Lematizador y Flexionador con Estimador Idiomático, usando algoritmos eficientes para idiomas muy flexivos como el español

Andrés Tomás Hohendahl [A]
andres.hohendahl@fi.uba.ar

José Francisco Zelasco [A,B]
jfzelasco@fi.uba.ar

[A] Laboratorio de Estereología y Mecánica Inteligente.
Dto. de Ing. Mecánica Facultad de Ingeniería,
Universidad de Buenos Aires, Paseo Colón 850,
(1065), Buenos Aires, Argentina.
[B] INTIA, Facultad de Ciencias Exactas, UNCPBA,
Universidad del Centro de la Provincia de Buenos Aires,
Campus Universitario, Paraje Arroyo Seco
(B7000) Tandil, Prov. de Buenos Aires, Argentina.

Abstract

We present a library for multilingual word recognition and flexion capable of language identification, linguistic word-tagging with semantic extraction, morphologic and sound-like *(phonetic)* estimation with automatic spelling error recognition and correction. We achieve low footprint and good efficiency by using special data structures combined with morphological rules. For Spanish 50k lemmas packed with 4.5k rules, *(equivalent to 1.2M words and >300M guessable)* fits into 200kb. Capable of morphological and sound-like *(phonetic)* inference, in a similar way as a human interpreter would perform. As flexion generator, it has semantic expression capability. It's targeted towards human-machine dialog in natural language; specially intended for handling error prone text inputs like automatic writing and speech recognition, dialog systems, among others.

Keywords: Natural Language Processing, NLP, Lemmatize, Flexion, Morphologic, Semantics, Language Recognition, Spanish Phonetic Similarity, Dialog Systems.

Resumen

Se presenta una librería para reconocimiento y flexión de palabras multilinguales capaz de identificación de lenguaje, etiquetado lingüístico con extracción semántica, estimación morfológica y fonética con reconocimiento y corrección automática de errores de ortografía. Obtenemos buena eficiencia con un bajo uso de recursos combinando estructuras de datos especiales con reglas morfológicas. Para el español, 50k lemas junto a 4.5k reglas, *(equivalentes a 1.2M palabras y >300M inferibles)* caben en 200kb. Capaz de realizar inferencia morfológica y suena-como *(fonética)* en forma similar a la de un intérprete humano. Como flexionador, posee capacidad de expresión semántica. Está orientada a diálogo hombre-máquina en lenguaje natural y fue diseñada especialmente para manejar textos con errores provenientes de sistemas de reconocimiento automático de texto escrito y hablado, sistemas de diálogo, entre otros.

Palabras claves: Procesamiento de Lenguaje Natural, PNL, Lematizador, Flexionador, Morfología, Semántica, Reconocimiento de Idioma, Similaridad Fonética Española, Sistemas de Diálogo.

1. Introducción

La primera etapa del procesamiento de texto natural *(NLP)* es sin duda, el etiquetado de palabras partiendo de un texto plano. Para esto se utilizan numerosas técnicas de búsqueda en diccionario, muchas de las cuales solamente funcionan con palabras exactas. Cuando el texto provee de fuentes con errores, los métodos tradicionales de búsquedas en diccionarios fallan o son muy poco eficientes. Este es el caso de textos provenientes del reconocimiento automático de voz y texto escrito. También hay otras fuentes de texto con errores involuntarios, sistemáticos o atribuibles al método, como los de mensajería *(SMS, e-mail y chat)*, en donde suele no usarse ortografía estricta, por ser mensajes breves, inmersos en contexto y provenientes de dispositivos con limitaciones prácticas *(celulares y PDA's sin tildes o con métodos engorrosos)*.

En momentos en donde la informática está tan difundida y hay sobrada capacidad de cálculo, aún quedan pendientes cosas que hace décadas se suponía que estarían resueltas en estos días. Una de las más sorprendentes es la comunicación hombre-máquina en lenguaje natural. Esta asignatura pendiente aún no ha salido airosa de los laboratorios académicos y una de las razones de esto es la enorme complejidad que conlleva. La ingeniería lingüística es la rama que investiga y desarrolla estas herramientas y actualmente es una industria en pleno desarrollo siendo una de sus piedras angulares, la clasificación y el etiquetado de palabras del lenguaje natural.

Aquí es precisamente en donde se hallan las mayores dificultades, puesto que la complejidad y extensión de algunas lenguas es inmensa. Es una costumbre de hoy día en informática y en especial a nivel académico, utilizar todos los recursos para lograr un objetivo, sin una mesura real y objetiva del alcance y aplicabilidad práctica del desarrollo. Este es precisamente el motivo por el cual sistemas de reconocimiento lingüístico actuales descansan sobre enormes bases de datos y potentes procesadores, los cuales están generalmente disponibles en centros especializados. Para peor, los datos lingüísticos cambian conforme aparecen nuevas acepciones y se perfeccionan o añaden reglas gramaticales y ortográficas, lo cual implica una constante actualización.

