Bases de Datos no Convencionales *

Diego Arroyuelo [1]**, 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.

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 cuan 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 [3].

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 [5] para más detalles); pero normalmente éstas no se pueden extender a los espacios métricos generales.

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 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, y en algunos casos admitiendo un número pequeño de errores.

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 no sólo las búsquedas en esos ámbitos, sino también algunas otras operaciones de interés en el área de bases de datos. Por lo tanto, la investigación apunta a poner estas nuevas bases de datos a un nivel de madurez similar al de las bases de datos tradicionales.

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

[1]De licencia en la U.N.S.L., se encuentra realizando el doctorado en el Dpto. de Ciencias de la Computación de la Universidad de Chile (darroyue@dcc.uchile.cl).

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 triangular*.

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.

2.1. Bases de datos métricas

Integrar la búsqueda de espacios métricos en un ambiente de bases de datos requiere además extender apropiadamente el álgebra relacional y diseñar soluciones eficientes para los nuevos operadores, teniendo en cuenta aspectos de memoria secundaria, concurrencia, confiabilidad, etc. Algunos ejemplos de las operaciones que podrían ser de interés resolver son: join espacial, operaciones de conjuntos (unión, intersección y diferencia) y otras operaciones de interés en bases de datos espaciales tales como los operadores topológicos (por ejemplo: A incluye B, A intersecta B, etc.). Algunos de estos problemas ya poseen solución en las bases de datos espaciales, pero no se han estudiado en el ámbito de los espacios métricos.

Un objeto de un espacio métrico puede ser una imagen, una huella digital, un documento, o cualquier otro tipo de objeto. Esta es una de las razones por las que los elementos de una base de datos métrica no se pueden almacenar en bases de datos relacionales, las cuales tienen tamaño fijo de tupla. Esto también implica que las operaciones sobre datos de un espacio métrico son en general más costosas que las operaciones relacionales estándar. Además los elementos de un espacio métrico usualmente son *dinámicos*. Se insertan, o eliminan objetos dinámicamente, por lo tanto las estructuras de datos para accederlos deben soportar estas operaciones eficientemente. Otro aspecto a tener en cuenta es que no existe un *álgebra estándar* sobre datos de un espacio métrico, aunque se han hecho algunas propuestas [4]. Esto implica que no existe un conjunto estándar de operadores; éste depende de la aplicación específica, aunque existen algunos operadores comunes (como la intersección o el join).

Como en el mundo de las bases de datos relacionales, existen dos clases de operadores que se pueden aplicar a bases de datos espaciales: selección y join. Ejemplos de consultas y operaciones que podrían ser de interés en bases de datos que contienen objetos de un espacio métrico, son las siguientes:

Consulta por Rango: dado r y un conjunto A encontrar todos los objetos que estén a distancia a lo más r de algún objeto de A.

Consulta de Vecino más Cercano: dado dos conjuntos A y B, encontrar el objeto más cercano en el conjunto B a cada uno de los objetos de A.

Consulta de k–Vecinos más Cercanos: dado dos conjuntos A y B y k, encontrar los k objetos más cercanos en el conjunto B a cada uno de los objetos de A.

Consulta de Covecindad: dado dos conjuntos A y B, encontrar los objetos del conjunto A que tienen a algún objeto del conjunto B como vecino más cercano.

Unión: dados dos conjuntos A y B obtener un nuevo conjunto C que represente la unión de ambos.

Intersección de Conjuntos: dados dos conjuntos A y B, responder todos los objetos que están en A y en B.

Diferencia: dados dos conjuntos A y B y r, encontrar todos los objetos que se encuentren en A y que en B no existan objetos a distancia menor que r de ellos.

Join por Rango: dados dos conjuntos A y B y r, obtener un nuevo conjunto C en el que se emparejen aquellos objetos de A y de B que estén a distancia a lo más r.

Join por Vecino más Cercano: dados dos conjuntos A y B, obtener un nuevo conjunto C en el que se empareje cada objeto de A con el vecino más cercano en B.

Join por k-Vecinos más Cercanos: dados dos conjuntos A y B y k, obtener un nuevo conjunto C en el que se emparejen cada objeto de A con los k–vecinos más cercanos en B.

Join por Covecindad: dados dos conjuntos A y B, obtener un nuevo conjunto C en el que se emparejen cada objeto de A con los objetos de B que lo tienen como vecino más cercano.

Se pretende analizar no sólo la aplicabilidad de cada una de las consultas y operaciones mencionadas, sino también las estructuras o índices que permitan obtener su buen desempeño. Para ello analizaremos también algunos de los índices conocidos para búsquedas en espacios métricos y cómo se los podría adaptar para resolver eficientemente este tipo de operaciones. Como en general se mantiene un índice para cada conjunto, tendría sentido también analizar la información que da el construir un índice sobre un conjunto, para ver si vale la pena construirlo cuando no está (por ejemplo en conjuntos obtenidos como resultado de una consulta anterior).

