Spaghettis en Memoria Secundaria

Roberto Uribe Paredes*
Centro de Investigación de la Web
Depto. de Ingeniería en Computación
Universidad de Magallanes, Bulnes 01855, Punta Arenas, Chile
(ruribe@ona.fi.umag.cl)

Christian Cárdenas Villarroel**
Depto. de Ingeniería en Computación
Universidad de Magallanes, Bulnes 01855, Punta Arenas, Chile
(ccardena@ona.fi.umag.cl)

Abstract

The Spaghettis is a data structure deviced for searching in metric spaces. Empirical results show that this structure achieves good performance for high-dimensional metric spaces. This structure is an array type and is based on pivots.

An unusual characteristic on the current metric structures is their manipulation on secundary memory. The utilization of this structure on real aplications could be uncertain if is not allow this characteristic.

This paper describe differents choices for storage on secundary memory for this structure, showing particularly one, that keep cost low in access and storage on disc without diminishing the cost for distance evaluations. Front part, allow real aplications structure use, even on large metric spaces databases.

Keywords: databases, data estructures, algorithms, metric spaces, similarity queries.

Resumen

El Spaghettis es una estructura de datos para búsquedas por similaridad en espacios métricos [CMBY99]. Esta estructura es prometedora dado que se ha demostrado que tiene buen desempeño en espacios de alta dimensión. Esta estructura es basada en pivotes y es del tipo arreglo.

Una característica poco común en las estructuras métricas actuales es la manipulación de éstas en memoria secundaria. No poseer dicha característica hace poco factible la utilización de estas estructuras en aplicaciones reales.

El presente trabajo describe distintas alternativas de almacenamiento en memoria secundaria para la estructura, mostrando en particular una, que resulta óptima para mantener bajos los costos de almacenamiento y accesos a disco sin disminuir los costos en términos de evaluaciones de distancia. Lo anterior, permite la utilización de dicha estructura en aplicaciones reales, dado el gran tamaño de las actuales bases de datos.

Palabras claves: bases de datos, estructuras de datos, algoritmos, espacios métricos, consultas por similaridad.

*Parcialmente financiado por Fondecyt 1060776, Conicyt, Chile.
**Parcialmente financiado por el programa de investigación PR-F1-002IC-06, Universidad de Magallanes.

1. Introducción

1.1. Antecedentes

Uno de los problemas de gran interés en ciencias de la computación es el de "búsqueda por similaridad", es decir, encontrar los elementos de un conjunto más similares a una muestra. Esta búsqueda es necesaria en múltiples aplicaciones, como ser en reconocimiento de voz e imagen, compresión de video, genética, minería de datos, recuperación de información, etc. En casi todas las aplicaciones la evaluación de la similaridad entre dos elementos es cara, por lo que usualmente se trata como medida del costo de la búsqueda la cantidad de similaridades que se evalúan.

Interesa el caso donde la similaridad describe un espacio métrico, es decir, está modelada por una función de distancia que respeta la desigualdad triangular. En este caso, el problema más común y difícil es en aquellos espacios de "alta dimensión" donde el histograma de distancias es concentrado, es decir, todos los objetos están más o menos a la misma distancia unos de otros.

El aumento de tamaño de las bases de datos y la aparición de nuevos tipos de datos sobre los cuales no interesa realizar búsquedas exactas, crean la necesidad de plantear nuevas estructuras para búsqueda por similaridad o búsqueda aproximada. Asimismo, se necesita que dichas estructuras sean dinámicas, es decir, que permitan agregar o eliminar elementos sin necesidad de crearlas nuevamente. Así también, las aplicaciones reales requieren que dichas estructuras permitan ser almacenadas en memoria secundaria eficientemente, como también que posean métodos optimizados para reducir los costos de accesos a disco.

1.2. Marco teórico

La similaridad se modeliza en muchos casos interesantes a través de un espacio métrico, y la búsqueda de objetos más similares a través de una búsqueda por rango o de vecinos más cercanos.

