Búsquedas en Bases de Datos no Convencionales *

Diego Arroyuelo, Verónica Ludueña y Nora Reyes - {darroy,vlud,nreyes}@unsl.edu.ar
Dpto. de Informática - Universidad Nacional de San Luis - Tel.: 02652-420822-257 - Fax: 02652-430224

Gonzalo Navarro - gnavarro@dcc.uchile.cl
Dpto. de Ciencias de la Computación - Universidad de Chile - Tel.: +56-2-6892736 - Fax: +56-2-6895531

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".

Los escenarios anteriores requieren modelos más generales tales como *bases de datos de texto* o *espacios métricos*, entre otros; y contar con herramientas que permitan realizar búsquedas eficientes sobre estos tipos de datos. Las técnicas que emergen desde estos campos muestran un área de investigación propicia para el desarrollo de herramientas que resuelvan eficientemente los problemas involucrados en la administración de bases de datos no convencionales.

Como los problemas han aparecido en áreas muy diversas, las soluciones también han surgido desde muchos campos no relacionados. Algunos ejemplos son bases de datos no convencionales (donde se buscan objetos similares); aprendizaje de máquina y clasificación (donde se debe clasificar un nuevo elemento por su elemento más cercano existente); cuantización y compresión de imágenes (donde sólo algunos vectores pueden ser representados y, aquéllos que no, deben ser codificados como su punto representativo más cercano); recuperación de texto (donde se buscan palabras permitiendo un pequeño número de errores, o se buscan documentos que sean similares a un documento dado, o se buscan todas las apariciones de un determinado patrón); etc.

La búsqueda por similitud es un tema de investigación que abstrae varias nociones de las ya mencionadas. Este problema se puede expresar como sigue: dado un conjunto de objetos de naturaleza desconocida, una función de distancia definida entre ellos, que mide cuán diferentes son, y dado otro objeto, llamado la consulta, encontrar todos los elementos del conjunto suficientemente similares a la consulta. El conjunto de objetos junto con la función de distancia se denomina espacio métrico.

En algunas aplicaciones, los espacios métricos resultan ser de un tipo particular llamado "espacio vectorial", donde los elementos consisten de D coordenadas de valores reales. Existen muchos trabajos que explotan las propiedades geométricas sobre espacios vectoriales (ver [GG98] para más detalles); pero normalmente éstas no se pueden extender a los espacios métricos generales.

Se han logrado algunos avances importantes para espacios métricos generales, en su gran mayoría alrededor de la idea de construir un índice que reduzca el número de evaluaciones de distancia durante la consulta. En muchas aplicaciones, aunque es muy importante reducir el número de evaluaciones de la distancia, también es importante reducir la cantidad de operaciones de E/S realizadas. Algunos trabajos recientes persiguen este doble objetivo (por ejemplo ver [CPZ97, Ver95]).

Por otra parte, una base de datos de texto es un sistema que debe proveer acceso eficiente a grandes volúmenes de texto no estructurado, donde existe la necesidad de construir índices que no sólo permitan realizar búsquedas

*Este trabajo ha sido fi nanciado parcialmente por el Proyecto RIBIDI CYTED VII.19 RIBIDI (todos los autores) y por el Centro del N´ucleo Milenio para Investigaci´on de la Web, Grant P01-029-F, Mideplan, Chile (´ultimo autor).

eficientes de patrones ingresados por el usuario, sino que además usen tan poco espacio como sea posible. En el escenario más simple, el texto se ve como una secuencia de símbolos y el patrón a buscar como otra secuencia más breve, y así el problema de búsqueda consiste en encontrar todas las apariciones del patrón en el texto.

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. En particular, nos vamos a dedicar a dos tipos de bases de datos no convencionales: los *Espacios Métricos* y las *Bases de Datos de Texto*, y cómo resolver eficientemente las búsquedas en esos ámbitos.

Existen indices que, en principio, resuelven ambos tipos de problemas; pero aún están muy inmaduros para ser usados en la vida real por dos motivos importantes:

- falta de dinamismo,
- necesidad de trabajar en memoria principal.

Estas caracteristicas son sobreentendidas en bases de datos tradicionales, y la investigación apunta a poner estas nuevas bases de datos a un nivel de madurez similar.

