Algoritmos y Estructuras de Datos para Búsqueda de Objetos Similares

Virna Cuquejo

Centro Nacional de Computación
Universidad Nacional de Asunción, Paraguay
vcuquejo@cnc.una.py

Ricardo Baeza-Yates
Gonzalo Navarro
Depto. de Ciencias de la Computación
Universidad de Chile, Chile
{rbaeza,gnavarro}@dcc.uchile.cl

Resumen

La búsqueda de objetos similares a un objeto dado dentro de una base de datos de objetos, bajo un cierto criterio de similaridad conocido de antemano, tiene varias aplicaciones en computación. Varios algoritmos existen para resolver esta búsqueda utilizando para la indexación los modelos de semejanza que definen un espacio vectorial y un espacio métrico.

En el espacio vectorial sobresale la estructura K-d tree, método de acceso en memoria principal, ya que propuso ideas que han sido utilizados en varios métodos de accesos a datos en disco. Por otro lado, el R-tree, es un método en memoria secundaria, que ha sido implementado en sistemas académicos y comerciales como POSTGRES e ILLUSTRA. Desafortunadamente, cuando la dimensión del objeto crece, ambas estructuras requieren más espacio y las consultas se hacen más lentas por lo que realizamos pruebas empíricas sobre las consultas del vecino más próximo en cuanto al tiempo de CPU y la cantidad de cálculos de distancia realizadas según el incremento de la dimensión.

Palabras clave: Búsqueda Similar, Espacio Métrico, Espacio Vectorial, Vecino más próximo, Vecinos a cierta distancia.

1 Introducción

Hoy en día surgen aplicaciones donde se desea buscar objetos similares en grandes bases de datos. Por ejemplo, encontrar una imagen similar a una dada o realizar consultas sobre datos que pueden tener más de un atributo, por ejemplo registros con d atributos.

De la misma forma surgen los siguientes tipos de consultas:

Vecino más próximo: el usuario especifica un punto o región, y el sistema debería retornar el objeto más cercano.

Vecinos a cierta distancia: se refiere a seleccionar los elementos de un conjunto de datos (un conjunto finito del espacio) que están a cierta distancia de un punto dado.

Representar datos como puntos en un espacio y usar métodos geométricos para su indexación, es una técnica aplicada en áreas como:

- Genética: encontrar secuencias de ADN o proteínas similares en alguna base de datos genética.
- Reconocimiento de voz: encontrar patrones vocales similares desde una base de datos de patrones vocales (Ej: bajo la transformada de Fourier).
- Reconocimiento de imágenes: encontrar imágenes similares (usando una métrica sobre una imagen dada desde una gran biblioteca de imágenes).
- Compresión de video: encontrar bloques de imagen de una figura previa que son similares a bloques en una nueva imagen (usando métricas L_1 o L_2, posiblemente después de una transformada *Discrete Cosine Transform (DCT)*).
- Minería de datos: encontrar series de tiempo aproximados o patrones no conocidos de una base de datos (ej. histórico de inventario).
- Recuperación de información: encontrar documentos relacionados a uno dado en una biblioteca digital.
- Detección de copia: encontrar sentencias similares a una consulta de un usuario en una gran base de datos de documentos (espacio métrico de strings).
- Base de Datos Médica: donde se almacenan imágenes 2-d como rayos-X e imágenes 3-d como la tomografía computarizada. Recuperar casos anteriores con similares síntomas es muy útil para diagnósticos, enseñanza médica e investigación.

La sección 2 del trabajo hacemos una revisión de los modelos de semejanza para la indexación. Presentamos las estructuras de K-d Tree y R-tree en la sección 3. Detallamos las implementaciones realizadas y los resultados de las comparaciones en la sección 4. En la sección 5 describimos los trabajos de investigación en el área y las conclusiones de este trabajo.

2 Modelos de semejanza para indexación

2.1 Modelo de espacio vectorial (VSM)

Se asume que cada objeto puede ser completa o parcialmente representado como un punto d-dimensional en un espacio vectorial, donde d es fijo. Luego utilizar métodos para manejar puntos, llamados *Point Access Methods* (PAMs). Otro método puede manejar también objetos espaciales (rectángulos, polígonos, etc) además de puntos multidimensionales, llamados *Spatial Access Methods* (SAMs).

