Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, República Argentina

Licenciatura en Ciencias de la Computación

Tesina de Grado

Combinando Métodos para Búsquedas en Espacios Métricos Anidados (*CMSNMS*)

Hugo Adrian Gercek

hugogercek@gmail.com

Noviembre 2011

Directora: Nora Reyes
Co Directora: Claudia Deco
nreyes@unsl.edu.ar, deco@fceia.unr.edu.ar

Dedicado a mi familia y a mi novia.
En especial a mis padres, Graciela y Hugo,
quienes tanto me alentaron para que cumpla con este objetivo.

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 los ***Espacios Métricos Anidados*** - ***Nested Metric Spaces***, que son algunos espacios métricos en donde los objetos de la colección pueden agruparse en clusters o subespacios. Aquí diferentes dimensiones o variables explican las diferencias entre los objetos dentro de cada subespacio anidado dentro de un espacio métrico más general.

En este trabajo se presenta una estructura de índice de dos niveles para procurar resolver problemas de búsquedas en espacios de esta topología, intentando aprovechar las virtudes de un conjunto de técnicas de indexación ya conocidas.

La idea es que un primer nivel posea una ***Lista de Clusters (LC)***, donde se tienen identificadas y ordenadas estas agrupaciones utilizando el ***Sparse Spatial Selection (SSS***) y técnicas de Listas de Clusters; y en un segundo nivel se tenga un índice por cada cluster denso, basado en selección de pivotes, empleando el ***SSS***. Además, se propone adaptar los índices del segundo nivel a las búsquedas que se están realizando, aplicando la "*Selección Dinámica de Pivotes que se Adaptan a las Búsquedas en Espacios Métricos"*, con el objetivo de adaptar los pivotes para mejorar las futuras búsquedas usando la información brindada por las búsquedas ya realizadas.

Contenido

1. Introducción

Con la evolución de las tecnologías de la información y las comunicaciones, han surgido almacenamientos no estructurados de información. No sólo se consultan sobre nuevos tipos de datos tales como texto libre, imágenes, audio y video, sino que además, en algunos casos, ya no se puede estructurar más la información en claves y registros. Estos escenarios requieren modelos más generales tales como las bases de datos métricas (es decir, bases de datos que incluyan objetos de un espacio métrico), y contar con herramientas que permitan realizar búsquedas eficientes sobre estos tipos de datos.

Las Bases de Datos Relacionales son las convencionales, donde se aplica el concepto de búsqueda por igualdad, ya que cada tabla está organizada en registros, y una consulta retorna aquellos registros donde su clave coincide con la clave de la consulta. Por ello se habla de búsqueda por igualdad o búsqueda exacta.

En los nuevos repositorios resultan poco útiles las búsquedas por igualdad, ya que por ejemplo debemos preguntarnos cuándo o para qué quisiéramos saber si dos segmentos de audio son exactamente iguales, seguramente estaríamos más interesados en saber si es la misma voz, o si está en el mismo ambiente, etc. Igualmente, no es de utilidad saber si dos imágenes son exactamente iguales; probablemente intentaríamos reconocer un mismo rostro, o un mismo lugar, etc. Por todo esto surge un concepto unificador: las búsquedas por similitud, donde se buscan elementos de la base de datos que sean similares a un elemento de consulta dado. La similitud es modelizada usando una función de distancia (o métrica), provista por un experto del dominio, que satisface ciertas propiedades, y el conjunto de objetos es llamado espacio métrico. Se sabe que la función distancia es bastante costosa de calcular. Luego, en la Sección 4, se describen diferentes técnicas que han surgido en la actualidad para intentar reducir el número de evaluaciones de la función distancia. 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). Ambos grupos tienen como objetivo dividir el espacio en clases de equivalencia y luego el índice se utiliza para filtrar algunas clases en tiempo de consulta. A grandes rasgos, la primera técnica define la clase de los elementos que están a una distancia i de todos los representantes, y la segunda define la clase de los elementos que está a una distancia menor que r de algún representante. Luego, en las búsquedas, los elementos de las clases que no son filtradas son comparados exhaustivamente contra la consulta. La mayoría de las técnicas fueron desarrolladas asumiendo que la topología de la colección de objetos es razonablemente regular, pero experimentaciones hechas sobre espacios donde la colecciones de objetos puede agruparse en subespacios o clusters han demostrado que estas técnicas no son tan eficientes.

En [1] se presenta una estructura de dos niveles, el ***Sparse Spatial Selection for Nested Metric Spaces (SSSNMS)***, para los espacios métricos anidados, que separa los subespacios en clusters utilizando ***SSS***, e indexa cada subespacio denso con ***SSS***.

En esta tesina se presenta una nueva versión de esa estructura que también posee dos niveles de índices. Un primer nivel, donde se identifican los agrupamientos con ***SSS*** [1] y se ordenan en una ***Lista de Clusters*** [24], y un segundo nivel, en donde, en base a una medida de densidad, se indexan con pivotes los clusters que se consideren altamente poblados, utilizando también el ***SSS***.

Lo innovador, además de la forma de construcción del índice y del nuevo algoritmo de búsqueda, es que luego de una determinada cantidad de consultas se utiliza lo propuesto en "*Selección Dinámica de Pivotes que se Adaptan a las Búsquedas en Espacios Métricos"* [25] para adaptar los pivotes de cada subespacio en el segundo nivel, a las búsquedas que se están realizando.

Se tiene por objetivo, construir una estructura que identifique estos subespacios, que sea eficiente en las búsquedas, y que a la vez sea dinámica y adaptativa.

Las hipótesis de trabajo son las siguientes:

- Que al identificar los representantes de cada cluster con ***SSS***, se logre identificar los subespacios y a la vez se obtenga un buen cubrimiento del espacio.
- Que al utilizar ***Listas de Clusters*** y darle un orden a los subespacios se pueda descartar un mayor número de regiones en tiempo de consulta obteniendo una estructura dinámica.
- Que al utilizar ***Listas de Clusters*** en el primer nivel se obtenga una estructura lo más resistente posible a la dimensionalidad intrínseca del conjunto de datos.
- Que al utilizar ***SSS*** para indexar los subespacios densos del segundo nivel se consiga un buen cubrimiento de cada subespacio.
- Que al utilizar ***Selección Dinámica de Pivotes que se Adaptan a las Búsquedas en Espacios Métricos*** para adaptar los pivotes de cada subespacio se logre adecuar los pivotes a las búsquedas que se están realizando y obtener mejores resultados en sucesivas consultas.

2. Organización de la Tesina

Este documento está organizado en tres partes: conceptos básicos, la propuesta de esta tesina con la experimentación y sus conclusiones.

En la primera parte se comienza con una introducción y presentación de los conceptos básicos utilizados a lo largo de esta tesina, explicando los problemas, los objetivos y alcances. Además, se hace una revisión del estado del arte de la búsqueda en espacios métricos haciendo hincapié en los índices que son combinados en la propuesta.

En la segunda parte se presenta la propuesta de esta tesina, los pseudocódigos derivados con sus pruebas experimentales y el análisis de dichos resultados.

Por último, en la tercera parte, se presenta la conclusión del trabajo, donde se evalúan los resultados experimentales teniendo en cuenta los objetivos de la propuesta. Además se propone el trabajo a futuro.

3. Marco Teórico

En este capítulo se explican algunos conceptos básicos utilizados a lo largo de la tesina junto con teorías que respaldan la experimentación.

Se define una base de datos métrica como una colección de objetos (de cualquier tipo) con una función de similitud y una manera formal de calcularla como una métrica. Las consultas por similitud extienden la búsqueda por exactitud porque el resultado obtenido es un conjunto de elementos cercanos al objeto de búsqueda. El objeto de referencia que se está buscando puede no pertenecer a la base de datos. Este concepto puede ser formalizado usando el modelo de espacio métrico, que consta de un espacio (conjunto de datos) y una medida de semejanza entre los elementos de ese espacio. No existe una definición general de esta medida dado que está estrechamente ligada a la aplicación, al tipo de objetos y a las características que se quieran evaluar.

3.1. Espacios Métricos

Sea X el universo de los objetos válidos, U un subconjunto finito de él ($|U| = n$), el conjunto de objetos donde se realizan las búsquedas. La función:

$$d: X \times X \rightarrow R^{+}$$

denota la medida de distancia entre los objetos. Recordemos que las funciones de distancia deben cumplir las siguientes propiedades usuales:

- Positividad: $\forall\, x, y \in X, d(x,y) \geq 0$.
- Simetría: $\forall\, x, y \in X, d(x,y) = d(y,x)$.
- Reflexividad: $\forall\, x \in X, d(x,x) = 0$.

Estas propiedades sólo aseguran una definición consistente de la función distancia, para que *d* sea una métrica debe satisfacer:

- Desigualdad triangular: $\forall\, x, y, z \in X, d(x,y) \leq d(x,z) + d(z,y)$.

Luego el par (X, d) se denomina ***espacio métrico***. [2]

La desigualdad triangular es una propiedad que es usada en los espacios métricos para ahorrar comparaciones en consultas por proximidad o similitud.

3.2. Consultas e Índices

3.2.1. Consultas

Interesan las búsquedas no exactas, es decir búsquedas por proximidad. Existen dos tipos de consultas típicas para este tipo de búsquedas: las *consultas por rango* y las *consultas por los k-Vecinos más cercanos*.

Consultas por rango: se desea recuperar los elementos de una base de datos que se encuentran a una distancia no mayor que un cierto radio de tolerancia de un elemento de consulta dado. Más precisamente una consulta por rango $(q, r)_d$ recupera todos los elementos que están a distancia menor o igual que r de q, donde d es la función de distancia definida. Esto es: $\{x \in U, d(q,x) \leq r\}$.

k-vecinos más cercanos: se desea recuperar los k elementos que se encuentran más cerca de una consulta dada. Más precisamente una consulta de *k-vecinos más cercanos* $k - NN(q)$ recupera los k elementos más cercanos a q en U.

Esto es recuperar un conjunto $A \subseteq U$ tal que $|A| = k$ y $\forall x \in A, y \in (X - A), d(q,x) \leq d(q,y)$.

En la Figura 1, se muestran un ejemplo de una consulta por rango y un ejemplo de una consulta por *k*-vecinos.

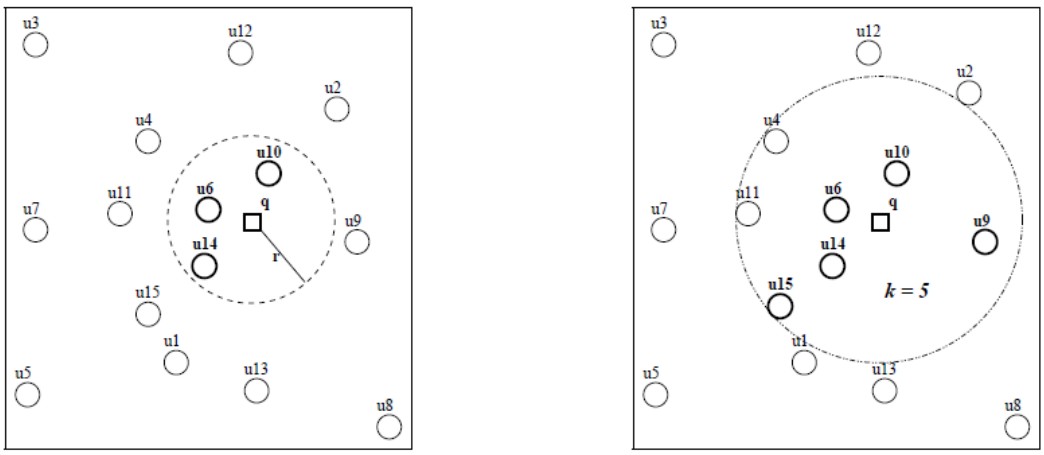

Figura 1: Un ejemplo de una consulta por rango (izquierda) y por k-vecinos más cercanos (derecha) sobre un conjunto de puntos en R^2. [1]

Una manera trivial de responder ambos tipos de consulta es realizando una búsqueda exhaustiva en la base de datos, es decir, comparando todos los elementos de la base de datos contra el elemento consultado y retornando aquellos elementos que se encuentren suficientemente cerca de éste, pero por lo general esto resulta demasiado costoso para aplicaciones reales.

3.2.2. Índices

Tratando de minimizar el costo de estas consultas se han desarrollado importantes avances para la búsqueda en espacios métricos generales alrededor de la idea de construir un índice.

Un índice es una estructura de datos que permite reducir el número de evaluaciones de distancia en tiempo de consulta. Todos los algoritmos de indexación para la búsqueda por proximidad consisten en construir clases de equivalencia, descartar algunas clases y buscar exhaustivamente en el resto. Todas estas estructuras trabajan sobre la base de descartar elementos usando la desigualdad triangular.

Generalmente, un algoritmo que resuelve consultas por proximidad tiene dos fases: la de preprocesamiento y la de consulta.

Inicialmente tenemos la base de datos, ésta se proyecta en un nuevo espacio que permite crear un índice (fase de pre-procesamiento). El índice se crea una vez y se almacena, por lo tanto, el costo de su construcción no se considera al momento de resolver consultas.

En la fase de consulta se recorre el índice (***comparaciones internas***) para conocer la lista de candidatos, cuyos elementos deberán ser comparados directamente contra la consulta (***comparaciones externas***), los elementos que no sean candidatos pueden ser descartados de manera segura. En la lista de candidatos también existen elementos que no son relevantes para la consulta, pero que no pudieron ser distinguidos por el índice.

El caso ideal es que todos los elementos en la lista de candidatos sean relevantes para la consulta.

[1] Figura extraída de: Nora Reyes. Índices Dinámicos para Espacios Métricos de Alta Dimensionalidad. Tesis para optar al Grado de Magister en Ciencias de la Computación. San Luis, Argentina.

Todas las estructuras de índice dividen los datos en subconjuntos, cada uno de ellos representa una clase de equivalencia, por ejemplo, un subconjunto de objetos relacionados de acuerdo a la relación de equivalencia elegida.

Realizando una búsqueda por semejanza, el algoritmo de búsqueda filtra algunas clases. Aquellas clases que no son eliminadas por el índice deben ser comparadas con el objeto consulta, ya que forman parte de la colección de objetos relevantes.

El costo asociado para recorrer el índice de forma transversal es la ***complejidad interna*** de la búsqueda, y el costo de comparar los elementos de las clases de equivalencia no filtradas contra el objeto consulta es la ***complejidad externa*** de la búsqueda.

En la Figura 2, se muestran un ejemplo de un índice construido sobre una Base de Datos y un ejemplo de una consulta q que usaría de este índice.

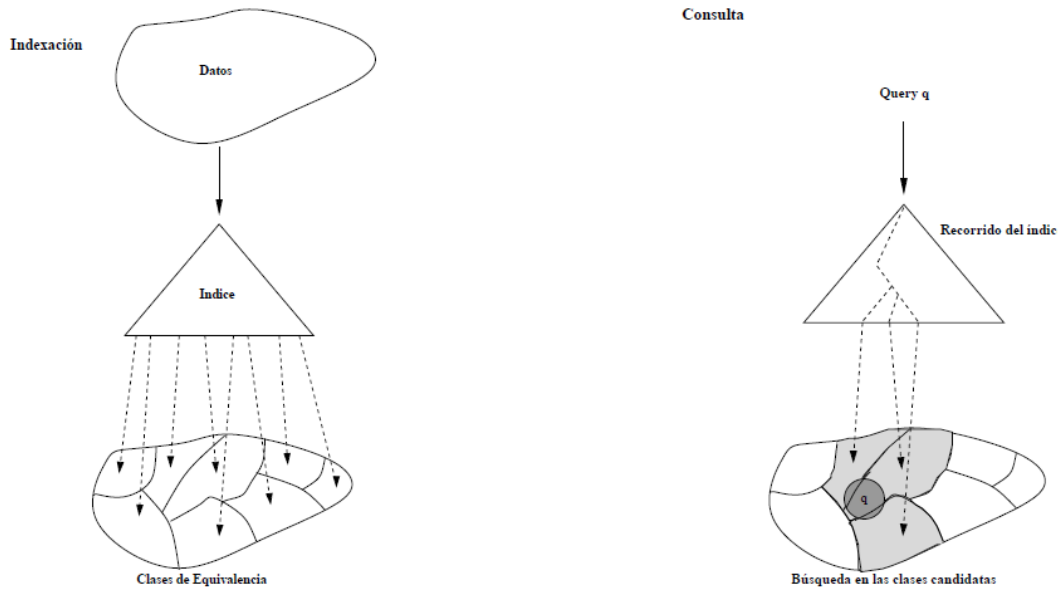

Figura 2: Un ejemplo de indexación y consulta.[2]

El costo de calcular las funciones distancia es muy elevado. El tiempo T necesario para calcular una consulta de similitud es:

$$T = número\ de\ evaluaciones\ de\ distancia * complejidad\ de\ d() + tiempo\ extra\ de\ CPU + tiempo\ de\ E/S$$

Donde el factor preponderante es la complejidad de evaluación de la función distancia d.

Los dos tipos principales de indexación son: ***indexación basada en pivotes*** e ***indexación basada en particiones compactas***.

