Spaghettis en Memoria Secundaria: Una Implementación Paralela

Carlos Subiabre[1], Enrique Árias[2], and
Roberto Uribe-Paredes[1,3]

[1] Depto. de Ingeniería en Computación[**],
Universidad de Magallanes, Chile
[2] Departamento de Sistemas Informáticos,
Escuela Politécnica Superior de Albacete, Universidad de Castilla-La Mancha, Albacete, España
[3] Grupo de Bases de Datos - UART,
Universidad Nacional de la Patagonia Austral, Rio Turbio, Argentina
[4] E-mail: ruribe@ona.fi.umag.cl

Abstract Similarity search in metric spaces allow to retrieve similar or alike object to query.
The present work show a development of parallel and enhanced for secondary memory version of *Spaghettis* metric data structure. It is pivot-based structure and array-type.
Parallel implementation it is of thickness grain and using MPI. Remarkable performance have been obtained in distance evaluations, execution time and in terms of speed-out and efficiency. Los resultados experimentales demuestran una notable reducción en los costos de evaluaciones de distancia, en los tiempos de ejecución y en términos de speed-up y eficiencia.

Keywords: Databases, data structures and algorithms, similarity search, parallel processing, MPI.

Resumen La búsqueda por similaridad en espacios métricos permite recuperar objetos parecidos o similares a una consulta dada.
El presente artículo muestra el desarrollo de una versión paralela y optimizada para memoria secundaria de la estructura de datos métrica *Spaghettis*. Esta estructura está basada en pivotes y es del tipo arreglo. La implementación paralela es de grano grueso y utilizando MPI. Los resultados experimentales demuestran una notable reducción en los costos de evaluaciones de distancia, en los tiempos de ejecución y en términos de speed-up y eficiencia.

Keywords: Estructuras de datos y algoritmos, bases de datos, búsqueda por similaridad, paralelismo, MPI.

1. Introducción

1.1. Antecedentes

Con la rápida evolución de las tecnologías de la información han surgido nuevos depósitos no estructurados de datos tales como texto libre, imagen, sonido y video. Realizar búsquedas exactas sobre estos datos sería poco útil. Por ejemplo, si se consultase por un elemento sobre una base de datos de imágenes, la consulta sólo podría encontrar su copia digital exacta en la base de datos. El verdadero interés reside, por ejemplo, en consultar sobre una base de datos de fotografías una imagen que contiene un rostro, donde no necesariamente existe una copia exacta de la misma fotografía; identificación de individuos a través de dispositivos biométricos, donde el dato consulta (voz, retina, etc) podría verse afectado por factores externos; encontrar una especie más parecida a otra en una base de datos de cadenas de ADN, etc. Este tipo de

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

búsqueda recibe el nombre de *búsqueda por similaridad* y consiste en recuperar todos los objetos mas relevantes o parecidos a una consulta dada.
Para manipular dichos datos, se deben generar estructuras que permitan almacenarlos y realizar búsquedas sobre ellos. Estructurar este tipo de datos es dificultoso ya sea manual o computacionalmente y restringe de antemano los tipos de búsqueda posibles.
En la actualidad, la mayoria de las estructuras han sido diseñadas como prototipos y carecen de dos características importantes, dinamismo y un adecuado desempeño en memoria secundaria. La segunda, que es la más escasa, determina el desempeño en términos de accesos a disco y espacio utilizado por la estructura. Finalmente, la necesidad de procesar grandes volumenes de datos obligan a aumentar la capacidad de procesamiento y con ello la paralelización de algoritmos y distribución de la base de datos.

1.2. Marco Teórico

La similaridad, en muchos casos, es modelada a través de un *espacio métrico* y la búsqueda de objetos más similares bajo una función conveniente de similaridad, a través de una búsqueda por rango o 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 $d(x, y) \leq r$.

El objetivo de los algoritmos de búsqueda es minimizar la cantidad de evaluaciones de distancia realizadas al 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.

Existen dos métodos para las construcción de estructuras métricas, los basados en Clustering y los basados en Pivotes. El primero divide el área en particiones de Voronoi, donde existe un centro por cada área y los demás objetos se almacenan en el centro más cercano.
En el caso de los *Algoritmos Basados en Pivotes*, un pivote es un objeto preseleccionado y que no necesariamente pertenece a la base de datos. Su objetivo es filtrar objetos en una consulta a través de la utilización de la desigualdad triangular, sin medir realmente la distancia entre el objeto consulta y los objetos descartados.