Definición 1 (*Espacios Métricos*): Un espacio métrico es un conjunto X con una función de distancia $d : X^2 \to R$, tal que $\forall x, y, z \in X$,

1. $d(x, y) \geq 0 \, and \, d(x, y) = 0 \, ssi \, x = y$. *(positividad)*
2. $d(x, y) = d(y, x)$. *(Simetría)*
3. $d(x, y) + d(y, z) \geq (d(x, z)$. *(Desigualdad Triangular)*

Definición 2 (*Consulta por Rango*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$, y un rango $r \in R$. La consulta de rango alrededor de x con rango r es el conjunto de puntos $y \in Y$, tal que $d(x, y) \leq r$.

Definición 3 (*Los k Vecinos más Cercanos*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$ y un entero k. Los k vecinos más cercanos a x son un subconjunto A de objetos de Y, donde la $|A| = k$ y no existe un objeto $y \in A$ tal que *d(y,x)* sea menor a la distancia de algún objeto de A a x.

El objetivo de los algoritmos de búsqueda es minimizar la cantidad de evaluaciones de distancia realizadas para resolver la consulta. Los métodos para buscar en espacios métricos se basan principalmente en dividir el espacio empleando la distancia a uno o más objetos seleccionados. El no trabajar con las características particulares de cada aplicación tiene la ventaja de ser más general, pues los algoritmos funcionan con cualquier tipo de objeto [CNBYM01].

Existen distintas estructuras para buscar en espacios métricos, las cuales pueden ocupar funciones discretas o continuas de distancia. Algunos son BKTree [BK73], MetricTree [Uhl91], GNAT [Bri95], VpTree [Yia93], FQTree [BYCMW94], MTree [CPZ97], SAT [Nav02], Slim-Tree [TTSF00].

Algunas de las estructuras anteriores basan la búsqueda en pivotes y otras en clustering. En el primer caso se seleccionan pivotes del conjunto de datos y se precalculan las distancias entre los elementos y los pivotes. Cuando se realiza una consulta, se calcula la distancia de la consulta a los pivotes y se usa la desigualdad triangular para descartar candidatos.

Los algoritmos basados en clustering dividen el espacio en áreas, donde cada área tiene un *centro*. Se almacena alguna información sobre el área que permita descartar toda el área mediante sólo comparar la consulta con su centro. Los algoritmos de clustering son los mejores para espacios de alta dimensión, que es el problema más difícil en la práctica.

Existen dos criterios para delimitar las áreas en las estructuras basadas en clustering, *hiperplanos* y *radio cobertor* (*covering radius*). El primero divide el espacio en particiones de *Voronoi* y determina el hiperplano al cual pertenece la consulta según a qué centro corresponde. El criterio de radio cobertor divide el espacio en esferas que pueden intersectarse y una consulta puede pertenecer a más de una esfera.

Definición 4 (*Diagrama de Voronoi*): Considérese un conjunto de puntos $\{c_1, c_2, \ldots, c_n\}$(centros). Se define el diagrama de Voronoi como la subdivisión del plano en n áreas, una por cada c_i, tal que $q \in$ al área c_i sí y sólo sí la distancia euclidiana $d(q, c_i) < d(q, c_j)$ para cada c_j, con $j \neq i$.

Existen dos características poco comunes en las estructuras actuales. La primera es el *dinamismo*, es decir, la posibilidad de insertar y eliminar objetos una vez construida la estructura. La segunda característica, aún menos frecuente, es la manipulación de estas estructuras en memoria secundaria. No poseer dichas características hace poco factible la utilización de estas estructuras en aplicaciones reales.

Spaghettis es una estructura basada en pivotes y es una variante de *LAESA* [MOV94]. Propone reducir el tiempo de CPU extra necesario al realizar una consulta utilizando una estructura de datos en donde las distancias a los pivotes están ordenadas por separado, construyendo un arreglo por cada pivote, lo que permite realizar una búsqueda binaria en el rango relevante.

Si para cada pivote se encuentra el conjunto $s_i = \{x : |d(x, p_i) - d(q, p_i| \leq r\}, i = 1, ..., k$ entonces la lista de candidatos está dada por la intersección de todos estos conjuntos.

Para este artículo se seleccionó, para la realización de las pruebas, un espacio métrico consistente en un diccionario de palabras en castellano de 86.061 objetos, donde la distancia utilizada es la *distancia de edición*, la cual entrega como resultado el número mínimo de inserciones, eliminaciones o reemplazos de caracteres, necesarios, para que una palabra sea igual a otra. Para la búsqueda se creó la estructura con el 90 % de los datos y se reservo el 10 % como consultas.

