Búsquedas en Bases de Datos de Texto y Bases de Datos Métricas *

Diego Arroyuelo † [1] **Verónica Ludueña** ‡ **Gonzalo Navarro** † **Nora Reyes** ‡

Departamento de Informática, Universidad Nacional de San Luis.‡
Departamento de Ciencias de la Computación, Universidad de Chile. †

darroyue@dcc.uchile.cl vlud@unsl.edu.ar gnavarro@dcc.uchile.cl nreyes@unsl.edu.ar

Resumen

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. Estos escenarios requieren modelos más generales tales como *bases de datos de texto* o *bases de datos métricas*.

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: las *Bases de Datos Métricas* 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. Así, la investigación apunta a poner estas nuevas bases de datos a un nivel de madurez similar al de las tradicionales.

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 *bases de datos métricas* (es decir, bases de datos que incluyan objetos de un espacio métrico), 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 estos tipos 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].

*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 (penúltimo 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 la Universidad de Chile.

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. El texto se puede ver como una secuencia de símbolos y el patrón a buscar como otra 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: las *Bases de Datos Métricas* 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.

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

Búsqueda por rango: recuperar todos los elementos de $\mathcal{S}$ 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 objetos de un espacio métrico en un ambiente de bases de datos requiere principalmente poder resolver consultas por similaridad eficientemente. En aplicaciones reales es posible tener grandes volúmenes de datos, ya sea por la cantidad de objetos o por el tamaño de cada objeto. Ello implica que debemos contar con estructuras adecuadas y eficientes para memoria secundaria. Además en un escenario real de un ambiente de base de datos es necesario contar con herramientas que permitan trabajar con grandes volúmenes de datos y usualmente serían *dinámicas*. Es decir, deberían permitir insertar, o eliminar objetos dinámicamente, por lo tanto los índices deberían soportar estas operaciones eficientemente. Un objeto de un espacio métrico puede ser una imagen, una huella digital, un documento, o cualquier otro tipo de objeto. Esto también implica que las operaciones sobre datos de un espacio métrico son, en general, más costosas que las operaciones relacionales estándar.

Existen índices que, en principio, no sólo resuelven ambos tipos de problemas; pero aún están muy inmaduros para ser usados en la vida real por dos motivos importantes: *falta de dinamismo* y *necesidad de trabajar en memoria principal*. Estas características son sobreentendidas en los índices para bases de datos tradicionales, y la investigación apunta a poner los índices para estas nuevas bases de datos a un nivel de madurez similar.

Hemos desarrollado una estructura para búsqueda por similitud en espacios métricos llamado *Árbol de Aproximación Espacial Dinámico (SATD)* [6] que permite realizar inserciones y eliminaciones, manteniendo una buen desempeño en las búsquedas. Muy pocos índices para espacios métricos son completamente dinámicos. Estamos actualmente desarrollando una versión del *SATD* que funcione adecuadamente en memoria secundaria, manteniendo su buen desempeño en las búsquedas y el dinamismo. Así, sería posible pensar en extender apropiadamente el álgebra relacional y diseñar

soluciones eficientes para los nuevos operadores, teniendo en cuenta aspectos no sólo de memoria secundaria, sino también de concurrencia, confiabilidad, etc. Algunos ejemplos de las operaciones que podrían ser de interés resolver son: join espacial, operaciones de conjuntos y otras operaciones de interés en bases de datos espaciales tales como los operadores topológicos. Algunos de estos problemas ya poseen solución en las bases de datos espaciales, pero no en el ámbito de los espacios métricos.

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* [5].

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 con Errores

Un problema abierto en la búsqueda combinatoria de patrones es la indexación del texto para permitir búsqueda aproximada sobre él. El problema de búsqueda aproximada 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* a P 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] considera que la distancia de edición satisface la *desigualdad triangular* y así esto define un *espacio métrico* sobre el conjunto de substrings de T. Entonces se puede replantear el problema 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, o sea sólo los k valores de las distancias mínimas de los substrings a los k pivotes.

Notar que no necesitamos construir ni guardar 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 $(ed(P, p_1), \ldots, ed(P, p_k))$.

