Capítulo 1.

Capítulo 2.

Capítulo 3.
Optimización de los Esquemas de Búsqueda Decreciente y Creciente

El tiempo de respuesta de la estructura **S-D** con ambos esquemas está condicionado por la utilización de una poda adecuada en el índice –su eficacia depende de la cantidad de ramales en los que no haya respuesta que se evite explorar– y por la capacidad de reducir el número de veces que se evalúa **DL** –influida por la proximidad existente entre el filtro y **DL**.

3.1. Podas en el Índice: PA y PP.

La porción de estructura recorrida en los esquemas de búsqueda *creciente* y *decreciente* está limitada por la **DIT** entre la cadena de búsqueda y las cadenas del diccionario. La exploración de una alternativa en cada nodo discriminante se realiza siempre que el valor acumulado de las componentes de **DIT**, **CA_DIT**, obtenidas a lo largo del camino desde la raíz, no supere al umbral **DTM**. Este tipo de poda, que se ha estado utilizando hasta ahora y cuyo valor de corte **CA_DIT** depende exclusivamente de los ancestros, se denomina **PA**.

Se propone una optimización de la poda **PA** que permite reducir la porción de estructura recorrida y, con ello, el número de componentes de **DIT** evaluadas. Se consigue al estimar en cada nodo el valor mínimo atribuible a **DIT** en el camino que va desde la raíz hasta cualquier *nodo_SIT* que penda de ese nodo. Este valor mínimo es el que poda el árbol si supera el umbral **DTM**. Por el carácter predictivo que importa, se ha denominado poda **PP**. Durante el proceso de búsqueda, en cada nodo, para definir el valor que como mínimo tendrá **DIT** es suficiente con suponer que existe una inclusión entre los conjuntos de caracteres no tratados –de la cadena de búsqueda en las cadenas del diccionario que penden de ese nodo o viceversa. Este valor se determina sin más que añadir a **CA_DIT** el valor absoluto de la diferencia entre la suma de las frecuencias de los caracteres no tratados de la cadena de búsqueda, $\sum_{\alpha i \in CNT} B_{\alpha i}$, y los de las cadenas del diccionario correspondientes, $\sum_{\alpha i \in CNT} X_{\alpha i}$; ambas sumas se evalúan sencillamente de la forma:

$$\sum_{\alpha i \in CNT} B_{\alpha i} = |B| - \sum_{\alpha i \in CT} B_{\alpha i}$$

$$\sum_{\alpha i \in CNT} X_{\alpha i} = lr - \sum_{\alpha i \in CT} X_{\alpha i}$$

donde **lr** es la longitud asociada al ramal que se está explorando; tanto $\sum_{\alpha i \in CT} B_{\alpha i}$ como $\sum_{\alpha i \in CT} X_{\alpha i}$ representan, respectivamente, la suma de frecuencias de los caracteres tratados de la cadena de búsqueda y de las cadenas del diccionario que comparten el nodo.

3.1.1. Resultados Experimentales.

Se ha realizado un estudio experimental para comprobar el efecto de la poda **PP** en las realizaciones de los esquemas de búsqueda *decreciente* y *creciente.*