La función de distancia más usada es la norma de Minkowsky, definida como:

$$L_s(P,R) = \left(\sum_{i=1}^{d} |p_i - r_i|^s\right)^{1/s}$$

Entre ellas la más utilizada es la Euclidiana L_2, pero otras como L_1 conocida como Manhattan, o L_∞ pueden ser utilizadas.

Las estructuras de índices en este modelo son más eficientes pero menos flexibles. Como ejemplos estan los k-d trees [Ben75] y los R-tree [Gut84].

2.2 Modelo de Espacio Métrico (MSM)

El modelo de espacio vectorial puede ser restrictivo para algunas aplicaciones.

Es difícil mapear cada objeto en un punto d-dimensional de un espacio vectorial mientras se mantiene la precisión de la representación de semejanza o distancias entre objetos usando alguna métrica en ese espacio.

Esto ocurre cuando la noción de semejanza es compleja y dependiente del dominio, como en el caso de similaridad entre huellas digitales.

Un espacio métrico es un espacio donde se tiene un conjunto X con una función de distancia d, $d : X \times X \to R$ tal que $\forall\, x, y, z \in X$:

$d(x,y) \geq 0$	positividad
$d(x,y) = 0 \Leftrightarrow x = y$	positividad estricta
$d(x,y) = d(y,x)$	simetría
$d(x,x) = 0$	refleja
$d(x,y) + d(y,z) \geq d(x,z)$	desigualdad triangular

Las cuatro primeras propiedades de similaridad permiten una definición consistente de la función $d(x,y)$. La desigualdad triangular permite descartar puntos imposibles en consultas aproximadas.

En estas aplicaciones, el índice es construido con sólo la función de distancia métrica, sin conocer la representación usada para calcular la distancia. Cualquier método de indexación diseñado para este modelo puede ser usado para el modelo vectorial, ya que el mismo es un caso particular de espacio métrico donde los objetos son representados por d coordenadas de valores reales.

¿Por qué un espacio métrico?

Es deseable buscar algoritmos en los cuales la complejidad no dependa estrictamente de la dimensión, sino mas bien de propiedades intrínsecas de los datos. Otra razón es que algunas medidas

de distancia no son aplicables a espacios con coordenadas reales (por ejemplo, distancias de edición sobre strings). Desde un punto de vista teórico, parece razonable encontrar las propiedades mínimas necesarias para algoritmos rápidos.

En este modelo existen varias técnicas conocidas para resolver las consultas en un número sublineal de cálculos de distancia si podemos preprocesar los datos. Faragó et al. [FLL93] probó que el número promedio de cálculos de distancia es constante y no depende del número de elementos del conjunto de datos.

3 Métodos de accesos a puntos (PAMS) y los métodos de accesos espaciales (SAMS).

3.1 K-d trees

Esta diseñado como Método de Acceso a Puntos (PAMs) y sentó las bases para importantes ideas que varios Métodos de Acceso Espacial (SAMs) han utilizado posteriormente [Ben75].

La idea es mantener la noción de un árbol binario, pero cortando el espacio usando un sólo hiperplano ortogonal. En cada nivel del árbol variamos el *eje de corte*. Por ejemplo, en 2 dimensiones en la raíz cortamos perpendicular al eje x; en el siguiente nivel cortamos perpendicular al eje y, etc.

Para representar esta estructura, se guarda un nodo por cada corte hecho. Cada nodo tiene 2 hijos. Todos los puntos menores que el punto de corte actual p, de acuerdo al eje discriminado, se guardan en los hijos de la izquierda, y en los de la derecha los mayores (si el eje de un punto es igual a p, se puede insertar en cualquiera de los dos hijos. Esto es para tratar de balancear los subárboles). Cuando se llega a un punto o a un número pequeño de puntos (menor o igual al tamaño del bucket), se almacena en las hojas.

No se necesita almacenar el eje de corte, esta información se tiene al recorrer el árbol. Si la dimensión del corte es i en algún nivel, se tiene que la dimensión del corte de sus hijos es $(i+1)\ mod\ d$, donde d es la dimensión.