Uno de los motivos de las dificultades en esta área es, precisamente por la enorme cantidad de datos que deben estar almacenados [6], la ambigüedad de los mismos [7] y sus estructuras, presentes en cada nivel sucesivo de análisis [24]. En el inglés, las flexiones de los verbos son apenas 4 por verbo, los sustantivos y adjetivos solamente existen en dos formas *(plural y singular*), los artículos no poseen género ni número.

En cambio en el español, solamente la flexión de los verbos involucra 3 personas, 2 números, 4 tiempos, 3 modos y ciertas combinaciones especiales; resultando cerca de 70 formas verbales simples. Si a esto anexamos las formas compuestas y los pronombres enclíticos anidables con sus géneros y números, tenemos varias centenas de posibles palabras y grupos a considerar por cada verbo [12]. Con los sustantivos y adjetivos, suceda algo similar [11] si consideramos los *"accidentes"* como sufijos y prefijos diminutivos, superlativos y peyorativos; además del plural, singular, el neutro y el colectivo *(para grupos)*. Como es de esperar, la dimensión de estas cifras es casi inmanejable sin algoritmos de reducción o bases de datos enormes. Un diccionario completo para el español que contenga las formas más usadas, tendría más de 300 millones de palabras [6], ocupando del orden de entre 2 y 5 Terabytes, dependiendo del

método de almacenamiento y la cantidad de información de etiquetado por palabra. Estos 2-5 Terabytes se calculan asumiendo palabras de 3.5 letras como promedio y etiquetas simples: lema, clasificación gramatical, persona, modo, tiempo, género y número. Esto no involucra información semántica ni palabras relacionadas: sinónimos, antónimos, meronimias e hiperonimias [7], entre otras. Para etiquetado POS *(Part-of-speech)*, se utilizan diferentes mecanismos y muy a menudo se recurre a técnicas estadísticas [2] [3] con buenos resultados, pero el etiquetado previo sigue siendo un problema complejo.

Se presenta en este trabajo un sistema robusto, capaz de realizar lematización y etiquetado de palabras, manejado por reglas simples y reversibles con capacidad de tolerar errores. Además se implementa un algoritmo de flexión y reconocimiento de idiomas [1].

La contribución del trabajo consiste en haber creado nuevos algoritmos para reconocimiento, lematización y extracción semántica simultáneos, usando formatos originalmente creados para revisión ortográfica [13]. Hemos incorporado a éstos, nuevas estructuras de datos [20] [21] manteniendo la compatibilidad con formatos de archivos anteriores [13]; pudiendo utilizar diccionarios y reglas [22] de dominio público y extenderlos fácilmente con nuevas funciones. Complementariamente se ha perfeccionado un algoritmo estadístico de reconocimiento de idiomas [1]. También se ha incorporado y adaptado al español un algoritmo de similitud fonética entre palabras *(ref: Metaphone* [4], *Soundex* [5]*, ambos ideados originalmente para el inglés)*.

La sección 2 discute y describe la estructura de datos interna, la sección 3 expone el formato usado para las reglas morfológicas extendidas. El algoritmo de lematización se describe en la sección 4. En la sección 5 discutimos factores de performance emergentes del algoritmo y las estructuras empleadas. La sección 6 muestra un ejemplo de ejecución y finalmente en la sección 7 presentamos conclusiones y perspectivas.

2. Clasificación de Palabras

Un sistema de clasificación de palabras puede parecer simple, aún con muchas palabras posibles, existiendo muchas técnicas de compresión [8] que llegan cerca del almacenamiento óptimo. El verdadero problema no es solo como ahorrar espacio, sino que para poder usar los datos en búsquedas rápidas, hay que tenerlos disponibles en estructuras adecuadas [25] [26] *(árboles)*. Las bases de datos SQL, son la solución comercial escogida por lo general en ambientes académicos y empresas comerciales. Estas suelen poseer tamaños importantes de instalación, costos de licencia y un alto consumo de recursos, no resultando apropiadas para búsquedas parciales, palabras con letras cambiadas, faltantes o insertadas. Además el estándar SQL no es óptimo pues sus estructuras de índices no son apropiadas para búsquedas parciales *(por comienzo y terminación de palabras)*. Queda por analizar la posibilidad de organizar todos los datos en memoria que es un recurso limitado y costoso. Para los sistemas de 32 bits hogareños actuales hay un límite práctico de 2.0 Gb de RAM y esto no alcanzaría ni sería viable para contener una base de datos plana en memoria para el español.

La solución que hallamos, es el poder representar los datos parcialmente en memoria mediante combinaciones de reglas con árboles tipo TRIE [20] y TST [21] adaptados, los que son útiles como índices para asegurar búsquedas veloces, así como también para

poder usarlos en búsquedas inversas. Otras implementaciones de sistemas para clasificación morfo-sintáctica para *NLP (Natural Lenguaje Processing)*, como ser el *FreeLing* [10] si bien no documentan el uso de memoria y recursos, los hemos probado y determinando el costo computacional de proceso, es varios órdenes de magnitud mayor al nuestro *(lee alrededor de 50 Mb del disco a memoria antes de cada proceso)*. El *FreeLing*, puede etiquetar heurísticamente pero no presenta una lista de palabras *"similares"* ni puede corregir errores de ortografía, tampoco ofrece la facilidad de detectar el idioma; en cambio implementa un parser estadístico superficial *(shallow parser)*, bastante interesante [15]. Existen recientemente publicadas, aplicaciones similares a la nuestra para el español [11] [12] pero ellas tratan en forma independiente las formas verbales y sustantivas; no hemos podido acceder a las cifras de performance ni al tamaño de sus bases de datos *(presumiblemente importante y vía SQL)* ni a la memoria requerida para su ejecución.