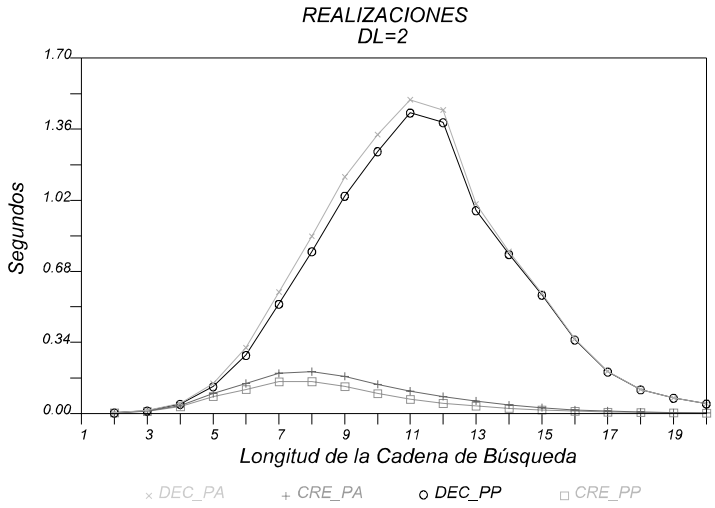

Figura 16

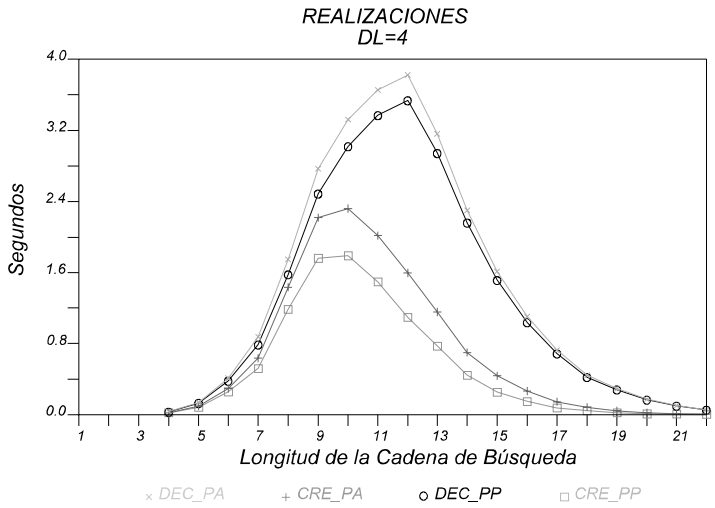

Figura 17

☞ En las figuras 16 y 17, se percibe nítidamente un descenso en los tiempos de búsqueda de los esquemas *decreciente* y

creciente que utilizan la poda **PP** frente a los correspondientes en los que actúa la poda **PA**; esta diferencia se manifiesta de forma más acusada en el esquema *creciente*. La mejora es más notable a medida que aumenta **DL**.

