Introducción

El almacenamiento de grandes volúmenes de datos textuales en un ordenador es un proceso relativamente sencillo, el problema se plantea a la hora de manejar y recuperar en un tiempo aceptable la información almacenada; la búsqueda de cadenas es la parte fundamental de este problema.

La búsqueda de coincidencias de cadenas consiste en encontrar las ocurrencias de un modelo o patrón en un texto, siendo tanto el patrón como el texto cadenas tomadas de algún alfabeto. Los algoritmos clásicos –válidos para documentos cortos– [KM77], [BM77] y [HO80], realizan la búsqueda de coincidencias directamente sobre el texto en un tiempo que es función lineal de la longitud del mismo; existen posteriores optimizaciones para ciertos casos, [SU90], [CO91] y [HS91].

En la práctica, también se necesita a menudo analizar situaciones donde los datos no son del todo correctos. Considérese la situación en la que la cadena de entrada contiene errores de sustitución o presenta un cierto número de diferencias con el patrón, [BY89], [TU90] y [UW90], y de todas formas es necesario encontrar las apariciones del patrón en el texto.

Se podría suponer que lo que se proporciona como cadena de búsqueda es precisamente algo parecido a lo que previamente ha sido almacenado en algún registro o registros. La cadena de búsqueda no coincide exactamente porque ha sido modificada por algún proceso de distorsión. Sin embargo, puede ocurrir que una cadena que haya sufrido alguna alteración se parezca mucho e incluso coincida con otra previamente almacenada; el objetivo de la búsqueda es precisamente la recuperación de estas cadenas.

Diversos han sido los autores que han estudiado el concepto de similitud o distancia entre dos cadenas. Alberga, [AL67], usó matrices binarias para evaluar una medida de la disimilitud entre cadenas. Szanser, [SZ73b], desarrolló un proceso matemático de coincidencia elástica que era efectivo en un 95%. Morgan, [MO70], realizaba un o_exclusivo entre dos cadenas para determinar si había ocurrido un único error simple. Wagner y Fischer, [WF74], desarrollaron un algoritmo de programación dinámica que evalúa la distancia entre dos cadenas, medida como el mínimo coste de la secuencia de operaciones de edición (sustitución, extracción e inserción). Ukkonen, [UK83], ideó una forma de cálculo eficiente de la matriz de diferencias de Levenshtein, [LE66], usando sus diagonales. Landau y Vishkin, [LV85a], [LV85b], [LV86a] y [LV86b]; Galil y Giancarlo, [GG86], y el propio Ukkonen, [UK85], utilizan en sus trabajos, con las adaptaciones necesarias, la computación eficiente de la distancia de edición introducida en [UK83]. Tanaka y Kojima, [TK87], continúan utilizando la introducida por Wagner y Fischer, [WF74].

Si se plantea el hecho de corregir los errores introducidos por la distorsión, mediante el proceso de recuperación se obtiene una tentativa de corrección de errores, y con la recuperación de una cadena no relevante se obtiene un error. Este punto de vista detección/corrección se adopta en teoría de comunicaciones y reconocimiento de patrones.

Los métodos de corrección de cadenas se pueden clasificar en dos categorías: los métodos estadísticos que usan N-gramas, probabilidades de confusión y de ocurrencia; y los métodos de diccionarios que están basados en similitudes o distancias. En general, los métodos estadísticos son más rápidos que los que usan diccionarios, mientras que los métodos de diccionarios pueden obtener mejores tasas de corrección que los estadísticos.

El algoritmo de Viterbi clásico, [FO73], y sus modificaciones, [NE75], [ST79] y [HS82], han sido sugeridos para corregir solamente errores de sustitución. La característica principal del algoritmo de Viterbi es que usa un modelo de Markov del diccionario de entrada, caracterizado por sus unigrama y digrama de frecuencias. La gran desventaja de estos algoritmos radica en un excesivo cálculo, que es independiente del tamaño del diccionario. La bondad de la corrección tan sólo alcanza valores en torno al 50%, [NE75]. Es bien sabido que, para mejorar la realización, es necesario usar el diccionario y no únicamente su modelo estadístico. Kashyap y Oomen, [KO85], proponen una mejora para la corrección de cadenas de longitud pequeña.

La principal dificultad de los métodos que usan la distancia de Levenshtein entre dos cadenas, con el objeto de elegir las cadenas más parecidas a la de búsqueda de entre las que forman el diccionario, radica en el hecho de que se ha de comparar cada cadena de búsqueda con la base de datos entera, o gran parte de ella, [OT76] y [TK87]. Existen algunos métodos de corrección, [RE71], [RH74] y [UL75], que utilizan el digrama o trigrama de frecuencias. Sin embargo, se pone de manifiesto, [OT76], que tales métodos son inferiores a los que utilizan la distancia de Levenshtein.

Recientemente, los investigadores han tomado un creciente interés por los métodos rápidos de corrección o búsqueda para vocabularios grandes. Una línea es encontrar métodos estadísticos con alta tasa de corrección: Kurita y Aizawa, [KA84], estudian un método rápido para corregir sólo errores de sustitución; Shinghal, [SH83], obtiene una tasa de corrección ligeramente menor pero resulta casi diez veces más rápido que el método de diccionario. Otra línea consiste en aumentar la velocidad de los métodos de diccionario: Itahashi y Yokoyama, [IY84], logran una mejora basada en la

selección de algunos fonemas; Tanaka, [TK86], presenta otro método de corrección para errores de sustitución, [TT86], posteriormente extendido a errores de inserción y extracción; Kaneko y Dixon, [KD83], discuten un método de decisión jerárquica para un vocabulario grande en un reconocedor de voz; Hall y Dowling, [HD80], e Ito y Kizawa, [IK82], también discuten técnicas de corrección de errores en ristras para ficheros grandes y organizados jerárquicamente. Tanaka y Kojima, [TK87], proponen un método multietápico de alta velocidad para la corrección de ristras de fonemas, basado en la distancia de Levenshtein, usando ficheros jerarquizados.

Cada cadena puede considerarse como un punto en el espacio multidimensional de secuencias de caracteres, donde no todas las secuencias de caracteres son posibles. Los métodos para la recuperación de datos multidimensionales permiten llevar a cabo búsquedas asociativas, del tipo que interesan a este trabajo, teniendo que acceder tan sólo a una porción reducida de la base de datos en cuestión. Los algoritmos más conocidos se encuentran en los trabajos de Bentley y Finkel, [FB74], [BE75], [FB77], [BE79] y [BE85], quienes proponen los árboles Quad y los K-D que son estructuras basadas en la comparación de claves; y Burkhard y Keller, [BK73], que presentan la estructura BK que se organiza a partir de las distancias.

Las bases de datos documentales requieren métodos de búsqueda más rápidos que los mencionados inicialmente –tiempo lineal respecto de la longitud del texto. Para ello se someten los escritos a un tratamiento previo, al objeto de construir una estructura auxiliar que aumente la velocidad de las ulteriores consultas. Estos métodos de búsqueda en texto, [FA85] y [FB92], pueden clasificarse en tres categorías: índices lexicográficos, [GO87]

y [MM90]; índices basados en dispersiones, [HA71], [RI77], [LA83], [FC84] y [FA88]; y técnicas de agrupamiento, [SS85] y [WI88].

Este trabajo trata aspectos teórico-prácticos en torno al problema de la búsqueda de las cadenas más similares a una dada en grandes volúmenes de datos. El concepto de similitud se entiende en el sentido de la distancia de Levenshtein, **DL**. Dado un diccionario de cadenas y una distancia en el espacio de las mismas, las cadenas más similares a una dada –que puede encontrarse o no en el diccionario– son todas las del diccionario que se encuentran a distancia mínima de la proporcionada. Se construye como índice una estructura de datos que evita el recorrido secuencial del diccionario. El objetivo que se persigue es la optimización de los recursos de tiempo y espacio, de los esquemas de búsqueda y de la estructura de datos que los soporta.

Planteados los antecedentes y el marco en el que se sitúa el presente trabajo se pueden concretar y resumir los capítulos de la siguiente forma:

En el Capítulo 1, se define la distancia de Levenshtein, [LE66], con el cálculo propuesto por Wagner y Fisher, [WF74], y la optimización de Ukkonen, [UK83]; asimismo se introduce una nueva medida que se ha denominado distancia invariante trasposicional, **DIT**, debido al hecho de que su valor no depende de las operaciones de trasposición a que pueda ser sometida una cadena, [SD87], y cumple las propiedades de distancia entre cadenas sin considerar la disposición secuencial de sus caracteres. Si bien **DIT** no puede usarse por sí sola para la determinación de las cadenas más similares en el sentido de Levenshtein, su importancia deviene de la circunstancia de que su valor entre dos cadenas es siempre inferior o igual a la **DL** entre estas dos mismas cadenas, siendo su coste computacional sensiblemente inferior; tales propiedades aconsejan la construcción de un

filtro adaptativo **DIT**|**DL** que tenga por misión reducir el número de cadenas de la base de datos a las que se les calcula la **DL** con la cadena de búsqueda.

En el Capítulo 2, se diseña una estructura cuyo propósito es llevar a cabo una compartición de las componentes de **DIT**, a fin de no tener que calcular completamente la **DIT** de la cadena de búsqueda a todas y cada una de las cadenas del diccionario. Esta estructura fue diseñada por Santana y Díaz, [SD87] y [SD89], y se denomina estructura **S-D**. Se estudian criterios para la reducción de la longitud del camino promedio, [SD87]. Se comprueba la evolución del comportamiento de esta estructura con los tamaños de los diccionarios. El esquema de búsqueda que se apoya en la estructura **S-D**, recorriéndola a través de las componentes de **DIT** y que usa este valor como criterio de poda, se denomina esquema *decreciente*, [SD87]. Se estudia un nuevo esquema de búsqueda denominado *creciente*, [DS90] y [SP90], donde el radio de búsqueda, en oposición a la evolución clásica decreciente, sigue una línea de modificación creciente. Además, se propone un esquema *decreciente* con radio ascendente tal que en función del incremento del radio de búsqueda define una *familia de esquemas* intermedios que conectan a los esquemas *creciente* y *decreciente*, [DS90]. Se presentan los resultados de los experimentos realizados con los diferentes esquemas.

Prolongando la línea de optimización de las realizaciones de los esquemas de búsqueda *decreciente* y *creciente* de las cadenas más similares a una dada en la estructura **S-D**, en el Capítulo 3, se recomienda una nueva poda en el índice que acorta su recorrido, [SR90], comprobándose experimentalmente su eficacia. Se define un nuevo filtro, **DS**, en función de

la subsecuencia común más larga entre cadenas, [SR90], que recoge propiedades ignoradas por **DIT**; se demuestra que es una distancia, [DS93], que cumple $DIT \leq 2*DS \leq 2*DL$ y que tiene un menor costo computacional que **DL**. Se verifica experimentalmente que dicho filtro es útil para descartar un número de cadenas a las que no es necesario evaluar su **DL** con la de búsqueda.

Si el tamaño de la estructura **S-D** es tal que no es posible ubicarla en memoria interna; en el Capítulo 4, se plantea el problema de su paginación con el fin de minimizar el número de accesos a disco. Los criterios a tener en cuenta son: el tiempo de respuesta a las peticiones y la ocupación. No se consideran los problemas relativos al mantenimiento –inserciones, extracciones y reorganizaciones– debido al carácter estático del diccionario. Un primer intento para resolver el problema consiste en llenar las páginas con los nodos al recorrer la estructura en *preorden* o en *postorden*. También se propone una nueva forma de paginar la estructura, *segunbusca*, de tal modo que se respete el siguiente principio: durante la búsqueda, una vez que se abandone una página no se vuelve a acceder a ella por otro camino. Todas las propuestas se ensayan experimentalmente.

