Eligiendo Raíces para el Árbol de Aproximación Espacial

Alejandro Gómez, Verónica Ludueña y Nora Reyes
Departamento de Informática, Universidad Nacional de San Luis.
San Luis, 5700, Argentina
{*agomez,vlud,nreyes*}*@unsl.edu.ar*

Abstract

Many computational applications need to search information in a database. At the present time the predominance of multimedia databases does that the *similarity search* or *proximity search*, that is to look for elements of the database that are similar to a given query element, becomes a preponderant concept.

The Spatial Approximation Trees have shown to be competitive for similarity search in spaces with medium to high dimensionality ("difficult" spaces) or for queries with low selectivity. Nevertheless, for its construction its root was chosen randomly and it completely determines the tree, not only in its shape but also in its searching performance. Thus, our interest was to optimize searches in this data structure trying to choose the tree root in a way that the characteristics of indexed space can be reflected. We consider that, by this way, the data structure can adapt itself better to the dimension of the considered metric space, which results in more efficient similarity searches.

Keywords: similarity search, metric spaces, databases, algorithms.

Resumen

Muchas aplicaciones computacionales necesitan buscar información en una base de datos. En la actualidad el predominio de las bases de datos multimedia hace que la *búsqueda por similitud* o *búsqueda por proximidad*, es decir buscar elementos de la base de datos que sean similares a un elemento de consulta dado, se vuelva un concepto preponderante.

El Árbol de Aproximación Espacial ha demostrado ser muy competitivo para la búsqueda por similitud en espacios métricos de media a alta dimensionalidad (espacios "difíciles") o para responder a consultas con baja selectividad. Sin embargo, para su construcción se elegía su raíz al azar y ello determinaba completamente el árbol tanto en su forma como en su desempeño. Así, nuestro interés fue el de optimizar las búsquedas en dicha estructura tratando de que la raíz sea elegida de manera tal que refleje alguna de las características propias del espacio métrico a indexar. Creemos que de esta forma permitimos que la estructura se adapte mejor a la dimensión intrínseca del espacio métrico considerado, lo cual redunda en búsquedas más eficientes.

Keywords: búsqueda por similitud, espacios métricos, bases de datos, algoritmos.

1. INTRODUCCIÓN Y MOTIVACIÓN

Con la evolución de las tecnologías de información y comunicación, han surgido almacenamientos no estructurados de información. No sólo se consultan nuevos tipos de datos tales como texto libre, imágenes, audio y vídeo; sino que, en algunos casos, no se puede estructurar más la información en claves y registros. Estos tipos de datos son difíciles de estructurar para adecuarlos al concepto tradicional de búsqueda. Aún cuando sea posible una estructuración clásica, nuevas aplicaciones tales como la minería de datos requieren acceder a la base de datos por cualquier campo y no sólo los marcados como "claves". Así, han surgido aplicaciones en grandes bases de datos donde se desea buscar objetos similares. Este tipo de búsqueda se conoce con el nombre de *búsqueda aproximada* o *búsqueda por similitud* y tiene aplicaciones en numerosos campos.

Como en toda aplicación que realiza búsquedas, surge la necesidad de tener una respuesta rápida y adecuada. El planteo general del problema es: existe un universo $\mathbb{U}$ de *objetos* y una función de distancia positiva d: $\mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}^+$ definida entre ellos. Esta función de distancia satisface los axiomas que hacen que el par (U, d) sea un *espacio métrico*: positividad estricta ($d(x, y) = 0 \Leftrightarrow x = y$), simetría ($d(x, y) = d(y, x)$) y desigualdad triangular ($d(x, z) \leq d(x, y) + d(y, z)$). Mientras más "similares" sean dos objetos menor será la distancia entre ellos. Tenemos una *base de datos* finita $S \subseteq \mathbb{U}$ que puede ser preprocesada (v.g. para construir un índice). Luego, dado un nuevo objeto del universo (una *query* q), debemos recuperar todos los elementos similares que se encuentran en la base de datos. Existen dos consultas básicas de este tipo:

Búsqueda por rango: recuperar todos los elementos de S a distancia a lo más r de un elemento q dado.

