Evaluación del Rendimiento de CMSNMS en Espacios Métricos Anidados

Cristina Bender[1], Nora Reyes[2], Hugo Gercek[1], Claudia Deco[1]
[1] Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional de Rosario
Rosario, Argentina
{ bender, deco }@fceia.unr.edu.ar, hugogercek@gmail.com
[2] Departamento de Informática. Universidad Nacional de San Luis
San Luis, Argentina
nreyes@unsl.edu.ar

Resumen. La mayoría de los métodos de búsqueda en espacios métricos asumen que la topología de la colección de objetos es razonablemente regular. Sin embargo, se sabe de la existencia de Espacios Métricos Anidados, que son espacios en donde los objetos de la colección pueden agruparse en *clusters* o subespacios. Aquí diferentes dimensiones explican las diferencias entre los objetos dentro de cada subespacio anidado dentro de un espacio métrico más general. En este trabajo se evalúa el rendimiento del *CMSNMS* que es una estructura de índice de dos niveles para resolver problemas de búsquedas en espacios de esta topología. En un primer nivel esta técnica utiliza una *Lista de Clusters* (*LC*), donde se identifican y ordenan estas agrupaciones utilizando el *Sparse Spatial Selection* (*SSS*) y técnicas de *LC*. En un segundo nivel se genera un índice por cada *cluster* denso, basado en selección de pivotes, empleando también *SSS*. Las experimentaciones muestran que el desempeño de *CMSNMS* es mejor que el de las demás en los espacios métricos anidados.

Palabras clave: espacios métricos anidados, búsqueda por similitud, bases de datos métricas.

1 Introducción

Actualmente, se tienen almacenamientos no estructurados de información donde se realizan consultas sobre tipos de datos tales como texto libre, imágenes, audio y video; y en algunos casos ya no se puede estructurar más esta información en claves y registros. Estos escenarios requieren modelos más generales tales como las bases de datos métricas y herramientas que permitan realizar búsquedas eficientes sobre estos tipos de datos. En estos repositorios resultan poco útiles las búsquedas por igualdad, y surge como concepto unificador las búsquedas por similitud. La similitud se modeliza usando una función de distancia, y el conjunto de objetos es llamado espacio métrico. La función distancia es bastante costosa de calcular, por esto, se han desarrollado diferentes técnicas para intentar reducir el número de evaluaciones de dicha función. Estas técnicas se basan en estructuras llamadas índices. Existen dos grandes grupos de técnicas de indexación: índices basados en selección de pivotes e índices basados en particiones compactas (*clustering*). La mayoría de estas técnicas se desarrollaron

asumiendo que la topología de la colección de objetos es razonablemente regular, pero experimentaciones hechas sobre espacios donde las colecciones de objetos puede agruparse en subespacios o *clusters* han demostrado que estas técnicas no son tan eficientes. Surge así la necesidad de desarrollar técnicas que resuelvan los problemas de indexación y búsqueda en Espacios Métricos Anidados.

En [1] se presenta una estructura de dos niveles, el *Sparse Spatial Selection for Nested Metric Spaces (SSS-NMS)*, para los espacios métricos anidados, que separa los subespacios en clusters utilizando *SSS*, e indexa cada subespacio denso con *SSS*. A partir de esta estructura, en [2] se propone una nueva versión (*Combining Methods for Searches in Nested Metric Spaces*, *CMSNMS*) que también posee dos niveles de índices. El primer nivel permite identificar los agrupamientos utilizando *SSS* [3] y ordenarlos en una *Lista de Clusters* [4]. En el segundo nivel, en base a una medida de densidad, se indexan utilizando también *SSS* con pivotes los clusters que se consideren altamente poblados. Lo innovador en [2] es la forma de construcción del índice, el algoritmo de búsqueda y el hecho de que luego de una determinada cantidad de consultas se utiliza lo propuesto en [5] para adaptar los pivotes de cada subespacio en el segundo nivel a las búsquedas que se están realizando. Así, se construye una estructura que identifica los subespacios, que es eficiente en las búsquedas, y que a la vez es dinámica y adaptativa. En este trabajo se presentan los resultados de las experimentaciones realizadas para evaluar el rendimiento de *CMSNMS*.

El resto del trabajo se organiza de la forma siguiente: en la Sección 2 se presentan conceptos básicos; en la Sección 3 trabajos relacionados; en la Sección 4 se discuten las experimentaciones realizadas. Finalmente se presentan las conclusiones.