[2] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

3.2.3. Índices Basados en Pivotes

La idea de los índices basados en pivotes es usar un conjunto de k elementos llamados "pivotes" $p_1 \dots p_k \in U$ y almacenar para cada elemento del conjunto de datos su distancia a los k pivotes $(d(x,p_1) \dots d(x,p_k))$. Dada una consulta q con rango de búsqueda r, sus distancias a los k pivotes son calculadas $(d(q,p_1) \dots d(q,p_k))$. Si para algún pivote p_i se tiene que $|d(q,p_i) - d(x,p_i)| > r$, por la desigualdad triangular se sabe que $d(q,x) > r$ y estos elementos pueden ser descartados sin ser confrontados contra la consulta. Todos los demás elementos que no puedan ser descartados usando esta regla deben ser directamente comparados contra la consulta.

Generalmente se obtiene una ganancia de tiempo al ahorrar un significativo número de evaluaciones de la función distancia d. Sin embargo, existe un costo de tiempo de preprocesamiento y un costo de almacenamiento de las distancias preprocesadas. Así se obtiene una tabla de distancias de $k * n$ elementos, donde es muy fácil agregar o quitar un elemento del conjunto de datos, simplemente se agrega (con k evaluaciones de la función distancia) o se quita (con ninguna evaluación de la función distancia) una columna de esta tabla. Reemplazar un pivote es un poco más complicado, tiene un costo de n evaluaciones de la función distancia y hay que evitar perder la calidad de los índices.

En la Figura 3 se muestra un ejemplo del uso de esta técnica, donde p es un pivote. El anillo formado por los círculos centrados en p, a distancias $d(p,q) + r$ y $d(p,q) - r$, contiene a los elementos que no podrían ser descartados por p, estos son: $u_2, u_3, u_4, u_5, u_6, u_7$. El resto de la base de datos sí es descartada, por ejemplo, en el caso de u_1, se cumple que $|d(p,q) - d(p,u_1)| > r$, lo mismo sucede con el resto.

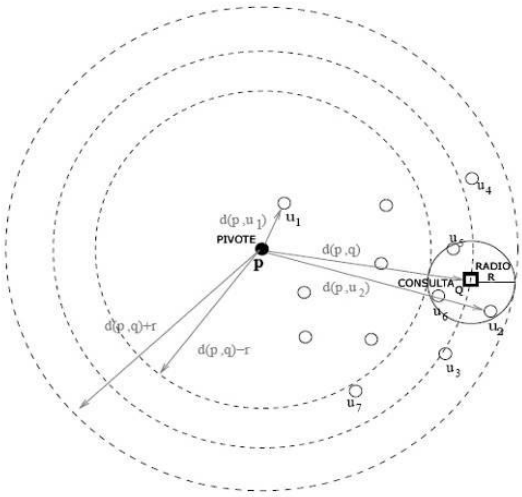

Figura 3: Algoritmo basado en pivotes para una consulta de rango (q, r).[3]

3.2.4. Índices Basados en Particiones Compactas

La segunda técnica consiste en dividir el espacio en zonas tan compactas como sea posible, usualmente en forma recursiva, almacenando un punto ("centro") representativo para cada zona junto con información que nos permita descartar esta zona de forma rápida. Existen dos criterios para delimitar la zona.

El primero es ***Voronoi region***, donde se elige un conjunto de centros y cada zona se delimita por hiperplanos, donde los "puntos" se reparten en función de cuál es el centro más cercano.

[3] Figura extraída de: M. Salvetti. Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos. Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Rosario, República Argentina.

Las regiones están acotadas por hiperplanos y las zonas son análogas a las regiones de Voronoi en los espacios vectoriales. Sea $\{c_1 \dots c_m\}$ el conjunto de centros, al momento de la consulta evaluamos $(d(q,c_1) \dots d(q,c_m))$, elegimos el centro c más cercano a q y descartamos cada zona cuyo centro c_i satisface $d(q,c_i) > d(q,c) + 2r$, es decir cuando su región de Voronoi no pueda intersecar la bola de consulta.

El segundo criterio es ***covering radius,*** donde notamos como $cr(c_i)$ a la máxima distancia entre el centro del cluster c_i y un elemento de su zona. Si $d(q,c_i) - r > cr(c_i)$ entonces no hay necesidad de considerar la zona i [3].

En la Figura 4 se muestra un ejemplo del uso de esta técnica. El radio de cobertura del centro p está representado por r_c . En la figura toda la zona del centro p puede ser descartada puesto que todos los elementos en ella cumplen con $d(q,p) - r_c \leq d(q,u)$ y $d(q,p) - r_c \geq r$.

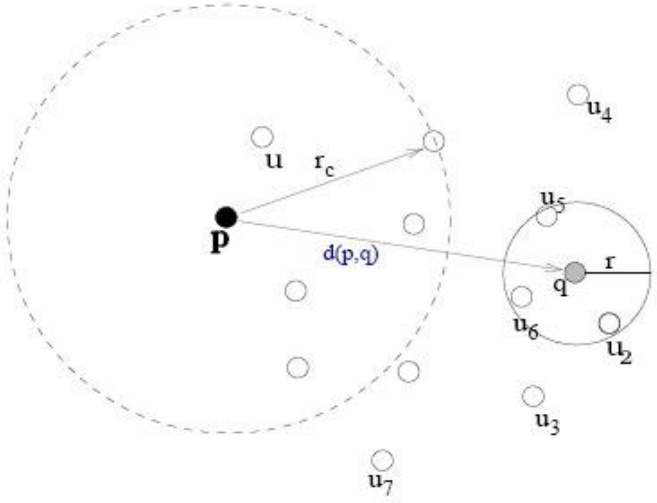

Figura 4: Algoritmo de clustering (particiones compactas), consulta de rango (q, r).[4]

Existen estructuras de datos que combinan ambas ideas dividiendo el espacio en zonas compactas y al mismo tiempo almacenan las distancias a algunos pivotes.

3.3. Espacios Vectoriales

En algunas aplicaciones los espacios métricos resultan ser de un tipo particular llamado ***espacio vectorial***, donde los elementos consisten de D coordenadas de valores reales. Existen muchos trabajos que aprovechan las propiedades geométricas de los espacios vectoriales, pero normalmente éstos no se pueden extender a los espacios métricos generales donde vimos que la única información disponible es la distancia entre objetos que en general es bastante costosa de calcular.

Si los elementos del espacio métrico (X,d) son tuplas de números reales, entonces el par se denomina ***espacio vectorial***. Un espacio vectorial de dimensión finita D, es un espacio métrico particular donde los objetos se identifican por D números reales $(x_1, x_2, \dots, x_D)$ y cada x_i, con $1 \leq i \leq D$, es llamada "coordenada" del objeto (se les llama *espacios vectoriales* o *espacios D-dimensionales*). Existen distintas funciones de distancia que se pueden usar en un espacio vectorial, pero las más usadas son las de la familia L_S o familia de ***Distancias de Minkowski***, que se define como:

[4] Figura extraída de: M. Salvetti. Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos. Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Rosario, República Argentina.

$$L_s\left((x_1,\ldots,x_D),(y_1,\ldots,y_D)\right) = \left(\sum_{i=1}^{D}|x_i - y_i|^s\right)^{1/s}$$

Dentro de la familia de distancias se encuentra la distancia L_1 conocida como ***Distancia de Manhattan***, la L_2 que es más conocida como ***Distancia Euclidea***, y se corresponde a nuestra noción habitual de distancia, y la distancia L_∞ conocida también como ***Distancia de Chebychev***.

La distancia L_1 se define como:

$$L_1\left((x_1,\ldots,x_D),(y_1,\ldots,y_D)\right) = \sum_{i=1}^{D}|x_i - y_i|$$

La distancia L_2 se define como:

$$L_2\left((x_1,\ldots,x_D),(y_1,\ldots,y_D)\right) = \sqrt{\sum_{i=1}^{D}|x_i - y_i|^2}$$

La distancia L_∞ se define como:

$$L_\infty\left((x_1,\ldots,x_D),(y_1,\ldots,y_D)\right) = \max_{1\le i\le D}|x_i - y_i|$$

En los espacios vectoriales se puede utilizar la información de coordenadas y geometría para facilitar los algoritmos de búsqueda. Esta información no está disponible en los espacios métricos generales.

Las estructuras de búsqueda para espacios vectoriales más populares son *Kd-trees*, los *R-trees*, los *Quad-trees* y los *X-trees* que son más recientes. Todas estas técnicas usan ampliamente la información de coordenadas para agrupar y clasificar puntos en el espacio. Desafortunadamente estas técnicas son muy sensibles a la dimensión del espacio.

Los espacios vectoriales pueden tener grandes diferencias entre su ***dimensión representacional*** (D) y su ***dimensión intrínseca*** (es decir, el número real de dimensiones en las cuales se puede embeber los puntos manteniendo la distancia entre ellos).

Para modelar datos multimedia como un espacio vectorial, se debe utilizar una función de transformación, que es sumamente dependiente del tipo de datos de multimedia. Esta función extrae rasgos importantes de los objetos multimedia y traza un mapa de estos rasgos en el espacio dimensional de vectores de *D-dimensiones* (también denominados en la literatura como descriptores). Entonces, la consulta original por similitud se reduce a buscar los puntos cercanos a ese vector de D-dimensiones.

Aquí se refiere a "*vector de propiedades*" - ***Feature Vector***, notado *FV*, como cualquier método de transformación de propiedades bien definidas.

Por lo general, la dimensionalidad del *FV* es un parámetro de la función de transformación: usando los valores más altos de D uno puede obtener una mejor representación (más fina) del objeto multimedia. Sin embargo, en usos prácticos hay, por lo general, un punto de saturación, donde si agregamos más dimensiones sólo añadimos ruido a la descripción. También se destaca que, para la mayoría de las aplicaciones, la transformación es irreversible, es decir, no se puede reconstruir el objeto multimedia original a partir de esta descripción con el *FV*.

3.4. Maldición de la Dimensionalidad

Las técnicas más tradicionales de indexación (como los *Kd-trees*) poseen una dependencia exponencial sobre la dimensión del espacio; así los espacios de alta dimensión son un problema para la eficiencia de las búsquedas en espacios métricos, ya que existen y se encuentran en muchas aplicaciones reales.

Para espacios de dimensión finita baja (no mayor que 20) existen técnicas que funcionan correctamente, llamaremos a estos espacios de dimensión finita D como espacios vectoriales D-dimensionales.

Los espacios *D-dimensionales* más altos tienen una distribución de probabilidad de distancias entre elementos cuyo histograma es más concentrado y con una media grande. Esto hace que cualquier algoritmo de búsqueda por similitud sea dificultoso. En el caso extremo tenemos un espacio donde $d(x,x) = 0\, y\, \forall x \neq y, d(x,y) = 1$, donde se debe comparar exhaustivamente la consulta contra cada elemento en el conjunto.

En la Figura 5, se muestran un ejemplo de un histograma de distancias para un espacio métrico de dimensión baja y un ejemplo de un histograma de distancias para un espacio métrico de dimensión alta.

Se extiende esta idea diciendo que un espacio métrico es más "difícil" (dimensión intrínseca más alta) que otro cuando su histograma de distancia es más concentrado.

Informalmente se dice que para espacios métricos difíciles (de dimensión intrínseca alta) los elementos se concentran más sobre la media, entonces aquí la distancia entre éstos es más cercana a cero, por lo que es más difícil descartar puntos en alguna consulta.

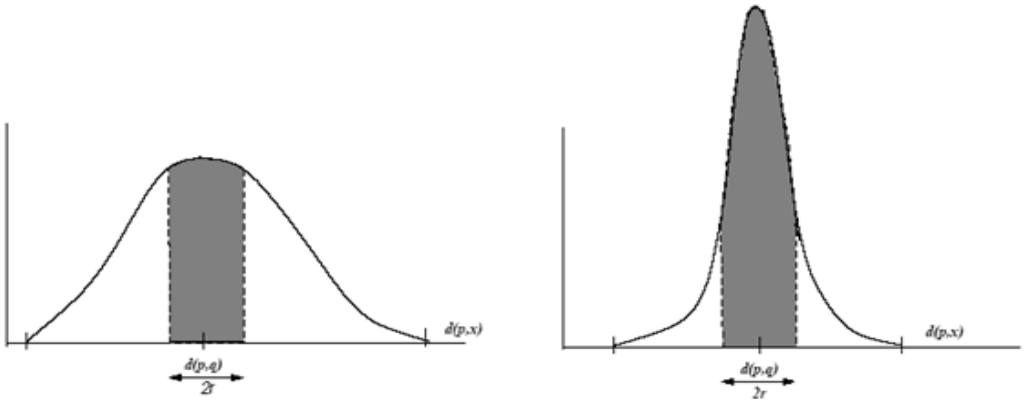

Figura 5: Un ejemplo de un histograma de distancias para un espacio métrico de dimensión baja (izquierda) y uno de dimensión alta (derecha).[5]

Formalmente la dimensionalidad intrínseca de un espacio métrico se define como:

$$\rho = \frac{\mu^2}{2\sigma^2}$$

Donde μ y σ son la media y la varianza del histograma de distancias, la dimensionalidad intrínseca crece con la media y se reduce con la varianza [27].

[5] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

Es importante remarcar que este fenómeno es independiente de la naturaleza del espacio métrico (en particular si es vectorial o no) y da una manera de determinar cuán difícil es buscar sobre un espacio métrico arbitrario.

3.5. Función Distancia

El problema central en las base de datos métricas es que generalmente la función distancia es muy costosa y compleja de calcular. Entonces, estas funciones son brindadas por expertos del dominio de aplicación y su funcionamiento interno es totalmente desconocido para el sistema de base de datos. Por esta razón una implementación razonable trataría de realizar la menor cantidad de evaluaciones de esta función, utilizando índices como veremos más adelante.

Ejemplos de funciones distancias en algunos dominios de aplicación:

> *Trivial*: $d(x, y) = 0\ si\ x = y,\ 1\ en\ otro\ caso$. Esta función lleva al caso de las bases de datos relacionales en donde los atributos son comparados por igualdad.
>
> *Imágenes*: para el caso de las imágenes se pueden definir varias funciones distancias, como la comparación de su vector *TVS* (*Transformated Vector of Symmetry*), comparación de histogramas de colores, descomposición espectral por la *transformada de Fourier*. Todas estas funciones implican mucho cálculo numérico.
>
> *Reconocimiento de voz:* aquí se pueden utilizar funciones basadas en el análisis espectral del espectro de voz, análisis de frecuencias promedio, etc. Cómo en el caso anterior todas necesitan mucho cálculo.
>
> Similitud textual: En este ámbito hay infinidades de funciones distancias. Entre ellas se encuentran:
>
> > *Mapeo a un vector de similitud:* en el cual el texto es representado por un vector donde cada componente representa la cantidad de veces que aparece una palabra en el texto.
> >
> > *Distancia de edición*: Esta distancia se define como la cantidad de inserciones, eliminaciones y modificaciones que tengo que hacer de una palabra para llegar a otra.

En todos los casos, se tiene que probar que la función distancia propuesta cumple con las condiciones necesarias para definir un espacio métrico. Si no cumplen algunas de las condiciones, la función puede ser modificada para conseguir un espacio métrico.

4. Trabajos Relacionados en Selección de Pivotes y Particiones Compactas

Existen dos grandes grupos de técnicas de indexación, índices basados en selección de pivotes e índices basados en particiones compactas (Clustering), en ambos grupos se construye un índice y se tienen como objetivo dividir el espacio en clases de equivalencia para minimizar el tiempo de consulta, para esto, en las búsquedas se utiliza el índice para ir filtrando algunas clases.

En esta sección se mencionan algunas estructuras de ambos grupos que se han propuesto en esta área, con el objetivo de dar un marco teórico-histórico para introducir la nueva estructura que se define en esta tesina.

Se hace hincapié en las estructuras que sirven como base para lo que aquí se propone, estas son, ***Sparse Spatial Selection*** (***SSS)***, ***SSS Mejorado con Adaptación Dinámica de Pivotes***, ***Listas de Clusters (LC)*** y ***Sparse Spatial Selection for Nested Metric Spaces (SSSNMS)***.

4.1. Índices Basados en Selección de Pivotes

La mayoría de los métodos de búsqueda basados en pivotes seleccionan los pivotes de forma aleatoria.

Además, no se conoce ninguna forma de determinar el número óptimo de pivotes, un parámetro que depende de la colección concreta con la que estamos trabajando.

Burkhard-Keller Tree (BKT)

Ésta fue probablemente la primera solución a la búsqueda en espacios métricos, fue presentada en [18]. Esta estructura es adecuada para funciones de distancia que entregan valores discretos.

La estructura se define recursivamente de la siguiente manera:

- Un elemento arbitrario $p \in U$ es escogido como raíz del árbol.
- Para cada distancia $i > 0$, se define $U_i = \{u \in U, d(u,p) = i\}$, como el conjunto de todos los elementos a distancia i de la raíz p.
- Luego para cada subconjunto no vacío U_i se construye un hijo de p que se etiqueta como i.
- Recursivamente se construye el BKT para U_i.
- Este proceso se puede repetir hasta que quede un solo elemento a procesar, o hasta que no queden más de b elementos (y se almacena un "bucket" (arreglo) de tamaño b).
- Todos los elementos seleccionados como raíz de los subárboles son llamados *pivotes*.