2 Espacios Métricos

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

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

- **Búsqueda por rango:** recuperar todos los elementos de $\mathcal{S}$ que están a distancia r de un elemento q dado.
- **Búsqueda de los k vecinos más cercanos:** dado q, recuperar los k elementos más cercanos a q.

En muchas aplicaciones, la evaluación de la función d, suele ser una operación costosa. Es por esta razón, que en la mayoría de los casos la cantidad de evaluaciones de distancias necesarias para resolver la búsqueda, se usa como medida de complejidad . En otros casos, cuando el índice deberá alojarse en memoria secundaria, es necesario prestar atención también a los tiempos involucrados en la E/S.

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 $\mathcal{S}$. En general las estructuras sobre las que trabajan dichos algoritmos suponen que los espacios son estáticos.

Los algoritmos de búsqueda en espacios métricos generales se dividen en dos grandes áreas: *algoritmos basados en pivotes* y *algoritmos basados en particiones compactas* (ver [CNBYM01] para un análisis detallado).

Una de las estructuras que ha mostrado tener un buen comportamiento en espacios de alta dimensión o consultas de baja selectividad es el *Árbol de Aproximación Espacial* (*SAT*), ver [Nav02b]. El *SAT* es una estructura de datos totalmente estática y nuestro objetivo fue estudiarlo, a fin de lograr una estructura completamente dinámica que realice eficientemente las búsquedas.

2.1 Árbol de Aproximación Espacial

Para mostrar la idea general de la aproximación espacial se utilizan 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 $a \in \mathcal{S}$ 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 está 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. La construcción del árbol se hace de manera recursiva. De la definición se observa que se necesitan de antemano todos los elementos de la base de datos para la construcción.

Luego de haber analizado distintas formas de realizar inserciones de elementos en el árbol, obtuvimos un nuevo método que permite realizar la construcción incremental de *SAT* a un costo muy inferior a la versión estática y que mantiene el costo de las búsquedas en la mayoría de los casos menor, y en otros levemente mayor que la versión original. En [NR02, Rey02] se presentó una versión completamente dinámica del *SAT*. El *Árbol de Aproximación Espacial Dinámico* (*SATD*) construye un árbol con *aridad acotada* y guarda en cada nodo un *timestamp* que marca el tiempo en que se insertó dicho elemento en el árbol.

Al incorporar un nuevo elemento x, éste se inserta en el punto más apropiado, si es que no se supera la cota de la aridad; en otro caso se continúa la búsqueda del lugar de inserción a partir del vecino más cercano. Sea b el punto resultante para la inserción, luego x se agrega como el vecino *más nuevo* de b y no se efectúa ningún tipo de reconstrucción. Ahora claramente, no se fuerza a un elemento a aparecer en la vecindad del más cercano. En tiempo de búsqueda alcanzaremos al elemento insertado porque los procesos de inserción y búsqueda son similares. Las búsquedas se pueden optimizar haciendo uso del *timestamp*.

Las eliminaciones son mucho más complicadas porque los cambios a realizar en la estructura son más costosos, y no son localizados, sino que se pueden propagar al resto de la estructura.

Los *SATD* son estructuras de datos adecuadas para búsquedas en espacios métricos de alta dimensionalidad o radios de búsqueda grandes, tienen también un desempeño razonable para búsquedas en espacios de baja dimensionalidad (si la aridad del árbol es sintonizada correctamente). Sin embargo, dado un conjunto de n elementos, utilizan un espacio fijo de almacenamiento, y aunque se disponga de más espacio disponible no son capaces de aprovecharlo para beneficio de las búsquedas.

Por otro lado, las estructuras basadas en pivotes se caracterizan principalmente por poder adaptarse al espacio de almacenamiento disponible y lograr mejorar su comportamiento durante las búsquedas. En otras palabras, si hay más espacio disponible se utilizan más pivotes por cada elemento de la base de datos, y se mejora el costo de búsqueda.

Buscando lo mejor de ambos métodos, hemos diseñado el *Árbol de Aproximación Espacial Dinámico con pivotes agregados* (*SATDP*), que es una estructura de datos híbrida, basada en los conceptos de aproximación espacial y algoritmos de pivotes. La idea general es permitir que el *SATDP* pueda hacer buen uso del espacio disponible almacenando pivotes en cada nodo y lograr una mejora en el costo de búsqueda.