2. Spaghettis

2.1. Construcción de Spaghettis

1. Se seleccionan k puntos (*pivotes*), $p_1, ..., p_k$, los cuales pueden o no pertenecer a la base de datos a indexar.
2. Se calcula y almacena la distancia entre los pivotes y cada objeto de la base de datos, almacenando además un identificador de cada objeto, obteniendo un arreglo de largo n para cada pivote.
3. Para los primeros *k-1* pivotes se cambia el identificador del objeto, por su posición en el pivote siguiente.

2.2. Búsqueda en Spaghettis

Se asume que se desea buscar todos los objetos con distancia $d \leq r$ a un objeto q. Una búsqueda en un *spaghettis*, se realiza como sigue:

1. Se calcula la distancia entre q y todos los pivotes p_k, para luego obtener k intervalos de la forma $[a_1, b_1], ..., [a_k, b_k]$, donde $a_i = d(p_i, q)$ - r y $b_i = d(p_i, q) + r$.

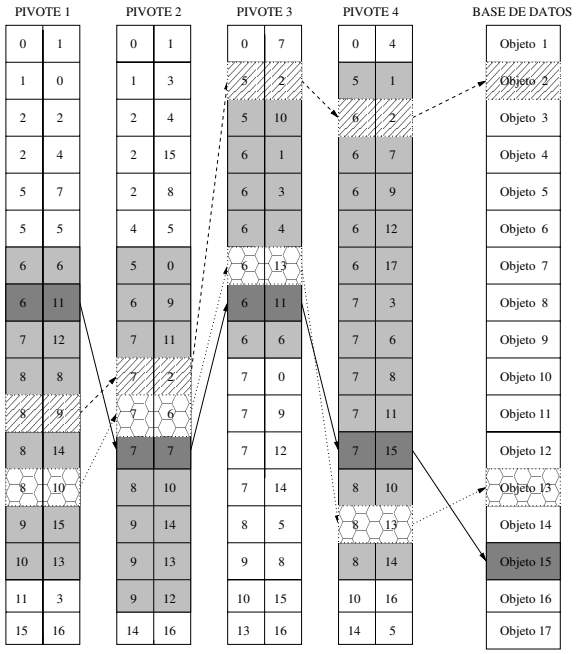


Figura 1: Spaghettis: Construcción y representación de objetos candidatos.

2. Los objetos que se encuentren en la intersección entre los k intervalos formados, se convierten en candidatos como posible respuesta a la consulta q.

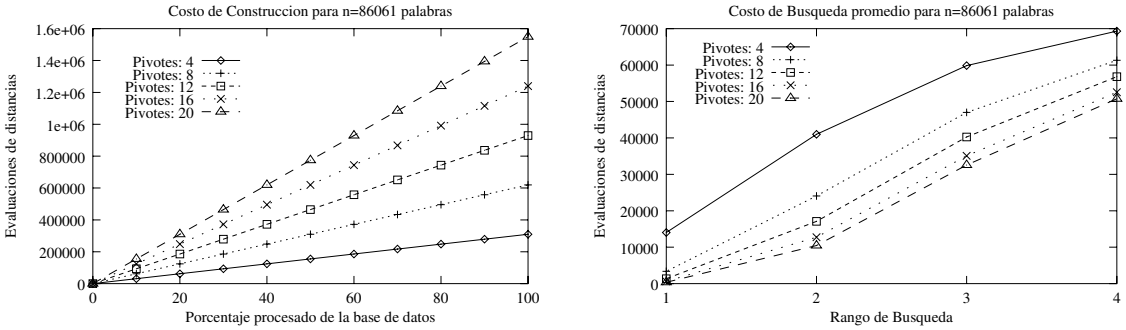
3. Para cada objeto candidato x, se calcula la distancia entre éste y la consulta q. Si se tiene que $d(q, x) \leq r$, entonces el objeto x es una respuesta a la consulta.