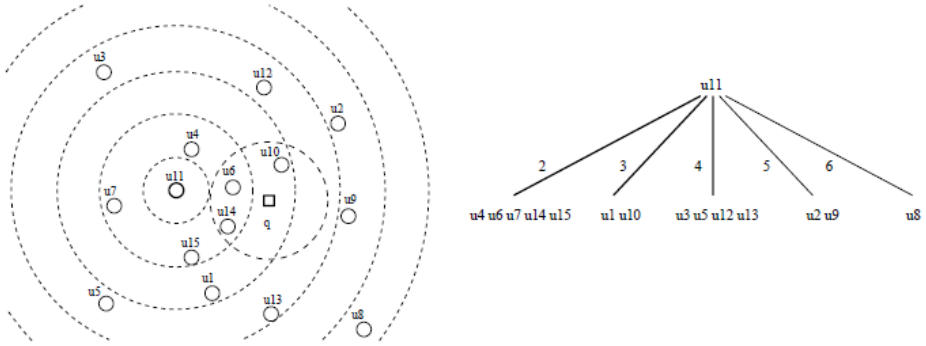

Figura 6: En la izquierda tenemos la división del espacio obtenido cuando tomamos u_{11} como pivote. A la derecha vemos el primer nivel de BKT con u_{11} como raíz. También vemos una consulta q y las ramas que debe atravesar.[6]

[6] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

Cómo se realiza una consulta:

- Ante una consulta q y una distancia r se comienza en la raíz del árbol y se recorren todos los hijos i tales que $d(p,q) - r \leq i \leq d(p,q) + r$, y se procede recursivamente.
- Si se llega a una hoja, se comparan secuencialmente todos sus elementos.
- Cada vez que se realiza una comparación ya sea contra una hoja o contra un pivote donde $d(q,u) < r$, se reporta el elemento u.

La desigualdad triangular asegura que no se pierde ningún valor de la respuesta. Todos los subárboles no recorridos contienen elementos u que están a distancia $d(p,u) = i$ de algún nodo p. Donde $|d(p,q) - i| > r$. Por la desigualdad triangular, $d(p,q) \leq d(p,u) + d(u,q)$, y luego la distancia $d(u,q) \geq d(p,q) - d(p,u) > r$.

En la Figura 6, se muestran un ejemplo de un BKT, donde se aprecia la división del espacio cuando se toma a u_{11} como raíz, una consulta q sobre este espacio y el primer nivel de árbol BKT obtenido a partir de la división de espacio antes mencionado.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.
- Tiempo de construcción: $O(n \log n)$.
- Tiempo demandado para una consulta: (n^{α}).

Fixed-Queries Tree (FQT)

Es un desarrollo posterior sobre los BKTs presentado en [4]. Básicamente es un BKT donde todos los pivotes almacenados en los nodos del mismo nivel son los mismos (y por supuesto no necesariamente deben pertenecer al mismo subárbol).

Ahora los elementos son solamente almacenados en las hojas.

La ventaja de esta construcción es que algunas comparaciones entre la consulta y los nodos son guardadas a lo largo del backtracking que ocurre en el árbol. Si se visitan muchos nodos del mismo nivel, no es necesario realizar más de una comparación porque todos los pivotes en este nivel son iguales. Esto es al costo de árboles un tanto más altos.

Se obtienen las siguientes complejidades:

- Espacial: $Entre\ O(n) y\ O(n \log n)$.
- Tiempo de construcción: $O(n \log n)$.
- Tiempo demandado para una consulta: (n^{α}), donde $0 < \alpha < 1$.

Fixed-Height FQT (FHQT)

Esta es una variante a los FQTs propuesta en [4; 20], donde todas las hojas se encuentran a una misma profundidad h, sin importar el tamaño del bucket.

Esto hace que algunas hojas estén más profundas de lo necesario.

Se obtienen las siguientes complejidades:

- Espacial: $Entre\ O(n)\ y\ O(n.h)$.
- Tiempo de construcción: $O(n.h)$.
- Tiempo demandado para una consulta: $O(\log n)$ (*).
- Tiempo extra de CPU: $O(n^{\alpha})$ (números de nodos recorridos).

(*) Si $h = \log n$.

Fixed Queries Array (FQA)

Esta estructura fue presentada en [21]. No es propiamente un árbol, y no es más que una representación compacta de un FHQT. Para dar una idea aproximada de su representación se puede pensar que un FHQT de altura fija h es construído sobre un conjunto de elementos.

Si se recorren todas las hojas del árbol de izquierda a derecha y se ponen los elementos en un arreglo se obtiene un FQA. Para cada elemento del arreglo se computan h números que representan las ramas que se tomarán del árbol para alcanzar el elemento desde la raíz (i.e. la distancia a los h pivotes).

Cada uno de esos h números es codificado en b bits y son concatenados en un número de alta precisión. Así se obtiene un *hb-bits* número que representa al FQA.

Se obtienen las siguientes complejidades:

- Espacial: Entre $O(n)$ y $O(n\ h\ b)$
- Tiempo de construcción: $O(n\ h)$.
- Tiempo demandado para una consulta: $O(log\ n)$ (*).
- Tiempo extra de CPU: $O(n^{\alpha}\ log\ n)$ (números de nodos recorridos).

(*) Si $h = log\ n$.

En la Figura 7, se muestran un ejemplo de un BKT, de un FQT, de un FQA y de un FQHT obtenidos aplicando las técnicas mencionadas sobre un mismo conjunto de datos.

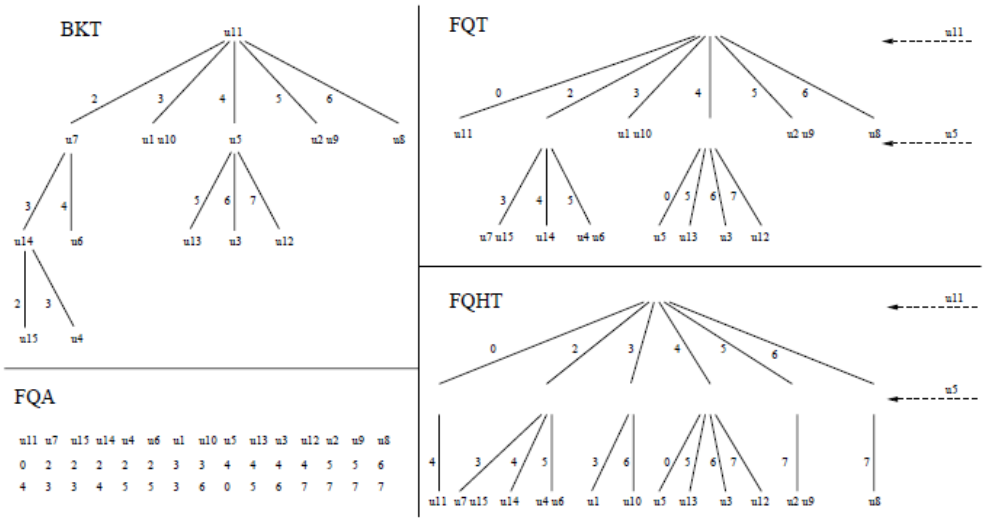

Figura 7: Un ejemplo de BKT, FQT, FHQT y FQA.[7]

VPT (Vantage-Point Trees)

Como antecedente, la primera estructura para funciones de distancia continuas fue presentada en [5], el *Metric Tree*. Luego el *Vantage-Point Trees* sigue la misma idea, éste fue presentado en [6; 7].

Esta última estructura (árbol balanceado) se construye de manera recursiva de la siguiente manera:

- Se elige un elemento *p* cualquiera como raíz y se toma la *media M* del conjunto de todas las distancias, donde $M = media\ \{\ d(p,u)\ /\ u\ \epsilon\ U\}$.

[7] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

- Luego los elementos $u \in U$ tales que $d(p,u) \leq M$ son insertados en el subárbol izquierdo. Mientras que los elementos que satisfacen $d(p,u) > M$ son insertados en el subárbol derecho.

Para resolver una consulta en este árbol:

- Se mide $d = d(q,p)$.
- Si $d - r \leq M$ se entra en el subárbol izquierdo, y si $d + r > M$ se ingresa en el subárbol derecho (Se puede entrar en ambos subárboles).

En la Figura 8, se muestra un ejemplo de un VPT y se traza el radio M = 3.1 utilizado para la raíz.

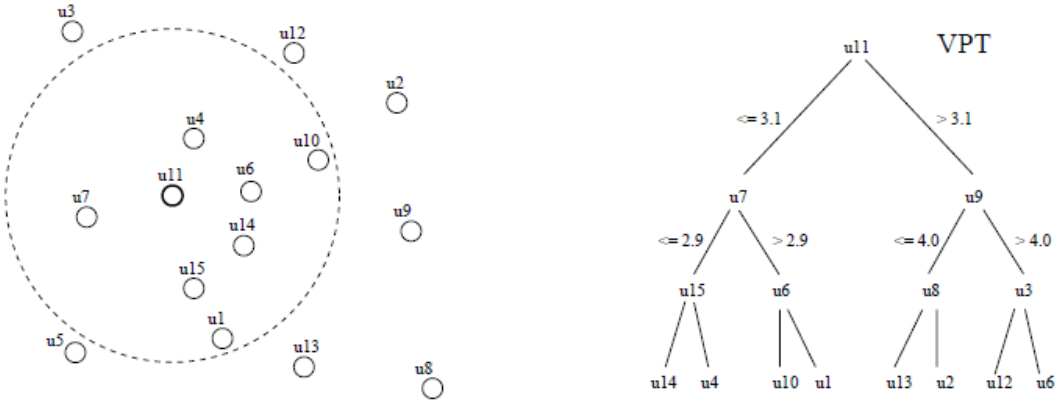

Figura 8: Un ejemplo del VPT con raíz u_{11}. Trazamos el radio M usado para la raíz.[8]

En árboles para distancia discreta, la distancia exacta entre un elemento en una hoja y cualquier pivote en el camino o la raíz puede ser inferida. Sin embargo, aquí sólo se conoce si la distancia es mayor o menor a M.

Salvo para el caso discreto es posible que se llegue a un elemento en una hoja que no necesitemos comparar, ya que nuestro árbol no tiene la suficiente información para descubrir esto.

En [6] se propuso almacenar estas distancias perdidas para podar más elementos antes de chequearlos. También se propuso que la mejor manera de elegir los pivotes, es tomando los más alejados del conjunto.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$. (Es el peor caso ya que está balanceado)
- Tiempo de construcción: $O(n \log n)$. (Es el peor caso ya que está balanceado)
- Tiempo demandado para una consulta: $O(\log n)$ (*).

(*) Sólo es válido con un pequeño radio de consulta, demasiado pequeño para ser considerado interesante.

MVPT (Multi-Vantage-Point Tree)

Esta estructura fue presentada en [8; 9]. Se propone que los VPT puedan ser extendidos a árboles m-arios usando $m - 1$ unidades de percentiles en vez de la media, donde los percentiles son una medida de posición que no refleja la tendencia central. Son los noventa y nueve valores de la distribución que la dividen en cien partes, de forma que dentro de cada una están incluidos el 1% de los valores.

Propusieron usar más de un elemento por nodo.

[8] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.
- Tiempo de construcción: $O(n\ log\ n)$.
- Tiempo demandado para una consulta: $O(log\ n)$.

VPF (short-hand for Excluded Middle Vantage Point Forest)

Esta es otra generalización del VPT. Fue presentada en [10]. El algoritmo fue diseñado para búsquedas $NN\ (q)$ con un radio máximo *r**, pero en realidad se puede adaptar perfectamente a las búsquedas por rango.

El método consiste en excluir en cada nivel a los elementos a distancias intermedias de su pivote (i.e. la parte "más popular" del conjunto).

Si r_0 y r_n representan al elemento más cercano y más lejano respectivamente de su pivote p, los elementos $u\ \epsilon\ U$ tales que $d(p,r_0)\ +\ \delta\ \leq\ d(p,u)\ \leq\ d(p,r_n)\ -\ \delta$ son excluidos del árbol.

Un segundo árbol es construido con la parte media excluida del árbol. Así continuamos hasta obtener una forestación.

Con esta idea se elimina el backtracking cuando se busca con un radio $r* \leq\ (r_n - r_0\ -\ 2\delta)\ /\ 2$, pero se debe buscar en todos los árboles de la forestación.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.
- Tiempo de construcción: $O(n^{2-p}\ )$.
- Tiempo demandado para una consulta: $O(n^{1-\rho}\ \ log\ n)$. (*)

Donde $0\ <\ \rho\ <\ 1$ depende de *r**.
(*) *r** debe ser demasiado pequeño.

AESA (Approximating Eliminating Search Algorithm)

El AESA se presentó en [16], éste es un algoritmo cercano a varias ideas ya presentadas pero sorprendentemente su performance es mejor en un orden de magnitud.

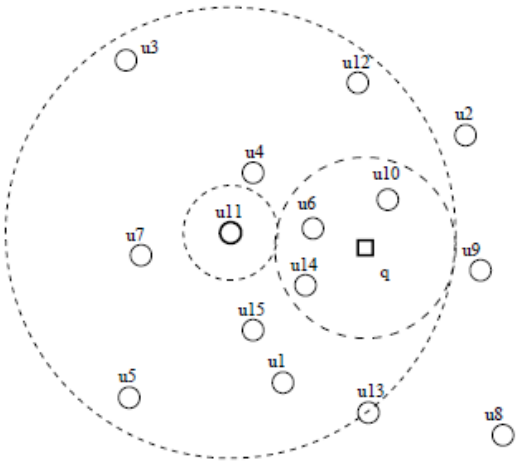

Figura 9: Un ejemplo de una primera iteración del AESA. Los puntos entre ambos anillos centrados en u_{11} califican para la siguiente iteración.[9]

[9] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

Su estructura es simplemente una matriz con $n(n-1)/2$ distancias precomputadas entre los elementos de U.

En una búsqueda se selecciona aleatoriamente un elemento $p \in U$ y se mide $r_p = d(p,q)$, eliminando todos los elementos $u \in U$ que no satisfagan $r_p - r \leq d(u,p) \leq r_p + r$. Como todas las distancias $d(u,p)$ ya están computadas sólo se debe calcular $d(p,q)$.

Este proceso de tomar un pivote aleatorio entre los elementos (no eliminados todavía) de U y eliminar más elementos de U se repite hasta que quedan "bastante pocos" elementos en el conjunto. Estos son comparados contra la consulta q directamente.

En la Figura 9 se muestra un ejemplo de la primera iteración de un AESA, donde los puntos entre ambos anillos centrados en u_{11} califican para la siguiente iteración.

Se obtienen las siguientes complejidades:

- Espacial: $O(n^2)$.
- Tiempo de construcción: $O(n^2)$.
- Tiempo demandado para una consulta: $O(1)$.

El problema de este algoritmo es que requiere un orden cuadrático de espacio y de tiempo de preprocesamiento, lo cual es inaceptable para grandes base de datos.

T-Spanners

En teoría de grafos se tiene el concepto de *t-spanner*, que consiste en un subgrafo G' de un grafo G tal que aproxima las distancias en G con un factor de precisión t.

Se considera un *t-spanner* G' (V, E) como la representación de la base de datos métrica. En G' el conjunto de vértices V corresponde a los objetos del espacio métrico y el conjunto de aristas E corresponde a una selección reducida de las distancias entre pares de objetos.

Se proponen varios algoritmos de construcción y de reconstrucción de *t-spanners*, donde se insertan y se borran objetos. Para mejorar las búsquedas por similitud la recuperación de objetos se realiza teniendo en cuenta la existencia de clusters [23].

SSS- Sparse Spatial Selection

En [1] se presenta un método basado en pivotes, donde se busca obtener una buena selección de pivotes sin especificar el número de pivotes a priori y lograr una visión de la complejidad del espacio métrico.

Se utiliza la definición de *espacio métrico* (X, d) usual, junto a las funciones de distancia y tipos de consulta conocidos.

El método presenta como características más importantes:

- Adaptación a la dimensionalidad del espacio.
- Dinamismo, ya que la colección puede crecer luego.
- Trabajar con distancias continuas y discretas.
- Posee buen desempeño en memoria secundaria.
- Puede ser fácilmente paralelizable.

La contribución principal es su estrategia para elegir pivotes. Ésta generará un número de pivotes que depende de la dimensionalidad intrínseca del espacio.

Sea (X, d) un espacio métrico, $U \subseteq X$, y M la máxima distancia entre dos pares de objetos $M = max\{d(x, y) / x, y \in U\}$.

Elección de los pivotes

Inicialmente el conjunto de pivotes contiene solamente el primer objeto de la colección.

Luego para cada elemento $u_i \in U$, u_i es elegido como un nuevo pivote si la distancia a cada pivote en el conjunto actual de pivotes es igual o mayor que $M * \propto$, donde $\propto$ es un parámetro constante que toma un valor alrededor de $0{,}4$.

El número de pivotes crece a medida que crece la dimensionalidad intrínseca del espacio, por lo que el ***SSS*** puede ser considerado como un método alternativo para medir la dimensionalidad del espacio. Por otro lado, también se puede afirmar que el número de pivotes no depende de la cantidad de elementos de la base de datos.

Consulta

Dada una consulta (q, r), se calcula la distancia de cada pivote a q y algunos elementos de la colección pueden ser directamente descartados usando la desigualdad triangular y las distancias precomputadas durante la construcción del índice.

Sea u un objeto de la colección podemos descartarlo si $|d(p_i, u) - d(p_i, q)| > r$ para algún pivote p_i, ya que por la desigualdad triangular si esta condición es verdadera, la distancia de u a q es mayor que r, $d(u, q) > r$.

Los objetos que no pueden ser descartados por esta condición formarán la Lista de Candidatos, y deberán ser comparados contra la consulta q.

SSS Mejorado con Selección Dinámica de Pivotes que se Adaptan a las Búsquedas en Espacios Métricos

Basándose en el método de indexación ***SSS***, en [25] se presenta una estructura de indexación y búsqueda por similitud que periódicamente intenta adaptar los pivotes del índice a las búsquedas que se están realizando. El objetivo de esta idea es poder mejorar la cantidad de discriminaciones hechas por los pivotes.

