Paginación + Discretización del Fixed Queries Trie

Carina Mabel Ruano, Ana Valeria Villegas y Norma Edith herrera
Departamento de Informática
Univ. Nacional de San Luis
Argentina
{nherrera,cmruano,anaville}@unsl.edu.ar

Resumen

La próxima generación de administradores de bases de datos deberá ser capaz de indexar datos multimedia y responder consultas de proximidad con tanta eficiencia como actualmente responden consultas de búsqueda exacta. Estas nuevas bases de datos se pueden modelar como un espacio métrico, sobre los cuales ya se han diseñando numerosas técnicas de indexación. El *Fixed Queries Trie (FQTrie)* es una de ellas y ha demostrado experimentalmente tener un buen desempeño.

En investigaciones anteriores hemos realizado trabajos en torno a mejorar la eficiencia del FQ-Trie desde varios tópicos diferentes: cantidad de evaluaciones de distancia,tiempo extra de CPU y tiempo de I/O. Con respecto a los dos primeros, hemos encontrado un método de discretización que logra mejorar la eficiencia del FQTrie. Con respecto al tiempo de I/O, hemos diseñado una técnica basada en el particionamiento del espacio que permite reducir el tiempo de I/O.

En este trabajo combinamos ambas técnicas a fin de lograr una implementación completa del FQTrie que sea eficiente tanto en memoria principal como en memoria secundaria.

1. Introducción

La problemática de búsquedas por similitud en bases de datos no tradicionales puede formalizarse por medio del modelo de *espacios métricos*. Un espacio métrico es un par $(\mathcal{X}, d)$, donde $\mathcal{X}$ es un conjunto de objetos y $d : \mathcal{X} \times \mathcal{X} \rightarrow R^+$ es una función de distancia que modela la similitud entre los elementos de $\mathcal{X}$. La función d cumple con las propiedades características de una función de distancia: $\forall x, y \in \mathcal{X},\ d(x, y) \geq 0$ (positividad), $\forall x, y \in \mathcal{X},\ d(x, y) = d(y, x)$ (simetría), $\forall x, y, z \in \mathcal{X},\ d(x, y) \leq d(x, z) + d(z, y)$(desigualdad triangular). La base de datos es un conjunto finito $\mathcal{U} \subseteq \mathcal{X}$.

Una de las consultas típicas que implica recuperar objetos similares de una base de datos es la *búsqueda por rango*, que denotaremos con $(q, r)_d$. Dado un elemento de consulta q, al que llamaremos *query* y un radio de tolerancia r, una búsqueda por rango consiste en recuperar aquellos objetos de la base de datos cuya distancia a q no sea mayor que r, es decir, $(q, r)_d = \{u \in \mathcal{U} : d(q, u) \leq r\}$.

Una forma trivial de resolver este tipo de búsquedas es examinando exhaustivamente la base de datos, es decir, comparando cada elemento de la base datos con la query. En general, esto resulta demasiado costoso en aplicaciones reales y no es posible realizarlo. Se han logrado avances importantes sobre espacios métricos en torno al concepto de construir un *índice*, es decir, una estructura de datos que permita reducir el tiempo necesario para resolver una consulta. En este tiempo T influyen tres factores; por un lado tenemos la cantidad de evaluaciones de la función de distancia d que se realizaron durante el proceso de búsqueda; por otro lado tenemos una cierta cantidad de operaciones adicionales que implican un tiempo extra de CPU; finalmente, tenemos un tiempo de I/O determinado por la cantidad de accesos a memoria secundaria, si es que fuera necesario; en símbolos:

T= #evaluaciones de $d\times$ complejidad(d) + tiempo extra de CPU + tiempo de I/O

Hay dos casos importantes a considerar: si el índice y los datos pueden ser mantenidos en memoria principal, o si es necesario utilizar memoria secundaria para los índices y/o datos. En el primer caso el primer término de T es el de mayor peso y, en consecuencia, el objetivo principal es reducir los cálculos de distancia realizados; en este sentido, un algoritmo se considera eficiente si puede contestar una consulta por similitud realizando un número pequeño de cálculos de distancia, sublineal respecto a la cantidad de elementos en la base de datos. Para los algoritmos en memoria secundaria, además de realizar pocos cálculos de distancia, se requiere que realicen pocos accesos a disco. Con respecto al tiempo extra de CPU, si bien suele ser el de menor peso en T, es importante reducirlo porque produce que en la práctica la búsqueda sea más rápida aún cuando estemos realizando la misma cantidad de evaluaciones de distancia y la misma cantidad de accesos a disco.

En [3] se presenta un desarrollo unificador de las soluciones existentes en la temática. En dicho trabajo se muestra que todos los enfoques para la construcción de índices en espacios métricos consisten en particionar el espacio en clases de equivalencia e indexar las clases de equivalencia. Luego, durante la búsqueda, usando el índice y la desigualdad triangular, se descartan algunas clases y se busca exhaustivamente en las restantes. La diferencia entre los distintos algoritmos de indexación radica en cómo construyen esta relación de equivalencia. Básicamente se pueden distinguir dos grupos:

Algoritmos basados en pivotes: estos algoritmos construyen la relación de equivalencia tomando la distancia de los elementos de la base a un conjunto preseleccionado de elementos denominados *pivotes*; sea $\{p_1, p_2, \ldots, p_k\}$ el conjunto de pivotes, dos elementos x e y son equivalentes si y solo si están a la misma distancia de todos los pivotes, es decir, $d(x, p_i) = d(y, p_i),\ \forall i = 1 \ldots k$. Durante la indexación, se seleccionan los k pivotes y se le asigna a cada elemento x de la base de datos el vector o firma $(d(x, p_1), d(x, p_2), \ldots, d(x, p_k))$. Ante una búsqueda $(q, r)_d$, se usa la

desigualdad triangular junto con los pivotes para filtrar elementos de la base de datos sin medir su distancia a la query q. Para ello se computa la distancia de q a cada uno de los pivotes p_i, y luego se descartan todos aquellos elementos a, tales que para algún pivote p_i se cumple que $\mid d(q,p_i) - d(a,p_i) \mid > r$. Los elementos no descartados forman parte de una *lista de candidatos*, que posteriormente se comparan directamente con la query q para decidir si forman o no parte de la respuesta.