La figura 1 representa la estructura spaghettis construida usando 4 pivotes para indexar una base de datos de 17 objetos. Sobre esta estructura se realiza la búsqueda como sigue. Suponga una consulta q con distancia a los pivotes $\{8, 7, 4, 6\}$ y rango de búsqueda $r = 2$. En la figura 1 se muestran más oscurecidos los intervalos $\{(6, 10), (5, 9), (2, 4), (4, 8)\}$ sobre los cuales se realizará la búsqueda. En la misma figura se aprecian con distintos achurados todos los objetos que pertenecen a la intersección de todos los intervalos. Dichos objetos son posibles candidatos a ser solución.

Como referencia, y en términos de evaluaciones de distancia, los valores para la construcción y búsqueda usando 4, 8, 12, 16 y 20 se muestran en la figura 2. Para la búsqueda se construye la estructura con el 90 % de los datos, dejando el 10 % restante como consultas.

3. Implementación en Memoria Secundaria

Dado que *spaghettis* considera, en el momento de una consulta, solamente los intervalos definidos para cada pivote, entonces no es necesario cargar completamente la estructura. Este trabajo está enfocado principalmente a éste punto, es decir, en redefinir la estructura de tal modo de no hacer necesario cargar la totalidad de la estructura en memoria principal. Considerando que en memoria secundaria, no sólo se debe considerar el costo de evaluaciones de distancias realizadas, si no también se debe tomar en cuenta los accesos a disco, como ser, lecturas, escrituras o movimientos de cabezal (*reads, writes, seeks*) y el uso adecuado de las páginas de disco.



(a) Construcción

(b) Búsqueda

Figura 2: Spaghettis: Cálculos de distancia para la construcción y búsqueda en RAM.

Uno de los objetivos más importantes en la implementación de la estructura en memoria secundaria es la reducción de los movimientos de cabezal, lo que se logra recorriendo el archivo en forma secuencial. Queda de manifiesto que las soluciones para estructuras de búsquedas exactas son inadecuadas, debido a que no se puede mantener algún orden en el sentido tradicional. Sin embargo, se puede lograr cierta secuencialidad de acceso al disco organizando los objetos según alguna característica, por ejemplo, distancias a un determinado objeto. Gran parte de las estructuras métricas existentes realizan una implementación de ésto, pero no garantiza la disminución de saltos dentro de un archivo. Un objeto, en el caso de los espacios métricos, puede estar cerca de otros objetos, lo que implica que durante una búsqueda por un mismo rango, este objeto sería respuesta a varias consultas y por lo tanto es indiferente donde se agrupe.

Para los experimentos mostrados en este capitulo se asumió que la estructura no entra completamente en memoria principal y que sólo se dispone de 2Mbytes de RAM. Las páginas de disco serán de 4Kbytes y los pivotes utilizados en las tres implementaciones serán los mismos.

3.1. Implementación Original

Con esta implementación se pretende construir el índice en memoria secundaria según su diseño original, para ésto, cada nodo almacena los mismos parámetros que la implementación de *spaghettis* en memoria principal, es decir, la distancia del objeto al pivote y la ubicación del objeto en el siguiente pivote. También cada nodo guarda la posición en disco del siguiente nodo dentro del archivo que almacena el índice.

La figura 3 muestra la forma de los nodos para la primera implementación. Para la construcción, se insertan las distancias a los pivotes y posición en el siguiente pivote hasta la capacidad máxima por cada nodo (4Kbytes) y luego por cada pivote, posteriormente se realiza el ordenamiento. Cada nodo se va almacenando luego de llenado y de acuerdo a la capacidad máxima de nodos que soporta la RAM. La figura 3 muestra la estructura una vez finalizada la construcción, en este ejemplo el nodo tiene un tamaño de 44 bytes y un máximo de 5 objetos (representación de 5 objetos). Adicionalmente al *spaghettis*, se mantiene una estructura que identifica a los nodos, ésta almacena la distancia mínima y máxima más la posición en memoria secundaria donde se encuentra el nodo. Esta estructura permite realizar las búsquedas más eficientemente, sabiendo que nodo cargar en RAM según el intervalo que tenga.

En la figura 3, se muestra también un ejemplo de búsqueda para una consulta q con $d(q, p_i) = \{6, 8, 3, 7\}$ y $r = 3$. Las celdas levemente oscurecidas corresponden a los intervalos de búsqueda y la celda más oscura a el objeto candidato a respuesta.