¿Cómo elegimos el valor de corte? Para un mejor rendimiento, la descomposición del espacio debería ser lo más balanceada posible. El método más común utilizado para ello es seleccionar el valor de corte basado en la mediana del eje de corte. Esto produce un árbol de altura $O(\log n)$.

Con un procedimiento recursivo se puede construir el árbol en $O(n \log n)$. El costo más alto es determinar la mediana del eje.

Consulta del Vecino más próximo: Dado un punto de consulta q , para buscar el vecino más próximo primero descendemos por el árbol. Mantenemos un punto candidato a ser el más próximo *nn* y una distancia de valor máximo *dist*. Para cada nodo visitado revisamos este punto, actualizándolo en caso necesario.

Luego necesitamos chequear los subárboles. Si q está en el lado inferior del hiperplano, nos vamos a la izquierda, en caso contrario a la derecha. Después de visitar el subárbol izquierdo hemos modificado el punto candidato *nn* y la distancia a este punto *dist*.

¿Necesitamos buscar en el subárbol derecho? ¿Hay alguna parte de la región de la derecha que puede estar más cerca de q que el actual candidato? Esto es equivalente a preguntar si el círculo centrado en q y cuyo radio es la distancia del vecino más próximo solapa la región

derecha. En vez de chequear la esfera misma, sólo chequeamos el punto de más a la derecha del círculo. Así buscamos en el subárbol derecho sólo si $q[cd] + dist \leq T \rightarrow data[cd]$ (raíz del subárbol). Un argumento similar se usa para el lado izquierdo

Si los objetos están uniformemente distribuidos el tiempo esperado es $O(\log n)$.

Este procedimiento es fácil de adaptar a dimensiones más altas, pero el rendimiento baja rápidamente, el tiempo de ejecución es realmente $O(2^d \log n)$, donde d es la dimensión. Cuando d crece este "factor constante" crece muy rápidamente.

Consulta a cierta distancia k: Para procesar una consulta a cierta distancia buscamos igual que en a la consulta del vecino más próximo, sólo reportamos cada vez que llegamos a un objeto cuya distancia al punto de consulta sea menor o igual a k, es decir, la distancia k es fija.

Para el caso estático, en el que se tiene un árbol perfectamente balanceado, el tiempo necesario para responder esta consulta con n puntos en 2 dimensiones es$O(r\sqrt{n})$, donde r es el número de puntos reportados. Cuando la dimensión crece esto empeora a $O(rn^{1-1/d})$. Esto es muy malo ya que $n^{1-1/d}$ es casi tan alto como n [Sam94].

En el caso de un árbol dinámico el tiemo de la consulta es $O(n^{1-1/d+\theta(1/d)})$ comparaciones donde $\theta(u)$ es una función estrictamente positiva de u para $0 < u < 1$, con un valor máximo de 0.07 [FP86].

3.2 R-tree

Diseñado como Método de Acceso Espacial (SAMs). Fue propuesto como una extensión de los B+-trees en dimensiones mayores a uno usando la técnica de solapamiento de regiones [Gut84]. Como los B-trees, es un árbol balanceado, pero además mantiene la flexibilidad de ajustar dinámicamente sus agrupaciones para soportar espacios muertos (*dead-space*) o áreas densas.

Representa a los objetos geométricos por un rectángulo d-dimensional más pequeño en el cual están contenidos. Está basado en la heurística de optimización de minimizar el área incluida por cada rectángulo en los nodos internos.

A menudo los nodos corresponden a páginas de disco. Notar que los rectángulos correspondientes a nodos diferentes pueden solaparse. Además, un rectángulo puede estar contenido espacialmente en varios nodos o ser asociado con sólo un nodo.

Reglas básicas de formación de un R-tree:

- Todos los nodos hojas están al mismo nivel
- El nodo raíz tiene al menos dos hijos a menos que sea una hoja.
- Cada nodo , con excepción de la raíz, contiene entre $2 \leq m \leq \lceil \frac{M}{2} \rceil$ y M entradas.
- Cada entrada en un nodo hoja es un par de la forma (MBR, O) tal que MBR (*Minimum Bounding Rectangle*) es el rectángulo más pequeño que contiene el objeto O.
- Cada entrada en un nodo interno es un par de la forma (MBR, P) tal que MBR es el rectángulo más pequeño que contiene los rectángulos en el nodo hijo apuntado por P.