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 usuario. En general el escenario más simple aparece 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 pueden aparecer dos problemas de búsqueda de texto: (1) consiste en encontrar todas las ocurrencias de P en T y (2) consiste en encontrar todas las ocurrencias de P en T que contengan a lo más r errores, es decir todos los substrings de T cuya *distancia de edición* (o *distancia de Levenshtein*) a P sea a lo más r.

En la actualidad las bases de datos de texto 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*.

Por otra parte, existen muchas aplicaciones en las cuales tiene sentido buscar todas las ocurrencias de un patrón P, pero admitiendo que nuestras respuestas contengan algunos errores. En este caso no son útiles los índices que se construyen para resolver el caso (1), sino que hay que diseñar nuevos índices que permitan responder eficientemente esta clase de consulta.

3.1. Búsqueda de texto sin errores

La estructura de datos *LZ–Index* está basada en el trie producido por el algoritmo de compresión de Ziv–Lempel (más precisamente en el algoritmo *LZ78*). La idea de la compresión de Ziv–Lempel es reemplazar substrings del texto por punteros a apariciones anteriores de dichos substrings. Si el puntero ocupa menos espacio que el substring que reemplaza, entonces se logró compresión. (ver [1] para una descripción del algoritmo LZ78).

El algoritmo LZ78 comprime el texto $T_{1...u}$ en $n+1$ bloques, $T = B_0 \dots B_n$, con $B_0 = \epsilon$, $B_l \neq B_k$ 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 B_0, se forman con un bloque anterior más una letra).

Para buscar un patrón $P_{1...m}$ en $T_{1...u}$ de manera eficiente, la idea de *LZ–Index* es usar el mismo trie producido por LZ78 más algunas estructuras de datos adicionales que a continuación enumeramos:

LZTrie: es el trie formado por los bloques $B_0 \dots B_n$. Por las propiedades de LZ78, este trie tiene exactamente $n + 1$ nodos.

RevTrie: es el trie formado por todos los strings reversos $B_0^r \dots B_n^r$. Esta estructura de datos no tiene las propiedades de *LZTrie*: pueden haber nodos internos que no representan a ningún bloque (nodos vacíos).

Node: es un mapeo de identificadores de bloque a su nodo en *LZTrie*.

RNode: es un mapeo de identificadores de bloque a su nodo en *RevTrie*.

Una característica importante de *LZ–Index* es que es un índice comprimido; luego de construir las estructuras de datos antes mencionadas, las mismas son comprimidas y el texto T puede ser borrado por completo, pudiéndose recuperar usando el índice (*self–indexing*). Esto es muy importante en la práctica ya que se requiere poca memoria.

Sin embargo, la construcción de *LZ–Index* aún requiere demasiada memoria. El trie reverso usado para la construcción del *RevTrie* comprimido es la estructura de datos que más memoria requiere: algunos experimentos realizados en [6] muestran que para lenguaje natural se necesita aproximadamente 4 veces el tamaño de texto, y para ADN 2.5 veces el tamaño del texto. Esto se debe al gran número de nodos vacíos en el trie reverso.

Actualmente estamos trabajando en una representación del trie reverso que garantiza tener $n + 1$ nodos (no existen nodos vacíos en él). Esto implica una importante disminución en la memoria requerida, pero también se espera un mayor tiempo de construcción debido a esto. Nuestra representación se basa en la conexión que existe entre *RevTrie* y *LZTrie*.

Otras características importantes son:

- En lugar de almacenar identificadores de bloques en los nodos del trie reverso, optamos por almacenar el puntero al nodo de *LZTrie* correspondiente al bloque. Esto hace que la estructura de datos *Node* no sea más necesaria, lo cual permite ahorrar más memoria y un acceso más rápido a los nodos de *LZTrie*.

- Con un barrido en–orden en el trie reverso aún podemos obtener los bloques ordenados lexicográficamente, lo que permite implementar las operaciones sobre el trie reverso mencionadas en [6] en tiempo constante.

Un problema de la construcción del *LZ-Index* es que el espacio que se necesita es mucho mayor que el de la estructura final de búsqueda. Nuestro primer objetivo es reducir el espacio necesario para la construcción, sin alterar la estructura final. Hasta el momento hemos conseguido una reducción cercana al 50 %, pero esperamos finalmente necesitar un espacio cercano al de la estructura final. El menor espacio de construcción se paga con mayor tiempo de construcción. Además, pretendemos agregar dinamismo a las estructuras de datos de modo de poder seguir agregando texto al ya existente sin tener que reconstruir el índice por completo. También planeamos obtener una implementación eficiente del índice en memoria secundaria, de manera tal que pueda ser usado en bases de datos de texto que no puedan mantenerse en memoria principal debido a su gran tamaño.

3.2. Búsqueda de texto con errores

Un problema abierto en la búsqueda combinatoria de patrones es la indexación del texto para permitir la búsqueda aproximada sobre él. El problema de búsqueda aproximada de un string es: dado un gran texto T de longitud n y un patrón P de longitud m (comparativamente más corto) y un valor r, devolver todas las ocurrencias del patrón, es decir, todos los substrings cuya *distancia de edición* al patrón es a lo sumo r.

La *distancia de edición* entre dos strings se define como la mínima cantidad de inserciones, supresiones o substituciones de caracteres necesaria para que los strings sean iguales. Esta distancia es usada en muchas aplicaciones, pero cualquier otra distancia puede ser de interés.

