Discretización binaria para el FQTrie *

Carina Mabel Ruano
Dpto de Informática
Univ. Nacional de San Luis
Argentina
cmruano@unsl.edu.ar

Edgar Chávez
Escuela de Ciencias Físico-Matemáticas
Universidad Michoacana
Morelia - México
elchavez@fismat.umich.mx

Norma Edith Herrera
Dpto de Informática
Univ. Nacional de San Luis
Argentina
nherrera@unsl.edu.ar

Resumen

Para el problema de consultas de proximidad en espacios métricos se ha determinado experimentalmente que el índice que mejor desempeño tiene es el Trie de Consulta Fija (FQTrie por sus siglas en inglés). La eficiencia del FQTrie depende fuertemente del tipo de discretización y de la *calidad* de los pivotes empleados.

En este trabajo atacamos el problema de la discretización. Presentamos varias alternativas de funciones de discretización, y mostramos un método que utiliza sólo un bit por pivote (la cantidad mas baja posible de memoria) y que tiene una eficiencia muy alta.

Las discretizaciones presentadas mejoran notablemente el desempeño del FQTrie en condiciones de igualdad de memoria sin utilizar discretización.

Palabras claves: Bases de Datos, Espacios Métricos, Funciones de Discretización, Pivotes.

1. Introducción

El concepto de *búsquedas por similitud* o *por proximidad*, es decir buscar elementos de una base de datos que sean similares o cercanos a uno dado, aparece en diversas áreas de computación, tales como reconocimiento de voz, reconocimiento de imágenes, compresión de texto, biología computacional, inteligencia artificial, minería de datos, entre otras.

En [5] se muestra que el problema se puede expresar como sigue: dado un conjunto de objetos $\mathcal{X}$ y una función de distancia d definida entre ellos que mide cuan diferentes son, el objetivo es recuperar todos aquellos elementos que sean similares a uno dado. Esta función d cumple con las propiedades características de una función de distancia: *positividad* ($d(x, y) \geq 0$), *simetría* ($d(x, y) = d(y, x)$) y *desigualdad triangular* ($d(x, y) \leq d(x, z) + d(z, y)$). El par $(\mathcal{X}, d)$ se denomina *espacio métrico*. La base de datos será un subconjunto finito $\mathcal{U} \subseteq \mathcal{X}$ de cardinalidad n.

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

*Este trabajo ha sido parcialmente subvencionado por CYTED VII.19 RIBIDI Project, por el proyecto CONACyT 36911A y por el proyecto 22/F314 - UNSL

Las búsquedas por similitud pueden ser resueltas trivialmente con una complejidad $O(n)$. Para evitar esta situación se preprocesa la base de datos $\mathcal{U}$ usando un algoritmo, al que denominamos *algoritmo de indización*, que permite construir una estructura de datos o índice diseñada para ahorrar cómputos al momento de la búsqueda.

El tiempo total T necesario para resolver una búsqueda puede calcularse como: *T= #evaluaciones de* $d\times$ *complejidad(d) + tiempo extra de CPU + tiempo de I/O* . En muchas aplicaciones la evaluación de la función d es tan costosa que las demás componentes de la fórmula anterior pueden ser despreciadas. Éste es el modelo usado en la mayoría de los trabajos de investigación hechos en esta temática. Sin embargo, hay que prestar especial atención al tiempo extra de *CPU*, dado que reducir este tiempo produce que en la práctica la búsqueda sea más rápida, aún cuando estemos realizando la misma cantidad de evaluaciones de la función d. De igual manera, el tiempo de *I/O* puede jugar un papel importante en algunas aplicaciones.

Básicamente existen dos enfoques para el diseño de algoritmos de indización en espacios métricos: uno basado en particiones compactas y otro basado en pivotes [5]. Nuestro trabajo se ha centrado en los algoritmos basados en pivotes.

La idea subyacente de los algoritmos de indización basados en pivotes es la siguiente. Se seleccionan k pivotes $\{p_1, p_2, \ldots, p_k\}$, y se le asigna a cada elemento a de la base de datos, el vector o firma $\delta(a) = (d(a, p_1), d(a, p_2), \ldots, d(a, p_k))$. 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 pivote 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 se comparan directamente con q para determinar si forman o no parte de la respuesta.

La familia de estructuras *FQ* (FQT [2], FHQT [2, 1], FQA [4], FQTrie [3]) forman parte de las estructuras basadas en pivotes; cada una de ellas fue presentada como una mejora de la anterior. Por esto, el punto de partida de este trabajo es el *Fixed Queries Trie (FQTrie)* [3]. El objetivo es lograr una implementación eficiente no sólo en términos de cantidad de evaluaciones de la función de distancia d, sino también en tiempo extra de *CPU*. Utilizamos la técnica de *Tablas Lookup* en la implementación, concentrándonos en la definición de buenas funciones de discretización.

El trabajo está organizado de la siguiente manera. Las secciones 2 y 3 están dedicadas a la explicación de las *Tablas Lookup* y del *FQTrie* respectivamente. En la sección 4 presentamos nuestra propuesta de funciones de discretización. En la sección 5 realizamos la evaluación experimental de las mismas, en donde mostramos la eficiencia de las reglas propuestas. Finalizamos en la sección 6 dando las conclusiones y el trabajo futuro.

