Búsqueda en espacios métricos: Árbol de aproximación espacial dinámico

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

Gonzalo Navarro
Universidad de Chile, Chile
gnavarro@dcc.uchile.cl

1. Introducción y motivación

La búsqueda es un problema fundamental en Ciencias de la Computación, presente virtualmente en cada aplicación de computación. Las aplicaciones simples tienen problemas de búsqueda simples, mientras que las aplicaciones más complejas requerirán en general, una forma más sofisticada de búsqueda. Las bases de datos tradicionales se construyen basándose en el concepto de *búsqueda exacta*. Las consultas a la base de datos retornan todos aquellos registros cuyas claves coinciden con la aportada en la búsqueda. Las búsquedas mas sofisticadas como búsqueda de rangos sobre claves numéricas o búsqueda de prefijos sobre claves alfabéticas todavía se basan en el concepto que dos claves son o no son iguales, y en la existencia de un orden lineal sobre las claves de búsqueda.

Actualmente las bases de datos han incluido la capacidad de almacenar nuevos tipos de datos tales como imágenes, sonido, video, etc.. Estos tipos de datos son difíciles de estructurar para adecuarlos al concepto tradicional de búsqueda. Así, han surgido aplicaciones en grandes bases de datos en las que se desea buscar objetos *similares*. Este tipo de búsqueda se conoce con el nombre de *búsqueda aproximada* o *búsqueda por similitud*, y surge en áreas tales como reconocimiento de voz, reconocimiento de imágenes, etc. La necesidad de una respuesta rápida y adecuada, y un eficiente uso de memoria, hace necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos.

El planteo general del problema es: dado un conjunto $S \subseteq U$, recuperar los elementos de S que sean similares a uno dado, donde la similitud entre elementos es modelada mediante una función de distancia positiva d. U denota el universo de objetos *válidos* y S , un subconjunto finito de U, denota la *base de datos* en donde buscamos. El par (U, d) es llamado espacio métrico. La función $d: U \times U \rightarrow \mathbb{R}^+$ cumple con las propiedades propias de una función de distancia: $\forall x, y, z \in U$:

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

Básicamente, existen tres tipos de búsquedas de interés en espacios métricos:

a) Búsqueda por rango: recuperar todos los elementos de S que están a distancia r de un elemento q dado. Es decir, $\{x \in S / d(x, q) \leq r\}$.

b) Búsqueda del vecino más cercano: dado q, recuperar el (o los) elemento(s) más cercano(s) a q en S. Es decir, $\{x \in S / \exists y \in S, d(x, q) \leq d(y, q)\}$.

c) Búsqueda de los k vecinos más cercanos: dado q, recuperar los k elementos más cercanos a q en S. Es decir, recuperar el conjunto $A \subseteq S$ tal que $|A| = k$ y $\exists x \in A, y \in S - A, d(x, q) \leq d(y, q)$.

Un caso particular de espacios métricos son los espacios vectoriales K-dimensionales, con las funciones de distancia L_p (p=1,2, ..., ∞). Para estos casos existen soluciones bien conocidas. Sin embargo, éste no es el caso general.

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 S. En general las estructuras sobre las que trabajan dichos algoritmos suponen que los espacios son estáticos.

Existe una nueva y muy prometedora estructura para búsqueda por similitud en espacios métricos, también en los de dimensión alta, que es el *Árbol de Aproximación Espacial* (*SAT*) y que supone que el espacio es estático. Por lo tanto, nos proponemos estudiar distintas maneras de admitir en dicha estructura las inserciones y eliminaciones de elementos.

2. Árbol de Aproximación Espacial (SAT)

Para mostrar la idea general de la aproximación espacial se utilizan las consultas de tipo *b*). En este modelo se está ubicado en un elemento de S y se trata de obtener el más cercano “espacialmente”

a q (es decir, moverse a otro elemento que esté más cerca de q que el corriente). Cuando no se posible realizar esto más, se está posicionado en el elemento más cercano a q de S.

Estas aproximaciones son efectuadas sólo vía los "vecinos". Cada elemento $a \in S$ tiene un conjunto de vecinos $N(a)$. La estructura natural para representar esto es un grafo dirigido. Los nodos son los elementos del conjunto y los elementos vecinos son conectados por un arco.

La búsqueda procede sobre tal grafo simplemente posicionándose sobre un nodo arbitrario a y considerando todos sus vecinos. Si ningún nodo está más cerca de q que a, entonces se responde a a como el vecino más cercano a q. En otro caso, se selecciona algún vecino b de a que esté más cerca de q que a y se mueve a b.

Para obtener el SAT se simplifica la idea general comenzando la búsqueda en un nodo fijo y realmente se obtiene un árbol. El SAT está definido recursivamente; la propiedad que cumple la raíz a (y a su vez cada uno de los siguientes nodos) es que los hijos están más cerca de la raíz que de cualquier otro punto de S. En otras palabras: $\exists x \in S, x \in N(a) \leftrightarrow \exists y \in N(a) - \{x\}, d(x, y) > d(x, a)$.

La construcción del árbol se hace de manera recursiva pudiendo construirlo siguiendo una de dos estrategias posibles: *best-fit* y *first-fit*, de acuerdo a si cada elemento b que no está en $\{a\} \cup N(a)$ se ubicará en el subárbol del vecino más cercano a él de $N(a)$ o en el primero que sea más cercano a él que a.

De la definición de SAT se observa que se necesitan de antemano todos los elementos de la base de datos para la construcción. Por lo tanto, tiene sentido que nos planteemos el estudio de cuán posible es llevar dicha estructura a que admita la inserción y eliminación, sin degradar los tiempos de búsqueda.

