes neuronales y otras de tipo probabilístico.

Las técnicas basadas en similaridad de claves consisten en generar, para cada palabra, una clave a partir de las letras que la componen. De esta forma lo que se comparan son las primeras, que suelen ser más cortas, en lugar de las palabras. Un ejemplo es SPEEDCOP, un sistema desarrollado por Pollock y Zamora [123]. Este método ordena el lexicón por la clave, de forma que palabras cercanas estarán también cercanas en el diccionario. Usa dos claves: la de esqueleto y la de omisión. La de esqueleto está formada por su primera letra seguida del resto de consonantes únicas en orden de aparición y, por último, las vocales únicas en orden de aparición también. En la de omisión las consonantes van ordenadas en orden inverso según la frecuencia con que suelen ser omitidas y que según los estudios de Pollock y Zamora es RSTNLCHDPGMFBYWVZXQKJ[5]. Después van las vocales, también en orden de aparición. Así, por ejemplo, la clave de esqueleto para la palabra CHEMICAL sería CHMLEIA mientras que la de omisión sería MHCLEIA, que coinciden con las que obtendríamos para las palabras erróneas CHEMICIAL y CHEMCIAL.

El algoritmo de SPEEDCOP consiste en cuatro pasos cada uno de los cuales se aplica sólo si falla el anterior. Primero consulta un diccionario de errores comunes, luego aplica la clave de esqueleto, después la de omisión y por último aplica una función para comprobar concatenaciones de palabras.

Las técnicas basadas en reglas son algoritmos o programas heurísticos que intentan representar los patrones de error comunes en forma de reglas que transformen las palabras erróneas en correctas. El proceso de generación de candidatos es sencillo y consiste en ir aplicando todas las reglas posibles y capturando aquellas palabras generadas que estén en el diccionario. La clasificación de los resultados suele realizarse asignando pesos a las reglas y computando luego los pesos de las aplicadas para la obtención de cada candidato. Yannakoudakis y Fawthrop [176, 177] desarrollaron un programa de corrección ortográfica basado en un conjunto de reglas que ellos mismos infirieron a partir de un conjunto de 1.377 palabras erróneas. Algunas de estas reglas incorporaban conocimiento acerca de la

[5] Lo que significa que es más frecuente omitir una R que una J aunque esto puede variar según el idioma.

longitud que debería tener la palabra correcta por lo que particionaron el diccionario en subconjuntos basándose en la longitud de las palabras y en la primera letra de las mismas.

Las técnicas de corrección ortográfica basadas en n-gramas son, después de las basadas en la distancia de edición, de las más utilizadas. Los n-gramas se han venido utilizando para diversos fines, tanto como claves de acceso que permitiesen buscar correcciones candidatas en un diccionario, como para calcular medidas de similaridad entre las palabras erróneas y las del diccionario [8]. Se ha utilizado también para capturar la sintaxis léxica del diccionario y proporcionar correcciones en aplicaciones de ROC [127]. Otras técnicas basadas en n-gramas consistían en representar tanto las palabras del diccionario como las incorrectas mediante vectores de rasgos léxicos a los que se les podían aplicar medidas de distancia entre vectores para clasificar las correcciones candidatas [86] .

A partir de las propuestas basadas en n-gramas derivaron de forma natural las otras probabilísticas, tanto en el paradigma de reconocimiento de textos como en el de corrección ortográfica. Podemos hablar de dos tipos de probabilidad: la de transición y la de confusión. La primera representa la probabilidad de que una letra, o secuencia de letras, vaya seguida de otra letra dada. Este tipo de probabilidad es dependiente del lenguaje y es referida a veces como *probabilidad de Markov* [98]. Puede ser estimada a partir de las frecuencias estadísticas de los n-gramas en grandes *corpora* de texto del dominio del discurso. La probabilidad de confusión representa la frecuencia con que una letra dada se confunde o es substituida por otra letra dada. Esta depende del origen del texto. Cada dispositivo ROC tendrá su propia distribución de probabilidad de confusión. Los pioneros en este caso fueron Bledsoe y Browning [18]

Las redes neuronales son también aplicadas en correctores ortográficos debido a su capacidad inherente para realizar recubrimientos asociativos basados en entradas incompletas o distorsionadas. Pueden adaptarse progresivamente a los patrones de error de una comunidad de usuarios de forma que se maximice la precisión para ese grupo. El algoritmo de entrenamiento de redes neuronales más utilizado es el de *retro-propagación*[6] [132]. Una red de *retro-propagación* típica consiste en tres capas de nodos: una capa de entrada, una intermedia u oculta y otra de salida. Cada nodo de la capa de entrada se conecta mediante un enlace etiquetado con un peso a cada uno de los de la capa intermedia y, de la misma forma, los de la capa intermedia con los de la de salida. Un ejemplo de utilización de redes neuronales para la corrección ortográfica es el de Karen Kukich [85] que utilizó un conjunto de 183 apellidos para entrenar una red de retro-propagación en la que la capa de salida consistía en 183 nodos, uno por cada nombre.

[6] Usualmente conocido por el término en inglés *back-propagation*.

Capítulo 9

Algoritmos de corrección ortográfica sobre AFs

En el capítulo 8 hemos visto distintas técnicas de detección y corrección ortográfica. Nos centraremos ahora en aquellas que hacen uso de diccionarios. En estos casos, adquiere una especial importancia la modelización del diccionario ya que de ella dependerá la eficiencia del algoritmo. En este sentido, hemos introducido en la sección 4.1 una implementación eficiente y compacta de un lexicón basada en AFs.

Los AFs se convierten, por tanto, en una importante herramienta a la hora de modelar nuestra propuesta. Estudiaremos, en concreto, dos técnicas de corrección global sobre AFs: el algoritmo de Oflazer [115, 116] y el algoritmo de Savary [140].

Si centramos nuestra atención sobre los algoritmos de corrección ortográfica basados en diccionario, más concretamente en aquéllos que implementan el diccionario como un AF, nos daremos cuenta de que estamos ante un problema de búsqueda de caminos en un grafo. En efecto, comprobar si una palabra está o no en el diccionario consistirá en comprobar si hay una ruta en el autómata, etiquetada con las letras que componen la palabra, que nos lleve desde el estado inicial hasta un estado final. En caso de que la palabra no esté en el diccionario, ofrecer correcciones para la misma consistirá en encontrar palabras alternativas, en este caso rutas, que nos permitan alcanzar estados finales y que además compartan con la palabra errónea el mayor número de caracteres, presentando además la misma disposición.

En cualquier problema de búsqueda hay un factor que adquiere una especial relevancia y es el ámbito. No es lo mismo buscar una aguja en un pajar que en un alfiletero, y la principal diferencia está precisamente en el ámbito de la búsqueda. De igual modo, tampoco es lo mismo buscar correcciones candidatas para una palabra errónea en un diccionario de varios cientos de miles de entradas, que hacerlo, por ejemplo, entre aquéllas que empiecen por un determinado prefijo. En este sentido los algoritmos de corrección global realizan una búsqueda exhaustiva a lo largo y ancho de todo el diccionario lo que garantiza la localización de todas y cada una de las correcciones posibles. Pero al igual que en tantas otras ocasiones la contrapartida a la eficacia es la eficiencia y el coste de

una aproximación global suele ser elevado y, a menudo, innecesario.

Por otro lado, si nos centramos en la estrategia de búsqueda empleada, podemos referirnos a los conocidos algoritmos de búsqueda en espacios de estados [133], como son el de búsqueda primero en anchura, el de búsqueda primero en profundidad o el algoritmo A* [67].

A continuación se presentan dos algoritmos de corrección ortográfica que podemos clasificar dentro de los métodos globales, y que implementan una estrategia de búsqueda primero en profundidad con el objetivo de encontrar cuanto antes una corrección.

9.1 Algoritmo de Oflazer

El algoritmo de Oflazer [115, 116] permite el reconocimiento de cadenas sobre AFs con tolerancia a errores. Este método acepta cadenas que se desvían ligeramente de alguna de las del conjunto regular reconocido por el AF. Esa ligera desviación ha de ser medida de alguna forma, y para ello Oflazer utiliza la distancia de edición entre cadenas. Más formalmente, dado el lenguaje $\mathcal{L}$ aceptado por el AF $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ se dice que la cadena v se reconoce con un umbral de error de $t > 0$, si se cumple que:

$$\{u \ / \ u \in \mathcal{L}, \mathrm{ed}\,(u, v) \leq t\} \neq \emptyset$$

Se trata pues de encontrar todos aquellos caminos desde el estado inicial hasta algún estado final de forma que, al concatenar las etiquetas de las transiciones utilizadas en cada camino, las cadenas resultantes se encuentren a una distancia de edición de la cadena de entrada errónea inferior a un umbral dado. Durante este proceso de búsqueda es necesario asegurar que en ningún momento la cadena candidata que se está generando se desvía de ciertos prefijos de la errónea más que el umbral permitido. Con el fin de detectar los casos en que esto se produce Oflazer define el concepto de *distancia de edición de corte* que mide la distancia de edición mínima entre prefijos de la cadena de entrada incorrecta y la cadena candidata[1]. De este modo, si tenemos una cadena errónea $u_{1..m}$ y estamos generando una cadena candidata $v_{1..n}$, la *distancia de edición de corte*, que denotaremos $edcorte\,(u, v)$, se define como:

$$edcorte\,(u, v) = \min_{i \leq j \leq k} ed\,(u_{1:j}, v)$$

donde $i = max\,(1, n - t)$ y $k = min\,(m, n + t)$.

Ejemplo 9.1. *Dada la palabra errónea* reprtero *y suponiendo que la palabra candidata parcial que hemos generado hasta el momento sea* repo *y que el umbral* $t = 2$*, en ese caso tendríamos que:*

[1] La cadena candidata puede ser parcial.

$$edcorte\,(reprtero, repo) = min\{ \quad \begin{array}{l} ed\,(re, repo) = 2 \\ ed\,(rep, repo) = 1 \\ ed\,(repr, repo) = 1 \\ ed\,(reprt, repo) = 2 \\ ed\,(reprte, repo) = 3 \end{array} \quad \} = 1$$

■

Se puede observar que, excepto en los extremos, la longitud de los prefijos de la cadena incorrecta a tener en cuenta nunca sobrepasarán en más de t la longitud de la cadena candidata y viceversa. Esto es lógico ya que cualquier prefijo que no cumpliese los requisitos anteriores necesitaría más de t borrados o inserciones respectivamente, lo cual violaría la restricción de distancia de edición impuesta por t.

Las cadenas candidatas parciales para una cadena incorrecta se generan mediante concatenaciones sucesivas de etiquetas de las transiciones, empezando en el estado inicial y realizando una exploración primero en profundidad del AF. Cada vez que se concatena un nuevo símbolo a la cadena candidata, se comprueba si la distancia de edición de corte de la misma con la cadena incorrecta se mantiene por debajo del umbral permitido.

En caso de que se sobrepase dicho umbral se retrocede hasta la última transición utilizada recortando al mismo tiempo el último símbolo de la cadena candidata y probando con otra transición diferente. Este retroceso se realiza de forma recursiva cuando la búsqueda no puede continuar desde un estado. En caso de que, durante este proceso, se alcance un estado final del AF y la distancia de edición entre la cadena errónea completa y la candidata esté por debajo del umbral se dice que esta última es una corrección válida para la palabra incorrecta inicial.

Este algoritmo requiere continuos cálculos de distancias de edición que pueden resultar costosos. Es por esto, que Oflazer propone un algoritmo de programación dinámica para el cálculo de la misma, centrándose en el mantenimiento de una matriz $\mathcal{M}$ de dimensiones $m \times n$, donde cada elemento $\mathcal{M}\,(i, j)$ se calcula como sigue:

$$\mathcal{M}\,(i, j) = ed\,(u_{1:i}, v_{1:j})$$

siendo u y v las cadenas incorrecta y candidata, respectivamente. En esta matriz, el cálculo del elemento $\mathcal{M}\,(i+1, j+1)$ depende de manera recursiva únicamente del elemento $\mathcal{M}\,(i, j)$, $\mathcal{M}\,(i, j+1)$, $\mathcal{M}\,(i+1, j)$ y $\mathcal{M}\,(i-1, j-1)$, según la definición de la distancia de edición [36, 92] (véase ecuación 8.1). De este modo, durante la exploración primero en profundidad del AF, las entradas de la columna n de la matriz han de ser calculadas, sólo cuando la cadena candidata sea de longitud n. Cuando sea necesario retroceder las entradas de la última columna se descartarán, pero las entradas en las columnas anteriores seguirán siendo válidas.

Ejemplo 9.2. *En la figura 9.1 podemos ver el árbol de exploración correspondiente a la palabra incorrecta* w = **coharizo**. *Esta palabra no pertenece al lenguaje reconocido por el* AF *de la figura 2.6, pero para un umbral* t = 2 *existe una palabra para la que, perteneciendo a dicho lenguaje, la distancia de edición entre la misma y* w *está por debajo del umbral. Los números sobre cada estado indican la distancia de edición de corte cuando la búsqueda alcanza ese nodo. Con un doble círculo se representa el estado en el que se encuentra la*

palabra **chorizo**. *Nótese que al alcanzar un estado con una distancia de edición de corte superior al umbral establecido, en este caso 2, la búsqueda no continúa por esa rama.*

■

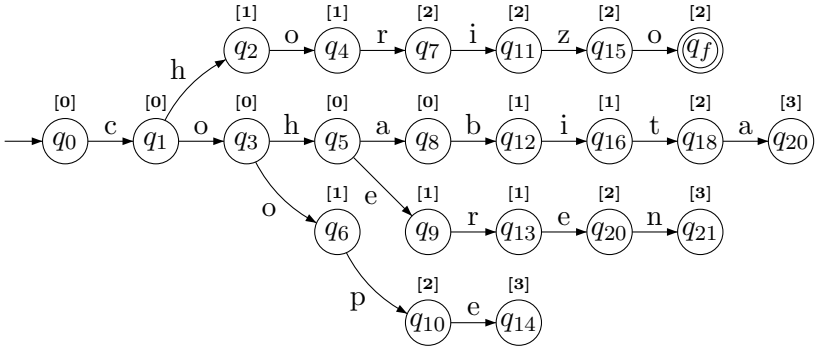

Figura 9.1: Árbol de exploración para umbral 2 correspondiente a la palabra errónea *coharizo* sobre el AF de la figura 2.6

9.2 Algoritmo de Savary

Savary [140] describe un método global de corrección ortográfica sobre AFs mediante el cual plantea el problema de una forma diferente a Oflazer [116]. Savary se centra en la búsqueda de los vecinos más cercanos[2], aunque la distancia mínima no debe superar tampoco un determinado umbral.

Dado que no existe un umbral de error teórico entre una palabra errónea y su corrección, se hace necesario llegar a un compromiso entre tres factores: la eficiencia en tiempo de búsqueda[3], el tamaño de la lista de correcciones candidatas y la probabilidad de que la palabra esperada esté en dicha lista. Así, Savary ofrece dos posibles perspectivas para afrontar el problema de la corrección ortográfica:

- Búsqueda de las palabras válidas cuya distancia de edición con la palabra de entrada no supere un determinado umbral.
- Búsqueda de las palabras válidas cuya distancia de edición con la palabra de entrada, además de no superar un determinado umbral, es mínima.

Teniendo en cuenta que ninguna de las dos definiciones anteriores garantiza que la corrección idónea estará en la lista de correcciones propuestas, Savary elige la segunda por tres razones:

[2]Las palabras válidas cuya distancia de edición con la palabra de entrada sea mínima.

[3]Tanto para el algoritmo de Savary [140] como para el de Oflazer [115, 116] se corresponde con el tamaño de la sección del AF que ha de ser explorado.

Algoritmo 4 Algoritmo de Savary

```
1: procedimiento BUSQUEDA-TOLERANTE(w_{1..n}, q, t)
2:     (ed, S) ← (∞, ∅)
3:     si n = 0 y TERMINAL(q) entonces retornar (0, {ε})
4:     si n > 0 y ∃q′ / q.w_1 = q′ entonces
5:         (ed, S) ← BUSQUEDA-TOLERANTE(w_{2:n}, q′, t)
6:         t ← min(t, ed)
7:         para cada suf ∈ S hacer suf ← w_1 ⊕ suf
8:     fin si
9:     si t > 0 entonces
10:        si n > 0 entonces
11:            (ed′, S′) ← BUSQUEDA-TOLERANTE(w_{2:n}, q, t − 1)
12:            (ed, S) ← AÑADIR-O-REEMPLAZAR((ed, S), (ed′ + 1, S′))
13:            t ← min(t, ed)
14:        fin si
15:        si n > 1 y w_1 ≠ w_2 y ∃q′ / q.w_2.w_1 = q′ entonces
16:            (ed′, S′) ← BUSQUEDA-TOLERANTE(w_{3:n}, q′, t − 1)
17:            para cada suf ∈ S′ suf ← w_2 ⊕ w_1 ⊕ suf
18:            (ed, S) ← AÑADIR-O-REEMPLAZAR((ed, S), (ed′ + 1, S′))
19:            t ← min(t, ed)
20:        fin si
21:        trans ← {(q′, c)} / q.c = q′
22:        para cada (q′, c) ∈ trans hacer
23:            (ed′, S′) ← BUSQUEDA-TOLERANTE(w_{1..n}, q′, t − 1)
24:            para cada suf ∈ S′ suf ← c ⊕ suf
25:            (ed, S) ← AÑADIR-O-REEMPLAZAR((ed, S), (ed′ + 1, S′))
26:            t ← min(t, ed)
27:            si n > 0 entonces
28:                (ed′, S′) ← BUSQUEDA-TOLERANTE(w_{2:n}, q′, t − 1)
29:                para cada suf ∈ S′ suf ← c ⊕ suf
30:                (ed, S) ← AÑADIR-O-REEMPLAZAR((ed, S), (ed′ + 1, S′))
31:                t ← min(t, ed)
32:            fin si
33:        fin para
34:    fin si
35:    retornar (ed, S)
36: fin procedimiento
```

- Los estudios estadísticos [126] demuestran que palabras con múltiples errores son poco probables (entre el 0,17 % y el 1,99 % de las palabras desconocidas en un *corpus*).
- Los usuarios se mostrarán desalentados si se les presenta una larga lista de alternativas. De hecho, hay aplicaciones para las que esta posibilidad se descarta de raíz, como es el caso de las desarrolladas en el ámbito de la RI.
- El tiempo de búsqueda crece exponencialmente con el umbral de distancia admitido.

En el algoritmo 4 se muestra una implementación recursiva del algoritmo de Savary donde $w_{1..n}$ es la palabra de entrada, q es el estado en que se encuentra actualmente el AF y t es el umbral de error permitido. En este contexto, la función BUSQUEDA-TOLERANTE trata de reconocer el sufijo $w_{1..n}$ partiendo del estado q y admitiendo t operaciones elementales de edición. La función devuelve un par (ed, S) donde S es el conjunto de sufijos, exactos o modificados, y ed es la distancia de edición entre el sufijo $w_{1..n}$ y cada uno de los sufijos contenidos en S. Si no hay sufijos en S, entonces $ed = \infty$, condición de inicialización en la línea 2. La primera llamada a la función se hace con la cadena completa de entrada, el estado inicial del AF y el umbral de distancia de edición deseado. Inicialmente se seguirá el procedimiento estándar de reconocimiento que consiste en ir avanzando por los estados del AF con cada carácter de la cadena de entrada, tal y como puede verse entre las líneas 4 y 8. Si la palabra w pertenece al lexicón, entonces en algún momento t se hará igual a cero en la línea 6 de forma que al llegar a la línea 9 se evitará la ejecución del código que introduce las hipótesis de error entre las líneas 9 y 34. Sin embargo, en el caso de que la palabra de entrada sea errónea, entonces el procedimiento estándar de reconocimiento acabará fallando por uno de los dos motivos siguientes: la cadena de entrada se ha agotado y no hemos alcanzado un estado final, o bien se ha alcanzado un estado desde el que no es posible transitar con el carácter actual. En ambos casos alcanzaríamos la línea 9 con $t > 0$, lo que nos revelaría que existe un error en la posición actual de la cadena de entrada y trataríamos de aplicar en este punto alguna de las cuatro hipótesis elementales de error:

- Inserción de la letra actual (líneas 10 a 14), en cuyo caso omitiríamos esa letra y trataríamos de reconocer el resto de la palabra de entrada ($w_{2..n}$) partiendo del estado actual.
- Inversión de la letra actual y la siguiente (líneas 15 a 20). Para que esto ocurra es necesario que nos resten al menos 2 caracteres por reconocer y deberá existir un estado $q\prime$ alcanzable desde el estado actual, utilizando para ello los dos caracteres siguientes invertidos ($\exists q\prime \ / \ q.w_2.w_1 = q\prime$). En ese caso, trataríamos de reconocer el sufijo restante ($w_{3..n}$) partiendo del estado $q\prime$.
- Omisión de una letra en la posición actual (líneas 23 a 26) en cuyo caso para cada transición que nos lleve desde el estado actual q a otro estado $q\prime$ por medio del carácter c (línea 22), trataremos de reconocer el sufijo actual, pero no desde el estado actual q sino desde cada uno de los $q\prime$, añadiendo luego a cada una de las soluciones encontradas el carácter c correspondiente (línea 24).

- Reemplazamiento de la letra correcta por la letra actual (líneas 27 a 32) en cuyo caso para cada transición que nos lleve desde el estado actual q a otro estado $q\prime$ por medio del carácter c, trataríamos de reconocer el resto de la palabra de entrada ($w_{2..n}$) partiendo de cada uno de los estados $q\prime$ y añadiendo luego a cada una de las soluciones encontradas el carácter c correspondiente (línea 29).

Cada vez que se encuentran nuevas soluciones el valor de ed y el contenido de S se actualizan por medio de la función AÑADIR-O-REEMPLAZAR (líneas 12, 18, 25 y 30). Esta función comprueba si las nuevas soluciones son más cercanas a la palabra original que las soluciones que ya se encuentran en S, en cuyo caso S es reemplazado por el conjunto de nuevas soluciones, y el valor de ed pasa a ser la distancia de edición que existe entre la cadena de entrada y cada una de las nuevas soluciones. En caso contrario, se realiza la unión de los dos conjuntos y ed no se modifica. De este modo, sólo se retienen en S las soluciones con la menor distancia de edición respecto al sufijo original de forma que t podrá verse reducido (líneas 6, 13, 19, 26 y 31), limitando así el rango de las búsquedas a partir de ese momento.

La principal diferencia entre el algoritmo de Savary y el de Oflazer está en la forma de calcular la *distancia de error*[4]. Mientras que Oflazer utiliza técnicas de programación dinámica para el cálculo de la misma, en la propuesta de Savary el cálculo de la distancia de edición está embebido en el algoritmo: cada vez que supone una modificación, la distancia de error se incrementa. Según Savary, esto le permite evitar el cálculo de la costosa matriz, pero tiene también una serie de desventajas que se describen a continuación:

- Es difícil adaptar el cálculo de la distancia de error a lenguajes o aplicaciones particulares. Por ejemplo, considerar intercambios de ciertas letras o grupos de letras motivados fonéticamente.

- Se puede alcanzar una misma corrección candidata varias veces con distintos valores de distancia de error. Esto deriva en la exploración de la misma ruta varias veces, lo cual no resulta eficiente desde el punto de vista computacional para valores altos del umbral de distancia de edición.

Podemos concluir que el algoritmo de Savary resulta más adecuado para aquellas aplicaciones en que sólo se requieran las correcciones de distancia mínima y en las que resulte interesante obtener una primera alternativa cuanto antes. Por su parte, el de Oflazer es más simple y elegante, en el sentido de que el cálculo de la distancia de edición es independiente del método de búsqueda.

[4]La distancia de error es igual que la distancia de edición con la salvedad de que las operaciones de edición se realizan de forma lineal, es decir, una operación posterior no puede modificar el resultado de una operación anterior. Además, para el cálculo de la distancia de error, todas las operaciones de edición tienen el mismo coste.

Capítulo 10

Corrección ortográfica regional

Como hemos visto en el capítulo 9, la corrección ortográfica sobre AFs es un problema de búsqueda de caminos en un grafo. Hemos visto también dos ejemplos de algoritmos de corrección ortográfica que aplican una exploración exhaustiva sobre el AF que implementa el diccionario. Ambos realizan una búsqueda primero en profundidad, sin utilizar ninguna otra información distinta de la del coste de las reparaciones a realizar para encontrar las correcciones candidatas. Haciendo una revisión de los métodos de navegación por árboles, el algoritmo A* [67] propone utilizar, además de la información del coste de las rutas, una heurística que nos permita predecir lo cerca que estamos del objetivo en cada uno de los caminos abiertos en cada momento.

En este capítulo presentamos nuestro algoritmo de corrección ortográfica regional [162] que aplica heurísticas para reducir el coste computacional de los algoritmos globales y tratar, al mismo tiempo, de mejorar la calidad de la respuesta obtenida, reduciendo para ello el número de correcciones candidatas.

10.1 Definición del modelo operacional

El proceso de análisis de una palabra del lenguaje reconocido por un AF consiste en ir transitando entre estados eligiendo, en cada salto de estado, el movimiento que se corresponde con el siguiente carácter en la cadena de entrada. Por tanto, el sistema irá evolucionando de manera que en cada paso avanzará un carácter en la palabra a reconocer, a la vez que cambia de estado en el AF. De este modo, podemos saber en cualquier momento en qué situación se encuentra el proceso de reconocimiento con sólo indicar la posición de la cadena de entrada y el estado del AF.

Necesitamos por tanto una estructura que nos facilite la representación de cada uno de los pasos de análisis ejecutados, lo que nos permitirá aplicar técnicas de programación dinámica que eviten la exploración de caminos ya visitados previamente.

Definición 10.1. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $w_{1..n} \in \Sigma^*$ *una palabra. Se define un* ítem *como un par* $[q, i]$ / $q \in \mathcal{Q},\ i \in [1, n+1]$. □

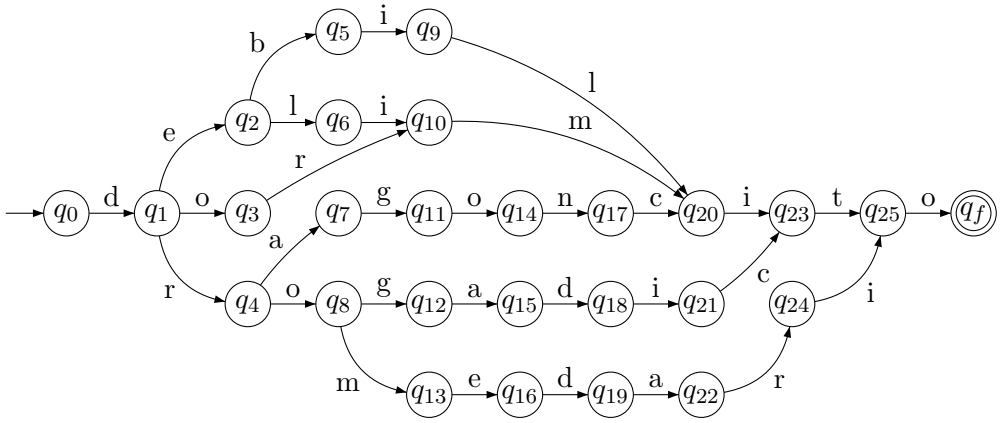

Figura 10.1: AF para: *debilito, delimito, dormito, dragoncito, drogadicto y dromedario.*

Intuitivamente, podemos pensar que un ítem $[q, i]$ describe una situación concreta en el proceso de reconocimiento de una palabra que nos indicaría que nos encontramos en el estado q del AF y en la posición i de la cadena de entrada. Sin embargo, con la definición 10.1, dada la palabra *"dormito"* y el AF de la figura 10.1, $[q_4, 2]$ sería un ítem aunque en el proceso de reconocimiento de esa palabra, jamás alcanzaríamos el estado q_4. Es por esto que necesitamos un procedimiento que nos permita saber qué ítems pueden ser generados durante el proceso de análisis de una cadena y cuáles no. Establecemos en este punto un marco formal para describir este proceso utilizando el esquema de análisis sintáctico[1] descrito por Sikkel [143].

