Análisis e Indexación de Datos no Convencionales *

Cristian Bustos, Susana Esquivel, Verónica Ludueña, Nora Reyes, Patricia Roggero
Departamento de Informática, Universidad Nacional de San Luis.
{cjbustos,esquivel,vlud,nreyes,proggero}@unsl.edu.ar

Edgar Chávez
Escuela de Ciencias Físico–Matemáticas, Universidad Michoacana de San Nicolás de Hidalgo
elchavez@umich.mx

Gonzalo Navarro
Departamento de Ciencias de la Computación, Universidad de Chile.
gnavarro@dcc.uchile.cl

1. Introducción y Motivación

La constante aparición de datos en forma digital de diferentes tipos, tamaños y en gran cantidad, concuerda con un crecimiento de las capacidades de almacenamiento a precios más moderados. Por otro lado, dos fenómenos se han manifestado últimamente: (a) mientras la velocidad de procesamiento de la CPU se ha duplicado casi anualmente, la de los almacenamientos masivos ha progresado poco; (b) han aparecido memorias caché con mayor capacidad, más rápidas y más pequeñas, aunque más costosas, que las memorias RAM. Estos fenómenos han cambiado los modelos de costos utilizados para diseñar algoritmos y estructuras de datos eficientes. Por tal motivo los costos que se pagaban al almacenar datos en forma comprimida, en términos de velocidad de procesamiento por la descompresión, hoy en día se tornan despreciables debido a que la diferencia entre los tiempos de CPU y acceso a disco es tan significativa que el esfuerzo de descompresión se paga a cambio de una pequeña disminución en el tiempo de I/O. Además, la transferencia de los datos sobre una red local cuesta casi lo mismo que la transferencia a disco, por lo cual ésta se ve favorecida con la compresión. Este escenario ha originado líneas de investigación que tienen en cuenta estas diferencias de costos de operaciones, así nos dedicamos a las *estructuras de datos*: compactas y/o con I/O eficiente.

Nuestro objetivo es contribuir a estas líneas de investigación, diseñando estructuras de datos más eficientes para memorias jerárquicas, haciendo uso de la compacticidad o la I/O eficiente. Particularmente nos centraremos en las estructuras de datos capaces de manipular los siguientes tipos de datos: secuencias, textos, grafos, y espacios métricos, entre otros, y en estudiar los problemas desde ambos puntos de vista teórico y empírico. Además de diseñar estructuras de datos, planeamos investigar otros aspectos tales como la construcción eficiente (en espacio o en I/O u otras medidas de eficiencia), el dinamismo (es decir actualizaciones eficientes) y operaciones de búsqueda complejas (más allá de las básicas soportadas por las estructuras de datos clásicas).

2. Métodos de Acceso Métricos

Aunque existen numerosas estructuras para búsquedas por similitud en espacios métricos, sólo unas pocas trabajan eficientemente en espacios de alta o mediana dimensión, y la mayoría no admiten dinamismo, ni están diseñadas para trabajar sobre grandes volúmenes de datos; es decir, en memoria secundaria. Por lo tanto, estudiamos distintas maneras de optimizar algunas de las estructuras que han mostrado buen desempeño, con el fin de optimizarlas teniendo en cuenta la jerarquía de memorias.

2.1. *SATD*

Hemos desarrollado una estructura para búsqueda por similitud en espacios métricos llamado *árbol de Aproximación Espacial Dinámico (SATD)* [14] que permite realizar inserciones y eliminaciones, manteniendo un buen desempeño en las búsquedas. Muy pocos índices para espacios métricos son completa-

*Enmarcado dentro de la línea Bases de Datos no Convencionales del Proyecto "Tecnologías Avanzadas de Bases de Datos" de la Universidad Nacional de San Luis.

mente dinámicos. Esta estructura se basa en el *Árbol de Aproximación Espacial* [13], el cual había mostrado un muy buen desempeño en espacios de mediana a alta dimensión, pero era estático.

El *SAT* está definido recursivamente; la propiedad que cumple la raíz a (y a su vez cada uno de los nodos) es que los hijos están más cerca de la raíz que de cualquier otro punto de S. La construcción del árbol se hace también de manera recursiva. De la definición se observa que se necesitan de antemano todos los elementos para la construcción y que queda completamente determinado al elegirle una raíz, lo cual en el *SAT* original se realizaba al azar.