Una visión abstracta de los algoritmos basados en pivotes es la siguiente:

- Se selecciona un conjunto de k pivotes $\{p_1, p_2, \ldots, p_k\} \in X$. En tiempo de indexamiento, para cada objeto x de la base de datos Y se calcula y almacena su distancia a los k pivotes $(d(x, p_1), \ldots, d(x, p_k))$.
- Dada una consulta (q, r), por desigualdad triangular se tiene que $d(p_i, x) \leq d(p_i, q) + d(q, x)$, con $x \in X$, de la misma forma se tiene que $d(p_i, q) \leq d(p_i, x) + d(q, x)$. De estas inecuaciones se tiene que una cota inferior para la distancia entre q y x es $d(q, x) \geq |d(p_i, x) - d(p_i q, q)|$. Como los objetos x que intersectan son aquellos en donde $d(q, x) \leq r$, entonces se pueden excluir todos los objetos que no cumplan con la condición de la ecuación (1).

$$|d(q,p_i) - d(x,p_i)| \leq r, \forall i = i...k \quad (1)$$

En otras palabras, si para algún pivote p_i se cumple que $|d(q,p_i) - d(x,p_i)| > r$, entonces por desigualdad triangular se conoce que $d(q,x) > r$ y por lo tanto, no es necesario evaluar explicitamente $d(x,q)$. Todos los objetos que no se puedan descartar por esta regla deben ser comparados directamente con la consulta.

Existen distintas estructuras para buscar en espacios métricos, las cuales pueden ocupar funciones discretas o continuas de distancia. Algunos son BKTree [3], MetricTree [11], GNAT [2], VpTree [14,5], FQTree [1], MTree [6], SAT [9,10], EGNAT [12].

El presente artículo propone una estrategia de distribución de carga sobre procesadores paralelos permitiendo una búsqueda más eficiente dada una consulta por rango. La estructura fue acondicionada para un buen desempeño en memoria secundaria en términos de accesos a disco. Para la implementación paralela se utilizó la librería llamada MPI (Message-Passing Interface), la cual proporciona independencia de la arquitectura y ha demostrado eficiencia en aplicaciones tales como texto y otros.

2. *Spaghettis* en Memoria Secundaria

La estructura *spaghettis* [4] es una variante de *LAESA* [8]. La Estructura aquí presentada varía de su algoritmo original ya que está optimizada para memoria secundaria [13]. Es una estructura del tipo arreglo y su algoritmo está basado en pivotes.