El R-tree puede garantizar una utilización de espacio de al menos 50% y permanece balanceado.

Algoritmos

Inserción: Análogo al algoritmo usado para el B-tree. Los nuevos rectángulos son agregados en los nodos hojas. Se determina la hoja apropiada recorriendo el árbol desde la raíz y en cada paso eligiendo el subárbol cuyo rectángulo deba extenderse lo menos posible para incluirlo; en caso de conflictos se elige el rectángulo de menor área. Si el nodo hoja se desborda, se lo divide, y los M+1 registros deben distribuirse en dos nodos. La necesidad de división se puede propagar hacia los niveles superiores del árbol.

Split (División): Hay varias formas posibles de realizar la división de un nodo, y es una de las operaciones más cruciales para el rendimiento del R-tree. Guttman sugiere algoritmos basados en la minimización del área total cubierta por los rectángulos (*coverage*).

- El primer algoritmo (exponencial) es un algoritmo exhaustivo que intenta todas las posibilidades. En tal caso el número de posibles particiones es $2^M - 1$. Esto no es razonable para la mayoría de los valores de M (ej. M=50 para tamaños de página de 1024 bytes).
- El segundo algoritmo (cuadrático) primero encuentra dos rectángulos que deberían desperdiciar más área si ellos estuvieran en el mismo nodo. Esto es determinado restando la suma de las área de los dos rectángulos del área del rectángulo que incluye a ambos. Estos dos rectángulos son ubicados en nodos separados, j y k. Se examina el resto de los rectángulos y por cada rectángulo i, calculamos d_{ij} y d_{ik}, que es el incremento del área requerida del MBR de los nodos j y k para incluir a i. Elegimos el rectángulo n tal que $|d_{nj} - d_{nk}|$ sea máxima, y agregamos n al nodo con menor incremento en el área. Se repite este proceso para el resto de los rectángulos.
- El tercer algoritmo (lineal) encuentra los dos rectángulos, que serán los primeros elementos de los grupos, con la más alta separación normalizada a lo largo de todas las dimensiones, o sea, seleccionando los dos que están más separados a lo largo de cualquier dimensión o solapa menos. La distancia para la separación o solapamiento es normalizada dividiéndola por el ancho del espacio cubierto por el conjunto completo de rectángulos a lo largo de esa dimensión. El resto de los rectángulos son procesados en orden arbitrario y ubicados en el nodo cuyo MBR tiene el mínimo incremento con su agregación.

Guttman demostró empíricamente que no existían muchas diferencias entre los 3 algoritmos de corte en el rendimiento de las búsquedas. Sin embargo, en [BKSS90] con diferentes distribuciones, solapamiento, número variable de datos y diferentes combinaciones de M y m, el algoritmo cuadrático tuvo un mejor rendimiento que la versión lineal.

Eliminación: Para eliminar un rectángulo R, se localiza el nodo hoja L que contiene a R y se borra R de L. Luego se ajusta el MBR del camino desde L hasta la raíz del árbol mientras se remueven todos los nodos en el cual quedan rectángulos huérfanos (*underflow*) y agregándoles al conjunto U. Una vez que se llega a la raíz, si sólo queda un hijo, éste se convierte en raíz. Los elementos del conjunto U son reinsertados.

Búsqueda del vecino más próximo: La búsqueda parte del nodo raíz, se calcula todas las entradas que intersectan al rectángulo de la consulta Para cada una de las entradas halladas,

se lee el hijo correspondiente desde el disco a la memoria principal y la consulta es realizada como en la raíz .

El R*-tree [BKSS90] tiene el mejor rendimiento entre las variantes del R-tree. Se basa en el concepto de forzar la re-inserción, es decir, postergar la división, esto mejora el rendimiento en un 30% aproximadamente. La idea es que si un nodo se desborda, los p primeros hijos son elegidos luego de ordenarlos en forma decreciente de sus distancias entre el centro de sus rectángulos y el centro del MBR del nodo y posteriormente eliminados y re-insertados. Los experimentos mostraron que $p = 30\%$ de M para los dos tipos de nodos produce el mejor resultado. La motivación del R*-tree es disminuir el solapamiento entre los límites de los rectángulos. Así, cada rectángulo es asociado con todos los MBR que lo intersectan. El resultado es que puede haber varios caminos para el mismo rectángulo. Esto lleva a un incremento en la altura del árbol; sin embargo, mejora el tiempo de respuesta de las consultas.