2 Conceptos Básicos

Un *espacio métrico* (X, d) consiste de un universo de objetos válidos X y una *función distancia* $d:X\times X\rightarrow\Re^+$ definida entre ellos. Esta función satisface las propiedades siguientes: positividad $d(x,y)>0$, simetría $d(x,y)=d(y,x)$, reflexividad $d(x,x)=0$, y desigualdad triangular $d(x,y)\leq d(x,z)+d(z,y)$. Un subconjunto finito U de X, con $|U|=n$, es el conjunto de elementos donde se realizan las búsquedas. La definición de la función distancia depende del tipo de objetos. En un espacio vectorial, d puede ser una función de la familia de Minkowski: $L_s((x_1, ..., x_k),(y_1, ..., y_k))=(\sum |x_i-y_i|^s)^{1/s}$.

En las bases de datos métricas las consultas pueden ser: por rango o por k-vecinos más cercanos. En el primer caso, dada una consulta q y un radio r, se recuperan los objetos que estén a una distancia menor a r $\{u\in U / d(u,q)\leq r\}$. En una consulta por k-vecinos más cercanos, se recuperan los k objetos más cercanos a la consulta, es decir: $A\subseteq U$ tal que $|A|=k$ y $\forall u\in A, v\in U-A, d(q,u)\leq d(q,v)$. La forma básica de implementar estas operaciones es comparar cada objeto de la colección con la consulta. El problema es que, en general, la evaluación de la función distancia tiene un muy alto costo computacional, por lo que buscar de esta manera no es eficiente cuando la colección tiene muchos elementos. Por esto, el objetivo principal de los métodos de búsqueda en espacios métricos es reducir la cantidad de evaluaciones de la función distancia. Con la construcción de un índice y usando la desigualdad triangular, se pueden descartar objetos sin tener que compararlos con la consulta. Existen dos tipos

de métodos de búsqueda: basados en *clustering* y basados en *pivotes* [6]. El primero parte el espacio en un conjunto de regiones de equivalencia, cada una representada en general por un centro de cluster y un radio. En las búsquedas se descartan regiones completas dependiendo del centro de cluster, la consulta y su radio. En el segundo, se selecciona un conjunto de objetos de la colección como pivotes. Se construye un índice computando las distancias de cada objeto en la base de datos a cada pivote. En las búsquedas se calculan las distancias de la consulta a cada pivote. Algunos objetos de la colección se pueden descartar utilizando la desigualdad triangular y las distancias precalculadas durante la fase de construcción del índice.

3 Trabajos Relacionados

La selección de los pivotes afecta la eficiencia del método de búsqueda, y la ubicación de cada pivote respecto a los otros determina su habilidad para excluir elementos del índice sin compararlos directamente con la consulta. Muchos métodos basados en pivotes los seleccionan aleatoriamente, y se han propuesto diversas heurísticas para la selección de pivotes. En [3] se presenta *Sparse Spatial Selection* (*SSS*), el cual selecciona dinámicamente un conjunto de pivotes bien distribuidos en el espacio métrico. La idea base es la siguiente: si los pivotes están dispersos en el espacio serán capaces de descartar más objetos durante la búsqueda. Un pivote se considera suficientemente alejado de otro si está a una distancia mayor o igual a $M*\alpha$, donde M es la máxima distancia entre dos objetos cualesquiera, y α es un parámetro constante que influye en la cantidad de pivotes seleccionados y toma valores experimentales óptimos alrededor de 0,4. En todas las técnicas analizadas para seleccionar pivotes, la cantidad de pivotes debe ser fijada previamente. En [7], resultados experimentales muestran que la cantidad óptima de pivotes depende del espacio métrico y este valor tiene una gran importancia sobre la eficiencia del método. Debido a esto, *SSS* es importante para ajustar la cantidad de pivotes tan bien como sea posible. En [5] se presenta una mejora al *SSS*, donde el índice se ajusta a las búsquedas, luego de que el índice se ajustó al espacio métrico al utilizar una selección dinámica de pivotes. La construcción inicial del índice se realiza con *SSS*, y la actualización es realizada durante las búsquedas. Otra mejora al *SSS* es el *SSS-Tree* [8] que utiliza árboles y las mejores propiedades de las técnicas de *clustering*. Su principal característica es que los centros de cluster se seleccionan utilizando *SSS*, con lo cual el número de clusters en cada nodo depende de la complejidad del subespacio que tiene asociado.

