Combinando Clustering con Aproximación Espacial para Búsquedas en Espacios Métricos *

Marcelo Barroso [1] **Gonzalo Navarro** [2] **Nora Reyes** [1]

Departamento de Informática, Universidad Nacional de San Luis.[1]
Departamento de Ciencias de la Computación, Universidad de Chile. [2]
mabarros@unsl.edu.ar gnavarro@dcc.uchile.cl nreyes@unsl.edu.ar

Resumen

El *modelo de espacios métricos* permite abstraer muchos de los problemas de búsqueda por proximidad. La búsqueda por proximidad tiene múltiples aplicaciones especialmente en el área de bases de datos multimedia. La idea es construir un índice para la base de datos de manera tal de acelerar las consultas por proximidad o similitud. Aunque existen varios índices prometedores, pocos de ellos son dinámicos, es decir, una vez creados muy pocos permiten realizar inserciones y eliminaciones de elementos a un costo razonable.

El *Árbol de Aproximación Espacial* (*dsa–tree*) es un índice recientemente propuesto, que ha demostrado tener buen desempeño en las búsquedas y que además es totalmente dinámico. En este trabajo nos proponemos obtener una nueva estructura de datos para búsqueda en espacios métricos, basada en el *dsa–tree*, que mantenga sus virtudes y que aproveche que en muchos espacios existen clusters de elementos y que además pueda hacer un mejor uso de la memoria disponible para mejorar las búsquedas.

1. Introducción y motivación

Con la evolución de las tecnologías de información y comunicación, han surgido almacenamientos no estructurados de información. No sólo se consultan nuevos tipos de datos tales como texto libre, imágenes, audio y video; sino que además, en algunos casos, ya no se puede estructurar más la información en claves y registros. Aún cuando sea posible una estructuración clásica, nuevas aplicaciones tales como la minería de datos requieren acceder a la base de datos por cualquier campo y no sólo por aquellos marcados como "claves". 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 tiene aplicaciones en un amplio número de campos. Algunos ejemplos son bases de datos no tradicionales; búsqueda de texto; recuperación de información; aprendizaje de máquina y clasificación; sólo para nombrar unos pocos.

Como en toda aplicación que realiza búsquedas, surge la necesidad de tener una respuesta rápida y adecuada, y un uso eficiente de memoria, lo que hace necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos.

El planteo general del problema es: existe un universo $\mathbb{U}$ de *objetos* y una función de distancia positiva d: $\mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}^+$ definida entre ellos. Esta función de distancia satisface los tres axiomas

*Este trabajo ha sido financiado parcialmente por el Proyecto RIBIDI CYTED VII.19 (todos los autores) y por el Centro del Núcleo Milenio para Investigación de la Web, Grant P01-029-F, Mideplan, Chile (último autor).

que hacen que el conjunto sea un *espacio métrico*: positividad estricta ($d(x,y) = 0 \Leftrightarrow x = y$), simetría ($d(x,y) = d(y,x)$) y desigualdad triangular ($d(x,z) \leq d(x,y) + d(y,z)$). Mientras más "similares" sean dos objetos menor será la distancia entre ellos. Tenemos una *base de datos* finita $S \subseteq \mathbb{U}$, que es un subconjunto del universo de objetos y puede ser preprocesada (v.g. para construir un índice). Luego, dado un nuevo objeto del universo (un *query* q), debemos recuperar todos los elementos similares que se encuentran en la base de datos. Existen dos consultas típicas de este tipo:

Búsqueda por rango: recuperar todos los elementos de S que están a distancia r de un elemento q dado.

Búsqueda de k vecinos más cercanos: dado q, recuperar los k elementos más cercanos a q en S.

La distancia se considera costosa de evaluar (por ejemplo, comparar dos huellas dactilares). Así, es usual definir la complejidad de la búsqueda como el número de evaluaciones de distancia realizadas, dejando de lado otras componentes tales como tiempo de CPU para computaciones colaterales, y aún tiempo de E/S. Dada una base de datos de $|S| = n$ objetos el objetivo es estructurar la base de datos de forma tal de realizar menos de n evaluaciones de distancia (trivialmente n bastarían).