En el Capítulo 5, se aplica la estructura **S-D** y los esquemas de búsqueda estudiados aportando una solución a la localización de palabras en texto libre; ello constituye una parte fundamental de un problema práctico de gran importancia: la organización y utilización de información procedente de fuentes heterogéneas. Los escritos se someten a un tratamiento previo; la intención es generar un índice, **S-D**, que haga viable los accesos posteriores al ejemplar cuando sea necesario. El corpus que se indiza, como ejemplo, para llevar a cabo las localizaciones textuales es un Diccionario de

Medicina, [JV87]. Se permiten los siguientes tipos de búsquedas: *exacta*, *más similares*, con *operadores booleanos*, *máscaras*, *truncamientos*, *cercanía*, *antecedencia*, *párrafos*, *sentencias*, *frases* y *compleja*, [SD92]. Además se construye un analizador sintáctico, [SD92], que cumple un doble cometido, ya que ha de distinguir cada tipo de búsqueda; en el caso de las *complejas*, diferenciar las componentes y los *conectores lógicos* y, a la vez, determinar la corrección sintáctica de cualquier petición. Como complemento al analizador, se dispone de un optimizador con el fin de obtener la solución de una búsqueda *compleja* en el menor tiempo posible. Por último, para hacer la aplicación más accesible a un mayor número de usuarios, se implanta en un ordenador personal utilizando la paginación que ha resultado más adecuada para el índice. Se realizan estudios experimentales que ilustran la aplicación.

Las conclusiones y principales aportaciones se recogen en el Capítulo 6.

Capítulo 1.
Distancias entre Cadenas

Sea **X** una cadena de caracteres sobre un alfabeto $\{\alpha_1,..., \alpha_m\}$, y sea μ la cadena nula –sin caracteres. Se denota por $X<i>$ el carácter que está en la **i**-ésima posición de la cadena **X**, $X<i,j>$ la secuencia de caracteres que va de $X<i>$ a $X<j>$ ambos inclusive; si $i>j$ entonces $X<i,j>=\mu$. $|X|$ denota la longitud de la cadena **X**.

Una operación de edición es el par $(\Phi,\Omega)\neq(\mu,\mu)$ donde Φ y Ω son cadenas de longitud menor o igual que uno, es decir, o son μ o son un único carácter. La cadena **Y** resulta de aplicar (Φ,Ω) a la cadena **X** y se denota por $X \rightarrow Y$, si $X=\sigma\Phi\pi$ e $Y=\sigma\Omega\pi$, donde σ y π son cadenas de longitud mayor o igual que cero.

Wagner y Fischer, [WF74], consideran tres tipos de operaciones de edición. (Φ,Ω) es una operación de *sustitución* si $\Phi\neq\mu$, $\Omega\neq\mu$ y $\Phi\neq\Omega$; es una operación de *extracción* si $\Omega=\mu$ y es una operación de *inserción* si $\Phi=\mu$.

Sea **S** una secuencia $\mathbf{S_1, S_2,..., S_n}$ de operaciones de edición. Una *derivación_S* de **X** hasta **Y** es una secuencia de cadenas $\mathbf{X_0, X_1,..., X_n}$ tal que $X=X_0$, $Y=X_n$ y $X_{j-1} \rightarrow X_j$ vía S_j $\forall j=1,...,n$.

1.1. Distancia de Levenshtein, DL.

Sea Γ una función arbitraria de costo que asigna a cada operación de edición (Φ,Ω) un número real no negativo $\Gamma(\Phi,\Omega)$. Se puede extender Γ a la secuencia **S** definiendo:

$$\Gamma(S) = \begin{cases} \sum_{j=1}^{n} \Gamma(S_j) & si\ n \geq 1 \\ 0 & si\ n = 0 \end{cases}$$

<u>Definición</u>: *Se llama distancia de Levenshtein, DL(X,Y), entre las cadenas **X** e **Y**, al mínimo costo de todas las secuencias de edición que transformen **X** en **Y**. Formalmente DL(X,Y)=min{*$\Gamma(S)$*}.*

DL, también conocida como distancia de edición, fue introducida por Levenshtein, [LE66], y evaluada por Wagner y Fischer, [WF74], a través de un algoritmo de programación dinámica.

En este trabajo se sigue el criterio de que el costo de cualquier operación de edición se considera *unitario*.

1.2. Cálculo de Wagner y Fischer de la Distancia de Levenshtein.

Si **X** e **Y** son dos cadenas cualesquiera, se define *X(i)=X<1,i>*, *Y(j)=Y<1,j>* y *WF(i,j)=DL(X(i),Y(j))*.

$$WF(0,0) = 0$$

$$WF(i,0) = \sum_{r=1}^{i} \Gamma(X\langle r\rangle, \mu)$$

$$WF(0,j) = \sum_{r=1}^{j} \Gamma(\mu, Y\langle r\rangle)$$

Es decir, el costo de convertir μ en sí mismo es cero, el costo de reducir la cadena **X** a μ es la suma de los costos de extraer todos los caracteres de **X** y el costo de obtener **Y** a partir de μ es la suma de los costos de añadir cada uno de los caracteres de **Y**.

Para calcular *WF(i,j) i=1..* $|X|$, *j=1..* $|Y|$ hay que tener en cuenta tres casos.

1.- Convertir *X(i-1)* en *Y(j-1)* y *X<i>* en *Y<j>*.

2.- Convertir *X(i-1)* en *Y(j)* y extraer *X<i>*.

3.- Convertir *X(i)* en *Y(j-1)* y añadir *Y<j>*.

De estas tres posibilidades se escoge aquella cuyo costo sea mínimo.

Por lo tanto:

$$WF(i,j) = \min\{\ WF(i-1,j-1) + \Gamma(X\langle i\rangle, Y\langle j\rangle),\ WF(i-1,j) + \Gamma(X\langle i\rangle, \mu),\ WF(i,j-1) + \Gamma(\mu, Y\langle j\rangle)\ \}$$

Y la distancia de Levenshtein entre **X** e **Y** viene dada por $WF(|X|,|Y|)$, es decir, $DL(X,Y)=WF(|X|,|Y|)$.

Algoritmo de Wagner y Fischer para el cálculo de la distancia de Levenshtein:

```
Procedimiento CWF_DL (X,Y)
WF(0,0)=0
para i=1 hasta |X| hacer
    WF(i,0)=WF(i-1,0)+Γ(X<i>,μ)
fin para
para j=1 hasta |Y| hacer
    WF(0,j)=WF(0,j-1)+Γ(μ,Y<j>)
fin para
para i=1 hasta |X| hacer
    para j=1 hasta |Y| hacer
        m1=WF(i-1,j-1)+Γ(X<i>,Y<j>)
        m2=WF(i-1,j)+Γ(X<i>,μ)
        m3=WF(i,j-1)+Γ(μ,Y<j>)
        WF(i,j)=min(m1,m2,m3)
    fin para
fin para
```

Ejemplo:

Si *X=trabajo* e *Y=pasajero,* la matriz **WF** es la siguiente:

WF		p	a	s	a	j	e	r	o
	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
t 1	1	1	2	3	4	5	6	7	8
r 2	2	2	2	3	4	5	6	6	7
a 3	3	3	2	3	3	4	5	6	7
b 4	4	4	3	3	4	4	5	6	7
a 5	5	5	4	4	3	4	5	6	7
j 6	6	6	5	5	4	3	4	5	6
o 7	7	7	6	6	5	4	4	5	5

Por tanto: *DL(X,Y)=5*.

Análisis del algoritmo:

La cantidad total de tiempo usada por este algoritmo es proporcional al número de asignaciones ejecutadas (excluyendo las implícitas de los

bucles *para*). Este número es exactamente $1+|X|+|Y|+4*|X|*|Y|$, por tanto el tiempo total es $O(|X|*|Y|)$.♦

1.3. Cálculo Optimizado de la Distancia de Levenshtein.

El cálculo de **DL** fue optimizado por Ukkonen en [UK83], y es posteriormente utilizado por el propio Ukkonen en [UK85], Galil y Giancarlo en [GG86] y Landau y Vishkin en [LV85a], [LV85b], [LV86a] y [LV86b], así como en el presente trabajo.

Este cálculo se realiza utilizando las diagonales de la matriz **WF** de Wagner y Fischer siguiendo a Landau y Vishkin. Una diagonal **d** de la matriz consiste en todos los $WF(i,j)$ tales que $i-j=d$.

Dada una distancia **e** y una diagonal **d**, se define:

$$LV(d,e) = máximo\{i / WF(i,j)=e \text{ con } i-j=d\}$$

Esto implica que:

$$e=WF(LV(d,e),LV(d,e)-d) \text{ y } X<LV(d,e)+1> \neq Y<LV(d,e)-d+1>$$

Además si

$$|X| \leq |Y|, \; df = |X|-|Y| \text{ y } LV(df,e) \geq |X|$$

entonces

$$e=WF(|X|,|Y|).$$

A continuación se muestra como se puede calcular, dados **e** y **d**, el valor de $LV(d,e)$. Supóngase que $\forall v<e$ y $\forall g$ ya está calculado $LV(g,v)$.

Si $LV(d,e)=i$, (o sea, $WF(i,j)=e$ y $i-j=d$) es porque ocurre alguna de las siguientes posibilidades:

a) ($WF(i-1,j-1)=e-1$ y $X<i> \neq Y<j>$) ó $WF(i,j-1)=e-1$ ó $WF(i-1,j)=e-1$.

b) $WF(i-1,j-1)=e$ y $X<i>=Y<j>$.

Para obtener el valor de $LV(d,e)$ es necesario conocer $LV(d-1,e-1)$, $LV(d,e-1)$ y $LV(d+1,e-1)$. El cálculo de la matriz **LV** se hará de forma que la fila **df** sea la que primero se calcule, dentro de lo posible, ya que la condición para obtener **DL** es que un valor de $LV(df,e)$ supere la longitud de la cadena más corta $/X/$.

Algoritmo para el cálculo optimizado de la distancia de Levenshtein:

Supóngase que $/X/ \leq /Y/$

```
Procedimiento COP_DL (X,Y)
LV(0,-1)=-1
para d=1 hasta |X| hacer
    LV(d,d-2)=-1
    LV(d,d-1)=d-1
fin para
para d=1 hasta |Y| hacer
    LV(-d,d-2)=-1
    LV(-d,d-1)=-1
fin para
df=|X|-|Y|
para k=0 hasta |X|/2 hacer
    para td=-1 hasta 0 hacer
        e=k-df+td
        para d=df-k hasta df-1 hacer
            e=e+1
            fila=max{LV(d-1,e-1)+1,LV(d,e-1)+1,LV(d+1,e-1)}
            mientras (fila+1≤|X|) y (X<fila+1>=Y<fila-d+1>) hacer
                fila=fila+1
            fin mientras
            LV(d,e)=fila
        fin para
        e=k+td
        para d=k hasta df paso -1 hacer
            e=e+1
            fila=max{LV(d-1,e-1)+1,LV(d,e-1)+1,LV(d+1,e-1)}
            mientras (fila+1≤|X|) y (X<fila+1>=Y<fila-d+1>) hacer
                fila=fila+1
            fin mientras
            LV(d,e)=fila
        fin para
        si LV(df,e) ≥ |X| entonces
            WF(|X|,|Y|)=e
            retornar
        fin si
    fin para
fin para
```

Ejemplo:

Sea *X=trabajo* e *Y=pasajero* entonces se tiene que $|X|=7$, $|Y|=8$ y, por tanto, *df=-1*. A continuación se muestra de nuevo la matriz **WF**, correspondiente a estas dos cadenas, a la que se le han añadido los valores, **d**, de las diagonales.

