Funciones de Discretización para Indices Métricos Basados en Pivotes *

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

Norma Herrera, Carina Ruano, Ana Villegas
Departamento de Informática
Universidad Nacional de San Luis
Argentina
{nherrera, cmruano, anaville}@unsl.edu.ar

Resumen

El modelo de Espacios Métricos permite formalizar el concepto de búsqueda por similitud en bases de datos no tradicionales. El objetivo es construir índices que permitan reducir el tiempo necesario para resolver una búsqueda por similitud.

Uno de los enfoques para la construcción de índices es el usado por los algoritmos basados en pivotes. Dentro de los índices basados en pivotes de mejor desempeño, se encuentra 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 diseño de funciones de discretización para el FQTrie, cuando el mismo se utiliza para indizar espacios métricos con funciones de distancia continuas. Presentamos una nueva función de discretización δ_{ma} basada en los histogramas de distancias de los pivotes usados en la construcción del índice. Mostramos experimentalmente que δ_{ma} es altamente competitiva en los espacios considerados.

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

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

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 [7] 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 cuán 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, $(q, r)_d = \{u \in \mathcal{U} : d(q, u) \leq r\}$.

Las búsquedas por similitud pueden ser resueltas trivialmente con una complejidad de $O(n)$. 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 [7]. 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 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 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 [6], FQTrie [5]) forman parte de las estructuras basadas en pivotes; cada una de ellas fue presentada como una mejora de la anterior, siendo el FQTrie (*Fixed Queries Trie*) el de mejor desempeño. Por esta razón, en este trabajo nos enfocamos sobre *Fixed Queries Trie (FQTrie)* [5]. La eficiencia de este índice depende no sólo de la calidad de los pivotes seleccionados sino también de la función de discretización utilizada. Una buena función de discretización junto con la técnica de *Tablas Lookup* permiten que el FQTrie sea eficiente no sólo en términos de cantidad de evaluaciones de distancia sino también en tiempo extra de CPU.

En [4] se presenta la función de discretización *Media*(δ_{μ_p}) que, utilizando sólo un bit por pivote (el menor espacio posible), resulta competitiva sobre espacios métricos con funciones de distancia

discretas. En este trabajo abordamos el diseño de funciones de discretización que resulten eficientes para espacios métricos con funciones de distancia continua.

Este artículo está organizado de la siguiente manera. En las sección 2 describimos la técnica de Tablas Lookup y el índice FQTrie. Luego, en la sección 3 introducimos las funciones de discretización existentes, describiendo su comportamiento para espacios métricos con funciones de distancias discretas. En la sección 4 presentamos una nueva función de discretización para espacios con funciones de distancia continua y realizamos la evaluación experimental de la misma. Finalizamos en la sección 5 dando las conclusiones y el trabajo futuro.

2. Tablas Lookup y Fixed Queries Trie

Las *Tablas Lookup*, propuestas en [5], han demostrado ser una buena opción para mejorar el desempeño del *Fixed Queries Array* [6] 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. 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 $[q,r]_d = \{x : \delta^*(x) \in \delta^*((q,r)_d)\}$

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 finales en $\delta^*((q,r)_d)$ se obtienen como concatenación ordenada de firmas parciales 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 palabra de computadora en lugar de hacerlo a nivel de b_p bits, lo que permite reducir el tiempo extra de CPU necesario para resolver una búsqueda.

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 [5] se utiliza un *Árbol Digital o Trie* [8] 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.

3. Funciones de Discretización sobre Espacios Discretos

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} particio-

nes del espacio que son enumeradas con valores del conjunto $\{0, \ldots, 2^{b_i} - 1\}$. Luego $\delta(d(p_i, u)) = j$ significa que el objeto u pertenece a la partición j del pivote p_i.

Mientras mayor sea el valor b_i, más precisión tendremos; como caso extremo podríamos mantener la distancia exacta sin discretizar. Pero mientras el valor b_i aumenta menos pivotes pueden ser contenidos en la misma cantidad de espacio. Es por eso que necesitamos encontrar el balance entre el poder de filtrado y el espacio utilizado.

Resumiendo, 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 [6] se introducen dos funciones de discretización, a saber:

- **Partes Iguales (denotaremos con δ_{pi})** : 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 (denotaremos con δ_{ci})**: 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.

En [4] se presentan cuatro alternativas de funciones de discretización basadas en histogramas de distancias. De la evaluación experimental de las mismas, se concluye que la de mejor desempeño es δ_{μ_p} :

- **Media (denotaremos con δ_{μ_p})** : para discretizar $d(p, u)$, esta función utiliza el histograma local de p. Este histograma permite visualizar la distribución de los elementos del espacio métrico respecto del punto p. La figura 1 muestra un ejemplo del histograma local de un punto p; el eje x representa los distintos valores para $d(p, u)$ 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 como límite divisor $\mu_p + x$, donde μ_p es la media del histograma y x es un número entero. 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).