3. Reglas para prefijos y sufijos

Las reglas de flexión para prefijos y sufijos, se han diseñado basándonos en un estándar de reglas morfológicas llamado compresión AFFIX [13] ampliado especialmente por nosotros para permitir recuperación y expresión gramática y semántica.

Este formato expresa la síntesis de una flexión, incluyendo las modificaciones que sufre la raíz o forma canónica al flexionarse. La notación propuesta puede describir patrones de coincidencia bastante sofisticadas, usando expresiones regulares [14], expresando la transformación en modo directa y fácilmente entendible. Se ha diseñado un algoritmo eficiente y reversible para ambos sentidos: el análisis y la síntesis.

Flexionando, transforma la raíz canónica en la flexión, aplicando la regla y en modo de análisis, conociendo la regla y la palabra flexionada se pueda hallar la raíz canónica.
Se diseñó un mecanismo alternativo, en donde dada una raíz y un conjunto de condiciones gramaticales, junto a condiciones semánticas, el sistema es capaz de armar una palabra flexionada resultante. Por ejemplo: *lema [amar] + presente + indicativo + primera persona + singular + reflexivo:(singular+1º persona)* → *amarme*

Esto proporciona una capacidad de generar palabras "sintéticas" que pueden no pertenecer al diccionario sin perder capacidad expresiva, en especial cuando se usan prefijos y sufijos en formas no verbales (médicos, latinos, griegos, superlativos, diminutivos, peyorativos). Esta capacidad de síntesis es similar a la de un hablante natural, quien conoce los afijos por su función semántica y los aplica para modificar o colorear el significado de una palabra o concepto, ignorando si la misma está presente en algún diccionario formal o de referencia como la DRAE [16] o algún otro.

3.1 Formato de Reglas

El algoritmo usa las palabras almacenadas en su forma canónica, seguida de un indicador del conjunto de reglas que son aplicables para flexionarla, usando un símbolo luego de un separador '/'. El número de reglas parciales se han agrupado por función gramática, tanto para prefijación como para sufijación, resultando en un número lo suficientemente pequeño para poder usar una sola letra *(menos de 64 símbolos)*.
Este mecanismo, al expresar el tipo de cambio o accidente gramatical que sufre la palabra raíz permite que durante el proceso de análisis, se obtenga una clasificación

inmediata. La misma puede involucrar aspectos tanto gramaticales como semánticos. El mecanismo de expresión tiene la capacidad de que una palabra posea diferentes funciones, significados *(poli-semitismo)* o etiquetas para su raíz en sus grupos de flexiones, para esto se prevé una serie de etiquetas, adicionadas luego de las reglas, separados por un separador '/'. Luego del mismo posee opcionalmente una etiqueta de caracterización fonológica *(suena como)*, con el mismo tipo de separador '/'.

Para permitir una construcción simple de las reglas, se utilizó el formato ISPELL [17], el cual permite una expresión muy clara y didáctica a la hora de sintetizar las reglas, permitiendo además comentarios. Se creó un módulo capaz de importar este formato, verificando su integridad y generando luego el formato más compacto de representación ASPELL[13] llamado *"affix"*.

Las palabras raíz, en el formato *"affix"* se representan del siguiente modo:

```
AMAR/XYV/VMN/3M2R

línea ::= palabra [/[reglas]/[etiquetas][/sound] [#comentario]<CR>
reglas ::= Letra1 [ Letra2..]
etiquetas ::= Etiqueta1 [, Etiqueta2..]
sound ::= Sonido1 [,Sonido2..]
```

Lo cual indica que la raíz es **AMAR** y las reglas aplicables son "**X**" "**Y**" y "**V**"; su etiqueta "**VMN**" indica: verbo regular infinitivo y suena como "**3M2R**". Todas son optativas.
Veamos una la regla "**V**" en detalle, expresada en formato ISPELL [17]:
(*NOTA:* todo lo que sigue al "**#**" es considerado comentario, salvo las meta-funciones a principio de línea: **#GT, #GR, #GS y #GP**)

```
[suffix]
flag *V: # Verbos de todas las conjugaciones regulares en
PRESENTE
#GS verbo regular infinitivo
#GR verbo regular
#GT PRESENTE INDICATIVO
#GP primera singular
    A R          > -AR, O          # amar amo
    [^CG] E R    > -ER, O          # comer como
    C E R        > -CER, ZO        # vencer venzo
    G E R        > -GER, JO        # coger cojo
    [^CGU] I R   > -IR, O          # vivir vivo
    C I R        > -CIR, ZO        # esparcir esparzo
    G I R        > -GIR, JO        # fingir finjo
    G U I R      > -UIR, O         # distinguir distingo
    Q U I R      > -QUIR, CO       # delinquir delinco
```