Búsqueda de k vecinos más cercanos: dado q, recuperar los k elementos más cercanos a q en S.

La distancia se considera costosa de evaluar (por ejemplo, comparar dos huellas dactilares). Por lo tanto, es usual definir la complejidad de la búsqueda como el número de evaluaciones de distancia realizadas, dejando de lado otras componentes. Entonces, el objetivo es generar un índice que permita reducir al máxino el cálculo de distancias durante una búsqueda.

Un caso particular de este problema surge cuando el espacio es un conjunto D-dimensional de puntos y la función de distancia pertenece a la familia L_p de Minkowski: $L_p = (\sum_{1 \leq i \leq d} |x_i - y_i|^p)^{1/p}$. Existen métodos efectivos para buscar sobre espacios D-dimensionales, tales como *Kd–trees* [2, 3] o *R-trees* [8]; pero, para 20 dimensiones o más dejan de trabajar bien. Aunque nos dedicamos a espacios métricos generales, las soluciones planteadas son adecuadas para espacios D–dimensionales.

En [6, 4, 5] se muestra que el concepto de dimensionalidad intrínseca se puede entender aún en espacios métricos generales, se da una definición cuantitativa de ella y se muestra analíticamente la razón para la llamada "*maldición de la dimensionalidad*". Es interesante notar que el concepto de dimensionalidad está relacionado con la "*facilidad*" o "*dificultad*" para buscar en un espacio vectorial D-dimensional. Se dice que un espacio métrico general es más "difícil" (dimensión intrínseca más alta) que otro cuando su histograma de distancia es más concentrado. Esto hace que el trabajo de cualquier algoritmo de búsqueda por similitud sea más dificultoso. En el caso extremo tenemos un espacio donde $d(x, x) = 0$ y $\forall y \neq x, d(x, y) = 1$. En este caso, la consulta q debe ser exhaustivamente comparada contra cada elemento en el conjunto.

De las situaciones descriptas vemos que necesitamos contar con índices que permitan responder eficientemente a cada consulta. Existen numerosos métodos para preprocesar un conjunto a fin de reducir el número de evaluaciones de distancia y, en general, todos ellos se basan en dividir la base de datos, lo que se ha heredado de las ideas clásicas de *dividir para conquistar* y de la búsqueda de datos

típicos (v.g. árboles de búsqueda binaria). Uno de estos métodos que sigue un enfoque distinto y que ha demostrado ser eficiente es el *Árbol de Aproximación Espacial* (*SAT* por su sigla en inglés), ver [12], que se basa en la aproximación espacial. Esta estructura de datos es estática; es decir, necesita contar con los elementos de la base de datos de antemano para construir el índice que luego permita responder consultas por similitud. Nuestro trabajo consistió en optimizar esta estructura de datos a fin de mejorar, principalmente, su desempeño en las búsquedas.

El *SAT* ha demostrado ser muy competitivo en espacios métricos de media o alta dimensionalidad o para responder a consultas con baja selectividad. Sin embargo, para su construcción se elegía su raíz al azar y ello determinaba completamente el árbol tanto en su forma como en su desempeño [9, 13]. Así, nuestro interés fue el de optimizar las búsquedas en dicha estructura seleccionando de diferente manera la raíz del árbol, de forma tal que refleje alguna de las características propias del espacio métrico a indexar. Creemos que vale la pena tomarse el trabajo de elegir mejor la raíz y de esta forma permitir que la estructura se adapte mejor a la dimensión intrínseca del espacio métrico considerado, lo cual redunda en búsquedas más eficientes.

2. ÁRBOL DE APROXIMACIÓN ESPACIAL

El método de aproximación espacial comienza la búsqueda en algún punto del espacio y trata de acercarse "espacialmente" a la consulta q. La estructura de datos que utiliza este método de búsqueda es el *SAT*. Comenzaremos viendo en qué consiste la aproximación espacial.