En [4] se muestra experimentalmente que la discretización δ_{μ_p} con $x = -1$ obtiene una alta eficiencia sobre espacios métricos con funciones de distancia discretas, logrando superar ampliamente a δ_{pi} y δ_{ci} . La eficiencia de esta función se debe en gran parte a que utiliza información sobre la distribución de los elementos del espacio para discretizar, dividiendo el histograma local de cada pivote en el punto de mayor concentración de elementos. Notar además que δ_{μ_p} utiliza sólo un bit por pivote, la menor cantidad de memoria posible.

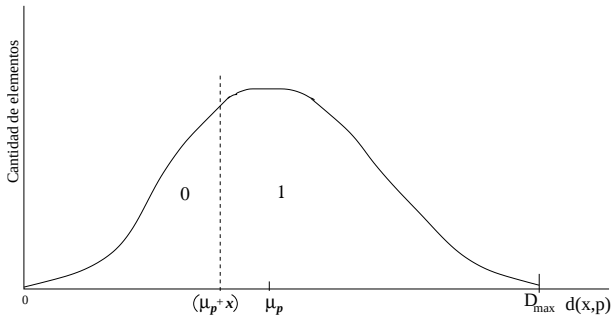

Figura 1: *Partición provocada por la función de discretización* δ_{μ_p} *con* $x < 0$.

4. Funciones de Discretización sobre Espacios Continuos

Tal como lo mencionáramos, la eficiencia lograda por δ_{μ_p} se debe en gran parte a que divide el histograma local de cada pivote en el punto de mayor concentración de elementos (la parte más alta del histograma). Esta función fue diseñada en base a un histograma en forma de campana donde la media se ubica cercana a ese punto. Sin embargo, esto no sucede con los histogramas de todos los espacios.

Analicemos por ejemplo el espacio métrico formado por documentos de texto con la función de distancia coseno [3]. En este modelo el espacio consta de una coordenada por término, y los documentos son vistos como vectores en este espacio de alta dimensionalidad. La distancia entre dos documentos se define como el ángulo formado por los vectores que representan a dichos documentos.

La figura 2 muestra ejemplos de histogramas locales para dos pivote elegidos aleatoriamente de la colección TREC-3 (http://trec.nist.gov) formada por 1265 documentos. Puede observase que estos histogramas no presentan una forma de campana; la mayor concentración de elementos se identifica en más de un punto y la media no se ubica cercana a ninguno ellos.

Para solucionar este inconveniente modificamos la función δ_{μ_p} de la siguiente manera: dado un pivote p, en lugar de dividir en la media μ_p de su histograma local, dividimos en el punto de mayor altura del histograma que representa la zona de mayor concentración de elementos. Luego, asignamos 0 a todos aquellos valores que se ubiquen a izquierda del límite divisor y 1 a los que se encuentren a derecha. Llamaremos a esta nueva función *máxima altura* y la denotaremos con δ_{ma}.

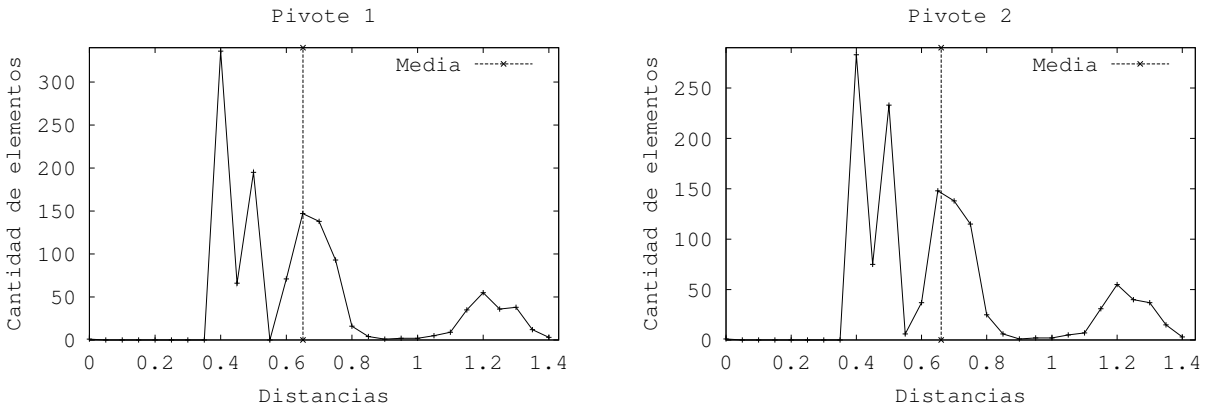

Figura 2: *Histogramas locales de pivotes elegidos aleatoriamente del espacio de documentos.*