2.1. Construcción de Multiples *Spaghettis*

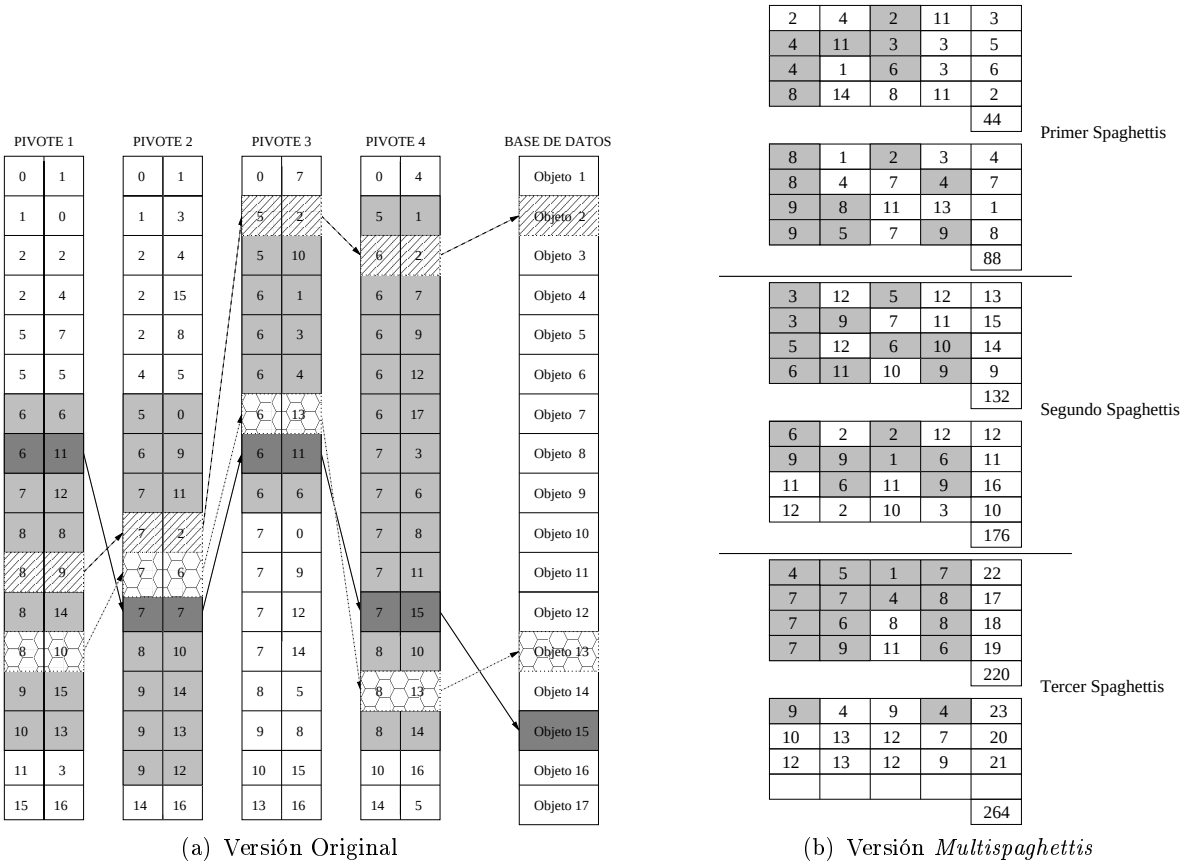
- Se selecciona un conjunto de k pivotes.
- Se calcula la distancia de cada objeto a todos los pivotes y se insertan en forma ordenada en los nodos, este orden se hace en base a la distancia del objeto con el primer pivote. Cada nodo corrresponderá a una página de disco.
- Al momento de tener la cantidad máxima de nodos soportados en RAM y estos estén totalmente llenos se procede a su almacenamiento en memoria secundaria, posteriormente se continúa con la construcción de un nuevo *spaghettis*.
- Al momento de ir guardando los nodos, uno a uno, se utiliza una estructura que almacena la ubicación del primer nodo del conjunto almacenado, de forma de poder determinar donde se encuentra ubicado dentro del índice el subconjunto de nodos formados.

En resumen, la diferencia entre la versión original de la estructura y ésta, es que se tienen N *spaghettis* o *subspaghettis*, con capacidad para *max* nodos, que corresponden al número de nodos que soporta la RAM.

2.2. Búsqueda

- Se determina la distancia entre la consulta q y cada pivote p_k, obteniendo así un intervalo por cada pivote de la forma:
$$[d(q,p_k) - r, d(q,p_k) + r] \quad (2)$$
- Se carga en RAM nodo a nodo hasta encontrar, un valor de distancia al primer pivote, que satisfaga el intervalo (2) para éste pivote.
- Un objeto candidato a la consulta es aquel que se encuentra en la intersección de los k intervalos formados.

- Para cada objeto candidato obtenido, se calcula la distancia entre él y la consulta q. Si la distancia obtenida está dentro del rango r, entonces el objeto es respuesta a la consulta realizada. Es decir, se cumple que $d(x, q) \leq r$.
- Lo anterior se realiza para cada *subspaghettis*.



(a) Versión Original (b) Versión *Multispaghettis*

Figura 1. Representación de dos versiones de *spaghettis* una vez finalizado el proceso de construcción y mostrando los intervalos obtenidos para distintas búsquedas.

Para notar la diferencia entre ambas estructuras (ver figura 1), cabe mencionar que en la estructura original las columnas de distancias están ordenados según la distancia a cada pivote, en la nueva versión sólo están ordenados según el primer pivote y por *subspaghetti*.