Actualización de *SATD*

Para el desarrollo del *SATD* [14] se han estudiado distintas maneras de realizar incorporaciones de nuevos elementos sobre el árbol, pero se optó por un método que inserta un nuevo elemento en un punto determinado del árbol, manteniendo la aridad acotada. Gracias a esos trabajos, quedó demostrado que existen otros posibles puntos de inserción válidos, aunque no se analizó cuál de ellos sería el mejor.

Por lo tanto, tiene sentido considerar si los otros puntos posibles de inserción para un elemento consiguen mejorar aún más los costos de búsqueda. Así, como resultado se espera brindar una estructura que mejore el comportamiento durante las búsquedas gracias a elegir adecuadamente los puntos de inserción de los nuevos elementos.

El tema está siendo desarrollado como trabajo final de la Lic. en Ciencias de la Computación.

***SATD* con Clustering**

El *SATD* es una estructura que realiza la partición del espacio considerando la proximidad espacial; pero, si el árbol agrupara los elementos que se encuentran muy cercanos entre sí, lograría mejorar las búsquedas al evitar recorrerlo para alcanzarlos.

Podemos pensar entonces que construimos un *SATD*, en el que cada nodo representa un grupo de elementos muy cercanos ("clusters") y relacionamos los clusters por su proximidad en el espacio. La idea sería que en cada nodo se mantenga 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.

Esperamos así obtener una estructura más adecuada a espacios en los que se sabe de antemano que pueden existir "clusters" de elementos, aprovechándonos de la existencia de los mismos para mejorar las búsquedas. Además, esta nueva estructura podría ser eficiente en memoria secundaria, lo cual la haría más adecuada a aplicaciones que traten con grandes volúmenes de datos.

Otro aspecto a analizar es cuán bueno es el agrupamiento o "clustering"que lograría esta estructura, lo cual podría estudiarse haciendo uso de nuevas estrategias de optimización de funciones a través de heurísticas bioinspiradas, las cuales han mostrado ser útiles en detección de clusters.

Estos temas han dado lugar a dos trabajos finales de la Lic. en Ciencias de la Computación, que se están desarrollando actualmente.

2.2. Join Métricos

El modelo de espacios métricos permite cubrir muchos problemas de búsqueda por similitud o proximidad, aunque en general se deja fuera de consideración al ensamble o "join"por similitud, otra primitiva extremadamente importante [7]. De hecho, a pesar de la atención que esta primitiva ha recibido en las bases de datos tradicionales y aún en las multidimensionales, no han habido grandes avances para espacios métricos generales.

Nos hemos planteado resolver algunas variantes del problema de join por similitud: (1) *join por rango:* dadas dos bases de datos de un espacio métrico y un radio r, encontrar todos los pares de objetos (uno desde cada base de datos) a distancia a lo sumo de r, (2) *k–pares más cercanos:* encontrar los k pares de objetos más cercanos entre sí (uno desde cada base de datos). Para resolver estas operaciones de manera eficiente hemos diseñado un nuevo índice métrico, llamado *Lista de Clusters Gemelos* (*LTC* por su sigla en inglés) [15], el cual construye el índice de ambas bases de datos conjuntamente, en lugar de lo que habitualmente se haría de indexar una o ambas bases de datos independientemente. Esta nueva estructura permite además resolver las consultas por similitud clásicas en espacios métricos sobre cada una de las bases de datos independientemente.

A pesar de que esta estructura ha mostrado ser competitiva y obtener buen desempeño en relación a las alternativas más comunes para resolver las operaciones de join, aún queda mucho por mejorar para que se vuelva una estructura práctica y mucho más eficiente para trabajar con grandes bases de datos métricas. De esta manera sería posible pensar en extender apropiadamente el álgebra relacional y diseñar soluciones eficientes para nuevas operaciones, teniendo en cuenta aspectos no sólo de memoria secundaria, sino también de concurrencia, confiabili-

dad, etc. Algunos de estos problemas ya poseen solución en las bases de datos espaciales, pero no en el ámbito de los espacios métricos.