Si los datos son estáticos y todos los objetos son conocidos a priori, se puede utilizar otra técnica para construir el R-tree. El Hilbert-Pack [KF93] se construye agrupando objetos calculando la curva de Hilbert (esto hace que el conjunto de datos en el mismo nodo esté más cerca uno del otro en un orden lineal, y más probablemente en el espacio original; así el R-tree resultante tiene áreas más pequeñas). Se ordenan los rectángulos en forma ascendente de acuerdo a e te valor. Una vez que un nivel completo del árbol ha sido construido, el algoritmo es llamado recursivamente para agregar nodos en los próximos niveles superiores, terminando cuando un nivel contiene sólo un nodo. En este proceso recursivo en un ciclo que recorre los N rectángulos, se genera un nuevo nodo y se le asignan los próximos C rectángulos al mismo. Luego, se ordenan estos nodos en forma ascendente por el tiempo de creación. Este método llena cada nodo hasta su capacidad.

4 Resultados Experimentales

Implementamos la estructura K-d tree como fue definido en [Ben75] y para la selección de la mediana utilizamos el algoritmo QuickSelect [Sed92] basado en el proceso de QuickSort. El caso promedio es de $O(n)$, pero con un peor caso $O(n^2)$. Es posible modificarlo para garantizar que el tiempo de ejecución sea lineal. Estas modificaciones son teóricamente importantes, pero complejas y no del todo prácticas. A fin de disminuir la probabilidad del peor caso elegimos el pivote utilizando la mediana-de-tres [Sed92].

El R-tree lo implementamos utilizando el algoritmo de búsqueda propuesto por Hjaltason y Samet [HS97] debido a que sólo recorre el subárbol más prometedor, evitando de este modo recorrer más de un subárbol en profundidad, para el caso del vecino más próximo. Además, usamos tamaños de página de 1024 bytes para los experimentos.

Las pruebas se realizaron sobre 9000 puntos generados aleatoriamente con una distribución uniforme, utilizamos la distancia euclidiana y para la consulta del vecino más próximo.

La figura 1 muestra la cantidad de cálculos de distancia realizados por cada algoritmo estudiado de acuerdo a la dimensión de los datos de entrada.

En la figura 2 se presenta el tiempo de CPU utizado para responder a la consulta deseada según la dimensión de cada entrada.