Definición 10.2. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un* AF *y sea $w_{1..n} \in \Sigma^*$ una palabra. Definimos el* esquema de análisis sintáctico *como una tripla $\langle \mathcal{I},\ \mathcal{H},\ \mathcal{D} \rangle$ donde:*

- $\mathcal{I} = \{[q, i], q \in \mathcal{Q}, i \in [1, n+1]\}$*, donde $[q, i]$ es un ítem, es el* conjunto de ítems *que hemos generado hasta el momento. Inicialmente $\mathcal{I} = \emptyset$.*
- $\mathcal{H} = \{[a, i], a = w_i\}$ *es un* conjunto finito de hipótesis *que codifican la palabra que se quiere reconocer.*
- $\mathcal{D} = \{\eta_1, \ldots, \eta_k \vdash \xi \ /\ conds\}$ *es un conjunto finito de* reglas de deducción *que permiten obtener nuevos ítems a partir de los ya existentes, y cuya interpretación es que si todos los antecedentes η_i están presentes y las condiciones* conds *se cumplen, entonces se generan los consecuentes ξ. En nuestro caso tendríamos que $\mathcal{D} = \mathcal{D}^{Inicial} \cup \mathcal{D}^{Salto}$, donde:*

$$\mathcal{D}^{Inicial} = \{\vdash [q_0, 1]\}$$

$$\mathcal{D}^{Salto} = \{[p, i] \vdash [q, i+1] \ /\ \exists\, [a, i] \in \mathcal{H}, q = p.a\}$$

[1] En terminología anglosajona *Parsing Schemata.*

□

El sistema aplicará estas reglas de deducción mientras sea posible y la palabra será reconocida si se alcanza un ítem final $[q_f, n+1], q_f \in \mathcal{Q}_f$.

Ejemplo 10.1. *Ilustraremos el funcionamiento del sistema mostrando cuál sería el proceso de reconocimiento de la palabra* "dormito" *sobre el* AF *de la figura 10.1. Inicialmente, tendríamos que por definición* $\mathcal{H} = \{[d,1],\ [o,2],\ [r,3],\ [m,4],\ [i,5],\ [t,6],\ [o,7]\}$ *e* $\mathcal{I} = \emptyset$ *y comenzaríamos aplicando la regla de deducción* $\mathcal{D}^{Inicial}$ *que, al estar el conjunto de ítems vacío, es la única aplicable. A partir de aquí comenzamos a aplicar la regla* $\mathcal{D}^{Salto}$*, que es la que nos permite, mientras sea posible, obtener nuevos ítems. La evolución del conjunto* $\mathcal{I}$ *de ítems sería la siguiente:*

1. $\mathcal{I} = \emptyset$
2. $\vdash [q_0, 1]$ $\Rightarrow \mathcal{I} = \{[q_0,1]\}$
3. $[q_0,1] \vdash [q_1,2]\ /\ \exists\, [d,1] \in \mathcal{H}, q_1 = q_0.d$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2]\}$
4. $[q_1,2] \vdash [q_3,3]\ /\ \exists\, [o,2] \in \mathcal{H}, q_3 = q_1.o$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3]\}$
5. $[q_3,3] \vdash [q_{10},4]\ /\ \exists\, [r,3] \in \mathcal{H}, q_{10} = q_3.r$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3], [q_{10},4]\}$
6. $[q_{10},4] \vdash [q_{20},5]\ /\ \exists\, [m,4] \in \mathcal{H}, q_{20} = q_{10}.m$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3], [q_{10},4], [q_{20},5]\}$
7. $[q_{20},5] \vdash [q_{23},6]\ /\ \exists\, [i,5] \in \mathcal{H}, q_{23} = q_{20}.i$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3], [q_{10},4], [q_{20},5],\ [q_{23},6]\}$
8. $[q_{23},6] \vdash [q_{25},7]\ /\ \exists\, [t,6] \in \mathcal{H}, q_{25} = q_{23}.t$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3], [q_{10},4], [q_{20},5],\ [q_{23},6],\ [q_{25},7]\}$
9. $[q_{25},7] \vdash [q_f,8]\ /\ \exists\, [o,7] \in \mathcal{H}, q_f = q_{25}.o$ $\Rightarrow \mathcal{I} = \{[q_0,1],\ [q_1,2],\ [q_3,3], [q_{10},4], [q_{20},5],\ [q_{23},6]\ [q_{25},7]\ [q_f,8]\}$

Dado que $[q_f, 8]$ *es un ítem final, la palabra sería reconocida.* ■

Sin embargo, cuando la palabra a reconocer presenta variaciones que hacen que ésta deje de pertenecer al lenguaje reconocido por el AF, inevitablemente alcanzaremos un punto en el que no es posible continuar, es decir, $\nexists\, [q,i] \in \mathcal{I},\ [a,i] \in \mathcal{H},\ p \in \mathcal{Q} \mid p = q.a$. Esta situación provoca que el proceso se pare y decimos que hemos alcanzado un *ítem de error.*

Definición 10.3. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF, $\langle \mathcal{I},\ \mathcal{H},\ \mathcal{D} \rangle$ *un esquema de análisis sintáctico, y sea* $w_{1..n} \in \Sigma^*$ *una palabra. Decimos que* $[q,i]$ *es un* ítem de error *sii:*

$$\nexists [a,i] \in \mathcal{H},\ p \in \mathcal{Q} \mid p = q.a$$

Diremos también que w_i *es un* punto de error. □

El ítem de error será, por tanto, establecido por el reconocedor.

Ejemplo 10.2. *Si intentamos reconocer la palabra* "delito" *sobre el* AF *de la figura 10.1, el sistema alcanzaría un punto en el que el conjunto de ítems sería* $\mathcal{I} = \{\ [q_0,1],\ [q_1,2],\ [q_2,3],\ [q_6,4],\ [q_{10},5]\ \}$ *y el de hipótesis sería* $\mathcal{H} =$

$\{\ [d,1],\ [e,2],\ [l,3],\ [i,4],\ [t,5],\ [o,6]\ \}$. *En esta situación, el sistema se encuentra en el estado* q_{10} *y trata de reconocer el carácter 't', lo que no es posible. Diremos pues que* $[q_{10},5]$ *es un* ítem de error. ■

Cuando alcanzamos un ítem de error, es posible que la aplicación de algunas operaciones de reparación en ese punto permitan al analizador continuar, y acabar ofreciendo una o varias alternativas de corrección local. Sin embargo, el rendimiento ofrecido por esta técnica no resulta aceptable para la mayoría de las aplicaciones en las que se requiere corrección automática de errores. Como alternativa tenemos los algoritmos de corrección global que dedican el mismo esfuerzo computacional en todos los puntos de la cadena, independientemente de dónde se haya detectado el error. Esto no parece muy lógico ya que si el analizador ha sido capaz de reconocer un determinado prefijo, parece acertado pensar que el error que acabamos de encontrar tenga su origen cerca del lugar en que lo hemos detectado y no varios caracteres atrás.

De este modo, centrándonos en el proceso de análisis de una palabra sobre un AF, podemos intuir que si alcanzamos un punto en el que el analizador se para será porque:

- Se ha producido un error en el punto de la cadena en que nos encontramos. En este caso una reparación local nos permitiría continuar con el proceso de análisis.

- Se ha producido un error en un punto anterior que ha provocado que nos adentrásemos en un camino del AF hasta que no hemos podido continuar. En este caso la topología del AF será determinante ya que si un error nos ha abocado a un camino incorrecto, forzosamente habrá sido en un estado en el que había una bifurcación. Esto nos da una pista para elegir el punto hasta el que retrocederemos intentando evitar una búsqueda global.

Necesitamos, por tanto, habilitar un mecanismo que nos permita establecer el punto hasta el cual retrocederemos en el proceso de análisis cada vez que detectemos un error, limitando el impacto sobre el prefijo ya reconocido. Para esto introduciremos algunas propiedades topológicas sobre AFs.

Definición 10.4. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF*, y sean* $p,\ q \in \mathcal{Q}$*. Decimos que* p es menor que q *sii existe un camino* $\{p, \ldots, q\}$*, y lo denotamos como* $p < q$*.* □

En el AF de la figura 10.1 tenemos que, por ejemplo, $q_1 < q_2$, $q_4 < q_{11}$ y $\forall i \in \{0 \ldots 25\}, q_i < q_f$. Este orden es inducido por el formalismo transicional, lo que resulta en una relación bien definida ya que el AF es acíclico. De esta forma es posible dar una dirección a los caminos.

Definición 10.5. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF*, decimos que* $q_o \in \mathcal{Q}$ *es un* estado origen *para algún camino* $\rho = \{q_1, \ldots, q_m\}$ *de* $\mathcal{A}$*, sii* $\exists\ a \in \Sigma\ /\ q_1 = q_o.a$*.* □

Definición 10.6. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF*, decimos que* $q_d \in \mathcal{Q}$ *es un* estado destino *para algún camino* $\rho = \{q_1, \ldots, q_m\}$ *de* $\mathcal{A}$*, sii* $\exists\ a \in \Sigma\ /\ q_m.a = q_d$*.* □

De forma intuitiva, hablamos de estados origen o destino en las transiciones salientes o entrantes respectivamente, las cuales orientan los caminos desde un origen a un destino. Así, para el AF de la figura 10.1 tenemos que en los caminos $\{q_7, q_{11}, q_{14}, q_{17}, q_{20}, q_{23}\}$, $\{q_8, q_{12}, q_{15}, q_{18}, q_{21}, q_{23}\}$, $\{q_8, q_{13}, q_{16}, q_{19}, q_{22}, q_{24}\}$, q_4 es el estado origen y q_{25} es el estado destino. En este punto, podemos introducir el concepto de *región*.

Definición 10.7. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF*, decimos que un par* (q_o, q_d), $q_o, q_d \in \mathcal{Q}$ *es una* región *en* $\mathcal{A}$*, denotada como* $\mathcal{R}_{q_o}^{q_d}(\mathcal{A})$*, sii verifica que:*

(1) $q_o = q_0$ **y** $q_d = q_f$ *(el* AF *global)*
o bien
(2) $\{\forall \rho,\ origen(\rho) = q_o\} \Rightarrow destino(\rho) = q_d$ **y** $\mid \{\forall \rho,\ origen(\rho) = q_o\} \mid > 1$

□

Cuando el contexto esté suficientemente claro, podremos omitir el AF al que pertenece la región denotando ésta como $\mathcal{R}_{q_o}^{q_d}$. También denotamos los caminos existentes en el interior de una región como $caminos(\mathcal{R}_{q_o}^{q_d}) = \{\rho \ / \ origen(\rho) = q_o,\ destino(\rho) = q_d\}$, y dado $q \in \mathcal{Q}$, decimos que $q \in \mathcal{R}_{q_o}^{q_d}$ sii $\exists\ \rho \in\ caminos(\mathcal{R}_{q_o}^{q_d}) \ / \ q \in \rho$.

De forma gráfica podemos decir que cada estado $q \in \mathcal{Q}$ del que parta más de una transición de salida[2] será el estado origen de una región. El estado destino será aquél en el que converjan todos los caminos que parten de q. Con el fin de garantizar que todos los caminos del AF convergerán en algún estado, debemos garantizar la existencia de un único estado final. Para esto, haremos una breve concesión a la minimalidad del AF insertando un nuevo estado q_f y añadiendo transiciones nulas desde cada uno de los estados finales a q_f [71], tal y como se ilustra en la figura 10.2.

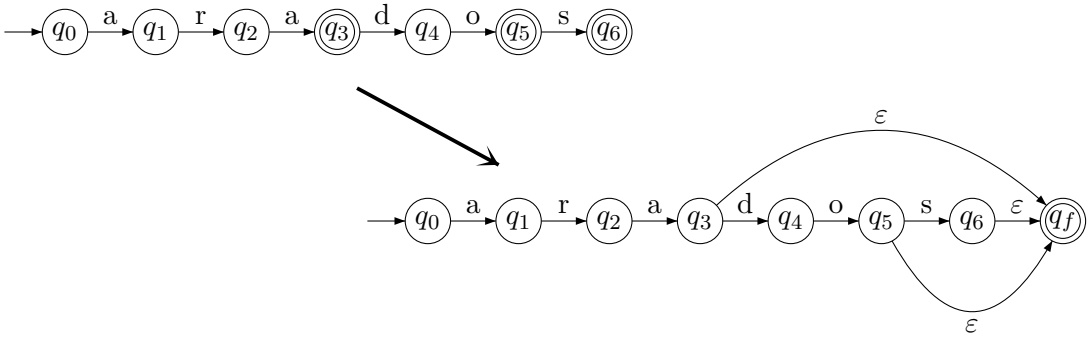

Figura 10.2: Transformación de un AF con varios estados finales.

Si tomamos de nuevo el AF de la figura 10.1 como ejemplo, podemos identificar las regiones siguientes:

[2] Es decir, cuando hay una bifurcación.

$$
\begin{array}{lcl}
\mathcal{R}_{q_0}^{q_f} & = & \mathcal{Q} \setminus \{q_0, q_f\} \\
\mathcal{R}_{q_1}^{q_{25}} & = & \mathcal{Q} \setminus \{q_0, q_1, q_{25}, q_f\} \\
\mathcal{R}_{q_2}^{q_{20}} & = & \{q_5, q_6, q_9, q_{10}\} \\
\mathcal{R}_{q_4}^{q_{25}} & = & \{q_7, q_8\} \cup \{q_{11}, \ldots, q_{24}\} \\
\mathcal{R}_{q_8}^{q_{25}} & = & \{q_{12}, q_{13}, q_{15}, q_{16}, q_{18}, q_{19}, q_{21}, q_{22}, q_{23}, q_{24}\}
\end{array}
$$

Por otra parte, podemos decir que:

$$
caminos(\mathcal{R}_{q_4}^{q_{25}}) \quad = \quad \left\{ \begin{array}{l} \{q_7, q_{11}, q_{14}, q_{17}, q_{20}, q_{23}\} \\ \{q_8, q_{12}, q_{15}, q_{18}, q_{21}, q_{23}\} \\ \{q_8, q_{13}, q_{16}, q_{19}, q_{22}, q_{24}\} \end{array} \right\}
$$

Se hace necesario un algoritmo que permita identificar todas las regiones del AF. Para esto, realizaremos un recorrido del mismo, de forma que cada estado con más de una transición de salida será el origen de una nueva región. Para determinar el estado destino de una región se utiliza el algoritmo 5. Se trata de un algoritmo recursivo que recibe un AF $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ y un estado $q \in \mathcal{Q}$, y devuelve una secuencia de los estados que son comunes a todos los caminos que parten de q, donde INSERTAR($p,\ \rho$) inserta el estado p al inicio del camino ρ y FINAL-COMUN($\rho,\ \sigma$) devuelve el trozo final que sea común a ambos caminos. De esta forma, una llamada a esta función pasándole como parámetro un estado que sea origen de una región, nos devolverá la secuencia de estados que están en todos los caminos que parten del mismo, siendo el primer estado de esa secuencia el estado destino de la región en cuestión.

Algoritmo 5 Cálculo del estado destino de una región.

```
 1: procedimiento DESTINO-REGION(A, q)
 2:     si q ∈ Q_f entonces
 3:         retornar ∅
 4:     sino
 5:         p ← q.a | a ∈ Σ
 6:         primer_camino ← DESTINO-REGION(A, p)
 7:         INSERTAR(p, primer_camino)
 8:         para cada p ∈ Q, b ∈ Σ | p = q.b, b ≠ a hacer
 9:             camino_aux ← DESTINO-REGION(A, p)
10:             INSERTAR(p, camino_aux)
11:             primer_camino ← FINAL-COMUN(primer_camino, camino_aux)
12:         fin para
13:         retornar primer_camino
14:     fin si
15: fin procedimiento
```

Para comprender mejor el funcionamiento de la función DESTINO-REGION, veamos un ejemplo de su funcionamiento sobre el AF de la figura 10.3, que representa las palabras *descontar, despuntar, recontar y repuntar*. A simple vista, se puede observar que dicho AF, además de la región global, que por definición es $\mathcal{R}_{q_0}^{q_f}$, tiene dos regiones que son

$\mathcal{R}_{q_0}^{q_4}$ *y* $\mathcal{R}_{q_4}^{q_7}$. Supongamos que queremos calcular el estado destino de la región que empieza en q_0 aplicando el algoritmo 5. Para ello el algoritmo progresaría por las etapas siguientes:

1. Inicialmente invocaríamos a la función DESTINO-REGION pasándole como parámetros el AF de la figura 10.3 y q_0.

2. El algoritmo alcanzaría la línea 6 en la que se produce una llamada recursiva a la que se le pasaría como parámetro, además del AF, el primer estado accesible desde q_0, que en este caso sería q_1. Se entraría de este modo en un ciclo de llamadas recursivas en las que se iría transitando por los estados $q_3, q_4, q_5, q_7, q_8, q_9, q_{10}$ y, finalmente, q_f.

3. Al alcanzar el estado final estaríamos ante el caso base, por lo que el algoritmo alcanzaría la línea 3 y devolvería un camino vacío a la llamada que se realizó desde el estado q_{10} y que puede, de este modo, continuar en la línea 7 en la que inserta, en el camino vacío que acaba de recibir, el estado q_f. Dado que no hay más transiciones que partan del estado q_{10} retornaría el camino $\{q_f\}$ a la llamada realizada desde el estado q_9, que es también un estado del que parte una sola transición, por lo que se insertaría en el camino el estado q_{10} y se retornaría el camino $\{q_{10}, q_f\}$ a la llamada realizada desde el estado q_8. De este modo se iría retrocediendo en la recursividad hasta llegar al estado q_4 con $primer\_camino = \{q_5, q_7, q_8, q_9, q_{10}, q_f\}$.

4. Dado que del estado q_4 parte más de una transición, se entraría en el bucle de la línea 8 en el que se realizaría una nueva llamada recursiva con q_6 como parámetro y que, de igual forma, avanzaría mediante llamadas recursivas hasta alcanzar q_f, retrocediendo luego en la recursividad y construyendo el camino $\{q_6, q_7, q_8, q_9, q_{10}, q_f\}$. En ese momento se invocaría, en la línea 11, a la función FINAL-COMUN que nos devolvería el trozo final común a $primer\_camino$ y $camino\_aux$, es decir, $primer\_camino = \{q_7, q_8, q_9, q_{10}, q_f\}$, que sería lo que la función devolvería a la llamada realizada desde q_3.

5. Nuevamente se retrocedería en la recursividad hasta alcanzar el estado q_0 con $primer\_camino = \{q_1, q_3, q_4, q_7, q_8, q_9, q_{10}, q_f\}$. Una vez más se entraría en el bucle de la línea 8 y se repetiría todo el proceso, esta vez empezando por el estado q_2.

6. Al resolverse esta llamada tendríamos que $camino\_aux = \{q_2, q_4, q_7, q_8, q_9, q_{10}, q_f\}$, por lo que el trozo final común a ambos caminos sería $\{q_4, q_7, q_8, q_9, q_10, q_f\}$; que sería lo que devolvería finalmente el algoritmo.

7. En efecto, el primer estado del camino obtenido es q_4, que es el estado destino de la región que empieza en q_0.

En este algoritmo se puede observar una anomalía bastante común en algoritmos recursivos, a saber, la repetición de cálculos en las distintas ramas del árbol de recursividad. Así, podemos observar que en el punto 4 de la enumeración anterior se invoca recursivamente a la función pasándole como parámetro el estado q_6, lo que provocará una nueva llamada recursiva con q_7 como parámetro. En este punto se debería aplicar alguna técnica de programación dinámica para aprovechar que DESTINO-REGION$(\mathcal{A}, q_7)$ ya ha

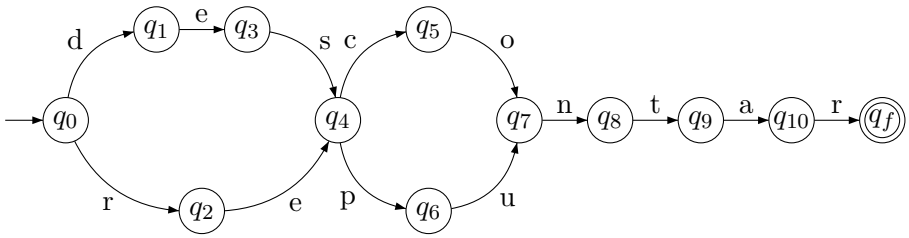

Figura 10.3: Cálculo del estado destino de una región.

sido calculada previamente desde el estado q_5, evitando de ese modo la repetición de una cantidad importante de cálculos.

Algoritmo 6 Cálculo del estado destino de una región aplicando programación dinámica.

```
1: procedimiento DESTINO-REGION-DINAMICA(A, q, T)
2:     si q ∈ Q_f entonces
3:         retornar ∅
4:     sino si q ∈ T entonces
5:         retornar T(q)
6:     sino
7:         p ← q.a | a ∈ Σ
8:         primer_camino ← DESTINO-REGION(A, p)
9:         INSERTAR(p, primer_camino)
10:        para cada p ∈ Q, b ∈ Σ | p = q.b, b ≠ a hacer
11:            camino_aux ← DESTINO-REGION(A, p)
12:            INSERTAR(p, camino_aux)
13:            primer_camino ← FINAL-COMUN(primer_camino, camino_aux)
14:        fin para
15:        T(q) ← primer_camino
16:        retornar primer_camino
17:    fin si
18: fin procedimiento
```

El algoritmo 6 es la versión del algoritmo 5, aplicando técnicas de programación dinámica. Simplemente se ha añadido el parámetro $\mathcal{T}$, que será una tabla indexada por los estados del AF, de forma que en $\mathcal{T}(q_n)$ se guardará el resultado de llamar a la función desde el estado q_n. De este modo, es posible recuperar esos resultados intermedios en lugar de recalcularlos. Nótese además que en el proceso de cálculo del estado destino de la región que comenzaba en q_0 hemos tenido que calcular también el estado destino de la región que comienza en q_4 que, de este modo, queda almacenado para su posterior recuperación. Es más, para cualquier AF $\mathcal{A}$, una llamada a la función DESTINO-REGION-DINAMICA($\mathcal{A}$, q_0, $\mathcal{T}$) nos proporcionaría en $\mathcal{T}$ el estado destino de todas las regiones del AF.

En una región, todos los prefijos reconocidos antes del estado origen pueden ser combinados con cualquiera de los sufijos reconocibles a partir del estado destino utilizando para ello los caminos de la región. Esto nos ofrece un criterio para delimitar una zona alrededor de un estado en la que cualquier cambio no tendrá efecto en su contexto.

Definición 10.8. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $w_{1..n}$ *una cadena, definimos la* región mínima *en* $\mathcal{A}$ que contiene $p = q_0.w_{1:i} \in \mathcal{Q}$, *y lo denotaremos como* $\mathcal{M}(\mathcal{A}, p, w)$ *o simplemente* $\mathcal{M}(p, w)$ *cuando el contexto sea suficientemente claro, como sigue:*

$$\mathcal{M}(p,w) = \mathcal{R}_{q_o}^{q_d} \Leftrightarrow p = q_0.w_{1:i},\ j < i,\ p_o = q_0.w_{1:j},\ p \in \mathcal{R}_{p_o}^{p_d} \Rightarrow q_o \geq p_o,\ q_d \leq p_d$$

□

Según la definición anterior, en el AF de la figura 10.1 podemos afirmar, por ejemplo, que $\mathcal{M}(q_5, debilito) = \mathcal{M}(q_{10}, delimito) = \mathcal{R}_{q_2}^{q_{20}}$ y $\mathcal{M}(q_{12}, drogadicto) = \mathcal{M}(q_{24}, dromedario) = \mathcal{R}_{q_8}^{q_{25}}$.

Un estado puede pertenecer a varias regiones, pero para el cálculo de la región mínima asociada a éste y a una cadena, sólo se tendrán en cuenta aquéllas cuyo estado inicial pertenezca a la traza de la cadena en el AF. De este modo, los estados origen de las regiones en cuestión estarán enlazados, lo que garantiza que las regiones con el estado origen menor contendrán a aquéllas que lo tengan mayor. Podemos, por tanto, enunciar el siguiente lema, cuya demostración es trivial a partir de la definición 10.8.

Lema 10.1. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $w \in \mathcal{L}(\mathcal{A})$ *una palabra, entonces* $p \in \mathcal{Q} \setminus \{q_0, q_f\} \Rightarrow \dot{\exists}\, \mathcal{M}(p, w)$.

Podemos introducir ahora formalmente el concepto de *ítem de detección de error*, que será el ítem a partir del cual el reconocedor deberá aplicar el algoritmo de reparación.

Definición 10.9. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF, $w \in \Sigma^*$ *una palabra,* $\langle \mathcal{I},\ \mathcal{H},\ \mathcal{D} \rangle$ *un esquema de análisis sintáctico, y sea* $[q, i] \in \mathcal{I}$ *un ítem de error, decimos que* $[p, j] \in \mathcal{I}$ *es un* ítem de detección de error *asociado a* $[q, i]$ *sii:*

$$\exists\ q_d \mid \mathcal{M}(q, w) = \mathcal{R}_p^{q_d}$$

Denotaremos esto como detección$([q, i]) = [p, j]$ *y diremos que* $\mathcal{M}(q, w)$ *es la región que define el ítem de detección de* $[q, i]$.

De forma análoga, diremos también que w_j *es el* punto de detección de error *asociado a* w_i *y lo denotaremos como* detección$(w_i) = w_j$.

□

A partir de la definición anterior y el lema 10.1 podemos enunciar un nuevo lema que garantiza la singularidad del ítem de detección de error.

Lema 10.2. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF, $\langle \mathcal{I},\ \mathcal{H},\ \mathcal{D} \rangle$ *un esquema de análisis sintáctico, y sea* $[q, i] \in \mathcal{I} \mid q \in \mathcal{Q} \setminus \{q_0, q_f\}$ *un ítem de error, entonces* $\dot{\exists}$ $[p, j] \mid$ detección$([q, i]) = [p, j]$

Ejemplo 10.3. *Retomando el ejemplo 10.2 en el que al tratar de analizar la palabra "*delito*" sobre el* AF *de la figura 10.1 el sistema alcanzaba el ítem de error* $[q_{10}, 5]$*, trataremos de determinar* detección$([q_{10}, 5])$*. Recordemos que el conjunto de ítems en el momento de generar* $[q_{10}, 5]$ *era* $\mathcal{I} = \{\ [q_0, 1],\ [q_1, 2],\ [q_2, 3],\ [q_6, 4]\ \}$*. Sabemos que para que* $[q, i]$ *pueda ser el ítem de detección asociado a* $[q_{10}, 5]$*, ha de cumplirse que* q *es el estado origen de la región mínima que contiene a* q_{10}*, esto es,* $\mathcal{M}(q_{10}, delito) = \mathcal{R}_q^{q_d}$*. Por tanto, dado que los únicos estados de los ítems de* $\mathcal{I}$ *que son el origen de alguna región son* q_1 *y* q_2 *y que* $q_2 > q_1$ *entonces* $\mathcal{M}(q_{10}, delito) = \mathcal{R}_{q_2}^{q_{20}}$ *por lo que* detección$([q_{10}, 5]) = [q_2, 3]$*.* ■

En este punto podemos decir que tenemos un mecanismo que nos permite limitar el impacto de un error en el proceso de análisis de una palabra. Si el analizador se detiene en un estado q, porque el siguiente carácter en la entrada no permite continuar el proceso, la región mínima que contenga a q parece una buena forma de delimitar la zona del AF en la que deberíamos realizar operaciones de reparación. En efecto, la región mínima que contiene a q comenzará en el primer estado anterior a q en el que exista una bifurcación de caminos en el AF, es decir, el estado más próximo a q, aparte del propio q, en el que hemos podido tomar un camino equivocado.