La construcción inicial del índice es realizada mediante el ***SSS***, y la "actualización" se realizará durante las búsquedas aplicando dos políticas de selección, una para eliminar del índice los pivotes que menos discriminen, es decir aquéllos que menos elementos descarten durante las consultas; y otra para la selección de elementos candidatos a pivotes, que son aquellos elementos que formen parte de la lista de candidatos una mayor cantidad de veces.

Mediante la aplicación de estas dos políticas se consigue lograr el objetivo de adaptar dinámicamente el índice a las búsquedas realizadas.

4.2. Índices Basados en Particiones Compactas

Estos algoritmos particionan el espacio métrico en varias regiones o clusters, cada uno de ellos representado por un centro de cluster. Así, el índice guarda información para que al momento de procesar una consulta puedan descartarse los clusters completos comparando la consulta con los

centros de cada cluster. En aquellos que no puedan descartarse, la consulta se compara con todos los objetos que pertenecen al cluster, y que forman la lista de candidatos.

BST (Bisector Trees)

El BST fue presentado en [11], es un árbol binario que se construye recursivamente como sigue:

- En cada nodo dos "centros" c_1 y c_2 son seleccionados.
- Los elementos más cercanos a c_1 que a c_2 van en el subárbol izquierdo y aquéllos que son más cercanos a c_2 van en el subárbol derecho.
- Para cada uno de los dos centros su radio de cobertura es almacenado (i.e. La máxima distancia del elemento a cualquier otro elemento en el subárbol).

En la consulta se inspecciona aquel subárbol que satisface que $d(q, c_i) - r$ no es más grande que el radio de cobertura de c_i. Por lo que se puede descartar una rama si la "bola de consulta" (la hiperesfera de radio r centrada en la consulta) no interseca a la bola que contiene a todos los elementos de la rama.

Más tarde, en [12] se propuso al *Monotonous BST*, donde uno de los dos elementos de cada nodo era el centro del padre. Esto hizo que el radio de cobertura decreciera en la parte inferior del árbol.

Se obtienen las siguientes complejidades:

- Espacial: *O(n).*
- Tiempo de construcción*: O(n log n).*

GHT (Generalized-Hyperplane Tree)

Esta estructura se propuso en [5] y su construcción es parecida a la del BST.

El algoritmo de búsqueda usa al hiperplano entre c_1 y c_2 como criterio de poda, en vez del radio de cobertura.

En la búsqueda se ingresa al subárbol izquierdo si $d(q, c_1) - r < d(q, c_2) + r$ o al subárbol derecho si $d(q, c_2) - r < d(q, c_1) + r$. Es posible entrar en ambos subárboles.

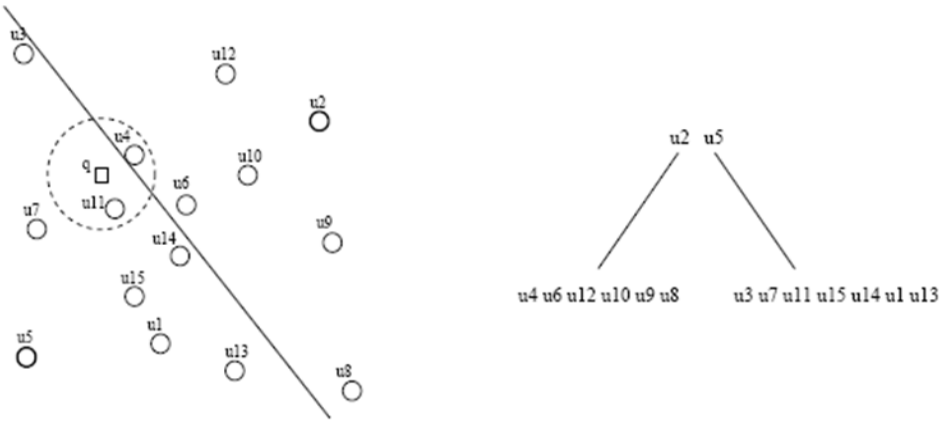

Figura 10: Un ejemplo del primer nivel de un BST o de un GHT y una consulta *q*.[10]

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.
- Tiempo de construcción: $O(n \log n)$.

[10] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

En la Figura 10, se muestra un ejemplo del primer nivel de un BST o de un GHT y una consulta q.

En [5] se sostiene que los GHTs pueden trabajar mejor que los VPTs en espacios de altas dimensiones.

GNAT (Geometric Nearneighbor Access Tree)

El GHT fue extendido a un árbol m-ario en [8] manteniendo la misma idea esencial:

- Para el primer nivel se seleccionan m centros $c_1, .., c_m$ y se define $U_i = \{u \in U, d(c_i, u) < d(c_j, u), \forall\, j \neq i\}$. En U_i están los elementos más cercanos a c_i que a otro c_j.
- Desde la raíz se construyen m hijos numerados de $i = 1..m$ recursivamente como un GNAT para cada U_i.

El algoritmo de búsqueda es un poco diferente:

- Cuando se indexa, el GNAT guarda en cada nodo una tabla de tamaño $O(m^2)$ $range_{ij} = [min_{u \epsilon Uj}(c_i, u), max_{u \epsilon Uj}(ci, u)]$, que almacena las distancias mínimas y máximas de cada centro a cada clase.
- En una búsqueda la consulta q es comparada contra algún centro c_i y luego se descarta cualquier otro centro c_j tal que $d(q, c_i) \,^{+}/_{-}\, r$ no interseca a $range_{ij}$.
- Todos los subárboles U_i pueden ser descartados usando la desigualdad triangular.
- El proceso se repite con centros aleatorios hasta que ninguno pueda ser descartado.
- Luego la búsqueda entra recursivamente en cada subárbol no descartado.
- En el proceso cualquier centro lo suficientemente cercano a q es reportado.

En la Figura 11, se muestra un ejemplo del primer nivel de un GNAT con $m = 4$.

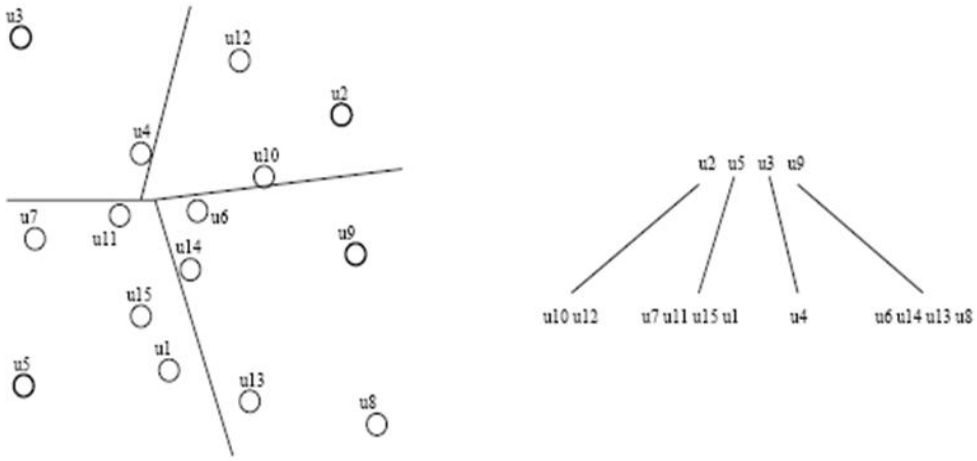

Figura 11: Un ejemplo del primer nivel de un GNAT con $m = 4$.[11]

Se obtienen las siguientes complejidades:

- Espacial: $O(n\, m^2)$.
- Tiempo de construcción: $O(n\, m\, log_m n)$.

VT (Voronoi Tree)

Se propuso en [13] como una mejora sobre los BSTs donde los árboles tienen 2 o 3 elementos (e hijos) por nodo.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.

[11] Figura extraída de: Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luís Marroquín. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.

- Tiempo de construcción: $O(n\ log\ n)$.

MT (M-tree)

Esta estructura de datos fue presentada en [14] apuntando a proveer capacidades dinámicas y una buena performance de I/O en adición de unas pocas computaciones de distancia.

La estructura tiene algunas semejanzas al *GNAT*, ya que es un árbol donde se eligen un conjunto de representantes en cada nodo y los elementos más cercanos a cada representante son organizados en un subárbol cuya raíz es este representante.

El algoritmo de búsqueda es más cercano al del *BST*. Cada representante almacena su radio de cobertura. En una consulta, ésta es comparada contra todos los representantes del nodo y el algoritmo de búsqueda entra recursivamente en todos aquellos subárboles que no puedan ser descartados usando el criterio del radio de cobertura.

La principal diferencia de los MT es la manera en que manejan las inserciones. Un elemento es insertado en "el mejor" subárbol, donde el mejor subárbol es aquel en el que su radio de cobertura se expanda menos, en caso de empate se escoge el de represente más cercano.

Si el elemento es agregado a una hoja, y se produce un overflow (obteniendo un tamaño $m + 1$), se divide el nodo en 2 y uno es promovido hacia arriba.

El MT es una estructura de datos más balanceada que la familia de los VP.

Hay muchos criterios para seleccionar los representantes y para dividir los nodos, los mejores resultados se obtienen tratando de dividir minimizando el radio de cobertura máximo obtenido.

Se mostró experimentalmente que los MT son resistentes a la dimensionalidad del espacio.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.
- Tiempo de construcción: $O(n\ (m..m^2) log_m n)$.

SAT (Spatial Approximation Tree)

El SAT fue desarrollado en [15]. Este algoritmo no usa centros para dividir el conjunto de objetos candidatos, confía más bien en una aproximación "espacial".

Un elemento *p* es seleccionado como raíz del árbol, y es conectado a un conjunto de "vecinos " *N*, definido como un subconjunto de elementos $u \in U$ tal que u es más cercano a *p* que a otro elemento en N. Los demás elementos (que no están en $N\ U\ \{P\}$) son asignados a su elemento más cercano en N. Cada elemento en N es recursivamente la raíz de un nuevo subárbol que contiene los elementos asignados a él.

Esto permite buscar elementos con un radio cero, simplemente moviéndose en el árbol desde la raíz hacia su vecino más cercano a la consulta q.

Si se permite un radio $r > 0$, entonces se busca un elemento desconocido $q' \in U$, del que sólo se sabe que $d(q, q') < r$. Se recorre como antes pero ahora se considera que la distancia puede tener un error de $^{+}/_{-}\ r$ como máximo. Por esto se debe entrar en varias ramas del árbol (no sólo la más cercana), ya que el "error" de medición podría hacer que un vecino diferente sea el más cercano. Si $c \in N$ es el vecino más cercano a q, se deben visitar todos los $c' \in N$ tales que $d(q, c') - r \leq d(q, c) + r$.

En la Figura 12 se da un ejemplo de un SAT y el recorrido hacia una consulta q a partir de u_{11}.

Se obtienen las siguientes complejidades:

- Espacial: $O(n)$.

- Tiempo de construcción: $O(n\ log\ n/\ log\ log\ n)$.
- Tiempo demandado para una consulta: $O(n^{1-O(1/log\ log\ n)})$.

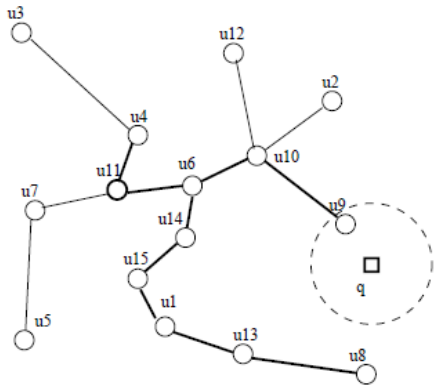

Figura 12: Un ejemplo de un SAT y el recorrido hacia una consulta q, a partir de U_{11}.[12]

List of Clusters

En general los índices pierden su eficacia a medida que la dimensión intrínseca de los datos crece. En [22; 24] se presenta un índice llamado ***List of Cluster (LC)***, que está basado en la partición compacta del conjunto de datos. Está demostrado que el ***LC*** requiere poco espacio para ser competente en memoria principal y secundaria, y lo más importante es que es muy resistente a la dimensionalidad intrínseca del conjunto de datos.

Además por cómo está construida la Lista de Clusters se le da un orden especial a sus miembros, donde los clusters en posiciones anteriores tienen una preferencia a la hora de contener elementos que se sitúan en regiones de intersección sobre los clusters posteriores.

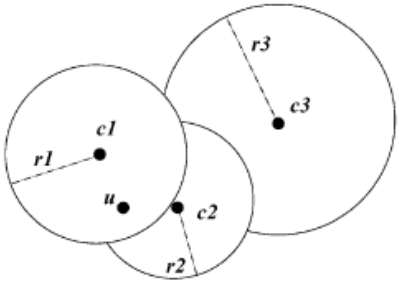

Figura 13: Las bolas de tres cluster de la lista $< C_1, C_2, C_3 >$.[13]

Como se muestra en la Figura 13, el primer centro elegido tiene preferencia sobre los posteriores en el caso de la superposición de las bolas. Todos los elementos que permanecen dentro de la zona del primer centro (c_1) son almacenados en el cluster de centro c_1 a pesar de que podrían estar dentro de los otros dos clusters, de centros c_2 y c_3.

Los centros y los radios de cada cluster se pueden elegir usando alguna de las siguientes políticas.

[12] Figura extraída de: Edgar Chávez. Searching in Metric Spaces. Journal ACM Computing Surveys (CSUR). Volume 33 Issue 3, September 2001. New York, NY, USA.
[13] Figura extraída de: Chávez E., Navarro G.: A compact space decomposition for effective metric indexing.

Selección del centro

Se pueden emplear 6 heurísticas para la selección del centro i:

(p1) Aleatoriamente.
(p2) El elemento más cercano a c_{i-1} en el conjunto restante.
(p3) El elemento más lejano a c_{i-1}en el conjunto restante.
(p4) El elemento que minimice la suma de las distancias a los centros ya elegidos.
(p5) El elemento que maximice la suma de las distancias a los centros ya elegidos.

Selección de radio

Existen dos alternativas simples:

(1) Partición de radios fijos: Es la alternativa más simple, consiste en seleccionar un radio fijo r^* para todas las bolas en la lista.
(2) Partición de tamaño fijo: Otra alternativa es tratar de tener un tamaño fijo de elementos m dentro de cada bola, y luego definir el radio acorde que contenga todos los elementos.

Búsquedas

Dada una consulta (q, r) la idea es aprovechar la particularidad de las Listas de Cluster, se recorre esta lista inspeccionando aquellos clusters en los cuales la bola de consulta se intersecte con él, pero se podrá detener la búsqueda cuando la bola de consulta esté completamente contenida dentro del cluster en cuestión.

4.3. Índices Combinados

En esta sección se presentan los índices y sus respectivos métodos de búsquedas donde se combinan ambas estrategias de selección de representantes.

LAESA (for Linear AESA)

Es una nueva versión del AESA, presentada en [17], donde propusieron usar k pivotes fijos.

Los elementos son simplemente linealmente recorridos, y aquéllos que no puedan ser eliminados después de ser comparados con los k pivotes son directamente enfrentados contra la consulta q.

Se obtienen las siguientes complejidades:

- Espacial: $O(k\ n)$.
- Tiempo de construcción: $O(k\ n)$.
- Tiempo demandado para una consulta: $k\ +\ O(1)$.

La técnica propuesta en [18] es la de dividir recursivamente el conjunto U en subconjuntos compactos U_i y elegir un representante c_i para cada subconjunto junto con su radio de cobertura. Para buscar el "vecino más cercano" la consulta q es comparada contra todos los representantes c_i. Los r_i se usan para determinar qué conjunto no posee elementos interesantes.

Los mejores resultados los obtuvieron cuando limitan el número de elementos más cercanos a m.

SSS-Tree

Este algoritmo es otra mejora al *SSS* en conjunto con las estructuras de datos árboles y las mejores propiedades de las técnicas de clustering.

Su principal característica es que los centros de cluster se seleccionan utilizando *Sparse Spatial Selection*, una técnica desarrollada originalmente para la selección de pivotes, capaz de adaptarse a la dimensionalidad intrínseca del espacio. Gracias a esto, el número de clusters en cada nodo depende de la complejidad del subespacio que tiene asociado.

Por otro lado, al seleccionar los centros de cluster de forma adaptativa, no tiene por qué tener en cada nivel el mismo número de divisiones cada cluster, si no las que se consideren necesarias en función del espacio métrico, una diferencia muy importante con todas las estructuras propuestas anteriormente.

4.3.1. Índices para Búsquedas en Espacios Métricos Anidados

El propósito de esta estructura es lograr que sea eficiente en los espacios de topología irregular. En esta sección además se introduce el concepto de *Espacios Métricos Anidados.*

SSSNMS - Sparse Spatial Selection for Nested Metric Spaces

En [1; 19] se presenta el ***Sparse Spatial Selection for Nested Metric Spaces (SSSNMS)***, como una nueva aproximación que trata de resolver problemas de indexación-búsqueda en un nuevo tipo de espacios llamados *Espacios Métricos Anidados*.

En este tipo de espacios los objetos de la colección pueden ser agrupados en diferentes clusters o subespacios, cada uno de éstos se encuentran anidados dentro de uno más general.

El objetivo de este método es identificar los subespacios y aplicar el SSS en cada uno de ellos.

El índice
El índice construido por ***SSS-NMS*** es estructurado en dos niveles:

- En un primer nivel se selecciona un conjunto de referencia con **SSS** y lo usa como centros de clusters para crear una ***Partición de Voronoi*** del espacio.
- En un segundo paso, aquellos clusters considerados densos son indexados con pivotes aplicando ***SSS*** en cada uno de ellos.