WF		p a s a j e r o								
		0	1	2	3	4	5	6	7	8	d
t r a b a j o	0	0	1	2	3	4	5	6	7	8	
	1	1	1	2	3	4	5	6	7	8	-8
	2	2	2	2	3	4	5	6	6	7	-7
	3	3	3	2	3	3	4	5	6	7	-6
	4	4	4	3	3	4	4	5	6	7	-5
	5	5	5	4	4	3	4	5	6	7	-4
	6	6	6	5	5	4	3	4	5	6	-3
	7	7	7	6	6	5	4	4	5	5	-2
d		7	6	5	4	3	2	1	0	-1	

Se construye la matriz **LV**, asociada a **X** e **Y**, siguiendo el algoritmo anterior. Para facilitar su comprensión se presenta una traza del mismo.

LV	-1	0	1	2	3	4	5	6	7
-8								-1	-1
-7							-1	-1	
-6						-1	-1		
-5					-1	-1			
-4				-1	-1				
-3			-1	-1	0				
-2		-1	-1	0	1	3			
-1	-1	-1	0	1	3	4	7		
0	-1	0	1	2	3	6			
1	-1	0	1	3	6				
2		-1	1	2					
3			-1	2					
4				-1	3				
5					-1	4			
6						-1	5		
7							-1	6	

k	td	d	e
0	-1	0	0
		-1	1
	0	0	1
		-1	2
1	-1	-2	2
		1	1
		0	2
		-1	3
	0	-2	3
		1	2
		0	3
		-1	4
2	-1	-3	3
		-2	4
		2	2
		1	3
		0	4
		-1	5

Cuando se obtiene $LV(-1,5)=7 \geq |X|$ se detiene el proceso, siendo $WF(|X|,|Y|)=5$ y por tanto $DL(X,Y)=5$.

Análisis del algoritmo:

De la misma forma que en el algoritmo de Wagner y Fischer, se evalúa la cantidad total de tiempo usada realizando el análisis en el peor caso; ello ocurre cuando las dos cadenas no tienen ningún carácter en común:

El número de asignaciones correspondientes a la fase de inicialización es:

$$1+2*|X|+2*|Y|+1=2*|X|+2*|Y|+2 \qquad (1)$$

El número de iteraciones de los dos bucles *para*, en función de la variable **d**, viene dado por la siguiente expresión:

$$(|Y|-|X|+1)+(|Y|-|X|+1)+(|Y|-|X|+3)+$$
$$(|Y|-|X|+3)+...$$

donde cada sumando entre paréntesis con término independiente del mismo valor, es debido a los dos posibles valores de la variable **td**; además dentro de cada uno de los dos bucles se realizan 3 asignaciones y dentro del bucle controlado por **td**, 2 asignaciones de la variable **e**; el número de términos de la expresión en el peor caso es $2*\lfloor |X|/2+1 \rfloor$; por tanto se obtiene:

$$3*[(|Y|-|X|+1)+(|Y|-|X|+1)+(|Y|-|X|+3)+$$
$$+(|Y|-|X|+3)+\overset{2*\lfloor |X|/2+1 \rfloor \text{términos}}{\ldots\ldots\ldots}]+2*2*\lfloor |X|/2+1 \rfloor=$$
$$=3*(|Y|-|X|)*(2*\lfloor |X|/2+1 \rfloor)+$$
$$+3*(1+1+3+3+5+5+\overset{2*\lfloor |X|/2+1 \rfloor \text{términos}}{\ldots\ldots\ldots})+4*\lfloor |X|/2+1 \rfloor=$$
$$=3*2*[(|Y|-|X|)*\lfloor |X|/2+1 \rfloor+(1+3+5+\overset{\lfloor |X|/2+1 \rfloor \text{términos}}{\ldots\ldots\ldots})]+$$
$$+4*\lfloor |X|/2+1 \rfloor=$$
$$=6[(|Y|-|X|)*\lfloor |X|/2+1 \rfloor+\lfloor |X|/2+1 \rfloor^2]+4*\lfloor |X|/2+1 \rfloor$$

añadiendo a esta expresión la representada en (1):

$$2*|X|+2*|Y|+2+6[(|Y|-|X|)*\lfloor |X|/2+1 \rfloor+\lfloor |X|/2+1 \rfloor^2]+$$
$$+4*\lfloor |X|/2+1 \rfloor=$$
$$=6*|Y|*\lfloor |X|/2 \rfloor-6*|X|*\lfloor |X|/2 \rfloor+6*\lfloor |X|/2 \rfloor^2+$$

$$+8*|Y|-4*|X|+16*\lfloor |X|/2 \rfloor+12$$

si $|X|$ es par se obtiene:

$$3*|Y|*|X|-3/2*|X|^2+8*|Y|+4*|X|+12$$

si $|X|$ es impar se obtiene:

$$6*|Y|*(|X|/2-1/2)-6*|X|*(|X|/2-1/2)+6*(|X|/2-1/2)^2+$$
$$+8*|Y|-4*|X|+16*(|X|/2-1/2)+12=$$
$$=3*|Y|*|X|-3/2*|X|^2+5*|Y|+4*|X|+11/2$$

por tanto, en cualquier caso, el tiempo total es $O(|Y|*|X|-|X|^2/2)$.♦

1.4. Distancia Invariante Trasposicional, DIT.

El alto costo computacional de la distancia de Levenshtein, en contraste con sus prestaciones para encontrar cadenas similares, aconseja introducir una medida menos costosa que seleccione previamente las cadenas a las que se ha de calcular **DL**. La distancia invariante trasposicional entre dos cadenas **X** e **Y**, *DIT(X,Y)*, se define en función del número de veces que está presente cada carácter en ambas cadenas.

Definición: *Dadas dos cadenas* ***X*** *e* ***Y*** *se define*

$$DIT(X,Y) = \sum_{i=1}^{m} abs(X_{\alpha i} - Y_{\alpha i}) + abs(|X| - |Y|)$$

donde $\boldsymbol{X_{\alpha i}}$ *e* $\boldsymbol{Y_{\alpha i}}$ *son las frecuencias de aparición del carácter* $\boldsymbol{\alpha_i}$ *en* ***X*** *y en* ***Y*** *respectivamente, y* ***m*** *el tamaño del alfabeto.*

Como se desprende inmediatamente de esta definición, el valor de **DIT** entre dos cadenas de caracteres no varía si se altera el orden secuencial

de los mismos, por esta razón se ha denominado distancia invariante trasposicional.

Ejemplo:

Sea *X=trabajo* e *Y=pasajero*, se tiene que:

$$X_a=2,\ X_b=1,\ X_j=1,\ X_o=1,\ X_r=1,\ X_t=1$$

$$Y_a=2,\ Y_e=1,\ Y_j=1,\ Y_o=1,\ Y_p=1,\ Y_r=1,\ Y_s=1$$

por tanto:

$$DIT(X,Y)=5+1=6.$$

Teorema: *DIT es una semidistancia en el conjunto de cadenas de caracteres sobre un alfabeto dado.*

Demostración:

a) $\forall X, Y: Si\ X=Y \Rightarrow DIT(X,Y)=0$

Por la propia definición.

b) $\forall X, Y: DIT(X,Y)=DIT(Y,X)$

Por la propia definición.

c) $\forall X, Y, Z: DIT(X,Y)+DIT(Y,Z) \geq DIT(X,Z)$

Por la desigualdad triangular para el valor absoluto:

$$DIT(X,Y) + DIT(Y,Z) = \sum_{i=1}^{m} abs(X_{\alpha i} - Y_{\alpha i}) + abs(|X| - |Y|) +$$

$$+ \sum_{i=1}^{m} abs(Y_{\alpha i} - Z_{\alpha i}) + abs(|Y| - |Z|) =$$

$$= \sum_{i=1}^{m} [abs(X_{\alpha i} - Y_{\alpha i}) + abs(Y_{\alpha i} - Z_{\alpha i})] +$$

$$+ abs(|X| - |Y|) + abs(|Y| - |Z|) \geq$$

$$\geq \sum_{i=1}^{m} abs(X_{\alpha i} - Z_{\alpha i}) + abs(|X| - |Z|) = DIT(X,Z). \blacksquare$$

Ya que **DIT** es una semidistancia sobre el conjunto de cadenas, se tiene que es una distancia en el conjunto cociente dado por el conjunto de cadenas y la relación de equivalencia inducida; dicho conjunto cociente es el conjunto de cadenas de caracteres sobre un alfabeto prescindiendo del orden entre los mismos. Cada clase de equivalencia estará formada por todas aquellas cadenas que tengan los mismos caracteres aunque se encuentren en diferente orden; este tipo de cadenas se denomina *sinónimos_DIT*.

Análisis del cálculo de DIT:

Se realizará el análisis suponiendo que las frecuencias de aparición de los caracteres en cada uno de las cadenas se encuentran ya calculadas, porque así sucede en los esquemas de búsqueda que posteriormente se expondrán. La cantidad total de tiempo usada en el cálculo de **DIT** es proporcional al número de valores absolutos que se calculan. Dicho número

en el peor caso –ocurre cuando ambas cadenas **X** e **Y** no tienen caracteres comunes y además en cada cadena no hay caracteres repetidos– es $|X|+|Y|+1$; debe tenerse en cuenta que para aquellos **i** tales que $X_{\alpha i}=0$ e $Y_{\alpha i}=0$ no es necesario calcular el valor absoluto correspondiente –véanse los apartados del Capítulo 2 correspondientes a los esquemas de búsqueda–; por tanto la complejidad de cálculo es $O(|X|+|Y|)$.♦

Teorema: *Dadas dos cadenas cualesquiera **X** e **Y**, se verifica:*

$$DIT(X,Y) \leq 2*DL(X,Y).$$

Demostración:

Supóngase que dadas dos cadenas **X** e **Y** con $|X| \leq |Y|$, tienen **i** caracteres coincidentes –en cuanto a su frecuencia de aparición– independientemente de su posición, los restantes caracteres de **X**, $j=|X|-i$, son todos diferentes a los que restan en **Y** y sea $k=|Y|-|X|$. Se puede decir que existen **i** caracteres comunes, **j** que han sido sustituidos por otros y **k** que han sido añadidos a **X** (o eliminados de **Y**). Bajo estas condiciones se puede afirmar que:

$$DIT(X,Y) = 2*j+k + k = 2*(j+k) \qquad (1)$$

Nótese que **k** es como mínimo el número de *inserciones* sobre **X** (o de *extracciones* sobre **Y**) y que no todos los **i** caracteres coincidentes han de ser necesariamente considerados como tales en el cálculo de **DL** (por ejemplo una trasposición para **DIT** son caracteres coincidentes y para **DL** son dos sustituciones). Puede suponerse que los **i** caracteres comunes están situados al principio en ambas cadenas, a continuación los **j** caracteres en cada una y por último, en **Y**, los **k** restantes. Es decir: $X<1,i> = Y<1,i>$, los caracteres de $X<i+1,i+j>$ *y de* $Y<i+1,i+j>$ son los **j** distintos y los caracteres de $Y<i+j+1,i+j+k>$ son los **k** restantes en **Y**.