Sea p_i un pivote dado y los sufijos $[T_{j...}]$ de T, si se cumple que $ed(P, p_i)+r < \min(ed([T_{j...}], p_i))$, 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, así para esta tarea se podría usar el R-tree, o alguna de sus variantes. Los nodos que no puedan ser eliminados usando algún pivote deberán ser directamente comparados contra P y para aquellos cuya distancia mínima a P sea a lo sumo r, se reportarán todas sus ocurrencias.

Algunas aspectos que estamos considerando actualmente incluyen:

- Como ya se ha realizado la implementación de la aproximación presentada, estamos actualmente realizando experimentos para analizar su competitividad.
- Mejorar esta propuesta basándonos en el conocimiento de que los pivotes con distancias mínimas grandes permiten la eliminación de una mayor cantidad de sufijos.
- Analizar otra posible mejora basada 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. Esta idea puede ser eficiente y competitiva en una implementación en memoria secundaria.

3.2. 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* [1]). 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.

Para buscar un patrón P en T de manera eficiente, la idea de *LZ–Index* es usar el mismo trie producido por *LZ78* más algunas estructuras de datos adicionales que a detallamos a continuación: *LZTrie*: es el trie formado por los bloques $B_0 \ldots B_n$. *RevTrie*: es el trie formado por todos los strings reversos $B_0^r \ldots B_n^r$. *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 para mantener el índice, a diferencia de las estructuras de datos tradicionales como los árboles y arreglos de sufijos.

Sin embargo, la construcción de *LZ–Index* aún requiere demasiada memoria, porque ésta se basa en representaciones no comprimidas de las estructuras de datos que conforman el índice. En los

experimentos de [5], el espacio necesario para construir el índice fue para lenguaje natural entre 4.8 a 5.8 veces y para ADN entre 3.4 a 3.7 veces el tamaño de T. Como comparación, la construcción de un arreglo de sufijos plano (sin ninguna estructura de datos extra) requiere un espacio de 5 veces el tamaño de T. La diferencia está en que, luego de construido, *LZ–Index* requiere de poca memoria, pero los arreglos de sufijos requieren siempre el mismo espacio.

Actualmente estamos trabajando en un algoritmo de construcción de *LZ–Index* que requiera poca memoria. La idea principal consiste en reemplazar las representaciones intermedias de *LZTrie* y *RevTrie* por representaciones que sean eficientes en el uso de espacio. Una idea es emplear la representación lineal de paréntesis balanceados de Munro y Raman [4]. Sin embargo, el problema de usar esta representación lineal es que debe ser recomputada por completo ante cada inserción de un nuevo nodo, lo cual es muy costoso. Para evitar este problema, hemos definido una *representación jerárquica de paréntesis balanceados*, de manera tal de tener que recomputar sólo una pequeña parte de la secuencia ante una inserción. Esto es implementado como un árbol de páginas, en donde cada página implementa una subsecuencia de paréntesis balanceados.

Hasta el momento hemos trabajado en definir un método para garantizar un cierto porcentaje de ocupación de las páginas del árbol, y así reducir el desperdicio de memoria y otro método para el tratamiento de rebalse de las páginas y un algoritmo de compresión de caminos unarios vacíos para reducir los requerimientos de espacio en la construcción del trie reverso.

Actualmente estamos implementando las propuestas con el fin de evaluarlas empíricamente en cuanto al tiempo y espacio requerido para la construcción. Se espera que el método propuesto reduzca significativamente los requerimientos de memoria al construir *LZ–Index*. Además, planeamos realizar el análisis teórico de la propuesta, para obtener garantías teóricas de la eficiencia del algoritmo.

Algunas líneas de investigación futura incluyen:

- Reducir el espacio requerido por *LZ–Index* eliminando algo de la redundancia que agregan algunas de las estructuras de datos que conforman el índice (sobre todo *Node* y *RNode*).
- Obtención de un prototipo de *LZ–Index* para memoria secundaria.
- Permitir el agregado de texto en el índice, como paso previo para el dinamismo total del índice.

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] I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. of the 38th IEEE Symposium on Foundations of Computer Science (FOCS'97)*, pages 118–126, 1997.

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

[6] 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.