Algoritmos basados en particiones compactas: en este caso la relación de equivalencia se construye teniendo en cuenta la cercanía de los elementos a un conjunto preseleccionado de elementos denominados *centros*; dos elementos son equivalentes si tienen al mismo centro c como su centro más cercano. Durante la indexación, seleccionan un conjunto de *centros* $\{c_1, c_2, \ldots, c_k\}$ y dividen el espacio asociando a cada centro c_i el conjunto de puntos que tiene a c_i como su centro más cercano. Existen muchos criterios posibles para descartar zonas durante una búsqueda. Los dos más populares son *criterio del hiperplano* y *criterio del radio de cobertura* [3].

Uno de los principales obstáculos en el diseño de buenas técnicas de indexación es lo que se conoce con el nombre de *maldición de la dimensionalidad*. El concepto de dimensionalidad está relacionado a la dificultad o facilidad de buscar en un determinado espacio métrico. La dimensión intrínseca de un espacio métrico se define en [3] como $\rho = \frac{\mu^2}{2\sigma^2}$, siendo μ y σ^2 la media y la varianza respectivamente de su histograma de distancias. Es decir que, a medida que la dimensionalidad intrínseca crece, la media aumenta y su varianza se reduce. Esto significa que el histograma de distancia se concentra más alrededor de su media, lo que influye negativamente en los algoritmos de indexación.

En investigaciones anteriores hemos trabajado en torno a mejorar la eficiencia del Fixed Queries Trie (FQTrie) [1], un índice basado en pivotes. En [4] se presenta un método de discretización, basado en los histogramas de distancias de los pivotes, que logra una implementación eficiente del FQTrie no sólo en términos de cantidad de evaluaciones de distancia de la función d, sino también en tiempo extra de CPU. En [6] se desarrolla una técnica de paginado que permite reducir el tiempo de I/O, que se basa en la idea de particionar la base de datos y agrupar en cada parte elementos similares.

En este artículo presentamos una implementación del FQTrie que combina ambas técnicas con el fin de lograr reducir las tres componentes que afectan el tiempo de resolución de una búsqueda por similitud: cantidad de evaluaciones de distancias, tiempo extra de CPU y cantidad de accesos a disco.

El artículo está organizado de la siguiente manera. Comenzamos en la sección 2 dando una breve reseña del índice FQTrie. Luego, en la sección 3, presentamos el método de discretización, el método de paginado y la forma de combinarlos. En la sección 4 exponemos la evaluación experimental de esta nueva propuesta y finalizamos en la sección 5 dando las conclusiones y el trabajo futuro.

2. Fixed Queries Trie

El *Fixed Queries Trie (FQTrie)* es una estructura basada en pivotes que fue presentado en [1] como una mejora al FQA [2] utilizando tablas lookup para mejorar los tiempos de búsquedas.

Habíamos visto en la introducción que todos los índices basados en pivotes asocian a cada elemento x de la base de datos, la firma del mismo $(d(x,p_1), d(x,p_2), \ldots, d(x,p_k))$, donde $\{p_1, p_2, \ldots, p_k\}$ es el conjunto de pivotes. La diferencia entre los distintos algoritmos basados en pivotes radica en cómo buscan dentro de ese conjunto de firmas. En el caso del FQTrie las firmas de los elementos son consideradas como cadenas de caracteres de longitud fija y se utiliza un *Árbol Digital* o *Trie* [5] para indexar las firmas de todos los elementos de la base de datos.

El FQTrie representa cada firma $(d(x,p_1),\ d(x,p_2),\ldots,d(x,p_k))$, haciendo uso de una función de discretización. Dicha función mapea los números reales positivos devueltos por la función d en valores discretos de tamaño b_p bits. Este valor discreto depende del valor devuelto por $d(x,p_i)$ y del pivote p_i. Formalmente la función de discretización se define como sigue:

$$\delta : \mathbb{R}^+ \times \mathbb{K} \rightarrow \{0,\ldots,2^{b_p}-1\}$$

De esta forma, para un elemento $x \in \mathbb{X}$ la firma de x se obtiene como la concatenación de la discretización de las distancias del objeto a cada pivote, en símbolos:

$$\delta^*(x) = \delta_{p_1}(d(x,p_1)) \centerdot \delta_{p_2}(d(x,p_2)) \centerdot \ldots \centerdot \delta_{p_k}(d(x,p_k))$$

Podemos extender la función δ_p a intervalos obteniendo así un conjunto de firmas individuales:

$$\delta_p([r_1,r_2]) = \{\delta_p(r)/r \in [r_1,r_2]\}$$

Formalmente δ_p está definida para números reales positivos, Podemos extender esta definición a todo el dominio de los números reales de la siguiente manera: $\delta([r_1,r_2]) = \delta([0,r_2])$ si $r_1 < 0$.

Ante una búsqueda $(q,r)_d$ se calcula la firma de la query definida de la siguiente manera:

$$\delta^*((q,r)_d) = \{\delta_{p_1}([d(q,p_1)-r, d(q,p_1)+r])\} \centerdot \ldots \centerdot \{\delta_{p_k}([d(q,p_k)-r, d(q,p_k)+r])\}$$

Es decir que $\delta^*((q,r)_d)$ es un conjunto de firmas que indica cuál debería ser la firma de un elemento para que sea candidato a formar parte de la respuesta de la query. Con estas definiciones, usando la desigualdad triangular, se puede demostrar que si x satisface la búsqueda $(q,r)_d$, entonces $\delta^*(x) \in \delta^*((q,r)_d)$. Luego, la lista de candidatos se puede definir como: $\{x/\delta^*(x) \in \delta^*((q,r)_d)\}$

Denotaremos con $\mathcal{U}^*$ al conjunto de firmas de la base de datos $\mathcal{U}$: $\mathcal{U}^* = \{\delta^*(x)/x \in \mathcal{U}\}$. Con las notaciones dadas, computar la lista de candidatos es equivalente a realizar la intersección $\mathcal{U}^* \cap \delta^*((q,r)_d)$. Como ya lo mencionáramos, en el FQTrie cada firma es considerada como una secuencia de caracteres y se utiliza un Árbol Digital o Trie para representar $\mathcal{U}^*$. Luego, la intersección anterior se calcula usando el Trie.