Si denotamos con hl_p al histograma local del punto p, luego :

$$hl_p(i) = |\{x/d(x,p) = i\}|$$

Sea $k = \text{máx}_{x,y \in U}\{d(x,y)\}$, entonces podemos definir la función δ_{ma} de la siguiente manera:

$$\delta_{ma}(d) = \begin{cases} 0 & \text{si } d < \text{máx}_{1 \leq i \leq k}\{hl_p(i)\} \\ \\ 1 & \text{si } d \geq \text{máx}_{1 \leq i \leq k}\{hl_p(i)\} \end{cases}$$

4.1. Evaluación Experimental

4.1.1. Descripción de los experimentos

Los experimentos se ejecutaron sobre una computadora con 3Gb de memoria RAM, linux versión 2.4 con gcc versión 3.2.2.

El objetivo final de los experimentos fue establecer la eficiencia de la función de discretización propuesta δ_{ma} respecto de las funciones ya conocidas δ_{ci} , δ_{pi} y δ_{μ_p} . Se evaluaron dos aspectos: cardinalidad de la lista de candidatos y tiempo de búsqueda. La cardinalidad de la lista de candidatos nos permite deducir la cantidad de evaluaciones de distancias necesarias para resolver una búsqueda (cardinalidad de la lista de candidatos más cantidad de pivotes).

Utilizamos como espacio métrico la colección TREC-3 de 1265 documentos de texto, indizado con un FQTrie. De esta colección se eligieron al azar 300 documentos, los que conformaron el lote de prueba utilizado para todos los experimentos. Para cada documento de este lote, se realizaron búsquedas por rango utilizando como radios de búsqueda r los valores $0{,}371, 0{,}383, 0{,}387, 0{,}390, 0{,}392$ que recuperan respectivamente el $0{,}10\,\%, 0{,}50\,\%, 1\,\%, 1{,}50\,\%$ y $2\,\%$ de elementos del espacio. Todos los resultados que se muestran en este artículo fueron obtenidos promediando los valores alcanzados con cada uno de los 300 documentos del lote de prueba.

La evaluación experimental de las funciones de discretización se realizó en dos etapas. En la primer etapa fijamos el tamaño de firma y analizamos cuál es el mejor desempeño que puede lograr cada función de discretización con ese espacio. En la segunda etapa analizamos si variar el tamaño del espacio métrico o variar el tamaño de firma afecta las conclusiones obtenidas en la primer etapa.

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

4.1.2. Fijando el tamaño de firma

Para la indización se usó inicialmente un FQTrie con un tamaño de firma de 16 bits. Para este tamaño de firma las funciones δ_{μ_p} y δ_{ma} tienen como única posibilidad usar 16 pivotes de 1 bit cada uno. En cambio, δ_{pi} y δ_{ci} pueden realizar las siguientes combinaciones entre cantidad de pivotes y cantidad de bits por pivote: 16 pivotes de 1 bit, 8 pivotes de 2 bits, 4 pivotes de 4 bits, 2 pivotes de 8 bits ó 1 pivote de 16 bits. Nuestro primer paso fue analizar cuál de estas combinaciones es la que resulta más adecuada para las funciones δ_{pi} y δ_{ci} .

Dado que para el espacio métrico considerado 1 ó 2 pivotes resultan insuficientes, sólo se realizaron experimentos para las tres primeras combinaciones. Las gráficas 3 y 4 muestran los resultados obtenidos por las funciones δ_{pi} y δ_{ci} respectivamente. En ambos casos se muestra, de las 300 búsquedas realizadas, el promedio de la cardinalidad de la lista de candidatos (izquierda) y del tiempo de búsqueda (derecha). Como se puede observar, la eficiencia de las funciones aumenta a medida que aumentamos la cantidad de pivotes, aún cuando estamos reduciendo el espacio usado por cada pivote.