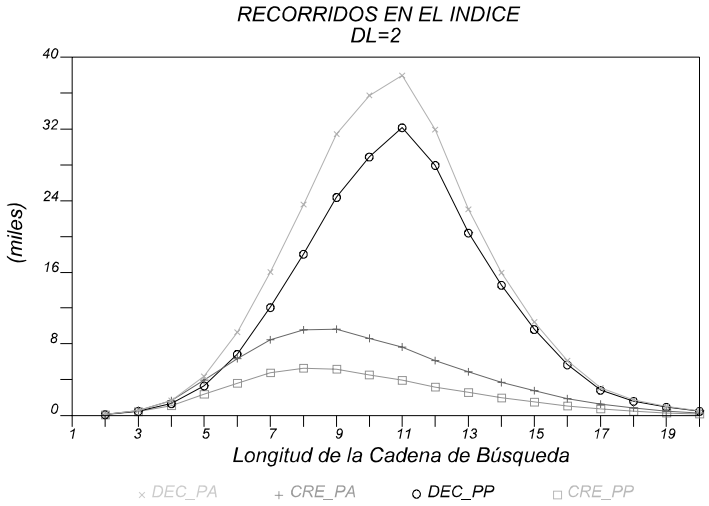

Figura 18

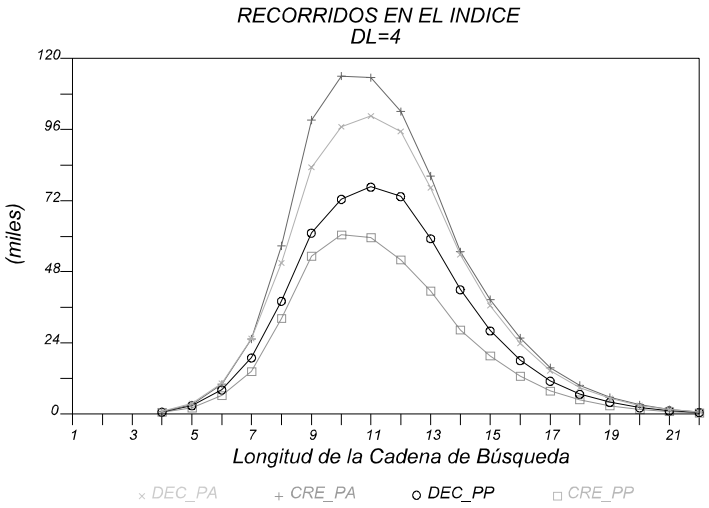

Figura 19

☞ Esta mejor realización se explica exclusivamente en base al ahorro de **C_DIT**, figuras 18 y 19, ya que la aplicación de la nueva poda afecta sólo al recorrido en el índice y no al número de **DL** que se han de evaluar.

3.2. Distancia de Santana, DS.

Dado que en el cálculo de **DIT** se tienen en cuenta los caracteres comunes aunque no las posiciones que éstos ocupan, siendo esta circunstancia relevante en el cálculo de **DL**, cabe pensar en un filtro intermedio que tenga en cuenta la subsecuencia común más larga entre las dos cadenas.

Diversos artículos presentan algoritmos para encontrar las subsecuencias comunes más largas entre dos cadenas. Wagner y Fischer, [WF74], desarrollan un algoritmo de aplicación del cálculo de **DL** a este fin, con un tiempo de ejecución y una ocupación proporcionales al producto de las longitudes de las cadenas. Hirschberg, [HI75], presenta un algoritmo con un tiempo también cuadrático y un espacio lineal. Posteriormente, Hunt y Szymanski, [HS77], han optimizado el tiempo de ejecución y el espacio; este algoritmo ha sido a su vez mejorado por Kuo y Cross, [KC89], para cadenas de gran longitud.

Definición: *Una cadena **C** es una subsecuencia de una cadena **X** si y sólo si existe una aplicación*

$$F : \{1, 2,..., |C|\} \longrightarrow \{1, 2,..., |X|\}$$

*tal que **F** es una función monótona estrictamente creciente y F(i)=j ⇔ C<i>=X<j>.*

Definición: *Una cadena **C** es una subsecuencia común de dos cadenas **X** e **Y** si y sólo si **C** es una subsecuencia de **X** y **C** es subsecuencia de **Y**.*

Se presenta un algoritmo para el cálculo de la longitud de las subsecuencias comunes más largas, **Lsc**, entre dos cadenas **X** e **Y**, basado en el de Hunt y Szymanski, [HS77]; utiliza un vector, **tr**, que se inicializa a $|X|+1$ y se actualiza con la posición en **X** de cada nuevo carácter coincidente con alguno de **Y**. Colecciona las posiciones de los caracteres de **X** que forman parte de alguna subsecuencia común y cada vez que se detecta el inicio de una nueva subsecuencia se guarda de nuevo a partir del primer elemento del vector –las posiciones anteriormente almacenadas se pierden. El método consiste en situar en **tr** cada nueva posición en el lugar que le corresponde en orden creciente sustituyendo al valor existente. La longitud de las subsecuencias comunes más largas es igual al número de elementos de **tr** distintos del valor inicial.