Un caso particular de este problema surge cuando el espacio es un conjunto D-dimensional de puntos y la función de distancia pertenece a la familia L_p de Minkowski: $L_p = (\sum_{1 \leq i \leq d} |x_i - y_i|^p)^{1/p}$. Existen métodos efectivos para buscar sobre espacios D-dimensionales, tales como kd-trees [1] o R-trees [5]. Sin embargo, para 20 dimensiones o más esas estructuras dejan de trabajar bien.

Nos dedicamos en este trabajo a espacios métricos generales, aunque las soluciones son también adecuadas para espacios D-dimensionales. Es interesante notar que el concepto de "dimensionalidad" se puede también traducir a espacios métricos: la característica típica en espacios de alta dimensión con distancias L_p es que la distribución de probabilidad de las distancias tiene un histograma concentrado, haciendo así que el trabajo realizado por cualquier algoritmo de búsqueda por similaridad sea más dificultoso [2, 4]. Para espacios métricos generales existen numerosos métodos para preprocesar la base de datos con el fin de reducir el número de evaluaciones de distancia [4]. Todas aquellas estructuras trabajan básicamente descartando elementos mediante la desigualdad triangular, y la mayoría usa la técnica dividir para conquistar.

El Árbol de Aproximación Espacial Dinámico (*dsa–tree*) es una estructura de esta clase propuesta recientemente [7], basado sobre un nuevo concepto: más que dividir el espacio de búsqueda, aproximarse al query espacialmente, y además completamente dinámica. El dinamismo completo no es común en estructuras de datos métricas [4]. Además de ser desde el punto de vista algorítmico interesante por sí mismo, se ha mostrado que el *dsa-tree* da un buen balance espacio-tiempo respecto de las otras estructuras existentes sobre espacios métricos de alta dimensión o consultas con baja selectividad, lo cual ocurre en muchas aplicaciones.

A diferencia de algunas otras estructuras de datos métricas [3], el *dsa–tree* no saca mayor provecho si el espacio métrico posee clusters, ni puede mejorar las búsquedas a costa de usar más memoria.

En este trabajo nos proponemos obtener una nueva estructura de datos para búsqueda en espacios métricos, basada en el *dsa-tree*, que mantenga sus virtudes, pero que aproveche que en muchos espacios existen clusters de elementos y que además pueda hacer un mejor uso de la memoria disponible para mejorar las búsquedas.

2. Árbol de Aproximación Espacial Dinámico

Describiremos brevemente aquí la aproximación espacial y el *dsa-tree*, para más detalles ver [6, 7].

Se puede mostrar la idea general de la aproximación espacial utilizando las *búsquedas del vecino más cercano*. En este modelo, dado un punto $q \in \mathcal{U}$ y estando posicionado en algún elemento $a \in \mathcal{S}$

el objetivo es moverse a otro elemento de $\mathcal{S}$ que esté más cerca "espacialmente" de q que a. Cuando no es posible realizar más este movimiento, 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 $N(a)$.

Para construir incrementalmente al *dsa–tree* se fija una aridad máxima para el árbol y se mantiene información sobre el tiempo de inserción de cada elemento. Cada nodo a en el árbol está conectado con sus hijos, los cuales forman el conjunto $N(a)$, los *vecinos* de a. Cuando se inserta un nuevo elemento x, se ubica su punto de inserción comenzando desde la raíz del árbol a y realizando el siguiente proceso. Se agrega x a $N(a)$ (como una nueva hoja) si (1) x está más cerca de a que de cualquier elemento $b \in N(a)$, y (2) la aridad del nodo a, $|N(a)|$, no es ya la máxima permitida. En otro caso, se fuerza a que x a elegir el vecino más cercano en $N(a)$ y se continua bajando en el árbol recursivamente, hasta que se alcance un nodo a tal que x esté más cerca de a que de cualquier $b \in N(a)$ y la aridad de a no haya alcanzado la máxima permitida, lo que eventualmente ocurrirá en una hoja del árbol. En ese punto se agrega a x como el vecino más nuevo en $N(a)$, se le adjunta a x la marca del tiempo corriente y éste se incrementa. En cada nodo a del árbol se mantiene la siguiente información: el conjunto de vecinos $N(a)$, la información del tiempo de inserción del nodo $tiempo(a)$ y el radio de cobertura $R(a)$ que es la distancia entre a y el elemento de su subárbol que está más lejos de a.