Dado que los índices pierden su eficacia a medida que la dimensión intrínseca de los datos crece, [9] presenta el índice *List of Clusters* (*LC*), basado en la partición compacta del conjunto de datos. *LC* es muy resistente a la dimensionalidad intrínseca del conjunto de datos, y, por cómo está construido, le da un orden especial a sus miembros: los *clusters* en posiciones anteriores tienen preferencia sobre los *clusters* posteriores, a la hora de contener elementos que se sitúan en regiones de intersección. A cada *cluster* de la lista, que representa un subespacio de centro c y radio r_c, se lo denomina *bola*. En el *LC* el primer centro elegido tiene preferencia sobre los posteriores en el caso de superposición de las bolas. Es decir, todos los elementos que caen dentro de la bola del primer centro son almacenados en dicho *cluster* a pesar de

que podrían estar dentro de otros. Dada una consulta *(q,r)* la idea es aprovechar esta particularidad y recorrer el *LC* inspeccionando aquellos *clusters* en los cuales la bola de consulta tenga intersección, y detener la búsqueda cuando la bola de consulta esté completamente contenida dentro del *cluster* en cuestión.

La mayoría de las estructuras de indexación y sus respectivos métodos de búsquedas fueron construidos para trabajar en colecciones de datos donde su distribución en el espacio es razonablemente regular, pero su desempeño puede depreciarse en colecciones de características irregulares. Dentro de los espacios irregulares se encuentran los *Espacios Métricos Anidados* [3]. En éstos, los objetos de la colección pueden agruparse en *clusters* o subespacios densos que están anidados dentro de un espacio más general. Pero no sólo los objetos están agrupados en estos subespacios densos, sino que además, la diferencia entre cada par de objetos de un subespacio es "explicada" por una dimensión diferente a la dimensión que explica la diferencia entre otro par de objetos de algún otro subespacio. [3] presenta el *Sparse Spatial Selection for Nested Metric Spaces (SSS-NMS)*, que es un método para tratar con este tipo de espacios. Esta estructura separa los subespacios en *clusters* utilizando *SSS*, e indexa con pivotes cada subespacio denso con *SSS*. Además, muestra comparaciones donde se obtienen mejores resultados para *SSS-NMS* con respecto a otros algoritmos sobre espacios de estas características.

En [2] se aborda el problema de la búsqueda en este tipo de espacios, y se propone una nueva estructura de índice (*CMSNMS*) que tiene por objetivo principal minimizar el tiempo de consulta. La estructura propuesta tiene dos niveles: una *Lista de Cluster*, construida con la ayuda del *SSS*, que identifica y mantiene un orden de cada subespacio anidado en el espacio métrico general; y un índice de pivotes construido con *SSS* para cada subespacio que se considere denso. En el primer nivel del índice se aprovechan las virtudes del *SSS* para identificar los subespacios. Además, al construir una *Lista de Clusters* con *SSS*, los centros están bien distribuidos y se asegura un orden de los clusters que permite optimizar la consulta. En el segundo nivel, al aplicar *SSS* para obtener los pivotes de cada cluster se obtiene un índice donde las referencias cubren la totalidad del subespacio. A la vez la estructura es dinámica y adaptativa. Dinámica, porque se puede comenzar con una colección vacía, a la que se le pueden agregar elementos. Adaptativa, en cuanto a la construcción, porque se adecúa a la complejidad del espacio, por lo que no se supone nada a priori sobre la cantidad de *clusters* necesarios, ni de sus características; al igual que no se hacen supuestos sobre el número de pivotes de cada subespacio denso. También se dice que es adaptativa, en cuanto a las búsquedas, porque luego de una determinada cantidad de consultas sobre algún subespacio sus pivotes se adaptan a las búsquedas que se están realizando.

Dada una consulta (*q*,*r*), donde *q* es un elemento en el mismo espacio métrico que la base de datos y *r* es el radio de búsqueda de la consulta, la consulta se compara contra todos los centros de *clusters*, siguiendo el orden en la lista de *clusters* hasta llegar al final o hasta encontrar que la bola de consulta está completamente contenida en uno de los *clusters*. Cada *cluster* no descartado, esto es, los *clusters* con los que hay intersección, es un *cluster* candidato y se debe revisar. Si se llega al final de la lista sin que la bola de consulta haya estado completamente contenida en un *cluster*, ya se han calculado las distancias a todos los centros de *clusters* y por consiguiente éstas se usan para descartar algunos de los elementos gracias al filtrado por las distancias a los pivotes (centros de la lista de *clusters*) [2].