2. Tablas Lookup

Las *Tablas Lookup*, propuestas en [3], han demostrado ser una buena opción para mejorar el desempeño del *Fixed Queries Array* [4] y de una búsqueda secuencial. Recordemos que, dada una búsqueda $(q, r)_d$, los elementos no relevantes para la query son aquellos a tales que para algún p_i se cumple que $\mid d(q, p_i) - d(a, p_i) \mid > r$. Esto significa que si $d(a, p_i)$ se codifica en b_i bits, debemos realizar operaciones de enmascaramiento y corrimiento para evaluar la condición anterior. Una tabla lookup es una estrategia de representación para la firma de una búsqueda que permite realizar comparaciones entre palabras de máquina completas en lugar de hacerlas por grupos de b_i bits (donde b_i es la cantidad de bits necesitados para codificar $d(a, p_i)$). Comenzaremos dando algunas definiciones.

- **Función o Regla de Discretización**: una regla de discretización es una función $\delta_p : \mathbb{R}^+ \times \mathbb{K} \rightarrow \{0, \ldots, 2^{b_p} - 1\}$ donde $\mathbb{K} = \{p_1, p_2, \ldots, p_k\}$ es el conjunto de pivotes.

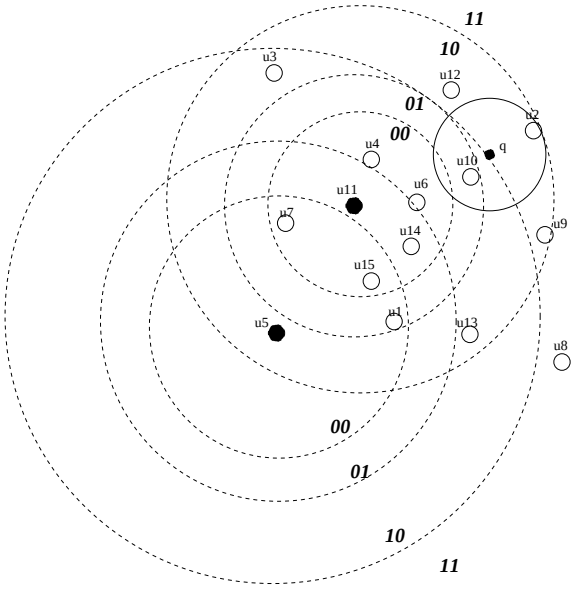

Figura 1: *Dos pivotes (u_{11} y u_5) y un esquema de discretización para el conjunto de puntos. También se muestra una posible búsqueda $(q, r)_d$.*

Esta regla asigna a cada número r un número natural de b_p bits; este número natural depende no sólo de r sino también del pivote p considerado.

- **Firma de un Elemento**: la función de firma de un elemento del espacio se define como $\delta^* : \mathcal{X} \rightarrow \{0,1\}^m$ con $\delta^*(x) = \delta_{p_1}(d(x,p_1))\ \delta_{p_2}(d(x,p_2)) \ldots \delta_{p_k}(d(x,p_k))$ donde $m = \sum_{i=1}^{k} b_{p_i}$
- **Discretización de Intervalos**: podemos extender la función δ_p a intervalos de la siguiente manera: $\delta_p([r_1, r_2]) = \{\delta_p(r)/r \in [r_1, r_2]\}$
- **Firma de una Búsqueda** $(q, r)_d$: dada una búsqueda por rango, la firma de la búsqueda es una función $\delta^* : \mathcal{X} \times \mathbb{R}^+ \rightarrow 2^{\{0,1\}^m}$ con:

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

donde $\bullet$ es la operación de concatenación. Lo que hacemos es una concatenación ordenada de los conjuntos que son firmas de los intervalos correspondientes a cada pivote. Con estas definiciones es fácil demostrar que si x satisface la búsqueda $(q, r)_d$, entonces $\delta^*(x) \in \delta^*((q, r)_d)$.

- **Lista de Candidatos**: la definición anterior nos dice cuál debería ser la firma de un elemento para que sea un candidato a formar parte de la respuesta de la query. Definimos entonces el conjunto de candidatos de la búsqueda $(q, r)_d$ como $[q, r]_d = \{x : \delta^*(x) \in \delta^*((q, r)_d)\}$

Veamos un ejemplo para clarificar las definiciones anteriores. Consideremos el espacio mostrado en la figura 1. Los círculos alrededor de los pivotes u_{11} y u_5, representan 4 zonas marcadas con 00, 01, 10 y 11, que han quedado determinadas por la regla de discretización usada. Esto significa que todos los elementos que están a distancia entre 0 y r_1 de u_{11} (siendo r_1 el diámetro de la primer circunferencia), tendrán 00 como firma para este pivote; los que tienen una distancia entre r_1 y r_2 se marcarán con 01, y así siguiendo. La firma de un elemento se obtiene concatenando las marcas que le corresponden a ese elemento, para ambos pivotes. Por ejemplo, $\delta^*(u_{14}) = 0001$ y $\delta^*(u_{10}) = 0110$.

En el momento de realizar una búsqueda $(q, r)_d$, la firma de esta búsqueda se obtiene concatenando las marcas de todas las regiones intersectadas por la query. En este caso, la bola de la query q intersecta las regiones de u_{11} marcadas con $00, 01$ y 10; y las regiones de u_5 marcadas con 10 y 11. Luego $\delta^*((q, r)_d) = \{00, 01, 10\} \bullet \{10, 11\} = \{0010, 0011, 0110, 0111, 1010, 1011\}$. Por lo tanto, la lista de candidatos para este ejemplo será $[q, r]_d = \{u_4, u_6, u_{10}, u_{12}, u_2, u_9\}$

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 $[q, r]_d$ es equivalente a realizar la intersección $\mathcal{U}^* \cap \delta^*((q, r)_d)$. El problema central de una búsqueda por rango es justamente computar esta lista de candidatos.