Notar que existe una cantidad exponencial de firmas en $\delta^*((q,r)_d)$. Las firmas se obtienen como concatenación ordenada de firmas respecto de cada pivote. Esto significa que, si cada pivote produce v_{p_i} firmas, entonces $|\delta^*((q,r)_d)| = \prod_{i=1}^{k} v_{p_i}$. Por ejemplo, si tenemos 32 pivotes, y para una búsqueda cada pivote produce 2 firmas, entonces $|\delta^*((q,r)_d)| = 2^{32}$. En consecuencia, no es viable calcular explícitamente el conjunto de firmas. En su lugar, se utiliza una representación implícita: *tablas lookup* [1].

Buscar una cadena en un Trie toma tiempo proporcional a la cantidad de caracteres en ella, independientemente de la cantidad de elementos contenidos en el conjunto. Dada una cadena, los caracteres que la conforman son los que direccionan la búsqueda en el Trie. Para el caso del FQTrie, la búsqueda se realiza con la asistencia de la *tabla lookup* la cual contiene el conjunto de firmas de la búsqueda ($\delta^*(q,r)$), permitiendo búsquedas de múltiples cadenas en el mismo recorrido.

3. Paginación y Discretización del FQTrie

En investigaciones anteriores hemos trabajado en torno a mejorar la eficiencia del FQTrie. En [4] se presenta un método de discretización, basado en los histogramas de distancias de los pivotes, que logra una implementación eficiente del FQTrie no sólo en términos de cantidad de evaluaciones de distancia, sino también en tiempo extra de CPU. En [6] se desarrolla una técnica de paginado que

permite reducir el tiempo de I/O, que se basa en la idea de particionar la base de datos y agrupar en cada parte elementos similares. En este trabajo combinamos ambas técnicas con el fin de lograr una implementación del FQTrie que sea eficiente tanto en memoria principal como en memoria secundaria.

Comenzaremos explicando la técnica de paginado, luego presentamos las funciones de discretización y finalmente detallamos la forma de combinar ambas técnicas.

3.1. Técnica de Paginado para el FQTrie

En [6] se presenta una técnica que permite manejar espacios métricos cuyo índice completo y/o datos exceda la capacidad de la memoria principal. Para ello, en lugar de usar el enfoque tradicional de modificar la estructura para que sea eficiente en memoria secundaria, se particiona el espacio de manera tal que cada una de las partes entre en memoria principal, las que posteriormente se indexan en forma separada. Luego, una búsqueda se resuelve buscando en cada parte, lo que puede ser hecho en memoria principal.

La técnica presentada en dicho artículo trata de agrupar objetos que sean similares dentro de cada parte con el objetivo de que, ante una consulta, una parte o contiene muchos elementos relevantes para la query o no contiene ninguno. De esta manera, se intenta amortizar los costos de accesos a las partes que residen en disco.

Una consulta $(q, r)_d$ se resuelve realizando los siguientes pasos con cada una de las partes que conforman la base de datos:

- se carga el índice de la parte.
- se busca en él para obtener la lista de elementos candidatos.
- si la lista de candidatos no es vacía, se carga la parte correspondiente en memoria principal, para poder comparar cada elemento de la lista contra la query q.

La técnica de particionamiento diseñada se basa en la distancia LCS(Longest Common Subsequence) que calcula la longitud de la máxima subsecuencia común entre dos cadenas. Esta técnica, que denotaremos con PLCS, detecta grupos de elementos parecidos a partir de los cuales genera la partición.

En [6] se estudia el comportamiento de PLCS sobre diccionarios de palabras con la función de distancia de edición y sobre documentos de texto con la función de distancia coseno. Allí se concluye que PLCS es competitiva cuando se utiliza como espacio métrico diccionarios de palabras, logrando disminuir la cantidad de accesos a disco en un $21\,\%$ respecto de un particionado totalmente aleatorio. También se demuestra que es eficiente disminuyendo cantidad de accesos a disco requeridos ante una consulta, respecto de no particionar la base de datos y dejar el manejo al sistema operativo. El comportamiento ha sido diferente cuando el espacio métrico utilizado son documentos de texto y los resultados no han sido tan alentadores respecto de no particionar la base de datos. En el mencionado artículo se concluye que, si los elementos de la base de datos son mayores que el tamaño de un página de disco, no es bueno particionar el espacio métrico con PLCS ya que eso implica almacenar punteros en cada parte y los resultados han mostrado que, para estos casos, no es posible obtener ventaja de dividir el espacio métrico en partes, aún cuando la división se realiza en forma controlada.

3.2. Funciones de Discretización

La principal ventaja de la función de discretización se centra en el grado de libertad en cuanto al uso de memoria disponible ya que podemos decidir cuántos bits asignar a cada pivote y determinar así

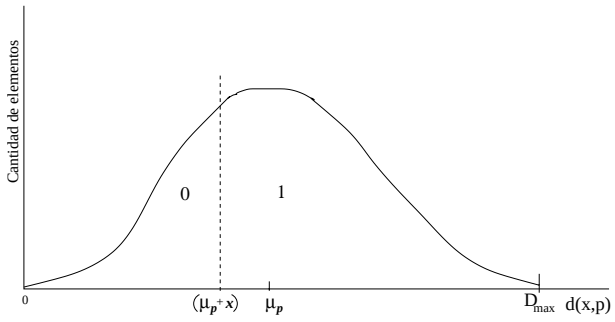

Figura 1: *Partición provocada por la función de discretización* δ_{μ_p} *con* $x < 0$.

el tamaño de la firma de cada objeto. Además permite establecer un balance entre el poder de filtrado y el espacio utilizado ya que, a medida aumenta la cantidad de bits asignados a un pivote, aumenta la presición pero disminuye la cantidad de pivotes que pueden ser contenidos en la misma cantidad de espacio.