Considerando la *derivación_S* de **X** a **Y** utilizada al calcular **DL**, sean: **i'** el número de caracteres coincidentes, $i' \leq i$, **j'** el número de sustituciones y **k'** el número de inserciones más extracciones, $k' \geq k$. Y sea $\mathbf{j_1}$ el número de sustituciones, que **DL** admite como tales, de entre las **j** anteriormente citadas, $j' \geq j_1$.

Se pueden reordenar los caracteres de **X** e **Y**, sin pérdida de generalidad en cuanto a la demostración se refiere, partiendo de la ordenación considerada anteriormente, de tal forma que: $X<1,i'> = Y<1,i'>$ y que los caracteres de $X<i+1,i+j_1>$ y de $Y<i+1,i+j_1>$ forman las $\mathbf{j_1}$ sustituciones.

Las **j'** sustituciones se forman además a partir de:

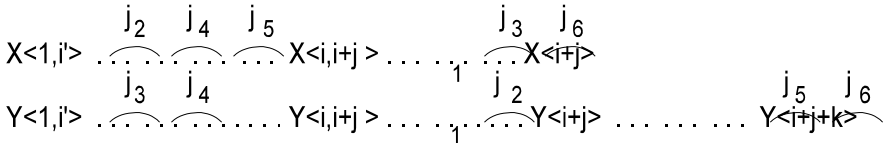
$\mathbf{j_2}$ caracteres de $X<i'+1,i>$ y de $Y<i+j_1+1,i+j>$

$\mathbf{j_3}$ caracteres de $X<i+j_1+1,i+j>$ y de $Y<i'+1,i>$

$\mathbf{j_4}$ caracteres de $X<i'+j_2+1,i>$ y de $Y<i'+j_3+1,i>$

$\mathbf{j_5}$ caracteres de $X<i'+j_2+j_4+1,i>$ y de $Y<i+j+1,i+j+k>$

$\mathbf{j_6}$ caracteres de $X<i+j_1+j_3+1,i+j>$ y de $Y<i+j+j_5+1,i+j+k>$.



entonces:

$$j' = j_1+j_2+j_3+j_4+j_5+j_6$$

y

$$k' = i-(i'+j_2+j_4+j_5)+i-(i'+j_3+j_4)+j-(j_1+j_3+j_6)+j-(j_1+j_2)+k-(j_5+j_6) =$$
$$= 2*i-2*i'+2*j-2*j_1-2*j_2-2*j_3-2*j_4-2*j_5-2*j_6+k = 2*(i-i')+2*(j-j')+k$$

Dado que:

$$DL(X,Y) = j'+k'$$

se tiene que:

$$DL(X,Y) = 2*(i-i')+2*j-j'+k$$

por tanto, teniendo en cuenta (1), se obtiene:

$$DL(X,Y) = DIT(X,Y)/2+2*(i-i')+j-j'$$

como

$$2*(i-i')+j-j' = 2*(i-i')+j-j_1-j_2-j_3-j_4-j_5-j_6$$

y puesto que

$$j-j_1-j_6 \geq 0$$

y

$$2*(i-i') \geq j_2+j_3+j_4+j_5$$

se cumple que

$$2*(i-i')+j-j' \geq 0$$

de ahí que

$$DIT(X,Y) \leq 2*DL(X,Y).\blacksquare$$

Capítulo 2.
Estructura de Santana y Díaz, S-D, y Esquemas de Búsqueda de las Más Similares

Se considera como diccionario, **D**, un determinado conjunto de diferentes cadenas de caracteres sobre cierto alfabeto. Con el fin de reducir el tiempo de respuesta frente a una búsqueda secuencial, el problema de la recuperación en el diccionario de las cadenas más similares a una cadena de búsqueda dada –pudiendo ésta pertenecer o no al mismo– requiere una estructura de almacenamiento del diccionario y un esquema de búsqueda eficiente. Las frecuencias de los caracteres de las palabras de igual longitud se organizan para permitir conjuntamente con el esquema apropiado el abandono de la exhaustividad en la búsqueda; ello conlleva el que sólo se trate una parte del diccionario.

2.1. Estructura de Santana y Díaz, S-D.

El diccionario se organiza como un árbol en el cual se estructuran las componentes que intervienen en el cálculo de **DIT**, como se muestra en la figura 1.

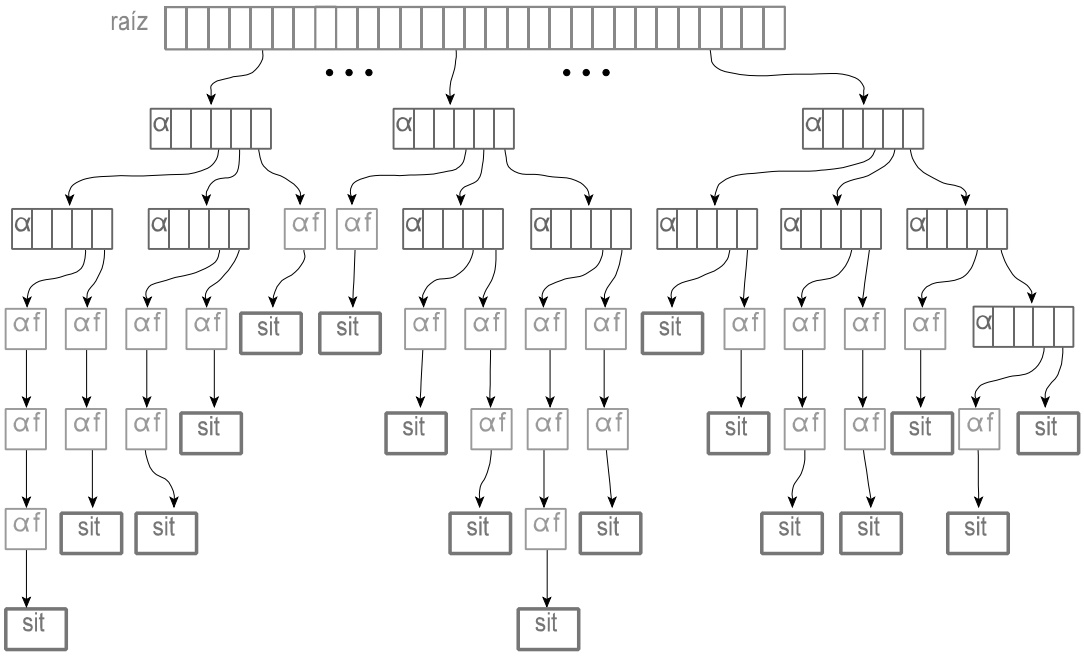


Figura 1

El árbol tiene un nodo *raíz* que discrimina por longitudes de cadenas. El encadenamiento asociado a cada longitud señala a la *parte_árbol* constituida por nodos discriminantes de la forma:

α	p	u	e_p	...	e_i	...	e_u

donde **α** es el carácter discriminante; $\mathbf{e_i}$ es el enlace que apunta al subconjunto de cadenas $D^i \subset D$ tal que la frecuencia del carácter **α** en cada una de las cadenas de $\mathbf{D^i}$ es **i**; **p** y **u** verifican:

$$0 \leq p < u$$

donde $\mathbf{e_p}$ y $\mathbf{e_u}$ son el primer y último encadenamiento no-nulo, aunque puede existir algún $i \in \{p+1,...,u-1\}$ tal que $e_i = nulo$.

Cuando una rama de la *parte_árbol* ya no discrimina se pasa a la *parte_cadena* que son listas encadenadas formadas por los restantes pares **αf**, cuyos nodos son:

α	f	e

donde **α** es un carácter, **f** es la frecuencia con que aparece el carácter **α** en los *sinónimos_DIT* correspondientes y **e** es un enlace que direcciona a un

nodo del mismo tipo o, al acabar la *parte_cadena*, a un *nodo_SIT* de la forma:

ns	C_1	...	C_i	...	C_{ns}

donde se encuentran las **ns** cadenas *sinónimas_DIT*, $\mathbf{C_1}$,..., $\mathbf{C_i}$,..., $\mathbf{C_{ns}}$.

Algoritmo de construcción de la estructura S-D:

```
Procedimiento Construye (raiz)
    raiz(.): contiene los punteros que señalan a la parte_árbol asociada a cada longitud
    SD(.): contiene los nombres de los subdiccionarios correspondientes a cada
    longitud
    longmin: mínima longitud de cadena en el diccionario
    longmax: máxima longitud de cadena en el diccionario
para long=longmin hasta longmax hacer
    raiz(long)=Construye_parte_árbol(SD(long),0)
fin para
```

```
Función Construye_parte_árbol (subdiccionario,nct)
    nodo: nodo de la parte_árbol, consta de los siguientes campos:
        cd: carácter discriminante
        pp: primera frecuencia asociada a un encadenamiemto no nulo
        pu: última frecuencia asociada a un encadenamiento no nulo
        e(.): contiene los enlaces a otros nodos
    nct: indica el número de caracteres tratados
si tamaño(subdiccionario) = 0 entonces
    devolver(nulo)
si no
    si tamaño(subdiccionario) = 1 entonces
        devolver(Construye_parte_cadena(subdiccionario,nct))
    si no
        cdisc=Elige_cdisc(subdiccionario,p,u)
        {la función Elige_cdisc selecciona el carácter discriminante en el conjunto de
        caracteres no tratados y devuelve p y u}
        si p = u entonces
            devolver(Construye_parte_cadena(subdiccionario,nct))
        si no
            {se divide el subdiccionario según la frecuencia del carácter discriminante}
            para j=1 hasta tamaño(subdiccionario) hacer
                f=frecuencia de cdisc en la j-ésima cadena de subdiccionario
                añadir la j-ésima cadena de subdiccionario a subdiccionario_f
            fin para
            crear un nodo de la parte_árbol, nodopa
            nodopa.cd=cdisc
            nodopa.pp=p
            nodopa.pu=u
            para i=p hasta u hacer
                nodopa.e(i)=Construye_parte_árbol(subdiccionario_i,nct+i)
            fin para
            devolver(nodopa)
        fin si
    fin si
fin si
```

```
Función Construye_parte_cadena (subdiccionario,nct)
    nodo: nodo de la parte_cadena, consta de los siguientes campos:
        c: carácter
        f: frecuencia
        e: enlace a otro nodo de la parte_cadena o a un nodo_SIT
si nct < long entonces
    crear un nodo de la parte_cadena, nodopc
    nodopc.c=carácter no tratado de las cadenas de subdiccionario
    nodopc.f=frecuencia de ese carácter
    nct=nct+nodopc.f
    pnodoc=nodopc
    mientras nct < long hacer
        crear un nodo de la parte_cadena, nodo
        nodo.c=carácter no tratado de las cadenas de subdiccionario
        nodo.f=frecuencia de ese carácter
        nct=nct+nodo.f
        nodopc.e=nodo
        nodopc=nodo
    fin mientras
    nodopc.e=Construye_nodo_SIT(subdiccionario)
    devolver(pnodoc)
si no
    devolver(Construye_nodo_SIT(subdiccionario))
fin si


Función Construye_nodo_SIT (subdiccionario)
    nodo: nodo_SIT, consta de los siguientes campos:
        ns: número de sinónimos_DIT
        s(.): cadenas sinónimas_DIT
crear un nodo_SIT, nodo
nodo.ns=tamaño(subdiccionario)
para i=1 hasta nodo.ns hacer
    nodo.s(i)=i-ésima cadena de subdiccionario
fin para
devolver (nodo)
```

2.1.1. Criterios de Selección del Carácter Discriminante.

Una primera opción para la elección del carácter discriminante es la que establece el siguiente criterio dinámico: se va creando el árbol según se van insertando las cadenas; cuando es necesario crear un nodo de la *parte_árbol* que separe los caminos de dos palabras se escoge, por ejemplo, el primer carácter por orden alfabético que establezca una diferencia.