Consideremos un espacio métrico $(\mathbb{U}, d)$ y a $S \subseteq \mathbb{U}$ que será nuestra base de datos. Para describir la idea básica de la aproximación espacial analizaremos las consultas de 1-*vecino más cercano* o 1-*NN*. Al buscar el vecino más cercano a q nos posicionamos en un elemento $a \in S$ elegido al azar, y nos acercarmos cada vez más y más a q moviéndonos a otro elemento $b \in S$ tal que $d(b, q) < d(a, q)$. Cuando no podamos acercarnos más a q será porque encontramos el elemento más cercano a él. Este proceso debe hacerse entre elementos que son "vecinos", o sea, estando en a sólo podemos acercarnos a q por medio de alguno de los vecinos de a, conjunto denotado de aquí en adelante como $N(a)$.

De acuerdo a esto, la estructura que más naturalmente se adapta a la restricción de movernos sólo hacia los vecinos de un elemento es un grafo dirigido, en donde los nodos se corresponden con los elementos de S y existen arcos entre un elemento y cada uno de sus vecinos. Los arcos denotan los posibles movimientos que podemos hacer para acercarnos a q. Concretamente, existe un arco desde a hasta b si es posible moverse de a a b en un único paso. En un espacio vectorial, el grafo minimal que buscamos corresponde a la triangulación clásica de Delaunay (un grafo donde los elementos que son vecinos de Voronoi están conectados). Así la respuesta ideal en términos de complejidad de espacio es el grafo de Delaunay (generalizado para espacios arbitrarios), y permitirá también búsquedas rápidas.

Lamentablemente no en todos los espacios métricos se puede construir un grafo de aproximación espacial. Esto nos lleva a realizar simplificaciones que nos permitirán resolver consultas del vecino más cercano a un $q \in S$ usando un árbol en lugar de un grafo, el *SAT*.

La construcción del *SAT* comienza con la selección, de manera *aleatoria*, de un elemento $a \in S$ que será la raíz o nodo de inicio del árbol, es decir el elemento desde el cual comienzan todas nuestras búsquedas. Luego se selecciona un conjunto adecuado de vecinos $N(a)$, que verifiquen:

Condición 2: (dados $a \in S$) $\forall x \in S, x \in N(a) \Leftrightarrow \forall y \in N(a) - \{x\},\ d(x, y) > d(x, a)$.

Así, los elementos de la base de datos que integran el conjunto $N(a)$ son aquellos que se encuentran más cerca de a que de cualquier otro elemento de la base de datos. Si comenzamos con el nodo raíz a debemos considerar la "bolsa" $B(a)$ para a, que inicialmente mantiene los elementos

en $S - \{a\}$. Se ordena el conjunto $B(a)$ por distancia a a de manera creciente, quedando primero el elemento que espacialmente es el más cercano a a. Sea $N(a)$ inicialmente vacío, se considera en orden cada elemento $b \in B(a)$; si ocurre que b es más cercano al nodo a que a cualquier elemento ya presente en $N(a)$, entonces se agrega b a $N(a)$. Analizados todos los $b \in B(a)$, se habrá determinado un conjunto $N(a)$ adecuado. Ahora hay que ocuparse de los elementos que fueron descartados como vecinos de a. Para ello se coloca cada elemento $b \in S - (a \cup N(a))$ en la bolsa $B(c)$ del elemento $c \in N(a)$ más cercano a b. Ya ubicado cada elemento en su bolsa, se considera a cada vecino de a como una raíz y se procesan recursivamente los elementos de su bolsa de la manera mencionada.