Como se observa, se debe leer de disco k nodos para cada objeto que sea candidato a solución, siendo k el

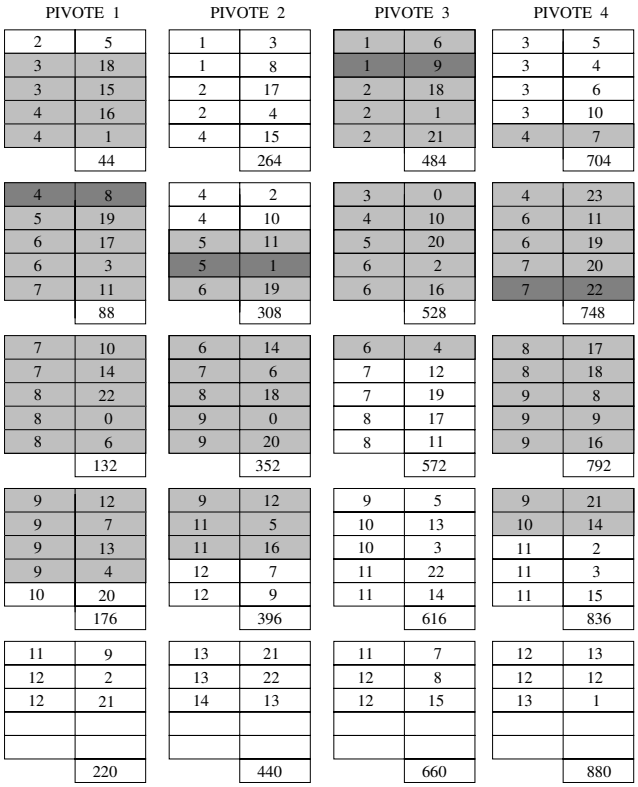


Figura 3: Spaghettis: Representación de la estructura original en memoria secundaria. Ejemplo de intervalos de búsqueda para cada pivote.

Pivotes	Nodos	Espacio	Writes	Seeks	Reads	% Página Usado
4	608	2.4 Mb	1674	66310	66310	100
8	1216	4.8 Mb	3498	154094	154014	100
12	1824	7.2 Mb	5322	240112	238416	100
16	2432	9.6 Mb	7146	319511	317199	100
20	3040	12 Mb	8970	406180	403252	100

Tabla 1: Spaghettis: Información general de la estructura en disco.

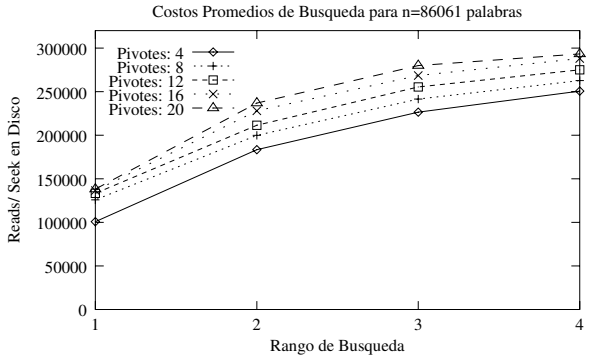


Figura 4: Costos promedios de lecturas en disco para la búsqueda.

número de pivotes de la estructura. Lo anterior es sin considerar aquellos nodos leídos y que finalmente no tienen candidatos. Esto puede resultar en extremo costoso si la cantidad de pivotes es elevada.

La tabla 1 muestra información general de la estructura en disco, como ser el número de pivotes, el total de nodos generados, el espacio de la estructura en memoria secundaria, las cantidades totales de escrituras, movimientos de cabezal y lecturas, finalmente se muestra el porcentaje usado de la página.. Se consideró el tamaño de la página de 4096 bytes y la cantidad de datos soportada por cada nodo fue de 511.

Lo interesante de una estructura de tipo arreglo es que la utilización de las páginas de disco puede resultar bastante eficiente, como en este caso.

3.2. Alternativas de Implementación

Si se observan los resultados experimentales obtenidos con la implementación anterior, se puede apreciar que existe una gran cantidad de accesos a disco durante la búsqueda, esto provoca que dicha alternativa no sea competitiva. Este problema se debe principalmente a la forma de la estructura.

A continuación se presentan alternativas de almacenamiento que reducen los costos de acceso a disco y espacio utilizado por la estructura.