De esta manera, al momento de inserción de un elemento x en el árbol se calculan las distancias entre x y sus pivotes, almacenando esas distancias en el nodo de x. Esto es para que esa información pueda ser usada durante las búsquedas. Hasta el momento hemos trabajado con dos formas de elegir los pivotes de cada nodo:

- Usar los ancestros de cada nodo como pivotes del mismo.
- Usar los ancestros de cada nodo, más los hermanos mayores de los ancestros como pivotes de cada nodo.

De esta manera el costo de inserción de cada elemento en el *SATDP* es el mismo que para el *SATD* original; simplemente almacenamos en cada nodo algunas de las distancias computadas durante la inserción de cada elemento. Durante las búsquedas se usan estas distancias almacenadas en cada nodo del árbol para disminuir la cantidad de evaluaciones de distancia. La idea es proceder con los criterios de aproximación espacial sólo con aquellos nodos del árbol que no puedan ser descartados usando los pivotes. El descarte por pivotes se produce sin evaluaciones de distancia adicionales, lo que ayuda disminuir la cantidad total de evaluaciones de distancia realizadas durante las búsquedas.

3 Bases de Datos de Texto

Una *base de datos de texto* es un sistema que provee acceso eficiente a amplias masas de datos textuales. El requerimiento más importante es que desarrolle búsquedas rápidas para patrones ingresados por el ususario. El escenario más simple es como sigue: el texto $T_{1...u}$ se ve como una única secuencia de caracteres sobre un alfabeto Σ de tamaño σ, y el patrón de búsqueda $P_{1...m}$ como otra (breve) secuencia sobre Σ. Luego el problema de búsqueda de texto consiste en encontrar todas las R ocurrencias de P en T.

Las bases de datos de texto modernas tienen que enfrentar dos objetivos opuestos. Por un lado, tienen que proveer acceso rápido al texto y por el otro, tienen que usar tan poco espacio como sea posible. Los objetivos son opuestos debido a que para proveer acceso rápido se debe construir un índice sobre el texto. Un índice es una estructura de datos construida sobre el texto y almacenada en la base de datos y así incrementa los requerimientos de espacio. Recientemente se ha investigado mucho sobre *bases de datos de texto comprimido*, enfocándose en comprimirlo y, de ser posible, hacerlo de tal forma que las estructuras que representan al texto comprimido nos sirvan también para buscar en él. Un ejemplo de este tipo de índice es el *LZ-Index*.

3.1 LZ-Index

Para describir el *LZ-Index*, y como está basado en el *Ziv-Lempel Trie*, daremos una breve reseña del mismo. La idea de la compresión *Ziv-Lempel* se basa en reemplazar substrings en el texto por punteros a apariciones anteriores de los mismos. Si el puntero ocupa menos espacio que el substring que él reemplaza, entonces se logró compresión. Existen distintas variantes de este tipo de compresión, aquí se verá sólo el *LZ78*.

El *LZ78* (algoritmo de compresión *Ziv-Lempel* de 1978) se basa en un diccionario de bloques. Supongamos que tenemos el texto T de longitud u para comprimir, al principio del proceso el diccionario posee un único bloque b_0 de longitud 0. Luego, suponiendo que ya se comprimió el prefijo $T_{1...j}$ de T en una secuencia de bloques $Z = b_1, \ldots, b_r$, todos ellos en el diccionario; el paso que sigue es buscar el prefijo más largo del texto $T_{j+1...u}$ que es un bloque del diccionario. Una vez encontrado este bloque, llamémoslo b_s de longitud l_s, construimos el nuevo bloque $b_{r+1} = (s, T_{j+l_s+1})$, y escribimos el par al final del archivo comprimido Z; es decir, $Z = b_1, \ldots, b_r b_{r+1}$, y agregamos el bloque al diccionario. Se agregará el carácter \$ al final del texto, sabiendo que es distinto que cualquier otro carácter, para asegurarnos que cada bloque corresponde a un bloque diferente. Se ve que este diccionario es cerrado bajo prefijo (es decir que cualquier prefijo de un elemento es también un elemento del diccionario) y se puede representar naturalmente por un *trie*.