Este tema forma parte de la tesis doctoral de uno de los investigadores de la línea.

3. Dimensión Intrínseca

En los espacios de vectores, la "maldición de la dimensionalidad" describe el fenómeno por el cual el desempeño de todos los algoritmos existentes se deteriora exponencialmente con la dimensión. En espacios métricos generales la complejidad se mide como el número de cálculos de distancias realizados, pero la ausencia de coordenadas no permite analizar la complejidad en términos de la dimensión.

En los espacios vectoriales existe una clara relación entre la dimensión (*intrínseca*) del espacio y la dificultad de buscar. Se habla de "intrínseca", como opuesta a "representacional". Los algoritmos más ingeniosos se comportan más de acuerdo a la dimensión intrínseca que a la representacional. Hay varios intentos de medir la dimensión intrínseca en espacios de vectores, como la transformada de *Karhunen-Loève (KL)* y otras medidas tales como *Fastmap* [8]. Otro intento de medir la dimensión de espacios de vectores no uniformemente distribuidos, es la *dimensión fractal* [2].

Existen sólo unas pocas propuestas diferentes sobre cómo estimar la dimensión intrínseca de un espacio métrico tales como el *exponente de la distancia* (basada en una ley de potencias empírica observada en muchos conjuntos de datos) [10], y la medida de dimensión intrínseca como una medida cuantitativa basada en el histograma de distancias [4]. Además, también parece posible adaptar algunos de los estimadores de distancia para espacios de vectores para aplicarlos a espacios métricos generales, como por ejemplo *Fastmap* y *dimensión fractal*.

Muchos autores [1, 3, 6, 4] han propuesto usar histogramas de distancia para caracterizar la dificultad de las búsquedas en espacios métricos arbitrarios. Existe al menos una medida cuantitativa [4], pero ella no refleja fielmente la facilidad o dificultad de buscar en un espacio métrico dado. Así, la idea es buscar una medida posible basada en Half-Space Proximal (HSP) [5].

El test HSP es un nuevo test local para extraer un subgrafo no dirigido de un grafo de disco unitario (GDU)[1]. Los vecinos HSP de cada vértice son únicos, dado un GDU fijo. El test HSP es un algoritmo computacionalmente simple y distribuído, que se aplica independientemente a cada vértice de un GDU. Por lo tanto, adaptamos el test HSP para aplicarlo a espacios métricos con el fin de obtener con él una medida de dimensionalidad intrínseca.

En aplicaciones reales de búsqueda en espacios métricos, sería muy importante contar con un buen estimador de la dimensión intrínseca porque nos permitiría decidir el índice adecuado a utilizar en función de la dimensión del espacio. Además, tener una buena estimación de la dimensión nos permitiría, en algunas ocasiones, elegir la función de distancia de manera tal que se obtenga una menor dimensión.

Este tema ha dado lugar a un trabajo final de la Lic. en Ciencias de la Computación, que está en desarrollo y además está siendo analizado desde un punto de vista más amplio por uno de los investigadores del grupo a fin de proponer nuevas medidas.

4. Estructuras de Datos Compactas

Las estructuras de datos *compactas*, son variantes de las estructuras clásicas sólo que funcionan en espacio reducido y se pueden dividir en dos grupos: *sucintas* y *comprimidas*. Una estructura de datos se dice comprimida cuando toma espacio proporcional al de la secuencia comprimida (existe cierta libertad para elegir un método razonable de compresión). En cambio, una estructura de datos es sucinta si necesita espacio asintóticamente despreciable sobre los datos en bruto. Una medida popular de compresibilidad de secuencias es la entropía de orden k-ésimo (H_k), según lo definido por Manzini [12], que mide la complejidad del espacio para cada secuencia individual, sin ninguna suposición sobre la entrada. La medida H_k es un límite inferior para el número de bits de salida cuando se comprime S con cualquier compresor que codifique cada símbolo de la entrada que dependa solamente de los k símbolos precedentes. Esto abarca los estándares populares tales como *PPM*, *la familia de Lempel-Ziv* y compresores basados en *Burrows-Wheeler*.