En [2] y [4] se introducen varias funciones de discretización, entre la que se destacan:

- **Partes Iguales**: esta técnica consiste en dividir el espacio en partes de igual tamaño. Sea $P = \{p_1, p_2, \cdots, p_k\}$ el conjunto de pivotes. Para cada p_i se calcula $D_{max} = max_{u \in (\mathcal{U}-P)}\{d(p_i, u)\}$ y $D_{min} = min_{u \in (\mathcal{U}-P)}\{d(p_i, u)\}$. Luego, el rango $D_{max} - D_{min}$ es dividido en 2^{b_i} partes iguales, asociando a cada número $v \in \{0 \ldots 2^{b_i} - 1\}$ el intervalo $[D_{min} + v(D_{max} - D_{min})/2^{b_s}, D_{min} + (v+1)(D_{max} - D_{min})/2^{b_s}]$. Si bién esta técnica asegura que todas las partes son del mismo tamaño, no asegura que la cantidad de elementos en cada parte sea la misma.

- **Cantidades Iguales** : esta técnica divide el espacio intentando dejar la misma cantidad de elementos en cada parte. Por cada pivote se determina los $b_i - 1$ cuantiles uniformes que dividen el conjunto de valores de distancias en b_i subconjuntos de la misma cardinalidad. Luego se asigna un cuantil a cada valor entre 0 y $b_i - 1$. Esta técnica asegura que en cada intervalo existen exactamente $n/2^{b_i}$ objetos.

- **Media** : esta función divide el histograma local de un pivote p en dos partes, utilizando como límite divisor $\mu_p + x$, donde μ_p es la media del histograma y x es un número real. Luego, asigna 0 a todos aquellos valores que se ubiquen en el histograma a izquierda del límite divisor y 1 a los que se ubiquen a derecha (ver figura 1). El objetivo de esta función es divir la zona de mayor concentración de elementos.

De acuerdo a los resultados expuestos en [4], se concluye que la función *media* resulta ser la mejor elección si el histograma tiene forma de campana, siendo más competitiva que *partes iguales* y *cantidades iguales*. Si el histograma no tiene forma de campana, se realiza una adaptación de la función *media* a fin de que la misma divida la zona de mayor concentración de elementos.

3.3. Combinando Ambos Métodos

En este trabajo el objetivo es combinar ambas técnicas a fin de lograr un implementación del FQTrie que sea eficiente en términos de las tres componentes que afectan el tiempo de resolución de una consulta: cantidad de evaluaciones de la función de distancia d, cantidad de accesos a disco y tiempo extra de CPU.

Para ello, paginamos usando el método PLCS y construimos el índice de cada parte usando las funciones de discretización al momento de construir las firmas de los elementos. La figura 2 muestra

```
PLCS+Discretización(in 𝒰 , in δ, in B)
     1. ParticionarLCS(𝒰 ,𝒫𝒰)
     2. ℱ = ∅
     3. Para cada 𝒰_i ∈ 𝒫𝒰 hacer
     4.    Cargar 𝒰_i en memoria principal
     5.    Obtener 𝒰_i* = {δ*(x) : x ∈ 𝒰_i}
     6.    Indexar (𝒰_i*, FQTrie)
     7.    Si (size(ℱ) + size (FQTrie) >B )
     8.       grabar(ℱ)
     9.       ℱ = FQTrie
     10.   sino ℱ = ℱ ∪ FQTrie
     11. Fin Para
```

Figura 2: *Algoritmo que combina particionado LCS con funciones de discretización*

el pseudocódigo del algoritmo que permite combinar el método de paginación con el método de discretización. Como parámetros de entrada recibe la base de datos $\mathcal{U}$, la función de discretización δ y el tamaño de la página de disco B. El proceso ParticionarLCS es el que aplica la técnica de particionado a $\mathcal{U}$, obteniendo la partición del mismo $\mathcal{PU}$; este proceso se encarga además de grabar cada parte obtenida en una página disco. Dado que generalmente el FQTrie de una parte es pequeño, se agrupan varios de estos índices por página de disco. En el algoritmo se usa la variable $\mathcal{F}$, que mantiene el conjunto de índices creados y aún no grabados. Cuando el tamaño de $\mathcal{F}$ se adecúa a B, se graba el grupo corriente de índices y se comienza la creación de un nuevo grupo de índices.

Para las búsquedas no hay modificaciones respecto de lo explicado en la sección 3.1

Cabe señalar que, dado que el método de paginado PLCS es eficiente sólo para espacios cuyos objetos son, en tamaño, menor que el tamaño de una página de disco, la propuesta que aquí presentamos sólo es aplicable a espacios métricos con esta característica.

4. Evaluación Experimental

La evaluación de la técnica de particionado se realizó usando como espacio métrico diccionarios de palabras con la función de distancia de edición. Esta función es discreta y calcula la mínima cantidad de caracteres que hay que agregar, cambiar y/o eliminar a una palabra para obtener otra.

Se usaron en total 3 diccionarios: Español de $86{,}061$ palabras, Inglés de $69{,}069$ palabras y Francés de $138{,}257$ palabras. Cada uno de estos diccionarios fue paginado y luego indexado con firmas de tamaño 1, 2 y 4 bytes, y usando las funciones de discretización *media*, *cantidades iguales* y *partes iguales*.

Para cada diccionario se eligieron al azar 500 palabras las que fueron utilizadas en todos los experimentos. Para cada palabra de este grupo, se realizaron búsquedas por rango usando como radio de búsqueda r los valores $1, 2, 3$ y 4. Para cada búsqueda se contabilizó la cantidad de evaluaciones de distancias realizadas, la cantidad de páginas de disco accedidas y el tiempo total usado para resolver la consulta. En cada gráfica se muestran los promedios sobre las 500 búsquedas realizadas.

A fin de evaluar la eficiencia de combinar los métodos de discretización y paginado, los resultados obtenidos fueron comparados con los obtenidos paginando pero sin discretizar. Por cuestiones de espacio sólo mostramos las gráficas que consideramos más representativas; las restantes se encuentran disponibles para quien así lo requiera.