El *dsa–tree* se puede construir comenzando con un único nodo a donde $N(a) = \emptyset$ y $R(a) = 0$, y luego realizando sucesivas inserciones.

La idea de la búsqueda por rango es replicar el proceso de inserción de los elementos relevantes para la consulta. Es decir, se procede como si se quisiera insertar q pero considerando que los elementos relevantes pueden estar a distancia hasta r de q, así en cada decisión al simular la inserción de q se permite una tolerancia de $\pm r$, por lo tanto puede ser que los elementos relevantes fueran insertados en diferentes hijos del nodo corriente, y es necesario hacer backtracking. Las búsquedas se optimizan haciendo uso de la información almacenada sobre el tiempo de inserción y el radio de cobertura.

Las eliminaciones son más complicadas porque los cambios a realizar en la estructura son más costosos, y no son localizados. Sin embargo, dado que se tiene la información sobre el tiempo de inserción de cada nodo, al eliminar un nodo x de $N(a)$ en el árbol, se pueden recuperar los nodos insertados en el subárbol de a posteriormente a la inserción de x y luego reinsertarlos desde a, manteniendo el árbol como si x nunca hubiese existido.

3. Nuestra Propuesta

El *dsa–tree* es una estructura que realiza la partición del espacio considerando la proximidad espacial, pero si el árbol lograra agrupar los elementos que se encuentran muy cercanos entre sí, lograría mejorar las búsquedas al evitarse el recorrido del árbol para alcanzarlos. Así, nos hemos planteado el estudio de una nueva estructura de datos que realice la aproximación espacial sobre clusters o grupos de elementos, en lugar de elementos individuales.

Podemos pensar entonces que construimos un *dsa–tree*, con la diferencia que cada nodo representa un grupo de elementos muy cercanos ("clusters"); y de este modo, logramos relacionar los clusters por su proximidad en el espacio. Por lo tanto, cada nodo de la estructura sería capaz de almacenar varios elementos de la base de datos. La idea sería que en cada nodo se mantenga un elemento, al que se toma como el centro del cluster correspondiente, y se almacenen los k elementos más cercanos a él; cualquier elemento a mayor distancia del centro que los k elementos, pasaría a formar parte de otro nodo en el árbol, que podría ser un nuevo vecino en algunos casos.

Al intentar insertar un nuevo elemento en un nodo cuyo cluster ya tiene sus k elementos presentes, debemos decidir cuáles son los k elementos más cercanos al centro que deberían quedar en el cluster. Entonces, para cada elemento se podrían almacenar esas distancias y mantener los elementos del cluster ordenados por distancia al centro; evitando así recalcular distancias para decidir quien queda y, por consiguiente, quien sale del cluster.

Como en el *dsa–tree* para cada nodo n, se mantiene el radio de cobertura $R(n)$, la información del tiempo de creación del nodo $tiempo(n)$, el conjunto de vecinos del nodo $N(n)$ y además, la distancia entre su centro a y el elemento más alejado de su cluster, es decir el radio del cluster $rc(a)$. Para cada elemento a en el árbol se mantiene su tiempo de inserción $tiempo(a)$. Durante las búsquedas, se pueden utilizar ambos radios para permitirnos descartar zonas completas del espacio.

Para insertar un nuevo elemento x en el árbol, por la aproximación espacial, deberíamos bajar por el árbol hasta encontrar el nodo n tal que x esté más cerca de su centro a que de los centros de los nodos vecinos en $N(n)$. Si en el cluster de ese nodo hay lugar para un elemento más, se lo insertaría junto con su distancia $d(x, a)$. Si no hay lugar, elegiríamos el elemento más distante b entre los k elementos del cluster y x, es decir el $k + 1$ en el orden de distancias con respecto al centro a. A continuación deberíamos analizar dos casos posibles:

Si b es x: x se debería agregar como centro de un nuevo nodo vecino de n, si la aridad de n lo permite; en otro caso, debería elegir el nodo cuyo centro, entre todos los nodos vecinos en $N(n)$, es el más cercano y continuar el proceso de inserción desde allí.