Sea s el alfabeto de una secuencia de símbolos S, $n(i)$ el número de ocurrencias del i-ésimo símbolo, y n la longitud de S. Entonces $H_0(S) = \Sigma(n(i)\log(n/n(i))$. Sea w una secuencia de longitud k y $w(S)$ la subsecuencia de símbolos de S que

[1] Un grafo de disco unitario es un grafo bipartito en el cual los vértices son objetos del universo y dos vértices estarán conectados por un arco si la distancia entre ellos es menor que una unidad dada.

siguen a w, entonces $H_k(S) = 1/n\Sigma(|w|H_0(w))$ sobre todos los posibles w. Esta medida es popular también en las estructuras de datos comprimidas, como aquellos índices comprimidos que codifican un texto T de n símbolos sobre un alfabeto s usando espacio de $H_k(T) + o(n \log s)$ bits y además pueden buscar eficientemente patrones en el texto. Observar que la codificación llana del texto toma $n \log s$ bits. La idea de las estructuras de datos compactas se diferencia de la compresión pura en su capacidad de manipular los datos en forma comprimida, sin tener que descomprimirlos primero. En la actualidad las estructuras de datos compactas pueden manipular secuencias de bits o de símbolos generales, árboles en general, grafos, colecciones de texto, permutaciones y mapping, sumas parciales, búsqueda por rango en una y más dimensiones, etc.

4.1. Búsqueda en Texto con Estructuras Comprimidas

El texto puede convertirse en un medio en el cual se necesite realizar búsquedas por similitud. El problema de la *búsqueda aproximada de un patrón en el texto* puede verse como: dado un *texto* $T = T[1, n]$ y un *patrón* $P = P[1, m]$ en el alfabeto Σ (con $m << n$) y un entero k, se desea encontrar y devolver todos los *substrings* en el texto que sean una ocurrencia aproximada de P, con a lo más k diferencias. La diferencia entre dos strings α y β se determina con la *distancia de edición* d; $d(\alpha, \beta)$ es el mínimo número de inserciones, eliminaciones y/o sustituciones de caracteres que se deben realizar para convertir β en α. Este tipo de búsqueda tiene aplicaciones tales como la recuperación de errores (en reconocimiento óptico de caracteres, spelling) , biología computacional, comunicaciones de datos, data mining, bases de datos textuales, entre otras.

La solución básica al problema de búsqueda aproximada de patrones se basa en la *programación dinámica* [11, 16]. Se puede mejorar introduciendo algún esquema de *filtración* para extraer las áreas del texto con potenciales coincidencias para luego inspeccionarlas por medio de programación dinámica, en una segunda fase del algoritmo.

Luego, el propósito es encontrar en el texto q-gramas que coincidan con los del patrón; esto se puede realizar de dos maneras: *dinámica* que escanea el texto en tiempo lineal, y *estática* que utiliza un *índice de q-gramas*. La propuesta estática es superior cuando el índice puede usarse varias veces.

Implementación de índices de q-gramas. Un índice de q-gramas es una estructura de datos que permite encontrar rápidamente en el texto todas las ocurrencias de un q-grama dado. Existen distintas implementaciones de índices para los métodos estáticos. La básica es un arreglo de punteros de tamaño $|\Sigma|^q$ (una entrada para cada posible subcadena). Cada posición del arreglo referencia a la lista de ocurrencias en el texto del q-grama correspondiente. El índice puede resultar inútilmente grande[9]. Una mejora a este esquema de indexación es reducir el tamaño del arreglo de punteros por medio de un hashing eficiente, sin una demora significativa en las búsquedas. Otro enfoque usa una estructura de trie construído con los distintos q-gramas que aparecen en el texto. Cada hoja del trie contiene un puntero a la lista de ocurrencias del correspondiente q-grama en el texto. El trie puede construirse con un algoritmo similar al de la creación del árbol de sufijos [17].

Las implementaciones anteriores encuentran todas las ocurrencias de un q-grama dado en tiempo óptimo en función del número de ocurrencias encontradas. Pero todas sufren el mismo inconveniente: el tamaño del índice se vuelve impráctico al crecer la longitud del texto.