La figura 3 muestra los resultados para el diccionario Español usando firmas de 1 byte. En la parte superior se ha graficado el promedio de cantidad de evaluaciones de distancias (izquierda) y de

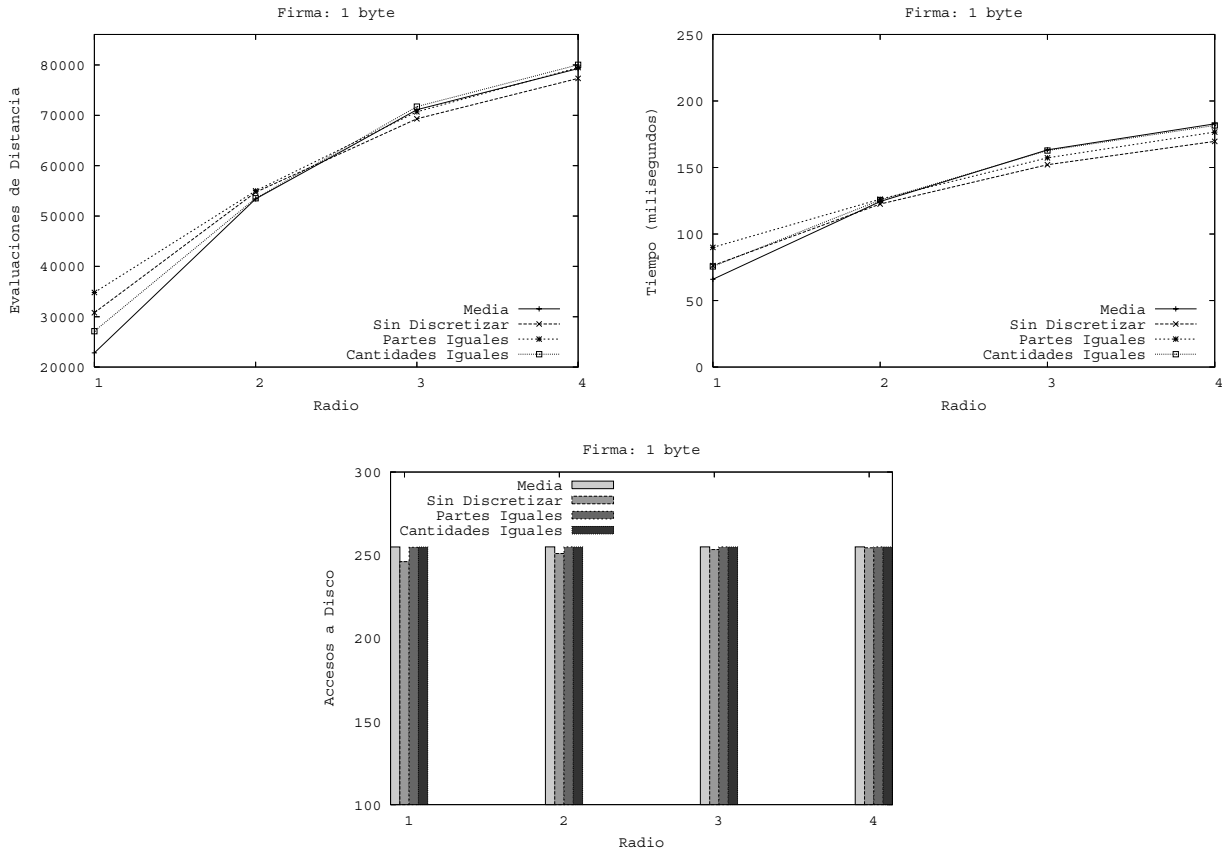

Figura 3: *Diccionario Español, tamaño de firma 1 byte*

tiempos totales de búsqueda (derecha); en la parte inferior se ha graficado el promedio de páginas accedidas durante el proceso de búsqueda. Se puede observar que, para búsquedas de alta selectividad (radio 1 y 2), PLCS con *media* y PLCS con *cantidades iguales* son más competitivas, en ese orden, que la opción de particionar con PLCS pero sin discretizar, logrando menor cantidad de evaluaciones de distancias y mejores tiempos de búsqueda. Notar que para radio 1, PLCS sin discretizar realiza menos accesos a disco que las restantes opciones, pero aún así no logra superarlas en tiempo. La razón de este comportamiento es que PLCS sin discretizar realiza aproximadamente un $30\,\%$ más de evaluaciones de distancias que PLCS con *media* y sólo un $5\,\%$ menos de acceso a disco; por lo tanto, en el tiempo total, termina pesando más esa cantidad extra de evaluaciones de distancias realizadas.

Esta situación cambia cuando consideramos búsquedas de baja selectividad (radios 3 y 4). Si bien todas las posibilidades consideradas obtienen resultados similares, PLCS sin discretizar y PLCS con *partes iguales* logran un desempeño levemente superior que las restantes. PLCS sin discretizar realiza un $5\,\%$ menos de evaluaciones de distancia que las restantes y tarda 10 milisegundos menos que PLCS con *partes iguales*, la que le sigue en eficencia.

La figura 4 muestra los resultados para tamaño de firma de 2 bytes. Se puede observar que, como era de esperar, todas las opciones logran mejorar su rendimiento dado que al tener mayor espacio se pueden usar mayor cantidad de pivotes lo que mejora la performance de cualquier índice basado en pivotes. Pero la que mayor beneficio obtiene es PLCS sin discretizar. Para búsquedas de radio 1, PLCS sin discretizar logra acercarse en cantidad de evaluaciones de distancia e igualar en tiempo a PLCS con *media*. Intuimos que la razón de esto es que PLCS sin discretizar, para radio 1, realiza menos accesos a disco que PLCS con *media* lo que redunda en beneficio del tiempo total de búsqueda.

Para búsquedas de baja selectividad, nuevamente PLCS sin discretizar y PLCS con *partes iguales*