Si b es distinto de x: b debería elegir al centro más cercano c entre a y los centros de los nodos vecinos en $N(n)$ que sean más nuevos que b, debido a que cuando b se insertó no se tuvo que comparar con ellos. Luego, si c es a, el proceso que sigue es idéntico a lo realizado cuando b es x; en otro caso, si c no es a, se continúa con la inserción de b desde el nodo cuyo centro es c.

Inicialmente, el primer elemento insertado se tomará como el centro del nodo raíz. Los siguientes k elementos se tomarán como los k objetos del cluster del nodo raíz y que luego, al insertar el $k + 2$, recién se creará un nuevo nodo cuyo centro será el elemento desalojado del cluster del nodo raíz.

Teniendo en cuenta la manera en que se realizan las inserciones, es posible observar que ninguna inserción nos modificaría el centro de un cluster y además que es posible que durante una inserción se cree a lo sumo un nuevo nodo y que ésta afecte posiblemente a varios nodos.

La búsqueda de un elemento q con radio r debería proceder de manera similar al *dsa–tree*, es decir realizando aproximación espacial entre los centros de los nodos. Sin embargo, al tener clusters en los nodos debemos además verificar si hay o no intersección entre la zona consultada y el cluster. Más aún, si no la hay se pueden descartar todos los elementos del cluster, sin necesidad de compararlos contra q. Si el cluster no se pudo descartar para la consulta, es posible usar al centro de cada nodo como un pivote para los elementos x_i que se encuentran en el cluster, ya que junto mantenemos las distancias $d(a, x_i)$ respecto del centro a. De esta manera es posible que, evitemos algunos cálculos de distancia entre q y los x_i, si $|d(q, a) - d(a, x_i)| > r$.

Cabe destacar que si la zona consultada cae completamente dentro de un cluster, podemos estar seguros que en ninguna otra parte del árbol encontraremos elementos relevantes para esa consulta.

En el caso de la eliminación de un elemento x debemos considerar los siguientes casos posibles:

a) Si x no es un centro de un nodo, entonces simplemente se lo elimina.

b) Si x es centro de un nodo n:

1. Si el cluster de n no tiene elementos, entonces se elimina a n y por consiguiente a x.

2. Si el cluster de n tiene elementos, se coloca como nuevo centro al objeto más cercano a x del cluster; pero, desde ese momento, debe tenerse en cuenta la brecha que existe entre el nuevo centro y x para las operaciones posteriores en el árbol que involucren a este nodo.

4. Trabajo Futuro

Esperamos poder evaluar el comportamiento de la estructura propuesta experimentando sobre distintos espacios métricos. Además pretendemos hacer comparaciones contra estructuras como el *dsa–tree*, así como contra la estructura de *Lista de Clusters* propuesta en [3]. También queremos analizar, entre otras, cuestiones tales como:

- Dado que los nodos tienen tamaño fijo, esta estructura parecería adecuada para memoria secundaria. Pero, ¿sería realmente eficiente en memoria secundaria?
- ¿Es posible mantener el cluster separado del nodo, y permitir que en cada nodo se almacene el centro, con su información asociada y los centros vecinos? ¿Sería una mejor opción para memoria secundaria?
- ¿Sería más adecuado mantener los clusters con cantidad fija de elementos o con radio fijo?; es decir, mantener en el cluster todos aquellos elementos que estén a lo sumo a una distancia prefijada del centro.
- Como almacenamos en cada clusters los elementos más cercanos al centro, ¿es posible que en muchos nodos no se alcance la aridad máxima permitida? Si es así, ¿sería mejor permitir aridad ilimitada en el árbol?
- ¿Existen otras maneras de combinar técnicas de clustering con aproximación espacial para búsquedas en espacios métricos?

Referencias

[1] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering*, 5(4):333–340, 1979.

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

[3] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*. To appear.

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

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

[6] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[7] G. Navarro and N. Reyes. Fully dynamic spatial approximation trees. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, LNCS 2476, pages 254–270. Springer, 2002.