2.1. Inserciones y eliminaciones en SAT

Estamos estudiando distintas formas de poder realizar inserciones y eliminaciones de elementos en el árbol, y mediante pruebas experimentales analizaremos tanto el costo de efectuar la inserción/eliminación de elementos como así también el impacto en las búsquedas luego de efectuar tales operaciones.

A continuación se muestran una serie de alternativas que estamos considerando para cada una de las operaciones, dependiendo de la estrategia con la que se construyó el árbol. Es posible que algunas de ellas finalmente se descarten o que surjan del estudio otras posibilidades.

2.1.1. Inserciones

Cada vez que un nuevo elemento es insertado, se debe bajar en el árbol hasta que el nuevo elemento se deba volver un vecino del nodo corriente, por estar más cerca de él que de cualquiera de sus vecinos.

2.1.1.1 Best-fit

Estamos actualmente analizando las siguientes opciones:

a) reconstruir el subárbol.
b) crear un bucket de overflow en cada nodo del árbol, demorando así la reconstrucción; pero ante una búsqueda se debe comparar contra todos los elementos del bucket.
c) se puede relajar la condición de ser vecino e, incluso cuando quiera serlo, colocarlo en el más cercano de los vecinos que ya existen. El no estar obligado a ponerlo como vecino abre muchas posibilidades:

   c.1) Solo poner a alguien como vecino cuando quiere serlo y el subárbol que hay que rearmar no tiene más de H nodos.

   c.2) Cuando quiere ser vecino de un elemento x, separar en A el conjunto de hijos de x que no querrán ser hijos del nuevo elemento z, y B el de hijos de x que sí querrán ser hijos de z. Si se puede determinar que todo hijo de b (b es elemento de B) también querrá ser hijo de z, entonces b y su subárbol pueden pasarse enteros dentro de z. Si el total de los subárboles que no cumplen con esto no suma más que un threshold H, poner a z como vecino y reconstruir sólo a ellos.

   c.3) Se puede tener una aridad máxima de nodo, si es que es útil para, por ejemplo, asegurar que entra todo en una página en memoria secundaria.

   c.4) Se pueden modificar decisiones anteriores acerca de vecinos ya existentes. Por ejemplo, si después surge un mejor vecino para la raíz, puede convertirse en vecino si resulta razonable el

costo (ver c.2). Esto permite, en el caso dinámico, recuperarse del hecho de que los mejores vecinos no se conocen de entrada.

d) colocarlo como último vecino guardando un "timestamp" que indique el momento de inserción, sin reconstruir el árbol, y en la búsqueda tener en cuenta el valor de dicho "timestamp" para cortar algunas ramas de búsqueda.

Además, se debe considerar que en una búsqueda demasiados vecinos implica mucho costo para bajar pero demasiado pocos implica que se baja demasiado para llegar a lo que se busca.

2.1.1.2 First-fit

De acuerdo a esta estrategia se puede simplemente agregar el elemento a insertar como último vecino del nodo corriente. Así, se podría construir el árbol por sucesivas inserciones. Se ha mostrado experimentalmente que la estrategia *first-fit* trabaja mucho peor en la práctica debido a su asimetría. Por lo tanto, aunque esta solución es elegante, no es realmente prometedora.

2.1.2. Eliminaciones

Las eliminaciones son mucho más complicadas porque los cambios a realizar en la estructura pueden ser muy costosos, y es posible que ellos no sean localizados y se deban propagar al resto de la estructura.

Al eliminar un elemento tenemos en general tres opciones posibles, independientemente de la estrategia de construcción elegida:

a) Marcar el elemento como eliminado y demorar la reconstrucción de la estructura.

b) Buscar un candidato para reemplazar el elemento eliminado de manera tal de minimizar las modificaciones posteriores a la estructura.

c) Incorporar a los vecinos del elemento eliminado como vecinos de su padre.

d) De acuerdo a alguna estimación del costo de la eliminación evaluar dinámicamente si conviene demorar el cambio en la estructura o no.

2.1.2.1. Best-Fit

Para el caso de la opción *b*) el candidato que parece surgir naturalmente para reemplazar al elemento eliminado *x* sería el más cercano de sus vecinos *z*. En ese caso se mantendrían, en comienzo, como vecinos de *z* los otros vecinos de *b* y los vecinos de *z*. El inconveniente que aún observamos es que se tendrían que analizar todos los elementos en los vecinos de *b* para verificar que estén bien ubicados de acuerdo a este cambio, lo cual puede ser excesivamente costoso. Una mejora se conseguiría haciendo uso del covering radius de los nodos para evitar así tener tantas evaluaciones de distancia.

Para el caso de la opción *c*) el trabajo a realizar parecería ser un poco menor, pero aún así es costoso. Aquí también puede lograrse alguna mejora haciendo uso del covering radius.

2.1.2.2. First-Fit

Para la opción *b*) estamos analizando dos posibilidades simples: elegir el primero de sus vecinos ó el último de sus vecinos. En ambos casos se deberían verificar todos los elementos de la vecindad del elemento eliminado para asegurarse que estén bien ubicados, además es posible que al elegir como reemplazo el último de los vecinos puedan aparecer nuevos vecinos. Nuevamente aquí se puede bajar el costo usando el covering radius.

3. Bibliografía

- E. Chávez, J. Marroquín and G. Navarro. *Overcaming the curse of dimensionality*. In European Workshop on Content-Based Multimedia Indexing (CBMI '99), pages 57-64, 1999.
- 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.
- G. Navarro. *Searching in metrics spaces by spatial aproximation*. In Procs. String Proccessing and Information Retrieval (SPIRE'99), pages 141-148. IEEE CS Press, 1999.