Las nuevas alternativas utilizan menos páginas de disco para su construcción, principalmente porque el índice ya no es construido en base a los pivotes, si no, en base a los objetos, evitando tener que almacenar información adicional como por ejemplo: la ubicación de un objeto dentro del siguiente pivote.

3.2.1. Optimización en Memoria Secundaria

La segunda alternativa redefine la estructura y almacena conjuntamente la distancia del objeto a todos los pivotes y la ubicación del objeto en la base de datos original. Además, se modifica el algoritmo de construcción ordenando

Pivotes	Nodos	Nº Objetos	Espacio	% Página Usado
4	378	204	1.5 Mb	99.8047
8	686	113	2.7 Mb	99.5117
12	994	78	3.9 Mb	99.2188
16	1291	60	5.1 Mb	99.8047
20	1614	48	6.3 Mb	98.6328

Tabla 2: Características comunes en memoria secundaria para las últimas dos alternativas.

los datos sólo en base al primer pivote. La forma de esta estructura se muestra en la figura 5 (a). Finalmente el nodo almacena la posición en disco del nodo siguiente.

La figura 5 (a) muestra el proceso de construcción una vez terminado, es decir, luego del ordenamiento de los objetos en base al primer pivote. La misma figura muestra celdas más oscuras que representan la búsqueda para una consulta q con $d(q, p_i) = \{6, 8, 3, 7\}$y $r = 3$. Se muestran oscurecidas todas las celdas de todos los pivotes que están dentro del intervalo. Sin embargo, en términos reales se descarta inicialmente en base al primer pivote.

3.2.2. Múltiples Spaghettis

La tercera alternativa es basada en la segunda, es decir, almacena todas las distancias de un objeto a los pivotes y los ordena en base al primero. La diferencia radical con la versión anterior es que se crean múltiples spaghettis, donde cada spaghetti será del tamaño de la memoria principal. De esta manera se evitan las lecturas y escrituras adicionales producto del ordenamiento posterior a la construcción. La figura 5 (b) muestra la última versión. En esta figura se entiende que el tamaño de los nodos es de 44 bytes y la capacidad en RAM es de 2 nodos.

Para la búsqueda se realiza un proceso similar al de la segunda versión, sin embargo, en la última versión necesariamente se deben revisar todos los spaghettis, lo que implica incrementar levemente las lecturas a disco.

Para las dos últimas alternativas se utiliza una estructura adicional para almacenar información de los nodos, como es: mínima y máxima distancia al primer pivote y la posición en memoria secundaria del nodo.

4. Resultados experimentales

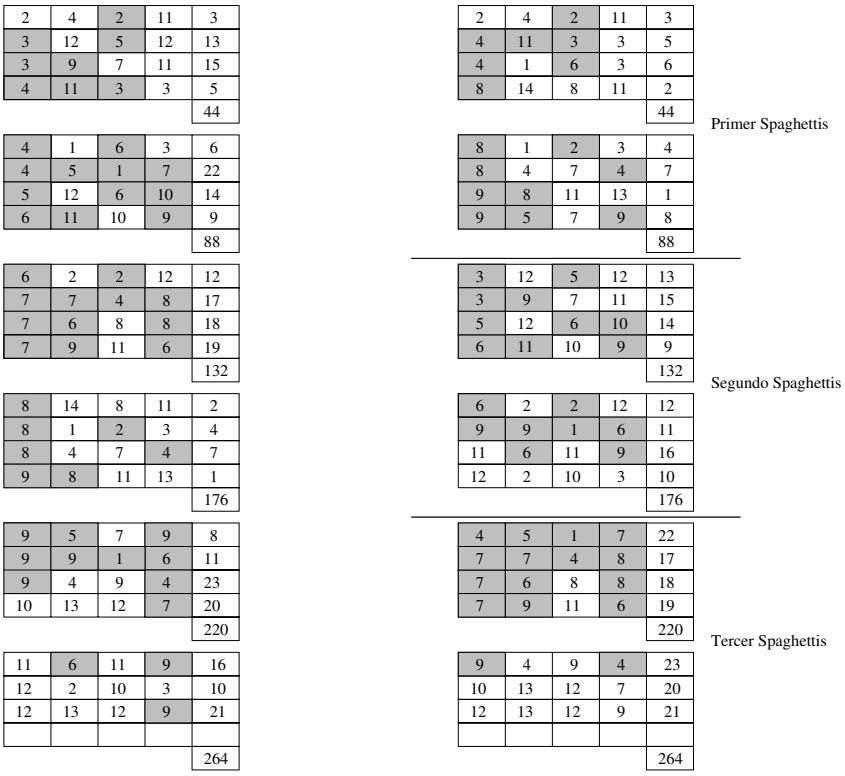
Las nuevas alternativas propuestas para el almacenamiento de la estructura resultan interesantes, principalmente porque hay una optimización en los costos de acceso a disco y de espacio utilizado por la estructura. La tabla 2 muestra información general equivalente en ambas estructuras. La información resulta ser igual en ambas debido a que la diferencia es a nivel del ordenamiento de las celdas.