Una vez que hemos identificado la región de reparación, tanto desde el punto de vista topológico como operacional, podemos aplicar las *modificaciones* necesarias para recuperar el proceso de reconocimiento de un error. Para esto aplicaremos las operaciones de edición descritas por Damerau [36].

Definición 10.10. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $w_{1..n} \in \Sigma^*$ *una palabra. Definimos una* modificación *en* w *como una serie de operaciones de edición,* $\{E_i\}_{i=1}^{n}$*, en donde cada* E_i *se aplica a* w_i *y puede consistir en una secuencia de inserciones después de* w_i*, reemplazamiento o borrado de* w_i*, o la trasposición con* w_{i+1}*. Denotamos esto como* $M(w)$*.* □

Usaremos ahora la estructura topológica de los AFs para restringir la noción de modificación, introduciendo así el concepto de *reparación*. De forma intuitiva, buscaremos recuperar el proceso de reconocimiento estándar, al mismo tiempo que tratamos de aislar las ramas de reparación utilizando para ello el concepto de región.

Definición 10.11. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF*,* $x_{1..m}$ *un prefijo en* $\mathcal{L}(\mathcal{A})$*, y* $\mathrm{w} \in \Sigma^+$*, tal que* xw *no es un prefijo en* $\mathcal{L}(\mathcal{A})$*. Definimos una* reparación de w siguiendo a x *como una modificación* $M(w)$*, tal que:*

(1) $\mathcal{M}(q_0.x_{1..m}, xw) = \mathcal{R}_{q_o}^{q_d}$ *(la región mínima que incluye el punto de error,* $x_{1..m}$ *)*
(2) $\exists\ \{q_0.x_{1..i} = q_o.x_i, \ldots, q_o.x_{i..m}.M(w)\} \in caminos(\mathcal{R}_{q_o}^{q_d})$

Denotamos esto por reparación(x, w)*, y* $\mathcal{R}_{q_o}^{q_d}$ *por* ámbito(M)*.* □

Sin embargo, la noción de *reparación*(x, w) no es suficiente para nuestros propósitos, dado que nuestra intención es extender el proceso de recuperación para considerar todas las posibles reparaciones asociadas a un punto de error dado, lo que implica considerar diferentes prefijos simultáneamente.

Definición 10.12. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $y_i \in y_{1..n}$ *un punto de error, definimos el* conjunto de reparaciones para y_i*, como*

$$\text{reparaciones}(y_i) = \{xM(w) \in \textit{reparación}(x, w) \mid w_1 = \textit{detección}(y_i)\}$$

□

Necesitamos ahora un mecanismo para filtrar procesos de reparación no deseados, con el fin de reducir el coste computacional. Para esto, debemos introducir criterios de comparación para seleccionar sólo aquellas reparaciones cuyo coste sea mínimo.

Definición 10.13. *Para cada* $a, b \in \Sigma$ *asumimos los costes de inserción,* $I(a)$*; borrado,* $B(a)$*, reemplazamiento,* $R(a, b)$*, y trasposición,* $T(a, b)$*. El* coste de una modificación $M(w_{1..n})$ *viene dado por* $\text{coste}(M(w_{1..n})) = \Sigma_{j \in J_\dashv} I(a_j) + \Sigma_{i=1}^{n}(\Sigma_{j \in J_i} I(a_j) + B(w_i) + R(w_i, b) + T(w_i, w_{i+1}))$*, donde* $\{a_j,\ j \in J_i\}$ *es el conjunto de inserciones aplicadas antes de* w_i*;* $w_{n+1} = \dashv$ *el final de la entrada y* $T(w_n, \dashv) = 0$*.* □

Con el fin de tomar la *distancia de edición* [36, 92] como la métrica de error para medir la calidad de una reparación, basta con considerar costes discretos $I(a) = D(a) = 1,\ y\ R(a, b) = T(a, b) = 1,\ \forall a, b \in \Sigma,\ a \neq b$. Por otro lado, cuando son posibles varias reparaciones en distintos puntos de detección, necesitamos una condición para asegurar que sólo aquéllas de coste mínimo serán tenidas en cuenta, con el fin de obtener la mejor calidad de reparación. Sin embargo, esto no contradice la consideración de umbrales de error o de métricas de error alternativas.

Definición 10.14. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* $y_i \in y_{1..n}$ *un punto de error, definimos el conjunto de* reparaciones regionales para y_i*, como sigue:*

$$\textit{regional}(y_i) = \{xM(w) \in \textit{reparaciones}(y_i) / \textit{coste}(M) = \textit{min}_{L \in \textit{reparaciones}(y_i)}\{\{\textit{coste}(L)\}\}$$

□

También es necesario tener en cuenta la posibilidad de errores en cascada, esto es, errores precipitados por un diagnóstico de reparación erróneo previo. Antes de tratar el problema, necesitamos establecer la relación existente entre reparaciones regionales para un punto de error dado y puntos de error futuros.

Definición 10.15. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sean* w_i, w_j *puntos de error en* $w_{1..n} \in \Sigma^+$*,* $j > i$*. Definimos el conjunto de* reparaciones viables para w_i en w_j*, como*

$$\textit{viable}(w_i, w_j) = \{xM(y) \in \textit{regional}(w_i) / xM(y) \ldots w_j \textit{ prefijo para } \mathcal{L}(\mathcal{A})\}$$

□

Intuitivamente, las reparaciones en $\textit{viable}(w_i, w_j)$ son las únicas capaces de asegurar la continuidad del reconocimiento en $w_{i..j}$ y, por lo tanto, las únicas reparaciones posibles en el origen del fenómeno de errores en cascada.

Definición 10.16. *Sea w_i un punto de error para $w_{1..n} \in \Sigma^+$, decimos que un punto de error w_k, $k > j$ es un* punto de error precipitado por w_j *sii*

$$\forall x M(y) \in \mathit{viable}(w_j, w_k),\ \exists\, \mathcal{R}^{q_d}_{q_0.w_{1..i}} \mathit{definiendo}\ w_i = \mathit{detección}(w_j)$$

tal que ámbito$(M) \subset \mathcal{R}^{q_d}_{q_0.w_{1..i}}$. □

En la práctica, un punto de error w_k es precipitado por el resultado de reparaciones previas en un punto de error w_j, cuando la región que define el punto de detección para w_k abarca todas las reparaciones viables para w_j en w_k. Esto implica que la información compilada a partir de aquellas regiones de reparación no ha sido suficiente para dar continuidad al proceso de reconocimiento localizando el nuevo error en una región que contiene a las anteriores y, por lo tanto, depende de ellas. Esto es, la estructura gramatical subyacente sugiere que el origen del error actual podría ser un tratamiento equivocado de errores pasados. En otro caso, la localización se fijará en una que no depende de aquellas reparaciones previas.

10.2 El algoritmo

El algoritmo de corrección que aquí proponemos se basa en que la reparación se obtenga explorando el AF para encontrar una configuración conveniente que permita al proceso de reconocimiento continuar, lo que supone una aproximación clásica en la reparación de errores. Sin embargo, en el estado del arte [116, 140] no hay límite teórico de tamaño para la región de error, sólo para la distancia de edición de las correcciones en ella. Por lo tanto, con el fin de evitar distorsiones debidas a la localización insegura de errores, los autores hacen uso de algoritmos globales limitando los cálculos con un umbral en la distancia de edición. Esto les permite restringir la sección del AF a ser explorada mediante la poda de todos los caminos de reparación que sobrepasan el umbral [116], o bien aquéllos que no mantienen una distancia mínima que no sobrepase el umbral [140].

Sin embargo, el hecho de no beneficiarnos del conocimiento lingüístico presente en el AF para localizar el error y delimitar su impacto nos conduce a costes computacionales no óptimos o a la precipitación de nuevos errores[3]. Con el fin de eliminar este problema haremos uso de una construcción en la que todas las fases de reparación son guiadas dinámicamente por el AF en sí y, por lo tanto, inspirada en la estructura gramatical subyacente.

Supongamos que estamos tratando con el primer error detectado en una palabra $w_{1..n} \in \Sigma^+$. Las principales características del algoritmo suponen empezar con el ítem de error, cuyo contador de error es cero. De este modo, extendemos la estructura del ítem, $[p, i, e]$, donde e es ahora el contador de error acumulado en el reconocimiento de w en la posición w_i en el estado p.

En un sistema de reconocimiento estándar, un ítem identifica la situación en que se encuentra el reconocedor en un momento determinado. Es decir, dado un ítem, un AF y una cadena, sabemos cuál es el prefijo reconocido hasta el momento, en qué estado del AF

[3] Es decir, errores provocados por una reparación previa equivocada.

nos encontramos, y cuál es el trozo de cadena que falta por analizar. Por el contrario, en un sistema de reconocimiento tolerante a errores, el determinismo desaparece ya que, en un momento determinado, podemos estar explorando diferentes caminos del AF y haber reconocido varios prefijos distintos que, además, podrán no coincidir con ningún prefijo de la cadena. Es por esto que, para identificar la situación en que se encuentra el proceso de reconocimiento, es necesario ampliar la estructura del ítem incluyendo el prefijo que se ha reconocido hasta el momento. De esta forma la nueva estructura del ítem sería $[p, i, \alpha, e]$.

Se hace necesario ahora redefinir las reglas de deducción que nos permiten obtener nuevos ítems en modo de reconocimiento estándar, para adaptarlas a la nueva estructura.

Definición 10.17. *Definimos las* reglas de deducción en modo estándar para un sistema tolerante a errores *como sigue:*

$$\mathcal{D} = \mathcal{D}^{Inicial} \cup \mathcal{D}^{Salto}$$

$$\mathcal{D}^{Inicial} = \{\vdash [q_0, 1, \varepsilon, 0]\}$$
$$\mathcal{D}^{Salto} = \{[p, i, \alpha, e] \vdash [q, i+1, \alpha a, e] \ / \ \exists\, [a, i] \in \mathcal{H},\ q = p.a\}$$

□

Cuando encontramos un error en la cadena de entrada, no es posible continuar aplicando la regla de deducción $\mathcal{D}^{Salto}$, por lo que el sistema se pararía. Para poder continuar con el reconocimiento, se necesitan una serie de reglas de deducción que nos permitan seguir obteniendo ítems en modo error. A este conjunto de reglas de deducción lo denotaremos como $\mathcal{D}_{\text{error}}$.

Definición 10.18. *Definimos las* reglas de deducción en modo error para un sistema tolerante a errores *como sigue:*

$$\mathcal{D}_{\text{error}} = \mathcal{D}_{\text{error}}^{\text{Inserción}} \cup \mathcal{D}_{\text{error}}^{\text{Borrado}} \cup \mathcal{D}_{\text{error}}^{\text{Reemplazamiento}} \cup \mathcal{D}_{\text{error}}^{\text{Trasposición}}$$

$$\mathcal{D}_{\text{error}}^{\text{Inserción}} = \{[p, i, \alpha, e] \vdash [p, i+1, \alpha, e + I(w_i)], \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}, w) = \mathcal{R}_{q_o}^{q_d} \\ p \in \mathcal{R}_{q_o}^{q_d} \textbf{ or } p = q_o \\ \nexists\ p.w_i \end{array} \}$$

$$\mathcal{D}_{\text{error}}^{\text{Borrado}} = \{[p, i, \alpha, e] \vdash [q, i, \alpha a, e + D(a)] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}, w) = \mathcal{R}_{q_o}^{q_d} \\ p.a = q \in \mathcal{R}_{q_o}^{q_d} \textbf{ or } q = q_d \\ a \neq w_i \end{array} \}$$

$$\mathcal{D}_{\text{error}}^{\text{Reemplazamiento}} = \{[p, i, \alpha, e] \vdash [q, i+1, \alpha a, e + R(w_i, a)], \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}, w) = \mathcal{R}_{q_o}^{q_d} \\ p.a = q \in \mathcal{R}_{q_o}^{q_d} \textbf{ or } q = q_d \\ a \neq w_i \end{array} \}$$

$$\mathcal{D}_{\text{error}}^{\text{Trasposición}} = \{[p, i, \alpha, e] \vdash [q, i+2, \alpha w_{i+1} w_i, e + T(w_i, w_{i+1})] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}, w) = \mathcal{R}_{q_o}^{q_d} \\ p.w_{i+1}.w_i = q \in \mathcal{R}_{q_o}^{q_d} \textbf{ or } q = q_d \end{array} \}$$

donde $w_{1..j}$ busca el punto de error actual. □

Nótese que, en cualquier caso, las reglas de deducción en modo error sólo se activarán cuando efectivamente exista un error mientras que las reglas de deducción en modo estándar permanecerán activas durante todo el proceso de reconocimiento. Mediante estas reglas, sólo se podrán generar ítems que se encuentren dentro de la región de reparación, $\mathcal{M}(q_0.w_{1..j}, w) = \mathcal{R}_{q_o}^{q_d}$. El proceso continúa hasta que se genera un ítem que sobrepasa la región de reparación, momento en el cual las reglas de deducción en modo error dejarían

de ser aplicables y, por lo tanto, sólo se podría continuar el proceso de reconocimiento a través de las regla de reconocimiento en modo estándar $\mathcal{D}^{Salto}$.

Las reglas de deducción en modo error, tratan de recuperar el sistema de cada uno de los 4 errores de edición descritos por Damerau [36].

La primera de ellas, $\mathcal{D}_{\text{error}}^{\text{Inserción}}$, cubre la hipótesis de que el origen del error sea la inserción de un carácter erróneo por lo que, para reparar este error, lo que hace es generar un nuevo ítem en el que se ha avanzado una posición en la cadena de entrada, precisamente para saltar el carácter que ha sido presuntamente insertado de forma errónea. Además se incrementa el contador de error con el coste de inserción de w_i, $I(w_i)$. Para poder aplicar esta regla debe cumplirse además que no exista transición desde el estado actual con el siguiente carácter de la cadena de entrada, esto es, $\nexists\ p.w_i$, ya que de existir se debería aplicar la regla de deducción en modo estándar $\mathcal{D}^{Salto}$.

La segunda regla, $\mathcal{D}_{\text{error}}^{\text{Borrado}}$, cubre la hipótesis de borrado de un carácter por lo que permite generar un nuevo ítem para cada transición cuyo origen sea el estado del ítem actual, siempre y cuando el destino de esta transición sea un estado perteneciente a la región de reparación o bien coincida con el estado destino de la misma. La transición en cuestión determinará tanto el estado del ítem generado como el carácter que deberá añadírsele al prefijo, mientras que el error deberá ser incrementado con el coste de borrado, $D(a)$, y la posición en la cadena de entrada no varía. Esta regla no podrá aplicarse para insertar un carácter que coincida con el siguiente carácter en la cadena de entrada, $a \neq w_i$, ya que en ese caso se debe aplicar la regla de deducción en modo estándar $\mathcal{D}^{Salto}$.

La tercera regla, $\mathcal{D}_{\text{error}}^{\text{Reemplazamiento}}$, cubre la hipótesis de que el siguiente carácter de la entrada hubiera sido reemplazado por otro distinto. Esta regla es una combinación de las dos anteriores ya que sustituir un carácter por otro es lo mismo que eliminarlo e insertar luego el nuevo. Por tanto, esta regla permitirá también generar un nuevo ítem para cada transición cuyo origen sea el estado del ítem actual, siempre y cuando el destino de esta transición sea un estado perteneciente a la región de reparación o bien coincida con el estado destino de la misma. De igual modo, la transición aplicada determinará tanto el estado del ítem generado como el carácter que deberá añadírsele al prefijo, mientras que el error deberá ser incrementado con el coste de reemplazamiento, $R(w_i, a)$, y la posición en la cadena de entrada se avanza un carácter.

Por último, la regla $\mathcal{D}_{\text{error}}^{\text{Trasposición}}$, cubre la hipótesis de trasposición de dos caracteres consecutivos por lo que sólo será aplicable cuando sea posible transitar desde el estado del ítem actual, con el carácter que sucede al siguiente en la cadena de entrada, a un estado intermedio y desde éste sea posible transitar con el carácter siguiente en la cadena de entrada. De este modo, el estado del nuevo ítem será el que se alcanza al aplicar las dos transiciones anteriores, esto es, $p.w_{i+1}.w_i$. La posición en la cadena se avanzará dos caracteres que serán los que se añadirán al prefijo en orden inverso mientras que el error se incrementará con el coste de trasposición, $T(w_i, w_{i+1})$.

Supongamos ahora que el proceso de reparación no es el primero de la palabra y, por tanto, puede modificar uno anterior. Esto se presenta cuando volvemos a un ítem de detección para el que alguna rama de reconocimiento incluye un proceso de reparación previo. Para ilustrar este caso, volvemos a la figura 10.1 suponiendo que $I_n = [q_5, k, \alpha, e_k]$ e $I_m = [q_{23}, l, \gamma, e_l]$ son ítems de error. Como consecuencia, I_m sería precipitado por I_n

dado que $\mathcal{A} = \mathcal{R}_{q_1}^{q_{25}}$ incluye $\mathcal{R}_{q_2}^{q_{20}}$, el ámbito de la reparación anterior.

Para tratar con errores precipitados, el algoritmo retoma los contadores de error anteriores, añadiendo el coste de una nueva hipótesis de reparación para aprovechar la experiencia ganada en fases de recuperación previas. En este punto, las reparaciones regionales tienen dos propiedades importantes. Primero, son independientes de la construcción del AF y en segundo lugar, no se pierde eficiencia en comparación con aproximaciones de reparación globales.

Lema 10.3. *(Lema de expansión) Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un AF y sean w_k, w_l puntos de error en $w_{1..n} \in \Sigma^+$, tal que w_l es precipitado por w_k, entonces:*

$$q_0.w_{1..i} < q_0.w_{1..j},\ \mathcal{M}(q_0.w_{1..l}, w) = \mathcal{R}_{q_0.w_{1..i}}^{q_d},\ w_j = y_1,\ xM(y) \in \mathit{viable}(w_k, w_l)$$

Demostración. Sea $w_j \in \Sigma$, tal que $w_j = y_1$, y sea $xM(y) \in \mathit{viable}(w_k, w_l)$ un punto de detección para w_k, para el cual alguna rama de reconocimiento derivada de una reparación en $\mathit{regional}(w_k)$ llegó con éxito a w_l. Sea también w_l un punto de error precipitado por $xM(y) \in \mathit{viable}(w_k, w_l)$. Por la definición 10.16, podemos afirmar que

$$\text{ámbito}(M) \subset \mathcal{M}(q_0.w_{1..l}, w) = \mathcal{R}_{q_0.w_{1..i}}^{q_d}$$

Dado que $\mathit{ámbito}(M)$ es la menor región que engloba $q_0.w_{1..j}$, se sigue que $q_0.w_{1..i} < q_0.w_{1..j}$. Concluimos la demostración extendiendo esto a todas las reparaciones en $\mathit{viable}(w_k, w_l)$.

□

Intuitivamente, probamos que el estado asociado al punto de detección en un error en cascada es menor que el asociado al origen del ámbito en las reparaciones que lo precipitaron. Como consecuencia, el mínimo ámbito posible de una reparación para errores en cascada incluye cualquier ámbito de esas reparaciones anteriores.

Corolario 10.1. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un AF y sean w_k, w_l puntos de error en $w_{1..n} \in \Sigma^+$, tal que w_l es precipitado por w_k, entonces*

$$\mathit{max}\{\text{ámbito}(M),\ M \in \mathit{viable}(w_k, w_l)\} \subset \mathit{max}\{\text{ámbito}(\tilde{M}),\ \tilde{M} \in \mathit{regional}(w_l)\}$$

Demostración. Se sigue inmediatamente del lema 10.3. □

Veamos de forma gráfica cómo el sistema se recupera de un error precipitado por otro previo. Para ello nos remitimos al AF de la figura 10.4 que representa las palabras *claudicar, clamoroso, caluroso, calimoso* y *cadalso*.

Supongamos que queremos reparar la palabra errónea "*caludicar*", que se obtiene a partir de "*claudicar*" mediante la trasposición de la 'a' y la 'l'. Este error provoca que nos adentremos en un camino erróneo hasta que el proceso de reconocimiento se detiene en el ítem de error $[q_{11}, 5, calu, 0]$. Con el fin de delimitar la región en la que deberemos realizar las operaciones de reparación, buscamos la región mínima que contiene a q_{11}, que

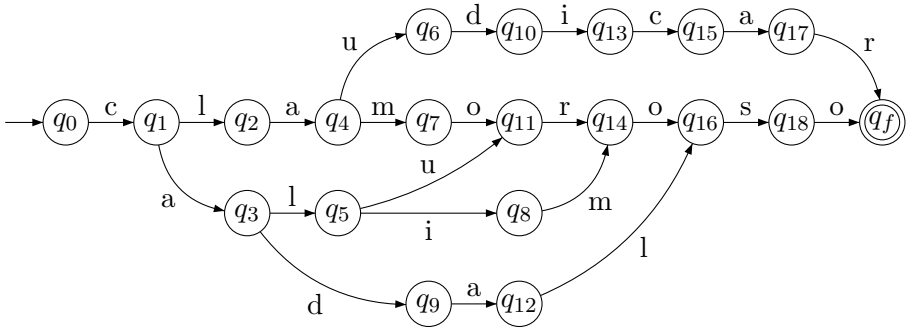

Figura 10.4: AF para: *claudicar, clamoroso, caluroso, calimoso y cadalso.*

en este caso es $\mathcal{M}(q_{11}, caludicar) = \mathcal{R}_{q_5}^{q_{14}}$ y, en consecuencia, *detección*$([q_{11}, 5, calu, 0]) = [q_5, 3, cal, 0]$.

Aplicamos entonces, tanto en el ítem de error como en el de detección, las reglas de deducción en modo error, que garantizan que todos los ítems generados se mantendrán dentro de $\mathcal{M}(q_{11}, caludicar)$, con el objetivo de alcanzar q_{14} en una situación que nos permita continuar en modo de reconocimiento estándar. En este caso, dado que la única transición de salida en el estado q_{14} está etiquetada con 'o' y que no hay ninguna 'o' a la derecha del punto de detección del error, no se generará ningún ítem que permita continuar en modo estándar a partir de q_{14}.

Así pues, en q_{14} se producirá, inevitablemente, una parada del AF que nos permitirá decir que el error que hemos tratado previamente ha sido precipitado por otro anterior. El proceso de reconocimiento continuará tratando de recuperar el nuevo error detectado en q_{14}. Ahora, $\mathcal{M}(q_{14}, caludicar) = \mathcal{R}_{q_3}^{q_{16}}$ que delimita la zona del AF en la que aplicaremos las reglas de deducción en modo error pero, al igual que en el caso anterior, no es posible alcanzar q_{16} con una configuración que nos permita continuar el reconocimiento en modo estándar. De esta forma, se producirá una nueva parada del AF en q_{16}, lo que nos indicará que hemos de retroceder aún más para encontrar el origen de este comportamiento anómalo.

Dado que *detección*$([q_{16}, i, \alpha, e]) = [q_1, 2, c, 0]$ y la aplicación de $\mathcal{D}_{\text{error}}^{\text{Trasposición}}$ sobre este ítem nos permite continuar aplicando de forma iterativa $\mathcal{D}_{\text{error}}^{\text{Salto}}$ hasta alcanzar $[q_f, 10, claudicar, 1]$, que es un ítem final, podemos concluir que el error detectado en q_{11} ha sido precipitado por la trasposición de la 'l' y la 'a' y que el sistema ha ido retrocediendo en la palabra de entrada hasta encontrar el punto en el que el error se había originado.

Esto nos permite obtener un comportamiento asintótico asimilable al de los métodos de reparación globales. Esto es, el algoritmo asegura una calidad comparable a estrategias globales, pero con el coste de una local. Esto tiene profundas implicaciones para la eficiencia, medida en tiempo, la simplicidad y la potencia de cálculo.

Lema 10.4. *Sea $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ un* AF *y sea w_i un punto de error en $w_{1..n} \in \Sigma^+$, el tiempo asociado para la reparación regional es, en el peor de los casos,*

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \textit{grado-salida}_{\mu}^{\tau})$$

donde τ *y* grado-salida$_{\mu}^{\tau}$ *son, respectivamente, el máximo contador de error calculado y el máximo grado de salida de los estados del* AF *en el ámbito de las reparaciones consideradas.*

Demostración. Aquí la prueba es una simple extrapolación de la estimación propuesta por el algoritmo de Savary [140]. En el peor de los casos, hay al menos $n!/(\tau! * (n-\tau)!)$ distribuciones posibles de τ modificaciones sobre n posiciones de error. Para cada distribución se siguen al menos $(1 + 2 *$ grado-salida$_{\mu}^{\tau})$ caminos, siendo cada camino al menos de longitud $n + \tau$. Por tanto, el peor caso de complejidad es el propuesto. □

Sin embargo, este lema no determina aún la relación con aproximaciones globales clásicas [116, 140], como es nuestra intención, sino sólo un caso medio de estimación de nuestra propia complejidad en tiempo. Para alcanzar esto, extendemos la región de reparación al AF total.

Corolario 10.2. *Sea* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ *un* AF *y sea* w_i *un punto de error en* $w_{1..n} \in \Sigma^+$*, el tiempo asociado para la reparación regional es, en el peor de los casos, el mismo que el alcanzado para una reparación global.*

Demostración. Se sigue inmediatamente del anterior lema 10.4 y del corolario 10.1, así como de [140]. En efecto, en el peor de los casos, el ámbito de la reparación es el AF global. □

Téngase en cuenta el tipo de demostración aplicada en el lema 10.4. Ésta implica que nuestra técnica tiene la misma complejidad en tiempo que la declarada por la global de Savary [140], hasta donde nosotros sabemos, la propuesta de corrección ortográfica global más eficiente.

10.3 Recuperación de cadenas incompletas

En determinadas aplicaciones, como el reconocimiento automático de caracteres o el reconocimiento de voz, es necesario manejar cadenas incompletas debido al ruido, fenómeno que puede deberse a defectos en el proceso de obtención del texto, a errores introducidos durante su transmisión, a imperfecciones en el soporte de almacenamiento, etc. Este tipo de errores dan lugar, en ocasiones, a cadenas incompletas, que son un tipo especial de errores ortográficos [165].

Para poder manejar cadenas incompletas, extendemos el alfabeto de entrada introduciendo dos nuevos símbolos. Así, '?' indicará la existencia de un carácter desconocido, mientras que '$*$' indicará la presencia de una secuencia de caracteres desconocida. Para saber qué hacer cuando durante el proceso de reconocimiento se encuentra uno de estos dos nuevos símbolos, necesitamos ampliar el conjunto de reglas de deducción en modo estándar $\mathcal{D}$ con el siguiente conjunto de reglas de deducción para cadenas incompletas, $\mathcal{D}_{\text{Incompleta}}$:

$$\mathcal{D}^{\text{Salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e] \vdash [q,i+1,\alpha a,e+I(a)],\ \exists\,[?,i] \in \mathcal{H},\ q = p.a\}$$
$$\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e] \vdash [q,i,\alpha a,e+I(a)],\ \exists\,[*,i] \in \mathcal{H},\ q = p.a\}$$
$$\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e] \vdash [q,i,\alpha a,e+I(a)],\ \exists\,[*,i] \in \mathcal{H},\ q = p.a,\ \exists\ q.w_{i+1}\}$$

donde $I(a)$ es el coste de insertar un símbolo $a \in \Sigma$.

Desde un punto de vista intuitivo, $\mathcal{D}^{\text{Salto}}_{\text{Incompleta}}$ aplica cualquiera de las transiciones de salto independientemente del símbolo que suceda al '?' en la cadena de entrada, teniendo en cuenta que esa transición ha de ser aplicable respecto de la configuración del AF y que el siguiente símbolo en la entrada es un carácter desconocido.

En lo que respecta a $\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}}$, simula acciones de salto sobre ítems que corresponden a configuraciones del AF para las que el siguiente símbolo en la entrada denota una secuencia desconocida de caracteres.