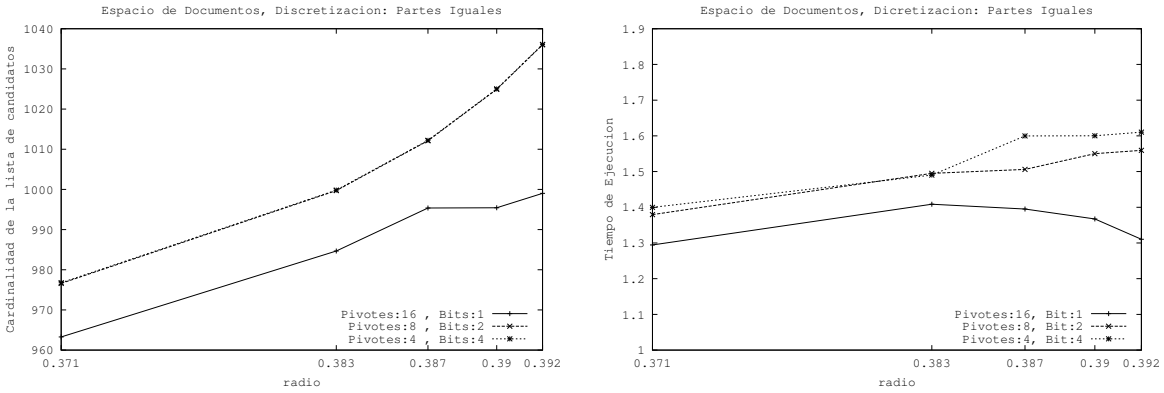

Figura 3: *Resultados para la función δ_{pi} con tamaño de firma de 16 bits. Cardinalidad de la lista de candidatos (izquierda) y tiempo de búsqueda (derecha).*

El mejor desempeño se logra justamente cuando utilizamos 16 pivotes de 1 bit. En consecuencia, ésta es la combinación elegida para realizar la comparación con δ_{μ_p} y δ_{ma} .

La figura 5 muestra los resultados para las cuatro funciones de discretización usando 16 bits como tamaño de firma. Nuevamente se ha graficado la cardinalidad de la lista de candidatos (izquierda) y el tiempo de búsqueda (derecha) para los distintos radios. Con respecto a la cantidad de comparaciones, se puede observar que la de mejor desempeño es δ_{ma}, logrando reducir en un $20\,\%$ la cardinalidad de la lista de candidatos respecto de δ_{ci} (la que le sigue en eficiencia) en este aspecto. Con respecto a los tiempos de búsquedas, la función δ_{ma} sigue siendo la que obtiene los mejores resultados logrando resolver las búsquedas en $0{,}25$ segundos aproximadamente, un $10\,\%$ menos que δ_{μ_p}, quien le sigue en eficiencia logrando ahora superar ampliamente δ_{ci} y δ_{pi} .

Algo importante para destacar es que, si bién δ_{μ_p} obtiene una lista de candidatos de mayor cardinalidad que δ_{ci} , en tiempo la supera ampliamente en eficiencia. Esto significa que, si bien δ_{μ_p} obtiene una lista de candidatos de mayor cantidad de elementos, realiza una menor cantidad de operaciones adicionales lo que reduce el tiempo extra de CPU y, en consecuencia, el tiempo total de búsqueda.

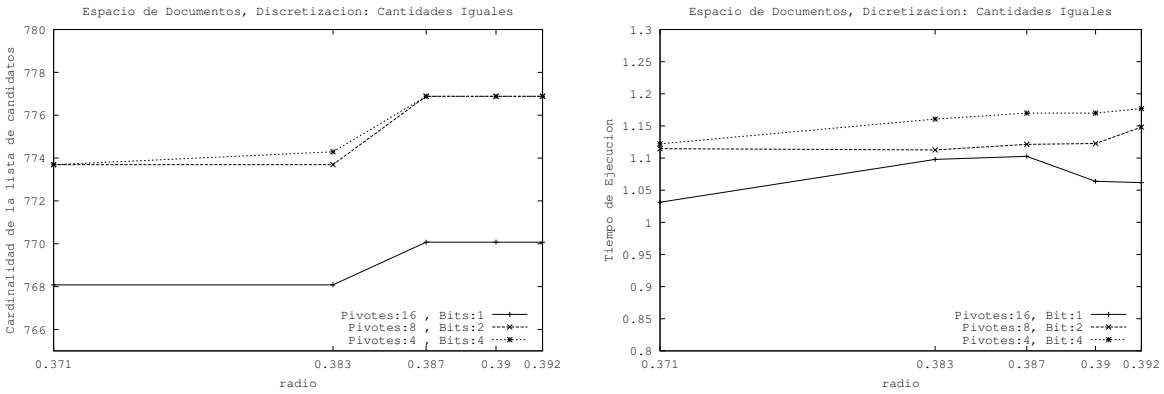

Figura 4: *Resultados para la función δ_{ci} con tamaño de firma de 16 bits. Cardinalidad de la lista de candidatos (izquierda) y tiempo de búsqueda (derecha).*

