Búsquedas por Similitud en Espacios Métricos: el FQtrie [*]

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

Norma E. Herrera, Carina M. Ruano, Ana V. Villegas
Departamento de Informática
Universidad Nacional de San Luis
{nherrera,cmruano,anaville}@unsl.edu.ar
Fax: +54 (2652) 430224

1 Introducción

La búsqueda de elementos cercanos o similares a uno dado, es un problema que aparece en diversas áreas. Este concepto fue motivado como una extensión natural del concepto de búsqueda exacta, ante el surgimiento de nuevos tipos de bases de datos tales como base de datos de imágenes, de sonido, de texto, etc.

Las bases de datos tradicionales se construyen basándose en el concepto de búsqueda exacta: la base de datos es dividida en registros y cada registro contiene campos completamente comparables. Las consultas a la base de datos retornan todos aquellos registros cuyos campos coinciden con los aportados en tiempo de búsqueda.

Estructurar datos no tradicionales (tales como imágenes, sonido, video, etc.) en registros para adecuarlos al concepto tradicional de búsqueda exacta, es difícil en muchos casos y hasta imposible si la base de datos cambia más rápido de lo que se puede estructurar (como por ejemplo la web). Aun cuando pudiera hacerse, las consultas que se pueden satisfacer con la tecnología tradicional, están limitadas en variaciones de la búsqueda exacta.

Nos interesan las búsquedas en donde se puedan recuperar objetos similares a uno dado. Este tipo de búsqueda, se conoce con el nombre de *búsqueda por proximidad o búsqueda por similitud*, y surge en áreas tales como reconocimiento de voz, reconocimiento de imágenes, compresión de texto, recuperación de texto, biología computacional, etc. La necesidad de una respuesta rápida y adecuada, y un uso eficiente de memoria, hacen necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos.

Todas las aplicaciones mencionadas poseen algunas características comunes. Existe un universo $\mathcal{X}$ de objetos y una función de distancia $d : \mathcal{X} \times \mathcal{X} \to R^+$ que modela la similitud entre los objetos. La función d cumple con las propiedades características de una función de distancia:

(a) $\forall x, y \in \mathcal{X},\ d(x, y) \geq 0$ (positividad)

[*]Este trabajo es parcialmente subvencionado por CYTED VII.19 RIBIDI Project (todos los autores) y por CONACyT 36911-A (Edgar Chávez)

(b) $\forall x, y \in \mathcal{X},\ d(x,y) = d(y,x)$ (simetría)

(c) $\forall x, y, z \in \mathcal{X},\ d(x,y) \leq d(x,z) + d(z,y)$ (desigualdad tringular)

El par $(\mathcal{X}, d)$ es llamado *espacio métrico* [8]. La base de datos es un conjunto finito $\mathcal{U} \subseteq \mathcal{X}$, el cual se preprocesa a fin de resolver búsquedas por similitud eficientemente.

Existen dos tipos básicos de búsquedas por similitud en espacios métricos :

Búsqueda por rango: consiste en recuperar todos los elementos de $\mathcal{U}$ que están a distancia r de un elemento q dado. En símbolos, $(q,r)_d = \{u \in \mathcal{U} : d(q,u) \leq r\}$

Búsqueda de los k-vecinos más cercanos: consiste en recuperar los k elementos más cercanos a q en $\mathcal{U}$. Esto significa encontrar un conjunto $A \subseteq \mathcal{U}$ tal que $|A| = k \wedge \forall u \in A,\ v \in (\mathcal{U} - A) : d(q,u) \leq d(q,v)$

Un caso particular de espacios métricos son los espacios vectoriales K-dimensionales, con las funciones de distancia L_p:

$$L_p((x_1 \ldots x_k),(y_1 \ldots y_k)) = (\sum_{1 \leq i \leq k} |x_i - y_i|^p)^{1/p}, 1 \leq p < \infty$$

$$L_p((x_1 \ldots x_k),(y_1 \ldots y_k)) = max_{1 \leq i \leq k} |x_i - y_i|,\ p = \infty$$

Para estos casos existen soluciones bien conocidas tales como KD-tree [2], R-tree [9], etc. Sin embargo, éste no es el caso general. Las investigaciones en la actualidad tienden al estudio de algoritmos en espacios métricos generales.