Búsqueda

Dada una consulta (q, r), la consulta es comparada contra todos los centros de cluster del primer nivel. Aquellos clusters $C_i = (c_i, r_c)$ para cuales $d(q, c_i) - r_c > r$ son directamente descartados del conjunto resultado, ya que la intersección de cada cluster con el conjunto resultado es vacía.

Para aquellos clusters que no son descartados existen dos posibilidades. Si el cluster no tiene asociado una tabla de distancia de sus objetos a los pivotes, la consulta es directamente comparada contra todos los objetos de éste. Si el cluster tiene asociado una tabla de distancias, la consulta es comparada contra los pivotes y la tabla es procesada para descartar tantos objetos como sea posible. Los objetos que no puedan ser descartados son directamente comparados contra la consulta.

5. Propuesta y Trabajo Realizado

5.1. Introducción

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.

Por ejemplo el ***SSS*** pertenece a la familia de índices que obtiene las mejores performances en espacios de características regulares, pero por otro lado, su desempeño no es el mejor para las colecciones de características irregulares.

Dentro de los espacios irregulares se encuentran los ***Espacios Métricos Anidados***. En estos espacios métricos los objetos de la colección pueden agruparse en clusters o subespacios densos que están anidados dentro de un espacio métrico más general. Pero no sólo los objetos están agrupados en estos subespacios densos, sino que 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.

En la Figura 14 se muestra una representación de este tipo de espacios. Aquí se aprecian dos clusters, que se anidan en un espacio métrico más general representado por el *eje x*, pero donde la dimensión que explica la diferencia entre cada par de objetos pertenecientes a un subespacio es diferente, el *eje y* para un cluster, y el *eje z* para el otro.

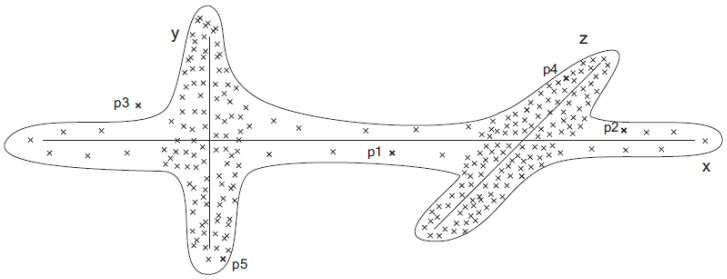

Figura 14: Subespacios densos agrupados dentro de un espacio métrico general.[14]

Los ***Espacios Métricos Anidados*** fueron descriptos por primera vez en [1]. En [19] se presenta el ***Sparse Spatial Selection for Nested Metric Spaces (SSS-NMS),*** un nuevo 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***. En [19] se muestran comparaciones donde se obtienen mejores resultados para ***SSS-NMS*** con respecto a los demás algoritmos sobre espacios de estas características.

En este trabajo se aborda el problema de la búsqueda en este tipo de espacios, se propone una nueva estructura de índice, que como todo índice tendrá por objetivo principal minimizar el tiempo de consulta.

Para esto se combinan estrategias empleadas en los espacios métricos convencionales y se trabaja bajo las siguientes hipótesis. Se utiliza el ***SSS*** para identificar los cluster anidados en el espacio métrico general, con él se obtienen los centros de los clusters, de manera tal de asegurarse un buen

[14] Figura extraída de: Pedreira O. An Effective Approach for Selecting Indexing Objects in Metric Spaces. Tesis Doctoral. Departamento de Computación. Universidad de la Coruña. 2009.

cubrimiento del espacio. Cada cluster se mantiene ordenado "jerárquicamente" en una ***LC***. Haciendo uso de este orden durante una búsqueda, si la bola de consulta se encuentra completamente contenida dentro de algún cluster, se puede omitir inspeccionar los siguientes. Por otro lado, esta estructura proporciona una alta resistencia a la dimensionalidad intrínseca de los datos. Se indexan con pivotes los subespacios que se consideren altamente poblados, en base a una medida de densidad que se da luego, procurando obtener un buen cubrimiento de cada subespacio. Los pivotes de cada subespacio denso se adaptan dinámicamente a las búsquedas que se están realizando dentro de su cluster, utilizando ***Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos***.

Por lo tanto la nueva estructura aquí propuesta tendrá 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.
- Un índice de pivotes construido con ***SSS*** para cada subespacio que se considere denso.

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 las 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.

5.2. Construcción del Índice

A partir de aquí cuando se haga alusión a un objeto "referencia" se puede estar hablando indistintamente de algún pivote para los índices basados en pivotes, o de algún centro de clusters para índices basados en particiones compactas.

La eficiencia de los métodos de búsqueda por similitud depende del conjunto elegido como referencia. El número de referencias, su ubicación en el espacio y su ubicación con respecto a las demás referencias determinan la capacidad actual del índice para descartar elementos sin compararlos contra la consulta.

Determinar el número de k referencias óptimas es un problema importante. La eficiencia de una búsqueda depende de este parámetro. Más aún, k puede variar enormemente para diferentes espacios métricos.

El índice tiene dos niveles, en el primer nivel se busca aprovechar las virtudes del ***SSS*** para identificar los subespacios. Pero no sólo se los identifica, sino que 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.

Sean (X, d) un espacio métrico, donde X es el universo de los objetos válidos, U es un subconjunto finito de él, $U \subseteq X$, con ($|U| = n$) y M es la máxima distancia entre dos pares de objetos $M = max\{d(x, y) / x, y \in U\}$ el índice se construye de la siguiente manera:

5.2.1. Primer Nivel: Lista de Clusters con SSS

En este primer nivel se identifican e indexan los subespacios anidados en el espacio métrico general. Se utiliza el ***SSS*** para obtener los centros de los clusters bien distribuidos y se mantiene cada cluster en una ***Lista de Cluster*** para conseguir y preservar un orden.

A continuación se dan algunas definiciones para ayudar a especificar el proceso de construcción de este nivel de la estructura.

Sea un centro $c \in U$ y un radio r_c. Se define la bola de centro (c, r_c) como el subconjunto de elementos de X que están a distancia máxima r_c de c; y donde $r_c < M * \propto$, luego se aclara el valor de $\propto$ y porque se impone esta restricción.

Entonces se define:

$$I_{U,c,r_c} = \{u \in U, 0 < d(c,u) \leq r_c\}$$

Como el "bucket de elementos internos", que permanecen dentro de la bola de centro c, y:

$$E_{U,c,r_c} = \{u \in U, d(c,u) > r_c\}$$

Como el resto, los "elementos externos".

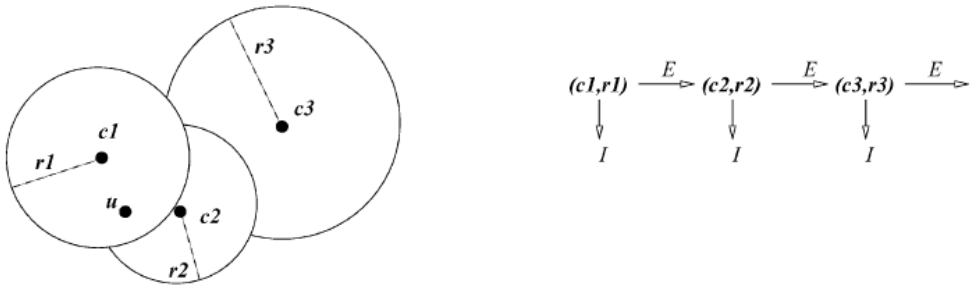

Figura 15: Representación de los cluster de la lista: $< (c_1, r_1, I_1), (c_2, r_2, I_2), (c_3, r_3, I_3) >$.[15]

La idea principal luego de elegir el primer centro es ir eligiendo los siguientes iterativamente con ***SSS*** sobre cada conjunto *E* y así obtener una lista de tripletas (c_i, r_i, I_i) (centro, radio, bucket), donde cada elemento representa a un cluster. Ver Figura 15.

La estructura de datos que se construye parece ser más bien simétrica, pero no lo es. El primer centro elegido tiene preferencia sobre los posteriores en el caso de la superposición de las bolas, como se muestra en la Figura 15. Todos los elementos que permanecen dentro de la bola del primer centro ($c_1$en la figura) son almacenados en el bucket I_1 a pesar de que podrían estar dentro de los bucket I de los subsecuentes centros (c_2 y c_3 en la figura). En Figura 15 se ve como la estructura de datos puede ser vista como una lista, donde los clusters en posiciones anteriores tienen una preferencia a la hora de contener elementos que se sitúan en regiones de intersección sobre los clusters posteriores. A esta propiedad se hace referencia cuando se dice que se tiene una Lista de Clusters con "orden jerárquico".

[15] Figura extraída de: Chávez E., Navarro G.: A compact space decomposition for effective metric indexing.

En la Figura 17 se esboza un pseudocódigo del algoritmo de construcción de este nivel del índice. Esta estructura es dinámica ya que permite comenzar con una colección vacía de elementos. Pero si se posee una colección inicial de datos, se puede aplicar un algoritmo de "bulk loading" para identificar los clusters y obtener un primer valor de M relacionado a estos datos. El pseudocódigo de este algoritmo se puede apreciar en la Figura 16. Este consiste en aplicar una variante del *SSS* sobre el conjunto inicial U, donde interesa conseguir un conjunto de representantes y la distancia entre ellos, $SSS(U,\ M)$. Para cada representante se promedia su distancia con el resto, luego se los ordena de mayor a menor de acuerdo a esta distancia, $SortByDistances(P)$, y finalmente se quitan de U sus apariciones, $Remove(U,\ p_i)$, agregando esta lista ordenada al comienzo, $U = P + + U$. Así se asegura que los primeros elementos inspeccionados por el algoritmo de construcción del índice estarán distantes, por lo que deberían pertenecer a distintos clusters y se obtendría una mejor representación de los subespacios anidados dentro del espacio métrico general.

$BulkLoading(U)$
$P = SSS(U, M)$ //P es el conjunto de representantes obtenidos al aplicar SSS, también se obtiene el M correspondiente a U
$SortByDistances(P)$ //Ordenar P descendentemente de acuerdo a la distancia promedio entre sus elementos
$For\ each\ p_i \in P\ do$
    $Remove(U,\ p_i)$ //Quitar los elementos de P en U
$U = P + + U$ //Agregar al comienzo de U los representantes ordenados

Figura 16: Pseudocódigo del algoritmo de bulk loading.

El algoritmo de construcción recibe como parámetros el conjunto de elementos U (preprocesados o no) a indexar, la Lista de Cluster L (vacía), y un conjunto B vacío de elementos que no pertenecen a ningún subespacio, pero que serán indexados con ***SSS***.

$Build\_Index(U, L, B)$
$R = \{\}$
$for\ each\ u_i \in U\ do$
    $if\ canBeCenter(u_i, L)$ //Si la distancia de u_i a cada centro actual es igual o mayor que $M * \propto$
        $setRadio(r_i, M, \propto)$ //Calcular el radio r_i, que depende de M y $\propto$
        $insertAtEndOfL((u_i, r_i, \{\}), L)$ //insertar la tripleta $(u_i, r_i, \{\})$al final de la Lista de Cluster L
        $if(\ updateM(M)\ )$ //Actualizar el valor de M si es necesario
            $reconsiderClusters()$ //Reconsiderar los clusters
    $else\ if\ isInSomeBallCj(u_i, L, (c_j, r_j, I_j))$ //Si el elemento u_i pertenece a alguna bola de centro c_j y radio r_j de la Lista de Cluster L, retornar //la primera tripleta(c_j, r_j, I_j) de L que cumple con esta condición
        $updateI((c_j, r_j, I_j), u_i)$ //Agregar el elemento u_i a I_J.
        $updateL((c_j, r_j, I_j), L)$ //Actualizar la bola de centro c_j de la Lista de Cluster L
    $else$
        $updateR(u_i, R)$ //Agregar u_i a la Lista R de elementos a reconsiderar
        $if\ (\ updateM(M)\ )$ //Actualizar el valor de M si es necesario
            $reconsiderClusters()$ //Reconsiderar los clusters
$reconsider(R, L, B)$ //Al final se reconsideran todos los elementos que no fueron indexados.

Figura 17: Pseudocódigo del algoritmo de construcción del primer nivel del índice.

Si no se preprocesa la entrada y se piensa el bucle for como sucesivas inserciones para cada $u_i \in U$, se estaría bajo la hipótesis de que se comienza con una base de datos vacía que va creciendo a medida que se insertan los elementos.

En la última línea del pseudocódigo, la lista R tiene dos tipos de datos, los elementos que debieran pertenecer a algún subespacio de la Lista de Cluster L pero que por el orden en que se eligieron los

centros no cayeron en ninguna bola de centro c_j y radio r_j, o los elementos que están fuera del alcance de cualquier radio de cobertura de alguna bola de la Lista de Cluster L. El método $reconsider(R, L, B)$ tiene en cuenta estas dos opciones, aquellos elementos que debieran pertenecer a algún subespacio de L que no pudieron ser observados en un primer momento son agregados a su respectivo cluster (el primero de la lista L si "cae" en más de uno), y aquellos elementos que no pertenecen a ningún subespacio de L son guardados en una bolsa de elementos B. Cada elemento de B es indexado de la manera usual con ***SSS***, utilizando cada centro c de L como un pivote.

La constante α y elección del radio

De forma experimental en [26] se muestra que el valor optimo para α debe estar en el rango $[0{,}35;\ 0{,}4]$ según sea la dimensionalidad de la colección. Se observa este resultado en la Figura 18 que muestra el número de evaluaciones de la función distancia en términos de $\propto$ para vectores espaciales de dimensionalidad 8, 10, 12 y 14, donde el costo de búsqueda es prácticamente el mismo para valores en este intervalo. Se puede observar que cuando $\propto > 0{,}4$ el número de evaluaciones de la función distancia toma mayores valores en espacios de dimensionalidad alta. Este resultado se debe al hecho de que al incrementar el número de $\propto$ se reduce el número de pivotes, y esta reducción tiene efectos más fuertes en espacios de dimensionalidad alta.

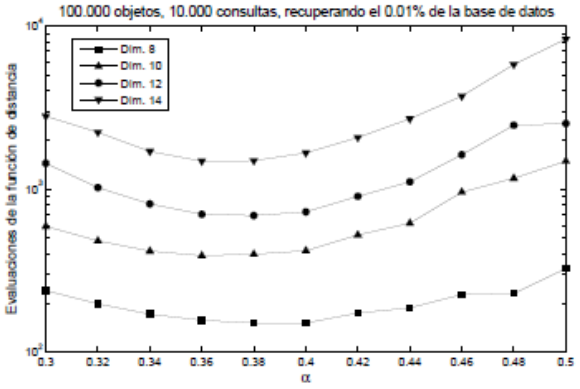

Figura 18: Número de evaluaciones de la función distancia para distintas dimensiones de vectores espaciales, en términos del valor de $\propto$.[16]

El radio r_c es el radio del cluster de centro c. Cada radio es estático, es decir, una vez que se elige no puede cambiar, porque si así fuera, al actualizar este valor seguramente se debería reconstruir el índice para no perder las propiedades de la Lista de Clusters. Se debe cumplir que $r_c < M * \propto$, para el valor actual de M, de manera tal que no colisionen las estrategias de selección de centros con *SSS* y las propiedades de la *LC*. Es decir, que un nuevo centro de cluster no esté contenido en algún cluster existente. Por lo tanto, el radio r_c deberá ser igual a $M * \alpha * \rho$, donde $\rho < 1$.

Por otro lado si se incrementa el valor de M puede ocurrir que algún centro de cluster deje de cumplir con la propiedad de que su distancia a todos los demás centros sea mayor que $M * \alpha$, en este caso, el método $reconsiderClusters()$ es el encargado de reconsiderar los elementos de estos tipos de clusters.

[16] Figura extraída de: 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).

5.2.2. Segundo Nivel: Elección de Pivotes en Subespacios Densos con SSS

Cuando se termina la construcción del primer nivel del índice, se tiene: por un lado, una Lista de Cluster L, de elementos (c, r, I) (centro, radio, bucket), y por el otro, una bolsa de elementos B no contenidos en ningún subespacio, indexados con SSS, utilizando los centros c de L como pivotes.

Aquellos clusters de L que se consideren densos en base a una medida que se da a continuación son indexados usando ***SSS***, obteniendo un conjunto de referencia formado por pivotes. Se calcula la densidad de cada cluster como el número de elementos del cluster dividido por la máxima distancia entre ellos.

$$density(C_i) = \left\{ \frac{|C_i|}{max\,\{d(x,y) | x,y \in C_i\}} \right\}$$

Calcular la densidad de cada cluster puede ser muy costoso si la distancia máxima entre cada par objetos es obtenida comparando todos los elementos del cluster. Para bajar este costo se obtiene una aproximación de la distancia máxima. Para hacer esto, un objeto del cluster es elegido al azar y es comparado contra todos los restantes. A continuación su objeto más lejano es comparado contra todos los demás, para obtener su objeto más lejano también. Luego de repetir este proceso por unas pocas iteraciones, una aproximación de la máxima distancia es obtenida (si no es la actual máxima distancia).

Se considera que el cluster C_i tiene densidad alta si $density(C_i) > \mu + 2\sigma$, donde μ y σ son la media y la desviación estándar de la densidad de todos los clusters. Para cada cluster denso, un conjunto de objetos es obtenido con ***SSS*** para ser usados como pivotes, y su tabla de distancias no debe ser calculada ya que fue obtenida previamente en la construcción del primer nivel.