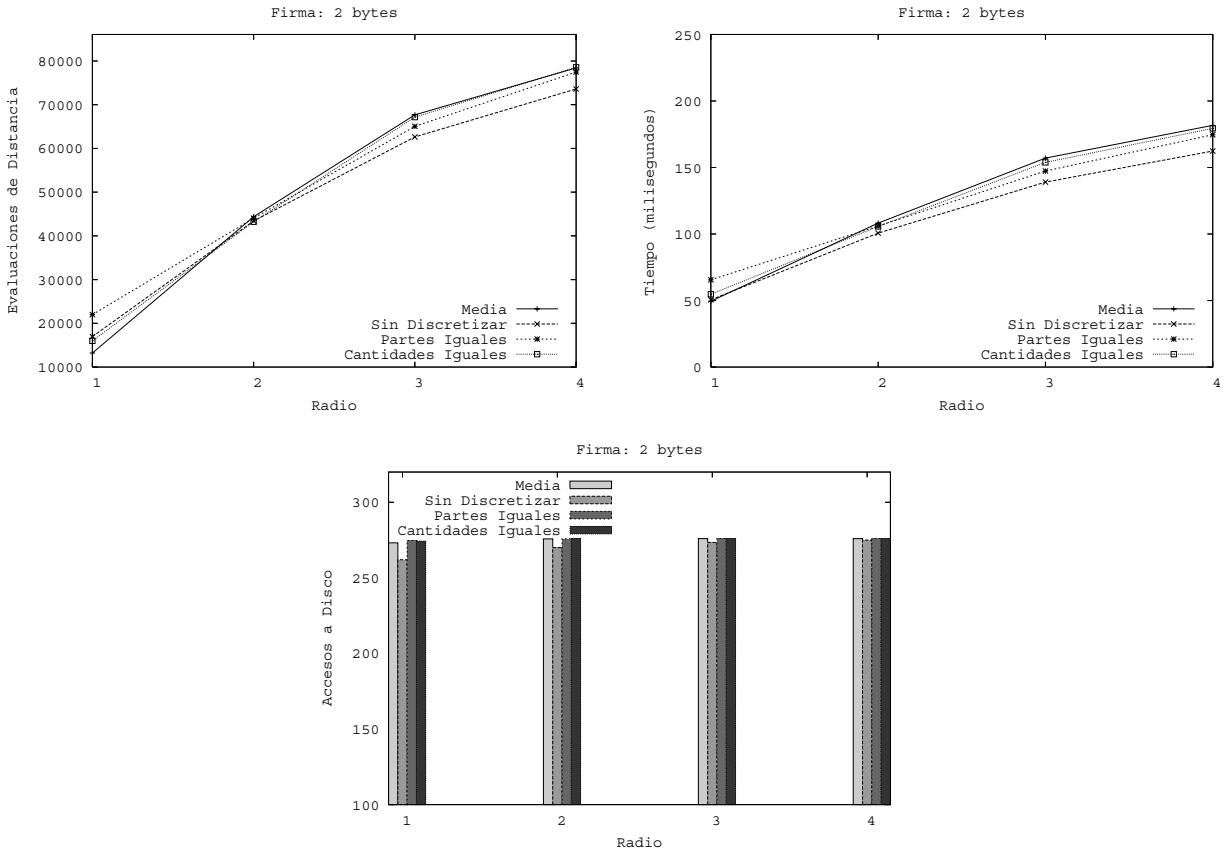

Figura 4: *Diccionario Español, tamaño de firma 2 bytes*

son las más competitivas. Con respecto a la cantidad de accesos a disco, puede observarse que en todos los casos se realizan una cantidad similar de accesos. La mayor diferencia se da en búsquedas de radio 1 siendo en este caso PLCS sin discretizar la que realiza menor cantidad de accesos a disco.

Finalmente, en la figura 5 se grafican los resultados para firmas de 4 bytes. En este caso, PLCS sin discretizar es la más competitiva logrando superar a PLCS con *media* para búsquedas de radio 1. Cabe destacar que para lograr este resultado, PLCS sin discretizar debió cuadruplicar el espacio pasando de firmas de 1 byte a firmas de 4 bytes.

Un punto importante para señalar es el efecto del tamaño de firma sobre la cantidad de accesos a disco. Aumentar el tamaño de firma beneficia a los índices porque permite aumentar la cantidad de pivotes lo que mejora el el poder de filtrado. Pero aumentar el tamaño de firma también provoca que los índices sean más grandes y se necesite una mayor cantidad de páginas para contenerlos. Dado que una búsqueda necesita acceder a todos los índices, aumentar el tamaño de firma tendrá como consecuencia un incremento en la cantidad de accesos a disco. Por ejemplo, para búsquedas de radio 1 y firmas de 1 byte las técnicas planteadas utilizan alrededor de 250 accesos a disco; para búsquedas de radio 1 y firmas de 4 bytes utilizan alrededor de 300 accesos a disco. La figura 6 ilustra la observación anterior para búsquedas de radio 1 y 3. Esta diferencia en cantidad de accesos a disco, se ve amortizada por la reducción en cantidad de evaluaciones de distancia lo que provoca finalmente menores tiempo de búsqueda.

Las observaciones hechas en el párrafo anterior nos indican que no se puede aumentar indiscriminadamente el tamaño de firma porque, aún cuando el índice de cada parte entre en memoria principal, en algún punto la cantidad de accesos a disco pesará más que los beneficios obtenidos por la reducción en cantidad de evaluaciones de distancias y perjudicaremos los tiempos de búsqueda.