Debido a la naturaleza estática del diccionario, se puede pensar en crear la estructura conociendo a priori qué palabras va a contener. De esta forma se puede escoger en todo momento el carácter que más eficazmente realice la tarea discriminatoria. Se elige el carácter que minimice la suma de los cuadrados de los cardinales de los subdiccionarios que genera. Se ha seguido este criterio ya que selecciona caracteres que reparten

uniformemente el número de cadenas en cada nodo incidiendo en la reducción de la longitud del camino promedio del árbol. Para una cadena del diccionario, el promedio del número de cadenas entre las que hay que realizar la búsqueda es mínimo desde cada nodo discriminante de la *parte_árbol*. En efecto, dado un nodo discriminante de la *parte_árbol*, sean $\mathbf{c_p}$,..., $\mathbf{c_u}$ el número de cadenas bajo los encadenamiento $\mathbf{e_p}$,..., $\mathbf{e_u}$ respectivamente, $c=c_p+...+c_u$, y sea **X** una de las cadenas del subdiccionario que penden del nodo. El valor esperado del número de cadenas entre las que hay que realizar la búsqueda de **X** es:

$$\frac{c_p}{c} * c_p + \ldots + \frac{c_u}{c} * c_u = \sum_{i \in [p..u]} \frac{c_i^2}{c}$$

Dado que el numerador es mínimo, debido al criterio de selección del carácter discriminante, es mínimo el valor esperado.

Ejemplo de la estructura S-D:

En la figura 2 se muestra un ejemplo de la estructura **S-D** para longitud cuatro, correspondiente al subdiccionario formado por: *amor*, *bebe*, *coro*, *diva*, *loro*, *meta*, *mora*, *ramo*, *raza*, *roma*, *tiro*, *toro* y *vaya*. Los caracteres de los nodos discriminantes se han seleccionado según el criterio de *mínimos cuadrados* descrito anteriormente.

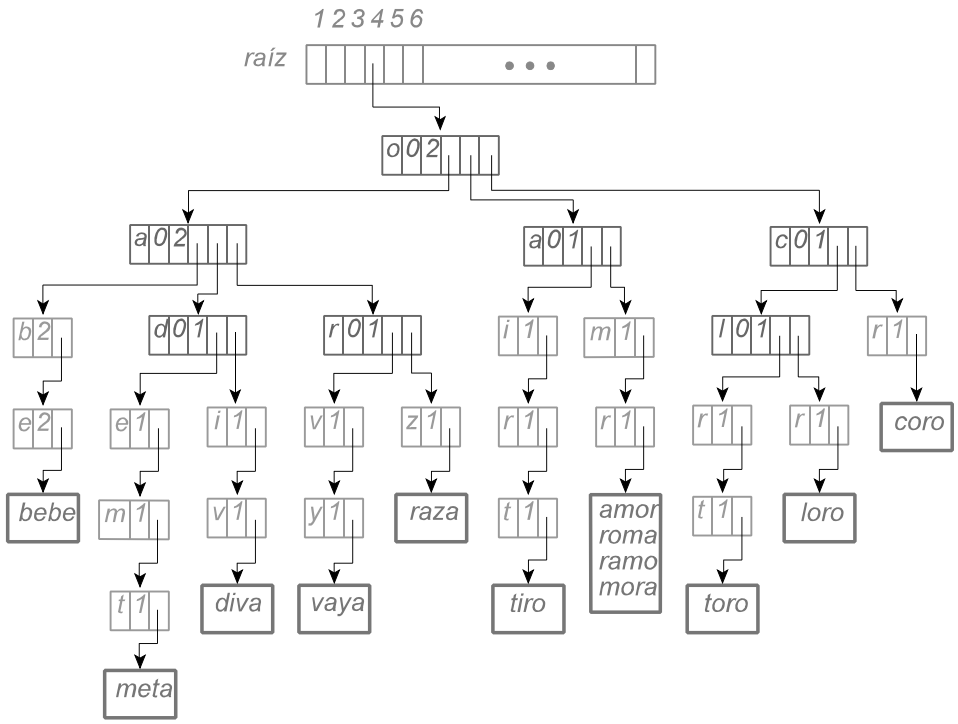


Figura 2

2.1.2. Relación Ocupacional.

Las cadenas situadas en los *nodos_SIT* se pueden considerar como los datos, se entiende que el resto de la estructura **S-D** es el índice.

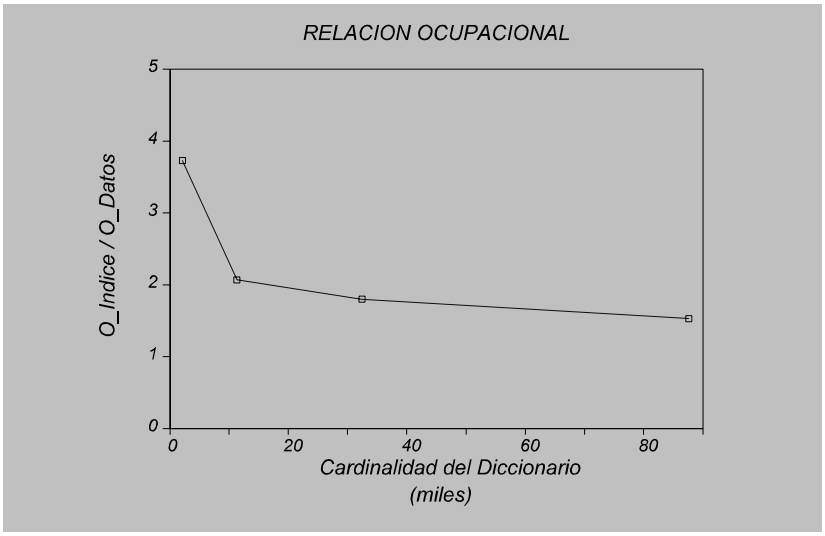


Figura 3

La razón *O_Índice/O_Datos* es el cociente entre la ocupación del índice y la ocupación de los datos expresada en bytes.

Se realizó un experimento, utilizando las palabras de diferentes diccionarios de la lengua española –sin considerar los signos ortográficos– para estudiar la relación entre la razón *O_Índice/O_Datos* y el tamaño del diccionario. Para ello se tiene que **α**, **p**, **u**, **f** y **ns** ocupan un byte cada uno. $\mathbf{C_j}$ usa $/C_j/$ bytes y cada uno de los enlaces de la estructura rinde cuatro bytes.

Como puede observarse, en la figura 3, la relación ocupacional cae rápidamente para cardinalidades bajas, para aminorar su descenso, manteniéndose a valores razonables, a medida que aumenta la cardinalidad del diccionario.

2.2. Búsqueda de las Cadenas Más Similares.

Definición: *Sea **D** el conjunto de todas las cadenas correctas, es decir, el diccionario, de cardinalidad **NCD**. Sea **B** la cadena de búsqueda, **d** una distancia definida sobre **D** y **SB** un subconjunto de **D** definido como sigue:*

$$SB = \{ X \in D \ / \ d(B,X) = min\{d(B,Y), \forall Y \in D\} \}$$

***SB** se denomina conjunto de cadenas más similares a **B**.*

En este trabajo se estudia el problema de la búsqueda de las cadenas más similares a una dada utilizando la distancia de Levenshtein, **DL**, como medida de similitud. Esta distancia requiere un alto costo computacional, una forma de reducirlo consiste en utilizar **DIT** como filtro de **DL** debido a su menor costo y a la relación existente entre ambas, $DIT(X,Y) \leq 2*DL(X,Y)$.

Las cadenas más similares a una cadena de búsqueda **B** según **DIT**, no son necesariamente las **DL** más similares. Por ejemplo, si **X** difiere de **B** en una trasposición e **Y** difiere de **B** en una inserción se tiene que:

$$DIT(B,X)=0 < DIT(B,Y)=1$$

y

$$DL(B,X)=2 > DL(B,Y)=1$$

por tanto, **X** es más similar a **B** que **Y** según **DIT** pero no según **DL**.

La estructura **S-D** permite evaluar **DIT** considerando únicamente las frecuencias no nulas. Además, durante el proceso de búsqueda, en el retorno

a cada nodo de la *parte_árbol*, se dispone del valor acumulado de las componentes de **DIT** de los caracteres situados en el camino desde la raíz hasta dicho nodo.

En un momento del proceso de búsqueda, sea **X** una de las cadenas actuales más similares a **B** e **Y** una cadena del diccionario de la que se quiere estudiar su similitud con **B. Y** será considerará como más similar si:

$$DL(B,Y) \leq DL(B,X).$$

Antes de evaluar *DL(B,Y)*, se puede calcular *DIT(B,Y)* para aplicarla como filtro. Si se verifica que

$$DIT(B,Y) > 2*DL(B,X)$$

puesto que

$$DIT(B,Y) \leq 2*DL(B,Y)$$

entonces

$$DL(B,X) < DL(B,Y)$$

y se rechaza **Y** sin necesidad de evaluar **DL**. De aquí que los esquemas de búsqueda se realizan sobre la estructura **S-D** evaluando **DIT** en su recorrido y **DL** sólo en los *nodos_SIT* que permite el filtro.

2.2.1. Esquema de Búsqueda Decreciente.

Se dispone de un umbral, **DTM**, que en todo momento es igual al doble de la distancia de Levenshtein mínima actual, **DLM**, de tal forma que sólo se calcula la **DL** entre la cadena de búsqueda y aquella cadena del diccionario cuya $DIT \leq DTM$, debido a la relación existente entre **DIT** y **DL**. A lo largo del proceso, tanto el radio de búsqueda, **DLM**, como la respuesta se actualizan cada vez que se obtiene una $DL<DLM$; si $DL=DLM$ se añade a la respuesta la cadena correspondiente. El valor inicial de **DLM** es infinito.

La búsqueda comienza en el nodo *raíz* por la posición correspondiente a la longitud de la cadena de búsqueda, y se van tomando alternativamente las longitudes próximas, situadas a diferencias crecientes de dicha longitud, hasta alcanzar la primera diferencia de longitudes que supere a **DLM**. Basta con explorar estos ramales ya que la diferencia de longitudes indica el número mínimo de inserciones (o de extracciones) y es por tanto una cota inferior de **DL**. En la *parte_árbol* se comienza por el ramal del nodo actual correspondiente a la frecuencia de aparición del carácter en la cadena de búsqueda, y se continúa alternativamente por las

frecuencias próximas correspondientes a diferencias crecientes de la frecuencia inicial. El tratamiento de la *parte_cadena* es secuencial. Durante el recorrido por la estructura se evalúa una aproximación progresiva de **DIT**, acumulando los valores de sus componentes. Si, en algún punto de la misma, este valor excede a **DTM** no se prosigue por ese ramal, estableciéndose así un criterio de poda. Para acceder a un *nodo_SIT* se ha de cumplir que $DIT \leq DTM$, en cuyo caso se evalúa **DL** entre la cadena de búsqueda y todas las de ese nodo.