En este ejemplo se observa que la primera línea **[suffix]** caracteriza la sección para describir reglas de sufijación, luego **flag *V:** indica el nombre de la regla "**V**". Le suceden una serie de líneas comenzando con letras o grupos, separados por espacios. Cada uno de ellos expresa una terminación, en el orden correspondiente, la cual debe coincidir con la terminación de la palabra raíz en cuestión. Para poder ser aplicada, en este caso a palabra **AMAR**, la regla que coincidirá será la primera. El símbolo "**>**" indica la transformación, luego se sucede el **> -AR**, que indica que hay que quitar esa partícula, luego "**, O**" indica lo que hay que agregar, una sola letra en este caso para transformar

AMAR en **AMO**. Veamos el segundo ejemplo, aquí se ve una expresión regular **[^CGU]** la cual indica que en esa posición es admitida toda letra que no sea ni "**C**" ni "**G**" ni "**U**". Hay que tener especial cuidado de no expresar una regla de modo que letra a "quitar" coincida con una expresión regular, puesto que inmediatamente se vuelve no reversible (en forma única) y simple al algoritmo, puesto que en este caso habría que generar el conjunto complementario a la expresión regular para luego hallar el conjunto de palabras que pudo sufrir la transformación, tarea un tanto compleja. Cabe destacar que entre las palabras en forma canónica o raíz y su forma flexionada pueden no haber ninguna raíz en común, tal es el caso del verbo irregular **IR** cuyo pasado indicativo en primera persona es **FUI** y su gerundio *(por cierto muy irregular)* es **YENDO.**

3.2 Funciones Adicionales

Las meta-funciones **#GP #GT #GR #GS** han sido agregadas en forma de comentario para no perder compatibilidad con cualquier sistema anterior que pueda usar las reglas creadas y al mismo tiempo poder expresar cómodamente atributos de las transformaciones o *"flexiones"* en forma acumulativa. Estas luego serán traducidas a una forma no acumulativa en un formato AFFIX ampliado un poco por nosotros para contener este tipo de información. Todas estas meta-funciones son válidas desde su aparición dentro de cada regla, acumulándose en forma selectiva, por ejemplo:

```
#GS   Elementos semánticos agregado a la forma canónica
#GR   Características de base para toda la regla
#GT   Adicionales a la base (#GR), elimina acumulaciones (#GP y #GT)
#GP   Formato Parcial no acumulable, adicionado a (#GR y #GT)
```

Esta nomenclatura, se compila en tiempo de importación del archivo **.aff*, acumulando las reglas como lista de palabras, adicionadas a cada sub-regla. Estas palabras forman un conjunto el cual se usa en un formato de compresión por diccionario, resultando en unas pocas letras en formato compactado, al final de cada regla en formato AFFIX.

Este formato incluye posibilidad multi-alternativa, es decir si una transformación (regla) es aplicable tanto a sustantivos comunes como a adjetivos calificativos, la regla se traduce como:

```
#GR sustantivo común | adjetivo calificativo
```

Esto permite una flexibilidad adicional para expresar reglas en forma compacta, puesto que prácticamente toda palabra a la que esta regla es aplicable, será considerada tanto sustantivo como adjetivo, generando la lógica multiplicidad de etiquetas.

4. Lematizador

El sistema para reconocer las flexiones que ha sufrido un lema es llamado lematizador y se basa en aplicar una a una las reglas de flexión en forma inversa *(des-flexión)* hallando una presumible raíz o forma canónica. Luego a ésta forma se la busca entre las palabras del diccionario con sus reglas. Si es hallada se coteja que posea la regla desde la cual se partió y en caso positivo, ésta raíz es la forma canónica buscada.

Durante este proceso se obtiene acumulativamente, un conjunto de etiquetas, tanto la regla aplicada, como la palabra hallada pueden tener etiquetas gramaticales, mientras que las reglas además pueden además contener datos semánticos. Estas reglas se combinan mediante un mecanismo acumulativo con reglas propias de prioridad que aseguran que se construirá la o las etiquetas gramaticales apropiadas y de haberla, también se adicionará la información semántica. El resultado se expresa en una estructura de datos en formato del grupo PAROLE/EAGLES 2.0 [18] extendido con el adicionado de un conjunto de etiquetas especiales para uso semántico.

4.1 Implementación de Reglas

El mecanismo original, propuesto por ASPELL [13] ISPELL [17] y NetSpell [19], se prueban todas y cada una de las reglas de flexión contra el diccionario completo de las palabras, buscando en un diccionario basados en dispersión *(hashtable)* hasta hallar coincidencia. Esto denota si tenemos NR número de reglas y NP de palabras una cantidad equivalente de operaciones *(des-flexión y búsqueda)*: NR*NP. Además en la mayoría de los diccionarios de uso libre para formato ASPELL [13] hay una importante cantidad de palabras sin etiquetas (20-30%), entre nombres propios y formas no verbales, los cuales abultan la búsqueda sin esperar resultado alguno. Este método funciona en un modo que está muy de moda desde que el recurso de computación se ha vuelto tan popular, económico y veloz, que llamaríamos *"fuerza bruta"*.