En el momento en que es posible una acción de salto estándar para el símbolo que sucede al '$*$', se activa la regla $\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}}$ que pone fin a la inserción de caracteres avanzando en la cadena de entrada y abandonando, por tanto, el modo de reconocimiento de cadenas incompletas. Nótese, sin embargo, que a la vez que se aplica la regla $\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}}$, es posible aplicar también la regla $\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}}$. Esto permite manejar el caso en que el símbolo que sucede al '$*$' haga posible una acción de salto estándar, aunque, no es ese el punto en que se debería abandonar el modo de recuperación de secuencias incompletas sino otro posterior.

Para ilustrar esto, nos remitimos nuevamente al AF de la figura 10.1, suponiendo que la palabra a reconocer sea "de$*$ito", se alcanzaría inicialmente el ítem $[q_2, 3, de, 0]$ por medio de la aplicación de las reglas de deducción del modo estándar. En este punto, se cumpliría la condición $\exists\ q.w_{i+1}$, lo que provocaría la activación simultánea de las reglas $\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}}$ y $\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}}$, por lo que se aplicarían ambas. La primera generaría los ítems $[q_5, 3, deb, 1]$ y $[q_6, 3, del, 1]$, mientras que la segunda generaría $[q_5, 4, deb, 1]$ y $[q_6, 4, del, 1]$. Nótese que los dos ítems generados por la regla $\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}}$ avanzan en la cadena de entrada dejando atrás el '$*$' por lo que se abandona el modo de reconocimiento de cadenas incompletas y se pasa al modo estándar. En el modo estándar se podrá aplicar la regla $\mathcal{D}^{\text{Salto}}$ generando los ítems $[q_9, 5, debi, 1]$ y $[q_{10}, 5, deli, 1]$ a partir de los que no sería posible continuar.

Por su parte, a los dos ítems generados por la regla $\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}}$ se les podría aplicar de nuevo esta misma regla dando lugar a los ítems $[q_9, 3, debi, 2]$ y $[q_{10}, 3, deli, 2]$. Nuevamente sería ahora posible continuar con la inserción de caracteres o bien pasar al modo de reconocimiento estándar. Se puede observar que, haciendo esto último, sería posible alcanzar los dos ítems finales $[q_f, 7, debilito, 3]$ y $[q_f, 7, delimito, 3]$. Así, este ejemplo ilustra cómo, aunque el símbolo que sucede al '$*$' coincida con la siguiente transición del AF, es posible que no sea ése el punto en el que hay que abandonar el modo de recuperación de secuencias incompletas.

De esta forma, cuando tratamos con secuencias de caracteres desconocidos, podemos examinar diferentes caminos en el AF para resolver el mismo símbolo '$*$'. Aunque esto podría ser de utilidad para posteriores procesos de análisis sintáctico o semántico, una sobre-generación incontrolada no es de interés práctico en la mayoría de los casos.

Resolvemos este problema tabulando el número de caracteres insertados para reconstruir la palabra, por medio del contador de error, y aplicando el principio de optimización. La complejidad del algoritmo sigue siendo, en el peor de los casos:

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \text{grado-salida}_{\mu}^{\tau})$$

10.4 Corrección ortográfica robusta

Una vez introducida la corrección de errores de edición y la recuperación de cadenas incompletas, podemos garantizar la capacidad de recuperar el reconocedor de cualquier situación inesperada derivada bien de vacíos en la obtención de los caracteres o bien de errores de edición [165]. Para esto, es suficiente con combinar todas las reglas de deducción que hemos ido introduciendo hasta el momento. Más concretamente, definimos un nuevo conjunto de reglas de deducción, $\mathcal{D}_{\text{robusta}}$, como sigue:

$$\begin{aligned}\mathcal{D}_{\text{robusta}} = \quad & \mathcal{D}^{\text{Inicial}} \cup \mathcal{D}^{\text{Salto}} \cup \mathcal{D}_{\text{Error}}^{\text{Inserción}} \cup \mathcal{D}_{\text{Error}}^{\text{Borrado}} \cup \mathcal{D}_{\text{Error}}^{\text{Reemplazamiento}} \cup \\ & \mathcal{D}_{\text{Error}}^{\text{Trasposición}} \cup \mathcal{D}_{\text{Incompleta}}^{\text{Salto}} \cup \mathcal{D}_{\text{Incompleta}}^{\text{Bucle\_salto}} \cup \mathcal{D}_{\text{Incompleta}}^{\text{Fin\_bucle\_salto}}\end{aligned}$$

donde no hay solapamiento entre las reglas de deducción. El reconocedor robusto final sigue teniendo la misma complejidad, en el peor de los casos, que el reconocedor de cadenas incompletas, es decir:

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \text{grado-salida}_{\mu}^{\tau})$$

respecto de la longitud n de la cadena errónea. La cadena de entrada será reconocida si se genera un ítem final $[q_f, n+1, \alpha, e]$.

Para explicar cómo funcionaría el reconocedor sobre una palabra que presenta tanto secuencias incompletas como errores de edición, veamos lo que ocurriría si intentásemos reconocer la palabra *"ca?uraso"* sobre el AF de la figura 10.4. Inicialmente, aplicando las reglas de deducción en modo estándar, alcanzaríamos el ítem $[q_3, 3, ca, 0]$ y nos encontraríamos, por tanto, con el símbolo '?' en la cadena de entrada, por lo que se activaría la regla $\mathcal{D}_{\text{Incompleta}}^{\text{Salto}}$ por medio de la cual se generarían los ítems $[q_5, 4, cal, 1] y [q_9, 4, cad, 1]$. Como se puede ver, en este caso sólo es posible continuar con el reconocimiento estándar a partir de $[q_5, 4, cal, 1]$ que, mediante la aplicación de la regla $\mathcal{D}^{\text{Salto}}$, generaría $[q_{11}, 5, calu, 1]$ y éste a su vez $[q_{14}, 6, calur, 1]$, que es un ítem de error. Entrarían entonces en funcionamiento las reglas $\mathcal{D}_{\text{Error}}$ sobre la región $\mathcal{M}(q_{14}, ca?uraso) = \mathcal{R}_{q_3}^{q_1 6}$, que permitirían alcanzar el ítem final $[q_f, 9, caluroso, 2]$.

10.5 Estrategias de poda

Los errores ortográficos pueden ser a menudo recuperados de diferentes maneras, lo que nos obliga a considerar un marco de trabajo que permita ambigüedades. Aunque muchas de éstas serán eliminadas en tareas de análisis posteriores y más sofisticadas, una parte de

ellas pueden ser tratadas aquí [165]. Dejando a un lado aspectos puramente estadísticos, nos centramos en la formalización de un esquema de podado que nos permita limitar el espacio de reparación y, en consecuencia, reducir el impacto computacional derivado de la exploración de rutas de reparación inútiles.

No obstante, la interpretación de un AF como un formalismo transaccional secuencial, nos impone una pauta esencial en el diseño de cualquier estrategia de poda. Si además tenemos en cuenta que el marco dinámico que define el comportamiento del reconocedor actualiza los contadores de error cada vez que se genera un nuevo ítem, tenemos que las técnicas de poda basadas en umbrales de error, parecen adaptarse particularmente bien en este caso. Por tanto, podemos considerar una serie de reglas de poda simples, combinando las hipótesis de reparación con el fin de permitir al usuario la implementación de estrategias de corrección similares a las que realizaría un humano.

La primera de las estrategias de poda que planteamos consiste simplemente en eliminar aquellas ramas que aparezcan repetidas en el árbol de reconocimiento. En efecto, la combinación de operaciones de edición puede conducirnos a ítems equivalentes. En concreto, si en una determinada posición de la cadena de entrada aplicamos, por ejemplo, la hipótesis de inserción e inmediatamente aplicamos la de borrado, el ítem obtenido será equivalente al que obtendríamos si aplicásemos una hipótesis de reemplazamiento, con la única diferencia de que este último tendría un contador de error menor. Esta poda puede simularse mediante la aplicación de técnicas de programación dinámica para evitar la repetición de ítems con un contador de error mayor o igual que otros calculados previamente.

En la figura 10.5 se muestra el árbol de ítems generado durante el proceso de reconocimiento de la palabra errónea "*caludicar*" sobre el AF de la figura 10.4 una vez eliminados los ítems repetidos.

10.5.1 Poda basada en el camino

Nos referimos aquí a una técnica clásica [116, 140] que consiste en podar aquellas ramas de reparación con ítems cuyo contador de error haya alcanzado un determinado umbral. Desde un punto de vista operacional, la consideración de este mecanismo de simplificación no requiere ninguna modificación en la estructura de ítem ya que una simple comprobación sobre el contador de error, cada vez que se genera uno nuevo, es suficiente para determinar si éste debe ser añadido o no al conjunto de ítems $\mathcal{I}$. Si el contador de error del nuevo ítem es mayor que el umbral definido, τ, simplemente se evitará la generación del mismo. Para integrar esta poda en nuestro esquema de análisis es necesario que en todas las reglas que actualizan el contador de error se establezca una nueva condición que evite que se generen ítems que sobrepasen el umbral de error permitido de forma que todos cumplan:

$$\vdash [p, i, \alpha, e] \in \mathcal{D}_{\text{Robusta}}, e < \tau, \forall I \in \mathcal{I}$$

Como ejemplo, considerando la métrica de edición [36, 92], podemos podar todas las las ramas de reparación con un contador de error mayor que una proporción fija de la longitud de la cadena.

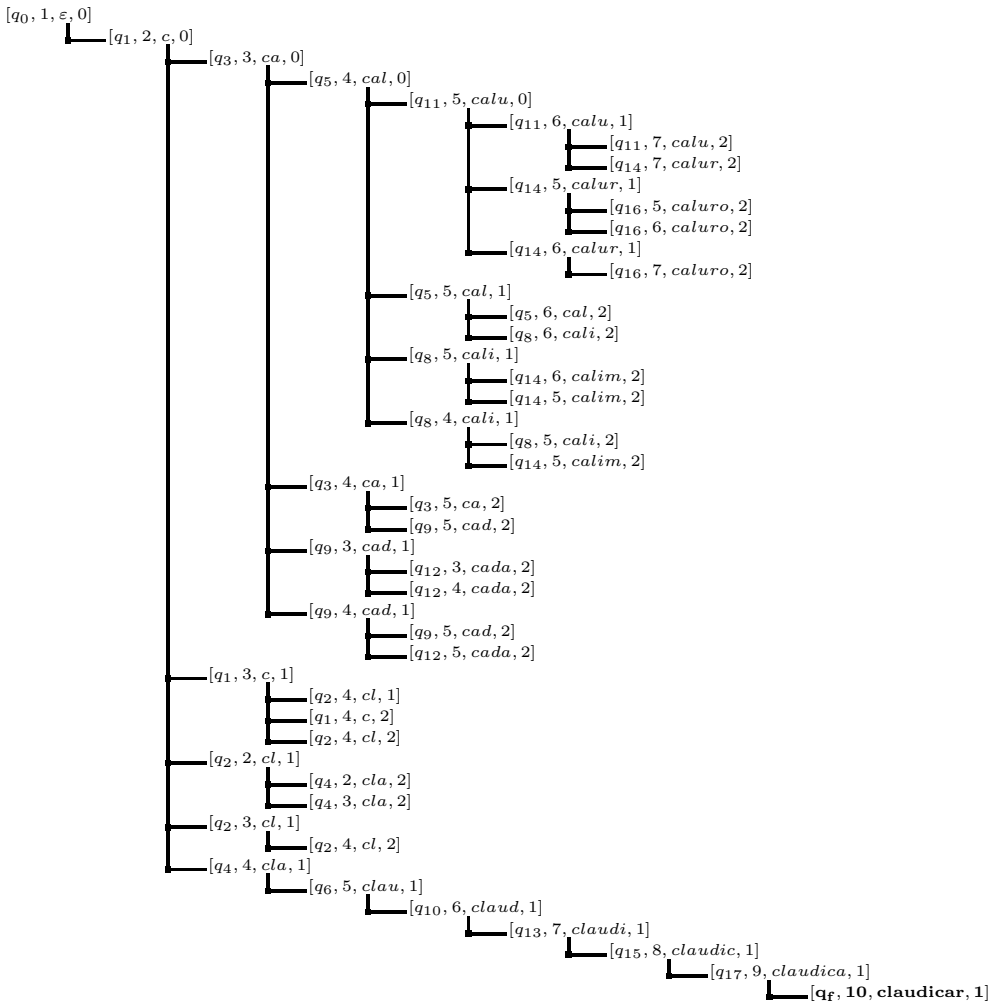

Figura 10.5: Árbol de ítems para el reconocimiento de la palabra errónea "*caludicar*" sobre el AF de la figura 10.4.

Si aplicásemos esta poda al árbol de la figura 10.4 obtendríamos el de la figura 10.6, lo que supone un notorio ahorro en términos de cálculos realizados.

10.5.2 Poda de errores consecutivos

Otra posible aproximación consiste en limitar el número de errores consecutivos incluidos en un camino, podando en los ítems que presenten una calidad inferior a un umbral determinado, σ. Para implementar esta estrategia de poda, es necesario introducir un nuevo contador de error, e_l, que representará la cuenta local de errores acumulados a lo largo de una secuencia de hipótesis de reparación en el camino que se está explorando. Por tanto, se hace necesario redefinir la estructura de ítem, $[p, i, \alpha, e_l, e_g]$, donde el contador e_g es el mismo que veníamos considerando en el algoritmo inicial. Es también necesario

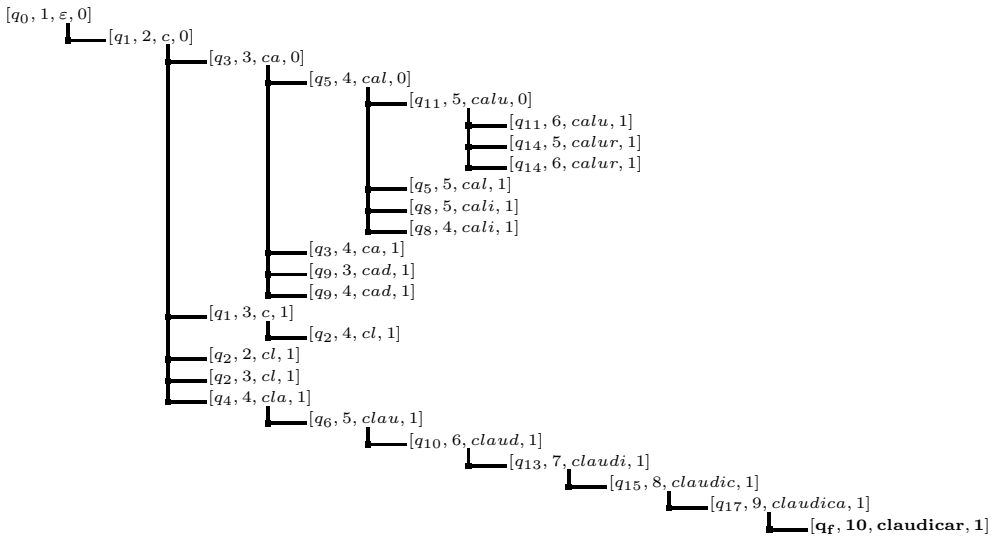

Figura 10.6: Árbol resultado de la aplicación de la poda basada en el número total de errores, para un umbral de 1, sobre el árbol de la figura 10.5.

redefinir algunas reglas de deducción del esquema original para el modo robusto:

$$\mathcal{D}^{\text{Salto}}_{\text{Error}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i+1,\alpha a,0,e_g],\ \exists\,[a,i] \in \mathcal{H},\ q = p.a\}$$
$$\mathcal{D}^{\text{Inserción}}_{\text{Error}} = \{[p,i,\alpha,e_l,e_g] \vdash [p,i+1,\alpha,e_l + I(w_i), e_g + I(w_i)],\ \nexists\ p.w_i\}$$
$$\mathcal{D}^{\text{Borrado}}_{\text{Error}} = \{[p,i,\alpha,e] \vdash [q,i,\alpha a, e + D(a)]/\psi^{Borrado}_{Error}\}$$
$$\mathcal{D}^{\text{Reemplazamiento}}_{\text{Error}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i+1,\alpha a, e_l + R(w_i,a), e_g + R(w_i,a)],/\psi^{Reemplazamiento}_{Error}\}$$
$$\mathcal{D}^{\text{Trasposición}}_{\text{Error}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i+2,\alpha w_{i+1} w_i, e_l + T(w_i,w_{i+1}), e_g + T(w_i,w_{i+1})]/\psi^{Trasposición}_{Error}\}$$
$$\mathcal{D}^{\text{Salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i+1,\alpha a, e_l + I(a), e_g + I(a)],\ \exists\,[?,i] \in \mathcal{H},\ q = p.a\}$$
$$\mathcal{D}^{\text{Bucle\_salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i,\alpha a, e_l + I(a), e_g + I(a)],\ \exists\,[*,i] \in \mathcal{H},\ q = p.a\}$$
$$\mathcal{D}^{\text{Fin\_bucle\_salto}}_{\text{Incompleta}} = \{[p,i,\alpha,e_l,e_g] \vdash [q,i,\alpha a, e_l + I(a), e_g + I(a)],\ \exists\,[*,i] \in \mathcal{H},\ q = p.a,\ \exists\ q.w_{i+1}\}$$

donde $\psi^{Borrado}_{Error}$, $\psi^{Reemplazamiento}_{Error}$ y $\psi^{Trasposición}_{Error}$, corresponden a las condiciones necesarias para que se pueda aplicar cada una de las reglas y coinciden con las de las reglas originales introducidas en la sección 10.2.

Este tipo de poda evita la aplicación reiterada de operaciones de reparación sobre la base de que es poco probable que un usuario cometa varios errores adyacentes en una misma palabra. Su aplicación para un umbral de 1 sobre el árbol de la figura 10.5, en este caso, daría como resultado el árbol de la figura 10.6.

10.5.3 Poda basada en el tipo de errores

A veces podemos estar más interesados en detectar la presencia de algunas hipótesis de error en particular en un camino del AF o, más concretamente, en una secuencia de ese camino. Esto se traduce en aplicar la poda basada en el camino y la de errores consecutivos, introducidas previamente, a un tipo de hipótesis de error en particular. Tomando, por ejemplo, el caso de $\mathcal{D}^{\text{Inserción}}_{\text{Robusta}}$ y asumiendo un umbral de error de τ para aplicar la acción de poda sobre un camino, tendríamos que la nueva regla de deducción sería ahora:

$$\mathcal{D}^{\text{Inserción}}_{\text{Error}} = \{ \quad [p, i, \alpha, e_l, e_g, e^i_l, e^i_g] \vdash \\ [p, i+1, \alpha, e_l + I(w_i), e_g + I(w_i), , e^i_l + I(w_i), e^i_g + I(w_i)],\ \nexists\ p.w_i,\ e^i_g + I(w_i) < \tau\}$$

mientras que, si estuviésemos tratando con una secuencia de un camino, tendríamos que:

$$\mathcal{D}^{\text{Inserción}}_{\text{Error}} = \{ \quad [p, i, \alpha, e_l, e_g, e^i_l, e^i_g] \vdash \\ [p, i+1, \alpha, e_l + I(w_i), e_g + I(w_i), , e^i_l + I(w_i), e^i_g + I(w_i)],\ \nexists\ p.w_i,\ e^i_l + I(w_i) < \tau\}$$

y, en ambos casos, asumimos que la aplicación de la regla de salto estándar reinicia todos los contadores locales a cero:

$$\mathcal{D}^{\text{Salto}}_{\text{Error}} = \{[p, i, \alpha, e_l, e_g, e^i_l, e^i_g] \vdash [q, i+1, \alpha a, 0, e_g, 0, e^i_g],\ \exists\ [a, i] \in \mathcal{H},\ q = p.a\}$$

Ha de tenerse en cuenta que necesitamos un par de contadores asociados a cada tipo de regla de deducción de forma que podamos considerar este tipo de poda para las hipótesis de inserción, borrado, reemplazamiento o trasposición. Su aplicación para un umbral de 1 convertiría el árbol de la figura 10.5 en el que se muestra en la figura 10.7.

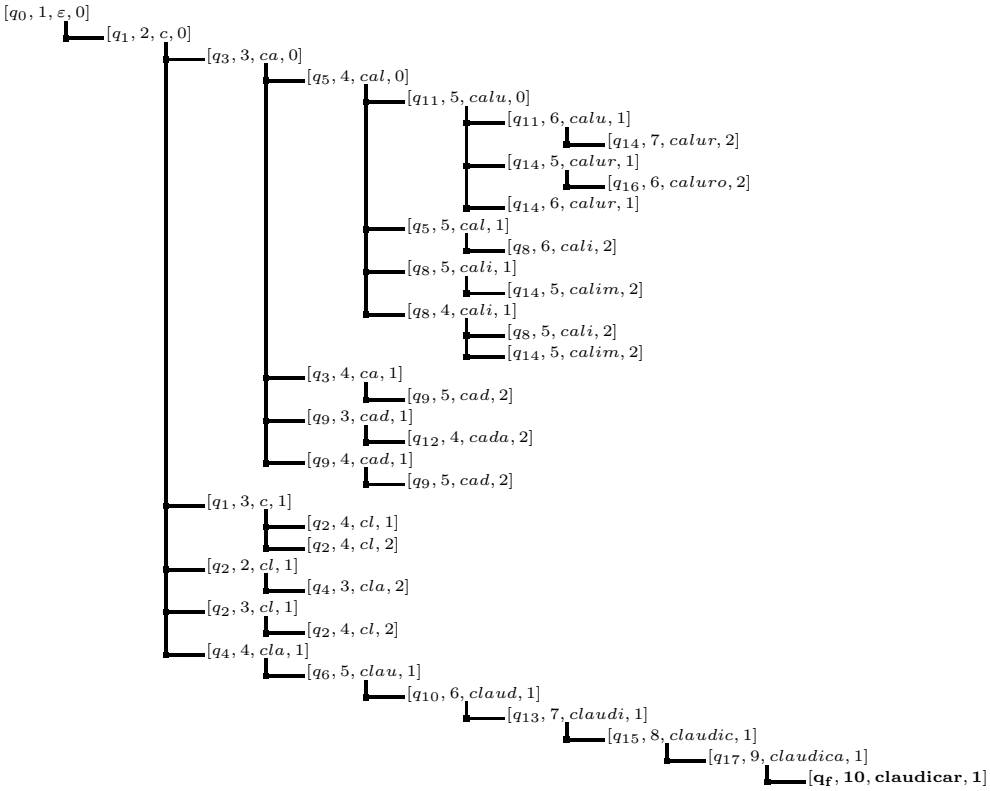

Figura 10.7: Árbol resultado de la aplicación de la poda basada en el tipo de errores, para un umbral de 1, sobre el árbol de la figura 10.5.

Capítulo 11

Corrección ortográfica contextual

En capítulos anteriores nos hemos centrado en el análisis y desarrollo de algoritmos de corrección ortográfica capaces de ofrecer una lista de alternativas para una palabra desconocida aislada, esto es, sin tener en cuenta ninguna información referente al contexto en que esa palabra aparece. Sin embargo, esto no es lo más usual ya que en la mayor parte de los entornos en que se requiere la aplicación de este tipo de técnicas, existe la posibilidad de examinar las palabras que rodean a aquélla que estamos procesando, con el objetivo de determinar cuál de las alternativas es la que mejor encaja en ese contexto.

Desde este punto de vista podemos clasificar las aplicaciones que requieren el uso de técnicas de corrección ortográfica en dos grupos:

- *Interactivas*: aquéllas en las que es posible ofrecer al usuario una lista de correcciones para que éste elija la que mejor se adapta al contexto. El ejemplo más representativo de este tipo de aplicaciones son los procesadores de textos que suelen incorporar una herramienta que indica al usuario cuáles son las palabras desconocidas y le ofrece una lista de alternativas. Sin embargo, aún en este caso en que la interacción con el usuario es posible, éste podría sentirse desalentado por una larga lista de alternativas en la que aquélla que él considera más adecuada no aparece en las primeras posiciones.

- *No interactivas*: aquéllas que se ejecutan en ausencia de interacción alguna con el usuario y que, por lo tanto, no pueden solicitar la intervención de éste. En este caso la decisión acerca de cuál es la alternativa más adecuada recae sobre la propia aplicación por lo que es necesario un método que permita elegir aquélla o aquéllas que mejor encajan en el contexto. Este es el caso de la mayoría de aplicaciones de RI, en las que el análisis léxico no es más que la primera fase del proceso y, por tanto, cualquier método capaz de descartar las alternativas menos probables quedándose únicamente con aquéllas más prometedoras, puede redundar en un ahorro considerable en las fases posteriores.

Resulta por tanto de especial interés poder ordenar las alternativas de corrección, en un caso para facilitar al usuario la elección de aquélla que considere más adecuada, y en

el otro para desechar aquellas correcciones que resulten menos prometedoras y ahorrar esfuerzo en fases de análisis posteriores.

En este capítulo describiremos algunas técnicas de corrección ortográfica contextual propuestas por otros autores para posteriormente detallar cómo el algoritmo de Viterbi [167] presentado en el capítulo 5 puede ser adaptado para conseguir una solución íntegra que nos permita manejar los tres tipos de ambigüedades que se describen a nivel léxico [117].

11.1 Resolución de la ambigüedad léxica

Al aplicar las técnicas de corrección ortográfica descritas en los capítulos 9 y 10 es frecuente obtener una salida ambigua, esto es, que el algoritmo retorne más de una corrección posible para una palabra errónea. Cuando esto sucede, dependiendo del tipo de aplicación, resultará más o menos interesante, y en algunos casos imprescindible, contar con un mecanismo que nos permita resolver esa ambigüedad. Esta problemática fue abordada por diferentes autores aplicando distintos métodos que podríamos clasificar según la información que utilizan para determinar el grado de idoneidad de cada una de las alternativas de reparación:

- *Información léxica*: Este tipo de métodos utilizan exclusivamente información relativa a la propia palabra. Es el caso del método propuesto por Pollock y Zamora [123] que utiliza dos criterios para graduar la idoneidad de las alternativas de corrección:
  - *Frecuencia relativa de las palabra*s: Esta frecuencia debe ser calculada previamente a partir de un *corpus* representativo, como el cociente entre el número de apariciones de cada palabra en el *corpus* y el tamaño de éste.
  - *Frecuencia del tipo de error*: Esta frecuencia se obtiene del estudio de cada uno de los cuatro tipos de errores de edición y nos permite ponderar el peso de cada una de las alternativas de corrección según el tipo de error que se haya producido en cada caso.

  La frecuencia del tipo de error resulta más fiable que la frecuencia relativa de las palabras por lo que la segunda suele utilizarse como criterio subsidiario para el caso en que exista coincidencia en la primera.

- Información morfo-sintáctica: La mayor parte de los métodos existentes utilizan información relativa al contexto morfo-sintáctico, esto es, a la forma y a la categoría sintáctica tanto de la palabra en cuestión como de las que la rodean. En concreto, podemos referirnos a propuestas que emplean trigramas de palabras [101], clasificadores bayesianos [50], listas de decisión [178], híbridos bayesianos [54], una combinación de trigramas de etiquetas morfosintácticas e híbridos bayesianos [55], aprendizaje basado en la transformación [96] o el algoritmo Winnow [53].