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 que es fácil de obtener. Supongamos que la palabra de máquina es de w bits, y que cada firma tiene un tamaño $m = tw$ bits. Entonces, podemos dividir cada firma $a_1 a_2 \dots a_m$ en t palabras de computadora, de la siguiente manera:

$$\underbrace{a_1\ a_2 \dots a_w}_{A_1}\ \underbrace{a_{w+1}\ a_{w+2} \dots a_{2w}}_{A_2}\ \dots\ \underbrace{a_{(t-1)w+1}\ a_{(t-1)w+2} \dots a_m}_{A_t}$$

Cada A_i es una palabra de computadora, a la que llamaremos ***i-ésima coordenada de la firma***.

Utilizando lo anterior, podemos representar $\delta^*((q, r)_d)$ como t conjuntos de coordenadas. Cada uno de esos t conjuntos, puede tener como máximo 2^w elementos. Una Tabla Lookup consiste en una representación binaria para cada uno de estos t conjuntos.

> **Definición**: una *Tabla Lookup* para $\delta^*((q, r)_d)$, es un arreglo L de t posiciones; donde cada posición es un vector de 2^w valores binarios. Denotamos con $L_j[i]$ a $L[j, i]$; luego $L_j[i] = 1$ si y sólo si i aparece en el j-ésimo conjunto de coordenadas.

Podemos decidir si $a_1 a_2 \dots a_m \in \delta^*((q, r)_d)$, evaluando la expresión $L_1[A_1] \wedge L_2[A_2] \wedge \dots \wedge L_t[A_t]$, con $A_i = a_{w(i-1)+1} \dots a_{iw}$, tarea que puede realizarse con a lo más t accesos a la Tabla Lookup. Usar esta técnica permite que las comparaciones se realicen a nivel de palabras de computadoras en lugar de hacerlo a nivel de b_p bits, lo que implica una reducción en el tiempo extra de CPU necesario para resolver una búsqueda.

3. Fixed Queries Trie (FQTrie)

Dado que estamos usando las firmas como cadenas, y queremos encontrar igualdad entre cadenas, entonces podemos usar alguna de las estructuras específicamente diseñadas para reconocimiento de patrones. En el FQtrie [3] se utiliza un *Árbol Digital o Trie* para indexar $\mathcal{U}^*$.

Un *Trie* es un árbol m-ario para búsqueda lexicográfica. En esta estructura, cada elemento se considera como una secuencia de caracteres sobre un alfabeto Σ; la cardinalidad del alfabeto determina la aridad del árbol, es decir $m = |\Sigma|$. Un nodo en un trie o es un nodo externo y contiene un elemento, o es un nodo interno y contiene m punteros a subtries. Dada una cadena, se usan los caracteres que la conforman para direccionar la búsqueda en el árbol. Estando en un nodo de nivel i, la selección del subtrie que le corresponde se realiza en función del i-ésimo caracter de la cadena. El nodo raíz usa el

primer caracter, los nodos hijos de la raíz usan el segundo caracter, y así sucesivamente. En un trie m-ario la búsqueda toma un tiempo que es proporcional a la longitud de la cadena, independientemente de cual sea el tamaño de la base de datos.

En nuestro caso, lo que representaremos sobre un trie es el conjunto de cadenas $\mathcal{U}^*$. Este trie será usado junto con la Tabla Lookup de $(q, r)_d$, lo que produce una pequeña modificación en la búsqueda. Estando en el nivel i, en lugar de seguir aquella rama que concuerda con el i-ésimo caracter de la cadena, seguimos un nodo si alguna coordenada en el conjunto concuerda con el rótulo del nodo; es decir, seguimos un nodo si la Tabla Lookup es $1(true)$ para el correspondiente nodo y nivel.

4. Funciones de Discretización

Recordemos que una función de discretización clasifica los objetos con respecto a la cercanía a un pivote p_i particular; si ese pivote usa una cantidad b_i de bits, entonces la función de discretización genera 2^{b_i} particiones del espacio que son enumeradas con valores del conjunto $\{0, \ldots, 2^{b_i} - 1\}$. Luego $\delta(d(p_i, o)) = j$ significa que el objeto o pertenece a la partición j del pivote p_i.

La función de discretización usada influye tanto en el espacio requerido por el índice (¿cuántos bits usamos por pivote?) como en el tiempo requerido para resolver una búsqueda (¿qué tan buenas son las particiones generadas?). Para una cantidad b_i de bits se pueden definir un número finito de reglas de discretización, algunas de las cuales producirán buenos filtrados y otras, posiblemente, no filtrarán en absoluto. La definición de la función de discretización no es esencial para la correctitud del método de filtrado, pero sí para su eficiencia.

En [4] se introducen dos funciones de discretización, a saber:

• **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.

Como ya dijimos, el objetivo de este trabajo ha sido el diseño de buenas funciones de discretización, buenas tanto en tiempo como en espacio; esto significa lograr un buen filtrado usando la menor cantidad de bits posible por pivote. Por esta razón comenzamos diseñando funciones de discretización que usan sólo un bit por pivote, y luego extendimos estas ideas para usar dos bits por pivote. Explicamos a continuación las funciones diseñadas.

4.1. Media δ_{μ_p}

La característica más importante de un espacio métrico es cómo se distribuyen los datos. Descubrir la estructura subyacente es sumamente útil en el diseño de algoritmos de indización. Una forma de visualizar la distribución de los datos en el espacio es por medio de los histogramas de distancias [5].