5. Mejoras Obtenidas

El mecanismo propuesto, como veremos a continuación economiza mucho el recurso computacional al tiempo que por utilizar estructuras óptimas, permite reducir el número de operaciones significativamente, en un orden de más de 100 veces respecto a sistemas tradicionales [16] *(para nuestras bases),* trasformando mecanismos de búsqueda de datos con tiempos promedio [26] **O**(n log(n)) a mecanismos lineales [25] **O**(p). Es importante destacar que el mecanismo de búsqueda, no es proporcional al número de palabras/reglas "**n**" en el diccionario, sino que en nuestros algoritmos, solamente es sensible linealmente al número de letras "**p**" de la palabra a hallar. En otras palabras en una búsqueda binaria, el factor de cálculo es sensible a la distribución estadística y el consecuente armado del árbol en el que se buscará el dato *(de allí sale el logaritmo en base 2)*; en cambio en nuestro mecanismo, el utilizar un árbol tipo TRIE [20] hace que la búsqueda sea únicamente proporcional al número de letras de la palabra buscada y un poco al tamaño de la tabla de lemas *(hashtable = tabla de dispersión)*. Esto lo hace más eficiente, puesto que el tiempo del proceso es prácticamente independiente de la riqueza o tamaño de la base de datos morfológicos *(reglas)*. Además el proceso *"mientras"* determina el lema con su posible flexión, construye internamente una lista adicional de las palabras *"parecidas"* en forma transparente casi sin agregar tiempo de proceso.

Se agrega una regla de sintaxis adicional al sistema ASPELL [13], ésta es la de poder anidar en una misma regla terminaciones optativas (como combinaciones), esto reduce el universo de expresión de reglas, dividiéndolo por el un número de reglas que se combinen en cada grupo, por ejemplo a las reglas de flexión verbales, las redujo en promedio de 2 a 4 veces y en algunos casos de pronombres enclíticos anidados hasta 30 veces. Esto no agrega tiempo de proceso en el análisis de los cambios morfológicos y flexión; agregando flexibilidad y riqueza de expresión a la hora de escribir la regla.

Veamos como se implementó para una regla llamada "**T**"

```
flag *T:    # Verbos transitivos con gerundio regular + enclítico
#GS verbo transitivo infinitivo
#GR verbo transitivo
#E1  LO     Singular Masculino      # dormir > dormírmelo
#E1  LA     Singular Femenino       # dormir > dormírmela
#E1  LE     Singular Neutro         # dormir > dormírmele
#E1  LOS    Plural Masculino        # dormir > dormírmelos
#E1  LAS    Plural Femenino         # dormir > dormírmelas
#E1  LES    Plural Neutro           # dormir > dormírmeles
#GT infinitivo enclítico
     [AEI] R      ?
#GT gerundio enclítico
     A R    ? -AR, ÁNDO
     E R    ? -ER, IÉNDO
     I R    ? -R, ÉNDO
```

Este ejemplo ilustra la regla anidada "**#E1**" la cual es aplicable a todas aquellas terminaciones en donde el separador es "**?**" en lugar de "**>**" este separador indica que esta sub-regla, utiliza todas las posibles combinaciones indicadas "**#E1**" al principio de la regla. El mecanismo prevé la posibilidad de expresar más profundidad de anidación puesto que el "**1**" que sigue la letra "**E**" indica que ese es el grupo de anidación. En la práctica no resultó necesario puesto que eran muy pocas las reglas que lo requerían y el tiempo de proceso extra sería contraproducente, pero el formato se plantea abierto y luego del signo de pregunta se pueden especificar una secuencia de números que denotan las sub-reglas y su orden de aplicación.

El resultado de esta regla compilada es el siguiente en formato ASPELL:

```
T Y 10 O5-mmA
1*LO Opcw
1*LA Opc4
1*LE Opeg
1*LOS Opsw
1*LAS Ops4
1*LES Opug
T1 0 0 [aei]r PJ-mnqA
T1 ar ándo ar PJ-mlqA
T1 er iéndo er PJ-mlqA
T1 r éndo ir PJ-mlqA
```

Se observa claramente la correspondencia entre ambos formatos, obviamente el ASPELL es más compacto pero resulta complejo para escribirlo o editarlo directamente. Las secuencias "**O5-mmA**" son el formato compactado con diccionario de las palabras: "**verbo transitivo infinitivo**" al igual que "**Opcw**" significa "**Singular Masculino**" y para las reglas la sigla "**PJ-mnqA**" simboliza "**infinitivo enclítico**".