Es claro que, una búsqueda por similitud, puede resolverse de forma ineficiente examinando exhaustivamente la base de datos $\mathcal{U}$. Para evitar esto, se preprocesa $\mathcal{U}$ usando algún algoritmo de indexación: un proceso off-line que construye una estructura de datos o índice, diseñada para ahorrar cálculos en el momento de resolver una búsqueda.

El tiempo total de resolución de una búsqueda puede ser calculado de la siguiente manera:

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, aun 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, dependiendo de la memoria principal disponible y del costo relativo de computar la función d. Los trabajos sobre espacios métricos, generalmente se han enfocado en algoritmos para descartar elementos en tiempo de búsqueda, dejando de lado las consideraciones sobre el tiempo de *I/O*. La única excepción, ha sido el *MTree* [5], diseñado específicamente para memoria secundaria.

Otro punto importante, es que la mayoría de las estructuras para espacios métricos se construyen bajo el supuesto de que el conjunto de datos es estático. En muchas aplicaciones esto no es razonable, dado que los elementos son insertados y eliminados dinámicamente. Algunas estructuras toleran inserciones pero muy pocas eliminaciones.

2 Algoritmos Basados en Pivotes

Básicamente existen dos enfoques para el diseño de algoritmos de indexación en espacios métricos: uno está basado en Diagramas de Voronoi [1, 8] y el otro está basado en pivotes [8].

La idea subyacente de los algoritmos 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 $\Phi(a) = (d(a, p_1), d(a, p_2), \ldots, d(a, p_k))$.

Ante una búsqueda $(q, r)_d$, se computa $\Phi(q) = (d(q, p_1), d(q, p_2), \ldots, d(q, p_k))$. Luego, se descartan todos aquellos elementos a, tales que para algún pivote p_i, $\mid d(q, p_i) - d(a, p_i) \mid > r$, es decir:

$$max_{1 \leq i \leq k} \mid d(q, p_i) - d(a, p_i) \mid = L_\infty(\Phi(a), \Phi(q)) > r$$

Esto significa que, todos los algoritmos basados en pivotes, proyectan el espacio métrico original, en un espacio vectorial k dimensional con la función de distancia L_∞. La diferencia entre todos ellos, radica en cómo implementan la búsqueda en el espacio mapeado.

La familia de estructuras *FQ (FQT, FHQT, FQA, FQtrie)* forman parte de las estructuras basadas en pivotes, y son algunas de las pocas que soportan tanto inserciones como eliminaciones. Cada una de ellas fue presentada como una mejora de la anterior:

- *FQT* [4]: construye un árbol cuya profundidad depende de la dimensión del espacio mapeado (cantidad de pivotes).
- *FHQT* [3]: es una mejora al FQT. Se construye el árbol de manera tal que todas las hojas están a una misma profundidad. Ésta estructura demostró experimentalmente ser mejor que su antecesora en tiempo, pero tiene como desventaja la cantidad de memoria utilizada.
- *FQA*[7]: produce una mejora importante en el tamaño del índice, reemplazando el árbol por un un arreglo ordenado de firmas. El barrido en el árbol es simulado por medio de una búsqueda binaria en el arreglo, lo que aumenta en un factor logarítmico la cantidad de cómputos extras necesarios para resolver la búsqueda. El *FQA* es un enfoque interesante y eficiente si la función de distancia d es muy costosa de calcular. Si esto no es así, entonces el tiempo extra de CPU representa una porción grande del costo.
- *FQtrie*[6]: es una mejora al FQA, en la que se logra eliminar el factor logarítmico en las búsquedas, utilizando un árbol digital para representar el conjunto de firmas de elementos de la base de datos.

