Búsqueda en espacios métricos: posibles optimizaciones al Geometric Near-neighbor Access Tree

Norma Edith Herrera
Nora Susana Reyes
Universidad Nacional de San Luis, Argentina
{nherrera, nreyes}@unsl.edu.ar

Ricardo Baeza Yates
Universidad de Chile, Chile
rbaeza@dcc.uchile.cl

1. Introducción

La evolución de las tecnologías de comunicación e información ha permitido que las bases de datos tengan la capacidad de almacenar nuevos tipos de datos tales como imágenes, sonido, video, etc. Es sumamente dificultoso, y en algunos casos imposible, estructurar estos tipos de datos en registros para que se adecuen al concepto tradicional de búsqueda exacta (esto es recuperar aquellos registros cuyos campos coincidan con los aportados en la búsqueda). Aún cuando pudiera hacerse, las consultas a la base generalmente serán por objetos *similares* a uno dado y no por aquellos exactamente iguales.

Estos tipos de búsquedas se conocen con el nombre de *búsquedas aproximadas o búsquedas por similitud*, y aparecen en áreas tales como reconocimiento de voz, reconocimiento de imágenes, recuperación de texto, detección de copias, compresión de videos, bases de datos médicas, etc.

Todas estas aplicaciones tienen algunas características comunes: existe un universo U de objetos y una función de distancia $d: U \times U \rightarrow \mathbb{R}^+$, que modela la similitud entre elementos.

La función d cumple con las propiedades propias de una función de distancia:

$\forall\, x, y, z \in U$:

1) $d(x, y) = 0 \Leftrightarrow x = y$
2) $d(x, y) = d(y, x)$
3) $d(x, y) \leq d(x, z) + d(z, y)$ desigualdad triangular

El par (U, d) es llamado *espacio métrico*.

La base de datos es un conjunto finito $C \subseteq U$. Llamaremos a C *diccionario*, *base de datos*, o simplemente nuestro *conjunto de objetos* o *elementos*.

Dado un punto q, debemos ser capaces de recuperar los elementos de C que sean similares a él. Básicamente, existen tres tipos de búsquedas de interés en espacios métricos:

a) ***Búsqueda por rango***: recuperar todos los elementos de C que están a distancia r de q .

b) ***Búsqueda del vecino más cercano***: recuperar el (o los) elemento(s) más cercano(s) a q en C.

c) ***Búsqueda de los K vecinos más cercanos***: recuperar los k elementos más cercanos a q en C.

Un caso particular de espacios métricos son los espacios vectoriales K-dimensionales, con las funciones de distancias L_p (p=1,2, ...,∞). Para estos casos existen soluciones bien conocidas, tales como Diagramas de Voronoi, KD-tree y R-tree.

Sin embargo, este no es el caso general. En algunas aplicaciones, resulta difícil mapear cada objeto en un punto k dimensional de un espacio vectorial, manteniendo la distancia del espacio métrico en el espacio vectorial.

Las investigaciones en la actualidad tienden al estudio de algoritmos en espacios métricos generales, donde existen varias técnicas conocidas para resolver el problema en un número sublineal de cálculos de distancia, con la condición del preprocesamiento de C.

Nuestro objetivo es estudiar principalmente el **Geometric Near-neighbor Access Tree (GNAT),** buscando nuevas optimizaciones al mismo y analizando su comportamiento para distintos tipos de datos

d) Geometric Near-neighbor Access Tree (GNAT)

El propósito de los diseñadores del GNAT [1] es que la estructura de datos actúe como un modelo geométricamente jerárquico de los datos. Mas específicamente, que el nodo raíz nos dé una noción de los datos como espacio métrico, y a medida que avanzamos en la jerarquía del árbol obtener una idea más exacta acerca de la geometría de los mismos. Esto se logra construyendo una jerarquía basada en Diagramas de Voronoi.

La construcción del GNAT procede de la siguiente manera: en el primer nivel de árbol seleccionamos m pivotes $p_1, p_2, \ldots, p_m$ de C, y asociamos a cada p_i el conjunto C_{pi} definido de la siguiente manera:

$$C_{pi} = \{c \in C \,/\, d(p_i, c) < d(p_j, c) \;\; \forall j=1 \ldots m, \; j \neq i \}$$

Es decir, en C_{pi} colocamos todos los elementos de C que están mas cerca de p_i que de cualquier otro pivote p_j. Con cada C_{pi} construimos recursivamente un GNAT.