5.1 Estructuras de Datos

El mecanismo completo que hemos utilizado en la implementación, difiere radicalmente del original, primeramente se utiliza una estructura de árbol Trie [20] [25], para almacenar los afijos terminaciones/comienzos transformadas con capacidad de múltiples elementos por nodo, esto permite que las reglas sean seleccionadas directamente en tantas comparaciones como letras tenga el afijo, esto reduce de 4500 comparaciones [19] a apenas 3.2 *(en promedio)*, para esa regla *(mejora de 1400:1)*. Para no complicar el algoritmo y generar un Trie enorme, se cicla por las 30 reglas globales con un lazo *for-next* el cual es eficiente y no justifica ingresar los datos en un Trie unificado, simplificando el armado de las estructuras de datos *(clases)* en memoria durante la carga del diccionario. Dentro de cada regla *"flag"* hay 2 Tries, uno directo y otro reverso, ambos orientados a agilizar la selección de reglas a aplicar/invertir, basado en la palabra a analizar/flexionar. Para el caso de estructuras anidadas, se utiliza un Trie unificado el cual guarda óptimamente todas las combinaciones, con complejidad ***O****(log k + n)*. *(Mehlhorn)* [26] y [25], siendo *"k"* el número de reglas y *"n"* las letras de la terminación en cada regla, siendo éstas de 1-3 letras (*en promedio = 3 comparaciones)*

Para el almacenamiento y búsqueda de palabras en formato canónico o raíz, decidimos también utilizar una variante nuestra de TST *(Ternary Search Tree)* [21], siendo éste un mecanismo con performance muy similar [25] al de una tabla de dispersión (*hashtable*), y una impronta de memoria ligeramente mayor y optimizable. Los beneficios de este mecanismo son que permite hallar palabras con cambios de *1..N* letras y con caracteres de búsqueda *(wildchars)*, es decir encontrar palabras similares a otras con determinado patrón o cota de medida de distancia entre palabras, permitiendo corregir errores.

Para esto se implementó una variante del TST que permite hallar independientemente de los acentos y eñes, para hallar raíces aproximadas prescindiendo de los diacríticos y generar alternativas viables en caso de no hallar la palabra deseada al lematizarla. Esto resulta eficiente ya que la longitud de las búsquedas es a lo sumo N*(P!) comparaciones adicionales; donde N es el número de letras de la palabra a buscar y P es el número de "errores" permitidos *(nº de vocales+ñ/n)* cuyo promedio está entre 2 y 4.

5.2 Formatos de Diccionarios

En el formato de reglas de etiquetado de palabras se permite un efecto acumulativo, para evitar numerosas etiquetas repetidas y permitir agrupar palabra bajo un mismo rótulo. Esto resulta muy útil a la hora de agregar palabras y editar la lista de las mismas. El formato ampliado es muy sencillo de entender y se mantiene total compatibilidad con el formato anterior; sólo se agregan 2 prefijos correspondiendo "*****" a las etiquetas gramatical y "**%**" al de reglas.

Los mecanismos de compresión ideados resultan eficientes, tanto a la hora de leer el archivo (<0.3seg.) como en su tamaño total, siendo mas compactos que los utilizados por Open-Office *(OO)* [22], con más capacidad expresiva y de reconocimiento. Para el Español: *"es-ES.dic"* de *(OO)* pesa 712k con 48k palabras mientras que el nuestro tiene de 545k *(~200k compactado con zip)* conteniendo 51k palabras e incluyendo clasificación gramática y semántica completa con 526 reglas gramaticales adicionales alas originales.

6. Ejemplos de Ejecución

Analizando una frase simple: *"como el bajo comía poco"* se incluyó el nombre abreviado de la etiqueta, y entre signos "< "y ">" el lema, separando las alternativas con el signo "|", este es el listado obtenido:

```
COMO    VerbInd(VMIP1S0<comer>) | ConjSubordinada(CS<como>) |
        PronRelativo(PR000000<como>) | Interjeccion(I<como>) |
        PronInterrogativo(PT0NN000<como>) | Adverbio(RG<como>)
EL      Articulo(DA0MS00<el>)|Pronombre(PP3MS000<él>)
BAJO    Adjetivo(AQ0000<bajo>) | Sustantivo(NC00000<bajo>) | VerbInd(VMIP1S0<bajar>)
COMÍA   VerbInd(VMII1S0<comer>)
POCO    DetIndefinido(DI0MS00<poco>) | PronIndefinido(PI0MS000<poco>) |
        Adverbio(RG<poco>)
```

Usando el *módulo de análisis gramatical exacto*, obtendremos, por ejemplo:

```
larguiruchos:   NCMP000<larguirucho> Sustantivo: Comun,Masculino,Plural
                AQEMP0<largo> Adjetivo: Calificativo,Despectivo,Masculino,Plural
muchísimo:      AQSMS0<mucho> Adjetivo:Calificativo,Superlativo,Masculino,Singular
hubiésemos:     VASI1P0<haber> Verbo: Auxiliar,Subjuntivo,Imperfecto,Primera,Plural
```

El módulo de *análisis heurístico* entregará varias formas alternativas. *(arrumaco: NO pertenece a nuestro diccionario de lemas)*

```
arrumacos:      NCMP000<arrumaco> Sustantivo: Comun,Masculino,Plural
                AQ0MP0<arrumaco> Adjetivo: Calificativo,Masculino,Plural
```