En este segundo paso el índice almacena más información sobre los subespacios densos. Un elemento u es elegido como pivote del subespacio de centro c_i si la distancia de u a cada pivote del subespacio es mayor que $M_i * \beta$, siendo M_i la máxima distancia entre cada par de objetos en el cluster de centro c_i (antes aproximada) y β un valor constante alrededor de $0{,}4$ como se indica en [26].

En la Figura 19 vemos el algoritmo de construcción del segundo nivel, éste recibe como parámetro la lista de clusters L.

```
IndexSecondNivel(L)
for each C_i ∈ L do
    (c_j, r_j, I_j) = C_i        //Cada cluster se compone de la tripleta (centro, radio, bucket de elementos internos)
    If isDense(I_i)              //Si el cluster es denso
        IndexWithSSS(I_j)        //Indexar el cluster
```

Figura 19: Pseudocódigo del algoritmo de construcción del segundo nivel del índice.

5.3. Búsquedas

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 L hasta llegar al final o hasta encontrar que la bola de consulta está completamente contenida en uno de los clusters. Cada cluster que no haya sido descartado, esto es, los clusters con los que haya 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 de la bolsa B gracias al filtrado

por las distancias a los pivotes (centros de la lista de clusters). El algoritmo de búsqueda por rango se presenta en el pseudocódigo de la Figura 20.

Básicamente, la lista es iterada y la relación entre cada cluster y la consulta es establecida basada en la distancia de la consulta al centro del cluster y en los dos radios. La función recursiva $SearchL$ tiene cuatro parámetros: La lista de clusters L, la consulta (q,r), la bolsa de elementos B y la lista de elementos candidatos K (que debe estar vacía al comenzar).

$SearchL(L,(q,r),B,K)$
$if\ L\ is\ empty$
    $if\ B\ es\ empty$
        $retun\ K$
    $else$
        $return\ K\ \cup SearchB((q,r),B)$
$let\ L =< (c,r_c,I): L' >$
$distQC \leftarrow d(q,c)$
$if\ distQC\ \leq r$ //La bola de consulta contiene al centro *c*
    $if\ distQC + r\ \leq r_c$ //La bola de consulta está dentro del cluster
        $return\ PivoteSearch\big(I,(q,r)\big) \cup K \cup \{c\}$
    $else$ //La bola de consulta contiene al cluster o la bola de consulta intersecta el cluster
        $K' \leftarrow PivoteSearch\big(I,(q,r)\big) \cup K \cup \{c\}$
        $return\ SearchL(L',(q,r),B,K')$
    $else$ //La bola de consulta no contiene al centro *c*
        $if\ distQC + r \leq r_c$ //La bola de consulta está dentro del cluster
            $return\ PivoteSearch\big(I,(q,r)\big) \cup K$
        $else\ if\ distQC > r + r_c$ //La bola de consulta está fuera del cluster
            $return\ SearchL(L',(q,r),B,K)$
        $else$ //La bola de consulta intersecta el cluster
            $K' \leftarrow PivoteSearch\big(I,(q,r)\big) \cup K$
            $Return\ SearchL(L',(q,r),B,K')$

Figura 20: Algoritmo de búsqueda donde se ilustran los 6 posibles casos.

La función $PivotSearch$ obtiene la lista de candidatos de cada cluster, utilizando los pivotes propios si el cluster es denso y está indexado, o retorna todos los elementos del cluster si éste no es denso. Sus parámetros son: el bucket de elementos del cluster I, que en nuestro caso lo podemos pensar como una referencia al índice y la consulta (q,r).

Dada la asimetría del conjunto de datos, se puede podar la búsqueda si la bola de consulta está totalmente contenida en la bola de centro c, por lo que se puede no considerar el resto de la lista. Si se llega al final de la lista sin que la bola de consulta haya estado completamente contenida en un cluster y la bolsa de elementos B no está vacía, el método $SearchB$ es el encargado de descartar algunos de los elementos de la bolsa B gracias al filtrado por las distancias a los pivotes (centros de la lista de clusters).

Esta es una característica esencial ausente en otros algoritmos de particionamiento compacto, donde la búsqueda necesita entrar en todas las particiones que son interceptadas por la bola de consulta. En esta estructura la consideración de particiones relevantes puede detenerse cuando la bola de consulta esté totalmente contenida en una partición.

En la función $PivotSearch$ se aplica la desigualdad triangular de la siguiente manera: sea e un elemento del índice I puede ser descartado si $|d(p_i,e) - d(p_i,q)| > r$ para algún pivote p_i del

subespacio, ya que por la desigualdad triangular si esta condición es verdadera, la distancia de e a q es mayor que r, $d(e,q) > r$.

Finalmente, una vez que se obtiene la lista de candidatos, la consulta se compara exhaustivamente contra ésta, donde la distancia a los centros no debe ser recalculada ya que la obtuvimos previamente.

5.4. Actualización del Índice

Luego de "S" búsquedas (en un cluster), donde S es un parámetro que se evalúa de forma experimental, se obtienen los porcentajes de discriminación por parte de los pivotes actuales en el índice. Luego si se detecta algún pivote muy poco relevante, es decir un pivote que discrimine poco, en relación a los demás pivotes del índice sujeto a las S búsquedas hechas en el cluster este pivote es candidato a ser reemplazado dentro del índice del subespacio.

Al mismo tiempo, se tiene que seleccionar un nuevo pivote que entre al índice, para esto se debe llevar una estadística de las apariciones en los resultados de búsqueda de los elementos. Es decir, se lleva una estadística sobre los elementos $x \in DB$ (más precisamente pertenecientes al Cluster en cuestión) que quedan más veces en el resultado de las consultas. Luego, el elemento que más veces fue candidato va a ser el que entre como pivote al índice del subespacio, en reemplazo de la víctima seleccionada con la política que recién se explicó, según los porcentajes de discriminación.

Para saber cuándo un pivote es malo o cuando un elemento se vuelve pivote se utilizan las técnicas propuestas en [25], adaptándolas al índice que aquí se propone.

¿Cuándo es "malo" un pivote dentro de un subespacio?

Una vez constituido el índice con el número de pivotes estabilizado y acorde a la dimensionalidad intrínseca del subespacio, se realiza una consulta q que "cae" dentro del cluster, y es aquí donde se calcula en tiempo de consulta $d(q,p_i), i = 1..k_c$, es decir, la distancia entre este elemento q consulta y todos los pivotes (k_c) del índice del subespacio.

Luego si $max_{1 \leq j \leq k_c} \left| d(q,p_j) - d(e,p_j) \right| > r$ para algún pivote p_j, entonces el elemento e se encuentra fuera del rango de consulta. Se tiene como conclusión que el pivote p_j discrimina de forma exitosa al elemento $e \in I$.

Sea m la cantidad de elementos del subespacio ($m \leq n$), entonces en una consulta al cluster se pueden eliminar a lo sumo m elementos (es decir, todos los elementos del cluster), lo cual habría repartido esa m discriminaciones entre los k_c pivotes, también en una consulta se puede saber qué pivote realizó la discriminación de un elemento $e \in I$.

Siguiendo con este mismo concepto de prestar atención a los elementos discriminados por los distintos pivotes, y con un conjunto interesante de consultas (S búsquedas, donde S tiene que ser un valor grande y representativo) realizadas al Cluster se define "el porcentaje de discriminación del pivote *i*-ésimo p_i [$\%\ Disc\ (p_i)$]:

$$[\%\ Disc\ (p_i)] = Disc\ (p_i)/(S * m)$$

Donde:

- $Disc\ (p_i)$: es la cantidad de elementos que discrimino el pivote p_i.
- $S * m$: representa el total de discriminaciones posibles dentro del cluster.

Luego, se afirma que un pivote p_i es "malo" cuando

$$[\%\ Disc\ (p_i)] < {}^{1}/_{k_c}$$

Donde $1/k_c$ es un umbral que se obtiene de forma experimental, trabajando sobre un espacio métrico sintético o de experimentación, que permita controlar su dimensión y la cantidad de elementos del espacio métrico junto con la cantidad de pivotes en el índice.

Si el pivote i-ésimo p_i tiene su porcentaje de discriminación por debajo de este umbral, se dice que el pivote p_i está siendo muy poco relevante a la hora de discriminar, por lo menos sujeto a las S búsquedas hechas al cluster. Entonces, es aquí cuando se lo selecciona como una víctima dentro del índice del subespacio, para en un futuro reemplazarlo.

En el caso de se tenga más de un elemento pivote que discrimine poco, o que "sea malo", siempre será el último el candidato a ser reemplazado en un futuro.

¿Qué elemento se vuelve pivote? El mejor candidato del cluster

Ya se tiene una política para la selección del pivote víctima dentro de cada cluster, que es aquel que "menos discrimina". Ahora es el momento de seleccionar su reemplazante, luego de realizadas S búsquedas en un cluster determinado, se desea que el índice nos dé la mayor cantidad de resultados cercanos a la consulta y en un buen tiempo de respuesta, utilizando información de búsquedas anteriores, y no cobertura del espacio.

La idea es la de contabilizar una estadística de los elementos $e \in I$ que quedan más veces en la lista de candidatos del cluster obtenida como resultado de las consultas.

Esta estadística se almacena en una estructura de datos simple por cada cluster, como lo es un vector de m componentes, el cual cuenta las veces que el i-ésimo elemento forma parte de la lista de candidatos del cluster. Luego, el elemento que más veces fue candidato va a ser el que la política de selección tome como pivote entrante al índice.

El fundamento de esta política de selección está basado en la idea de que un elemento que formó parte muchas veces de la lista de candidatos del cluster es difícil de discriminar por los pivotes actuales que están dentro del índice del cluster. Esto implica que si se selecciona a este elemento como pivote del índice va a mejorar los porcentajes de discriminación en torno a esa región que lo circunda en futuras búsquedas.

Este no es el único beneficio de esta política de selección. Esta técnica también adapta los pivotes a la región donde se realizan la mayoría de las búsquedas, lo que transforma al índice en una estructura dinámica y que se adapta de forma automática para seguir cumpliendo su principal objetivo: reducir la cantidad de evaluaciones de la función distancia al momento de una búsqueda.

De esta forma, se puede identificar los subespacios con ***SSS*** y ordenar los clusters dentro de una ***Lista de Clusters*** para optimizar las consultas, luego gracias a la construcción inicial del índice con ***SSS*** dentro de cada uno de los subepacios densos se tiene solucionado el problema de la dimensionalidad y un buen cubrimiento de cada subespacio, y finalmente gracias a las dos políticas de actualización del índice se pueden ir adaptando las estructuras de cada subespacio a las búsquedas que se realizan a lo largo del tiempo.

6. Experimentación

En esta sección del trabajo se presentan los resultados experimentales de las pruebas en las que compara el rendimiento de nuestra propuesta, *CMSNMS*, contra las demás, *SSS-NMS*, *SSS* y *Random.*

La implementación de los algoritmos fue desarrollada en Java utilizando la JDK 1.6.0_29 y el entorno de desarrollo de código abierto Eclipse. Además se utilizo MySQL como sistema de gestión de Base de Datos, y la Distancia Euclidia para medir la similitud de los objetos.

La implementación de los índices y políticas de búsquedas de *SSS* y *SSS-NMS* se hicieron de acuerdo a cómo se definieron en [1, 19].

La política *Random* es simple: 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 mostrará el desempeño de la técnica en Espacios Vectoriales sintéticos de dimensiones: $2, 4, 8, 10, 12\ y\ 14$, creando colecciones de datos que condigan con la definición de Espacios Métricos Anidados, utilizando la distribución uniforme.

A lo largo de esta sección se utilizará la experimentación para validar la propuesta en diferentes escenarios.

Primero veremos cómo se comporta la adaptación de pivotes en estos tipos de espacios y si es necesaria su inclusión. A continuación, mostraremos el desempeño de nuestra estructura en un espacio métrico de dimensión conocida (dos), donde podremos ver detalles de la construcción del índice: cómo se reconocen los clusters, cómo quedan esparcidos los pivotes dentro de cada cluster, cómo se conforma la bolsa de elementos sin clusters y el resultado de alguna búsqueda. Finalmente para cada dimensión compararemos el comportamiento de los cuatro algoritmos ya mencionados, teniendo en cuenta su orden espacial y temporal (para indexación y búsqueda), cuando analicemos el orden temporal tendremos en cuenta la cantidad de evaluaciones de la función distancia, ya que este factor es el que pondera esta magnitud.

Cada experiencia se organizó de la siguiente manera:

Se tienen dos data sets que se generan, utilizando la misma distribución, el "*dataSetDatos"* utilizado para crear el índice y el "*dataSetQuery"* utilizado para obtener los elementos de consultas.

Con el fin de lograr aleatoriedad, en un primer nivel se tienen N "corridas", en cada *corrida* se selecciona de manera aleatoria el 90% de los datos de *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 él índice actual.

Se obtienen la cantidad total de evaluaciones de la función distancia (complejidad interna + complejidad externa).

Data Sets y radio de consulta para los primeros tres experimentos

Se puede ver la conformación de los data sets para las tres primeras experimentaciones en el Apéndice A, al igual que los radios de consulta para cada dimensión.

6.1. Experimento 1: ¿Cómo se comporta la Adaptación de Pivotes en los Espacios Métricos Anidados?

Cómo primera experimentación se analiza cómo se comporta la adaptación de pivotes en los espacios métricos anidados. Donde, en el segundo nivel del índice los pivotes se adaptarán a las búsquedas que se están realizando, utilizando las estrategias del pivote entrante y del pivote saliente propuestas en [25], con el fin de mejorar el rendimiento en futuras consultas.

Para esto se evalúa su desempeño en las dimensiones mencionadas anteriormente: $2, 4, 8, 10, 12\ y\ 14$.

Primero, se muestra gráficamente, para R^2, cómo los pivotes se adaptan a las búsquedas que se están realizando a lo largo de las épocas. En la Figura 21 pueden observarse las dos zonas donde se agrupan los datos que forman el índice y los sectores donde se realizan las consultas.

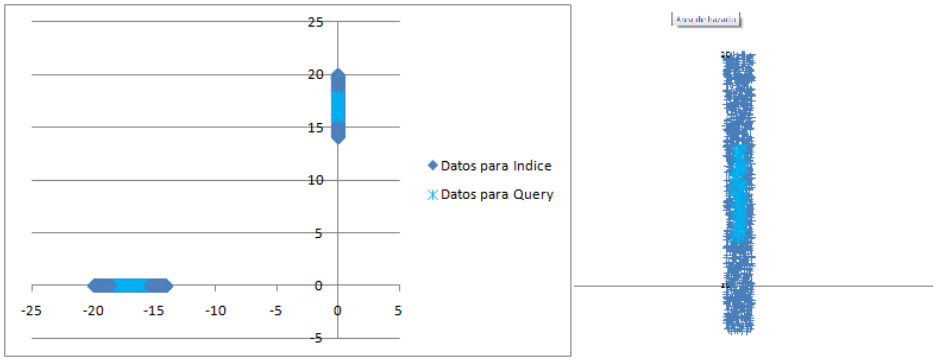

Figura 21: A la izquierda, los datos del índice y los datos para consulta. A la derecha, la ampliación sobre un subespacio.

En la Figura 22 se distinguen los clusters identificados por el algoritmo en el primer nivel del índice, y los pivotes de cada cluster que forman el segundo nivel.

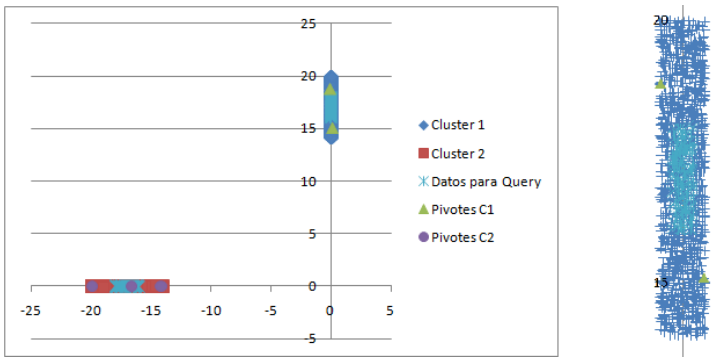

Figura 22: A la izquierda, los clusters del primer nivel y los pivotes del segundo. A la derecha, la ampliación del Cluster 1.

Finalmente en Figura 23 se muestra una secuencia de figuras donde se puede observar cómo los pivotes se adaptan a las búsquedas que se están realizando dentro de cada cluster.

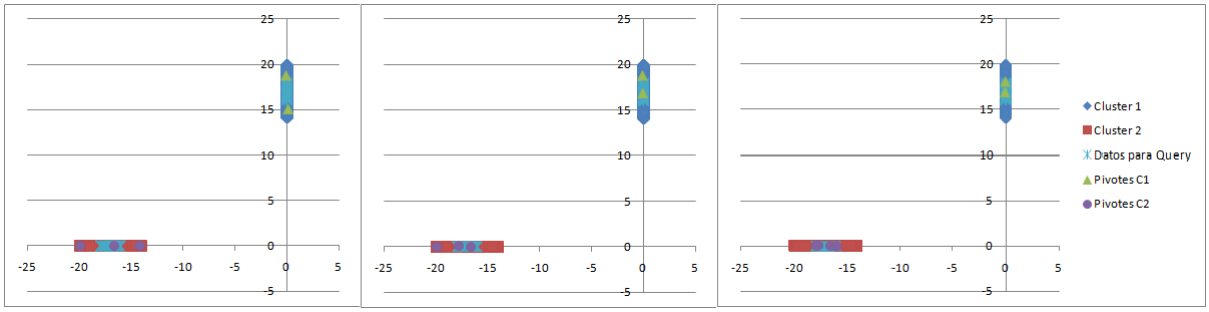

Figura 23: Adaptación de pivotes a las consultas en cada cluster del Espacio Métrico. Épocas 0, 4 y 8 respectivamente.