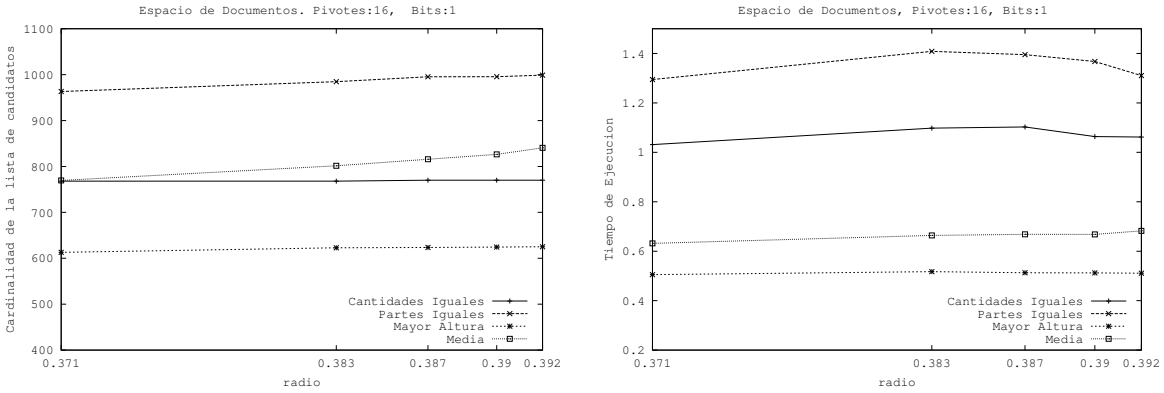

Figura 5: *Resultados para tamaño de firma de 16 bits. Cardinalidad de la lista de candidatos (izquierda) y tiempo de búsqueda (derecha) para distintos radios de búsqueda.*

4.1.3. Variando el tamaño del espacio y el tamaño de firma

Como ya lo mencionáramos, otro punto que nos interesaba evaluar era el efecto del tamaño del espacio sobre las distintas funciones. Para ello realizamos experimentos tomando el $50\,\%, 60\,\%, 70\,\%, 80\,\%$ y $90\,\%$ de los documentos de la colección para indizar y realizar el grupo de 300 búsquedas. Para las funciones δ_{pi} y δ_{ci} sólo se experimentó con la combinación que mostró mejor performance en la etapa anterior: 16 pivotes de 1 bit. En las figura 6, 7 y 8 mostramos, a modo de ejemplo, los resultados para el $50\,\%$ (632 documentos), el $70\,\%$ (885 documentos) y el $90\,\%$ (1138 documentos) respectivamente.

Respecto de la cardinalidad de la lista de candidatos, las funciones δ_{ma} , δ_{ci} y δ_{pi} son las que muestran un comportamiento más estable, siendo δ_{ma} siempre la más eficiente de todas. En cambio, la función δ_{μ_p} varía dependiendo del tamaño del espacio, logrando en algunos casos superar a δ_{ci} y δ_{pi} pero en otros no. Intuimos que la causa de este comportamiento es que la media del histograma se aleja o acerca del punto de mayor concentración en función del tamaño del espacio, lo que afecta su capacidad de filtrado.

Respecto de los tiempos de búsquedas, δ_{ma} nuevamente es la de mejor desempeño logrando resolver las búsquedas con tiempos que van desde los $0{,}2$ a $0{,}5$ segundos, dependiendo del tamaño del espacio. Como sucedía anteriormente, en algunos casos δ_{μ_p} pierde frente a δ_{ci} en cardinalidad de lista de candidatos pero gana en tiempo de búsqueda.

Esto queda aún más claro en la figura 9, en donde se ha graficado los resultados obtenidos para los