Hay variadas soluciones al problema de búsqueda aproximada basadas en el preprocesamiento del patrón pero no del texto, y como el tiempo de búsqueda es proporcional al tamaño del mismo no son aceptables cuando el texto es muy grande. Recientemente ha recibido mayor atención la posibilidad de indexar el texto para la búsqueda aproximada de strings. Sin embargo, la mayoría de los esfuerzos se orientan a la búsqueda en textos de lenguaje natural y las soluciones no se pueden extender a casos generales como ADN, proteínas, símbolos orientales, etc.

La presente aproximación [2] toma en cuenta que la distancia de edición satisface la *desigualdad triangular* y por lo tanto esto define un *espacio métrico* sobre el conjunto de substrings del texto. Entonces se puede replantear el problema de búsqueda aproximada como un problema de búsqueda por rango sobre un espacio métrico.

Una propuesta de indexación del texto para búsqueda aproximada de strings, usando técnicas de espacios métricos, tiene el problema que hay $O(n^2)$ diferentes substrings en un texto y además habría que indexar $O(n^2)$ objetos, lo cual es inaceptable.

En esta aproximación la idea es indexar los n sufijos del texto. Cada sufijo $[T_{j...}]$ representa todos los substrings que comienzan en la posición j. Existen diferentes opciones de cómo indexar este espacio métrico formado por $O(n)$ conjuntos de strings, la elegida aquí es una propuesta basada en *pivotes*. Entonces seleccionamos k pivotes y para cada conjunto de sufijos $[T_{j...}]$ del texto y cada pivote p_i, computamos la distancia entre p_i y todos los strings representados por $[T_{j...}]$. De ese conjunto de distancias desde $[T_{j...}]$ a p_i, se almacena sólo la mínima obtenida, es decir que se almacenarán sólo los k valores de las distancias mínimas de los substrings a los k pivotes.

Notar que no necesitamos construir ni guardar en ninguna estructura auxiliar los sufijos, ya que ellos se pueden obtener e indexar directamente del texto. Así, la única estructura necesaria sería el *índice métrico*.

Si se considera la búsqueda de un patrón dado P con a lo sumo r errores se está ante un query por rango de radio r en el espacio métrico de sufijos. Como en todos los algoritmos basados en pivotes comparamos el patrón P contra los k pivotes y obtenemos una coordenada k-dimensional $(ed(P, p_1), \ldots, ed(P, p_k))$. Los pivotes elegidos no deben ser muy cortos ya que su mínima distancia a cualquier $[T_{j...}]$ es a lo sumo $|p_i|$, por lo tanto cualquier pivote de longitud menor que $m + r$ es inútil.

Sea p_i un pivote dado y los sufijos $[T_{j...}]$ de T, si se cumple que $ed(P, p_i) + r < \min(ed([T_{j...}], p_i))$, entonces por la desigualdad triangular se sabe que $ed(P, [T_{j...}])) > r$ para todo substring que esté representado por $[T_{j...}]$. La eliminación se puede hacer usando cualquier pivote p_i. Al buscar qué sufijos podrán ser eliminados se cae en un clásico problema de búsqueda por rango en un espacio multidimensional, por lo que para esta tarea se podrían usar estructuras como el R-tree, o alguna de sus variantes. Los nodos que no puedan ser eliminados usando algún pivote deberán ser directamente comparados contra P. Para aquellos cuya distancia mínima a P sea a lo sumo r, se reportarán todas sus ocurrencias.

Se pretende implementar la aproximación presentada analizando su competitividad. Luego se tratará de mejorar esta propuesta basándonos en el conocimiento de que los pivotes con distancias mínimas grandes permiten la eliminación de una mayor cantidad de sufijos. Entonces se podrían almacenar para cada pivote p_i sólo los s valores más grandes de las distancias $\mathbf{min}(ed(p_i, [T_{j...}]))$; siendo s fijado de antemano.

Otra posible mejora a analizar se basa en tomar un primer pivote, determinar sus s sufijos más lejanos, almacenar esos sufijos y sus distancias mínimas en una lista en forma ordenada y luego descartar esos sufijos para próximas consideraciones. Este proceso se repetiría para los otros pivotes hasta que todos los sufijos hayan sido incluidos en alguna lista. Esta idea puede ser eficiente y competitiva en una implementación en memoria secundaria.

Referencias

[1] T. Bell, J. Cleary, and I. Witten. *Text Compression*. Prentice Hall, 1990.

[2] E. Chávez and G. Navarro. A metric index for approximate string matching. In *Proc. of the 5th Latin American Symposium on Theoretical Informatics (LATIN'02)*, LNCS 2286, pages 181–195, 2002.

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

[4] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. Similarity join in metric spaces. In *Proc. of the 25th European Conference on IR Research (ECIR'03)*, LNCS 2673, pages 452–467. Springer-Verlag, 2003.

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

[6] G. Navarro. Indexing text using the ziv-lempel trie. *Journal of Discrete Algorithms (JDA)*, 2(1):87–114, 2004.