Algoritmo para el cálculo de la longitud de las subsecuencias comunes más largas:

```
Función Lsc (X,Y)
    primero: primer carácter del alfabeto
    ultimo: último carácter del alfabeto
    vf(.): vector de frecuencias de X
    lis(.,.): matriz que en cada fila contiene, en orden decreciente, las posiciones en X
    de cada uno de sus caracteres
    lc: longitud de X más uno
    tr(.): almacena las posiciones de los caracteres de X que forman parte de alguna
    subsecuencia común de X e Y
    lista(.): almacena una copia de una fila de la matriz lis correspondiente al carácter
    tratado
para c=primero hasta ultimo hacer
    vf(c)=0
fin para
para i=|X| hasta 1 hacer
    vf(X<i>)=vf(X<i>)+1
    lis(X<i>,vf(X<i>))=i
fin para
lc=|X|+1
para i=1 hasta |Y| hacer
    tr(i)=lc
fin para
```

```
para i=1 hasta |Y| hacer
    si vf(Y<i>) ≠ 0 entonces
        frec=vf(Y<i>)
        lista=lis(Y<i>)
        z=1
        repetir
            pos=lista(z)
            k=1
            mientras tr(k) < pos hacer
                k=k+1
            fin mientras
            tr(k)=pos
            z=z+1
        hasta que z > frec
    fin si
fin para
k=1
mientras tr(k) < lc hacer
    k=k+1
fin mientras
devolver (k-1)
```

Ejemplo:

Sean *X=trabajo* e *Y=pasajero,* el vector de frecuencias de **X**, **vf**, contiene:

$$vf(a)=2,\ vf(b)=1,\ vf(j)=1,\ vf(o)=1,\ vf(r)=1,\ vf(t)=1$$

y

$$vf(\alpha)=0 \quad \forall \alpha \in \{a,b,c,...,z\}-\{a,b,j,o,r,t\}$$

cada fila de la matriz **lis** se corresponde con un carácter y contiene sus posiciones en **X** en orden decreciente, en este caso:

$$lis(a)=(5,3),\ lis(b)=(4),\ lis(j)=(6),\ lis(o)=(7),\ lis(r)=(2),\ lis(t)=(1)$$

En el algoritmo de Hunt y Szymanski el contenido de esta matriz consiste solamente en las listas de posiciones de los caracteres coincidentes de ambas cadenas.

El vector **tr** inicialmente:

$$(8, 8, 8, 8, 8, 8, 8, 8)$$

se comprueban los caracteres coincidentes de ambas cadenas recorriendo la cadena *Y=pasajero* de izquierda a derecha. El primer carácter coincidente es **a** por tanto:

lista=(5, 3), pos=5

se actualiza **tr**:

(5, 8, 8, 8, 8, 8, 8, 8)

pos=3

(3, 8, 8, 8, 8, 8, 8, 8)

de nuevo para el carácter **a**:

lista=(5, 3), pos=5

(3, 5, 8, 8, 8, 8, 8, 8)

pos=3

(3, 5, 8, 8, 8, 8, 8, 8)

para el carácter **j**:

lista=(6), pos=6

(3, 5, 6, 8, 8, 8, 8, 8)

para el carácter **r**:

lista=(2), pos=2

(2, 5, 6, 8, 8, 8, 8, 8)

finalmente para el carácter **o**:

lista=(7), pos=7

(2, 5, 6, 7, 8, 8, 8, 8)

por tanto

$Lsc(X,Y)=4$.

Análisis del algoritmo:

La cantidad total de tiempo usada por este algoritmo es proporcional al número de asignaciones ejecutadas. Realiza: **m** –número de caracteres del alfabeto– para inicializar el vector de frecuencias de la cadena **X**, $2*|X|$ para actualizar dicho vector y la matriz **lis**, una para inicializar **lc**, e $|Y|$ para inicializar el vector **tr**; en conjunto

$$m+2*|X|+1+|Y|$$

El siguiente bucle *para* está controlado por el número de elementos del conjunto $\{(u,v)/\ X<u>=Y<v>\}$, denotado por **r** –7 es el número máximo de asignaciones realizadas en cada coincidencia. Dentro de este mismo bucle se realiza una búsqueda secuencial de la variable **pos** en el vector **tr**; el número esperado de asignaciones es $|Y|/2$, ya que se espera que **pos** se encuentre en la mitad del vector. Por último se realiza una inicialización de **k** y de nuevo una búsqueda secuencial en el vector **tr** que aporta $|Y|/2$ asignaciones; por tanto, se tienen:

$$r*(7+|Y|/2)+1+|Y|/2$$

en total:

$$m+2*|X|+1+|Y|+r*(7+|Y|/2)+1+|Y|/2$$

Las búsquedas secuenciales se pueden realizar utilizando búsquedas dicotómicas, para cadenas de gran longitud, con lo que se pasa de $|Y|/2$ a $log_2|Y|$. Si además se supone que $|X|\leq|Y|$, se tiene:

$$m+2+2*|X|+|Y|+r*(7+log_2|Y|)+log_2|Y| \leq$$
$$\leq m+2+3*|Y|+r*(7+log_2|Y|)+log_2|Y|$$

por tanto la complejidad de cálculo es $O(r*log_2|Y|+|Y|)$.♦

Definición: *Dadas dos cadenas **X** e **Y**, se define la distancia de Santana, DS(X,Y), como la diferencia entre la longitud mayor de ambas cadenas y la longitud de las subsecuencias comunes más largas, es decir:*

$$DS(X,Y) = máximo\{|X|,|Y|\} - Lsc(X,Y).$$

Teorema: *DS es una distancia en el espacio de las cadenas.*

Demostración:

a) $\forall X, Y: DS(X,Y)=0 \Leftrightarrow X=Y$,0

$$DS(X,Y) = 0 \Leftrightarrow máximo\{|X|,|Y|\} - Lsc(X,Y) = 0 \Leftrightarrow$$

$$\Leftrightarrow máximo\{|X|,|Y|\} = Lsc(X,Y) \Leftrightarrow |X|=|Y|=Lsc(X,Y) \Leftrightarrow X=Y.$$

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

Por la propia definición.

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

Sea **LC** la longitud de las subsecuencias comunes más largas de **X**, **Y** y **Z**.

$$DS(X,Y) + DS(Y,Z) =$$
$$= máximo\{|X|,|Y|\}-Lsc(X,Y)+máximo\{|Y|,|Z|\}-Lsc(Y,Z) =$$
$$= máximo\{|X|,|Y|\}+máximo\{|Y|,|Z|\}-LC-[Lsc(X,Y)+Lsc(Y,Z)-LC]$$

como:

$$Lsc(X,Y) + Lsc(Y,Z) - LC \leq |Y|$$

ya que **LC** es mayor o igual que el número de caracteres de **Y** que forman parte simultánemente de una subsecuencia común más larga de **X** e **Y** y de una subsecuencia común más larga de **Y** y **Z**; se tiene que:

$$DS(X,Y) + DS(Y,Z) \geq$$
$$\geq máximo\{|X|,|Y|\}+máximo\{|Y|,|Z|\}-LC-|Y|$$

dado que:

$$máximo\{|X|,|Y|\}+máximo\{|Y|,|Z|\}-|Y| \geq máximo\{|X|,|Z|\}$$

se tiene que:

$$DS(X,Y) + DS(Y,Z) \geq máximo\{|X|,|Z|\}-LC$$

y como:

$$Lsc(X,Z) \geq LC$$

entonces:

$$DS(X,Y) + DS(Y,Z) \geq máximo\{|X|,|Z|\}-Lsc(X,Z) = DS(X,Z).\blacksquare$$

Lema: *Sea* ***C*** *una subsecuencia común entre dos cadenas* ***X*** *e* ***Y****, se verifica que:*

$$DIT(X,Y) = DIT(X-C,Y-C)$$

donde ***X-C*** *(****Y-C****) representa la subcadena de* ***X*** *(****Y****) que se obtiene al eliminar la subsecuencia* ***C****.*

Demostración:

Se desprende inmediatamente de la definición de **DIT**.■

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

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

Demostración:

Sean:

$$l_{max} = máximo\{|X|,|Y|\},\ l_{min} = mínimo\{|X|,|Y|\}$$

y **C** una subsecuencia común más larga entre **X** e **Y**.