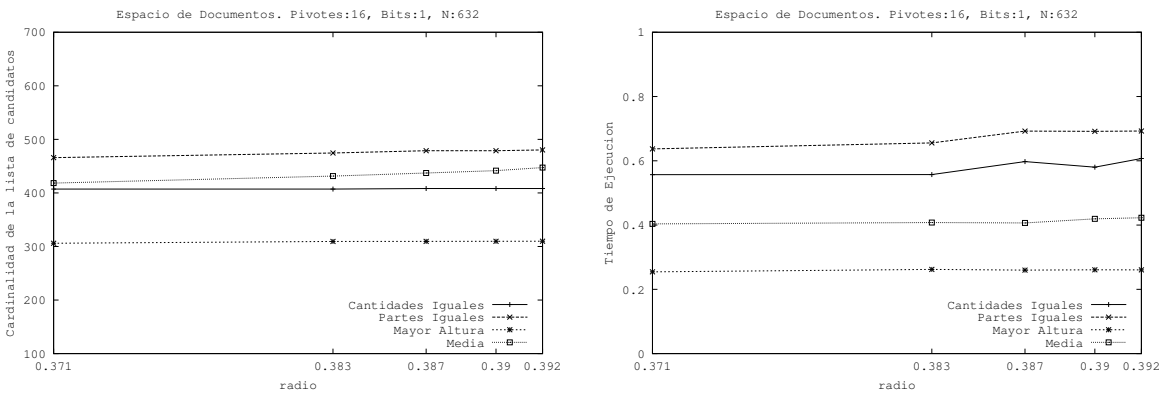

Figura 6: *Resultados para* 632 *documentos (*$50\,\%$ *de la colección de 1265 documentos).*

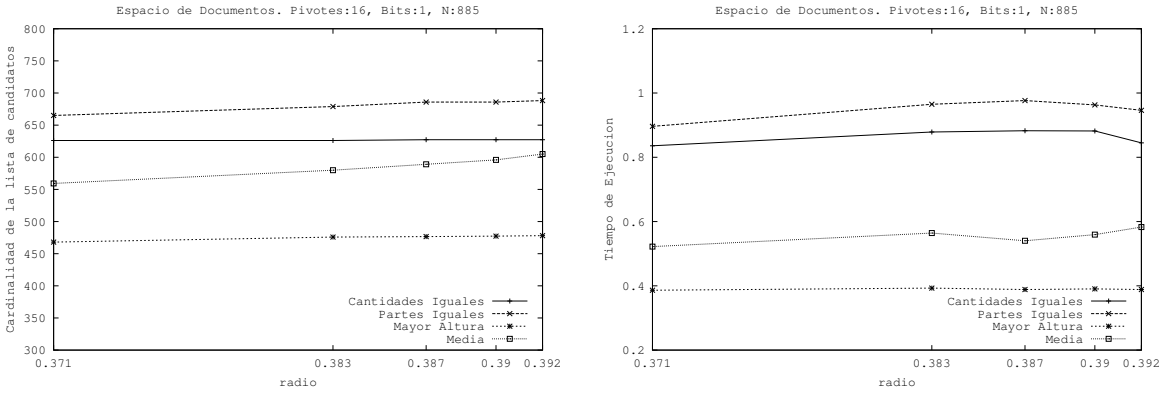

Figura 7: *Resultados para* 885 *documentos (*$70\,\%$ *de la colección de* 1265 *documentos).*

radios de búsquedas $0{,}371$ y $0{,}392$, en función del tamaño del espacio. Se observa claramente que δ_{ma} es la más competitiva independientemente del tamaño del espacio. También puede apreciarse cómo la cardinalidad de lista de candidatos obtenida por δ_{μ_p} se ve afectada tanto por el tamaño del espacio como por el radio de búsqueda.

El próximo paso fue establecer si incrementar el tamaño de firma varía el comportamiento de las funciones de discretización. Para ello repetimos los experimentos utilizando ahora como tamaño de firma 32 y 48 bits (ver figura 10). En el caso de las funciones δ_{pi} y δ_{ci} , basándonos en los resultados que obtuvimos con un tamaño de firma de 16 bits, utilizamos un bit por pivote maximizando la cantidad de pivotes.

Tal como sucedió en los experimentos anteriores, δ_{ma} es las más competitiva y δ_{pi} es la de menor performance. La función δ_{ma} logra una lista de candidatos con un $20\,\%$ menos de elementos que δ_{ci}, la que le sigue en eficiencia, para todos los radios de búsqueda. Respecto del tiempo sucede algo similar pero en este caso las mejoras logradas por δ_{ma} son más significativas cuando se aumenta el radio de búsqueda. Podemos ver que δ_{μ_p} se ve afectada por el tamaño de firma, perdiendo frente a δ_{ci} tanto en cardinalidad de lista de candidatos como en tiempo de búsqueda.

Todos los experimentos realizados nos permiten concluir que δ_{ma} es una buena opción para este tipo de espacios métricos, superando a las demás funciones conocidas tanto en cardinalidad de lista de candidatos (cantidad de comparaciones) como en tiempo total de búsqueda.