Luego de ver gráficamente cómo se realiza la adaptación de pivotes en el segundo nivel del índice se continúa con la parte importante de este experimento. Se analiza cómo varía la cantidad de evaluaciones de la función distancia para cada dimensión, a medida que los pivotes se adaptan a las búsquedas que se están realizando en el correr de las épocas. En la Figura 24 para cada una de las 10 épocas se suma el resultado de las 5 corridas, donde la época 0 corresponde a la selección inicial de los pivotes y en sucesivas épocas se van adaptando los pivotes a las búsquedas, si es considerado necesario de acuerdo a las estrategias de pivote entrante y saliente formuladas con anterioridad.

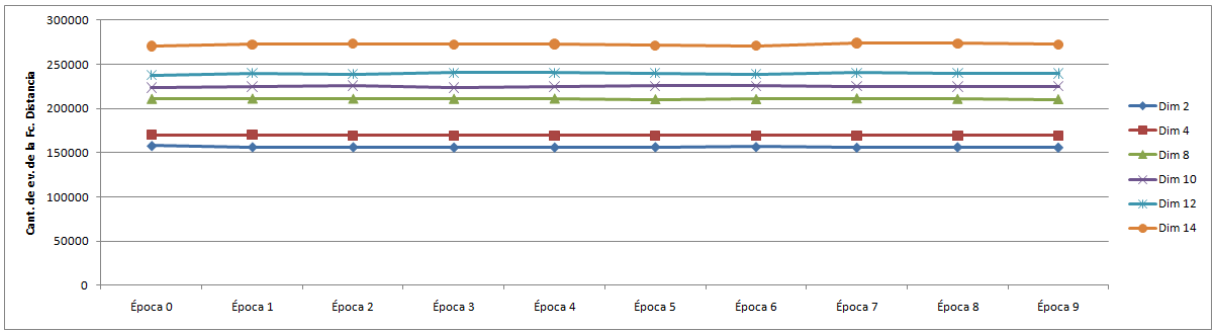

Figura 24: Evolución del rendimiento de la estructura con Adaptación de Pivotes.

Vemos que la adaptación de pivotes no mejora el rendimiento en los Espacios Métricos Anidados para ninguna dimensión, esto se debe a la irregularidad en la topología de los datos que forman estas colecciones, por lo que para futuras experimentaciones omitiremos adaptar los pivotes.

6.2. Experimento 2: Mostrando la Implementación de CMSNMS *en* R^2

Ya se ha visto cómo es la mecánica de la adaptación de pivotes y que no mejora el rendimiento de nuestra estructura en los Espacios Métricos Anidados, por lo que a partir de ese experimento omitimos realizarla.

La segunda 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 ser realiza en R^2.

Primer Nivel del índice

El radio del cluster C, r_c, es igual a $M * \alpha * \rho$, donde M es la máxima distancia entre cada par de elementos en la base de datos, $\alpha = 0{,}4$ es una constante que regula la cantidad de centros de clusters, y $\rho < 1$ es la constante que dictamina la amplitud del radio de cada cluster, que debe ser

menor que uno para que no colisionen la estrategia de selección de centros con *SSS* y las propiedades de la *Listas de Clusters*.

En la Figura 25 se muestran variantes de cómo se construye el primer nivel del índice para diferentes valores de ρ.

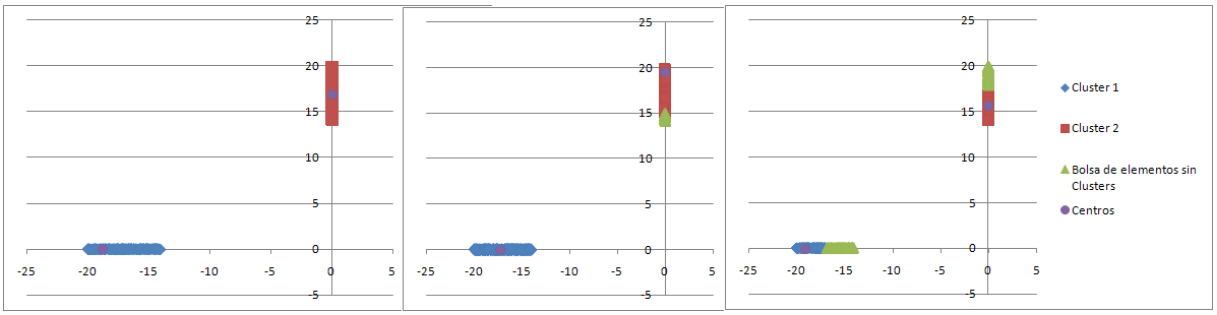

Figura 25: Primer nivel de índice, con $\rho = 0{,}9;\ 0{,}5\ y\ 0{,}2$; en este orden.

Notar como a medida que se disminuye el valor de ρ aumenta la cantidad de elementos sin cluster, y cómo este conjunto queda vacío con $\rho = 0{,}9$. Razón por la cual, de aquí en adelante, se usará este valor para escoger el radio.

Segundo Nivel del índice

Ambos clusters se consideran densos en base a la medida que se da en la propuesta, por lo que en la Figura 26 se pueden observar los pivotes que forman el segundo nivel del índice.

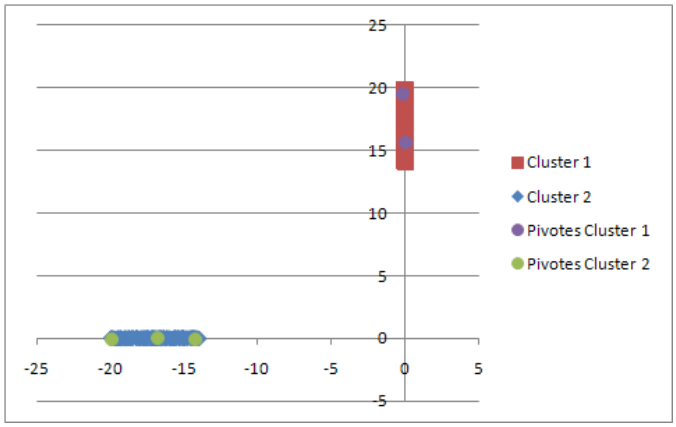

Figura 26: Pivotes que forman el segundo nivel del índice.

Consulta

Se realiza la consulta $Q = (q, r_q)$, con $q = (-16{,}78;\ -0{,}04)$ y $r_q = 0{,}14$.

Q está contenida en el cluster número dos, por lo que se omite inspeccionar el otro, cómo el cluster dos es denso, se utilizan sus pivotes para obtener la lista de candidatos que se muestra en la Figura 27.

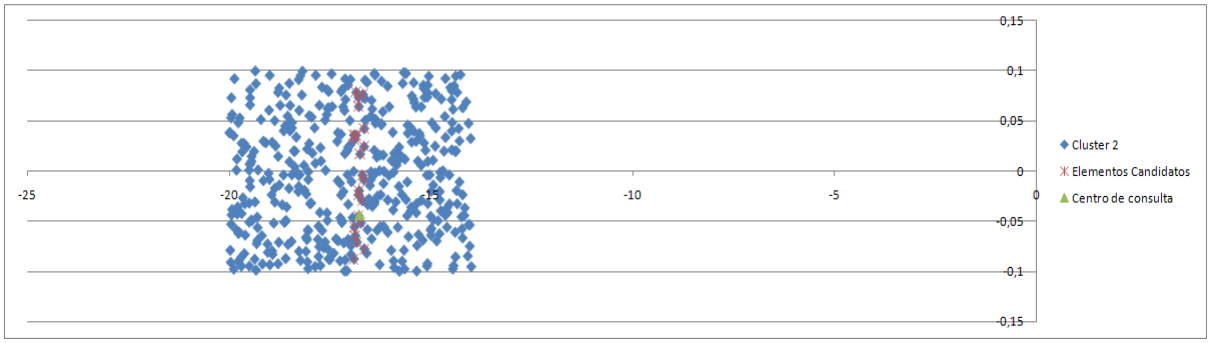

Figura 27: Lista de candidatos de la consulta de centro $(-16{,}78;\ -0{,}04)$ y radio $0{,}14$.

Finalmente la lista de candidatos obtenida debe ser comparada exhaustivamente contra la consulta, para obtener los elementos cercanos cómo se puede observar en la Figura 28.

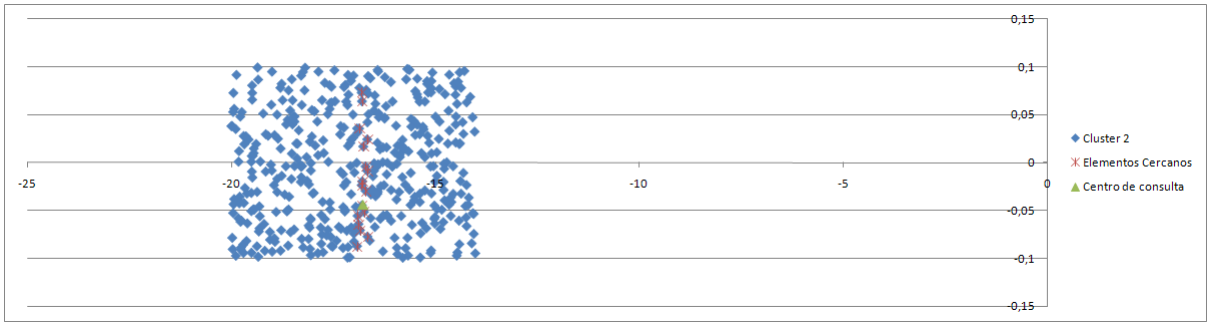

Figura 28: Elementos que forman la respuesta a la consulta dada.

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 ahorrarnos un gran número evaluaciones de la función distancia.

6.3. Experimento 3: Performance y Análisis de Resultados

En esta sección se realiza el experimento más importante, para cada dimensión se compara el comportamiento del nuevo índice *CMSNMS* contra los demás, *SSS-NMS*, *SSS*, y *Random*.

Se utilizan los data sets ya mencionados con anterioridad y se analiza el orden espacial y temporal (para indexación y búsqueda) de cada uno de los índices.

Indexación, orden espacial

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, ($n = 4.000$ en nuestro experimento), es necesario mantener en memoria una matriz triangular, de $\frac{n^2}{2}$ elementos, que conserve la distancia entre cada par de objetos en el índice.

Indexación, orden temporal

Lo mismo sucede para el orden temporal durante la indexación, si tenemos en cuenta el número de evaluaciones de la función distancia como factor preponderante, a medida que insertamos elementos en el índice es necesario compararlos contra todos los elementos ya insertados, para

obtener la matriz de distancias triangular de $\frac{n^2}{2}$ elementos, por lo que el orden temporal de indexación es de $\frac{n^2}{2}$ para todos los algoritmos de construcción.

Búsqueda, orden temporal

Para analizar el orden temporal de las búsquedas se cuentan la cantidad de evaluaciones distancias necesarias para realizar todas las consultas, obteniendo la *Complejidad Total* (*Complejidad Interna* + *Complejidad 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 lograr los elementos cercanos a la consulta.

En la gráfica de la Figura 29 se compara la cantidad de evaluaciones de la función distancia de los cuatro algoritmos a medida que variamos 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 data sets contienen sólo dos clusters. Por lo que a lo sumo el *CMSNMS* puede omitir compararse contra un cluster por consulta. Si aumentamos el número de clusters la diferencia debería ser mayor, lo corroboraremos en el siguiente experimento.

También, como era de esperar, se observa que el *SSS* pierde y gana con respecto a *Random*, esto se debe a la topología irregular del conjunto de datos empleado.

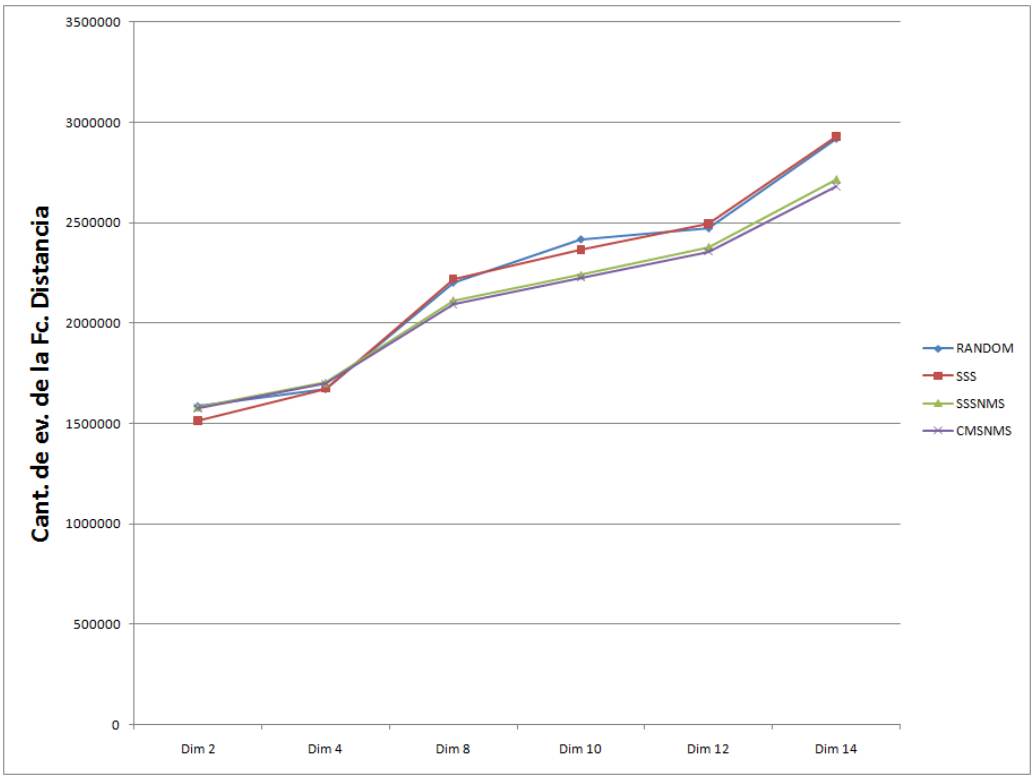

Figura 29: Comparación de los cuatro algoritmos utilizando data sets con dos clusters.

6.4. Experimento 4: *CMSNMS* vs *SSS-NMS*

Finalmente luego de demostrar experimentalmente que para el caso más general de los Espacios Metricos Anidados, conformado por data sets de dos clusters, nuestra propuesta se comporta mejor que las demás, se analiza qué pasa si aumentamos el número de clusters en los data sets.

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.

Data Sets y radio de consulta

Para este experimento se generan nuevos data sets que representen las colecciones propuestas. Podemos encontrar estos data sets en el Apéndice B. Al igual que los radios utilizados para cada dimensión.

En el gráfico de la Figura 30 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 el Experimento 3.

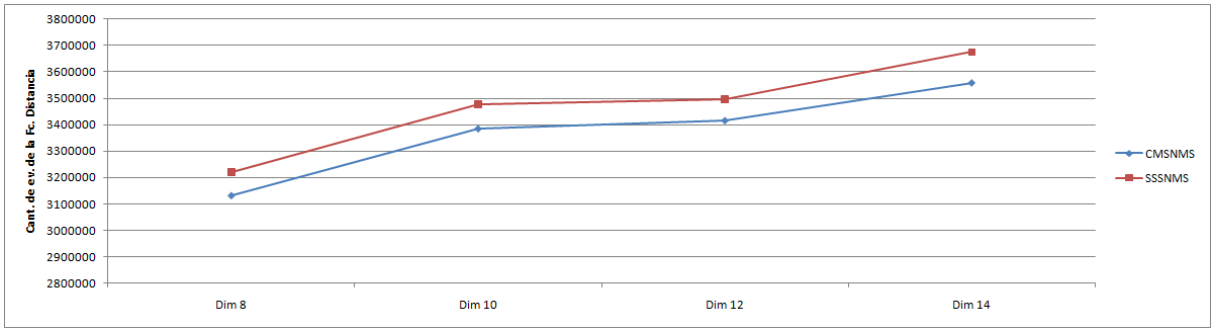

Figura 30: *CMSNMS* vs *SSSNMS*, con data sets con cuatro clusters.

Esto se debe a que al que al aumentar el número de clusters el *CMSNMS* se ve favorecido ya que al utilizar *Listas de Clusters* en el primer nivel puede omitir compararse contra 0, 1, 2 o 3 clusters, mientras que el *SSS-NMS* debe hacerlo contra los 4 en cada comparación. Esta tendencia continúa si se sigue aumentando el número de clusters.

7. Conclusiones

Luego de estudiar los índices actuales, y sus algoritmos de búsquedas, para trabajar sobre las Bases de Datos Métricas encontramos que existen los Espacios Métricos Anidados.

Definimos los Espacios Métricos Anidados cómo los Espacios Métricos donde un gran número de objetos se agrupan en subespacios densos, anidados en 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 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.

¿Por qué el interés en trabajar en los Espacios Métricos Anidados?

- Porque los índices convencionales, cómo el *SSS* no se comportan de manera adecuada en este tipo de Espacios Métricos, cómo vimos en el Experimento 3.
- Porque estos Espacios Métricos pueden ser encontrados en Bases de Datos reales, donde el ejemplo más evidente es el de una colección de imágenes representadas por Feature Vectors, donde las imágenes podrían agruparse por compartir el mismo color principal, diferenciándose entre ellas por las demás características.