El algoritmo de la Figura 1 detalla este proceso de construcción. Para construir un *SAT* éste se debe invocar inicialmente con $BUILDTREE(a, S - a)$, en donde a será la raíz del árbol.

```
BuildTree(Nodo a, Conjunto de Elementos S)

1.   N(a) ← ∅          /* vecinos de a */
2.   R(a) ← 0          /* radio de cobertura */
3.   Ordenar S por distancia a a (más cercano primero)
4.   For v ∈ S
5.       R(a) ← máx(R(a), d(v,a))
6.       If ∀b ∈ N(a), d(v,a) < d(v,b) Then
7.           N(a) ← N(a) ∪ {v}
8.   For b ∈ N(a) S(b) ← ∅          /* subárboles */
9.   For v ∈ S − N(a)
10.      Sea c ∈ N(a) el que minimiza d(v,c)
11.      S(c) ← S(c) ∪ {v}
12.  For b ∈ N(a) BuildTree(b, S(b))          /* construir subarboles */
```

Figura 1: Algoritmo para construir un *SAT*.

Para evitar algunas comparaciones de distancia durante las búsquedas, se almacena en cada nodo b del árbol su radio de cobertura $R(b)$, que es la máxima distancia entre b y un elemento del subárbol del cual b es la raíz.

Una vez definida la estructura que nos va a permitir buscar por aproximación espacial, consideraremos las búsquedas por rango con radio r. La clave es que aunque $q \notin S$, la respuesta a la búsqueda son elementos $q' \in S$. La idea es usar el árbol para simular que buscamos un $q' \in S$ desconocido. Se sabe que $d(q, q') \leq r$ y que por la desigualdad triangular $\forall x \in U$, $d(x, q) \leq d(x, q') + d(q', q)$. Pero, como $d(q, q') \leq r$, podemos reemplazar $d(q, q')$ por r y manteniendo la desigualdad $d(x, q) - r \leq d(x, q')$. Por otro lado, por desigualdad triangular tenemos que $d(x, q') \leq d(x, q) + r$.

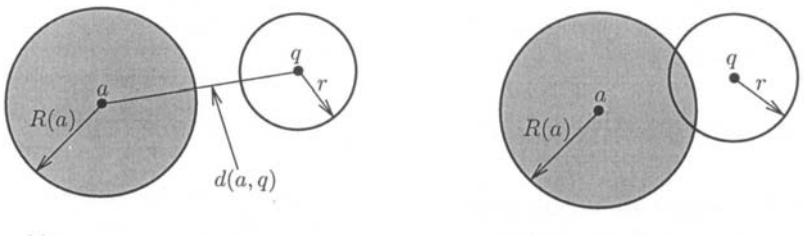
Si buscamos un q conocido, vamos directamente al vecino de a más cercano a q. Al ser q' desconocido no sabemos cuál es el vecino de a más cercano a q'. Entonces, estando en un nodo a, debemos determinar el vecino c más cercano a q entre los elementos de $\{a\} \cup N(a)$. Así, para $b \in \{a\} \cup N(a)$ se cumple que $d(c, q) < d(b, q)$. Pero es posible que $d(c, q') > d(b, q')$ y por lo tanto no encontraremos a q' si solamente buscamos dentro del subárbol de c. Entonces, debemos ingresar en todos los vecinos $b \in N(a)$ que cumplan que $d(q', b) \leq \text{mín } d(c, q) + 2r, c \in \{a\} \cup N(a)$. En otro caso, no estamos seguros de ello y debemos entrar al subárbol de b; porque el q' que estamos buscando puede diferir de q en a lo más r, por lo que podría haber sido insertado dentro de los subárboles de aquellos vecinos b.

Por lo tanto, lo que fue concebido como una búsqueda por aproximación espacial siguiendo un único camino, ahora se combina con backtracking para que busquemos por varios caminos.

Se puede mejorar el algoritmo si consideramos que cuando buscamos un elemento $q \in S$ seguimos un único camino desde la raíz hasta q pero sabemos que, si la búsqueda se encuentra en el nodo a del árbol, se puede evitar entrar en cualquier elemento $x \in N(a)$ tal que $d(q,x) > 2r + \min\{d(q,c), c \in \{a'\} \cup N(a'), a' \in A(a)\}$, donde $A(a)$ es el conjunto de ancestros de a (incluyendo a a) y $N(A(a)) = \bigcup_{a' \in A(a)} N(a')$, debido a que se puede mostrar, usando la desigualdad triangular, que ningún q' con $d(q,q') \leq r$ se pudo almacenar dentro de x.