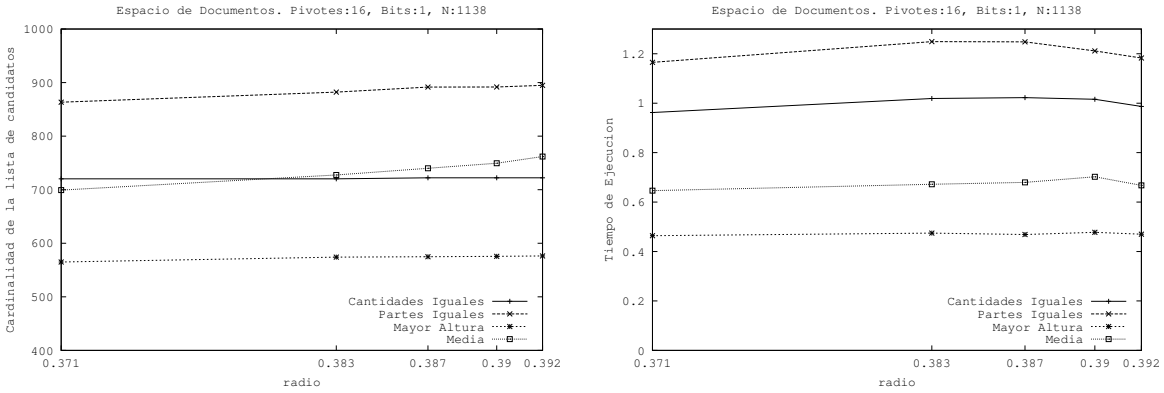

Figura 8: *Resultados para* 1138 *documentos (*$90\,\%$ *de la colección de* 1265 *documentos).*

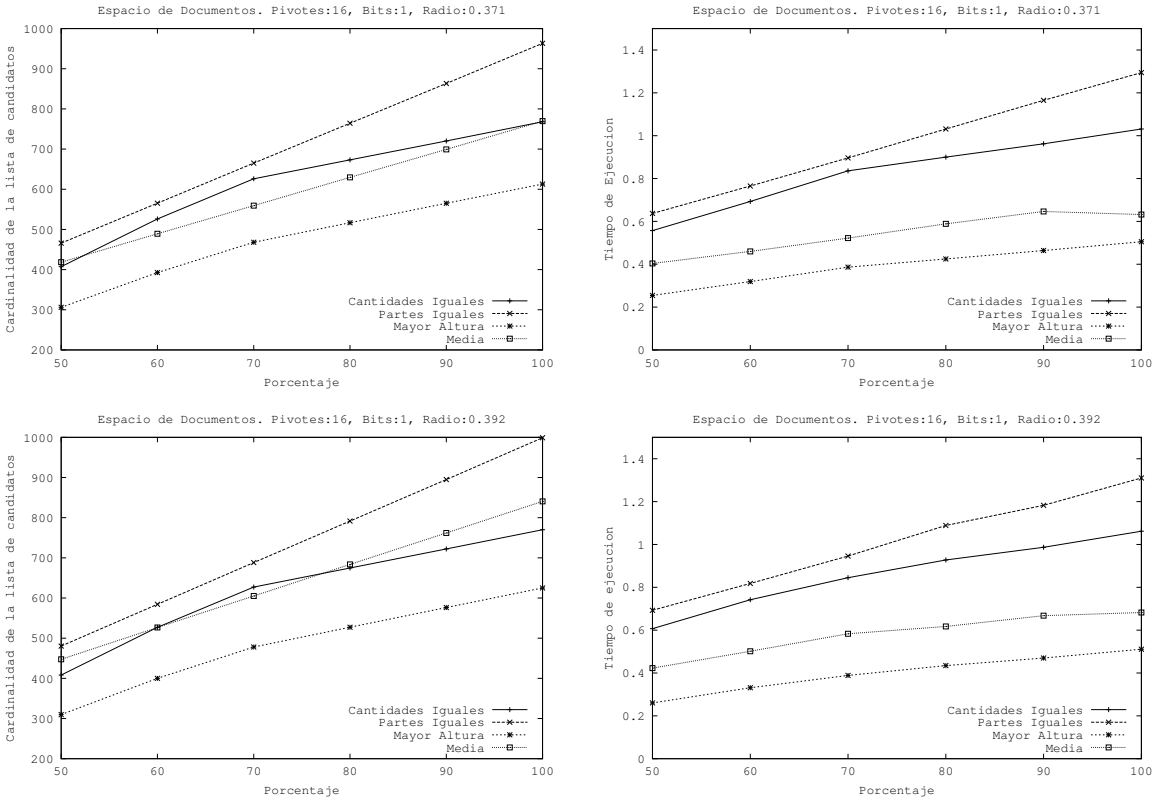

Figura 9: *Resultados para radios* $0{,}371$ *(arriba) y* $0{,}392$ *(abajo), considerando distintos tamaños de espacio y usando 16 bits para la firma.*