6.1 Performance

Con un *PC, i386 Celeron-D 2.66GHz, 512k RAM DDR, XP y .NET 1.1* el sistema etiqueta 2700 palabras por segundo, entregando clases instanciadas *(C# .NET)* con su lista de palabras similares, las etiquetas EAGLES 2.0 (como clases) con extensión semántica y el grado de verosimilitud (+1/../-1) de cada acepción. En cambio procesa más de 53.000 palabras por segundo en búsqueda aproximada con errores gramaticales (diacríticos). Esto fue realizado sobre un corpus de 2666 palabras de 5.02 letras en promedio texto: *"Verdad y Perspectiva de Ortega y Gasset"*. Ocupando apenas 12 Mb de memoria RAM, a pesar de la alineación de 4 bytes en 32 bits y al almacenamiento en Unicode del C# (2 bytes por letra). La tasa de detección de errores es del 89.64% y la tasa de reconocimiento gramatical exitoso fue de 38% para este texto ya que contiene una importante cantidad de palabras y nombres propios. El sistema fue utilizado con un diccionario con ~51.000 formas canónicas y ~4500 reglas de flexión. La velocidad de estimación de idioma, entre alemán, inglés y español, fue de 21.300 palabras por segundo. Procesando el diccionario del FreeLing 1.4 *(~70.600 palabras)*, el sistema etiqueta ~70.000 correctamente; analizando en detalle las 600 no halladas hallamos mayormente errores del diccionario FreeLing (por conjugación y/o flexión erróneas e irregulares). Algunos analizadores morfológicos [22] [23] utilizan bases de datos de 19 Mb con un número similar de términos a la nuestra, pero no hallamos información de su performance y requisitos del sistema.
Los errores de etiquetado del sistema son solamente dependientes de la calidad del diccionario interno, el cual estamos depurando día a día con fuentes fidedignas [16].

7. Conclusiones

El sistema presentado, conforme a las intenciones de su diseño, posee la capacidad de manejar diferentes niveles de reconocimiento, desde exacto hasta aproximado, brindando luego de cada análisis, un conjunto de palabras alternativas "similares" útiles para una posible evaluación adicional en un proceso posterior de *NLP*.
El sistema posee una velocidad útil y un bajo consumo de recursos computacionales, tanto de proceso como en almacenamiento. Esto lo hace candidato para aplicaciones de móviles y embebidas del mundo real: módulos *"plug-ins"* para procesadores de texto, reconocedores de lenguaje, traductores, sistemas de comandos y diálogo, búsqueda de respuestas y otros que requieran procesamiento lingüístico robusto, complejo y veloz. Resultó práctico la incorporación de un algoritmo de detección idiomática [1], con un costo de apenas un par de sumas y unos pocos Kb de memoria. Las mediciones de nuestro algoritmo de similitud fonética requieren de cuidadosos experimentos perceptivos realizados con personas, los cuales están en desarrollo junto con un grupo de investigación en *Biología del Comportamiento* del *I.B.Y.M.E.* y son motivo de un próximo trabajo. Simultáneamente estamos trabajando activamente en un *"parser"* GLR de español usando heurística + estadística apuntado a un nuevo modelo de gramática operacional, orientado a comprensión y razonamiento.

7.1 Aplicaciones Posibles

Los sistemas informatizados de uso cotidiano como las PC's, PDA's, Tablet-PC y Teléfonos Celulares, se están convirtiendo en elementos de uso cada vez más frecuente. Una de los principales problemas de esta explosión de tecnología y prestaciones es precisamente la complejidad de las mismas y en especial la interfase con el usuario.

Actualmente se usan teclados, ratones, sensores de presión y posición combinados con pantallas para lograr funciones de interacción complejas, pero necesitan realimentación perceptiva combinada y compleja (*visual: luces/display; sonora: parlantes/clics)*. Una alternativa lógica es sin duda, comunicarse directamente en lenguaje natural. Precisamente el procesamiento de texto natural *(NLP)*, ha abordado estos temas con los enormes desafíos planteados por la dificultad de modelar mecanismos *"naturales"* como el habla y la comprensión de ideas mediante software. Estas aplicaciones no serían posibles sin sistemas que aportan la posibilidad de poder reconocer palabras tanto en forma precisa como aproximada, incluyendo su idioma, junto al poder sintetizar las mismas mediante información gramatical y semántica simultáneas. Todo esto realizado en forma eficiente y con poco uso de memoria, permitirá sin duda la incorporación de herramientas con algoritmos lingüísticos a dispositivos portátiles y de uso cotidiano con limitaciones de memoria y procesamiento. Estos dispositivos, muñidos de sistemas de procesamiento de lenguaje natural, mejorarán sin duda nuestra calidad de vida y nos ayudarán cada vez más y mejor con las tareas cotidianas. Más precisamente, la posibilidad de poseer mecanismos de síntesis y reconocimiento de palabras eficientes y veloces permitirá toda una nueva clase de aplicaciones. Finalmente, la capacidad combinada y robusta de lematizar, flexionar palabras y reconocer idiomas [1] por *"como suena"* lo hace candidato a ser utilizado en sistemas de diálogo en lenguaje natural de uso cotidiano y masivo.
El sistema fue programado en lenguaje C# *(MS .NET 1.1)* el cual permite portabilidad a plataformas móviles y embebidas *(incluso a Java 2 y Java2ME).*