Otra posibilidad de optimizar las búsquedas es usando la información almacenada sobre el radio de cobertura $R(a)$ de un nodo a, que nos permiten podar cierta parte del árbol, ya que no deberíamos ingresar a un subárbol con raíz a que verifica que $d(q,a) > R(a) + r$, porque esto implica que $d(q',a) > R(a)$ para cualquier q' tal que $d(q,q') \leq r$. Por la forma en que hemos definido $R(a)$, q' no puede encontrarse en el subárbol de a.

Las Figuras 2 y 3 ilustran las situaciones con las que nos podemos encontrar durante las búsquedas.



(a) **(b)**

Figura 2: En **(a)** se cumple que $d(a,q) > R(a) + r$, por lo que no debemos entrar en el subárbol de a ya que ninguno de sus elementos va a estar dentro del rango de la consulta. En **(b)** se cumple que $d(a,q) \leq R(a) + r$, así debemos entrar en el subárbol de a ya que es posible (no seguro) que alguno de sus elementos se encuentre dentro del rango.

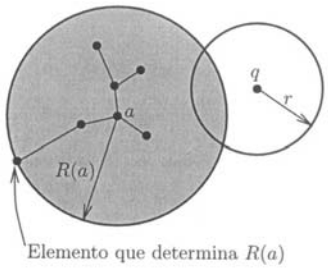


Figura 3: En este caso se cumple que $d(a,q) \leq R(a) + r$ y se observa que aunque debemos entrar en el subárbol de a, no existe ningún elemento en ese subárbol que esté dentro del rango de búsqueda.

En [12] se muestra que el *SAT* posee mejor desempeño en las búsquedas respecto de otras estructuras. El algoritmo de la Figura 4 describe el proceso de búsqueda que hemos explicado.

3. SELECCIÓN DE LA RAÍZ PARA EL SAT

Como se ya se mencionó, al construir el *SAT* éste queda completamente determinado por la elección de la raíz; es decir que, si para la misma base de datos tomáramos otro objeto como la raíz el árbol resultante sería distinto. Como esta elección se realiza al azar, es posible encontrar buenas raíces

```
BúsqRango (Nodo a, Query q, Radio r, Distancia d_mín)

1. If d(a,q) ≤ R(a) + r Then
2.     If d(a,q) ≤ r Then Informar a
3.     d_mín ← mín {d_min} ∪ {d(q,c), c ∈ N(a)}
4.     For b ∈ N(a)
5.         If d(b,q) ≤ d_mín + 2r Then BúsqRango (b, q, r, d_mín)
```

Figura 4: Algoritmo para buscar q con radio r en un *SAT*.

logrando que el árbol generado se comporte mejor principalmente durante las búsquedas, pero también podemos encontrar raíces poco convenientes. La Figura 5 muestra un ejemplo de un espacio y dos posibles árboles para dicho espacio generados a partir de la elección de diferentes raíces.

Sin embargo, elegir la raíz aleatoriamente es atractivo porque no hay costos adicionales, la elección es gratis dado que medimos nuestros costos en *cantidad de evaluaciones de la función de distancia*. Por otra parte, parece razonable utilizar información de la base de datos para seleccionar una buena raíz a costa de pagar un cierto costo en evaluaciones de distancia, esperando que éste se compense al disminuir principalmente los costos de las búsquedas, o al menos de la construcción.