La idea de la estructura aquí propuesta es combinar 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 una estructura con mejor desempeño en los Espacios Métricos Anidados.

Las experimentaciones fueron realizadas en Espacios Vectoriales. Si bien estos espacios son un tipo particular de Espacios Métricos, cuando se comparan computacionalmente dos objetos pertenecientes a un mismo Espacio Métrico se lo hace a través de sus Feature Vectors, que se obtienen a partir de una función de transformación adecuada. Además las experimentaciones se realizaron sólo haciendo uso de una función de distancia. Por lo que al no considerar la información de sus coordenadas por separado, sino utilizando los vectores como un objeto completo, la estructura es resistente a las altas dimensiones representacionales de esta clase de vectores.

Cómo resultado hemos encontrado que el desempeño de nuestra estructura es mejor que el de las demás en los Espacios Métricos Anidados propuestos, incluso que para *SSS-NMS*, donde realizaron pruebas adicionales.

En comparación a todos siempre se obtuvieron mejores resultados a partir de la dimensión 8, esto se debe a la dificultad para expresar Espacios Métricos Anidados en espacios de baja dimensión, donde algoritmos como *SSS* o *Random* evidentemente funcionan mejor independientemente de la topología del conjunto de datos.

Al compararse solamente contra *SSS-NMS* siempre se obtuvieron mejores resultados. Como era de esperarse al aumentar el número de clusters en los juegos de datos la diferencia fue aún más acentuada, ya que la estructura aquí propuesta emplea *Listas de Clusters* en el primer nivel, lo que permitió omitir comparaciones contra clusters posteriores del primer nivel si la consulta estaba contenida en algún cluster anterior.

Otro resultado interesante fue que la adaptación de pivotes a las búsquedas que se están realizando dentro de cada cluster no aportó mejores resultados en los Espacios Métricos Anidados. Esto tiene una doble lectura, por un lado cómo nuestra estructura separa los datos en clusters densos, ya se hace una especie de adaptación a las búsquedas cuando se utiliza el primer nivel del índice, ya que

solo se inspecciona dentro del subespacio correspondiente, y la vez este subespacio es un cluster denso y compacto, donde sus pivotes escogidos con el *SSS* ya están ubicados en la mejor posición que pueden estar y no es necesario reubicarlos. Entonces al no adaptar los pivotes del segundo nivel continuamos teniendo una estructura que se comportará bien si las búsquedas posteriores se hacen en cualquier sector del espacio.

Finalizando, si bien los espacios utilizados para llevar a cabo las experimentaciones son Espacios Vectoriales, hemos usado soluciones para Espacios Métricos, tratando a los objetos sólo a través de la función distancia, y ello hace que las estructuras sean resistentes a trabajar en dimensiones altas.

8. Trabajo a Futuro

Algunas cuestiones por abordar son la evaluación de la propuesta en Espacios Vectoriales de mayor dimensión de los aquí empleados, o con colecciones obtenidas de Base 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.

Si bien se probó el algoritmo en Espacios Vectoriales de hasta de dimensión 14 resultaría interesante evaluar su desempeño en dimensiones mayores, ya que se conocen funciones de transformación que convierten objetos multimedia en vectores, y estos generalmente son de dimensión 25 o superior.

Utilizando estas funciones de transformación se podría comparar la estructura contra las demás empleando como datos objetos multimedia, cómo podría ser una Base de Datos de imágenes.

Como se propuso, se ha logrado una estructura que tiene un mejor desempeño en los Espacios Métricos Anidados que las existentes, pero también se podrían hacer experimentaciones en Espacios Métricos Convencionales, para analizar su aplicabilidad en los casos más generales.

Existe en la actualidad la biblioteca SISAP, una biblioteca que contiene varias de las soluciones existentes para espacios métricos en código abierto en lenguaje C (disponible en: http://sisap.org/Metric_Space_Library.html). Por lo tanto, también se propone como trabajo a futuro, la implementación de nuestra propuesta de acuerdo a los requerimientos de SISAP. Para permitir la comparación de nuestra estructura contra otros algoritmos de indexación allí disponibles utilizando las bases de datos que son utilizadas en la comunidad de investigadores en el área de búsquedas en espacios métricos como "*benchmarks*".

9. Bibliografía y Referencias

[1] [Pedreira, 2007] Pedreira, O. and Brisaboa, N. R. Spatial Selection of Sparse Pivots for Similarity Search in Metric Spaces. In Proc. of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07), Lecture Notes in Computer Science (4362), pp. 434-445, 2007. Harrachov, Czech Republic. Springer.

[2] [Arroyuelo] Arroyuelo, D., Ludueña V., Navarro, G., and Reyes, N. Búsquedas en Bases de Datos de Texto y Bases de Datos Métricas. VII Workshop de Investigadores en Ciencias de la Computación (WICC' 2005), pp. 411-415.

[3] [Navarro] Navarro, G. and Reyes, N. Dynamic Spatial Approximation Trees. Journal of Experimental Algorithmics (JEA). Volume 12, June 2008. New York, NY, USA.

[4] [Baeza-Yates, 1994] Baeza-Yates, Cunto, Manber and Wu. Proximity matching using fixed queries trees. In Proc 5th Combinatorial Pattern Matching (CPM '94), LNCS 807 (1994), pp.198-212.

[5] [Uhlmann, 1991b] Uhlmann, J. Satisfying general proximity/similarity queries with metric trees. Information Processing Letters 40, pp. 175-179.

[6] [Yianilos, 1993] Yianilos, P. Data structures and algorithms for nearest neighbor search in general metric spaces. In Proc 4th ACM-SIAM Symposium on Discrete Algorithms (SODA '93), pp. 311-321.

[7] [Chiueh, 1994] Chiueh, T. Content-based image indexing. In Proc of the 20th Conference on Very Large Databases (VLDB '94), pp. 582-593.

[8] [Brin, 1995] Brin, S. Near neighbor search in large metric spaces. In Proc 21st Conference on Very Large Databases (VLDB '95), pp. 574-584.

[9] [Bozkaya, 1997] Bozkaya, T. and Ozsoyogl, M. Distance-based indexing for high-dimensional metric spaces. In Proc ACM SIGMOD International Conference on Management of Data, pp. 357-368.

[10] [Yianilos, 1999] Yianilos, P. Excluded middle vantage point forests for nearest neighbor search. In DIMACS Implementation Challenge, ALENEX '99.

[11] [Kalantari, 1983] Kalantari, I. and McDonald, G. A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering, 9:631–634. IEEE Press. 1983.

[12] [Nolteimer, 1992] Nolteimer, H., Verbarg, K. and Zirkelbach, C. Monotonous Bisector Trees a tool for efficient partitioning of complex schenes of geometric objects In Data Structures and efficient Algorithms, LNCS pp 186-203.

[13] [Dehne, 1987] Dehne, F. and Nolteimer, H. Voronoi trees and clustering problems. Book Syntactic and structural pattern recognition Springer-Verlag New York, Inc. New York, NY, USA.

[14] [Ciaccia 1997] Ciaccia P., Patella, M. and Zezula, P. M-tree: an efficient access method for similarity search in metric spaces. In Proc. of the 23rd Conference on Very Large Databases (VLDB'97) pp. 426-435.

[15] [Navarro, 1999] Navarro, G. Searching in metric spaces by spatial approximation. In Proc. String Processing and Information Retrieval (SPIRE '99) pp. 141-148. IEEE CS Press.

[16] [Vidal, 1986] Vidal, E. An algorithm for finding nearest neighbors in approximately constant average time. Pattern Recognition Letters 4. 145-157.

[17] [Micó 1994] Micó, L., Oncina, J. and Vida, E. 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.

[18] [Burkhard, 1973] Burkhard, W. and Keller, R. Some approaches to best match file searching. Comm. of the ACM 16, 4, 230-236.

[19] [Brisaboa] Brisaboa, N. R., Luaces, M. R., Pedreira, O., Places, A. S. and Seco, D. Indexing Dense Nested Metric Spaces for Efficient Similarity Search. In Proc. of the 7th International Andrei Ershov Memorial Conference (Perspectives of System Informatics) (PSI'09), Novosibirsk, Russia, 2009. To appear in Lecture Notes in Computer Science, Springer.

[20] [Baeza-Yates, 1997] Baeza-Yates. Searching: an algorithmic tour. In A. Kent and J. Williams Eds. Encyclopedia of Computer Science and Technology. Volume 37, pp 331-359. Marcel Dekker Inc.

[21] [Chávez, 1999]. Chávez, E. Optimal discretization for pivot based algorithms. Manuscript. ftp://-garota.fismat.umich.mx.pub.users.elchavez.minimax.ps.gz.

[22] [Mamede] 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), LNCS 3733, pages 843–853, 2005.

[23] [Paredes, 2002] Paredes Moraleda, R. A. Uso de t-spanners para búsqueda en espacios métricos. Universidad de Chile, 2002. Tesis para optar al Título de Ingeniero Civil en Computación y Grado de Magíster en Ciencias, Mención Computación.

[24] [Chávez, 2004] Chávez, E. and Navarro, G. A compact space decomposition for effective metric indexing. Pattern Recognition Letters, 26(9):1363–1376, 2005.

[25] [Salvetti, 2009] Salvetti, M. Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos. Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Rosario, República Argentina.

[26] [Brisaboa] Brisaboa, N. R., Fariña, A., Pedreira, O. and Reyes, N. Similarity search using sparse pivots for efficient multimedia information retrieval. In: Proc. of the 8th IEEE International Symposium on Multimedia (ISM'06), San Diego, California, USA, IEEE Press (2006) 881-888.

[27] [Chávez] Edgar Chávez., Gonzalo Navarro. 2001a. Towards measuring the searching complexity of metric spaces. In Proceedings of the Mexican Computing Meeting, vol. II, 969–972.

10. Agradecimientos

A mis padres que más allá de bancarme todos estos años me hicieron todo lo que soy. Quienes me inculcaron a transitar en cada paso de la vida con responsabilidad, pero remarcándome que jamás debería perder la alegría. Quienes incondicionalmente estuvieron y para que los que siempre, por más de que pasen los años, mis hermanos y yo, seremos sus niños. ¡Los amo, respeto y admiro muchísimo!!

A mis hermanos, con quienes, como todos hermanos, tenemos una relación muy particular. Emi, como dice Maxi, yo sé que me querés y extrañás, aunque me pidas seguido que me vaya a Paraná. Pepi, estos años lejos con Emí nos perdimos mucho de vos… ¿Cuánto paso desde que perdías los mocos y pedías por escuchar la "Chole" todo el tiempo? ¡Los Amo a los 2!!

Mi amor, cuanto preguntaste, estuviste, acompañaste… "¿Estudiaste?..., ¿Cómo vas con la materia?..., ¿Por qué no te sentas un ratito más?...". Te ligaste alguna malhumorada respuesta, pero ¡gracias!!! No sólo por lo que estoy escribiendo, sino por todo lo que no está escrito, y es más importante… ¡Te amo!!

Al resto de mi familia, tíos, abuelos y primos, que los nombraría a todos pero me haría falta otro apéndice, pero en especial a vos Gerar, que sos como mi hermano más grande, y de las mejores personas que conozco. ¡Te quiero un montón loco! ¡Gracias!!

Pasó mucho tiempo desde que llegué a Rosario, pero no me olvido de la primera familia que me abrió las puertas de su casa y me hizo uno más, para vos Kily, Patri y Hugo.

Aquellos primeros años de Rosario, que también los compartí con vos Chaca, viviendo en ese departamento me recibí de *Abogado* antes que de *Licenciado en Ciencias de la Computación*. Para ustedes también amigos… Mono, Pata, Renzo y Flaco.

Para ustedes casi "colegas" y amigos, Mariano y Damián quienes me ayudaron tanto, tanto en este último envión y estuvieron desde el principio. No les fallé… ¡Acá estamos! ¡Muchas gracias!!!

A los amigos que me dejaron mis dos primeros laburos, en Accenture y Cfyar, Juampi, Uli, Javier y Pablo, de quienes aprendí mucho y me ayudaron a crecer como profesional.

A mis amigos de toda la vida de Maciá y de Cutral Co: Luisi, Jose, Nancy, Anto, Marco, Ruso, Martin, Gusti, Gordo, Maxi, Fabri, Fede y Lucho. ¡Para ustedes también!

A Fernanda, Diego, Euge D., Luis, Euge Z. y Seba… Que los conocí en este último año, pero pareciera que fuésemos amigos de toda la vida… Lo comparto con ustedes porque sé que están contentos como yo. "El cuchi está feliz". ¡Los quiero!!

A Nora Reyes, Claudia Deco y Cristina Bender, mi directora, co directora y "co co directora" respectivamente. Gracias por la paciencia, por la dedicación y por la ayuda que me brindaron para llevar adelante este trabajo que me permite cerrar este ciclo.

11. Apéndices

Apéndice A - Conformación de los data sets y radios de consulta para los experimentos 1, 2 y 3.

Para cada dimensión los data sets utilizados se obtienen la siguiente manera:

Dimensión 2

El *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].
- Los datos para x_1 se generan aleatoriamente entre [-0,1; 0,1].

En el segundo cluster:
- Los datos para x_1 se generan aleatoriamente entre [14; 20].
- Los datos para x_0 se generan aleatoriamente entre [-0,1; 0,1].

El *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].
- Los datos para x_1 se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:
- Los datos para x_1 se generan aleatoriamente entre [16; 18].
- Los datos para x_0 se generan aleatoriamente entre [-0,05; 0,05].

Dimensión 4

El *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].
- Los datos para x_1, x_2, x_3 se generan aleatoriamente entre [-0,1; 0,1].

En el segundo cluster:
- Los datos para x_0, x_1, x_2 se generan aleatoriamente entre [-0,1; 0,1].
- Los datos para x_3 se generan aleatoriamente entre [14; 20].

El *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].
- Los datos para x_1, x_2, x_3 se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:
- Los datos para x_0, x_1, x_2 se generan aleatoriamente entre [-0,05; 0,05].
- Los datos para x_3 se generan aleatoriamente entre [16; 18].

Dimensión 8

El *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].
- 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].
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].

El *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].
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].
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].

Dimensión 10

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

Dimensión 12

El *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].
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 aleatoriamente entre [-0,1; 0,1].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [14; -20].
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 [-0,1; 0,1].

El *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].
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 aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [16; 18].
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].

Dimensión 14

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

Radio de consulta

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 Datos que retornará una consulta. Este valor es el 2%.

Para dos dimensiones el radio de consulta es igual a 0,14.

Para cuatro dimensiones el radio de consulta es igual a 0,15.

Para ocho dimensiones el radio de consulta es igual a 0,193.

Para diez dimensiones el radio de consulta es igual a 0,211.

Para doce dimensiones el radio de consulta es igual a 0,219.

Para catorce dimensiones el radio de consulta es igual a 0.245.

Apéndice B - Conformación de los data sets y radios de consulta para el experimento 4.

Para cada dimensión los data sets utilizados se obtienen la siguiente manera:

Dimensión 8

El *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].
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 primer cluster:

Los datos para x_2 se generan aleatoriamente entre [-20; -14].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7$ se generan aleatoriamente entre [-0,1; 0,1].

En el tercer cluster:

Los datos para x_4 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7$ se generan aleatoriamente entre [-0,1; 0,1].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [14; 20].
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].

El *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].
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_2 se generan aleatoriamente entre [-18; -16].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7$ se generan aleatoriamente entre [-0,05; 0,05].

En el tercero cluster:

Los datos para x_4 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7$ se generan aleatoriamente entre [-0,05; 0,05].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [16; 18].
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].

Dimensión 10

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

En el primer cluster:

Los datos para x_2 se generan aleatoriamente entre [-20; -14].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

En el tercer cluster:

Los datos para x_4 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,1; 0,1].

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [-18; -16].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

En el tercero cluster:

Los datos para x_4 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9$ se generan aleatoriamente entre [-0,05; 0,05].

Dimensión 12

El *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].
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 aleatoriamente entre [-0,1; 0,1].

En el primer cluster:

Los datos para x_2 se generan aleatoriamente entre [-20; -14].
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 [-0,1; 0,1].

En el tercer cluster:

Los datos para x_4 se generan aleatoriamente entre [14; 20].
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].
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 [-0,1; 0,1].

El *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].
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 aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [-18; -16].
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 tercero cluster:

Los datos para x_4 se generan aleatoriamente entre [16; 18].
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].
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].

Dimensión 14

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

En el primer cluster:

Los datos para x_2 se generan aleatoriamente entre [-20; -14].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

En el tercer cluster:

Los datos para x_4 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [14; 20].
Los datos para $x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,1; 0,1].

El *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].
Los datos para $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

En el segundo cluster:

Los datos para x_2 se generan aleatoriamente entre [-18; -16].
Los datos para $x_0, x_1, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

En el tercero cluster:

Los datos para x_4 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

En el cuarto cluster:

Los datos para x_6 se generan aleatoriamente entre [16; 18].
Los datos para $x_0, x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$ se generan aleatoriamente entre [-0,05; 0,05].

Radio de consulta

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 datos que retornará una consulta. Este valor es el 2%.

Para ocho dimensiones el radio de consulta es igual a 0,28.

Para diez dimensiones el radio de consulta es igual a 0,304.

Para doce dimensiones el radio de consulta es igual a 0,306.

Para catorce dimensiones el radio de consulta es igual a 0,326.