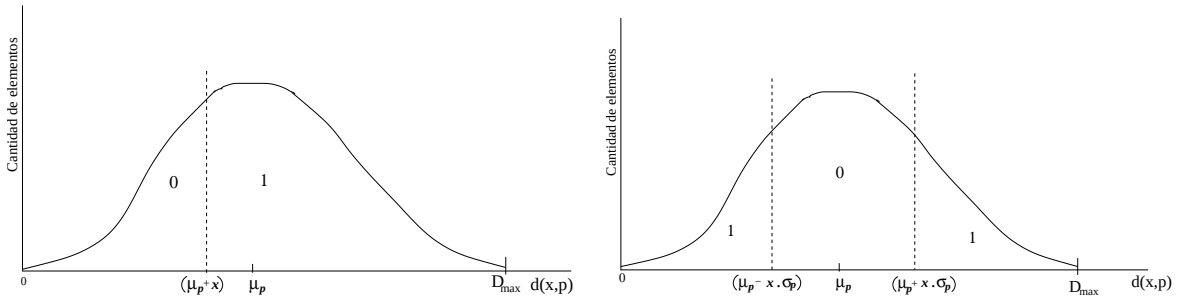

Figura 2: *Partición provocada por la función de discretización* δ_{μ_p} *con* $x < 0$ *(izquierda) y* $\delta_{(\mu_p,\sigma_p)}$ *(derecha)*

El histograma local de un punto p es la distribución de distancias de p a los elementos $u \in \mathcal{U}$. Este histograma permite visualizar la distribución de los elementos del espacio métrico respecto del punto p. La figura 2 (izquierda) muestra un ejemplo del histograma local de un punto p. El eje x representa los distintos valores para $d(p, x)$ y el eje y representa la cantidad de elementos del espacio que se encuentra a una determinada distancia de p.

La técnica δ_{μ_p} divide el histograma local de un pivote p en dos partes, utilizando la media μ_p del histograma para calcular el límite divisor. Luego se 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.

El límite divisor se calcula como $\mu_p + x$ donde μ_p es la media del histograma local de p y x es un número entero. Cuando $x = 0$ el histograma se divide justamente en el valor indicado por la media.

Formalmente, la función de discretización se define como:

$$\delta_{\mu_p}(d) = \begin{cases} 0 & \text{si } d < \mu_p + x \\ 1 & \text{si } d \geq \mu_p + x \end{cases}$$

La figura 2 (izquierda) muestra la partición provocada por esta técnica, indicando las zonas que serán mapeadas en 0 y las que serán mapeadas en 1 para un caso con $x < 0$.

4.2. Banda_Desviación $\delta_{(\mu_p,\sigma_p)}$

Esta técnica divide el histograma de un pivote p en tres partes, utilizando la media μ_p y la desviación estándar σ_p del histograma de p para determinar los límites divisores. El objetivo es separar la parte más alta del histograma de las más bajas, asignándole 0 a los valores incluidos en la banda central de histograma, y 1 a los valores que se encuentran en las zonas laterales.

La idea es tomar como banda central la zona correspondiente al intervalo $[\mu_p - \sigma_p, \ldots, \mu_p + \sigma_p]$. Como esta banda podría resultar demasiado ancha o angosta (dependiendo de la forma del histograma de p), permitimos variar el tamaño de la banda central en un factor x (ver figura 2, derecha).

Formalmente, la función $\delta_{(\mu_p,\sigma_p)}$ se define como:

$$\delta_{(\mu_p,\sigma_p)}(d) = \begin{cases} 1 & \text{si } d \in [0, \ldots, \mu_p - x\sigma_p) \text{ ó si } d \in (\mu_p + x\sigma_p\ , \ldots, D_{max}) \\ 0 & \text{si } d \in [\mu_p - x\sigma_p\ , \ldots, \mu_p + x\sigma_p] \end{cases}$$

El valor D_{max} representa la mayor distancia que hay de p a algún elemento del espacio.

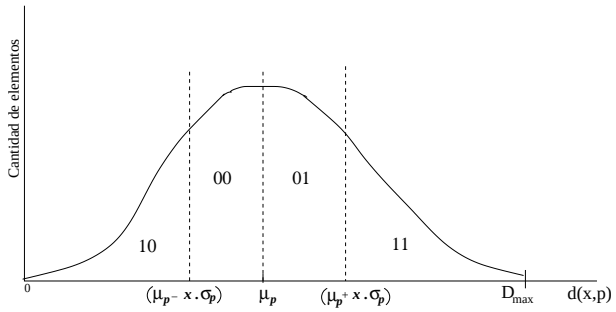

Figura 3: *Partición del histograma local de p generada por la función $\delta 2_{(\mu_p,\sigma_p)}$*

4.3. Banda_Valor $\delta_{(\mu_p,v)}$

Ésta es una variación de la técnica anterior. En este caso, en lugar de usar σ_p para delimitar la banda central, se utiliza un valor fijo v (el mismo para todos los pivotes):

$$\delta_{(\mu_p,valor)}(d) = \begin{cases} 1 & \text{si } d \in [0,\ldots,\mu_p - v) \text{ ó si } d \in (\mu_p + v,\ldots,D_{max}] \\ 0 & \text{si } d \in [\mu_p - v,\ldots,\mu_p + v] \end{cases}$$

4.4. Una extensión a dos bits: $\delta 2_{(\mu_p,\sigma_p)}$

Esta técnica es una extensión de $\delta_{(\mu_p,\sigma_p)}$ que permite dividir el histograma en cuatro partes. Los límites de estas partes se definen nuevamente en función de la media μ_p y la desviación estándar σ_p del histograma de p. Definimos esta función de discretización de la siguiente manera:

$$\delta 2_{(\mu_p,\sigma_p)}(d) = \begin{cases} 2 & \text{si } d \in [0,\ldots,\mu_p - x\sigma_p) \\ 0 & \text{si } d \in [\mu_p - x\sigma_p\ ,\ldots,\mu_p) \\ 1 & \text{si } d \in [\mu_p,\ldots,\mu_p + x\sigma_p) \\ 3 & \text{si } d \in [\mu_p + x\sigma_p,\ldots,D_{max}] \end{cases}$$

Nuevamente x es un número entero que permite variar el ancho de las bandas. La figura 3 muestra gráficamente la partición provocada por esta función.