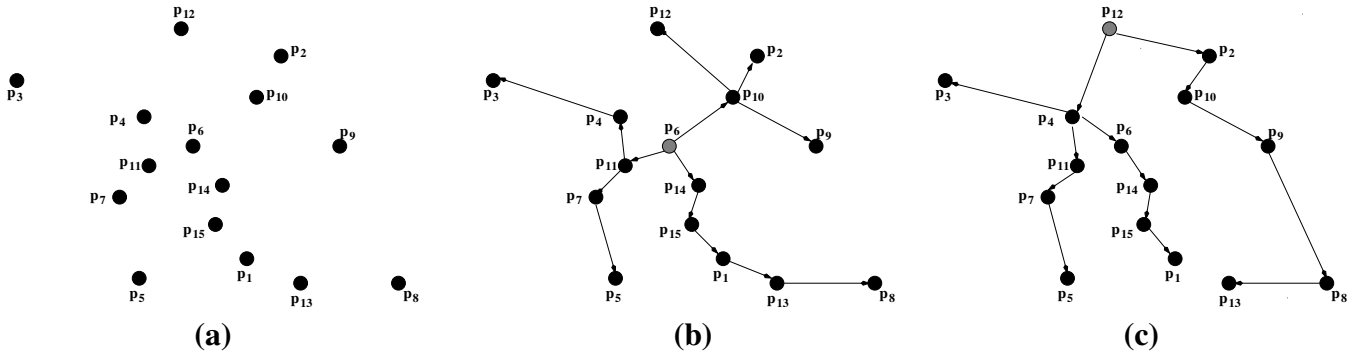


Figura 5: En (a) vemos un posible espacio de puntos, en (b) se muestra el *SAT* obtenido para ese espacio tomando como raíz al punto p_6, y en (c) se ve el *SAT* generado eligiendo a p_{12} como raíz.

3.1. Métodos de Selección de la Raíz

Aquí asumiremos que un elemento es mejor raíz que otro si en el árbol, que desde él se genere, se obtienen mejores costos principalmente en las búsquedas. El énfasis estaría puesto en las búsquedas, porque al ser el *SAT* una estructura estática, no admite ni inserciones ni eliminaciones, la selección de la raíz se realiza una única vez antes de construir el árbol.

Hemos estudiado diferentes métodos para seleccionar un elemento como raíz en el *SAT* y a través del análisis experimental determinamos qué método obtiene los mejores costos de búsqueda. Aunque nos interesarán más los costos de las búsquedas, destacamos para cada método el costo de la elección de la raíz medido en cantidad de evaluaciones de la función de distancia. En algunos de ellos subyacen ideas de teoría de grafos, otros son adaptaciones de métodos usados en otras estructuras para efectuar procesos similares y el método *CSA*, propuesto en [13], pensado específicamente para el *SAT*.

Los métodos considerados son: Método CSA (Centroid Selection Algorithm), Aleatorio, Sampling, M-LB-DIST (Maximum Lower Bound on DISTance), mM-LB-RAD (minimun Maximun Lower Bound RADius), mM-AVG-RAD (minimun Maximun Average RADius).

Los últimos cuatro métodos se han adaptado desde métodos utilizados en el *M–tree* para seleccionar el centro de un nuevo nodo luego de una operación de *split* [7]. Fueron adaptados de manera tal que pudieran usarse en el *SAT* para elegir una raíz, manteniendo los costos razonables. Para identificar fácilmente cada método del *M–tree* hemos mantenido sus nombres originales.

A continuación se hará una breve descripción de cada uno. Para el análisis de los costos de selección de la raíz, y suponiendo que $S \subseteq \mathbb{U}$ es nuestra base de datos, consideramos $N = |S|$.

3.1.1. Método CSA (Centroid Selection Algorithm)

Este método propuesto para elegir la raíz del *SAT*, fue presentado por Penarrieta, Morriberón y Cuadros-Vargas en [13]. La idea subyacente es que un elemento que podría ser una buena raíz para el *SAT* sería aquél que es cercano al centroide ideal de la base de datos. Está basado en el algoritmo *HF* presentado en la Familia Omni [14] y se refleja en el siguiente algoritmo:

1. Seleccionar aleatoriamente un elemento $s \in S$.
2. Encontrar el elemento más lejano e_1 de s.
3. A partir de e_1 determinar su elemento más distante e_2.
4. Seleccionar como raíz al elemento c que satisfaga que $d(e_1, c) \simeq (e_2, c)$ y que minimice $|d(e_1, e_2) - (d(e_1, c) + d(e_2, c))|$.

