Políticas de Selección de Pivotes

Norma E. Herrera
Departamento de Informática
Universidad de San Luis
nherrera@unsl.edu.ar

Anabella C. De Battista, Andrés J. Pascal
Departamento de Sistemas de Información
Universidad Tecnológica Nacional -
Regional Concepción del Uruguay
{debattistaa, pascalj}@frcu.utn.edu.ar

1. Introducción

Con la evolución de los sistemas y las tecnologías de información ha surgido la necesidad de procesamiento de datos no estructurados, o de consultas sobre datos estructurados, donde la búsqueda se realiza sobre cualquier campo o grupo de campos y no solo por clave, y a veces también, por partes de un campo. Para poder resolver estos nuevos desafíos se están desarrollando nuevos conocimientos, técnicas y herramientas, entre los cuales se encuentra la búsqueda por similitud en espacios métricos.

La busqueda por similitud (o por proximidad) intenta resolver el siguiente problema: "dado un conjunto de objetos de naturaleza desconocida, una función de distancia definida entre ellos que mide cuan diferentes son, y dado otro objeto, llamado la consulta, encontrar todos los elementos del conjunto suficientemente similares a la consulta". Este tipo de búsqueda se puede utilizar por ejemplo para recuperar objetos como imágenes, sonido, texto sin formato, y otros. También tiene aplicación en la biología computacional, en la predicción de funciones, el reconocimiento de patrones, y la minería de datos.

El conjunto X de todos los objetos sobre los cuales se puede realizar la búsqueda, mas la función de distancia d, se denomina espacio métrico, y se denota (X, d). Esto significa que debe ser posible definir una función que devuelva un número real positivo entre dos objetos cualesquiera del espacio, que representa cuan diferente es uno del otro. Dicha función definida como d: $X \times X \rightarrow R^+$ debe satisfacer las propiedades de positividad estricta $\forall x,y \in X$,(d(x,y)>=0 $\wedge$ d(x,y)=0 $\Rightarrow$ x=y), simetría $\forall x,y \in X$, (d(x,y) = d(y,x)) y desigualdad triangular $\forall$ x,y,z $\in X$, (d(x,z) $\leq$ d(x,y) + d(y,z)). Esta última propiedad, nos permite realizar algoritmos de búsqueda que descarten grupos de objetos sin necesidad de calcular su distancia a la consulta. Disminuir la cantidad de veces que se realiza este cálculo es fundamental para mejorar la eficiencia de la búsqueda, ya que en general, el cálculo de la función de distancia (por ejemplo la comparación de dos imágenes) tiene un costo mucho mas elevado que el tiempo extra de CPU o el tiempo de I/O. Llamaremos U al conjunto de objetos que posee la base de datos sobre la cual se realizan las consultas: $U \subseteq X$.

Existen distintos tipos de búsqueda por similitud. Los dos más comunes son la búsqueda por rango (range query) y la búsqueda de los k-vecinos más cercanos (Near Neighbors query). La búsqueda por rango $(q, r)_d$ consiste en encontrar todos los elementos de U cuya distancia a la consulta q, es menor o igual a r. En símbolos, $(q, r)_d = \{x \in U \mid d(q, x) \leq r\}$. En tanto que la búsqueda de los k-vecinos más cercanos $NN_k(q)$ consiste en recuperar los k elementos de U más cercanos a q. $NN_k(q)$ = A, donde |A|=k y $\forall x \in A, \forall y \in (U\text{-}A)$: d(q, x) $\leq$ d(q, y).

Las técnicas de búsqueda por similitud se basan en la construcción de índices especialmente diseñados para este tipo de objetos, que se utiliza para evitar la comparación de todos los elementos de la base de datos con la consulta. Los algoritmos de indización particionan la base de datos *U* en subconjuntos de elementos potencialmente relevantes a una consulta.

Existen dos enfoques para el diseño de algoritmos de construcción de estos índices: los basados en particiones campactas (de Voronoi) y los basados en pivotes [1]. Este trabajo trata sobre estos últimos.

Los algoritmos basados en pivotes definen una relación de equivalencia en función de la distancia de los elementos de *U* a un subconjunto de elementos seleccionados previamente denominados pivotes. Sea $\{p_1, p_2, ..., p_k\}$ el conjunto de pivotes, dos elementos de *U* son equivalentes, si y solo si, están a la misma distancia de todos los pivotes:

$$x \sim_{\{pi\}} y \Leftrightarrow d(x, p_i) = d(y, p_i), \ \forall i=1..k$$

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 calcula el vector de la query q, $\Phi(q)=(d(q, p_1), d(q, p_2), ..., d(q, p_k))$, y luego se descartan todos aquellos elementos x tales que para algún pivote p_i, $| d(q, p_i) - d(x, p_i) | > r$.

Existen algoritmos de indización que seleccionan los pivotes en forma aleatoria entre los objetos del espacio métrico. Sin embargo se sabe que la elección de los mejores pivotes disminuye drásticamente el tiempo de búsqueda [2, 3, 4]. Además, un grupo pequeño de pivotes bien elegidos puede tener mejor performance que un grupo más grande seleccionado al azar.