5. Resultados Experimentales

La evaluación de las funciones de discretización presentadas en la sección anterior 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 palabras que hay que agregar, cambiar y/o eliminar a una palabra para obtener otra.

En una primera etapa intentamos identificar el valor óptimo de los límites divisores para cada una de las funciones. Para esto, usamos un diccionario español de $86{,}061$ palabras, que se indizó con 16 pivotes elegidos aleatoriamente. Sobre este 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. Esto nos permitió identificar no sólo los límites divisores óptimos, sino también la discretización que tiene el mejor desempeño.

Luego, la discretización con mejor desempeño fue evaluada con otros diccionarios (Francés, Alemán, Inglés e Italiano), variando además la cantidad de pivotes en $8, 16, 32, 40$ y 48.

Finalmente comparamos nuestros resultados con los que se obtienen si se usan las funciones propuestas en [4] (partes iguales y cantidad iguales) y con los resultados del caso trivial (es decir sin discretizar).

Por cuestiones de espacio en las siguientes secciones sólo mostramos las gráficas que consideramos más representativas.

5.1. Determinación de límites divisores

En esta etapa los límites divisores de cada función fueron establecidos de la siguiente manera:

- Para la función Media δ_{μ_p} se particionó usando para x los enteros en el rango $[-8, \ldots, 8]$.
- Para la función Banda_Desviación $\delta_{(\mu_p,\sigma_p)}$ se particionó con $x = 0{,}25, 0{,}50, 0{,}75, 1, 2$ y 3.
- Para la función Banda_Valor $\delta_{(\mu_p,v)}$ se particionó con $v = 0{,}20, 0{,}50, 1, 1{,}50, 2, 3, 4$ y 5.
- Para la función con 2 bits $\delta 2_{(\mu_p,\sigma_p)}$ se particionó sólo con $x = 1$.

La figura 4 muestra los promedios sobre las 500 búsquedas realizadas usando la discretización δ_{μ_p}. Se han graficado la cardinalidad de la lista de candidatos (izquierda) y los tiempos de búsquedas (derecha) para los distintos radios, en función del desplazamiento del límite divisor. La cardinalidad de la lista de candidatos nos permite deducir la cantidad de evaluaciones de distancias necesarias para resolver las búsquedas.

La gráfica muestra que el menor valor, tanto en tiempo como en cantidad de comparaciones, para búsquedas de radio 1 se obtiene con $x = 0$. En este punto la función de discretización logra descartar el $90,7\,\%$ de los elementos de la base de datos con un tiempo medio de búsqueda de $0{,}018$ segundos aproximadamente. No sucede lo mismo para los demás radios; para el caso $r = 2$ el mínimo valor se encuentra en $x = -1$ descartando el $54{,}08\,\%$ de los elementos; para radio 3 y 4 el mejor valor se logra en $x = -2$ descartando el $25\,\%$ y $12{,}3\,\%$ de los elementos en cada caso respectivamente. En todos los casos los mejores resultados se alcanzan para valores de x entre -4 y 4. Además, siempre sucede que alejar el límite divisor de la media empeora los resultados.

Como las búsquedas se realizarán con distintos valores de r no podemos en el momento de indizar realizar una elección que dependa de r. Por esto seleccionamos como mejor variación de δ_{μ_p} a aquella que, si bien no obtiene el mejor desempeño en cada radio de búsqueda, se mantiene cerca del mínimo en todos ellos. En la figura 5 se grafican los resultados para $x = -4 \ldots 4$, en función de r. Aquí podemos observar que para $x = -1$ los valores obtenidos se mantienen cercano al óptimo en todos los radios de búsquedas, alejándonos en el peor caso sólo un $10\,\%$. Por lo tanto este será el valor que utilizaremos en la comparación global.

La figura 6 muestra la cardinalidad de la lista de candidatos y el tiempo de la técnica $\delta_{(\mu_p,\sigma_p)}$ para cada variación de x, en función de r. Nuevamente sucede que ninguna de las variaciones obtiene el mejor valor para todos los radios. Por lo tanto, usando el mismo criterio que en el caso anterior, seleccionamos la variación con $x = 0{,}75$ dado que obtiene resultados cercanos al óptimo para todos los radios de búsquedas. En los mejores casos se elimina el $73\,\%$ de elementos en $0{,}05$ segundos aproximadamente.

La figura 7 corresponde a los resultados obtenidos para la discretización $\delta_{(\mu_p,v)}$. Como nuevamente no existe una variación que sea óptima, seguimos con el criterio anterior y seleccionamos la variación con $v = 1{,}50$. Para el caso $r = 1$, este valor de v resulta ser el óptimo logrando descartar el $82,75\,\%$ de los elementos en $0{,}032$ segundos aproximadamente.