Durante la construcción la diferencia entre la segunda y tercera alternativa es que la última no requiere de lecturas en disco, y por lo tanto tampoco de movimientos de cabezal, por lo que dichos valores son cero. Respecto de las escrituras, éstas también disminuyen, y en la última son del orden de la cantidad de páginas de disco generada por la estructura. La tabla 3 muestra las diferencias entre las dos estructuras en términos de accesos a disco durante la construcción.

La figura 6 muestra gráficos comparativos durante la búsqueda para las dos alternativas. La diferencia es muy levemente favorable a la alternativa 2.

De la información obtenida en los experimentos se puede deducir que entre la implementación original (alternativa 1) y la segunda implementación, hay una reducción de un 55 % en los writes a disco, para 4 pivotes. Para 8, 12, 16 y 20 pivotes la reducción de los writes es del orden del 64 % en promedio. En el caso de los reads y seeks, estos se reducen cerca de un 99 %.

Entre la primera y la tercera alternativa, los valores son de un 77.3 de reducción en los writes con 4 pivotes, de un 81.5 para el resto de los pivotes. En el caso de las lecturas y movimientos de cabezales es de un 100 %.



(a) Alternativa 2

(b) Alternativa 3 (multi spaghettis)

Figura 5: Alternativas en Memoria Secundaria: Construcción y representación de intervalos de búsqueda.

Respecto de las lecturas en disco durante las búsquedas, éstas se ven notoriamente reducidas con las últimas dos alternativas, del orden del 90 % en comparación con la primera.

Es importante señalar que tanto en la construcción como en las búsquedas, los costos debido a los cálculos de distancias son exactamente iguales en las 3 implementaciones.

5. Conclusiones

5.1. Aspectos Relevantes y Aportes

Un gran número de las estructuras que han sido desarrolladas para búsquedas por similaridad están diseñadas sólo como prototipos para memoria principal. Por lo tanto, es relevante contar con estructuras sobre memoria secundaria de tal manera que puedan ser implementadas en aplicaciones reales. En este sentido, se considera que el aporte más relevante del presente trabajo es la presentación de 2 alternativas de la estructura *Spaghettis* para su utilización en memoria secundaria.

Dichas alternativas presentan reducciones considerables en los costos de accesos a disco, tanto en la construcción, como en la búsqueda (sección 4). En función del espacio utilizado en disco, la reducción es del orden de un 40 % y

Alternativa 2			Multispaghetis
Writes	Seeks	Reads	Writes
399	761	780	380
721	1373	1408	688
1030	1989	2025	996
1342	2583	2634	1293
1676	3229	3291	1616

Tabla 3: Accesos a disco para las alternativas 2 y 3 (multispaghettis).