En cada p_i se almacena además el rango de distancias desde p_i a los puntos asociados con otros pivotes p_j. Es decir, por cada p_i calculamos $(m-1)$ rangos de valores definidos de la siguiente manera:

$$\text{rango}(p_i, p_j) = [\, min \{ d(p_i, c) \,/\, c \in C_{pj} \}, \max \{ d(p_i, c) \,/\, c \in C_{pj} \}]$$

Esta información junto con la desigualdad triangular, se usa en el momento de una búsqueda para descartar subárboles:

> Sea q un punto de query, r el rango de búsqueda y $P=\{p_1, p_2, \ldots, p_m\}$ el conjunto de pivotes del primer nivel. Se compara q con algún pivote p_i y se descartan los C_{pj} para aquellos p_j tales que:
>
> $$d(q, p_i) \pm r \cap \text{rango}(p_i, p_j) = \varnothing$$
>
> Se repite este proceso hasta que ningún subárbol pueda ser descartado.
> La búsqueda continúa ahora recursivamente sobre los subárboles no eliminados.
> Durante el proceso de búsqueda cualquier pivote p_i tal que $d(q, p_i) \leq r$, se agrega al resultado.

e) Optimizaciones en estudio

En la construcción del GNAT hay varios puntos a tener en cuenta porque afectarán la performance de la estructura en el momento de la búsqueda. Nos proponemos estudiarlos, buscando nuevas optimizaciones y analizando experimentalmente su comportamiento para distintos conjuntos y tipos de datos.

- ***Elección de los pivotes***

Poco se conoce sobre las políticas de selección de pivotes, por lo tanto una posibilidad es hacer una elección absolutamente al azar. De esta forma la selección puede resultar en puntos que sean cercanos a los centros de clusters. Sin embargo no queremos que los pivotes estén agrupados, porque en ese caso no tendremos información sobre contenidos (la información almacenada sobre los rangos no nos permitiría descartar subárboles)

Una estrategia para evitar esta situación es seleccionar al azar un conjunto de puntos P, cuya cardinalidad sea mayor a la cantidad de pivotes necesarios. De este conjunto elegimos los pivotes de forma tal que estén tan alejados unos de otros como sea posible: el primer pivote p_1 se elige al azar; elegimos como segundo pivote p_2 a aquel punto de P que está más alejado de p_1; el próximo punto será aquel que esté mas alejado de p_1 y p_2 y así seguimos hasta completar la cantidad de pivotes requeridos.

En [1] se ha observado comportamientos patológicos de este método, por lo que nos proponemos estudiar métodos alternativos.

- ***Elección del grado de cada nodo (cantidad de pivotes)***

Una primera posibilidad es asignar el mismo grado a todos los nodos. Pero esta estrategia nos puede producir árboles desbalanceados, dado que aquellos subárboles con menor cantidad de elementos tendrán una altura considerablemente menor que aquellos con mayor cardinalidad.

Para solucionar esto, en [1] se propone la siguiente estrategia: si el grado de un nodo es K, entonces a cada hijo se le asigna un grado proporcional al número de elementos que contiene. Allí también se menciona que el grado de un nodo depende de su profundidad en el árbol, pero no dan criterios claros sobre cómo hacer esto.

En este punto, nuestro objetivo es encontrar estrategias que nos permitan decidir el grado de un nodo y criterios para determinar cuándo usarlas.

- ***Estructuras de datos híbridas***

Una alternativa es no construir un GNAT para cada C_{pj}, sino que, dependiendo de la distribución de los elementos en el espacio, prever estructuras de datos alternativas que permitan mejorar la performance global de la representación.

Para esto nos proponemos analizar mediante histogramas de distancias, las características de subárboles pequeños y encontrar mejores alternativas de representación para ellos.

- ***Algoritmo de búsqueda***

En este punto nuestro objetivo es encontrar formas alternativas de recorrer el árbol, de forma tal de acelerar los tiempos de búsqueda.

- ***El espacio de datos.***

Las respuestas a los puntos planteados anteriormente se verán influenciadas por las características particulares del conjunto de datos y su distribución en el espacio métrico.

Por este motivo, nos proponemos estudiar el comportamiento de tres tipos de datos: *vectorial, palabras e imágenes*, y en función de ello buscar las respuestas a las problemáticas anteriormente planteadas.

REFERENCIAS

[1] S. Brin. **Near neighbor search in large metric spaces.** In Proc. 21st Conference on Very Large Databases (VLDB '95), pages 574-584, 1995.

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

[3] E. Chavez, G. Navarro, R. Baeza-Yates, J. Marroquín. **Searching in metric spaces**. Technical report TR/DCC-99-5, Dept. of Computer Science, Univ. of Chile. 1999.

[4] E. Chavez, J. Marroquín, R. Baeza-Yates. **Spaghettis: an array based algorithm for similarity queries in metric spaces**. In Proc. String Processing and Information Retrieval (SPIRE '99), Cancún, México. September 1999, pages 38-46.

[5] E. Chavez, J. Marroquín, G. Navarro. **Overcoming the curse of dimensionality**. In European Workshop on Content-Based Multimedia Indexing (CBMI '99), pages 57-64

[6] P. Yianilos. **Data structures and algorithms for nearest neighbor search in general metric spaces.** In Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA '93), pages 311-321, 1993.