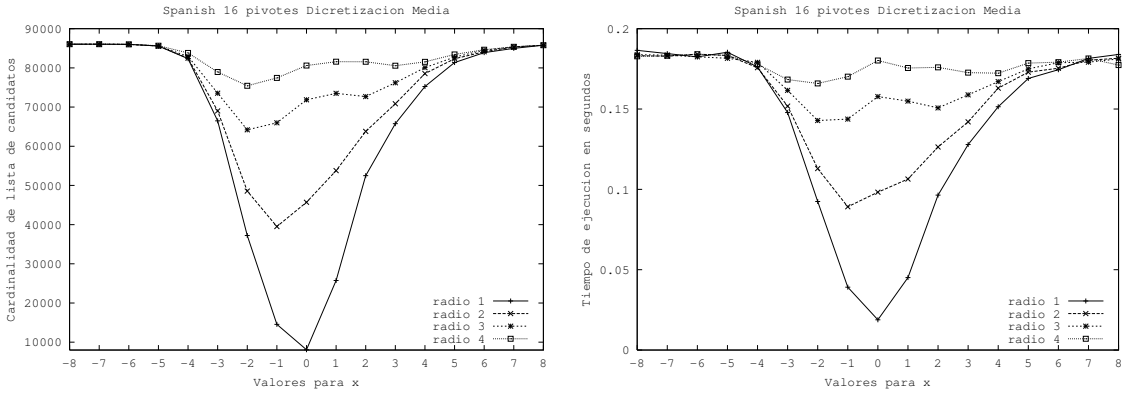

Figura 4: *Resultados para δ_{μ_p} con distintas variaciones de x.*

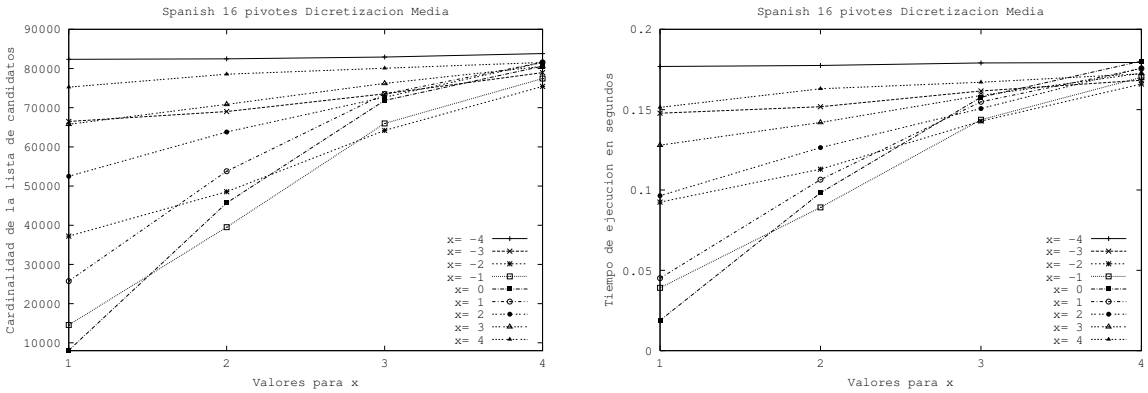

Figura 5: *Resultados para δ_{μ_p} con distintas variaciones de x, en función de r.*

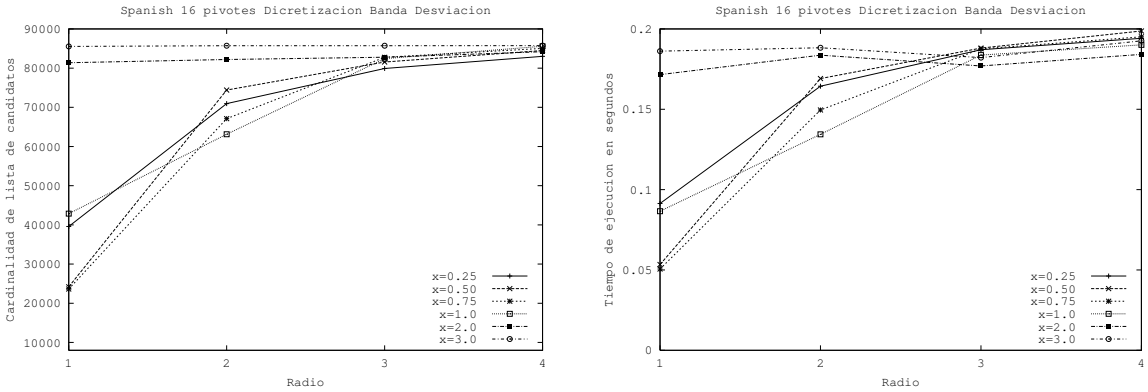

Figura 6: *Resultados para $\delta_{(\mu_p,\sigma_p)}$ con distintas variaciones de x, en función de r.*

Finalmente la figura 8 grafica la mejor variación de cada función de discretización de un bit junto con los resultados obtenidos con la función $\delta 2_{(\mu_p,\sigma_p)}$ (la única que utiliza dos bits). La discretización $\delta 2_{(\mu_p,\sigma_p)}$ gana sólo en búsquedas de radio 1, pero sin mejorar significativamente los resultado respecto de δ_{μ_p}. Es más, para lograr esto, $\delta 2_{(\mu_p,\sigma_p)}$ necesita el doble de espacio que las restantes.

Por esta razón no se experimentó con otros valores de x para $\delta 2_{(\mu_p,\sigma_p)}$. En esta gráfica también queda claro que la función δ_{μ_p} tiene un desempeño marcadamente superior respecto de las restantes, por lo que resultó seleccionada para los restantes experimentos.