En la figura 1(a) se representa la forma original de la estructura *spaghettis* (solo para RAM). Ésta está 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(a) 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.
La opción implementada en este trabajo es la explicada anteriormente (*multispaghettis*). La figura 1(b) representa esta estructura una vez finalizada la construcción, es decir, luego del ordenamiento de los objetos en base al primer pivote (en el ejemplo, usando 23 objetos). En esta implementación cada fila representa a un objeto y cada columna de cada objeto corresponde

a la distancia al pivote. La ultima columna representa el identificador del objeto. En el ejemplo se asume que una página de disco mide 44 bytes y que la RAM sólo tiene capacidad para dos nodos, por ello se tienen 3 spaghettis. En la figura 1(b), se muestra también un ejemplo de 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. Implementación Paralela de la Estructura Spaghettis

El objetivo principal para el presente trabajo, es reducir los *tiempos de ejecución* y *evaluaciones de distancia* que se realizan durante la búsqueda, así como obtener para que pivote o pivotes la búsqueda se comporta más eficiente.

Los resultados experimentales mostrados en el artículo corresponden a una estructura construida en el front-end (o broker) y replicada a todos los procesadores. Las consultas fueron distribuidas en porcentajes similares sobre el clúster de PCs, procesadas localmente y las respuestas recuperadas fueron recopiladas por el front-end.

Para los experimentos se utilizó un espacio de palabras en español de 86.061 objetos, de donde el 90 % se utilizó para construir la estructura y el restante 10 % corresponde al conjunto de consultas que se distribuyen en el cluster.

Para medir la distancia en un espacio de palabras, se utilizó la *distancia de edición* o *distancia de Levenshtein*. La distancia de edición se define como la cantidad de inserciones, eliminaciones o modificaciones de caracteres que deben realizarse sobre una palabra para convertirla en otra.

Las pruebas realizadas se efectuaron construyendo la estructura para 4, 8, 12, 16 y 20 pivotes. La selección de pivotes es aleatoria y son extraídos desde la base de datos. El tipo de búsquedas fue para *rangos* 1, 2, 3 y 4.

Los experimentos fueron ejecutados sobre un clúster compuesto por 9 procesadores (un front-end y 8 procesadores para cómputo). Cada procesador esta compuesto por un Procesador Intel PIV de 3.00 GHz y 3GB de RAM. Este corre sobre el sistema operativo Debian Linux. Las librerias BLAS y SPARSKIT han sido compiladas en esta máquina para obtener el mejor funcionamiento. Para realizar la paralelización se utilizó MPI [7] (Interfaz de Paso de Mensajes) con una plataforma de memoria distribuida.

4. Resultados Experimentales

Los resultados obtenidos en esta implementación paralela serán evaluados en términos de:

- Tiempo de ejecución: Es el tiempo que lleva solucionar el problema.
- Speed-up: Se define como la proporción de tiempo que toma solucionar el problema sobre un procesador y el tiempo requerido para solucionar el mismo problema sobre un computador paralelo con p procesadores idénticos.
- Eficiencia: Es la medida de fracción de tiempo durante el cual un procesador es empleado útilmente; es definido como la proporción entre el Speed-up y el número de procesadores.

En las figuras 2, 3, 4 y 5 se presentan los gráficos de: Tiempos de Ejecución, Speed-up, Eficiencia y Evaluaciones de Distancia (E.D.) respectivamente. Los resultados corresponden a la estructura *spaghettis* construida para 4, 8, 12, 16 y 20 pivotes y búsquedas con rango 1, 2,

3 y 4. Los resultados se presentan para el caso secuencial (1 procesador) y la implementación paralela (2, 4 y 8 procesadores).