Algoritmo de búsqueda decreciente:

Se invoca desde el programa principal como Decreciente(B,∞,0)

```
Procedimiento Decreciente (B,DLM,nms)
    B: cadena de búsqueda
    DLM: valor de DL mínimo
    nms: número de cadenas más similares
    primero: primer carácter del alfabeto
    ultimo: último carácter del alfabeto
    vf(.): vector de frecuencias de B
    DTM: es el doble de DLM
    cadit: componentes acumuladas de DIT
    lalt: longitud alternativa
    longmax: máxima longitud de cadena en el diccionario
    longmin: mínima longitud de cadena en el diccionario
para c=primero hasta ultimo hacer
    vf(c)=0
fin para
para i=1 hasta |B| hacer
    vf(B<i>)=vf(B<i>)+1
fin para
DTM=∞
cadit=0
Busquedadec_pa(raiz(|B|),cadit,|B|)
i=1
mientras i ≤ DLM hacer
    cadit=i
    lalt=|B|+i
    si lalt ≤ longmax entonces
        Busquedadec_pa(raiz(lalt),cadit,|B|)
    fin si
    lalt=|B|-i
    si lalt ≥ longmin entonces
        Busquedadec_pa(raiz(lalt),cadit,|B|)
    fin si
    i=i+1
fin mientras
```

```
Procedimiento Busquedadec_pa (nodo,cadit,cbnt)
{Realiza el recorrido de la parte_árbol en la búsqueda decreciente}
    nodo: nodo actual
    cbnt: número de caracteres de B no tratados
    frec: frecuencia en B del carácter asociado a nodo
    emp: encadenamiento más próximo a frec
    vcadit: valor de las componentes acumuladas de DIT para la alternativa actual
    falt: frecuencia alternativa
si nodo ≠ nulo entonces
    si nodo es un nodo_SIT entonces
        cadit=cadit+cbnt
        si cadit ≤ DTM entonces
            Tratardec_nS(nodo)
        fin si
    si no
        si es un nodo de la parte cadena entonces
            Tratardec_pc(nodo,cadit,cbnt)
        si no  {es un nodo de la parte_árbol}
            frec=vf(nodo.cd)
            cbnt=cbnt-frec
            si (nodo.pp ≤ frec) y (frec ≤ nodo.pu) entonces
                Busquedadec_pa(nodo.e(frec),cadit,cbnt)
            si no
                si frec < nodo.pp entonces
                    emp=nodo.pp
                si no
                    emp=nodo.pu
                fin si
                vcadit=cadit+abs(frec-emp)
                Busquedadec_pa(nodo.e(emp),vcadit,cbnt)
                frec=emp
            fin si
            i=1
            vcadit=cadit+i
            mientras vcadit ≤ DTM hacer
                falt=frec+i
                si falt ≤ nodo.pu entonces
                    Busquedadec_pa(nodo.e(falt),vcadit,cbnt)
                fin si
                falt=frec-i
                si falt ≥ nodo.pp entonces
                    Busquedadec_pa(nodo.e(falt),vcadit,cbnt)
                fin si
                i=i+1
                vcadit=cadit+i
            fin mientras
        fin si
    fin si
fin si


Procedimiento Tratardec_nS (nodo)
{Realiza el tratamiento de los nodos_SIT en la búsqueda decreciente}
    dl: valor de DL entre B y el sinónimo_DIT actual
para j=1 hasta nodo.ns hacer
    dl=COP_DL(B,nodo.s(j))
    si dl < DLM entonces
        DLM=dl
```

```
        DTM=2*DLM
        nms=1
        eliminar el conjunto respuesta
        crear un conjunto respuesta formado por nodo.s(j)
        si dl = 0 entonces
            retorno de final de búsqueda
        fin si
    si no
        si dl = DLM entonces
            añadir nodo.s(j) al conjunto respuesta
            nms=nms+1
        fin si
    fin si
fin para


Procedimiento Tratardec_pc (nodo,cadit,cbnt)
{Realiza el recorrido de la parte_cadena en la búsqueda decreciente}
si nodo es un nodo_SIT entonces
    cadit=cadit+cbnt
    si cadit ≤ DTM entonces
        Tratardec_nS(nodo)
    fin si
si no
    cadit=cadit+abs(nodo.f-vf(nodo.c))
    cbnt=cbnt-vf(nodo.c)
    si cadit ≤ DTM entonces
        Tratardec_pc(nodo.e,cadit,cbnt)
    fin si
fin si
```

2.2.2. Familia de Esquemas.

La diferencia entre el valor inicial y final de **DLM** permite la aceptación de cadenas como similares, muy distantes de la cadena de búsqueda, que no van a estar presentes en la respuesta, prolongando el tiempo de obtención de la misma. ¿Es posible mejorar la estrategia de búsqueda en función del valor inicial de **DLM**?. El esquema *decreciente* busca en un dominio *{0, 1, 2,..., +∞}*, decreciendo dinámicamente el valor del radio. Una alternativa a esta estrategia sería realizar el mismo esquema de búsqueda pero en un dominio más pequeño. Dado que el valor final de **DLM** estará próximo a cero, se realiza una búsqueda inicial de radio cero y, mientras no se obtenga respuesta, se llevan a cabo sucesivas búsquedas para los consecutivos dominios a partir de cero con un tamaño de dominio previamente fijado. Esto da lugar a una *familia de esquemas* de búsqueda en función del tamaño del dominio, o lo que es lo mismo, del incremento del radio de búsqueda, **IR**.

En cada dominio, se trata de encontrar las cadenas $\mathbf{X_i}$ cuyas $DL(B,X_i)=\tau$, siendo τ el valor más bajo alcanzable que cumple que:

$$\delta < \tau \leq \delta + IR$$

siempre que no existan cadenas $\mathbf{X_j}$ tales que $DL(B,X_j) \leq \delta$; se utiliza un esquema de búsqueda *decreciente* en el que **DLM** se inicializa a $\delta+IR$. Si existe respuesta termina el proceso, en caso contrario se actualiza δ:

$$\delta = \delta + IR$$

y se recomienza la búsqueda desde el nodo *raíz*. Inicialmente $\delta=0$. Evidentemente el incremento del radio de búsqueda permanece constante en cada experiencia.

El esquema *decreciente* se corresponde con el esquema de la *familia* para el cual $IR=\infty$. Si $IR=1$ se obtiene un esquema en el que el radio de búsqueda crece de uno en uno invirtiendo el proceso de actualización del radio del esquema *decreciente*.

Algoritmo de búsqueda de la familia de esquemas:

```
Procedimiento Familia (B,IR)
    B: cadena de búsqueda
    IR: incremento del radio
    nms: cardinalidad de la respuesta
    DLM: valor de DL mínimo
delta=0
nms=0
Decreciente(B,0,nms)
si nms=0 entonces
    repetir
        DLM=delta+IR
        Decreciente(B,DLM,nms)
        delta=delta+IR
    hasta que nms > 0
fin si
```

2.2.3. Esquema de Búsqueda Creciente.

En este esquema se recorre la estructura partiendo del nodo *raíz* y con radio de búsqueda $DLM=0$; si no se encuentra respuesta, se aumenta en una unidad el radio de búsqueda y se comienza la exploración nuevamente desde el nodo *raíz*, repitiendo este proceso hasta que al finalizar un recorrido exista respuesta. En tal caso, se obtiene el conjunto de cadenas más similares y su valor de distancia es **DLM**. Debido a que el radio de

búsqueda en cada fase o es menor o es igual que la **DL** mínima, es por lo que el umbral **DTM** permanece constante en dicha fase.

La forma de recorrer el árbol es análoga al esquema de la *familia* con *IR=1*, pero la selección de las alternativas en los nodos discriminantes es diferente. En este último esquema se seleccionaban en vaivén a diferencias crecientes de la posición inicial, para lograr aproximaciones a cadenas más similares que posibiliten un acortamiento de su rango de exploración. En el esquema *creciente* este rango no se altera en cada fase, porque tampoco lo hace **DTM**, lo que permite una exploración lineal de izquierda a derecha de los subconjuntos que penden del nodo.

Algoritmo de búsqueda creciente:

```
Procedimiento Creciente (B)
    B: cadena de búsqueda
    primero: primer carácter del alfabeto
    ultimo: último carácter del alfabeto
    vf(.): vector de frecuencias de B
    DLM: valor de DL mínimo
    DTM: doble de DLM
    nms: número de cadenas más similares
    linf, lsup: longitud mínima y máxima alcanzadas en la selección de alternativas
    cadit: componentes acumuladas de DIT
    lalt: longitud alternativa
    longmin: mínima longitud de cadena en el diccionario
    longmax: máxima longitud de cadena en el diccionario
para c=primero hasta ultimo hacer
    vf(c)=0
fin para
para i=1 hasta │B│ hacer
    vf(B<i>)=vf(B<i>)+1
fin para
linf=│B│
lsup=│B│
DLM=0
DTM=0
nms=0
cadit=0
Busquedacre_p0(raiz(│B│))
mientras nms = 0 hacer
    DLM=DLM+1
    DTM=2*DLM
    linf=linf-1
    lsup=lsup+1
    si linf < longmin entonces linf=longmin fin si
    si lsup > longmax entonces lsup=longmax fin si
    para lalt=linf hasta lsup hacer
        cadit=abs(lalt-│B│)
        Busquedacre_pa(raiz(lalt),cadit,│B│)
    fin para
fin mientras
```

```
Procedimiento Busquedacre_p0 (nodo)
{Comprueba si la cadena de búsqueda se encuentra en el diccionario}
    nodo: nodo actual
    dl: valor de DL entre B y el sinónimo_DIT actual
    nms: número de cadenas más similares
    frec: frecuencia en B del carácter asociado a nodo
si nodo ≠ nulo entonces
    si nodo es un nodo_SIT entonces
        j=1
        repetir
            dl=COP_DL(B,nodo.s(j))
            j=j+1
        hasta que (dl = 0) o (j > nodo.ns)
        si dl = 0 entonces
            crear un conjunto respuesta formado por nodo.s(j)
            nms=1
        fin si
    si no
        si es un nodo de la parte cadena entonces
            si vf(nodo.c) = nodo.f entonces
                Busquedacre_p0(nodo.e)
            fin si
        si no  {es un nodo de la parte_árbol}
            frec=vf(nodo.cd)
            si (nodo.pp ≤ frec) y (frec ≤ nodo.pu) entonces
                Busquedacre_p0(nodo.e(frec))
            fin si
        fin si
    fin si
fin si
```

```
Procedimiento Busquedacre_pa (nodo,cadit,cbnt)
{Realiza el recorrido de la parte_árbol en la búsqueda creciente}
    nodo: nodo actual
    cbnt: número de caracteres de B no tratados
    frec: frecuencia en B del carácter asociado a nodo
    dfm: diferencia de frecuencias máxima
    finf: frecuencia mínima alcanzada en la selección de alternativas
    fsup: frecuencia máxima alcanzada en la selección de alternativas
    falt: frecuencia alternativa
    vcadit: valor de las componentes de DIT para la alternativa actual
si nodo ≠ nulo entonces
    si nodo es un nodo_SIT entonces
        cadit=cadit+cbnt
        si cadit ≤ DTM entonces
            Tratarcre_nS(nodo)
        fin si
    si no
        si es un nodo de la parte cadena entonces
            Tratarcre_pc(nodo,cadit,cbnt)
        si no  {es un nodo de la parte_árbol}
            frec=vf(nodo.cd)
            cbnt=cbnt-frec
            dfm=DTM-cadit
            finf=frec-dfm
            fsup=frec+dfm
            si finf < nodo.pp entonces finf=nodo.pp fin si
            si fsup > nodo.pu entonces fsup=nodo.pu fin si
            para falt=finf hasta fsup hacer
                vcadit=cadit+abs(frec-falt)
                Busquedacre_pa(nodo.e(falt),vcadit,cbnt)
            fin para
        fin si
    fin si
fin si


Procedimiento Tratarcre_nS (nodo)
{Realiza el tratamiento de los nodos_SIT en la búsqueda creciente}
    dl: valor de DL entre B y el sinónimo_DIT actual
para j=1 hasta nodo.ns hacer
    dl=COP_DL(B,nodo.s(j))
    si dl = DLM entonces
        añadir nodo.s(j) al conjunto respuesta
        nms=nms+1
    fin si
fin para
```

```
Procedimiento Tratarcre_pc (nodo,cadit,cbnt)
{Realiza el recorrido de la parte_cadena en la búsqueda creciente}
si nodo es un nodo_SIT entonces
    cadit=cadit+cbnt
    si cadit ≤ DTM entonces
        Tratarcre_nS(nodo)
    fin si
si no
    cadit=cadit+abs(nodo.f-vf(nodo.c))
    cbnt=cbnt-vf(nodo.c)
    si cadit ≤ DTM entonces
        Tratarcre_pc(nodo.e,cadit,cbnt)
    fin si
fin si
```