Un índice que usa compresión. En este índice *Lempel-Ziv* (LZ) para q-gramas las listas de ocurrencias se reemplazan con una estructura de datos más compacta. La estructura (hashing o trie) para los distintos q-gramas, ahora llamada *índice primario*, es aún necesaria para proveer un punto de comienzo para las búsquedas.

La representación compacta de las listas de ocurrencias toma ventaja de las *repeticiones* en el texto. La primera ocurrencia del string será llamada *definición* y las siguientes se llamarán *frases*. Luego, cada ocurrencia de un q-grama es o el primero de su clase o parte de alguna frase. La primera ocurrencia se almacenará en el índice primario y las demás se encontrarán usando un parsing de Lempel-Ziv[18] que usa la información sobre repeticiones. El índice LZ encontrará todas las ocurrencias de un q-grama en igual tiempo que los índices tradicionales.

Esta temática se investiga en el marco de la tesis de maestría de un investigador de la línea.

5. Conclusiones y Trabajos Futuros

Nuestras investigaciones además se encuadran en el marco de dos proyectos dentro del Programa de Promoción de la Universidad Argentina para el Fortalecimiento de Redes Interuniversitarias III en los que participa nuestra universidad junto con las uni-

versidades de: Chile, da Coruña (España), Michoacana (México) y Zaragoza (España).

Como trabajo futuro de esta línea de investigación se consideran varios aspectos relacionados al diseño de estructuras de datos que, concientes de que existe una jerarquía de memorias y de las características particulares de los datos a ser indexados, saquen el mejor partido haciendólas eficientes tanto en espacio como en tiempo.

Se trabajará en particular con estructuras de datos compactas para textos, implementando un índice LZ para q-gramas y estudiando su comportamiento en comparación con los índices tradicionales.

En el caso de espacios métricos, se intentará que las estructuras se adapten mejor al espacio métrico particular considerado, gracias a la determinación de su dimensión intrínseca, y también al nivel de la jerarquía de memorias en que se deba almacenar. Es importante destacar que estos estudios sobre espacios métricos y sobre algunas estructuras de datos particulares (como el *SATD*) permitirán no sólo mejorar el desempeño de las mismas sino también aplicar, eventualmente, muchos de los resultados que se obtengan a otras estructuras para espacios métricos.

Referencias

[1] S. Brin. Near neighbor search in large metric spaces. In *In Proc. 21st Conference on Very Large Databases*, page 574.

[2] Francesco Camastra. Data dimensionality estimation methods: a survey. *Pattern Recognition*, 36(12):2945–2954, 2003.

[3] E. Chávez and J. Marroquín. Proximity queries in metric spaces. In *In R. Baeza-Yates, editor. American Workshop on String Processing*, pages 21–36. Carleton University Press, 1997.

[4] E. Chávez and G. Navarro. Towards measuring the searching complexity of metric spaces. In *Proc. International Mexican Conference in Computer Science*, volume II, pages 969–978, 2001. Sociedad Mexicana de Ciencias de la Computación.

[5] E. Chávez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, H. Tejeda, and J. Urrutia. Half-space proximal: A new local test for extracting a bounded dilation spanner of a unit disk graph. In *OPODIS*, pages 235–245, 2005.

[6] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. *In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1998.

[7] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. Similarity join in metric spaces. In *Proc. 25th European Conf. on IR Research*, LNCS 2633, pages 452–467, 2003.

[8] C. Faloutsos and K. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174. ACM Press, 1995.

[9] Petteri Jokinen and Esko Ukkonen. Two algorithms for approximate string matching in static texts. In *MFCS*, pages 240–248, 1991.

[10] C. Traina Jr., A. M. Traina, and C. Faloutsos. Distance exponent: A new concept for selectivity estimation in metric trees. In *ICDE*, page 195, 2000.

[11] G. Landau and U. Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988.

[12] G. Manzini. An analysis of the burrows-wheeler transform. *J. ACM*, 48(3):407–430, 2001.

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

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

[15] R. Paredes and N. Reyes. List of twin clusters: a data structure for similarity joins in metric spaces. In *Proc. of the 1st International Workshop on Similarity Search and Applications*, pages 131–138. IEEE, 2008.

[16] E. Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6(1):132–137, 1985.

[17] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[18] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.