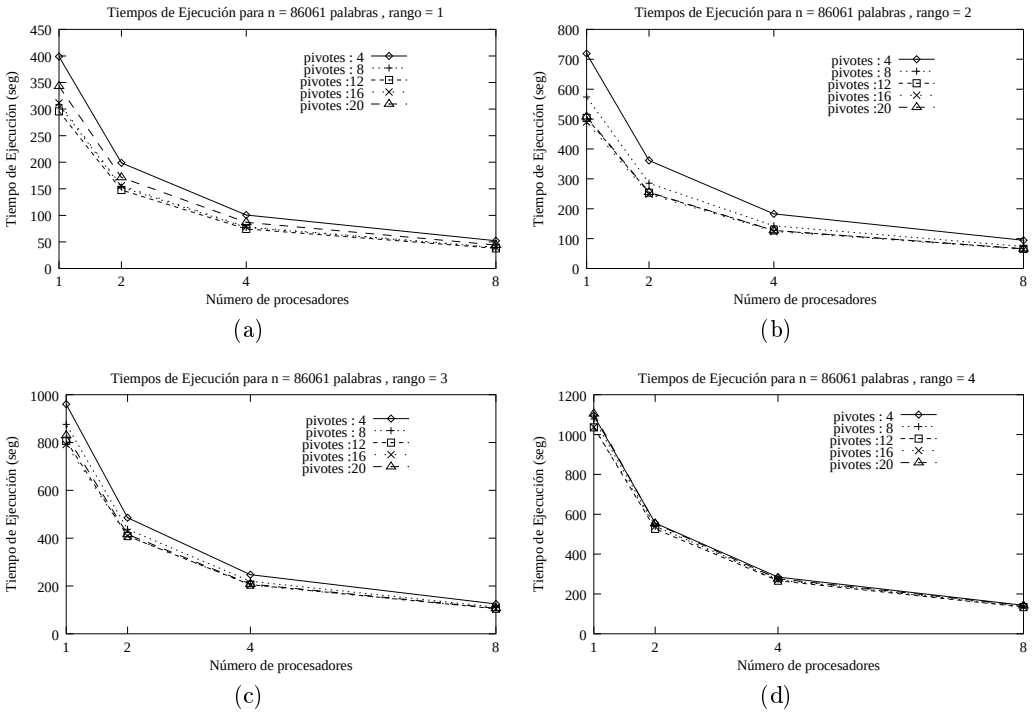


Figura 2. Gráficos de Tiempos de Ejecución

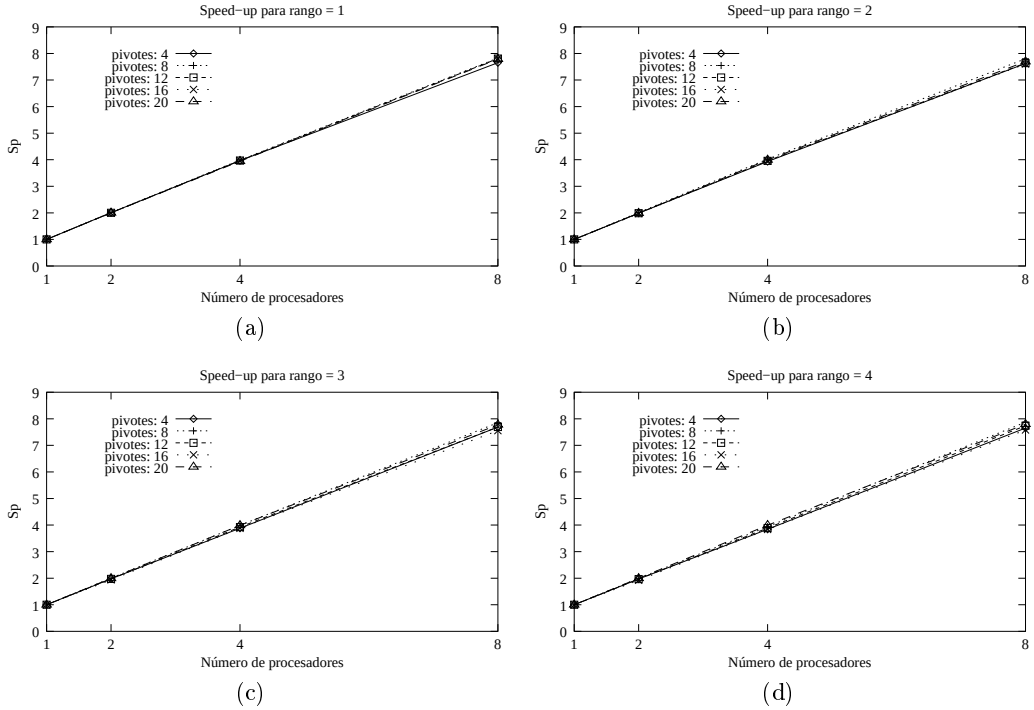


Figura 3. Gráficos de Speed-up