4 Evaluación del Rendimiento

A los efectos de evaluar el rendimiento de la propuesta del *CMSNMS*, se realiza una comparación del mismo contra *SSS-NMS*, *SSS* y *Random*. Los algoritmos, que se describen en [2], se implementaron en Java utilizando la JDK 1.6.0_29 y el entorno de desarrollo Eclipse. Se utilizó MySQL como sistema de gestión de base de datos, y la distancia euclidiana para medir la similitud de los objetos. La implementación de los índices y las políticas de búsquedas de *SSS* y *SSS-NMS* se hicieron de acuerdo a como se definen en [1] y [3] respectivamente. Para la política *Random* se seleccionan 3 pivotes al azar del conjunto de elementos que conforman la base de datos y se utiliza la desigualdad triangular para descartar los elementos durante las consultas. Se utilizaron espacios vectoriales sintéticos de dimensiones 2, 4, 8, 10, 12, y 14, creando colecciones de datos que concuerden con la definición de espacios métricos anidados.

En un paso previo, se evaluó el comportamiento de la adaptación de pivotes en los espacios métricos anidados. Para esto, en el segundo nivel del índice los pivotes se adaptaron a las búsquedas utilizando las estrategias pivote entrante y pivote saliente propuestas en [5]. Como resultado se observó que la adaptación de pivotes no mejora el rendimiento en los espacios anidados para ninguna de las dimensiones consideradas. Esto se debe a la irregularidad en la topología de los datos que forman estas colecciones. Por esto, en las experimentaciones detalladas a continuación, se descartó realizar la adaptación de pivotes.

Cada experiencia se organizó de la siguiente manera. Se generaron, aplicando la misma distribución, dos conjuntos de datos: *dataSetDatos* utilizado para crear el índice, y *dataSetQuery* usado para obtener los elementos de consultas. Para lograr aleatoriedad, en un primer nivel se tienen *N* corridas. En cada corrida se selecciona de manera aleatoria el 90% de los datos del *dataSetDatos* para crear el índice. Cada corrida está compuesta por *M* épocas. En cada época se utiliza el 90% de los datos de *dataSetQuery* obtenidos de manera aleatoria para realizar las consultas sobre el índice actual. Se obtiene la cantidad total de evaluaciones de la función distancia.

A modo de ejemplo, la conformación de los conjuntos de datos y el radio de consulta para el espacio de dimensión 8 utilizados en las dos primeras experimentaciones son los siguientes:

- *dataSetDatos* está compuesto por dos *clusters* de 2.000 datos cada uno. En el primer cluster los datos para x_0 se generan aleatoriamente entre [-20; -14]; y los datos para x_1, x_2, x_3, x_4, x_5, x_6, x_7 se generan aleatoriamente entre [-0,1; 0,1]. En el segundo cluster los datos para x_6 se generan aleatoriamente entre [14; 20]; y los datos para x_0, x_1, x_2, x_3, x_4, x_5, x_7 se generan aleatoriamente entre [-0,1; 0,1].
- *dataSetQuery* está compuesto por dos *clusters* de 200 datos cada uno. En el primer cluster los datos para x_0 se generan aleatoriamente entre [-18; -16]; y los datos para x_1, x_2, x_3, x_4, x_5, x_6, x_7 se generan aleatoriamente entre [-0,05; 0,05]. En el segundo cluster los datos para x_6 se generan aleatoriamente entre [16; 18]; y los datos para x_0, x_1, x_2, x_3, x_4, x_5, x_7 se generan aleatoriamente entre [-0,05; 0,05].
- El radio de consulta varía de acuerdo a la dimensión del espacio, ya que se mantiene constante el porcentaje de datos de la base de batos que retornará una consulta, en un 2%. Para ocho dimensiones el radio de consulta es igual a 0,193.