2.2.4. Resultados Experimentales.

Los resultados experimentales se llevaron a cabo en lenguaje C sobre un **HP9000-835**, utilizando un diccionario de 89.655 palabras de la lengua española —sin signos ortográficos, la distribución de frecuencias por longitudes se muestra en la figura 4. La estructura **S-D** construida con este diccionario consta de: 63.994 nodos en la *parte_árbol*, 136.232 en la *parte_cadena* y 77.717 *nodos_SIT*. Se construyen ficheros –de cardinalidad 50– que serán utilizados como argumento de búsqueda de las más similares, para ello se selecciona al azar una palabra del diccionario y se transforma aplicándole diversas operaciones de edición –elegidas aleatoriamente– hasta obtener una palabra distorsionada, con una **DL** determinada respecto a la correcta. **DL** toma los valores comprendidos entre uno y la mitad de la longitud de la palabra correcta. La longitud de la palabra original, la de la distorsionada y el valor de **DL** indican el fichero al que se va a asignar.

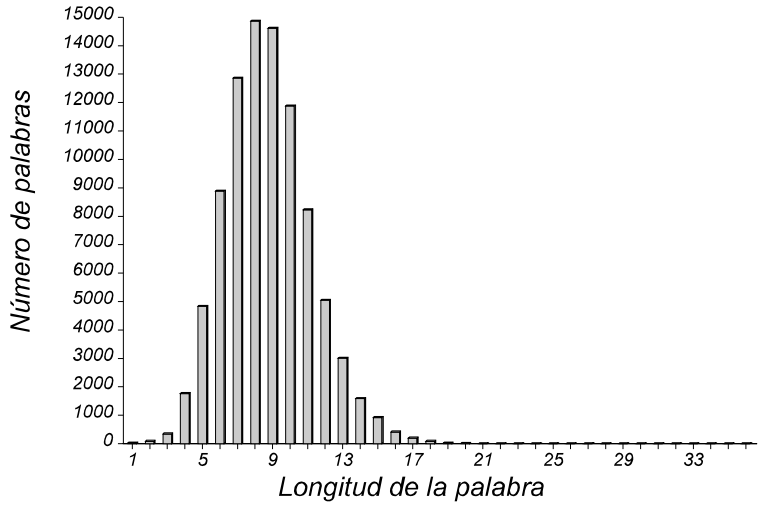


Figura 4

A modo de ejemplo, se muestran en la tabla 1 las palabras más similares a *desmxtadt* que se ha obtenido a partir de *desmayado* aplicándole una *DL=3*.

longitud=8	longitud=9	longitud=10
desmotar	desmolado	desmontado
desatado	desmayado	desmotador
desmatar	desmanado	desmontada
desatada	desmolada	
	desmayada	
	desmanada	

Tabla 1

Se ha realizado el estudio experimental con el fin de comparar los tiempos de búsqueda de la *familia de esquemas* y de los esquemas *decreciente* y *creciente*. Se toma como valor representativo el promedio de los tiempos de búsqueda para cada uno de los ficheros mencionados, teniendo en cuenta que la variabilidad en cada uno de ellos es poco significativa.

Se eligen, como muestra para las representaciones, algunos valores significativos de **DL** ya que, aún siendo éste el parámetro de mayor influencia cuantitativa, el comportamiento relativo de los esquemas es análogo a los que se presentan. Las figuras 5 y 6 muestran la representación tridimensional –tiempo promedio frente a las longitudes de la palabra original y la distorsionada– de los esquemas *decreciente* y *creciente* para *DL=2*.

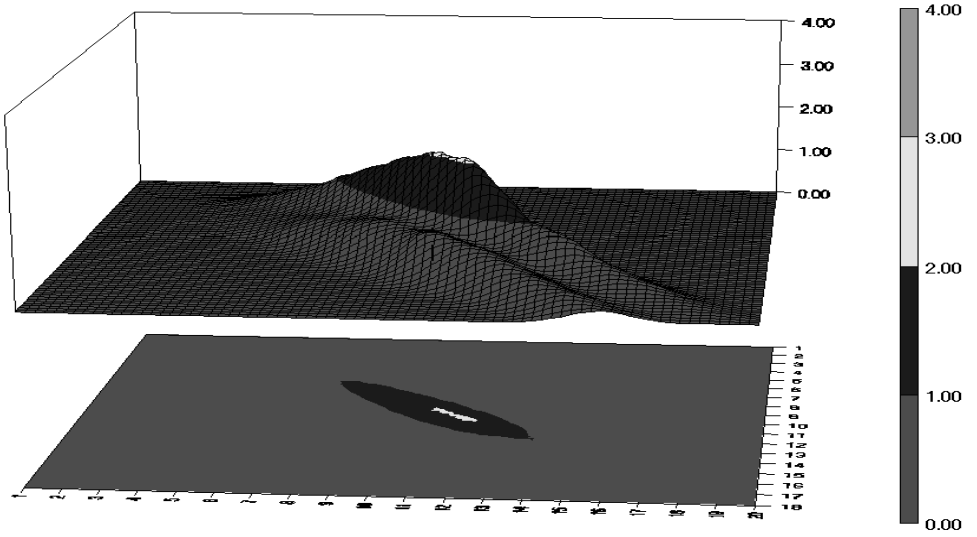


Figura 5

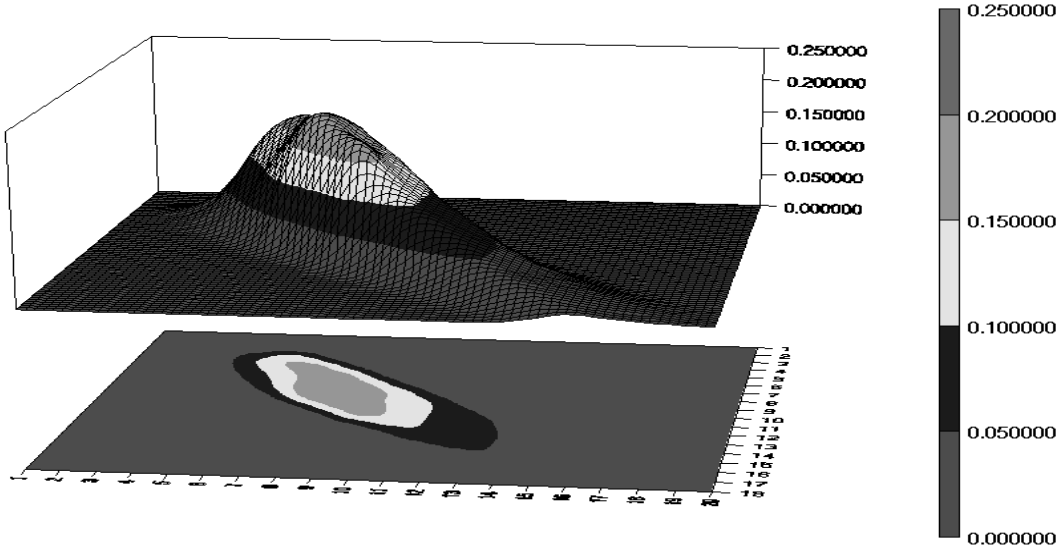


Figura 6

☞ En las figuras 5 y 6, el tiempo de respuesta está fuertemente influenciado por la distribución de longitudes en el diccionario, así los máximos valores se alcanzan para las longitudes intermedias. El esquema *decreciente* presenta un valle central, figura 5, que coincide con la igualdad de longitudes entre la palabra correcta y la distorsionada, debido a que reduce más rapidamente el radio de búsqueda cuando explora

en primer lugar el ramal donde se encuentra la palabra correcta.

Debido a las limitaciones que presenta el papel para las representaciones tridimensionales en adelante se harán representaciones bidimensionales. Existen dos razones para seleccionar la longitud de la cadena de búsqueda como variable independiente en las representaciones bidimensionales: en primer lugar, la menor influencia en los tiempos de respuesta de la longitud de la palabra correcta y por otro lado, la conveniencia de referenciar los resultados a partir de la palabra distorsionada por ser ésta la conocida por el usuario.

En las representaciones bidimensionales posteriores, se utiliza la poligonal que une los puntos medios resultantes para cada valor de la longitud de la cadena de búsqueda, para un determinado valor de **DL**.

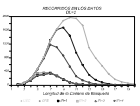


Figura 8

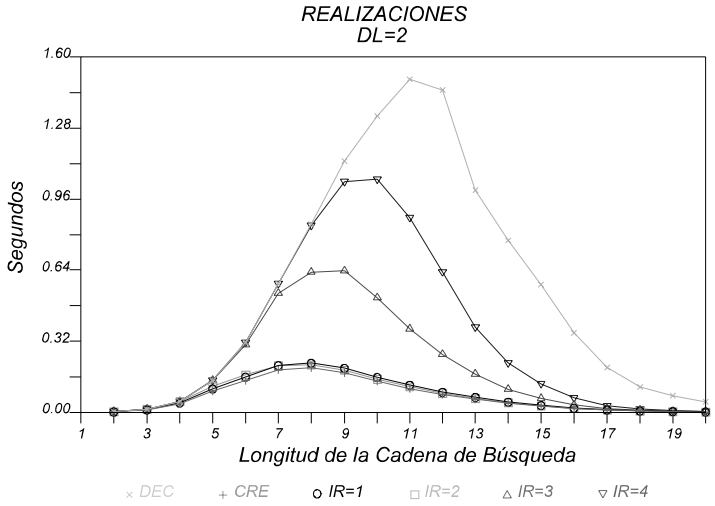


Figura 7

☞ Para *DL=2*, a medida que se incrementa el valor de **IR** crece el tiempo de búsqueda, figura 7, que se emplea en el cálculo de **DL**, figura 8, y en las componentes que intervienen en el cálculo de **DIT**, **C_DIT**, figura 9.