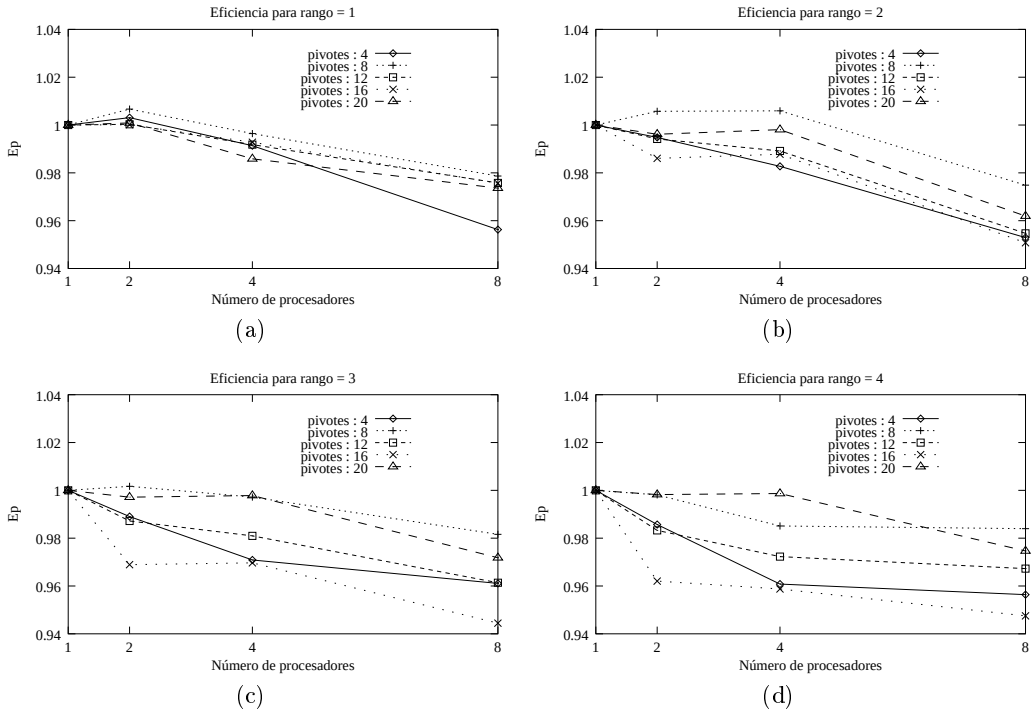


Figura 4. Gráficos de Eficiencia

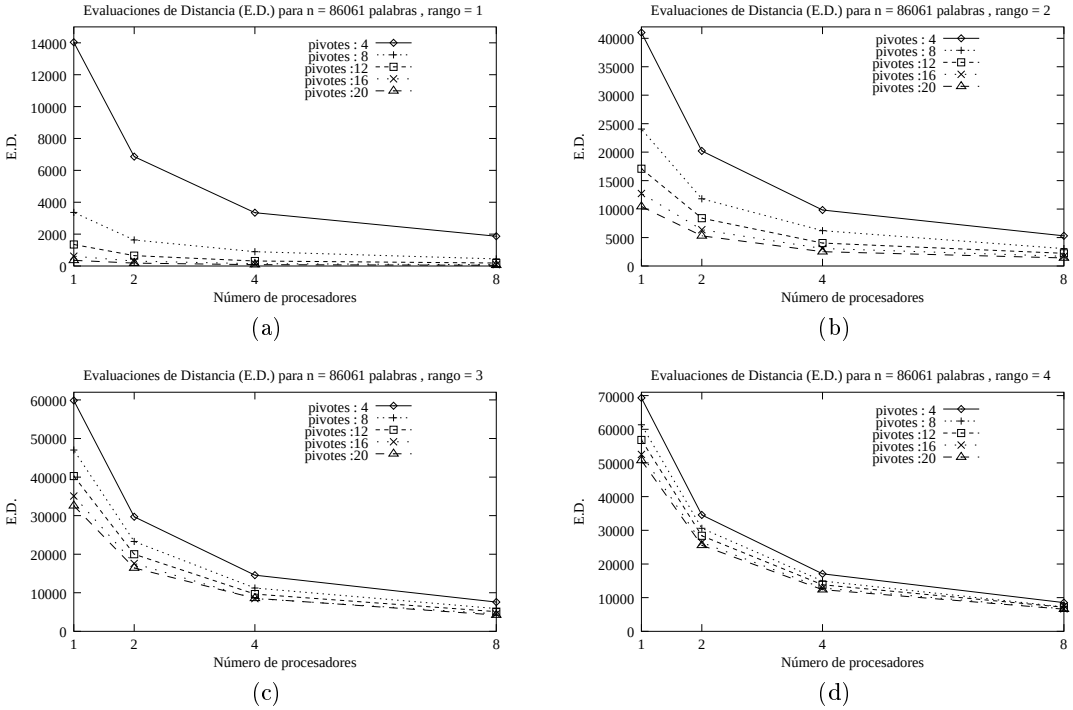


Figura 5. Gráficos de Evaluaciones de Distancia (E.D.)