4.1 Implementación de *CMSNMS* en R^2

Esta experimentación tiene como objetivo mostrar detalles de la construcción del índice. Es decir, exponer cómo se reconocen los *clusters*, cómo quedan esparcidos los pivotes dentro de cada *cluster*, cómo se conforma la bolsa de elementos sin *cluster*, y el resultado de alguna búsqueda. Por lo cual este experimento se realiza en R^2. El radio del cluster C es $r_c = M * \alpha * \rho$, donde M es la máxima distancia entre cada par de elementos en la base de datos, α es una constante que regula la cantidad de centros de *clusters* (α=0,4 como se muestra en [2]), y ρ es una constante que dictamina la amplitud del radio de cada *cluster* (debe ser $\rho < 1$ para que no colisionen la estrategia de selección de centros con *SSS* y las propiedades de la *Lista de Clusters* [2]).

En la Figura 1 se muestran variantes de cómo se construye el primer nivel del índice para diferentes valores de ρ.

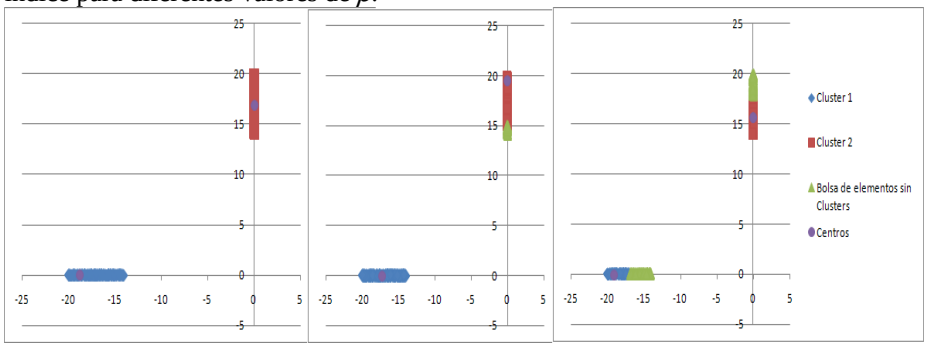

Fig. 1. Primer nivel de índice, con ρ = 0,9; 0,5 y 0,2; en este orden.

Notar como a medida que se disminuye el valor de ρ aumenta la cantidad de elementos sin cluster. Dado que la bolsa de elementos sin *cluster* queda vacía con ρ=0,9, en las siguientes experiencias se usa este valor para escoger el radio.

Para el segundo nivel del índice, ambos clusters se consideran densos. En la Figura 2 se pueden observar los pivotes que forman el segundo nivel del índice.

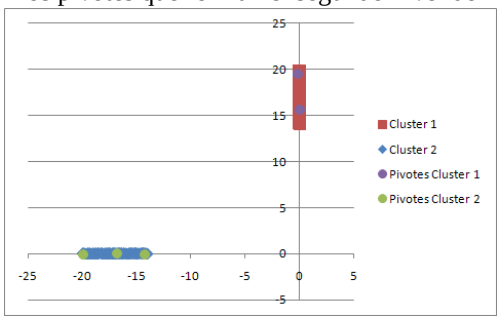

Fig. 2. Pivotes que forman el segundo nivel del índice.

El siguiente paso es realizar una consulta: $Q = (q, r_q)$, con q = (-16,78; -0,04) y r_q = 0,14. La consulta Q está contenida en el *cluster* número dos, por lo que se omite inspeccionar el otro. Como el *cluster* 2 es denso, se utilizan sus pivotes para obtener

la lista de candidatos (en rojo) que se muestra en la Figura 3.

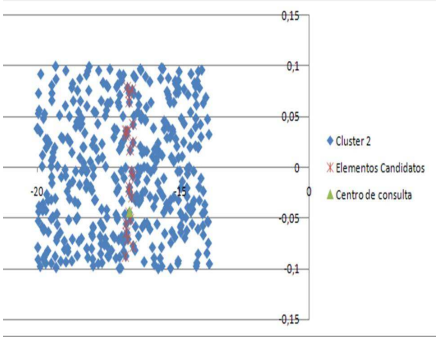

Fig. 3. Lista de candidatos de *Q*.

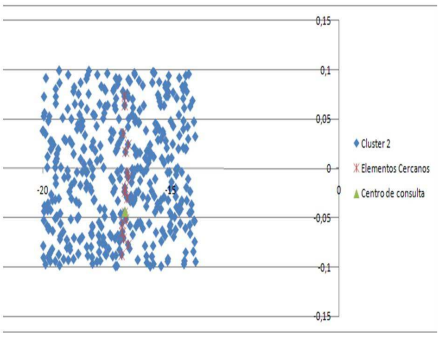

Fig. 4. Elementos de la respuesta a *Q*.