- Información semántica: Estas técnicas han surgido a partir de la aparición de tesauros como MeSH [113] o WordNet [14, 46, 63, 105, 106], que han permitido el desarrollo

de métodos basados en la *distancia semántica* entre términos [3, 69, 91, 125, 152]. Destacan en este ámbito los trabajos realizados por Budanitsky y Hirst [23, 24, 70] o Aguirre *et al.* [2, 3, 4]

En realidad, todos los métodos tratan de imitar lo que haría una persona a la que se le pide que corrija un texto escrito por otra y que, por lo tanto, desconoce lo que aquélla pretendía escribir. En esta línea, por ejemplo, probablemente la corrección más natural en la frase "*saltaré con la pertiga negra*", en que la palabra "*pertiga*" resulta ser errónea, consistiría en colocar una tilde sobre la 'e' obviando que existe otra alternativa que consistiría en substituir la '*t*' por una '*s*' obteniendo así "*persiga*", que sin embargo descartamos porque aunque de forma aislada es una opción, no parece la más adecuada en el conjunto de la frase. En realidad, de forma inconsciente, las personas aplicamos una combinación de los criterios propuestos por los distintos autores de modo que la gran mayoría de las personas a las que les pidiésemos que corrigieran la palabra "*pertiga*" en la frase anterior se darían cuenta rápidamente de lo siguiente:

- Es mucho más frecuente que una persona olvide poner una tilde que que confunda una '*t*' con una '*s*'.
- La palabra "*pértiga*" es un sustantivo mientras que "*persiga*" es un verbo y, al observar el contexto en el que ésta ocurre, es fácil advertir que está precedida por "*la*", que puede ser un determinante, y seguida de "*negra*" que es un adjetivo, lo que nos llevaría a vaticinar con un alto grado de certeza que un sustantivo encaja mejor que un verbo en esa posición.
- En la frase de ejemplo aparece la palabra "*saltaré*" que resulta mucho más fácil de asociar con la palabra "*pértiga*" que con la palabra "*persiga*". No en vano, es frecuente el uso de pértigas para saltar.

Así pues, lo ideal sería poder combinar la información que cada nivel de análisis puede ofrecernos al respecto del contexto de una palabra concreta, sin embargo, esto no siempre es posible. En nuestro caso nos centraremos en el manejo de la información morfosintáctica embebida en el etiquetador descrito en el capítulo 5.

11.2 El algoritmo Viterbi-L para manejar la ambigüedad léxica

Con el objetivo de diseñar una herramienta que integre de forma eficiente la funcionalidad necesaria para manejar la ambigüedad a nivel morfosintáctico, sea ésta del tipo que fuera, analizaremos ahora la adaptación necesaria para que el modelo propuesto en el capítulo 5, sea capaz también de lidiar con el problema de la ambigüedad léxica descrito en la sección anterior. Aquel modelo, además de resolver la ambigüedad presente en las palabras que pueden llevar distintas etiquetas, era capaz de resolver también situaciones en las que la falta de determinismo venía provocada por la posibilidad de segmentar la oración de más de una forma diferente, utilizando para ello una estructura común que permitiese evitar la repetición de cálculos en aquellas partes de la frase que no presentan ambigüedades.

En la sección 5.4 comprobamos cómo la escasa flexibilidad de los enrejados sobre los que suele aplicarse el algoritmo de Viterbi [167] no permitía representar ni manejar las ambigüedades de segmentación que pueden ocurrir durante el proceso de división de un texto en unidades léxicas. Para resolver este inconveniente proponíamos substituir la estructura de enrejado por la de retículo, sobre la que podíamos representar de forma cómoda e intuitiva las alternativas de segmentación, aunque para poder realizar los cálculos sobre esta nueva estructura era necesaria una ligera adaptación de las ecuaciones del algoritmo de Viterbi [167] original.

Podemos observar algunas similitudes entre la desambiguación segmental y la léxica. Mientras que la primera trata de encontrar la mejor alternativa de segmentación, la segunda intenta determinar cuál es la mejor alternativa de corrección, teniendo únicamente en cuenta en ambos casos la información morfosintáctica de las unidades léxicas adyacentes. Por lo tanto, los retículos parecen constituir una excelente estructura para manejar la ambigüedad léxica introducida por los algoritmos de corrección ortográfica.

Sin embargo, el modelo propuesto en la figura 5.7, para representar las alternativas de segmentación sobre un retículo no es directamente aplicable al problema que nos ocupa. En dicha figura, cada arco se etiquetaba únicamente con la información morfosintáctica correspondiente a las palabras que abarcaba, que permanecían inalterables en los huecos existentes entre los nodos origen y destino de cada arco. En el caso de la ambigüedad léxica, además de las alternativas de segmentación y de etiquetación, se hace necesario poder indicar alternativas de corrección para las propias palabras por lo que éstas deben ser representadas en los arcos. De este modo, las etiquetas de los arcos pasan ahora a ser pares *etiqueta/lema* y la ejecución del algoritmo de Viterbi [167] sobre un retículo de este tipo no sólo nos facilitaría la segmentación ideal de la frase y las etiquetas más probables de cada una de las unidades léxicas resultantes, sino también la alternativa de corrección que mejor encaja en el contexto de una posible palabra errónea.

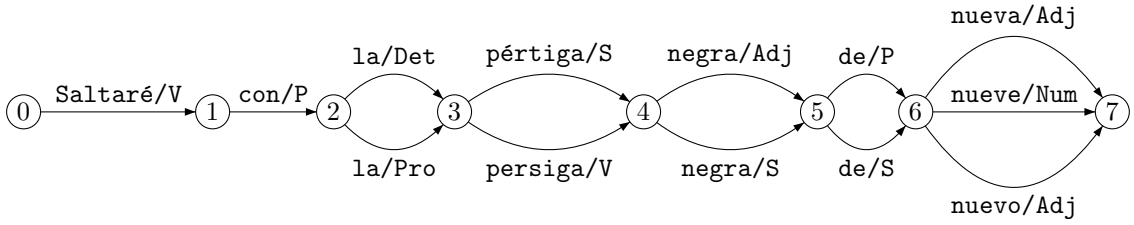

Figura 11.1: Retículo correspondiente a la frase "*Saltaré con la pertiga negra de nuevi*".

Para ilustrar la forma en que los tres tipos de ambigüedades pueden ser representados sobre una misma estructura, consideramos la frase "*Saltaré con la pertiga negra de nuevi*" en la que las palabras "*pertiga*" y "*nuevi*" son desconocidas. En la figura 11.1 se muestra el retículo correspondiente en el que, además de las ambigüedades de etiquetación se representan también las ofrecidas por el corrector ortográfico. En el primer caso, para la palabra "*pertiga*" el corrector encuentra dos alternativas de corrección de coste mínimo, "*pértiga*" y "*persiga*", que son representadas en el retículo de forma directa. Sin embargo,

el caso de la palabra "*nuevi*" es distinto. Al colocar las tres alternativas de corrección obtenidas sobre el retículo, *nuevo*, *nueve* y *nueva*, observamos que una de ellas puede ser combinada con la palabra anterior, en este caso "de", para formar la locución adverbial "de nuevo". Estas situaciones deben ser identificadas y resueltas colocando todas las combinaciones posibles sobre el retículo como puede verse en la figura 11.2.

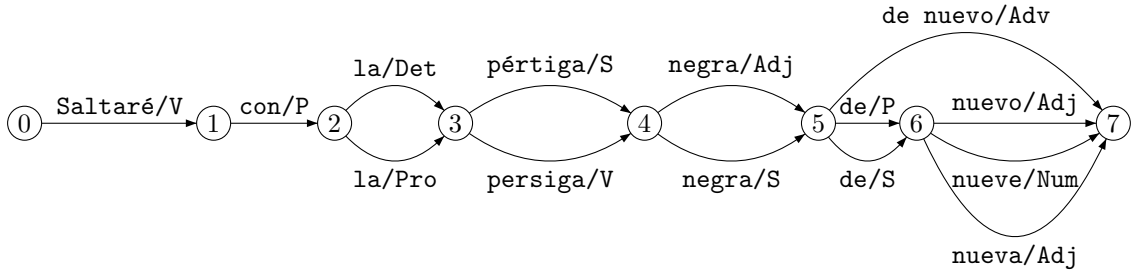

Figura 11.2: Retículo correspondiente a la frase "*Saltaré con la pertiga negra de nuevi*" combinando ambigüedad léxica y segmental.

Hasta este punto hemos supuesto que todas las palabras desconocidas provenían de errores ortográficos, lo cual constituye una suposición ciertamente optimista, ya que el simple hecho de que una palabra no aparezca en el lexicón es condición necesaria, pero no suficiente, para determinar que estamos ante un error ortográfico. Con el objetivo de no descartar la posibilidad de que una palabra, aún no estando en el lexicón, sea correcta, podemos incluir además de las alternativas propuestas por el corrector ortográfico la palabra original. Para esto es necesario la utilización de un *adivinador*[1] [21] que nos permita asignar una etiqueta a estas palabras. En la figura 11.3 se puede observar nuestro retículo de ejemplo incluyendo las palabras originales como alternativas con las etiquetas predichas por el adivinador.

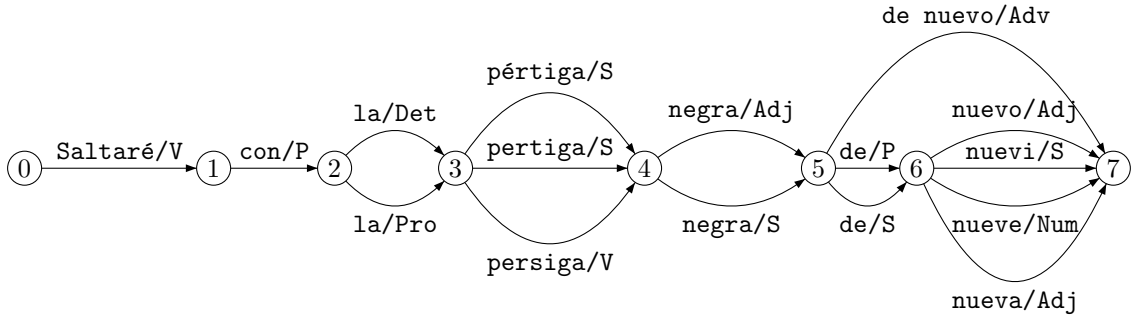

Figura 11.3: Retículo correspondiente a la frase "*Saltaré con la pertiga negra de nuevi*" incluyendo las palabras originales como alternativas.

[1]Resulta frecuente el uso del término en inglés *guesser*.

En definitiva, la flexibilidad que nos proporciona la estructura de retículo presentada en la sección 5.4 nos permite representar todos los tipos de ambigüedades que podemos encontrarnos a nivel léxico de forma que una única ejecución del algoritmo de Viterbi [167] nos proporcione de forma sencilla y eficiente la combinación de unidades léxicas más probables a partir de la información morfosintáctica embebida en el modelo probabilístico obtenido en una fase previa de entrenamiento.

Capítulo 12

Resultados experimentales

En este capítulo mostraremos y analizaremos los resultados de los experimentos realizados durante la realización de este trabajo de investigación. Se describen aquí los recursos lingüísticos utilizados y se realiza una exhaustiva comparación entre el método regional desarrollado en el capítulo 10 y el método global propuesto por Savary [140]. También se realizan experimentos para verificar el comportamiento de nuestra técnica de corrección contextual descrita en el capítulo 11 y, finalmente, se detalla una metodología de generación de errores ortográficos que nos permite comparar la aplicación de corrección ortográfica contextual, en sistemas de RI basados en extracción de raíces, con la indexación y la recuperación basadas en n-gramas de caracteres superpuestos, definiendo para ello un marco de evaluación adecuado.

12.1 Recursos lingüísticos

Para la realización de los experimentos se han utilizado diversos recursos lingüísticos que se describen a continuación.

12.1.1 El *corpus* Erial

El *proyecto Erial* [12], *Extracción y Recuperación de Información mediante Análisis Lingüístico*, consistió en la construcción de un entorno para recuperación de información basado en la utilización de técnicas de PLN. Por el carácter interdisciplinar del proyecto, el equipo de investigación estaba formado por miembros de la Universidad de A Coruña, que aportaba el conocimiento computacional, las Universidades de Santiago de Compostela y Vigo, y el Centro Ramón Piñeiro para la Investigación en Humanidades, que aportaban conocimiento lingüístico, la empresa Compaq, que proporcionaba equipamiento informático y asesoramiento técnico, y la Editorial Compostela, como usuario del sistema. El proyecto ha sido financiado, entre los años 1999 y 2001, por la Secretaría de Estado de Política Científica y Tecnológica (Fondos Europeos para el Desarrollo Regional, FEDER[1]).

[1] http://ec.europa.eu/regional_policy/funds/feder/index_es.htm

El lexicón del proyecto Erial correspondiente a la parte de español contiene 94.628 lemas con información morfológica, agrupados por categorías según se indica en la tabla 12.1, así como datos de subcategorización para 1.284 verbos[2].

La lista de lemas ha sido generada a partir de un lexicón de propósito general, construido de acuerdo con criterios lexicográficos a partir de diccionarios electrónicos y *corpora* cuya exhaustividad implica un alto índice de ambigüedad evitable [6]. Mediante la aplicación de filtros, basados principalmente en datos estadísticos y de uso real, se resolvieron a nivel léxico determinados problemas de ambigüedad.

	lemas	formas
sustantivos	60.264	131.380
adjetivos	22.450	110.236
verbos	11.914	363.699
adverbios		4.079
conjunciones		169
preposiciones		158
determinantes		525
pronombres		617

Tabla 12.1: El lexicón de Erial.

En lo que respecta al *corpus*, en el que fueron incluidos 27 documentos con aproximadamente 16.500 palabras, se ha eliminado la ambigüedad manualmente. En la etiquetación del mismo se empleó el conjunto completo de 918 etiquetas en el que se basa el lexicón, a partir del cual es posible derivar automáticamente subconjuntos menores más ajustados a los requisitos de una aplicación concreta.

12.1.2 El *corpus* Xiada

El *corpus* Xiada[3] [28] se encuentra en constante evolución. El objetivo principal de este proyecto es el desarrollo de herramientas que permitan el reconocimiento y el análisis automático del gallego actual. En particular, se centra en el desarrollo de un etiquetador y un lematizador de muy alta precisión que permita etiquetar y lematizar automáticamente los documentos del *proyecto* CORGA, y poder así desarrollar un sistema de consultas que utilice esta información lingüística (categorías gramaticales, lemas, etc.).

La primera versión operativa data del año 2003. Esta trabajaba con ficheros de texto que se adecuaban a la normativa, y para la que se desarrolló un juego de etiquetas apropiado, alrededor de 400 diferentes, de un lexicón formado por aproximadamente 31.200 lemas y 6.300.000 formas, y de un subconjunto anotado de entrenamiento de unas 100.000

[2]La información sintáctica contenida en el lexicón procede de la *Base de Datos Sintácticos del Español* (BDS), desarrollada por el Grupo de Sintaxis del Español de la USC y disponible en **http://www.bds.usc.es**.

[3]http://corpus.cirp.es/xiada/

	lemas	formas
sustantivos	11.706	23.346
adjetivos	3.700	5.169
verbos	6.789	544.690
adverbios		3.734
conjunciones		40
preposiciones		43
determinantes		347
pronombres		476

Tabla 12.2: El lexicón de Xiada.

formas. En este trabajo se ha utilizado un subconjunto de este lexicón cuya distribución por categorías se muestra en la tabla 12.2.

Durante 2006 se adaptó el etiquetador para que pudiese trabajar con ficheros codificados en XML y, por lo tanto, con los ficheros de la nueva codificación de los documentos del CORGA. También en este año se desarrolló un sistema genérico de resolución de ambigüedades segmentales, y se amplió el lexicón considerablemente, que ahora además incluye, por ejemplo, muchas formas no normativas para que puedan ser reconocidas.

12.1.3 El *corpus* ITU

Uno de los textos etiquetados y libremente disponibles es el *International Telecommunications Union* SSITT *Handbook*[4], también conocido como *The Blue Book* en el ámbito de las telecomunicaciones, y como *corpus* ITU en el ámbito lingüístico. Este *corpus* es la principal colección de textos sobre telecomunicaciones existente y, debido a su gran tamaño, constituye un excelente marco de pruebas.

El texto original no etiquetado tiene alrededor de 5 millones de palabras. En lo que se refiere al texto ya anotado, existen 2 versiones: el *corpus* entero etiquetado con la versión para español del etiquetador de XEROX [34, 138, 139], y un subconjunto de aproximadamente medio millón de palabras corregido a mano por la Universidad de Lancaster. Esta segunda versión es la que se ha utilizado como uno de los *corpora* de referencia en este trabajo, aunque ambos se pueden descargar de las URLs correspondientes a las páginas que soportan toda la información relativa al proyecto CRATER: *Corpus Resources And Terminology ExtRaction* [90].

Se describen a continuación las principales características de este *corpus*:

- Tiene 486.073 palabras. El tamaño del fichero es de 7.564.781 caracteres. La sintaxis de cada línea es la siguiente:

  `palabra/etiqueta/lema palabra/etiqueta/lema ... palabra/etiqueta/lema`

[4] http://www.itu.int/ITU-T/

El fichero contiene 14.919 líneas, cada una de las cuales se corresponde con una frase del *corpus*. Es decir, cada frase tiene un número medio de 22 palabras.

- El *corpus* contiene 45.679 formas verbales, donde 581 formas están en voz pasiva, 870 son formas verbales compuestas, y 3.415 son formas verbales con pronombres enclíticos.

- Las características del lexicón o diccionario constituido por todas las formas que aparecen en el *corpus* son las siguientes:

15.745 formas con 1 etiqueta,	1.097 formas con 2 etiquetas,
223 formas con 3 etiquetas,	61 formas con 4 etiquetas,
8 formas con 5 etiquetas,	3 formas con 6 etiquetas y
1 forma con 7 etiquetas.	

  Esto es, 17.138 formas, con 18.917 etiquetas posibles, y correspondientes a 12.462 lemas diferentes.

12.1.4 El *corpus* MULTEX-JOC

El *corpus* MULTEX-JOC [156] es en realidad una parte del *corpus* desarrollado en el proyecto MULTEX[5] financiado por la Comisión Europea. Los datos originales de este *corpus* se componen de preguntas realizadas por escrito, en una amplia variedad de temas, por los miembros del Parlamento Europeo y las correspondientes respuestas de la Comisión Europea en 9 versiones paralelas, publicadas como una sección de la serie C del Boletín Oficial de la Unión Europea del año 1993. El *corpus* está codificado en SGML, a nivel de párrafo, usando las especificaciones del estándar de codificación de *corpus ces* [6] y contiene alrededor de un millón de palabras para cada una de las cinco lenguas consideradas: inglés, alemán, francés, italiano y español. Para cada una ellas, excepto el alemán, hay etiquetado un subconjunto de 200.000 palabras, siguiendo las especificaciones desarrolladas en el proyecto.

12.2 Corrección regional frente a la global

Con el ánimo de obtener algunos resultados preliminares que nos permitieran evaluar nuestra propuesta de corrección regional, la comparamos con la propuesta global de Savary [140] intentando corroborar los resultados teóricos del capítulo 10. Creemos que este es un procedimiento objetivo para medir la calidad de un algoritmo de reparación, dado que el punto de referencia es una técnica que garantiza la mejor calidad para una métrica de error dada, cuando está disponible toda la información contextual.

[5] http://www.lpl.univ-aix.fr/projects/multext
[6] http://aune.lpl.univ-aix.fr/projects/multext/CES/CES1.html

12.2.1 Corrección sobre el lexicón de Erial

El español, por ser una lengua con una gran variedad de procesos morfológicos, resulta de especial interés para el estudio de la corrección ortográfica [163]. En particular, nuestras pruebas preliminares han sido ejecutadas sobre el español. Las características más notorias se asocian a los verbos que presentan un paradigma de conjugación de gran complejidad, así como en la inflexión de género y número.

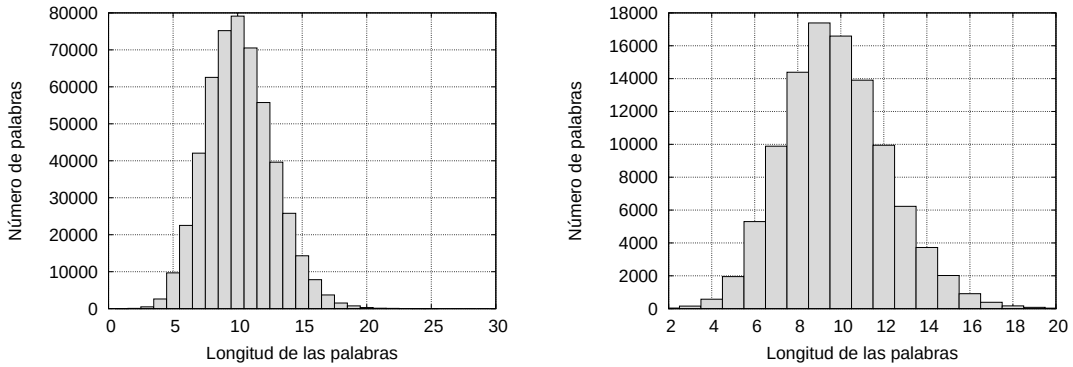

Figura 12.1: Distribución del lexicón Erial y el conjunto de pruebas según longitudes de las palabras.

Elegimos para trabajar con el español el lexicón del proyecto Erial, descrito en la sección 12.1.1. Al compilar este lexicón obtenemos un AF que contiene 58.149 estados conectados mediante 153.503 transiciones que forman 32.276 regiones, de tamaño suficiente para permitirnos considerar este AF como un punto de partida representativo para nuestros propósitos. De este lexicón, hemos seleccionado una muestra para la evaluación práctica del algoritmo con la misma distribución observada en el original, en términos de longitudes de palabras. Ello tiene cierta importancia dado que, como indican los autores, la eficiencia de propuestas previas depende de esos factores [116, 140]. No se han detectado otras dependencias a nivel morfológico y, por lo tanto, no se han considerado. Los errores se han generado aleatoriamente en número y posición en la cadena de entrada para las distintas longitudes.

Nos centramos aquí en la evaluación tanto de aspectos computacionales como de calidad. Para tener en cuenta datos relacionados con el rendimiento tanto desde el punto de vista del usuario como del sistema, introducimos las medidas de *rendimiento* y *cobertura*[7], para una palabra dada, w, que contiene un error. Las fórmulas utilizadas para el cálculo de estos dos parámetros se muestran en las ecuaciones 12.1 y 12.2 respectivamente, pero de forma intuitiva podemos decir que el rendimiento nos indica qué proporción de los cálculos realizados han sido finalmente de utilidad, mientras que la cobertura nos da el *ratio* de correcciones encontradas frente al número total de ellas que en realidad existían.

$$rendimiento(w) = \frac{\textit{ítems útiles}}{\textit{ítems totales}} \tag{12.1}$$

[7]Es muy frecuente la utilización del término en inglés *recall*.

$$cobertura(w) = \frac{correcciones\ propuestas}{correcciones\ totales} \tag{12.2}$$

Allí donde sea necesario medir el coste computacional, dado que nuestro sistema evoluciona aplicando reglas de deducción que generan nuevos ítems, utilizaremos como unidad de medida el concepto de ítem definido en la sección 10.1. En este sentido, hablamos de *ítems útiles* para referirnos al número de ítems generados que finalmente contribuyen a la obtención de una reparación, y de *ítems totales* para indicar el número de esas estructuras generadas durante el proceso. Por otra parte, denotamos por *correcciones propuestas* el número de correcciones que ofrece el algoritmo, y por *correcciones totales* el número de correcciones posibles, en términos absolutos.

Como medida complementaria del rendimiento y la cobertura, ofrecemos una medida global de la *precisión* de la reparación de error en cada caso, esto es, el *ratio* en que el algoritmo ofrece la corrección esperada por el usuario.

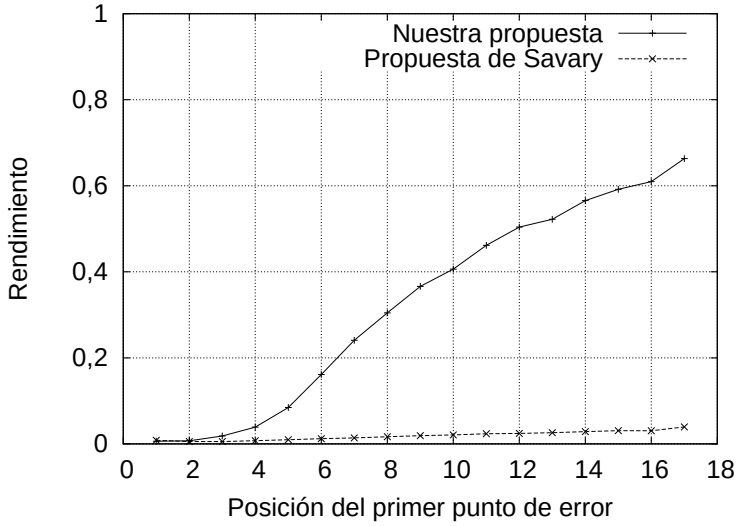

Figura 12.2: Gráfica de rendimiento sobre el lexicón de Erial.

Los resultados prácticos en cuanto a rendimiento y cobertura mostrados en las figuras 12.2 y 12.3 parecen corroborar que no sólo el rendimiento es mejor en el caso de la corrección regional que en el de Savary [140], sino también que la diferencia entre ambos crece con la localización del primer punto de error. Cuanto mayor sea el prefijo ya reconocido en el momento en que se detecta el error, mayor diferencia hay entre el método regional y el global en cuanto a rendimiento. Esto es así porque el método regional evita, en la medida de lo posible, aplicar operaciones de corrección en el prefijo ya reconocido mientras que el global se emplea de igual modo en todas las posiciones de la cadena aún cuando no exista error alguno en dichas posiciones.

Con respecto a la relación entre la cobertura y la posición del primer punto de error, suponemos que las correcciones que buscamos son sólo aquéllas de coste mínimo por lo que el algoritmo de Savary muestra un gráfico constante dado que la aproximación es global y, en consecuencia, el conjunto de correcciones ofrecidas contiene exactamente todas las

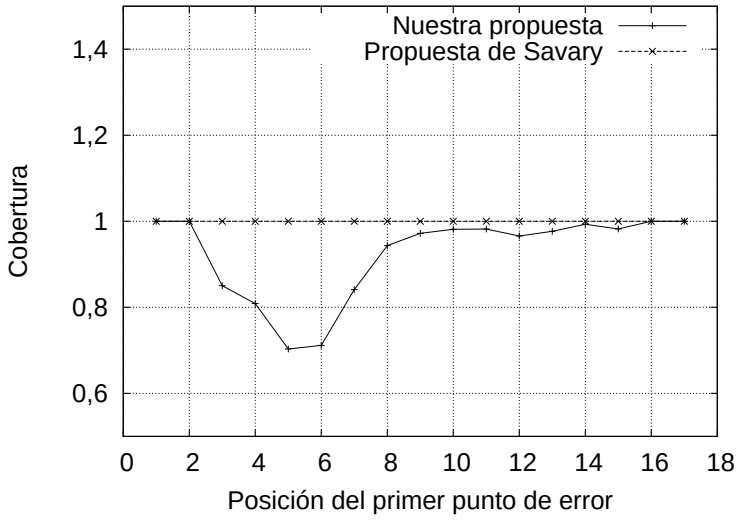

Figura 12.3: Gráfica de cobertura sobre el lexicón de Erial.

posibles. En nuestra propuesta, los resultados prueban que la cobertura es menor que para la de Savary, lo cual ilustra la ganancia en eficiencia computacional en comparación con el método global.

Otro aspecto a tener en cuenta es el *ratio* de acierto del método a lo que llamamos precisión, que para el caso regional es del 77 %, frente al 81 % del global. En este sentido, se debe recordar que sólo tenemos en cuenta información morfológica, que tiene impacto en la precisión para una aproximación regional, pero no para una global, que siempre ofrece todas las alternativas de reparación posibles. Por tanto, la medida de precisión representa una desventaja para nuestra propuesta dado que basamos la eficiencia en la limitación de espacio de búsqueda. Sería de esperar que la integración de información lingüística, tanto desde el punto de vista semántico como sintáctico, debería reducir de forma significativa esta diferencia de precisión, que es menor que el 4 %, o podría incluso eliminarlo.