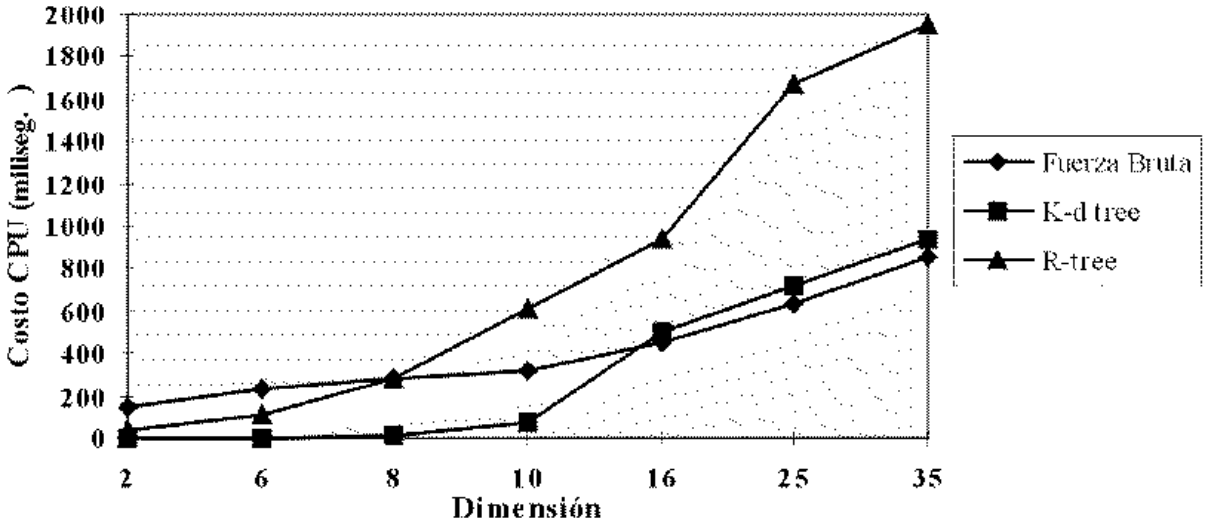

Figura 1: Número de cálculos de distancia con respecto a la dimensión.

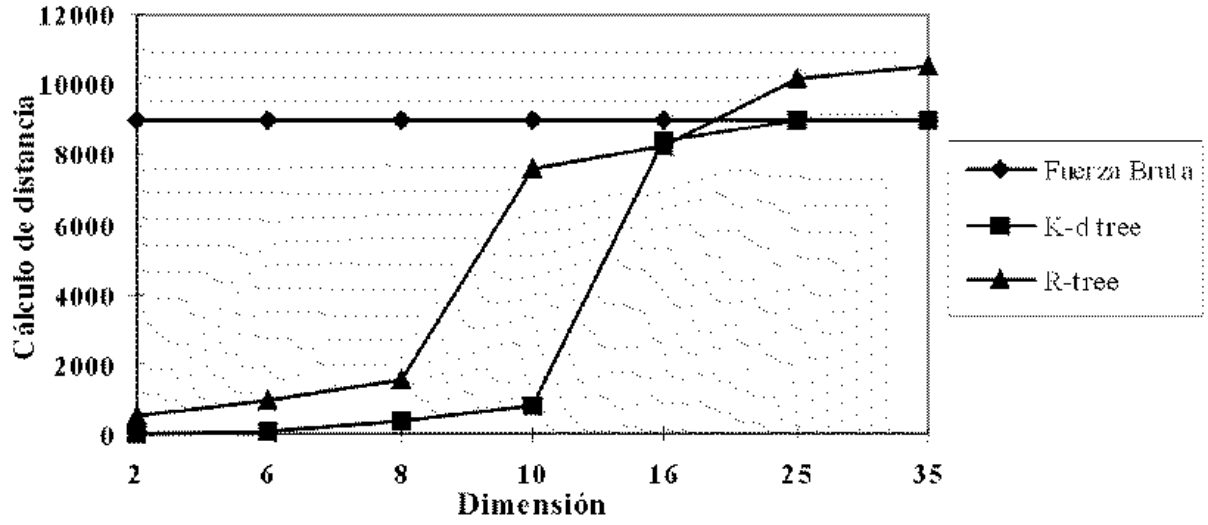

Figura 2: Tiempo de procesamiento por consulta con respecto a la dimensión.

5 Conclusiones y Discusión

De acuerdo a lo observado estos problemas tienen buenas soluciones si el dato está en un espacio vectorial de baja dimensión, no así para grandes espacios métricos. La complejidad de la mayoría de las técnicas existentes crecen exponencialmente con la dimensión, para $d > 25$ aproximadamente. Esto puede ser un problema cuando trabajamos con datos de alta dimensión.

Sergey Brin definió el término *"alta dimensión"* como: Espacios en donde pequeños incrementos en el rango de búsqueda producen grandes incrementos en el número de puntos dentro del rango.

Las investigaciones se dirigen al caso general donde el criterio de similaridad define un espacio métrico, en vez del caso más restrictivo del espacio vectorial. Se ha desarrollado un gran número de algoritmos para este problema en un espacio métrico, que se pueden clasificar de las dos formas siguientes:

Algoritmos que realizan un particionamiento estático del espacio:

Técnicas como la de BK-trees [BK73] y árboles métricos [Uhl91] donde el espacio es dividido jerárquicamente. En el nodo inicial se elige uno o varios puntos de datos. Luego se calculan las distancias entre el(los) punto(s) seleccionado(s) y el resto. Basados en estas distancias, los puntos son separados en 2 o más nodos diferentes, la estructura es construída recursivamente. Así surgen otros métodos como GNAT (Bri95), VP-tree (Yia93), FQ-trees (BCM+ 94) y M-tree (CPZ 97).