Finalmente la lista de candidatos obtenida es comparada exhaustivamente contra la consulta, para obtener los elementos cercanos cómo se puede observar en la Figura 4.

Notar la gran cantidad de elementos descartados por el índice y lo parecido de la respuesta a la lista de elementos candidatos. Esto expresa el buen trabajo del índice al descartar una gran cantidad de elementos en el filtrado por clases y así ahorrar un gran número evaluaciones de la función distancia.

4.2 Rendimiento y Análisis de Resultados

En la segunda experiencia, para cada dimensión se compara el comportamiento del nuevo índice *CMSNMS* contra los demás, *SSS-NMS*, *SSS*, y *Random*, teniendo en cuenta su orden espacial y temporal (para indexación y búsqueda).

El orden espacial es el mismo para todos los índices, ya que en el momento en que se tienen n elementos insertados en la base de datos (n=4000 en esta experiencia), es necesario mantener en memoria una matriz triangular, de $n^2/2$ elementos, que conserve la distancia entre cada par de objetos en el índice. Lo mismo sucede para el orden temporal durante la indexación. Si se tiene en cuenta el número de evaluaciones de la función distancia como factor preponderante, a medida que se insertan elementos en el índice es necesario compararlos contra todos los elementos ya insertados para obtener la matriz de distancias triangular de $n^2/2$ elementos. Por esto, el orden temporal de indexación es de $n^2/2$ para todos los algoritmos de construcción.

Para analizar el orden temporal de las búsquedas se cuenta la cantidad de evaluaciones distancias necesarias para realizar todas las consultas, obteniendo la *Complejidad Total* como la suma de la *Interna* y la *Externa*. La *Complejidad Interna* es la que se obtiene durante el filtrado por clases al conseguir la lista de candidatos, y la *Complejidad Externa* es la que se obtiene al analizar cada elemento de la lista de candidatos para determinar los elementos cercanos a la consulta.

En la Figura 5 se compara la cantidad de evaluaciones de la función distancia de los cuatro algoritmos a medida que se varía la dimensión del conjunto de datos. Los resultados obtenidos son mejores para *CMSNMS* desde la dimensión 8 en adelante. Esto se debe a la imposibilidad que se tiene para generar conjuntos de datos que representen un espacio métrico anidado cuando la dimensión es baja. Por otro lado los

resultados son similares a los de *SSS-NMS*. Esto se debe a que los conjuntos de datos contienen sólo dos *clusters*, por lo que *CMSNMS* a lo sumo puede omitir compararse contra un *cluster* por consulta. Si se aumenta el número de *clusters* la diferencia debería ser mayor. Esta última afirmación se corrobora en el siguiente experimento. También, como era de esperar, se observa que el *SSS* pierde y gana con respecto a *Random*, lo cual se debe a la topología irregular del conjunto de datos empleado.

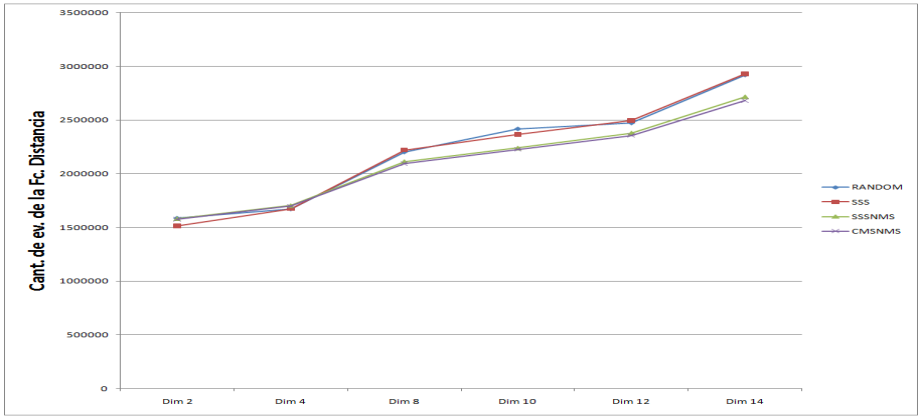

Fig. 5. Comparación de los cuatro algoritmos utilizando data sets con dos clusters.