2. Estructuras basadas en pivotes

Se han estudiado varias estructuras basadas en pivotes para la búsqueda por proximidad en espacios métricos y sus algoritmos asociados, como por ejemplo Burkhard-Keller Tree (BKT) [5], Fixed-Queries Tree (FQT) [6], Fixed-Height FQT (FHQT) [6], Fixed Queries Array (FQA) [7], Vantage Point Tree (VPT) [8], Multi Vantage Point Tree (MVPT) [9], Excluded Middle Vantage Point Forest (VPF) [10], Approximating Eliminating Search Algorithm (AESA) [11], Linear AESA (LAESA) [12] y Spaghettis [13]. A continuación se describen las más importantes.

BKT (Burkhard-Keller Tree)

El Burkhard-Keller Tree (BKT) es un árbol que se construye recursivamente de la siguiente manera [14]: se selecciona un objeto arbitrario $p \in X$ como el nodo raíz del árbol, donde *X* es el conjunto de datos indizado. Para cada distancia $i \geq 0$, se define el conjunto X_i como el grupo de todos los objetos que están a distancia *i* de la raíz *p*, es decir, $X_i = \{o \in X: d(o, p) = i\}$. Luego se construye un nodo hijo de la raíz *p* por cada conjunto no vacío X_i. Los nodos hijos pueden ser reparticionados recursivamente. El proceso se repite hasta que quede un solo objeto en el subconjunto o hasta que queden menos de b objetos, los cuales se almacenan en un *bucke*t. Los objetos asignados como raíces de subárboles (guardados en nodos internos) son llamados *pivotes.*

El algoritmo para consultas por rango es muy simple. La búsqueda por rango *R(q, r)* comienza en el nodo raíz *p* del árbol y compara el objeto *p* con el objeto de consulta *q*. Si *p* satisface la consulta, esto es si $d(p, q) \leq r$, el objeto *p* es formará parte del resultado. Subsecuentemente, el algoritmo entra todos los nodos hijos o_i tales que $max\{d(p, q) - r, 0\} \leq i \leq d(p, q) + r$.

Los árboles BKT son lineales en espacio O(n). La complejidad de construcción es O(n logn), medida en términos del número de cálculos de distancia necesarios para construir el árbol. La complejidad del tiempo de búsqueda es $O(n^{\alpha})$ donde α es un número real que satisface $0 < \alpha < 1$ y depende del radio de búsqueda y la estructura del árbol.

FQT
Un desarrollo adicional sobre los BKTs es el "Fixed-queries tree" o FQTs [15]. Este árbol es básicamente un BKT donde todos los pivotes almacenados en los nodos del mismo nivel son iguales (y por supuesto no necesariamente pertenecen al conjunto almacenado en el subárbol). Todos los elementos reales están almacenados en las hojas. La ventaja de tal construcción es que se evitan algunas comparaciones entre la consulta y los nodos a lo largo del recorrido del árbol. Si visitamos muchos nodos del mismo nivel, no necesitamos ejecutar más que una comparación porque todos los pivotes en ese nivel son iguales. Bajo supuestos simplificados se muestra que los FQTs construidos sobre n elementos tiene altura O(log n) en promedio; se construyen usando O(n log n) evaluaciones de distancia; y el número promedio de cálculos de distancia es $O(n^{\alpha})$, donde $0 < \alpha < 1$ es un número que depende del rango de la búsqueda y de la estructura del espacio.

FHQT
En [15, 16] los autores proponen una variante llamada "Fixed-Height FQT" (o FHQT), donde todas las hojas están a la misma profundidad h, a pesar del costo del tamaño del bucket. Esto hace algunas hojas más profundas de lo necesario, lo que tiene sentido porque podemos manterner calculada la comparación entre la consulta y el pivote de un nivel intermedio, y por lo tanto eliminar la necesidad de considerar la hoja. En [16, 17] se muestra que al usar O(log n) pivotes, la búsqueda toma O(log n) evaluaciones de distancia (aunque el costo depende exponencialmente del radio de la búsqueda r).

FQA
En [12] se presenta el Fixed Queries Array o FQA. Esta estructura no es precisamente un árbol, sino una representación compacta de un árbol FHQT. Supongamos que hemos construido un FHQT sobre el conjunto de elementos; si barremos las hojas de izquierda a derecha, poniendo los elementos en un arreglo, obtenemos un FQA. Los algoritmos basados en pivotes asocian a cada elemento x del espacio métrico, el vector $(d(x, p_1); d(x, p_2); ...; d(x, p_k))$. Este vector se denomina *firma del elemento* x. Si cada número $d(x; p_i)$ se codifica en b bits, la firma de cada elemento se codifica en kb bits. En el FQA, se propone una representación compacta del conjunto de firmas, eliminando el árbol que utilizan la mayoría de las estructuras que le precedieron. Para lograrlo, la firma de cada elemento se considera como una secuencia de k enteros. La estructura simplemente almacena los elementos de la base de datos ordenados lexicográficamente por esta secuencia de distancias. Esto significa que, los elementos primero se ordenan respecto de su distancia al primer pivote; aquellos elementos que están a igual distancia del primer pivote se ordenan respecto de su distancia al segundo pivote, y así sucesivamente.
Todo barrido en el árbol puede simularse por medio de una búsqueda binaria, lo que agrega un costo O(log n) en tiempo extra de CPU. Sin embargo, usando la misma memoria, la simulación del FQA es capaz de usar muchos más pivotes que el FHQT original, lo que mejora la eficiencia.