Algoritmos que realizan un particionamiento dinámico del espacio:

Algoritmos como los de Vidal [Vid85] y Nene-Nayar no suponen que los datos están estructurados dentro de un espacio vectorial, y sólo hacen uso de las propiedades de una distancia dada. La regla de descarte es esencialmente la desigualdad triangular.

Es nuestro propósito en un trabajo próximo estudiar estos algoritmos propuestos a fin de organizar las ideas conocidas e idear nuevos algoritmos a través la combinación de conceptos.

Referencias

[Ben75] J.L. Bentley. *Multidimensional Binary Search Trees Used for Associative Searching.* Communications of the ACM, 18(9) : 509-517, setiembre 1975.

[BCM+94] R. Baeza-Yates, W. Cunto, U.Mamber y S. Wu. *Proximity Matching Using Fixed-Queries Trees.* 5th Symp. Combinatorial Pattern Matching, Springer Verlag LNCS 807, pág. 198-212, junio 1994.

[BK73] W. A. Buckhard y R. M. Keller, *Some approaches to best-match file searching.* Communications of the ACM, vol. 16, Abril 1973.

[BKSS99] N. Berckmann, H. Kriegel, R. Schneider, B. Seeger.*The R*-tree: An Efficient and Robust Access Method for Points and Rectangles.*

[Bri95] S.Brin. *Near Neighbor Search in Large Metric Spaces*, Proceedings of the 21st VLDB Conference, pág 574-584, 1995.

[CPZ97] P. Ciaccia, M. Patella, P.Zezula. *M-tree : An efficient access method for similarity search in metric spaces.* Proceedings of the 23th International Conference on VLDB, pág. 426-435, 1997.

[FBF+94] C. Faloutsos, R. Barber, M.Flickner, J. Hafner, W. Niblack, D. Petkovic, y W.Equitz. *Efficient and effective querying by image content.* Journal of Intell. Inf. Systems, 3(3/4) :231-262, Julio 1994.

[FBY92] W.Frakes y R.Baeza-Yates. *Information Retrieval : Data Structures and Algorithms.* Prentice-Hall, 1992.

[FLL93] A. Faragó, T. Linder, G. Lugosi. *Fast nearest-neighbor search in dissimilarity spaces.* IEEE PAMI, 15(9), 1993.

[FP86] P. Flajolet, C. Puech. *Partial Match Retrieval of Multidimensional Data.* ACM Journal, vol. 33, pág. 371-407, Abril 1986.

[GG95] Volker Gaede y Oliver Guenther. *Survey on multidimensional access methods.* Technical Report ISS-16, Institut fuer Wirtschaftsinformatik, Humboldt- Universitaet zu Berlin, Agosto 1995.

[Gut84] A. Guttman. *R-trees : A Dynamic Index Structure for Spatial Searching.* ACM-SIGMOD, pág :47-57, Junio 1984.

[HS97] Gísli Hjaltason y Hanan Samet. *Distance Browsing in Spatial Databases Using R-trees*.* ..... , Febrero 1997.

[KF93] Ibrahim Kamel and Christos Faloutsos. *Hilbert R-tree: An Improved R-tree Using Fractals.* In Proceedings of VLDB Conferencie, pág. 500-509, Santiago, Chile, Setiembre 1994.

[Uhl91] J. Uhlmann. *Satisfying general proximity/similary queries with metric trees. Information Processing Letters.* 40(4) :175-9. November 1991.

[Yia93] P.N. Yianilos. *Data Structures and algorithms for nearest neighbor search in general metric spaces.* In ACM-SIAM Symposium on Discrete Algorithms, pág 311-321.1993.

[Vid85] E. Vidal. *An algorithm for finding nearest neighbours in (approximately) constant average time.* Patt. Recogn. Lett. 4(3), julio 1986.

[Sam94] Samet Hanan, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, p. 493, 1994

[Sed92] Robert Sedgewick, *"Algorithms in C++"*, ed. Addison-Wesley, 1992, p.672 ISBN 0-201-51059-6