Si todos los caracteres de las subcadenas **X-C** e **Y-C** son diferentes, *DIT(X,Y)* alcanza el máximo valor posible, que será:

$$l_{max}-|C|+l_{min}-|C| + (l_{max}-|C|-(l_{min}-|C|)) =$$
$$= 2*(l_{max}-|C|) = 2*DS(X,Y)$$

por tanto, en general:

$$DIT(X,Y) \leq 2*DS(X,Y)$$

Por otro lado, $DL(X,Y)$ alcanza su valor mínimo cuando la transformación de una cadena en otra, se lleva a efecto a través de $l_{min}-|C|$ sustituciones y $(l_{max}-|C|-(l_{min}-|C|))$ inserciones en la cadena de menor longitud (o extracciones en la más larga); este valor es:

$$l_{min}-|C|+(l_{max}-|C|-(l_{min}-|C|)) = l_{max}-|C| = DS(X,Y)$$

implicando, en cualquier caso, que:

$$DS(X,Y) \leq DL(X,Y).\blacksquare$$

La acotación del teorema anterior, conjuntamente con el hecho de que **DS** tiene un menor costo computacional que **DL**, sugiere la inclusión de **DS** en los esquemas de búsqueda como filtro que tenga por objeto reducir el número de **DL** que se evalúan. El filtro **DS** se utiliza cada vez que se alcanza un $nodo\_SIT$. A continuación se muestran las modificaciones de los procedimientos que tratan dichos nodos en ambos esquemas.

*Algoritmos de tratamiento de los nodos_SIT para el esquema decreciente y creciente con el filtro **DS**:*

```
Procedimiento Tratardec_nS (nodo)
    dl: valor de DL entre B y el sinónimo_DIT actual
    l_max: máximo entre las longitudes de B y del ramal del nodo raíz que se está
    explorando
{l_max se calcula una vez conocido el ramal de descenso en el nodo raíz}
para j=1 hasta nodo.ns hacer
    DS=l_max-Lsc(B,nodo.s(j))
    si DS ≤ DLM entonces
        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 si
fin para
```

```
Procedimiento Tratarcre_nS (nodo)
    dl: valor de DL entre B y el sinónimo_DIT actual
    l_max: máximo entre las longitudes de B y del ramal del nodo raíz que se está
    explorando
{l_max se calcula una vez conocido el ramal de descenso en el nodo raíz}
para j=1 hasta nodo.ns hacer
    DS=l_max-Lsc(B,nodo.s(j))
    si DS ≤ DLM entonces
        dl=COP_DL(B,nodo.s(j))
        si dl = DLM entonces
            añadir nodo.s(j) al conjunto respuesta
            nms=nms+1
        fin si
    fin si
fin para
```

La cantidad total de tiempo empleada en la obtención de **DS** depende del tiempo de cálculo de la longitud de las subsecuencias comunes más largas. Se reduce el tiempo de ejecución de **Lsc** debido a que, dada una cadena de búsqueda **B**, las instrucciones correspondientes a la inicialización

y actualización de su vector de frecuencias, la actualización de la matriz **lis** y la inicialización de **lc** sólo se ejecutan una vez cuando *Lsc(B,X)* se utiliza en los esquemas de búsqueda.

Las cadenas más similares a **B** según **DS** no son necesariamente las **DL** más similares. En efecto, si **X** difiere de **B** en una extracción y en una inserción, entonces:

$$DL(B,X)=2$$

y

$$DS(B,X)=1$$

se puede encontrar una cadena, **Y**, que difiera de **B** en una sustitución, siendo:

$$DL(B,Y)=1$$

y

$$DS(B,Y)=1$$

por tanto **X** es una cadena más similar a **B** según **DS** que no es **DL** más similar. Por ejemplo: *B=antejo, X=tanteo* e *Y=antojo*.

3.2.1. Resultados Experimentales.

Se ha realizado un estudio experimental para comprobar la influencia del filtro **DS** en los esquemas de búsqueda *decreciente* y *creciente* dotados de la poda **PP**.