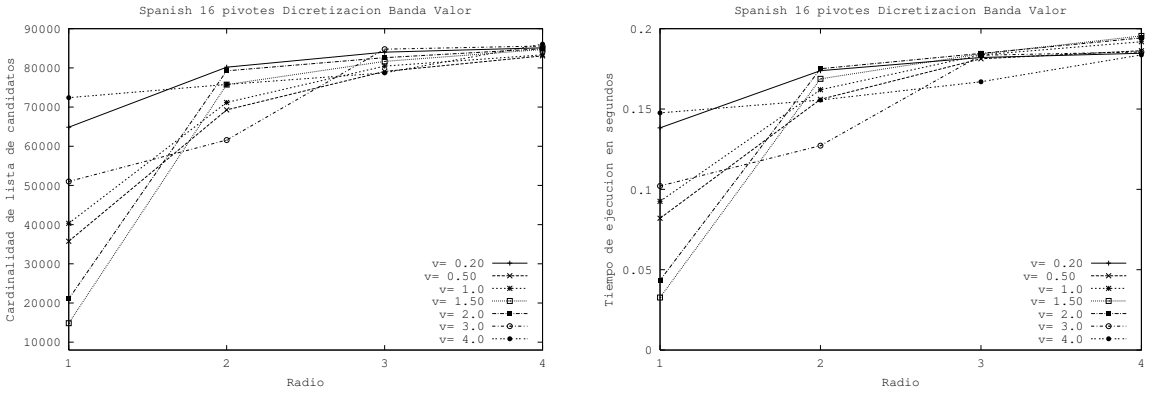

Figura 7: *Resultados para $\delta_{(\mu_p,v)}$ con distintas variaciones de v en función de r*

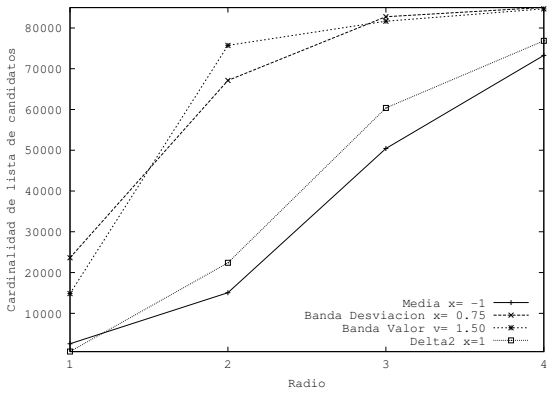

Figura 8: *Comparación global entre todas las discretizaciones*

La ventaja marcada por δ_{μ_p} se debe a que:si bien todas las funciones propuestas dividen el histograma en dos zonas, zona 0 y zona 1, las discretizaciones $\delta_{(\mu_p,v)}$ y $\delta_{(\mu_p,\sigma_p)}$ mantienen como zona 0 a la banda central del histograma la cual contiene la mayoría de los elementos y en consecuencia asignan la misma firma a mayor cantidad de objetos, perdiendo así selectividad en el proceso de búsqueda.

5.2. Variando cantidad de pivotes y diccionarios

El próximo paso fue variar la cantidad de pivotes para δ_{μ_p}. Los resultados obtenidos fueron los esperados: aumentar la cantidad de pivotes mejora la performance de la búsqueda. Esto se ilustra en la figura 9, en la que se han graficado los resultados para el diccionario Español (izquierda) y el diccionario Francés de 138.257 palabras (derecha). Un punto importante para destacar aquí es que agregar un pivote a la estructura implica agregar sólo un bit a la firma; esto significa que, con poco espacio adicional, podemos producir mejoras importantes en los resultados.

Con los restantes diccionarios se obtuvieron resultados similares. Por cuestiones de espacio no mostramos aquí esas gráficas.

5.3. Comparación con otras discretizaciones

La comparación de δ_{μ_p} con otras funciones de discretización se realizó fijando el espacio (es decir, el tamaño de la firma) y analizando cuáles son los mejores resultados que pueden lograr cada

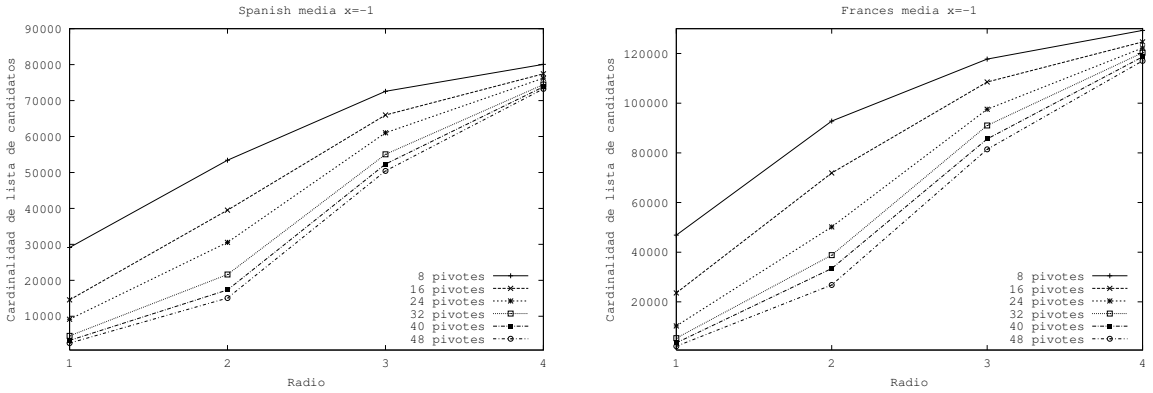

Figura 9: *Resultados de δ_{μ_p}, variando cantidades de pivotes, para los diccionarios Español (izq.) y Francés (der.)*