7.2 Perspectivas

La computación nos brinda claras muestras de tornarse cada vez más poderosa, veloz, compacta y eficiente *(en términos de consumo)*. Gran mayoría de estas virtudes, son derivadas de la aplicación intensiva de mejoras tecnológicas en los procesos de fabricación y arquitectura interna, junto a componentes cada vez más veloces, eficientes y pequeños. Este devenir de virtudes debe de estar necesariamente acompañado con prestaciones cada vez más amigables al usuario, achicando la brecha entre el humano y el computador. Sin duda los algoritmos y métodos de procesamiento, no tienen la misma curva de crecimiento cuasi-geométrica de la miniaturización y eficiencia en términos de potencia de cálculo. Será pues necesario mucho trabajo para no desperdiciar el enorme poder de procesamiento disponible. El resultado, según vislumbramos, tenderá ciertamente a una relación más fluida del hombre con las computadoras. Es de esperarse que esta interacción pueda realizarse en el futuro, en lenguaje natural. Este trabajo es sin duda, un aporte en esta dirección.

Bibliografía

[1] Hohendahl, Andrés T. & Zelasco, José F. 2006. Algoritmos eficientes para detección temprana de errores y clasificación idiomática para uso en procesamiento de lenguaje natural y texto, WICC2006 - ISBN 950-9474-35-5

[2] Ratnaparkhi, Adwait. 1996. A Maximum Entropy Part-Of-Speech Tagger. *EMNLP'96*

[3] Zhou, Guodong and Jian Su. 2002. Named Entity Recognition using an HMM-based Chunk Tagger

[4] Lawrence Phillips, Hanging on the Metaphone, *Computer Language*, vol. 7, no. 12, pp. 39-43, 1990. The Double Metaphone Search Algorithm, *C/C++ Users Journal*, 18(6), June, 2000.

[5] The Soundex Algorithm: http://www.archives.gov/research_room/genealogy/census/soundex.html

[6] Diccionarios españoles: http://www3.unileon.es/dp/dfh/jmr/dicci/012.htm

[7] Pedro Luis Díez Orzas 1999. Estudios de Lingüística Española LA RELACIÓN DE MERONIMIA EN LOS SUSTANTIVOS DEL LÉXICO ESPAÑOL: CONTRIBUCIÓN A LA SEMÁNTICA COMPUTACIONAL Volumen 2 (1999) ISSN: 1139-8736

[8] Shannon, Huffman compression: http://www.cbloom.com/algs/statisti.html

[9] Estructuras de árboles "Trie": http://www.nist.gov/dads/HTML/trie.html

[10] FreeLing: www.lsi.upc.es/~nlp/freeling/

[11] FLANOM: Flexionador y lematizador automático de formas nominales. Santana, O.; Pérez, J.; Carreras, F.; Duque, J.; Hernández, Z.; Rodríguez, G. *Lingüística Española Actual XXI, 2, 1999. Ed. Arco/Libros, S.L. 253/297*

[12] FLAVER: Flexionador y lematizador automático de formas verbales. Santana, O.; Pérez, J.; Hernández, Z.; Carreras, F.; Rodríguez, G. Lingüística Española Actual XIX, 2, 1997. Ed. Arco/Libros, S.L. 229/282

[13] ASPELL Affix compression: http://aspell.sourceforge.net/man-html/Affix-Compression.html

[14] Expresiones Regulares: http://www.regular-expressions.info/

[15] Padró, L. (1996). POS tagging using relaxation labelling. In Proceedings of the 16th International Conference on Computational Linguistics (COLING-96). http://citeseer.ist.psu.edu/169791.html

[16] DRAE Diccionario de la Real Academia Española http://buscon.rae.es/diccionario/drae.htm

[17] ISPELL www.gnu.org/software/ispell/ispell.html

[18] Etiquetas EAGLES / PAROLE www.lsi.upc.es/~nlp/tools/parole-sp.html

[19] NetSpell http://sourceforge.net/projects/netspell/

[20] TRIE http://www.cs.bu.edu/teaching/c/tree/trie/

[21] TST -Ternary Search Tree: http://www.nist.gov/dads/HTML/ternarySearchTree.html

[22] Open Office Dictionaries: http://lingucomponent.openoffice.org/spell_dic.html

[23] http://www.gedlc.ulpgc.es/investigacion/desambigua/morfosintactico.htm

[24] Relaciones morfoléxicas prefijales del español. Santana, O.; Carreras, F.; Pérez, J.; Rodríguez, G. Boletín de Lingüística, Vol. 22. ISSN: 0798-9709. Jul/Dic, 2004. 79/123.

[25] J Bentley & R - Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 1997

[26] Mehlhorn, K. Dynamic Binary Search. SIAM Journal on Computing 8, 2 (May 1979), 175-198.