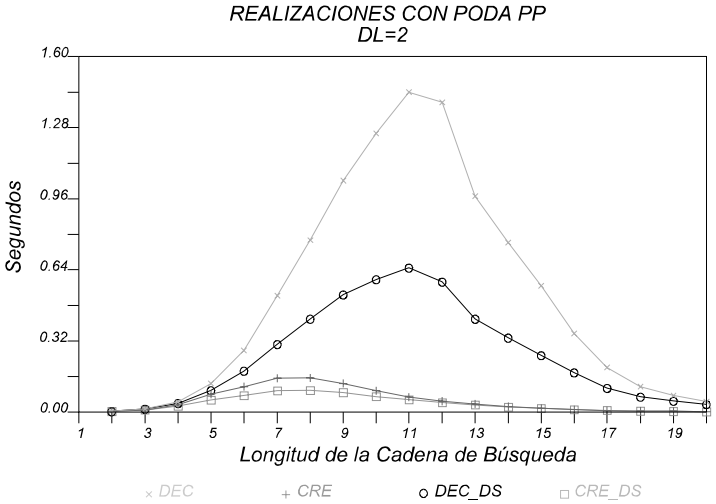

Figura 20

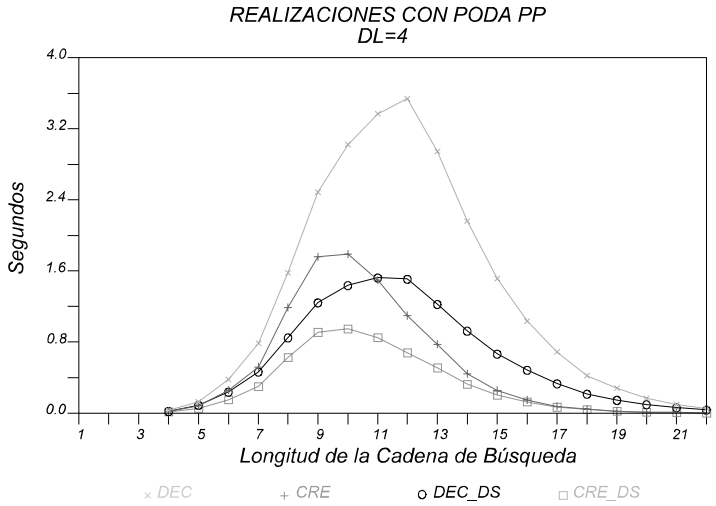

Figura 21

☞ En las figuras 20 y 21 se aprecia un descenso en los tiempos de búsqueda de los esquemas con poda **PP** al incorporarles el filtro **DS**, esta diferencia es más acusada en el esquema *decreciente* y aumenta de forma notoria a medida que crece **DL**.