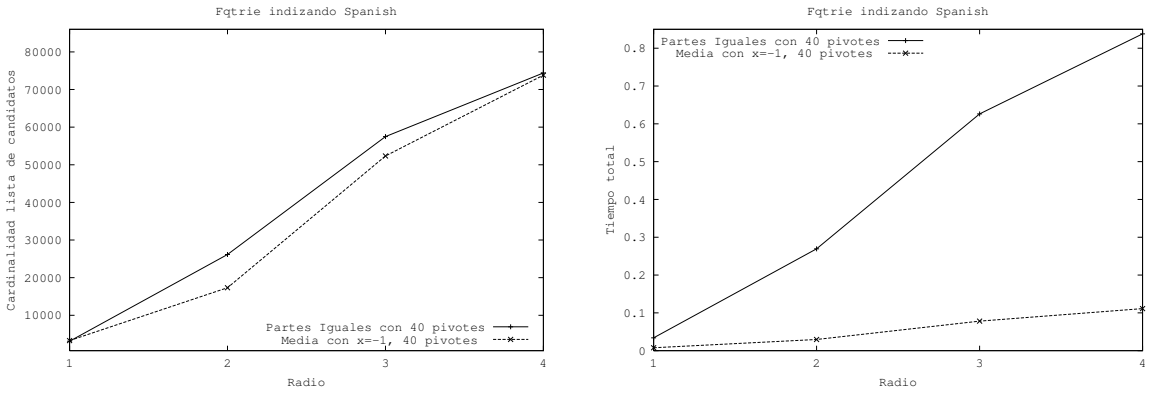

Figura 10: δ_{μ_p} *versus Partes Iguales*

una de ellas con ese espacio. Los experimentos se realizaron con tamaño de firma de 2 hasta 6 bytes.

Para la comparación con Cantidades Iguales y Partes Iguales los resultados mostraron que δ_{μ_p} las supera a ambas tanto en cardinalidad de la lista de candidatos como en tiempo. La discretización δ_{μ_p} logra una reducción de 50 % y 34 % en cuanto a evaluaciones de distancias y del 45 % y 86 % en cuanto al tiempo de ejecución respecto de Cantidades y Partes Iguales respectivamente. La figura 10 muestra, a modo ejemplo, los resultados de comparar δ_{μ_p} con Partes Iguales con tamaño de firma de 6 bytes. Notar que las mejoras son más significativas cuando se aumenta el radio de búsqueda r.

Al comparar δ_{μ_p} con el caso trivial (sin discretizar) se observó que, para la mayoría de los radios, δ_{μ_p} obtiene los mejores resultados en cuanto al tiempo requerido para una búsqueda pero no en cuanto a la cardinalidad de la lista de candidatos. Esto comprueba lo que mencionábamos al principio del artículo: reducir el tiempo extra de CPU produce que en la práctica la búsqueda sea más rápida, aún cuando estemos realizando la misma cantidad de evaluaciones de la función d. Se pudo observar también que aumentar el tamaño de firma no afecta las reducciones de tiempo logradas, es decir, δ_{μ_p} siempre gana en la misma proporción independientemente del tamaño de firma. Sin embargo este comportamiento es diferente cuando consideramos cardinalidad de lista de candidatos. La figura 11 ilustra el caso para tamaño de firma de 5 bytes.

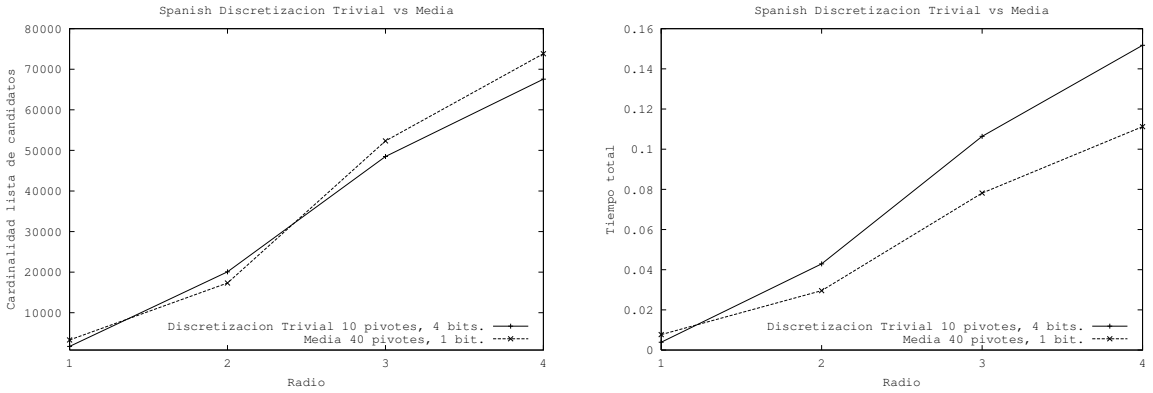

Figura 11: δ_{μ_p} *versus la solución sin discretizar.*

6. Conclusiones y Trabajo Futuro

En este trabajo presentamos cuatro alternativas de funciones de discretización. La función δ_{μ_p} con $x = -1$ mostró un desempeño marcadamente superior respecto de las restantes, logrando descartar el $80\,\%$ de los elementos de la base de datos, con un tiempo medio de búsqueda de $0{,}030$ segundos aproximadamente.

La discretización δ_{μ_p} supera considerablemente a las discretizaciones existentes, cantidades iguales y partes iguales, obteniendo una reducción aproximada del 50 % y 34 % en cuanto a evaluaciones de distancias y del 45 % y 86 % en cuanto al tiempo de ejecución respectivamente.

En cuanto a la solución sin discretizar, δ_{μ_p} la supera en tiempo de ejecución pero no en la cardinalidad de la lista de candidatos. Se pudo observar que aumentar el tamaño de firma no afecta las reducciones de tiempo logradas, pero sí la cardinalidad de la lista de candidatos.

Con respecto al trabajo futuro queremos investigar las condiciones que gobiernan la selección óptima de pivotes y su relación con las funciones de discretización. Estamos trabajando en integrar este trabajo con una versión en memoria secundaria del FQTrie que compite eficientemente con el M-tree (el único índice públicamente disponible, para usuario final).

Referencias

[1] R. Baeza-Yates. Searching: an algorithmic tour. In A. Kent and J. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 37, pages 331–359. Marcel Dekker Inc., 1997.

[2] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

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

[4] 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.

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