Proponemos además dos aproximaciones complementarias que tratan de medir el esfuerzo computacional necesario para obtener una corrección, teniendo en cuenta la posición de la palabra en la que se detecta el primer error. Utilizaremos, por tanto, una vez más, el concepto de ítem como unidad de medida. Así, ilustraremos primeramente cómo varía el número de ítems generados respecto del grado de penetración en el AF, esto es, de la posición del primer punto de error. En efecto, al trabajar con métodos regionales, el grado de penetración en el AF determina el número de regiones que contienen el punto de error y, por lo tanto, la posibilidad de que el método regional no acabe siendo equivalente al global. Y por otra parte, resulta también interesante evaluar los resultados en base a la longitud del sufijo que resta por reconocer en el momento en que se detecta el primer error.

En los gráficos de las figuras 12.4 y 12.5 se muestra la relación entre el número de ítems generados en modo error por cada uno de los algoritmos y la posición del primer punto de error, en el primer caso, o la longitud del sufijo pendiente de analizar en el segundo. Se

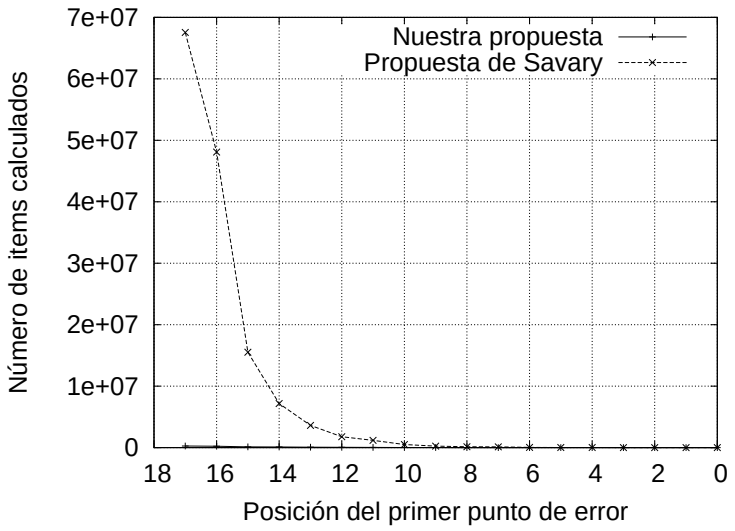

Figura 12.4: Número de ítems generados según la posición del primer punto de error.

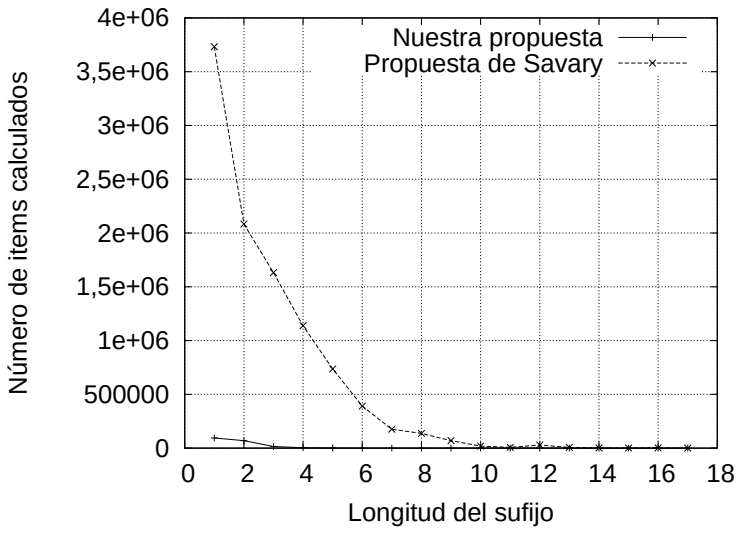

Figura 12.5: Número de ítems generados según la longitud del sufijo pendiente de reconocer.

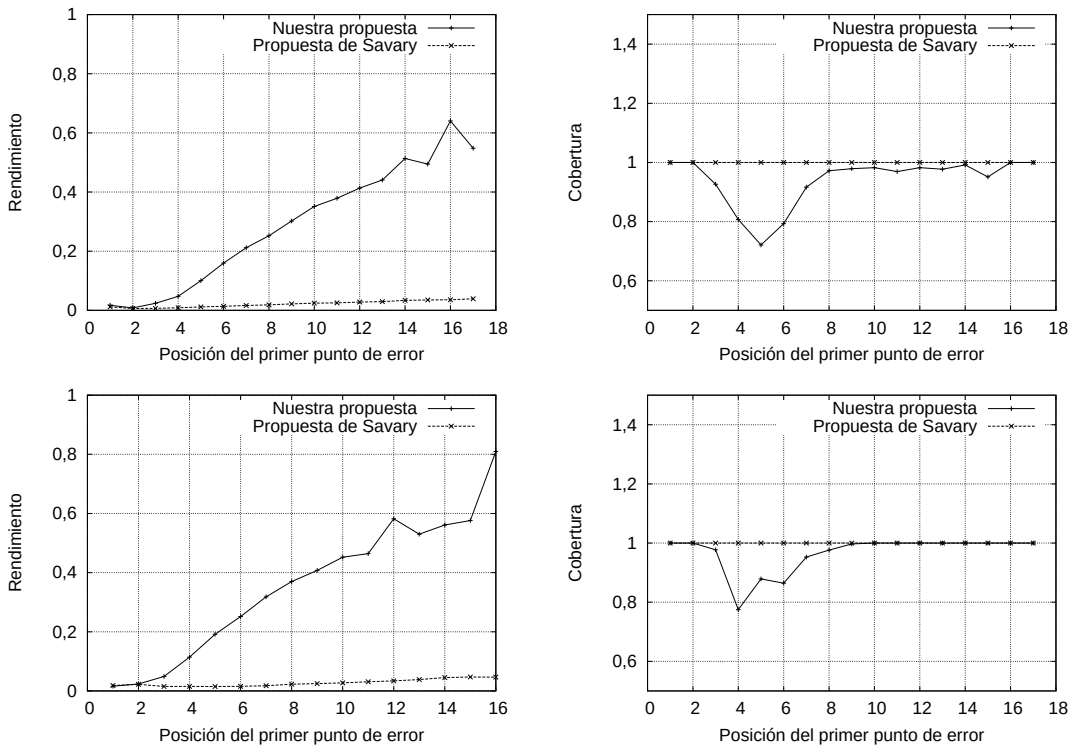

Figura 12.6: Gráficas de rendimiento y cobertura sobre los lexicones de los *corpora* Xiada (arriba) e ITU (abajo).

observa como, mientras que el método regional apenas depende de este factor, en el caso del global el número de ítems crece de forma exponencial con la posición del primer error detectado. Esto viene a corroborar el hecho de que el método regional intenta realizar las reparaciones en las posiciones cercanas al punto de error mientras que el global se emplea con igual intensidad en todas las posiciones de la cadena de entrada.

12.2.2 Corrección sobre los lexicones Xiada e ITU

Con el fin de contrastar los resultados obtenidos para el español, se realizaron experimentos similares [160, 164] sobre otros dos lexicones:

- Xiada[8] [28], del *corpus* del mismo nombre, descrito en la sección 12.1.2 que genera un AF que contiene 16.837 estados conectados mediante 43.446 transiciones que forman 8.408 regiones.

- ITU, del *corpus* del mismo nombre, descrito en la sección 12.1.3 que genera un AF que contiene 11.193 estados conectados mediante 23.278 transiciones que forman 4.159 regiones.

[8] http://corpus.cirp.es/xiada/

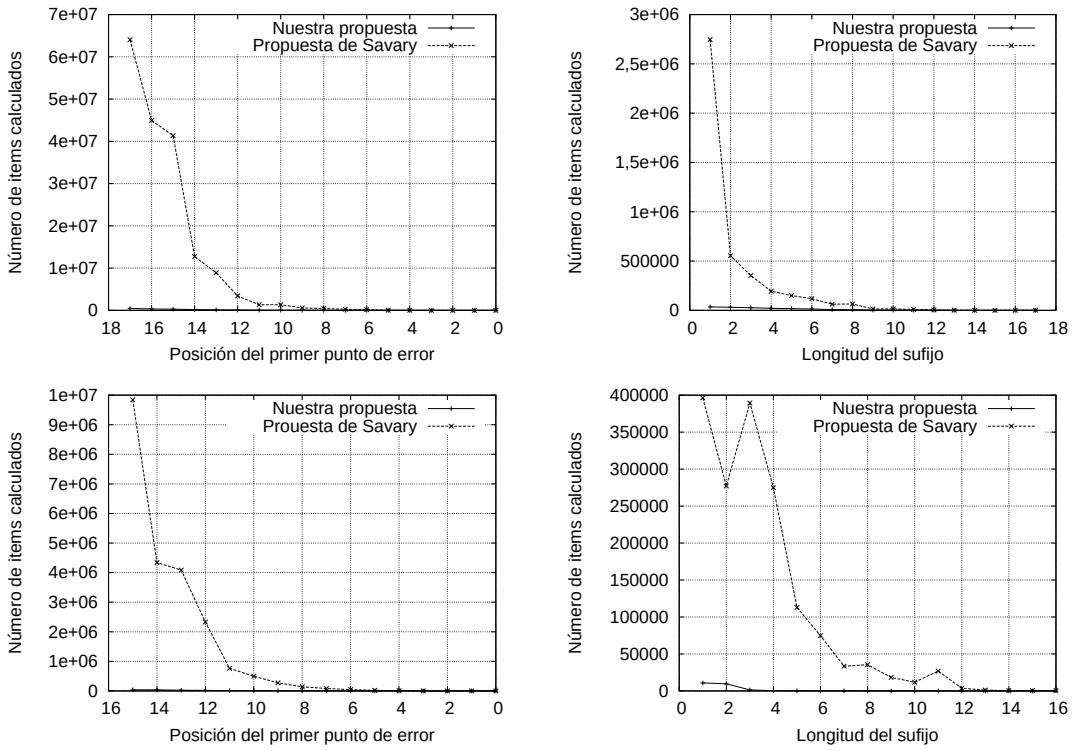

Figura 12.7: Número de ítems generados sobre los lexicones de los *corpora* Xiada (arriba) e ITU (abajo): según la posición del primer punto de error y según la longitud del sufijo pendiente de reconocer.

Se trata de dos lexicones similares al utilizado en los primeros experimentos, pero que presentan características interesantes desde el punto de vista de la corrección ortográfica.

El *corpus* Xiada [28] es un *corpus* para gallego que, aunque tiene un léxico cercano al español, presenta también diferencias morfológicas que podrían incidir sobre los resultados obtenidos. Una de las diferencias más significativas es la utilización de pronombres enclíticos que, además de resultar mucho más frecuente en gallego, son de utilización mayoritariamente forzosa, en contra de lo que ocurre en español donde su uso suele ser opcional[9]. Por otra parte, en gallego puede llegarse hasta cuatro enclíticos, lo que provoca una gran cantidad de sufijos posibles para las formas verbales susceptibles de presentar esta característica.

El *corpus* ITU pertenece al ámbito de las telecomunicaciones, por lo que el léxico presenta gran cantidad de palabras técnicas que nos permite probar nuestra técnica en un ámbito diferente del de los experimentos iniciales.

No se detectan grandes diferencias respecto a los resultados obtenidos para español, lo cual confirma la versatilidad del sistema ya que su comportamiento parece independiente del lexicón utilizado. En la figura 12.6 se muestran los resultados en términos de

[9]Por ejemplo, en lugar de decir "*dáselo*", podríamos decir "*se lo das*".

rendimiento y cobertura mientras que en la figura 12.7 se puede observar la relación entre el número de ítems generados en modo error por cada uno de los algoritmos y la posición del primer punto de error, en el primer caso, o la longitud del sufijo pendiente de analizar en el segundo.

12.3 Corrección ortográfica contextual

El experimento descrito en esta sección trata de evaluar el funcionamiento del método de corrección ortográfica contextual descrito en la sección 11.2. Se parte de la idea de que un menor número de alternativas de corrección aumenta las posibilidades de nuestro corrector contextual de elegir la palabra correcta. Bajo esta premisa compararemos el rendimiento ofrecido por el sistema cuando se varía el método de corrección ortográfica utilizado, teniendo en cuenta que nuestro método regional ofrece un menor número de correcciones candidatas.

Consideramos que existen 2 factores determinantes a la hora de conseguir una elección exitosa. En primer lugar, tenemos lo que podríamos llamar el *ratio* de acierto del algoritmo de corrección, que sería el porcentaje de casos en que la palabra correcta se encuentra en la lista de alternativas ofrecida por el sistema. Este supone el primer filtro, ya que para que la palabra elegida tras la ejecución del algoritmo de Viterbi [167] sobre el retículo que representa todas las alternativas de reparación coincida con la esperada, es condición imprescindible que ésta se encuentre entre las posibilidades ofrecidas por el corrector. El segundo filtro es el que supone la ejecución del propio algoritmo de Viterbi sobre el retículo ya que, aún estando la palabra correcta entre las candidatas, nuestro método contextual podría no elegirla.

Con estas consideraciones, se han realizado dos experimentos para los que se ha utilizado un subconjunto de frases del *corpus* CLiC-TALP [141] para el español, que consta de 100.000 palabras anotadas manualmente a nivel morfosintáctico, sobre el que se han introducido errores ortográficos aleatorios, similares a los que cometería un humano, por medio de la herramienta Misplel [15]. Esto nos permite saber cuál era la palabra correcta original para cada palabra errónea y, por tanto, podemos evaluar el porcentaje de acierto del método.

Teniendo en cuenta el *ratio* de acierto de los algoritmos de corrección podemos considerar la siguiente casuística:

- $\emptyset$: La palabra correcta no aparece en la lista de alternativas ofrecidas por ninguno de los dos métodos.
- R: La palabra correcta sólo aparece en la lista de alternativas ofrecidas por el método regional.
- G: La palabra correcta sólo aparece en la lista de alternativas ofrecidas por el método global.
- GR: La palabra correcta aparece en la lista de alternativas ofrecidas por ambos métodos.

Caso	Num. frases	Regional	Global
∅	120	0	0
R	1	1	0
G	43	0	37
GR	3074	2899	2892
TOTAL	3238	2900	2929

Tabla 12.3: Resultados para la corrección contextual con 1 error por frase.

En un primer conjunto de pruebas, hemos introducido exactamente 1 error por frase. En este caso, el número medio de caminos diferentes por frase existentes en el retículo generado es aproximadamente de 5 para ambos métodos. Los resultados obtenidos se detallan en la tabla 12.3 en la que se indica el número de frases que se encontrarían en cada uno de los casos descritos anteriormente y cuántas de éstas serían recuperadas por completo mediante el uso de cada uno de los métodos de corrección ortográfica. En general, podemos decir que el método global ofrece una precisión ligeramente superior al recuperar 2.929 frases de las 3.238 sobre las que se realizó el experimento, lo que supone un *ratio* de acierto del $90,46\,\%$, frente al $89,56\,\%$ que arrojan las 2.900 frases recuperadas por el regional.

Si analizamos los dos factores que influyen en la precisión global del algoritmo, podemos observar que en lo que respecta al *ratio* de acierto del algoritmo de corrección, el método global incluye la palabra correcta en la lista de correcciones ofrecidas en un $96,26\,\%$ de los casos, mientras que el regional lo hace en un $94,97\,\%$.

Sin embargo, en lo que respecta a la precisión del algoritmo de Viterbi [167], observamos el comportamiento contrario ya que, cuando la palabra correcta se encuentra en la lista de alternativas ofrecidas por el corrector, el algoritmo acaba eligiéndola en el $94,31\,\%$ de los casos cuando utilizamos el método regional, frente al $93,97\,\%$ que alcanza el global. Esto se debe a que el algoritmo global retorna en media un mayor número de alternativas, lo que hace disminuir las probabilidades de que el algoritmo de Viterbi acabe eligiendo la correcta.

En un segundo conjunto de pruebas, establecemos un máximo de 3 errores por frase para evaluar el impacto de la densidad de éstos sobre el algoritmo. Concretamente, el número medio de errores por frase es de 2,21. Lo primero que detectamos es que el tiempo de ejecución de la prueba se incrementa de forma muy notable para ambos métodos, pero especialmente para el global. Ello es debido a que, si suponemos una media de 3 alternativas para cada palabra errónea, un error en una frase implicaría multiplicar por 3 el número de caminos en el retículo, 2 errores implicaría multiplicar por 9, 3 errores por 27, y así sucesivamente de forma exponencial. Esto hace que un pequeño incremento en el número de alternativas provoque un importante aumento de caminos en el retículo, o lo que es lo mismo un coste computacional mucho mayor. En consecuencia, mientras que con un error por frase apenas se notaba diferencia en el número de caminos generados por ambos métodos, en este caso tenemos una media de 89 caminos por frase para el método regional y 108 para el global. Aunque en media puede parecer una diferencia despreciable, si tenemos en cuenta que nuestro conjunto de pruebas constaba de 3.280 frases, supone

Caso	Num. frases	Regional	Global
∅	260	0	0
R	0	0	0
G	123	0	95
GR	2897	2579	2571
TOTAL	3280	2579	2656

Tabla 12.4: Resultados para la corrección contextual con un máximo de 3 errores por frase.

un incremento en términos absolutos de 61.474 caminos.

En lo que respecta a la precisión, los resultados obtenidos por palabra no difieren prácticamente de los obtenidos para la primera prueba realizada. El porcentaje de frases completamente recuperadas desciende hasta un todavía respetable $80,98\,\%$ para el método global y un $78{,}63\,\%$ para el regional, si bien esto era de esperar ya que al aumentar el número de errores por frase aumenta también el número de caminos del retículo y ello hace descender la probabilidad de que el algoritmo de Viterbi [167] elija la palabra correcta. Estos resultados se muestran en la tabla 12.4.

En conclusión, el algoritmo global ofrece un rendimiento ligeramente superior desde el punto de vista de la precisión, es decir, el algoritmo de corrección contextual acaba eligiendo la palabra correcta en un mayor número de ocasiones. Sin embargo, esta pequeña mejoría se obtiene a costa de un importante incremento del coste computacional del algoritmo. En este contexto, los esfuerzos en este campo deben centrarse en el diseño de estrategias que permitan decidir cuando es preferible aplicar nuestra técnica de reparación regional, manteniendo el rendimiento a un nivel similar a la global, pero reduciendo el número de alternativas de corrección.

12.4 Aplicación de la corrección ortográfica a la recuperación de información

Con el objetivo de probar la utilidad de la corrección ortográfica en RI, incluimos una serie de test experimentales. Nos centramos en concreto en el tratamiento de consultas degradadas en español [119] aplicando tres técnicas diferentes y comparando los resultados obtenidos para cada una de ellas. La primera es una técnica basada en n-gramas que no necesita recursos lingüísticos extra y que ha sido introducida en la sección 6.5. La segunda consiste en aplicar el algoritmo de Savary [140] en las palabras desconocidas y expandir la consulta con todas las alternativas de corrección obtenidas. Finalmente, se utiliza la estrategia de corrección ortográfica contextual presentada en la sección 11 para elegir de entre las alternativas de reparación aquélla que mejor encaja en el entorno de la palabra errónea.

Los modelos clásicos de RI no contemplan, inicialmente, el caso de degradación en las consultas del usuario tales como la introducción de errores ortográficos o palabras desconocidas, bien sea de forma accidental, o porque el término que se está tratando de buscar presenta ambigüedades ortográficas en la colección documental. Se hace, por tanto,

necesario estudiar este problema ya que puede afectar de forma notoria el rendimiento del sistema.

En este sentido, muchos autores aplican directamente técnicas de corrección de errores en las formas léxicas de la consulta para así dotar a la herramienta de cierta robustez. Esta estrategia es a menudo empleada para el análisis de textos degradados en el ámbito del PLN. Sin embargo, si bien las herramientas de PLN suelen tolerar una fase de corrección poco eficiente en la que se interactúa con el usuario mostrándole múltiples alternativas de corrección para que sea éste el que realice la elección final, esto no es habitual en los sistemas de RI, lo que incrementa la complejidad del problema.

Por otra parte, las aproximaciones de corrección ortográfica [140, 162] aplican modificaciones en las palabras con el fin de minimizar la *distancia de edición* [92].

Trabajos más recientes interpretan la corrección ortográfica como una cuestión estadística, donde una consulta con errores es vista como una degeneración probabilística de una correcta [22]. Esta aproximación, conocida como *modelo de canal ruidoso*[10] [80], también proporciona formas de incorporar información de pronunciación para mejorar el rendimiento por medio de la captura de similaridades en la pronunciación de las palabras [153].

12.4.1 Recuperación de texto mediante n-gramas de caracteres

La utilización de n-gramas de caracteres superpuestos para la indexación y la recuperación, introducida en la sección 6.5, supone un prometedor punto de partida sobre el que desarrollar una estrategia de indexación y recuperación efectiva para el tratamiento de consultas degradadas. Además, la utilización de índices basados en n-gramas parece desmontar el principal argumento que justifica la integración de métodos de corrección ortográfica en aplicaciones de RI robustas: la necesidad de una coincidencia exacta con los términos almacenados en los índices. Esto nos permite prescindir de la disponibilidad de un diccionario. De este modo, con el empleo de n-gramas en lugar de palabras completas, sólo se requeriría la coincidencia en subcadenas de éstas. En la práctica, esto elimina la necesidad de normalizar los términos, minimizando además el impacto de los errores ortográficos, a los que no se les prestaría especial atención. En general debería, además, reducir de forma considerable la incapacidad del sistema para manejar las palabras desconocidas.

12.4.2 Corrección ortográfica

Introducimos también una aproximación más clásica asociada a nuestro corrector ortográfico contextual [117], lo que nos permite definir un marco de pruebas comparativo. En un principio aplicamos el algoritmo global de corrección ortográfica sobre AFs, propuesto por Savary [140], que encuentra todas las palabras cuya distancia de edición con la palabra errónea sea mínima.

Desafortunadamente esta técnica puede devolver varias reparaciones candidatas posibles que, desde un punto de vista morfológico, tengan una calidad similar, es decir,

[10] *Noisy channel model* en inglés.

cuando existan varias palabras cuya distancia de edición con la errónea es la misma.

Sin embargo, es posible ir más allá de la propuesta de Savary aplicando corrección ortográfica contextual que aproveche la información lingüística, embebida en un proceso de etiquetación con el fin de ordenar las correcciones candidatas.

12.4.3 Evaluación

La primera fase en el proceso de evaluación consiste en introducir errores ortográficos en el conjunto de consultas de prueba. Estos errores son introducidos de forma aleatoria por un generador de errores automático de acuerdo con un *ratio* de error dado. Inicialmente se genera un *fichero maestro de errores* como sigue. Para cada palabra de más de 3 caracteres de la consulta, se introduce en una posición aleatoria uno de los cuatro errores de edición descritos por Damerau [36]. De este modo, los errores introducidos son similares a aquéllos que cometería un ser humano o un dispositivo de ROC. Al mismo tiempo se genera un valor aleatorio entre 0 y 100 que representa la probabilidad de que la palabra no contenga ningún error ortográfico. Esto nos permite obtener un fichero maestro de errores que contiene, para cada palabra, su forma errónea correspondiente, y un valor de probabilidad asociado.

Todos estos datos hacen posible generar de una forma sencilla conjuntos de prueba diferentes para distintos *ratios* de error, permitiéndonos así valorar el impacto de esta variable en los resultados. El procedimiento consiste en recorrer el fichero maestro de errores y seleccionar, para cada palabra, la forma original en el caso de que su probabilidad sea mayor que el *ratio* de error fijado, o la forma errónea en caso contrario. Así, dado un *ratio* de error T, sólo el $T\,\%$ de las palabras de las consultas contendrán un error. Una característica interesante de esta solución es que los errores son incrementales, ya que las formas erróneas que están presentes para un *ratio* de error determinado continuarán estando presentes para valores mayores, evitando así cualquier distorsión en los resultados.

El siguiente paso consiste en procesar las consultas con errores. En el caso de nuestra propuesta basada en n-gramas no se precisan recursos extra, ya que el único procesamiento necesario consiste en dividir las consultas en n-gramas. Sin embargo, para las aproximaciones de corrección ortográfica se necesita un lexicón y, en el caso de la corrección contextual, también un *corpus* de entrenamiento revisado manualmente para entrenar el etiquetador. En nuestros experimentos hemos trabajado con el *corpus* de español MULTEX-JOC [156] descrito en la sección 12.1.4.

Por otra parte, en lo que respecta al sistema de RI, se ha empleado el *corpus* de español de la *robust task* del CLEF 2006 [112][11], formado por 454.045 documentos (1,06 GB) y 160 *topics* —a partir de los cuales generar las consultas— de los que hemos empleado únicamente un subconjunto del mismo (*training topics*) formado por 60 *topics* proporcionados por el CLEF específicamente para tareas de entrenamiento y puesta a punto.[12] Dichos *topics* están formados por tres campos:

- *título*, un breve título como su nombre indica.

[11] Estos experimentos han de considerarse no oficiales, ya que los resultados no han sido evaluados por la organización.

[12] *Topics* C050-C059, C070-C079, C100-C109, C120-C129, C150-159 y C180-189.

- *descripción*, una somera frase de descripción.
- *narrativa*, un breve texto especificando los criterios de relevancia.

En cualquier caso únicamente hemos empleado el campo de *título* para así simular el supuesto de las consultas cortas utilizadas en motores comerciales.

Partiendo de dicha colección de documentos se han generado dos índices diferentes. Primeramente, para probar las propuestas basadas en corrección ortográfica, se ha usado una aproximación clásica basada en *extracción de raíces*[13] empleando SNOWBALL[14], basado en el algoritmo de Porter [124], y la lista de *palabras vacías*[15] de la Universidad de Neuchatel[16]. Ambos recursos son ampliamente utilizados entre la comunidad de RI. Asimismo, en el caso de las consultas, se ha utilizado una segunda lista de *meta-palabras vacías*[17] [107, 108]. Dichas *palabras vacías* corresponden a metacontenido, es decir, expresiones de formulación de la consulta que no aportan ninguna información útil para la búsqueda, como en el caso de la expresión

"encuentre aquellos documentos que describan ..."

En segundo lugar, a la hora de probar nuestra solución basada en n-gramas, los documentos han sido convertidos a minúsculas y se han eliminado los signos de puntuación, aunque no los signos ortográficos. El texto resultante ha sido dividido e indexado utilizando 4-gramas como longitud de compromiso tras estudiar los resultados previos del JHU/APL [102]. En este caso no se han empleado *palabras vacías*.

Finalmente, ya a nivel de implementación, nuestro sistema emplea como motor de recuperación la plataforma de código abierto TERRIER [120] con un modelo InL2[18] [7].

Nuestra propuesta ha sido probada para un amplio rango de *ratios* de error T con el fin de estudiar el comportamiento del sistema no sólo para densidades de error bajas, sino también para elevadas, propias de entornos ruidosos como aquéllos en los que la entrada se obtiene de dispositivos móviles o basados en escritura a mano —PDAs y tabletas digitalizadoras, por ejemplo. De este modo se ha trabajado con:

$$T \in \{0\,\%, 10\,\%, 20\,\%, 30\,\%, \ldots, 100\,\%\}$$

donde T=0 % significa que no se han introducido errores.

En el primer conjunto de experimentos realizados se utilizaron las consultas sin corregir aplicando una aproximación clásica basada en *extracción de raíces*. Los resultados obtenidos para cada *ratio* de error T se muestran en las gráficas de la figura 12.8 tomando como referencia tanto los resultados obtenidos para las consultas originales aplicando *extracción de raíces* —es decir, para T=0 %— (*stm-noerr*), como los obtenidos aplicando la aproximación basada en n-gramas (*4gr-noerr*). También se dan los valores de precisión media (MAP[19]). Estos primeros resultados muestran que la *extracción de raíces* es sensible

[13] Usualmente denominada por el término en inglés *stemming*.
[14] `http://snowball.tartarus.org`
[15] Usualmente denominada por el término en inglés *stopwords*.
[16] `http://www.unine.ch/info/clef/`
[17] Usualmente denominadas por el término en inglés *meta-stopwords*.
[18] *Inverse Document Frequency model with Laplace after-effect and normalization 2.*
[19] *Mean average precision* en inglés.