En el caso particular del FQtrie, si bien en su concepción original permite inserciones y eliminaciones, hasta el momento no se ha realizado una implementación del mismo en la que se incorporen estas operaciones.

3 Trabajo Futuro

El punto de partida de nuestro trabajo es el $FQtrie$. Esta estructura fue presentada recientemente [6] y por consiguiente no se ha realizado, hasta el momento, un estudio exhaustivo de la misma.

Nos proponemos atacar tres problemas: tiempo extra de *CPU*, tiempo de *I/O* y dinamismo. El objetivo es lograr una implementación eficiente (en términos de tiempo extra de *CPU*), totalmente dinámica y con manejo de espacios métricos cuyo índice completo exceda la capacidad de la memoria principal.

El trabajo a realizar puede resumirse en los siguientes puntos:

- Con respecto al tiempo extra de *CPU*, las *tablas lookup* han demostrado ser una buena opción para mejorar el desempeño del FQA y del *scan secuencial* [6]. Dada una búsqueda $(q, r)_d$, una tabla lookup es una estrategia de representación de $\Phi(q)$, que permite realizar comparaciones entre palabras de máquina completas en lugar de hacerlas por grupos de b bits (donde b es la cantidad de bits necesitados para codificar $d(a, p_i)$). Utilizaremos esta técnica en la implementación del *FQTrie*, determinando posibles variaciones de la misma.

- Para el manejo de espacio métricos cuyo índice completo exceda la capacidad de memoria principal, en lugar de modificar la estructura para que sea eficiente su manejo en disco, particionaremos el espacio métrico, de manera tal que el índice de cada una de las partes entre en memoria principal. Luego, una búsqueda $(q, r)_d$ se resuelve buscando separadamente en cada uno de los índices, lo que puede ser hecho en memoria principal y en paralelo.

- Con respecto al particionamiento, actualmente estamos analizando distintas posibilidades. En este sentido, la estrategia de partición en dos núcleos presentada en [10], ha mostrado un buen desempeño para estructuras basadas en Diagramas de Voronoi, pero presenta algunas debilidades cuando se aplica a estructuras basadas en pivotes. Estamos estudiando el origen de este problema, para poder, en función de ello, diseñar una nueva estrategia de partición, que sea eficiente para estructuras basadas en pivotes, y que además sirva para dividir el espacio en una cantidad arbitraria de partes.

- Como nos proponemos realizar una implementación dinámica del *FQtrie*, el realizar inserciones y eliminaciones puede provocar desbalances en las cardinalidades de las partes. Esto debe ser tomado en cuenta a fin de decidir el momento y el modo apropiado para reorganizar la partición.

Como producto final, se obtendrá una implementación del *FQTrie*, eficiente en términos de tiempo extra de *CPU*, totalmente dinámica y con manejo de espacios métricos en disco.

Referencias

[1] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 1991.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9), 509:517, 1975.

[3] R. Baeza-Yates. Searching: an algorithm tour. Allen Kent and James G. Willias, editors, *Encyclopedia of Computer Science and Technology*, Vol 37 331-359, Marcel Dekker, 1997.

[4] R. Baeza-Yates, W. Cunto, U. Manber, S. Wu. Proximity matching using fixed-queries trees. *Proc. 5th. Combinatorial Pattern Matching (CPM94), LNCS 807*, 198-212, 1994.

[5] P. Ciaccia, M. Patella, F. Rabitti, P. Zezula. Indexing Metric Spaces with M-Tree. *Proc of Sistemi Evolui per Basi di Dati*, 67:86, 1997.

[6] E. Chávez. Faster Proximity Searching Using Lookup Tables. *Comunicación interna, Universidad Michoacana, México*, 2002.

[7] E. Chávez, J. Marroquín, G. Navarro. Fixed Queries Array: a fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2): 113-135 2001.

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

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD International Conference on Management of Data*,47:57, 1984.

[10] N. Herrera, E. Chávez. Parametrización local de espacios métricos. *Proc. Congreso Argentino de Ciencias de la Computación (CACIC'02)*.