5. Conclusiones

5.1. Aspectos Relevantes y Aportes

La mayor parte de las estructuras que se han desarrollado para búsquedas por similaridad estan diseñadas sólo como prototipos para memoria principal. En este sentido, se considera que el aporte más relevante de éste artículo es la presentación de una versión paralela de la estructura *Spaghettis*, optimizada para memoria secundaria, lo que permite contar con una estructura que puede ser preparada para aplicaciones reales.

De acuerdo con los resultados experimentales los tiempo de ejecución y las evaluaciones de distancia se han reducido notablemente para la estructura construída con cada número de pivotes y los rangos de búsquedas utilizados para estos. En la utilización de 4 pivotes versus 8 pivotes y la realización de búsquedas para rangos 1 y 2 (ver Figuras 2(a) y 2(b)) es donde más se acentúa esta disminución de tiempo.

En general los costos de tiempo de ejecución se han reducido significativamente debido al balance de consultas que existe, donde cada procesador tiene acceso al mismo número de estas. Por otra parte, la reducción de *evaluaciones de distancia*, el proceso mas costoso dentro de la búsqueda, influye en gran medida en la reducción de estos tiempos. En la Tabla 1 se presentan los porcentajes de disminución de evaluaciones de distancia de acuerdo al número de procesadores utilizado.

num. procs	4 Pivotes	8 Pivotes	12 Pivotes	16 Pivotes	20 Pivotes
2	50.6 %	50.8 %	50.8 %	49.8 %	49.3 %
4	75.7 %	74.8 %	76.2 %	75.9 %	75.4 %
8	87.2 %	87.4 %	87 %	86.7 %	86.6 %

Cuadro 1. Promedios de disminución de Evaluaciones de Distancia.

Conforme aumenta el rango de búsqueda los tiempos de ejecución para todos los pivotes es muy similar llegando incluso a ser casi idénticos para las búsquedas de rango 4 (ver Figura 2). Esto se debe a que para consultas de mayor rango son más los objetos que se recuperaran, lo que implica recorrer en mayor medida la estructura.

Por último, teniendo en cuenta la Figura 4, se puede apreciar que la estructura construída mediante 8 pivotes es quien mejor se comporta con respecto al algoritmo secuencial. Esto para cada número de procesadores y rangos de búsquedas utilizados.

6. Trabajos Futuros

- Emplear heurísticas para la selección de pivotes. Entre estas, las propuestas de elegir objetos más alejados entre sí y la selección de objetos espacialmente esparcidos.
- Probar el comportamiento de la estructura con otros tipos de espacios métricos.
- Implementar métodos de eliminación y en general modificar la estructura para que sea dinámica.
- Implementar distintas alternativas de distribución de la base de datos sobre la estructura paralela.

7. Agradecimientos

Lo autores desean dar expresos agradecimientos al grupo *RETICS* del *Instituto de Investigación Informática de Albacete* (*I3A*) de la Universidad de Castilla La Mancha, España por las facilidades otorgadas en el uso de sus instalaciones.

Referencias

1. 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.
2. Sergei Brin. Near neighbor search in large metric spaces. In *the 21st VLDB Conference*, pages 574–584. Morgan Kaufmann Publishers, 1995.
3. W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communication of ACM*, 16(4):230–236, 1973.
4. 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.
5. T. Chiuch. Content-based image indexing. In *The 20th Conference on Very Large Databases (VLDB'94)*, pages 582–593, 1994.
6. 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.
7. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with Message-Passing Interface*. MIT Press, 1994.
8. 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.
9. Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

10. Nora Reyes. Índices dinámicos para espacios métricos de alta dimensionalidad. Master's thesis, Universidad Nacional de San Luis, Argentina, 2002.
11. J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, pages 40:175–179, 1991.
12. Roberto Uribe-Paredes. Manipulación de estructuras métricas en memoria secundaria. Master's thesis, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Santiago, Chile, Abril 2005.
13. Roberto Uribe-Paredes and Christian Cárdenas. Spaghettis en memoria secundaria. In *XII Congreso Argentino de Ciencias de la Computación (Cacic2006)*, San Luis, Argentina, Oct. 2006.
14. 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.