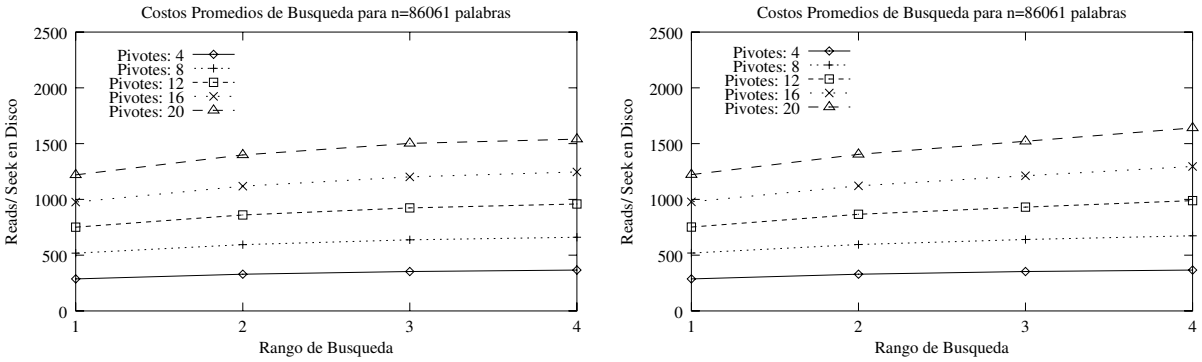


Figura 6: Costos de lectura para las alternativas 2 y 3 (multispaghettis).

mayor a medida que se aumenta la cantidad de pivotes.

La reducción de los costos de lectura se debió principalmente a la redefinición de la estructura, es decir, a la agrupación de los objetos y sus distancias a los pivotes, de esta manera se puede discriminar con una sola lectura en disco si un objeto es o no candidato.

Con el presente trabajo se ha comprobado que la estructura resultante es más competitiva, desde el punto de vista de espacio en memoria secundaria, que muchos índices del tipo árbol. Ya que en la mayoría de las estructuras de tipo árbol, el gran inconveniente que se presenta es la inadecuada utilización de páginas de disco lo que genera una estructura de excesivo tamaño. Por ejemplo, al realizar la construcción, en base a un diccionario en español de 86.061 objetos y utilizando 20 centros (pivotes), en el *gnat* se tiene que el número de páginas utilizadas fue de 15.405 y en el caso del *egnat* 4.057 [Uri05]. En las implementaciones realizadas en este trabajo, en el caso de la implementación original, el número de páginas utilizadas fúe de 3.040 y en las alternativas presentadas 1.614 páginas.

5.2. Trabajos Futuros

Algunas extensiones futuras que se están haciendo en base al presente trabajo son:

- Analizar y desarrollar ideas de modo que las implementaciones propuestas, especialmente la segunda alternativa, obtenga capacidades dinámicas. Por ejemplo para la reinserción de objetos, se puede ir creando nuevos *subspaghettis* de modo de evitar reconstruir la estructura por completa, de tal manera de usar ordenar sólo el último *subspaghetti*.
- Analizar y probar si es posible aplicar métodos existentes de eliminación para estructuras basadas en particiones compactas, como es el método de planos fantasmas [Uri05], en estructuras basados en pivotes. En este sentido, en el caso de la eliminación de objetos para la tercera versión de *spaghettis*, se podría hacer realizando un intercambio entre el objeto a eliminar y uno del o los último(s) *subspaghettis* que estén a la misma distancia del primer pivote.
- Generar nuevos algoritmos para mejorar los costos de búsquedas sobre las estructuras propuestas.
- Obtener distintas formas de administración de páginas para las implementaciones propuestas.
- Probar las distintas alternativas presentadas, sobre otras estructuras de tipo arreglo.
- Distribuir y paralelizar la estructura *spaghettis*.
- Encontrar valores óptimos de números de pivotes y determinar los efectos cuando la cantidad de pivotes supera el óptimo.

Referencias

[BK73] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communication of ACM*, 16(4):230–236, 1973.

[Bri95] Sergei Brin. Near neighbor search in large metric spaces. In *the 21st VLDB Conference*, pages 574–584. Morgan Kaufmann Publishers, 1995.

[BYCMW94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixedqueries trees. In *5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[CMBY99] E. Chavéz, J. Marroquín, and R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS Press, 1999.

[CNBYM01] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José L. Marroquín. Searching in metric spaces. In *ACM Computing Surveys*, pages 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 *the 23st International Conference on VLDB*, pages 426–435, 1997.

[MOV94] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[Nav02] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[TTSF00] Caetano Traina, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *VII International Conference on Extending Database Technology*, pages 51–61, 2000.

[Uhl91] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, pages 40:175–179, 1991.

[Uri05] Roberto Uribe. Manipulación de estructuras métricas en memoria secundaria. Master's thesis, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Santiago, Chile, Abril 2005.

[Yia93] P. Yianilos. Data structures and algoritms for nearest neighbor search in general metric spaces. In *4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.