Una propiedad interesante de esta forma de compresión es que cada bloque representa un substring diferente del texto, la única excepción puede ser el último bloque; pero eso se salva con el carácter que agregamos al final (\$). Esta propiedad es aprovechada por este algoritmo, al igual que la posibilidad de ordenar lexicográficamente el conjunto de strings almacenados.

Al finalizar el proceso hemos comprimido nuestro texto $T_{1...u}$ en $n+1$ bloques $T = b_0 \ldots b_n$, con $b_0 = \varepsilon$, $b_l \neq b_k$ si $l \neq k$ (no hay dos bloques iguales), y $\forall k \geq 1$, $\exists\, l > k$, $c \in \Sigma$, $b_k = b_l \cdot c$ (todos los bloques, salvo el b_0, se forman con un bloque anterior mas una letra); ahora debemos poder devolver todas las ocurrencias en el mismo de un determinado patrón $P_{1...m}$.

Para buscar un patrón en el texto debemos distinguir tres tipos de ocurrencias de P en T dependiendo de su distribución sobre los bloques:

- La ocurrencia cae dentro de un único bloque.
- La ocurrencia cruza dos bloques, b_k y b_{k+1}, tal que el prefijo $p_{1...i}$ coincide con un sufijo de b_k y el sufijo $P_{i+1...m}$ coincide con un prefijo de b_{k+1}.
- La ocurrencia cruza tres o más bloques, $b_k \ldots b_l$, tal que $P_{i...j} = b_{k+1} \ldots b_{l-1}$, $P_{1...i-1}$ coincide con un sufijo de b_k y el sufijo $P_{j+1...m}$ coincide con un prefijo de b_l.

Para realizar la búsqueda junto con el *LZtrie* guardamos un *RevLZTrie*, formado por todos los strings $b_0 \ldots b_n$ invertidos. Estas dos estructuras nos permitirán realizar búsquedas eficientes.

El algoritmo que realiza este tipo de compresión y búsqueda aparece en [Nav02a], donde aparecen detalles de implementación de todas las estructuras necesarias, de forma tal que el espacio utilizado sea pequeño y las búsquedas se resuelvan de manera eficiente.

4 Trabajo Futuro

- *SATD* se prevé analizar su comportamiento frente a otro tipo de operaciones no consideradas aún y existentes en Bases de Datos Espaciales. También se estudiará cómo modificarlo, a fin de obtener una buena estructura para memoria secundaria. Claramente en este aspecto puede ayudar el definir la aridad máxima, de manera tal de asegurar que entra todo en una página en memoria secundaria.

- *SATDP* se pretenden analizar otras técnicas de selección de pivotes que obtengan la mejor estructura para búsqueda cuando se cuenta con memoria acotada. Además, se prevé analizar más detalladamente las operaciones de inserción y eliminación para lograr mejorarlas haciendo uso de la información que brindan los pivotes.

- *LZ-Index* el primer paso seria reemplazar el trie que se usa para construir RevTrie por uno que use caminos comprimidos, exactamente como la versión comprimida. En una segunda etapa se va a tratar de construir tanto el *LZTrie* como el *RevTrie* ya en forma comprimida, en vez de pasar por una versión descomprimida. Además se pretende agregar dinamismo a las estructuras de modo de poder seguir agregando texto al ya existente sin tener que reconstruir todo. Con la idea de permitir usar estas técnicas en bases de datos de texto que no puedan mantenerse en memoria principal, se tiene también como objetivo obtener una implementación eficiente del índice en memoria secundaria.

Referencias

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435, 1997.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[Nav02a] G. Navarro. Indexing text using the ziv-lempel trie. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, LNCS 2476, pages 325–336. Springer, 2002.

[Nav02b] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[NR02] 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.

[Rey02] N. Reyes. Índices dinámicos para espacios métricos de alta dimensionalidad. Tesis M.Cs., Univ. Nac. de San Luis, Argentina, 2002. G. Navarro, Director.

[Ver95] K. Verbarg. The C-Tree: a dynamically balanced spatial index. *Computing Suppl.*, 10:323–340, 1995.