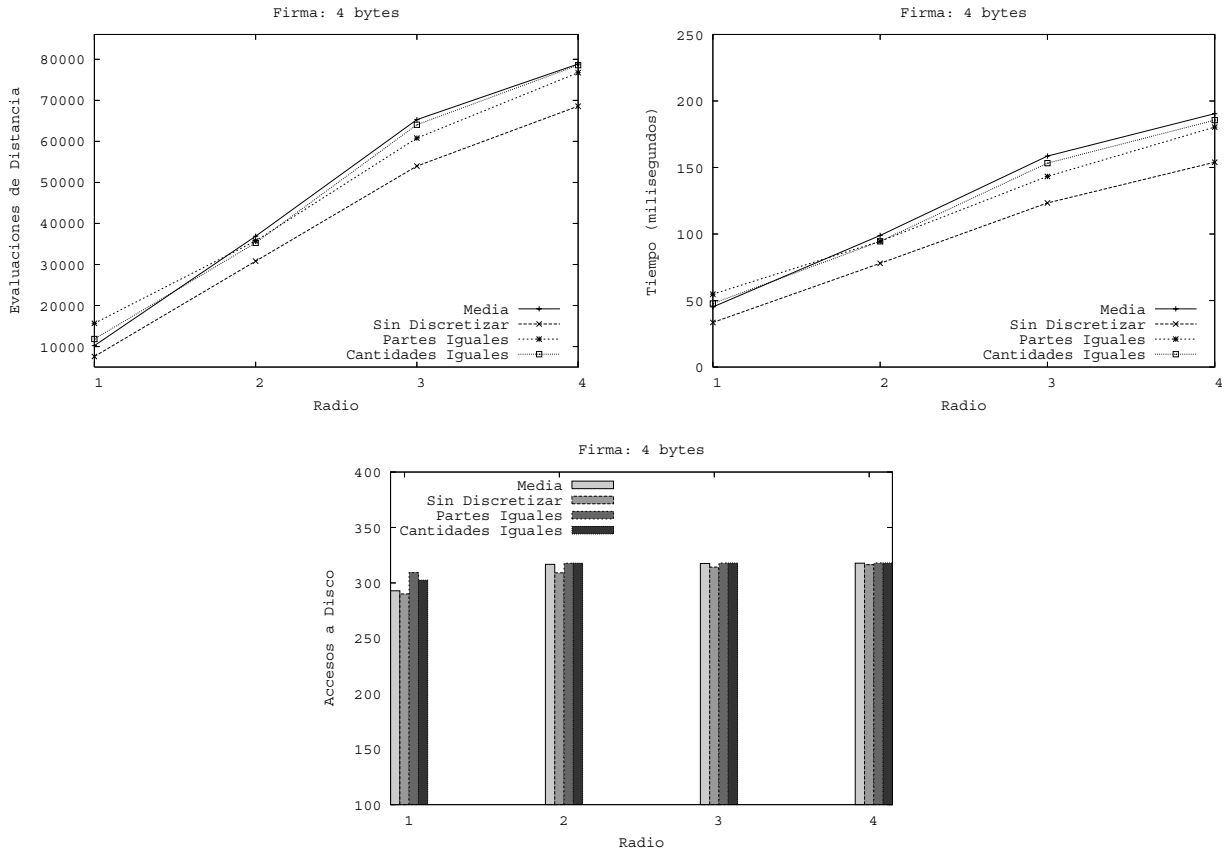

Figura 5: *Diccionario Español, tamaño de firma 4 bytes*

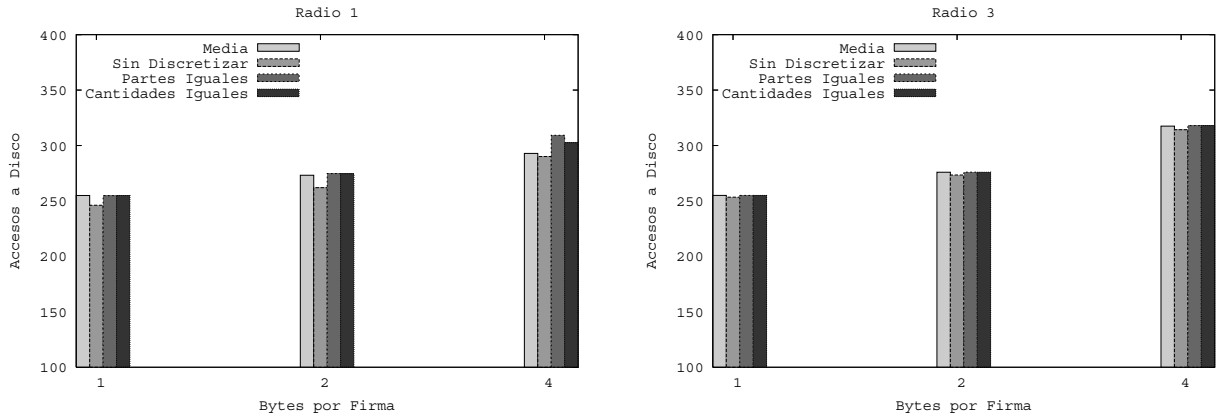

Figura 6: *Diccionario Español, efecto del tamaño de firma sobre la cantidad de accesos a disco*

Los resultados sobre los restantes diccionarios fueron similares y, en consecuencia, se obtienen las mismas conclusiones que las detalladas para el diccionario Español. Las figuras 7 y 8 muestran los resultados para el diccionario Inglés y el dicionario Francés con tamaño de firma de 1 byte.