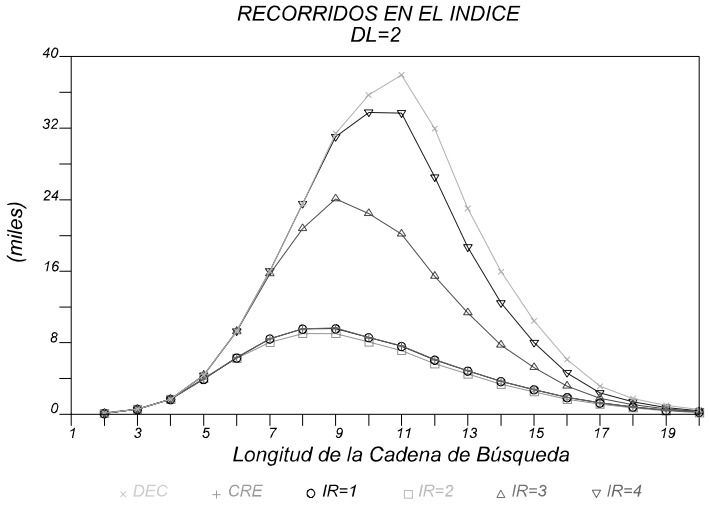


Figura 9

☞ Para *DL=4* se observa una alternancia en los resultados que se muestran en las figuras 10, 11 y 12. Cada valor de **IR** determina una secuencia de valores del radio inicial que se van aproximando a la **DL** mínima. El tiempo de búsqueda puede ser mejor para valores altos de **IR** si con ellos se alcanza la respuesta en un menor número de fases. Para longitudes de la cadena de búsqueda menores o iguales que ocho, los resultados de tiempos son prácticamente coincidentes debido a que se recuperan palabras con valores de *DL<4*, lo que da lugar a que no se realice el número esperado de fases para cada valor de **IR**. Para longitudes mayores los resultados aparecen diferenciados siguiendo la alternancia, puesto que la **DL** con las palabras recuperadas es en general igual a cuatro.

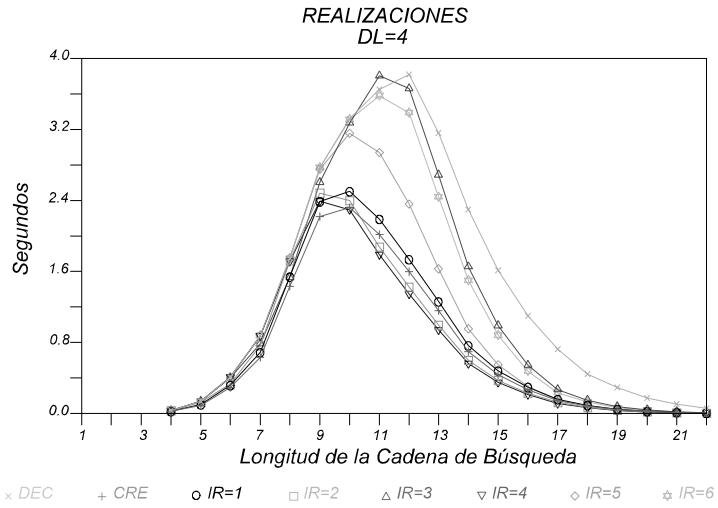


Figura 10

☞ Se muestra, en función del valor de **IR**, toda una gama de resultados que conectan de una manera no abrupta los esquemas *decreciente* y *creciente*, figuras 7-12. Puede observarse que, para los mayores valores de **IR**, las gráficas que representan a las **C_DIT** y al número de **DL** evaluadas, así como las de tiempos se acercan cada vez más a las correspondientes del esquema *decreciente*.

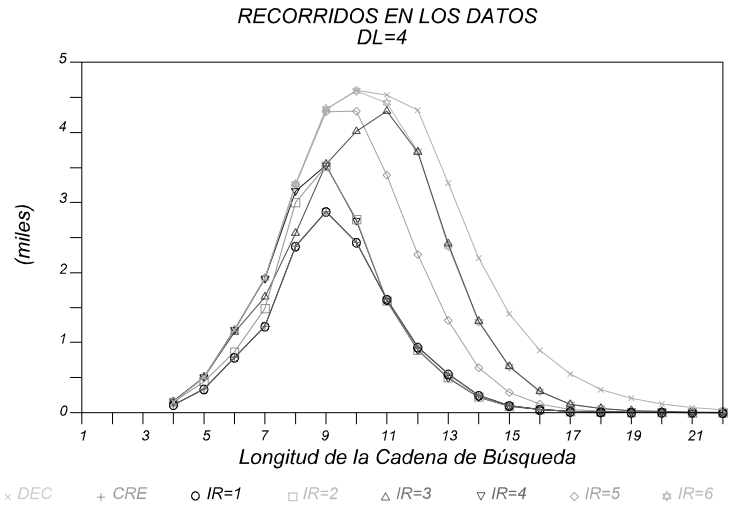


Figura 11

☞ El acercamiento máximo de la *familia de esquemas* al *creciente* se obtiene para *IR=1*, en el que se llevan a cabo exactamente el mismo número de **C_DIT** y **DL**, figuras 8, 9 y 11, 12, pero cuya realización, figuras 7 y 10, es ligeramente peor que la del *creciente* debido a la menor

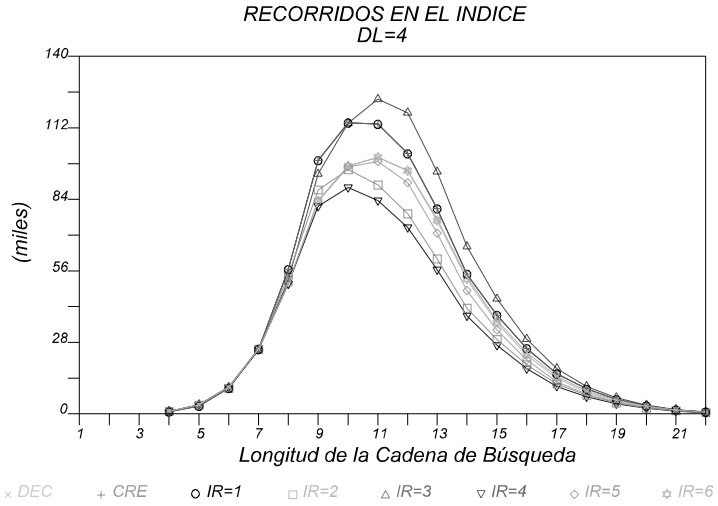


Figura 12

eficacia en la toma de alternativas.

☞ La mejor realización del *creciente* respecto al *decreciente* proviene del acentuado descenso en el número de **DL** evaluadas, para todas las longitudes estudiadas, (en las figuras 8 y 11 la gráfica del *creciente* aparece solapada con *IR=1*) incluso a pesar del notable aumento en las **C_DIT** para *DL=4* (en la figura 12 la gráfica del decreciente está solapada con *IR=6*) no le hace perder su atractivo.

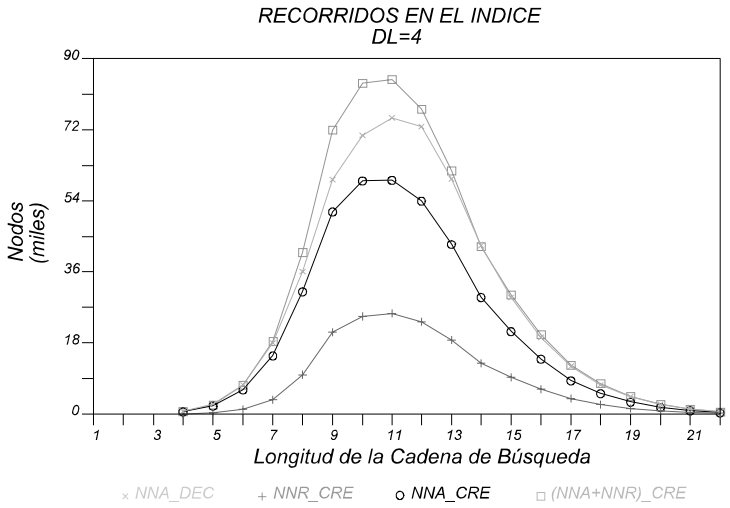


Figura 13

☞ En la figura 13 se ha representado el número de nodos afectados, **NNA**, –nodos del índice que se han explorado alguna vez– para ambos esquemas, observándose que, en todo momento, es inferior para el *creciente*. ¿Cómo es entonces posible el comportamiento expuesto anteriormente acerca de las **C_DIT**? ello se explica por el hecho de que existen nodos que se visitan más de una vez, característica que no posee el esquema *decreciente*; el número de **C_DIT** que se han de calcular depende, no sólo del **NNA** sino también del número de veces que algunos de éstos se han de revisitar, **NNR**.

☞ Si el esquema *creciente* se lleva a cabo con aprovechamiento de los cálculos previos, el ahorro debido a la recuperación de las **DL** evaluadas para radios de búsqueda inferiores al de la respuesta no es significativo respecto al número de cadenas **DL** exploradas en la fase final; por tanto, el mantenimiento de los valores de **DL** en la estructura no es crítico desde el punto de vista de la realización y su inclusión implicaría un gasto adicional de memoria, figura 14.

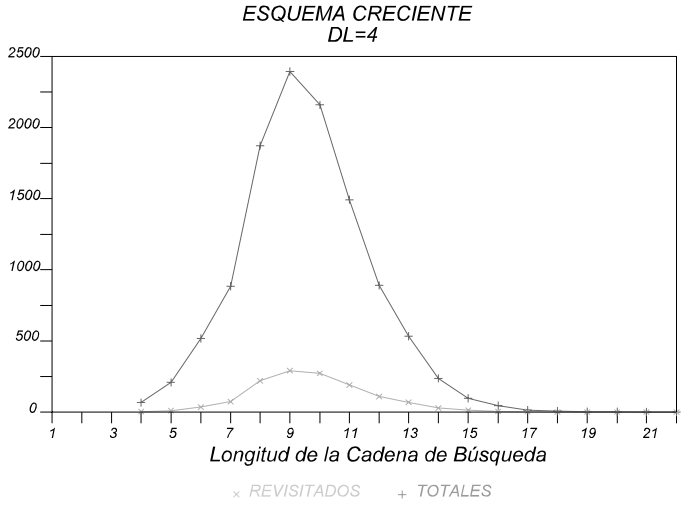


Figura 14

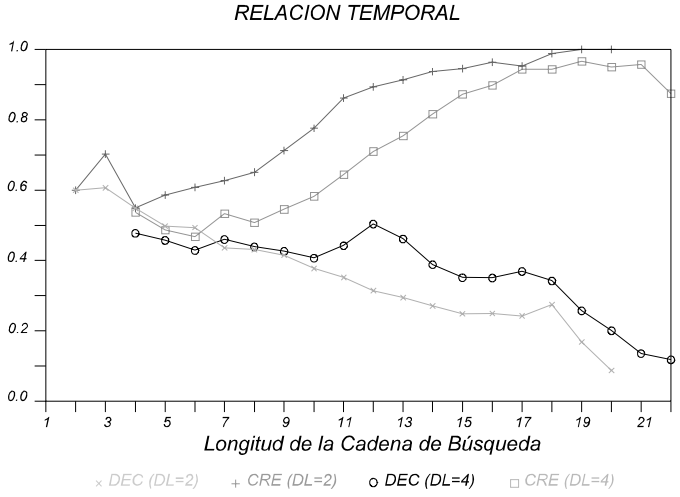


Figura 15

☞ Aproximadamente, en las realizaciones de los esquemas *decreciente* y *creciente*, el recurso temporal se distribuye en el recorrido por el índice evaluando **C_DIT** y en los cálculos de **DL**. En la figura 15, se representa la razón entre el tiempo empleado en el índice y el tiempo total; puede observarse cómo en el *decreciente* es aún importante el tiempo empleado en cálculos de **DL** –el índice no llega a demandar el 50% de este recurso–; en el *creciente*, la mayor parte del tiempo de cálculo se invierte en el recorrido por el índice, lo cual sugiere que se han de investigar formas de optimizar estos recorridos. Para ambos esquemas, podrán ser empleados nuevos filtros –que aporten propiedades ignoradas por

DIT con un costo atractivo– con el fin de reducir aún más el número de **DL** que se han de calcular.