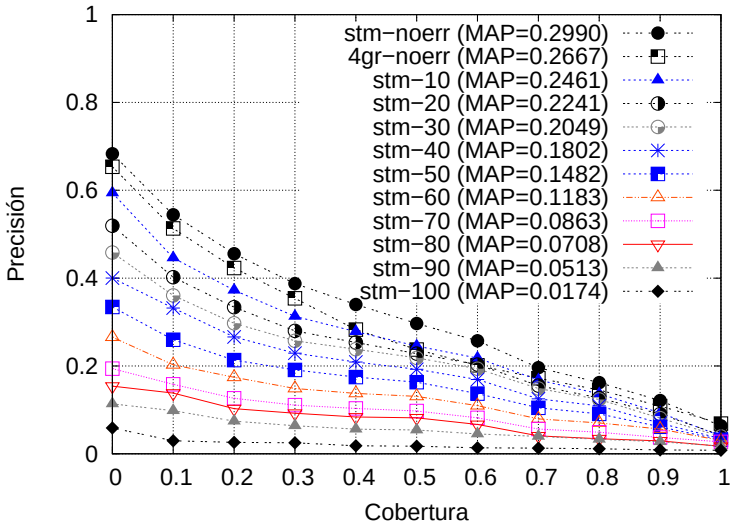

Figura 12.8: Precisión vs. cobertura para las consultas sin corregir (empleando *extracción de raíces*).

a los errores ortográficos. Como se puede apreciar, aún un *ratio* de error bajo como T=10 % tiene un impacto significativo sobre el rendimiento[20] —la MAP decrece el 18 %—, empeorando conforme aumenta el número de errores introducidos: pérdida del 25 % para T=20 %, 50 % para T=50 % (con 2 consultas que ya no recuperan ningún documento) y 94 % para T=100 % (con 13 consultas sin documentos), por ejemplo. Esto se debe al hecho de que con el tipo de consultas que estamos utilizando aquí —con unas 4 palabras de media—, cada palabra es de vital importancia, ya que la información perdida cuando un término ya no encuentra correspondencia debido a un error ortográfico no puede ser recuperada a partir de ningún otro término.

En nuestra segunda ronda de experimentos se estudió el comportamiento del sistema al usar la primera de las aproximaciones de corrección consideradas en este trabajo, esto es, cuando lanzamos las consultas con errores tras ser procesadas con el algoritmo de Savary [140]. En este caso el módulo de corrección toma como entrada la consulta con errores, obteniendo como salida una versión corregida donde cada palabra incorrecta ha sido sustituida por el término más cercano del lexicón de acuerdo a la distancia de edición [36, 92]. En caso de empate —es decir, cuando existen varias palabras en el lexicón a la misma distancia— la consulta es expandida con todas las correcciones empatadas. Por ejemplo, tomando como entrada la oración *"No es fácile trabajar baio presión"*, la salida sería *"No es fácil fáciles trabajar bajo baño presión"*. Analizando los resultados obtenidos, mostrados en la figura 12.9, vemos que la corrección tiene un efecto general significativamente positivo sobre el rendimiento, disminuyendo en gran medida —aunque no eliminando— el impacto de los errores ortográficos, no sólo para *ratios* de error bajos[21],

[20] A lo largo de este trabajo se han empleado tests-t bilaterales sobre las MAP con α=0.05.
[21] La pérdida de MAP disminuye del 18 % al 13 % para T=10 % y del 25 % al 15 % para T=20 %.

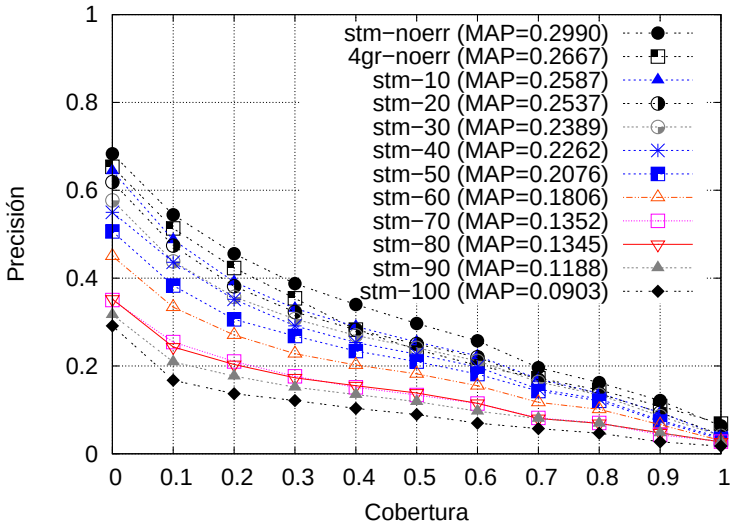

Figura 12.9: Precisión vs. cobertura para las consultas corregidas mediante el algoritmo de Savary [140] (empleando *extracción de raíces*).

sino también para los altos y muy altos[22], reduciéndose también el número de consultas que no devuelven documentos[23]. Asimismo, el análisis de los datos muestra que la efectividad relativa de la corrección aumenta con el *ratio* de error.

Con el fin de eliminar el ruido introducido por los empates al emplear el algoritmo de Savary [140], se ha realizado un tercer conjunto de pruebas usando nuestro corrector ortográfico contextual [117]. Dichos resultados se muestran en la figura 12.10 y, como era de esperar, éstos mejoran consistentemente con respecto a la aproximación original, si bien la mejora obtenida mediante este procesamiento extra no llega a ser significativa: un 2 % de pérdida de MAP recuperado para 10 %$\leq T \leq$60 % y un 7–10 % para $T >$60 %.

Finalmente, hemos probado nuestra propuesta basada en n-gramas. La figura 12.11 muestra los resultados obtenidos cuando las consultas sin corregir son lanzadas contra nuestro sistema de RI basado en n-gramas.

Aunque la *extracción de raíces* funciona significativamente mejor que los n-gramas para las consultas originales, no ocurre lo mismo cuando hay errores ortográficos, superando claramente el segundo método al primero no sólo cuando no se aplica ningún tipo de corrección, siendo la mejora significativa para $T \geq$40 %, sino también cuando se aplica cualquiera de los dos métodos basados en corrección ortográfica, si bien la diferencia no es significativa hasta $T \geq$70 %.

Además, la robustez de nuestra propuesta basada en n-gramas en presencia de errores ortográficos demuestra ser claramente superior a cualquiera de las aproximaciones previas basadas en *extracción de raíces*.

Como ejemplo, la pérdida de MAP para estas últimas —como se dijo previamente—

[22] Del 50 % al 31 % para T=50 % y del 94 % al 70 % para T=100 %.
[23] Ahora sólo 1 para T=50 % y 5 para T=100 %.

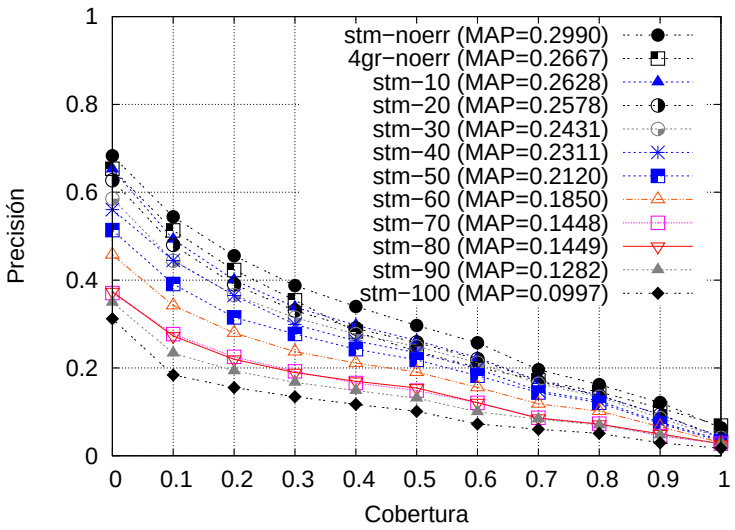

Figura 12.10: Precisión vs. cobertura para las consultas corregidas mediante el algoritmo de corrección contextual (empleando *extracción de raíces*).

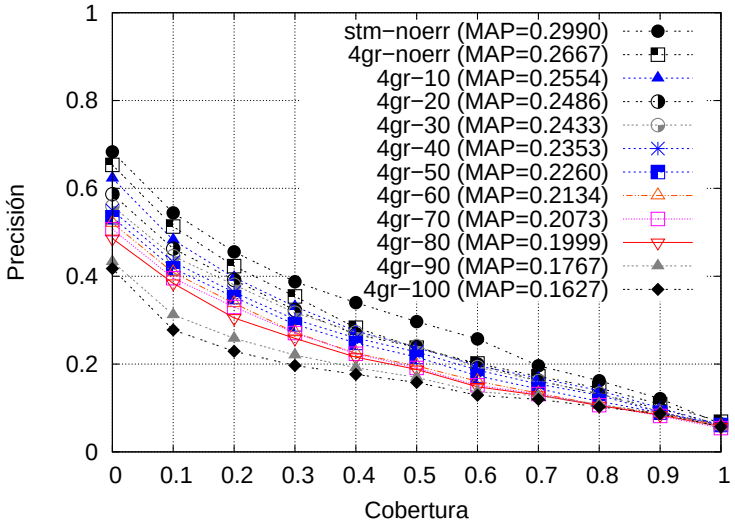

Figura 12.11: Precisión vs. cobertura para las consultas sin corregir (empleando *n-gramas*).

era significativa incluso para T=10 %, con una reducción del 18 % para T=10 %, 25 % para T=20 %, 50 % para T=50 % y 94 % para T=100 %. Para los mismos valores de T, la aplicación de nuestro corrector ortográfico contextual —ligeramente superior a la propuesta de Savary[140]— reducía dichas pérdidas a 12 %, 14 %, 29 % y 67 %, respectivamente, con lo que dichas caídas ya no eran significativas hasta T=20 %. Sin embargo, los n-gramas superan a ambos de forma clara, siendo la pérdida de MAP significativa sólo a partir de T=40 %, y casi reduciendo a la mitad la cuantía de dichas pérdidas: 4 %, 7 %, 15 % y 39 %, respectivamente. Además, ya no hay consultas que no devuelven documentos, ni siquiera para T=100 %.

Como conclusión podemos decir que el uso de técnicas de corrección ortográfica tiene un impacto sensiblemente positivo en el rendimiento de los sistemas de RI basados en extracción de raíces. Sin embargo, en los casos en los que no estén disponibles los recursos necesarios para aplicar estas técnicas, o resulte de interés un sistema independiente del idioma, el uso de n-gramas permite obtener también mejoras significativas. Un factor a tener muy presente es la longitud de las consultas y los documentos, así como el *ratio* de error presente en las consultas.

Capítulo 13

Conclusiones y trabajo futuro

Se recogen en este capítulo las conclusiones y aportaciones realizadas en este trabajo de tesis, así como las líneas de investigación a explorar en el futuro.

La cantidad de información relevante disponible a la hora de resolver cualquier problema condiciona irremediablemente la calidad de la solución obtenida, aunque eso no quiere decir que no se puedan obtener soluciones parciales o aceptables en el supuesto de desconocer o no tener la capacidad de manejar toda la información. El PLN no es una excepción, por lo que la escasez de recursos lingüísticos de calidad para el español libremente accesibles, en comparación con otras lenguas dominantes en la comunidad científica como el inglés, hace que adquieran especial relevancia los esfuerzos destinados al desarrollo de métodos que no requieran una excesiva cantidad de medios. Nuestra motivación es aún mayor por el hecho de pertenecer a una Comunidad Autónoma con lengua propia, como es el caso de Galicia, para la que la cantidad de recursos lingüísticos disponibles resulta muy inferior.

El objetivo principal de este proyecto de tesis ha sido el desarrollo y evaluación de la tecnología de base necesaria para el PLN, más concretamente en el ámbito del análisis léxico, la corrección ortográfica y la etiquetación morfosintáctica. En este sentido, nuestros mayores esfuerzos se han centrado en el desarrollo de una nueva técnica de corrección ortográfica sobre AFs con el objetivo de mejorar el rendimiento ofrecido por otras técnicas similares.

En este trabajo hemos descrito tres tipos de ambigüedad a nivel léxico proponiendo un método integral para su resolución, utilizando para ello únicamente la información léxica presente en el lexicón y la relativa al contexto morfosintáctico embebida en las probabilidades de un modelo estocástico basado en MMOs. Con el objetivo de probar la utilidad del método desarrollado, se han realizado experimentos exhaustivos en un entorno de RI. Para ello, se ha definido una metodología para la generación de consultas con distintos *ratios* de error utilizadas posteriormente para evaluar el impacto de la aplicación de nuestro método de corrección ortográfica contextual sobre los resultados obtenidos al interrogar un sistema de RI basado en extracción de raíces. Además, se han contrastado estos resultados con los arrojados por un sistema que utiliza n-gramas de caracteres superpuestos para la indexación y la recuperación evitando la aplicación de técnicas de corrección ortográfica.

13.1 Conclusiones

Durante el desarrollo de este trabajo hemos ido afrontando sucesivos retos con el propósito de refinar y ampliar las herramientas de análisis léxico y etiquetación morfosintáctica disponibles en el seno del grupo de investigación Compiladores y Lenguajes[1].

En primer lugar, hemos desarrollado un nuevo método de corrección ortográfica regional sobre AFs cuya característica diferencial radica en el concepto de región que nos permite delimitar el área de reparación de una palabra errónea, en contraposición con los métodos de corrección globales que aplican las operaciones básicas de reparación en todas las posiciones de la palabra sin tener en cuenta el punto en que el error es detectado. Adicionalmente, hemos implementado el método de corrección ortográfica propuesto por Savary [140] que nos ha servido como referencia a la hora de evaluar el rendimiento, la cobertura y la precisión de nuestra propuesta. Los resultados preliminares corroboraban no sólo que el rendimiento ofrecido por nuestra técnica de corrección regional era superior al que arrojaba el método de Savary [140], sino también que la diferencia entre ambos crecía con la localización del primer punto de error. Además, el método regional ofrecía un menor número de alternativas de corrección debido a que acotaba la zona del AF a explorar a la región en la que se detecta el error. Otro aspecto a tener en cuenta era el *ratio* de acierto del método. En el caso de nuestro método regional es del 77 % frente al 81 % del global, aunque cabía esperar que la integración de información lingüística, tanto desde el punto de vista semántico como sintáctico, debería reducir de forma significativa esta diferencia de precisión, que es menor que el 4 %, o podría incluso eliminarlo.

Una vez evaluado nuestro método regional de forma aislada, es decir, sin tener en cuenta su contexto morfosintáctico, realizamos su integración con el sistema de etiquetación MrTagoo [57, 58, 60], que además de resolver la ambigüedad asociada a aquellas unidades léxicas que presentan más de una etiqueta morfosintáctica posible, cuenta con una ampliación que le permite resolver las ambigüedades de segmentación [58], obteniendo así una herramienta de corrección ortográfica contextual. Esta integración ha sido posible al contar con la implementación de una versión del algoritmo de Viterbi sobre retículos [109], que resultan ser mucho más flexibles que los enrejados clásicos y que, de este modo, nos facilitaban la incorporación de las alternativas de corrección en la estructura. Así, la herramienta resultante, no sólo saca provecho de la probada eficiencia del algoritmo de Viterbi [167], sino también del preprocesador incluido en la herramienta original de etiquetación. Por tanto, hemos construido un entorno que nos permite manejar al mismo tiempo y sobre la misma estructura las ambigüedades morfosintácticas, las ambigüedades de segmentación y las ambigüedades léxicas introducidas por el corrector ortográfico cuando existe más de una alternativa de corrección de coste mínimo.

Al comparar el rendimiento de nuestro método de corrección regional con el propuesto por Savary [140] comprobamos que el algoritmo global ofrece un rendimiento ligeramente superior desde el punto de vista de la precisión, es decir, el algoritmo de corrección contextual acaba eligiendo la palabra correcta en un mayor número de ocasiones. Sin embargo, esta pequeña mejoría se obtiene a costa de un importante incremento del coste computacional del algoritmo.

[1] http://www.grupocole.org/

La utilidad práctica de los desarrollos llevados a cabo se refleja en la aplicación de éstos a la RI. En este campo, se ha desarrollado una metodología de pruebas con el fin de evaluar la incidencia de la densidad de errores sobre nuestro sistema de RI. Para ello, hemos realizado nuestros experimentos sobre consultas degradadas en español aplicando tres técnicas diferentes. La primera de ellas es una técnica basada en n-gramas y que no requiere la utilización de recursos lingüísticos extra. La segunda consiste en expandir las consultas con todas las alternativas de corrección para cada palabra errónea. Por último, hemos utilizado nuestro corrector ortográfico contextual para elegir la alternativa de corrección que mejor encajaba en el contexto de cada palabra errónea.

Los resultados de nuestros experimentos con consultas degradadas en RI ponen de manifiesto que el uso de técnicas de corrección ortográfica tiene un impacto sensiblemente positivo en el rendimiento de los sistemas de RI basados en extracción de raíces. Sin embargo, en los casos en los que no estén disponibles los recursos necesarios para aplicar estas técnicas, o resulte de interés un sistema independiente del idioma, el uso de n-gramas permite obtener también mejoras significativas. Un factor a tener muy presente es la longitud de las consultas y los documentos, así como el *ratio* de error presente en las consultas.

13.2 Trabajo futuro

Este trabajo constituye por una parte la continuación de una línea de investigación consolidada en el grupo de investigación Compiladores y Lenguajes[2] y por otra el inicio de futuros desarrollos y experimentos en el campo del análisis léxico y su aplicación a las fases posteriores del PLN, en particular a la RI.

En el campo de la corrección ortográfica, nos proponemos el estudio de estrategias y heurísticas que nos permitan determinar en qué situaciones puede resultar más provechosa la eficiencia de nuestro método de corrección regional y en cuáles es preferible optar por la precisión de un método global.

En lo que respecta a nuestra técnica de corrección contextual, debemos ampliar el tipo de errores manejados por el corrector. En la actualidad únicamente se tienen en cuenta los cuatro errores de edición descritos por Damerau [36]. Sin embargo, la flexibilidad del algoritmo de Viterbi sobre retículos hace posible el tratamiento de caminos de diversas longitudes mediante la aplicación de una técnica de normalización que pondera los valores obtenidos para las ecuaciones en cada camino según la longitud de éste [58, 109]. Esta característica resulta muy adecuada para el manejo de errores interpalabra, es decir, aquéllos que resultan de la unión de dos palabras debido a la omisión del espacio entre ellas o de la división de una palabra como consecuencia de la inserción indebida de un espacio.

Esta ampliación ha de ser integrada de forma cuidadosa en la herramienta actual con el fin de facilitar el tratamiento de situaciones en las que varios de los fenómenos tratados se solapen. Por ejemplo, en la frase "*Lo hizo apropósito*" en la que la palabra "*apropósito*" sería detectada por el corrector ortográfico como un error, ofreciendo entre otras posibles

[2] http://www.grupocole.org/

alternativas de corrección la de introducir un espacio después de la '*a*', en cuyo caso se introduciría una nueva ambigüedad segmental ya que "*a propósito*" podría ser una locución adverbial. Este tipo de situaciones resultan muy comunes en el caso del gallego por tratarse de una lengua extremadamente rica en el uso de contracciones y pronombres enclíticos que hacen que el análisis léxico resulte mucho más complejo.

Resulta también de especial interés continuar con nuestros experimentos con consultas degradadas en sistemas de RI. En este sentido, estamos interesados en repetir todos los experimentos realizados utilizando errores humanos en lugar de generarlos de un modo aleatorio. Para ello hemos solicitado a un grupo de voluntarios que hagan cinco copias de las consultas de prueba sin prestar atención a los errores ortográficos que puedan cometer, esto es, evitando autocorregirlos. Sin embargo, no es posible reproducir de un modo natural aquellas situaciones en que una persona comete un mayor número de errores por lo que, para obtener muestras con un elevado *ratio* de error, será necesario definir una metodología que nos permita combinar los errores cometidos por los diferentes autores.

Por otra parte, el hecho de que tanto la indexación y la recuperación basadas en n-gramas de caracteres como nuestra técnica de corrección ortográfica contextual sean independientes del idioma, justifica nuestro interés en la repetición de las pruebas realizadas para otros idiomas distintos del español. El *corpus* MULTEX-JOC [156] resulta muy adecuado para realizar este tipo de experimentos ya que contiene preguntas realizadas por escrito, en una amplia variedad de temas, por los miembros del Parlamento Europeo y las correspondientes respuestas de la Comisión Europea en nueve versiones paralelas. Además, contamos también con versiones paralelas de las consultas utilizadas en nuestros experimentos.

Chapter 14

Conclusions and future work

The conclusions and contributions contained in this PhD thesis are reflected in this chapter, as well as some research to be done in the near future.

The amount of relevant information available for resolving any problem inevitably affects the quality of the resulting solution, but this does not mean that partial or acceptable solutions are impossible to obtain in the case of ignoring or not having the ability to manage all the information. *Natural Language Processing* (NLP) is no exception, so the lack of freely available high quality linguistic resources for Spanish, if we compare it with other dominant languages in the scientific community such as English, makes all efforts to develop methods that do not require an excessive amount of resources acquire special significance. Our motivation is even greater due to the fact of belonging to an autonomous community with its own language, such as Galicia, where the amount of linguistic resources is much lower.

The main goal of this PhD thesis was the development and evaluation of the basic technology required for NLP, most notably in the field of lexical analysis, spelling correction and *Part-of-Speech* (PoS) tagging. In this regard, our major efforts have focused on developing a new spelling correction technique over *Finite Automata* (FAs) with the aim of improving on the performance offered by other similar techniques.

In this work, we have described three types of lexical ambiguity proposing a standard approach for their resolution using only the lexical information present in the lexicon and in the morphosyntactic context embedded in the probabilities of a stochastic model based on *Hidden Markov Models* (HMMs). With a view to testing the usefulness of the method developed, exhaustive tests in an *Information Retrieval* (IR) environment were performed. In order to do this, a methodology for query generation with several error ratios has been defined. These queries were later used to evaluate the impact of our contextual spelling correction method over the results obtained with a stemming-based IR system. Moreover, these results were contrasted with those obtained when using an overlapping character n-gram based system for indexing and retrieving, avoiding the application of spelling correction techniques.

14.1 Conclusions

During the development of this work, we have faced successive challenges to expand and refine the tools for lexical analysis and PoS tagging available within the Compilers and Languages research group[1]

First, we developed a new method for *regional spelling correction* over FAs whose main idea lies in the concept of *region*, which allows us to delimit the area of repair of a wrong word, as opposed to global correction methods, which apply the basic repair operations in all positions of the word, without taking into account the point where the error is detected. In addition, we have implemented the spelling correction method proposed by Savary [140] as a baseline for evaluating the performance, precision and recall of our proposal. Preliminary results not only corroborated that the performance offered by our regional correction technique was higher than the one shown by Savary's method, but also that the difference between them grew with the location of the first point of error. In addition, the regional approach offers fewer correction alternatives because the area of the FA to be explored is limited to the region where the error is detected. Another aspect to be considered was the ratio of correct choices of the method. In the case of our regional approach, this ratio is 77% versus 81% for the global one, although it was expected that the integration of linguistic information, from both semantic and syntactic levels, would significantly reduce this difference in accuracy, which is less than 4%, or could even eliminate it.

Having evaluated our regional method in isolation, i.e. without taking into account its PoS context, we integrated it with MrTagoo [57, 58, 60], a PoS tagger that in addition to resolving the ambiguity associated with those lexical units that have more than one possible PoS tag, has an extension which allows us to resolve segmentation ambiguities [58], obtaining as a result a contextual spelling correction tool. This integration has been possible due to the availability of an implementation of a version of the Viterbi algorithm over lattices, which are much more flexible than conventional trellises and, thus facilitate the incorporation of correction alternatives into the structure. Thus, the resulting tool not only takes advantage of the proven efficiency of the Viterbi algorithm [167], but also of the preprocessor included in the original PoS tagging tool. Therefore, we have built an environment that allows us to handle, at the same time and over the same structure, PoS tagging ambiguities, segmentation ambiguities, and lexical ambiguity introduced by the spell checker when there is more than one alternative for correction at minimum cost.

When comparing the performance of our regional correction method with the one proposed by Savary [140], we found that the global algorithm had a slightly higher performance from the standpoint of accuracy, i.e. the contextual correction algorithm chose the correct word on a greater number of occasions. However, this small improvement is obtained at the expense of an important increase in the computational cost of the algorithm.

The practical utility of our work is reflected in the application of it to IR. In this field, we have developed a methodology to perform tests and evaluate the impact of error

[1] http://www.grupocole.org/

density on our IR system. To do this, we have conducted our experiments on corrupted queries in Spanish by applying three different techniques. The first is based on n-grams and requires no extra linguistic resources. The second consists of expanding the queries with all the correction alternatives for each word. Finally, we have used our contextual spelling corrector to choose the correction alternative which fits best in the context of each misspelled word.

The results obtained from our experiments with corrupted queries in IR show that the use of spelling correction techniques has a significantly positive impact on the performance of stemming-based IR systems. However, when the resources needed to implement these techniques are not available, or a system with no dependence on the language is desired, the use of n-grams can also obtain significant improvements. One factor to bear in mind is the length of queries and documents as well as the error ratio present in queries.

14.2 Future work

This work was conceived, on one hand, as a continuation of a consolidated line of research in the Compilers and Languages research group[2] and the other hand, the beginning of further experiments and developments in the field of lexical analysis and their application to the subsequent phases of NLP, and particularly, IR.

In the spelling correction field, we propose the study of strategies and heuristics that allow us to identify situations in which the efficiency of our regional method may prove more fruitful and those others where the precision of a global approach would be preferable.

With regards to our contextual correction technique, we need to expand the types of error handled by the corrector. Nowadays, we only take into account the four edit errors described by Damerau [36]. However, the flexibility of the Viterbi algorithm over lattices makes it possible to deal with paths of different lengths, by means of the application of a normalization technique that weights the values obtained for the equations in each path depending on its length [58, 109]. This feature is very suitable for handling inter-word errors, those resulting from the joining of two words because of the omission of the space between them, or from the division of a word as a result of the improper insertion of a space.

This extension should be carefully integrated into the current tool in order to facilitate the handling of situations in which several of the phenomena being considered overlap. For example, in the sentence "*Lo hizo apropósito*" ("he did it onpurpose") where the word "apropósito" ("*onpurpose*") would be detected by the spell checker as an error, offering among other possible alternatives for correction the one which consists of typing a space after the '*a*', in which case a new segmental ambiguity would be introduced since "*a propósito*" ("*on purpose*") could be an adverbial phrase.

Such situations are very common in the case of Galician as it is an extremely rich language in the use of enclitic pronouns and contractions, which makes lexical analysis much more complex.

It is also of special interest to continue our experiments with degraded queries on

[2] http://www.grupocole.org/

IR systems. In this sense, we are interested in repeating all the experiments performed using human errors, instead of randomly generated ones. To do this, we have asked a group of volunteers to write five copies of all the test queries without paying attention to the spelling errors they may commit, that is, avoiding self-correction. However, it is not possible to reproduce in a natural way those situations in which a person makes more errors and therefore to obtain examples with a high *error ratio* it will be necessary to define a methodology that allows us to combine mistakes made by different authors.

Moreover, the fact that both n-gram based indexation and retrieval techniques and our contextual spelling correction technique have no dependence on a concrete language justifies our interest in the repetition of tests for languages other than Spanish. The MULTEX-JOC *corpus* is very suitable for such experiments because it contains written questions on a wide variety of topics by members of the European Parliament and the responses of the European Commission in nine parallel versions. In addition, we also have parallel versions of the queries used in our experiments.