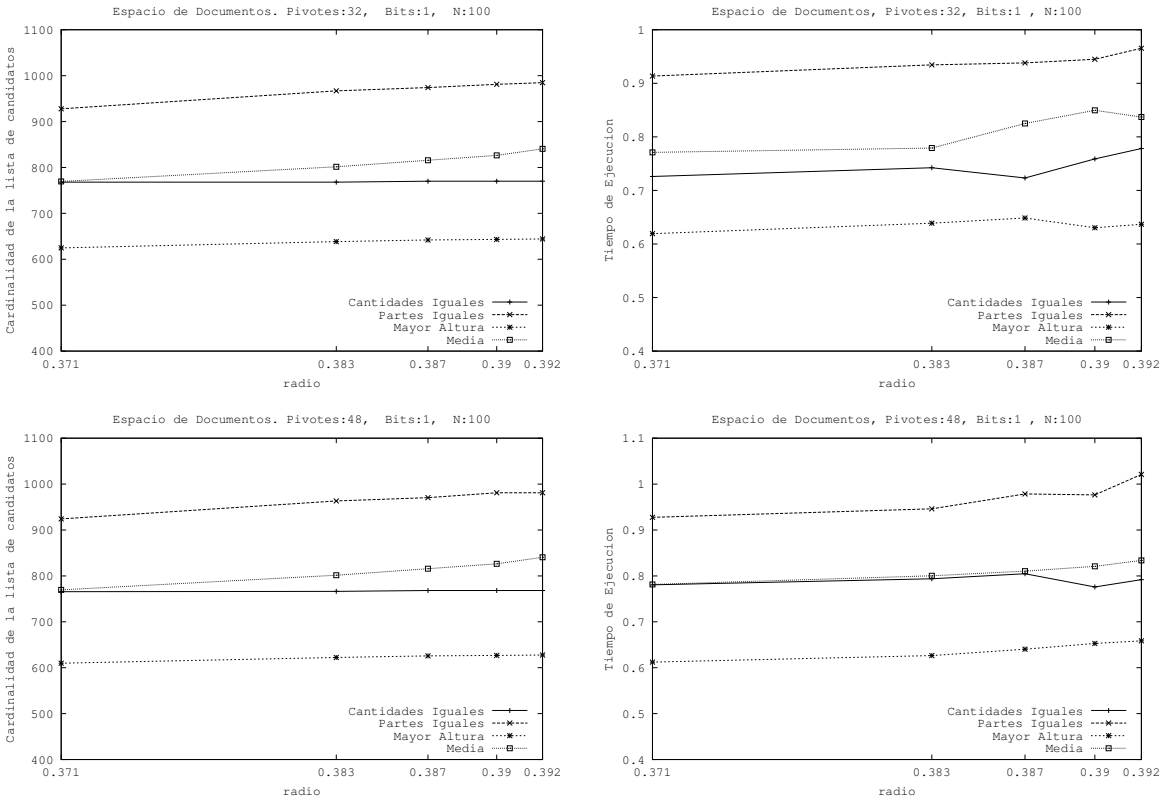

Figura 10: *Resultados usando 32 bits (arriba) y 48 bits (abajo) como tamaño de firma.*

5. Conclusiones y Trabajo Futuro

Basándonos en los resultados dados en [4] y los resultados expuestos en este artículo, podemos concluir que las funciones de discretización que obtienen el mejor desempeño son aquellas que se basan en el histograma local de cada pivote. En particular, dividir la zona de mayor concentración de elementos, asignando valores distintos a cada una de ellas, parece ser la clave para obtener funciones con buen rendimiento.

Recordemos que, dada una consulta q, cada pivote p sólo puede descartar a aquellos u que cumplan con $d(p,u) \notin [d(p,q)-r, d(p,q)+r]$. Por otro lado hay una alta probabilidad de que la query q se ubique en la zona de mayor concentración de elementos. Si dividimos esa zona, al momento de realizar una consulta cada pivote está en condiciones de descartar una porción significativa de elementos del espacio. Ésta es la razón por la cual δ_{ma} y δ_{μ_p} obtienen mejor desempeño para espacios continuos y discretos respectivamente.

Con respecto al trabajo futuro, nos proponemos estudiar el efecto de las funciones de discretización sobre otros índices basados en pivotes. Si bién los resultados que mostramos en este artículo fueron obtenidos sólo con el FQTrie (el mejor índice de la familia FQ), los mismos nos permiten extrapolar el compartamiento de otros índices integrados con las funciones δ_{μ_p} y δ_{ma} .

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] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] E. Chávez C. Ruano and N. Herrera. Discretización binaria para el ftrie. In *Actas del X Congreso Argentino de Ciencias de la Computación (CACIC'04)*, pages 100–111, Buenos Aires, Argentina, 2004.

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

[6] 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.

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

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