A partir de estos resultados experimentales, para el caso más general de espacios métricos anidados conformados por conjuntos de datos de dos *clusters*, *CMSNMS* se comporta mejor que las demás técnicas. Por esto, es de interés ahora analizar qué pasa si se aumenta el número de *clusters* en los conjuntos de datos. En esta última experimentación se compara el rendimiento de *CMSNMS* contra el de *SSS-NMS* en espacios vectoriales de 8, 10, 12 y 14 dimensiones, para colecciones donde se identifican cuatro agrupamientos de datos.

Para esto se generan nuevos conjuntos de datos que representen las colecciones propuestas. A modo de ejemplo, la conformación de los conjuntos de datos y el radio de consulta, para el espacio de dimensión 12 son los siguientes:

- *dataSetDatos* está compuesto por dos clusters de 1.000 datos cada uno. En el primer cluster los datos para x_0 se generan aleatoriamente entre [-20;-14]; y los datos para x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan entre [-0,1; 0,1]. En el segundo cluster los datos para x_2 se generan aleatoriamente entre [-20;-14]; y los datos para x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan en forma aleatoria entre [-0,1; 0,1]. En el tercer cluster los datos para x_4 se generan aleatoriamente entre [14;20]; y los datos para x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan aleatoriamente entre [-0,1; 0,1]. En el cuarto cluster los datos para x_6 se generan aleatoriamente entre [14; 20]; y los datos x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9, x_{10}, x_{11} se generan aleatoriamente entre [-0,1; 0,1].
- *dataSetQuery* está compuesto por dos clusters de 200 datos cada uno. En el primer cluster los datos para x_0 se generan aleatoriamente entre [-18;-16]; y los datos para x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan en forma aleatoria entre [-0,05; 0,05]. En el segundo cluster los datos para x_2 se generan aleatoriamente entre [-18; -16]; y los datos para x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan

aleatoriamente entre [-0,05; 0,05]. En el tercer cluster los datos para x_4 se generan aleatoriamente entre [16; 18]; y los datos para x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} se generan aleatoriamente entre [-0,05; 0,05]. En el cuarto cluster los datos para x_6 se generan aleatoriamente entre [16; 18]; y los datos para x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9, x_{10}, x_{11} se generan aleatoriamente entre [-0,05; 0,05].

- El radio de consulta varía de acuerdo a la dimensión del espacio, ya que se mantiene constante, en el 2%, el porcentaje de datos de la base de datos que retornará una consulta. Para 12 dimensiones el radio de consulta es igual a 0,306.

En la Figura 6 se compara la cantidad de evaluaciones de la función distancia de los dos algoritmos a medida que varía la dimensión del conjunto de datos, se puede apreciar que existe una mayor diferencia entre *CMSNMS* y *SSS-NMS* que en la experimentación anterior. Esto se debe a que al que al aumentar el número de *clusters* *CMSNMS* se ve favorecido ya que al utilizar *Lista de Clusters* en el primer nivel puede omitir compararse contra 0, 1, 2 o 3 *clusters*, mientras que *SSS-NMS* debe hacerlo contra los 4 en cada comparación. Esta tendencia continúa si se sigue aumentando el número de *clusters*.

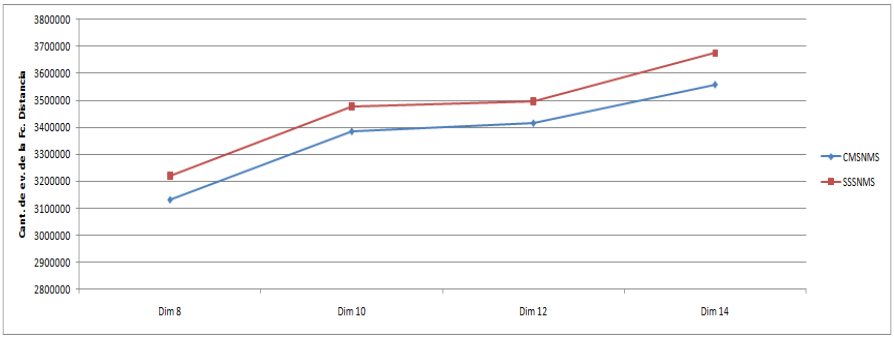

Fig. 6. *CMSNMS* vs *SSS-NMS*, con data sets con cuatro *clusters*.