Bibliografía

[1] Steven Abney. Part-of-speech tagging and partial parsing. In S. Young and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*, pages 118–136. Dordretcht: Kluwer Academic, 1996.

[2] E. Agirre, X. Arregi, X. Artola, A. Díaz De Ilarraza, and K. Sarasola. Conceptual distance and automatic spelling correction. In *Proceedings of the workshop on Computational Linguistics for Speech and Handwriting Recognition*, Leeds, United Kingdom, 1994.

[3] Eneko Agirre and German Rigau. Word sense disambiguation using conceptual density. In *Proceedings of the 16th Conference on Computational Linguistics (COLING '96), August 5-9, 1996, Copenhagen, Denmark*, pages 16–22. Association for Computational Linguistics, Morristown, NJ, USA, 1996.

[4] Eneko Agirre, Euskal Herriko Unibertsitatea, and German Rigau. A proposal for word sense disambiguation using conceptual distance. 1995.

[5] Alfred V. Aho and M. J. Corasick. Fast pattern matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June 1975.

[6] C. Álvarez Lebredo, P. Alvariño Alvariño, A. Gil Martínez, T. Romero Quintáns, M.P. Santalla del Río, and S. Sotelo Docío. AVALON: Una Gramática Formal basada en Corpus. *Procesamiento del Lenguaje Natural*, (23):132–139, 1998.

[7] Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, 2002.

[8] R. C. Angell, G. E. Freund, and P. Willet. Automatic spelling correction using a trigram measure. *Inf. Process. Manage.*, 19:255–261, 1993.

[9] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley and ACM Press, Harlow, England, 1999.

[10] L.R. Bahl and R.L. Mercer. Part-of-speech assignementby a statistical decision algorithm. *International Symposium on Information Theory*, 1976.

[11] J.K. Baker. *Stochastic modeling for automatic speech understanding*, pages 297–307. Academic Press, New York, 1975.

[12] Fco. Mario Barcala Rodríguez, Eva M. Domínguez, Miguel A. Alonso Pardo, David Cabrero Souto, Jorge Graña Gil, Jesús Vilares Ferro, Manuel Vilares Ferro, Guillermo Rojo Sánchez, M. Paula Santalla, and Susana Sotelo. Una aplicación de RI basada en PLN: el proyecto ERIAL. In *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI)*, pages 165–172, UPV, Valencia, Spain, 2002. Editorial UPV.

[13] L.E. Baum, L Petrie, G Soules, and N Weiss. A maximization technique ocurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

[14] R. Beckwith, G.A. Miller, and R. Tengi. Design and implementation of the WordNet lexical database and searching software, 1993. Revised version of "Implementing a Lexical Network"in CSL Report 43, prepared by R. Tengi, August 1993.

[15] J. Bigert, L. Ericson, and A. Solis. Missplel and Autoeval: Two generic tools for automatic evaluation. In *Proceedings of Nodalida*, Reykjavic, 2003.

[16] Bodo Billerbeck and Justin Zobel. Techniques for efficient query expansion. In Alberto Apostolico and Massimo Melucci, editors, *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004, Padova, Italy, October 5-8, 2004, Proceedings*, volume 3246 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2004.

[17] Charles R. Blair. A program for correcting spelling errors. *Information and Control*, 3(1):60–67, March 1960.

[18] W.W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the Eastern Joint Computer Conference*, volume 16, pages 225–232, 1959.

[19] R. H. Boivie. Directory assistance revisited. Technical Report 12, AT & T Bell Labs, Jun 1981.

[20] Thorsten Brants. Cascaded markov models. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 118–125, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

[21] Thorsten Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[22] Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction. In *In Proc. of the 38th Annual Meeting of the ACL*, pages 286–293. ACL, 2000.

[23] Alexander Budanitsky and Graeme Hirst. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources at the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-01), June 2001, Pittsburgh, PA, USA*, pages 29–34. Carnegie Mellon University, Pittsburgh, PA, USA, June 2001.

[24] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[25] Robin D. Burke, Kristian J. Hammond, Vladimir A. Kulyukin, Steven L. Lytinen, Noriko Tomuro, and Scott Schoenberg. Question answering from frequently-asked question files: Experiences with the faq finder system. Technical Report 18, AI Magazine, Chicago, IL, USA, 1997.

[26] Claire Cardie. Empirical methods in information extraction. *AI magazine*, 18:65–79, 1997.

[27] Claudio Carpineto, de Mori, Renato, Giovanni Romano, and Brigitte Bigi. An information-theoretic approach to automatic query expansion. *ACM Transactions on Information Systems*, 19(1):1–27, 2001.

[28] Centro Ramón Piñeiro para a Investigación en Humanidades. Etiquetador/Lematizador do Galego Actual (XIADA), versión 2.1. http://corpus.cirp.es/xiada, (consultado el 22 de enero de 2009), 2009.

[29] J.P. Chanod and P. Tapanainen. Creating a tagset, lexicon and guesser for a French tagger. In *ACL SIGDAT Workshop on From Texts to Tags: Issues in Multilingual Language Analysis*, pages 58–64, University College, Dublin, Ireland, 1995.

[30] E. Charniak, C. Hendrickson, and Jacobson. Equations for part-of-speech tagging. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789, 1993.

[31] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Menlo Park, CA, 1997. AAAI Press/MIT Press.

[32] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959. Also in 'Readings in mathematical psychology', R.D. Luce, R. Bush, and E. Galanter (eds.), New York: Wiley, pp. 125-155, 1965.

[33] K.W. Church. A stochastic parts program and noun phrase parser for unrestricted text. *Procedings of the second conference on Applied Natural Language Processing*, pages 136–143, 1988.

[34] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140, Trento (Italy), 1992.

[35] J. Daciuk, S. Mihov, B.W. Watson, and R.E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational Linguistics*, 26(1):3–16, 2000.

[36] F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), March 1964.

[37] Leon Davidson. Retrieval of mis-spelled names in an airline passenger record system. *Communications of the Association for Computing Machinery*, 5(3):169–171, March 1962.

[38] Richard I. A. Davis and Brian C. Lovell. Comparing and Evaluating HMM Ensemble Training Algorithms Using Train and Test and Condition Number Criteria. *Pattern Analysis and Applications*, 6(4):327–335, 2004. Originally published as Davis, Richard I. A. and Lovell, Brian C. (2004) Comparing and Evaluating HMM Ensemble Training Algorithms Using Train and Test and Condition Number Criteria. Pattern Analysis and Applications 6(4):327-336. doi: 10.1007/s10044-003-0198-6 Copyright 2004 Springer-Verlag. All rights reserved. The original article is available from www.springerlink.com.

[39] Owen de Kretser and Alistair Moffat. SEFT: A search engine for text. *Software-Practice & Experience*, 34(10):1011–1023, August 2004. Source code available in: `http://www.cs.mu.oz.au/~oldk/seft/` (visitada en febrero 2005).

[40] R.J. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31–39, 1988.

[41] A.M. Derouault and B. Merialdo. Natural language modeling for phoneme-to text transcription. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:642–649, 1986.

[42] Michael R. Dunlavey. Letter to the editor: On spelling correction and beyond. *Communications of the ACM*, 24(9):608–608, September 1981.

[43] R.J. Elliott. Annotating spelling list words with affixation classes. Technical report, AT & T Bell Labs, 1988.

[44] J.E. Ellman, I. Klincke, and J.I. Tait. Word sense disambiguation by information filtering and extraction. *Computers and the Humanities*, 34:127–134, 2000.

[45] J. Fagan. Automatic phrase indexing for document retrieval. In *SIGIR '87: Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–101, New York, NY, USA, 1987. ACM.

[46] Christiane Fellbaum. English verbs as a semantic net. *International Journal of Lexicography*, 3(4):278–301, 1990.

[47] G.F. Foster. Statistical lexical disambiguation. Master's thesis, School of Computer Science, McGill University, 1991.

[48] Edward A. Fox. Characterization of two new experimental collections in computer and information science containing textual and bibliographical concepts. Technical Report 83-561, 1983.

[49] E. Fredkin. Trie memory. *CACM*, 3:490–499, 1969.

[50] W. A. Gale, K. W. Church, and D. Yarowsky. A method for disambiguating word senses in a large corpus. In *computers and the humanitzes*, volume 26, pages 415–439, 1993.

[51] R.G. Garside, G.N. Leech, and Sampson. *The computational analysis of English: a corpus-based approach*. Longman, London, 1987.

[52] Eric Goedelbecker. Using grep. *Linux Journal*, 18, October 1995.

[53] A. R. Golding and D. Roth. A winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999. Special Issue on Machine Learning and Natural Language.

[54] Andrew R. Golding. A Bayesian hybrid method for context-sensitive spelling correction. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 39–53, Somerset, New Jersey, 1995. Association for Computational Linguistics, Association for Computational Linguistics.

[55] Andrew R. Golding and Yves Schabes. Combining trigram-based and feature-based methods for context-sensitive spelling correction. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 71–78, Santa Cruz, CA, February 13 1996. ACL.

[56] R. E. Gorin. *SPELL: A spelling checking and correction program. Online documentation*, 1971.

[57] Jorge Graña Gil. *Técnicas de Análisis Sintáctico Robusto para la Etiquetación del Lenguaje Natural*. PhD thesis, Universidad de A Coruña, September 2000.

[58] Jorge Graña Gil, Miguel Alonso Pardo, and Manuel Vilares Ferro. A Common Solution for Tokenization and Part-of-Speech Tagging: One-Pass Viterbi Algorithm vs. Iterative Approaches. *Lecture Notes in Artificial Intelligence*, 2448:3–10, 2002.

[59] Jorge Graña Gil, Francisco Mario Barcala Rodríguez, and Miguel Ángel Alonso Pardo. Compilation methods of minimal acyclic automata for large dictionaries. *Lecture Notes in Computer Science*, 2494:135–148, 2002.

[60] Jorge Graña Gil, Francisco Mario Barcala Rodríguez, and Jesús Vilares Ferro. Formal Methods of Tokenization for Part-of-Speech Tagging. *Lecture Notes in Computer Science*, 2276:240–249, 2002.

[61] Jorge Graña Gil, J.C. Chappelier, and Manuel Vilares Ferro. Integrating external dictionaries into part-of-speech taggers. In *Proceedings of the Euroconference on Recent Advances in Natural Language Processing (RANLP-2001)*, pages 122–128, Tzigov Chark, Bulgaria, 2001.

[62] B.B. Greene and G.M. Rubin. Automatic grammatical tagging of English. Ma, Brown University, Providence, 1971.

[63] Derek Gross and Katherine J. Miller. Adjectives in WordNet. *International Journal of Lexicography*, 3(4):265–277, 1990.

[64] Allen R. Hanson, Edward M. Riseman, and E. Fisher. Context in word recognition. *Pattern Recognition*, 8(1):35–45, 1976.

[65] Donna Harman. Relevance feedback revisited. In *Proceedings of the 15th annual international ACM SIGIR conference on research and development in Information Retrieval (SIGIR'92), June 21-24, Copenhagen, Denmark*, pages 1–10. ACM Press, 1992.

[66] L.D. Harmon. Automatic recognition of print and script. pages 1165–1176. IEEE, 1972.

[67] P. E. Hart, N. J. Nilsson, and Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 2:100–107, 1968.

[68] Marti A. Hearst. TextTiling: segmentating text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[69] G. Hirst and D. St-Onge. Lexical chains as representation of context for the detection and correction malapropisms, 1997.

[70] Graeme Hirst and Alexander Budanitsky. Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(1):87–111, 2005.

[71] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, MA, 1979.

[72] David A. Hull. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84, 1996.

[73] J. J. Hull and S. N Srihari. Experiments in text recognition with binary n-gram and Viterbi algorithms. *IEEE Trans. Patt. Anal. Machine Intell.*, PAMI-4(5):520–530, 1982.

[74] Armin Hust. Query expansion methods for collaborative information retrieval. *Inform, Forsch. Entwickl*, 19(4):224–238, 2005.

[75] Peter Jackson and Isabelle Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization*, volume 5 of *Natural Language Processing*. John Benjamins Publishing Company, Amsterdam/Philadelphia, 2002.

[76] Christian Jacquemin. Syntagmatic and paradigmatic representations of term variation. In *37th Annual Meeting of the Association for Computational Linguistics (ACL'99), Proceedings*, pages 341–348, Maryland, 1999.

[77] F. Jelinek. *Markov source modeling of text generation*, pages 569–598. E91 of NATO ASI series. Dordrecht: M. Nijhoff, 1985.

[78] Karen Sparck Jones. Retrieval system tests 1958 — 1978. In Karen Sparck Jones, editor, *Information Retrieval Experiment*, pages 213–255. Butterworths, London, 1981.

[79] Daniel Jurafsky and James H. Martin. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Upper Saddle River, New Jersey, 2000.

[80] Mark D. Kernighan, Kenneth W. Church, and William A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics*, pages 205–210, Morristown, NJ, USA, 1990. Association for Computational Linguistics.

[81] S. Klein and R.F. Simmons. A computational approach to grammatical coding of English words. *Journal of the Association for Computing Machinery*, 10:334–347, 1963.

[82] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Publishing Company, Reading, 1973.

[83] Gerald Kowalski. *Information Retrieval Systems: Theory and Implementation*. The Kluwer international series on Information Retrieval. Kluwer Academic Publishers, Boston-Dordrecht-London, 1997.

[84] H. Kucera and W.N. Francis. *Computational analysis of present-day American English*. Providence, R.I., 1967.

[85] K. Kukich. Variations on a back-propagation name recognition net. In *Advanced Technology Conference*, volume 2, pages 722–735, Washington D.C., May 1988. U.S. Postal Service.

[86] K. Kukich. Spelling correction for the telecommunications network for the deaf. *Communications of the ACM*, 35(5):80–90, May 1992.

[87] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[88] Julian Kupiec. Murax: a robust linguistic approach for question answering using an on-line encyclopedia. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 181–190, New York, NY, USA, 1993. ACM.

[89] F.W. Lancaster. *Information Retrieval Systems: Characteristics, Testing and Evaluation*. John Wiley & Sons, New York, 1968.

[90] Lancaster University (UK), Computers Communications and Visions (France), and Universidad Autónoma de Madrid (Spain). CRATER Project. Corpus Resources And Terminology ExtRaction (MLAP93/20), creation of a set of tools an resources for multilingual corpus linguistic work., 1993.

[91] C. Leacock and M. Chodorow. *Combining local context and WordNet similarity for word sense identification*, pages 305–332. In C. Fellbaum (Ed.), MIT Press, 1998.

[92] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[93] Julie B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31, 1968.

[94] C.L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software-Practice and Experience*, 23(1):15–30, January 1993.

[95] H.P. Luhn. The automatic creation of literature abstracts. *IBM journal of research and development*, 2(2):159–165, April 1958.

[96] Lidia Mangu and Eric Brill. Automatic rule acquisition for spelling correction. In *Proceedings of the 14th International Conference on Machine Learning (ICML-97)*, pages 187–194, Nashville, TN, 1997. Morgan Kaufmann.

[97] Mano, Hiroko and Ogawa, Yasushi. Selecting expansion terms in automatic query expansion. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 390–391, 2001.

[98] A. A. Markov. An example of statistical investigation in text of Eugene Onyegin illustrating coupling of tests in chains. In *Proceedings of the Academy of Sciences*, volume 7, pages 153–162, St. Petersburg, 1913.

[99] I. Marshall. *Tag selection using probabilistic methods*, pages 42–65. 1987.

[100] Yuji Matsumoto and Takehito Utsuro. *Lexical Knowledge Acquisition*, chapter 24, pages 563–610. 2000.

[101] Eric Mays, Fred J. Damerau, and Robert L. Mercer. Context based spelling correction. *Information Processing and Management*, 27(5):517–522, 1991.

[102] Paul Mcnamee and James Mayfield. Character n-gram tokenization for European language text retrieval. *Information Retrieval*, 6:73 – 97, 2004.

[103] Paul McNamee, James Mayfield, and Christine D. Piatko. A language-independent approach to european text retrieval. In *CLEF '00: Revised Papers from the Workshop of Cross-Language Evaluation Forum on Cross-Language Information Retrieval and Evaluation*, pages 129–139, London, UK, 2001. Springer-Verlag.

[104] Rada Mihalcea. *Encyclopedia of Machine Learning*, chapter Word Sense Disambiguation. Springer-Verlag New York Inc, November 2007.

[105] George A. Miller. Nouns in Wordnet: A lexical inheritance system. *International Journal of Lexicography*, 3(4):245–264, 1990.

[106] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[107] Markus Mittendorfer and Werner Winiwarter. A simple way of improving traditional IR methods by structuring queries. In *Proc. of the 2001 IEEE International Workshop on Natural Language Processing and Knowledge Engineering (NLPKE 2001) , October 7-10, Tucson, Arizona, USA*, 2001.

[108] Markus Mittendorfer and Werner Winiwarter. Exploiting syntactic analysis of queries for information retrieval. *Data & Knowledge Engineering*, 42(3):315–325, 2002.

[109] Miguel A. Molinero Álvarez, Fco. Mario Barcala Rodríguez, Juan Otero Pombo, and Jorge Graña Gil. Practical application of one-pass Viterbi algorithm in tokenization and part-of-speech tagging. In Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nicolov, and Nikolai Nikolov, editors, *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 35–40, Borovets, Bulgaria, 2007.

[110] J. L. A. Moreno, A. Álvarez Lugrís, and X. G. Guinovart. Aplicacións do etiquetario morfosintáctico do sli ó corpus de traduccións tectra. *Viceversa: Revista Galega de Traducción*, (7/8):189–212, 2003.

[111] R. Morris and L. L. Cherry. Computer detection of typographical errors. *IEEE Trans. on Professional Communication*, PC-18:54–56, March 1975. 18.

[112] A. Nardi, C. Peters, and J.L. Vicedo. Working Notes of the CLEF 2006 Workshop. `http://www.clef-campaign.org` (visitada en octubre 2008), 2006.

[113] S.J. Nelson, M. Shopen, J. Shulman, and N. Arluk. An interlingual database of MeSH translations. In *Proceedings of 8th International Conference on Medical Librarianship*, London, 2000.

[114] M. K. Odell and R. C. Russell. Patentes nº, 1, 261, 167 (1918) y 1, 435, 663 (1922) de la u.s. pattent office, washington, d.c., 1918.

[115] Kemal Oflazer. Spelling correction in agglutinative languages. October 07 1994.

[116] Kemal Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. July 21 1995.

[117] Juan Otero Pombo, Jorge Graña Gil, and Manuel Vilares Ferro. Contextual Spelling Correction. *Lecture Notes in Computer Science*, 4739:290–296, 2007.

[118] Juan Otero Pombo, Jesus Vilares Ferro, and Manuel Vilares Ferro. Corrupted Queries in Spanish Text Retrieval: Error Correction vs. N-Grams. 2008.

[119] Juan Otero Pombo, Jesus Vilares Ferro, and Manuel Vilares Ferro. Text retrieval through corrupted queries. *Lecture Notes in Artificial Intelligence*, 5290:362–371, 2008.

[120] I. Ounis, G. Amati, v. Plachouras, B. He, C. Macdonald, and C. Lioma. TERRIER: A high performance and scalable Information Retrieval platform. In *Proceedings of the ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, pages 18–25, 2006. Herramienta disponible en `http://ir.dcs.gla.ac.uk/terrier/` (visitada en octubre 2008).

[121] J. L. Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980.

[122] Jean Eric Pin. *Formal properties of finite automata applications*. Springer-Verlag, 1988.

[123] J. J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, April 1984.

[124] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[125] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. 19:17–30, 1989.

[126] X. Ren and F. Perrault. The typology of unknown words: an experimental study of two corpora. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pages 408–414, Nantes, 1992. International Committee on Computational Linguistics.

[127] Edward M. Riseman and Allen R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Transactions on Computers*, 23(5):480–493, May 1974.

[128] S.E. Robertson. The probability ranking principle in IR. *Journal of Documentation*, (33), 1977.

[129] S.E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Sciences*, (27):129–146, May–June 1976.

[130] S.E. Robertson, M.E. Maron, and W.S. Cooper. Probability of relevance: A unification of two competing models for document retrieval. *Information Technology: Research and Development*, (1):1–21, 1982.

[131] J.J. Rocchio. *The Smart Retrieval System: Experiments in Automatic Document Processing*, chapter Relevance feedback in information retrieval, pages 313–323. 1971.

[132] D. E. Rumelhart and J. L. McClelland. Learning internal representations by error propogation. In *Explorations in the Micro-Structure of Cognition Vol. 1 : Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[133] Russel S. and Norving P. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[134] G. Salton and C. Buckley. Term-weighting approaches in automatic retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[135] G. Salton and M.E. Lesk. Computer evaluation of indexing and text processing. *Journal of the ACM*, 15(1):8–36, January 1968.

[136] Gerard Salton. The SMART document retrieval project. In *SIGIR '91: Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 356–358. ACM Press, 1991.

[137] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[138] F. Sánchez León and A.F. Nieto Serrano. Desarrollo de un etiquetador morfosintáctico para el español. *Revista de la Sociedad Española para el Procesamiento del Lenguaje Natural*, 17:14–28, 1995.

[139] F. Sánchez León and A.F. Nieto Serrano. Development of a Spanish version of the XEROX tagger. Technical report, Facultad de Filosofía y Letras, Universidad Autónoma de Madrid, 1995.

[140] A. Savary. Typographical nearest-neighbor search in a finite-state lexicon and its application to spelling correction. In *CIAA: International Conference on Implementation and Application of Automata, LNCS*, 2001.

[141] N. Sebastián, M.A. Martí, M.F. Carreiras, and F. Cuetos. *LEXESP: léxico informatizado del español*. Edicions de la Universitat de Barcelona, 2000.

[142] B. A. Sheil. Median split trees: a fast lookup technique for frequently occuring keys. *Communications of the ACM*, 21(11):947–958, 1978.

[143] K. Sikkel. *Parsing Schemata*. PhD thesis, University of Twente, The Netherlands, 1993.

[144] E.J. Sitar. Machine recognition of cursive script: the use of context for error detection and correction. Technical report, Bell Labs, 1961.

[145] Alan F. Smeaton. Progress in the application of natural language processing to information retrieval tasks. *The Computer Journal*, 35(3):268–278, 1992.

[146] A. Solak and K. Oflazer. Design and implementation of a spelling checker for Turkish. *Literary and Linguistic Computing*, 8(3), 1993.

[147] Richard Sproat. *Morphology and Computation*. The MIT Press, Cambridge, Massachusetts, 1992.

[148] Mark Stevenson. *Word Sense Disambiguation: The Case for Combinations of Knowledge Sources*. Studies in Computational Linguistics. CSLI, Stanford, 2003.

[149] W.S. Stolz, P.H. Tannenbaum, and Carstensen. A stochastic approach to the grammatical coding of English. *Communications of the ACM*, 8(6):399–405, 1965.

[150] Tomek Strzalkowski. Natural language information retrieval. *Information Processing Management*, 31(3):397–417, 1995.

[151] Tomek Strzalkowski and Jose Perez-Carballo. Recent developments in natural language text retrieval. In D. K. Harman, editor, *NIST Special Publication 500-215: The Second Text REtrieval Conference (TREC-2)*, pages 123–136, Gaithersburg, MD, USA, 1994. Department of Commerce, National Institute of Standards and Technology.

[152] M. Sussna. Word sense disambiguation for free-text indexing using a massive semantic network. In *Proc. of 2nd International Conference on Information and Knowledge Management*, Arlington, Virginia, 1993.

[153] Kristina Toutanova. Pronunciation modeling for improved spelling correction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 144–151, 2002.

[154] Thomas N. Turba. Checking for spelling and typographical errors in computer-based text. *SIGPLAN Notices*, 16(6):51–60, 1981.

[155] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2^{nd} edition, 1979.

[156] J. Véronis. Multext-Corpora. An annotated corpus for five European languages. CD-ROM, Paris distribuído por ELRA/ELDA, 1999.

[157] Jean Veronis. Correction of phonographic errors in natural language interfaces. In *Proceedings of the Eleventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Natural Language Processing (3), pages 101–115, 1988.

[158] Jesús Vilares, Fco. Mario Barcala, Santiago Fernández, and Juan Otero. Manejando la variación morfológica y léxica en la recuperación de información textual. *Procesamiento del Lenguaje Natural*, 30:99–106, March 2003.

[159] Jesús Vilares Ferro. *Aplicaciones del procesamiento del lenguaje natural en la recuperación de información en Español*. PhD thesis, Universidade Da Coruña, May 2005.

[160] Manuel Vilares Ferro, Juan Otero Pombo, Fco. Mario Barcala Rodríguez, and Eva Domínguez. Automatic spelling correction in galician. *Lecture Notes in Artificial Intelligence*, 3230:45–57, 2004.

[161] Manuel Vilares Ferro, Juan Otero Pombo, and Víctor Manuel Darriba Bilbao. Regional vs. Global Robust Spelling Correction. *Lecture Notes in Computer Science (LNCS)*, 3878:575–586, 2006.

[162] Manuel Vilares Ferro, Juan Otero Pombo, and Jorge Graña Gil. Regional finite-state error repair. *Lecture Notes in Computer Science (LNCS)*, 3317:269–280, 2004.

[163] Manuel Vilares Ferro, Juan Otero Pombo, and Jorge Graña Gil. Regional vs. Global Finite-State Error Repair. *Lecture Notes in Computer Science (LNCS)*, 3406:120–131, 2005.

[164] Manuel Vilares Ferro, Juan Otero Pombo, and Jorge Graña Gil. Spelling Correction on Technical Documents. *Lecture Notes in Computer Science (LNCS)*, 3643:131–139, 2005.

[165] Manuel Vilares Ferro, Juan Otero Pombo, and Jesús Vilares Ferro. Robust Spelling Correction. *Lecture Notes in Computer Science (LNCS)*, 3845:319–328, 2006.

[166] Manuel Vilares Ferro, Alberto Valderruten Vidal, Jorge Graña Gil, and Miguel Alonso Pardo. Une approche formelle pour la génération dánalyseurs de langages naturels. In P. Blache, editor, *Actes de la Seconde Conférence Annuelle sur le Traitement Automatique du Langage*, Marseille, France, June 1995.

[167] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, April 1967.

[168] E. M. Voorhees and D. K. Harman. Overview of the 6th Text REtrieval Conference (TREC-6). In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-240: The Sixth Text REtrieval Conference (TREC-6)*, pages 1–24, Gaithersburg, MD, USA, 1997. Department of Commerce, National Institute of Standards and Technology.

[169] Ellen M. Voorhees. Query expansion using lexical-semantic relations. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pages 61–69, Dublin, Ireland, July 1994. ACM.

[170] Piek Vossen. *EuroWordNet. A Multilingual Database with Lexical Semantic Networks.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998. Reprinted from Computers and the Humanities, Volume 32, Nos. 2–3, 1998.

[171] Robert A. Wagner. Order-*n* correction for regular languages. *Communications of the ACM*, 17(5):265–268, May 1974.

[172] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[173] Robert A. Wagner and Roy Lowrance. An extension of the string-to-string correction problem. *J. ACM*, 22(2):177–183, 1975.

[174] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes.* Morgan Kaufmann, 1999.

[175] J. Xu and W.B. Croft. Query expansion using local and global analysi. In *Proc. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '96). Zurich, Switzerland*, volume 32, pages 4–11, 1996.

[176] E. J. Yannakoudakis and D. Fawthrop. An intelligent spelling error corrector. *Information Processing and Management*, 19(2):101–108, 1983.

[177] E. J. Yannakoudakis and D. Fawthrop. The rules of spelling errors. *Information Processing and Management*, 19(2):87–99 (or 87–100??), 1983.

[178] David Yarowsky. Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *ACL-94*, pages 88–95, Las Cruces, NM, 1994. ACL.

[179] E. M. Zamora, J. J. Pollock, and Antonio Zamora. The use of trigram analysis for spelling error detection. *Information Processing & Management*, 17(6):305–316, 1981.

Índice alfabético