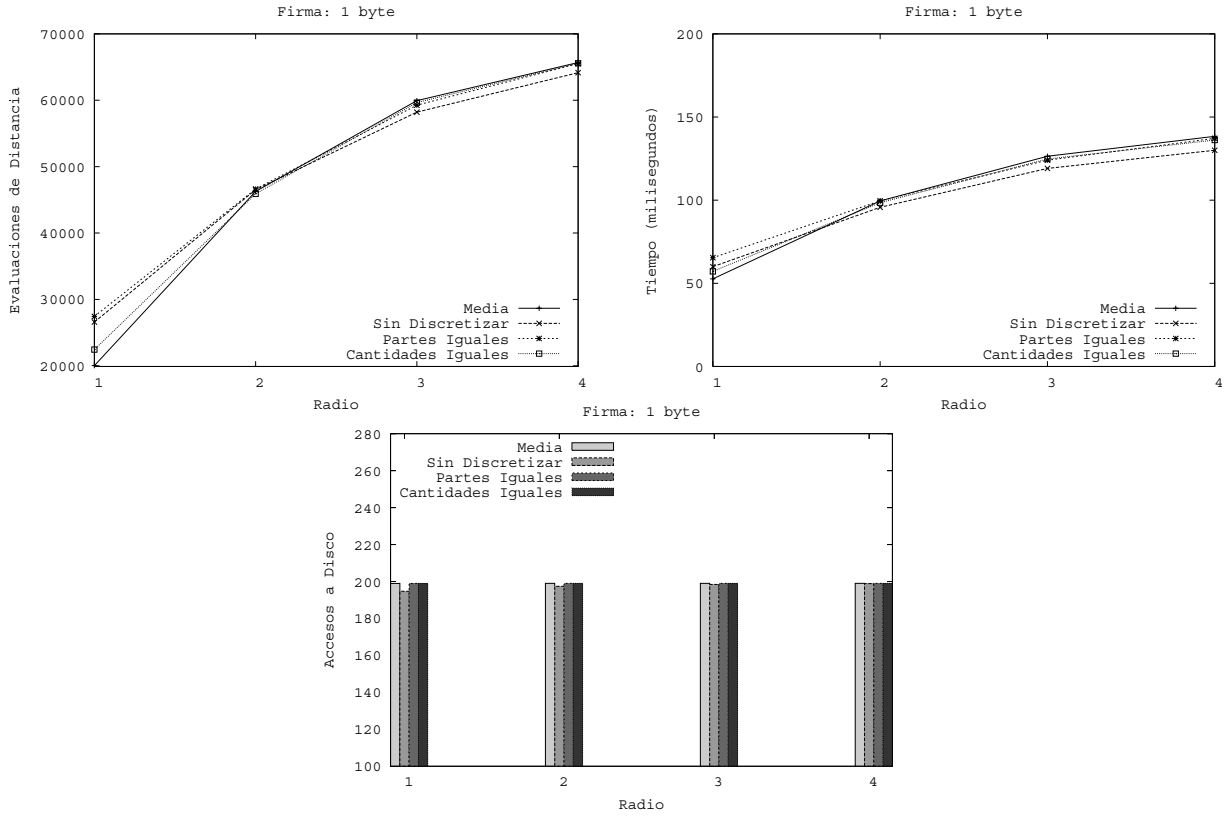

Figura 7: *Diccionario Inglés, tamaño de firma 1 byte*

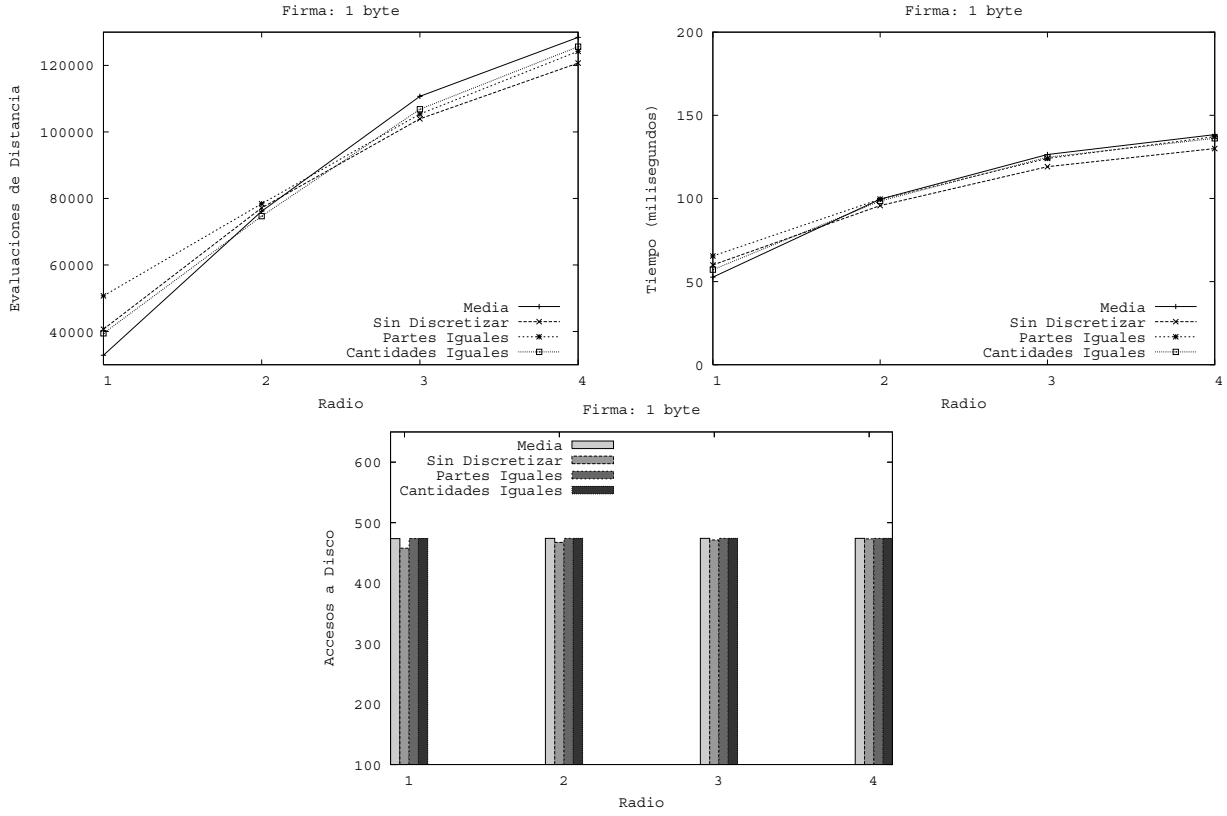

Figura 8: *Diccionario Francés, tamaño de firma 1 byte*

5. Conclusiones y Trabajo Futuro

En este trabajo combinamos el método de paginado PLCS con los métodos de discretización *media, partes iguales y cantidades iguales* con el objetivo de obtener una implementación del FQTrie que sea eficiente en términos de las tres componentes que afectan el tiempo de resolución de una consulta: cantidad de evaluaciones de la función de distancia d, cantidad de accesos a disco y tiempo extra de CPU.

La técnica PLCS con *media* resultó ser la más competitiva para búsquedas de alta selectividad y tamaños de firmas pequeños (1 y 2 bytes). La técnica PLCS sin discretizar debió cuadruplicar el espacio, usando firmas de 4 bytes, para lograr superar a PLCS con *media* en búsquedas de alta selectividad. Si se consideran búsquedas de baja selectividad PLCS sin discretizar resulta ser la opción más conveniente.

En cuanto al trabajo futuro nos proponemos estudiar en detalle las causas por las cuales PLCS con *media* no es competitivo en búsquedas de baja selectividad y solucionar dichos problemas. Además, estamos analizando métodos de paginado que puedan ser usados sobre espacios métricos cuyos objetos ocupan más de una página de disco.

Referencias

[1] E. Chávez and K. Figueroa. Faster proximity searching in metric data. In *Proceedings of MICAI 2004. LNCS 2972, Springer*, Cd. de México, México, 2004.

[2] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

[3] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[4] E. Chávez, N. Herrera, C. Ruano, and A. Villegas. Funciones de discretización basadas en histogramas de distancia. In *Actas de la Conferencia Latinoamericana de Informática (CLEI'06)*, Santiago, Chile, 2006.

[5] G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1991.

[6] A. Villegas, E. Chávez, and N. Herrera. Métodos de paginación para índices métricos basados en pivotes. In *Actas del X Congreso Argentino de Ciencias de la Computación (CACIC'04)*, pages 306–316, Buenos Aires, Argentina, 2004.