3. Selección de pivotes

Un buen pivote, en el momento de realizar una consulta debe permitir eliminar la mayor cantidad posible de cálculos de la función de distancia entre la consulta y los objetos de la base de datos. Se

sabe que la selección de pivotes es uno de los factores principales que determinan la performance de la búsqueda [2, 3, 4].

Para efectuar dicha selección, se han propuesto técnicas basadas en maximizar la media de la distribución D (distancia entre particiones), donde $D([x], [y]) = \max_{1 \le i \le k} \{| d(x, p_i) - d(y, p_i)|\}$; p_i es el i-ésimo pivote y [x], [y] son las particiones correspondientes a los objetos x e y [4]. Estas técnicas son: selección de N grupos aleatorios, selección del óptimo local y selección incremental. De las tres, la última nombrada es la que posee mejor performance. En el estudio se muestra también que un buen grupo de pivotes debe tener dos características: los pivotes están alejados entre sí, y además, están alejados del resto de los elementos. Los objetos que cumplen con ambas propiedades se denominan *outliers*. Sin embargo, no todos los outliers son buenos pivotes.

En [19] se estudia una alternativa a la selección de pivotes generalizados. Para ello, se definen varios índices constituidos por grupos de pivotes elegidos al azar y, en tiempo de consulta, se realiza una optimización local a través de la selección del índice más adecuado a la consulta específica. Para realizar la selección, se pueden utilizar distintas políticas: selección por votos, el pivote más cercano, el pivote más lejano, menor masa total, pivote de menor masa y una combinación de todas. En los resultados experimentales se muestra que esta última posee el mejor desempeño, posiblemente debido a que logra una visión más amplia del espacio métrico.

4. Trabajo futuro

A continuación se resume el trabajo planificado a futuro:

- Adaptar las políticas de selección del índice adecuado a políticas de selección de pivotes. Si bien la problemática no es exactamente la misma, ya que en el primer caso la optimización es local a una consulta, esperamos que algunas de las estrategias utilizadas sean efectivas al generalizar el proceso a todas las consultas posibles.

- Efectuar experimentos con las nuevas políticas y evaluar los resultados con el fin de determinar su eficacia y de establecer nuevos criterios de selección.

Referencias

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

[2] A. Faragó, T. Linder, G. Lugosi. Fast nearest-neighbor search in dissimilarity spaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(9):957–962, 1993.

[3] L. Micó, J. Oncina, E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[4] B. Bustos, G. Navarro, E. Chávez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pp 33–40. IEEE CS Press, 2001.

[5] A. Faragó, T. Linder, G. Lugosi, Fast nearest-neighbor search in dissimilarity spaces, IEEE Trans. on Pattern Analysis and Machine Intelligence 15 (9) (1993) 957–962.

[6] R. S. Filho, A. Traina, C. T. Jr., C. Faloutsos, Similarity search without tears: The OMNI family of all-purpose access methods, in: ICDE, 2001, pp. 623–630.
[7] S. Brin, Near neighbor search in large metric spaces, in: Proc. 21st Conference on Very Large Databases (VLDB'95), 1995, pp. 574–584.
[8] P. Yianilos, Excluded middle vantage point forests for nearest neighbor search, in: DIMACS Implementation Challenge, ALENEX'99, Baltimore, MD, 1999.
[9] E. Chávez, 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), IEEE CS Press, 1999, pp. 38–46.
[10] E. Vidal, An algorithm for finding nearest neighbors in (approximately) constant average time, Pattern Recognition Letters 4 (1986) 145–157.
[11] T. Bozkaya, M. Ozsoyoglu, Distance-based indexing for high-dimensional metric spaces, in: Proc. ACM SIGMOD International Conference on Management of Data, 1997, pp. 357–368, Sigmod Record 26(2).
[12] 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) (2001) 113–135.
[13] 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), 1993, pp. 311–321.
[14] W. Burkhard and R. Keller. Some approaches to best-match file searching. Comm. Of the ACM, 16(4):230-236, 1973.
[15] 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.
[16] 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.
[17] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In Porc. 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98), pages 14-22. IEEE CS Press, 1998.
[18] E. Chávez, J. Marroquín and G. Navarro. Overcoming the curse of dimensionality. In European Workshop on Content-Based Multimedia Indexing (CBMI'99), pages 57-64, 1999.
[19] N. Herrera. Diseño e implementación de estructuras optimizadas para búsquedas en espacios métricos. *Tesis de Maestría en Ciencias de la Computación, Universidad Nacional de San Luis*, 2003.