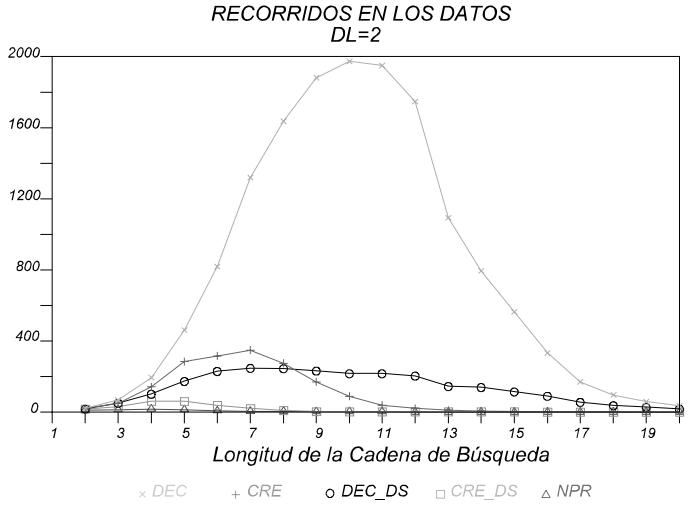

Figura 22

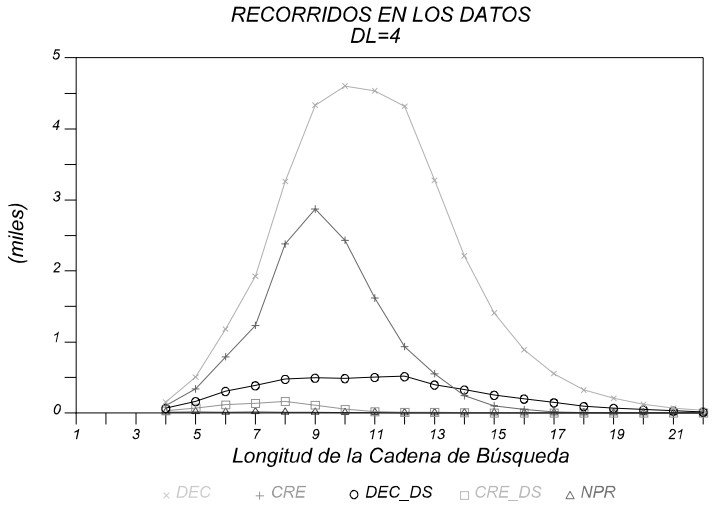

Figura 23

☞ La mejor realización de ambos esquemas al incluir el filtro **DS** se debe exclusivamente al ahorro del número de **DL** evaluadas, figuras 22 y 23, ya que no afecta en absoluto al recorrido en el índice. Obsérvese, en las figuras 22 y 23, como el número de **DL** evaluadas en ambos esquemas se acerca, bastante más en el *creciente* aunque con mayor ímpetu en el *decreciente,* al tamaño o multiplicidad de la respuesta, **NPR**. **NPR** constituye

el nivel mínimo teórico alcanzable para el número de **DL** a calcular.

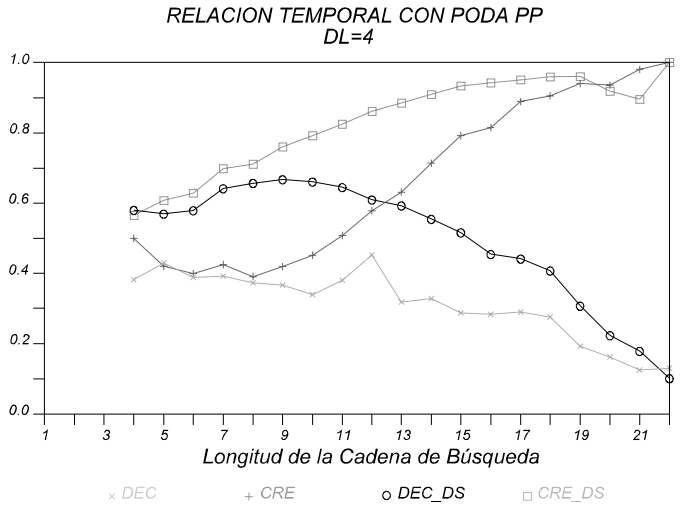

Figura 24

☞ La introducción de la poda **PP** acorta el recorrido en el índice, produciendo en consecuencia una disminución en la relación entre el tiempo empleado en el índice y el total; sin embargo, este efecto se ve ampliamente sobrepasado, figura 24, por la disminución del número de **DL** evaluadas como consecuencia de la introducción del filtro **DS**. La relación temporal del esquema *creciente* con **DS** indica que el tiempo empleado en el índice está por encima del 70% del total. El esquema *decreciente* con **DS** en cambio presenta un reparto más equitativo del tiempo, salvo para grandes longitudes en las que la fracción de tiempo consumido en el índice disminuye rápidamente.

En posteriores referencias a los esquemas de búsqueda *creciente* y *decreciente* se debe entender que llevan incorporados la poda **PP** y el filtro **DS**.