5 Conclusiones

Los Espacios Métricos Anidados son espacios métricos donde un gran número de objetos se agrupan en subespacios densos, anidados en un espacio más general. Además, la diferencia entre cada par de objetos pertenecientes a un subespacio es explicada por una dimensión diferente a la dimensión que explica la diferencia entre otro par de objetos de algún otro subespacio. Estos espacios pueden ser encontrados en bases de datos reales, por ejemplo en una colección de imágenes representada por *Features Vectors*, si un grupo grande de imágenes comparten el mismo color principal sus vectores de características se agrupan en el espacio, muy cerca entre ellos; y la distancia entre dos de esas imágenes será muy pequeña. En este tipo de espacios, los índices convencionales no se comportan de manera adecuada en este tipo de espacios métricos. La estructura *CMSNMS* propuesta en [2], combina cuatro estructuras ya conocidas (*SSS*, *SSS con Adaptación Dinámica de Pivotes a las Búsquedas que se están Realizando*, *LC* y *SSS-NMS*) intentando explotar sus características para obtener un mejor desempeño en espacios métricos anidados.

Las experimentaciones que se presentan en este artículo, muestran que el

desempeño de esta estructura es mejor que el de las demás en los espacios métricos anidados. Se obtuvieron mejores resultados a partir de la dimensión 8, dada la dificultad para expresar espacios métricos anidados en espacios de baja dimensión, donde algoritmos como *SSS* o *Random* funcionan mejor independientemente de la topología del conjunto de datos. En la última experimentación presentada, al comparar *CMSNMS* contra *SSS-NMS* siempre se obtuvieron mejores resultados. Al aumentar el número de clusters en los conjuntos de datos la diferencia fue aún más acentuada, ya que *CMSNMS* emplea *Listas de Clusters* en el primer nivel, lo que permite omitir comparaciones contra *clusters* posteriores del primer nivel si la consulta estaba contenida en algún *cluster* anterior.

Algunas cuestiones por abordar son la evaluación de la propuesta en espacios de mayor dimensión de los aquí empleados, o con colecciones obtenidas de bases de datos reales, de objetos multimedia. Además, a partir de los resultados de la experimentación se puede proponer la implementación de algoritmos que trabajen con índices en memoria secundaria.

Referencias

1. Brisaboa N. R., Luaces M. R, Pedreira O., Places Á. S., Seco D.: Indexing Dense Nested Metric Spaces for Efficient Similarity Search. In: Proceedings of the 7th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI 2009) - LNCS 5947, Springer, Novosibirsk (Rusia), pp. 98-109, (2010).
2. Gercek H., Reyes N., Deco C., Bender C., Salvetti M. Combining Methods for Searches in Nested Metric Spaces. Anales del VIII Workshop Bases de Datos y Minería de Datos (WBDDM), XVII Congreso Argentino de Ciencias de la Computación CACIC 2011, ISBN 978-950-34-0756-1, La Plata, Argentina. pp. 949-958, (2011).
3. Pedreira O., Brisaboa N.R.: Spatial Selection of Sparse Pivots for similarity search in metric Spaces. In: 33nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07), LNCS vol: 4362, pp. 434-445. Springer (2007).
4. Chávez E., Navarro G.: A compact space decomposition for effective metric indexing. *Pattern Recognition Letters 26*, 9, 1363–1376. (2005).
5. Salvetti M., Deco C., Reyes N., Bender C.: Adaptive and Dynamic Pivot Selection for Similarity Search. Journal of Information and Data Management, Ed. Sociedade Brasileira de Computação. Vol. 2, No. 1, February 2011, pp. 27–35, (2011).
6. Chávez E., Navarro G., Baeza-Yates R., Marroquín J. L.: Searching in Metric Spaces. ACM Computing Surveys. 33(3), pp 273--321. (2001).
7. Bustos B., Navarro G., Chávez E.: Pivot selection techniques for proximity search in metric spaces. In: XXI Conference of the Chilean Computer Science Society, pp. 33-44. IEEE Computer Science Press. (2001).
8. Uribe, R.; Solar, R.; Brisaboa, N. R.; Pedreira, O.; Seco, D. SSSTree: búsqueda por similitud basada en clustering con centros espacialmente dispersos, en Actas de las Jornadas Chilenas de Computacion (JCC 2007), Iquique (Chile), pp. 1-13 (2007).
9. Mamede M.: Recursive Lists of Clusters: A Dynamic Data Structure for Range Queries in Metric Spaces. in: Proc. 20th Intl. Symp. on Computer and Information Sciences (ISCIS'05), in: LNCS, vol. 3733, pp. 843–853. (2005)