Este último paso es muy importante porque pueden existir varios candidatos para c, pero se elige aquél que minimice el perímetro del triángulo formado por e_1, e_2 y c y que además no se encuentre retirado del centro del triángulo. La Figura 6 permite visualizar los pasos a desarrollar en el algoritmo. Este método necesita realizar $3N$ evaluaciones de distancia.

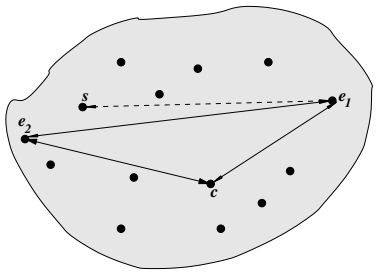


Figura 6: Idea geométrica que da origen al Método CSA.

3.1.2. Método Sampling

Es una política aleatoria que trabaja sobre una muestra de objetos de S de tamaño fijo $s > 1$. Seleccionamos al azar s objetos de la base de datos, y elegimos como raíz al elemento que tenga *menor distancia máxima* a los $s - 1$ objetos restantes. Es fácil observar que en este caso se realizan $\frac{s * (s-1)}{2}$ cálculos de la función de distancia, lo que nos daría un costo de $O(s^2)$. Con el fin de mantener este costo lineal respecto de la cardinalidad de S, elegimos a $s = \sqrt{N}$; así el método debe realizar N cálculos de distancia para elegir la raíz.

3.1.3. Método M-LB-DIST (Maximum Lower Bound on DISTance)

Esta política difiere de la anterior en que necesita utilizar todo el conjunto de elementos. Elegimos un objeto al azar y calculamos las distancias a todos los demás objetos, luego seleccionamos el objeto más distante como raíz. Formalmente, elegimos un elemento $x \in S$ aleatoriamente y determinamos el $y \in S$ tal que $d(x, y) = \max_{z \in S}\{d(x, z)\}$. Claramente este proceso realiza N evaluaciones de distancia para obtener la raíz.

3.1.4. Método mM-LB-RAD (Minimun Maximun Lower Bound Radius)

Se elige un objeto $x \in S$ al azar, y se calculan las distancias de todos los elementos $z \in S$ a x, luego elegimos como raíz el objeto y para el que $\text{máx}_{z \in S}\{|d(y,x) - d(z,x)|\}$ sea mínimo. Este criterio selecciona un elemento y que está a una distancia intermedia de x. Aunque la manera en que se elige a un elemento y como raíz es más compleja que en el método anterior, la cantidad de cálculos de distancia es también N.

3.1.5. Método mM-AVG-RAD (Minimun Maximun Average Bound Radius)

Este método surgió como una alternativa al anterior y trata de ser neutral con respecto a la información de los límites superior e inferior, tomando el promedio y aplicando el criterio de selección $\text{mín} - \text{máx}$. Así, elegimos nuevamente al azar un elemento $x \in S$, calculamos para todo $z \in S$ la distancia $d(x,z)$ y elegimos un elemento y como raíz tal que $\text{máx}_{z \in S}\{(d(y,x) + d(z,x))/2\}$ sea mínimo. Este método también necesita N cálculos de distancia para poder seleccionar la raíz.

4. RESULTADOS EXPERIMENTALES

Para analizar el desempeño de cada uno de los métodos de selección de la raíz del *SAT* se realizaron experimentos sobre diferentes espacios métricos, para así poder estudiar el comportamiento de cada uno de ellos de manera más objetiva. Consideramos principalmente los costos de las búsquedas, sin embargo analizamos también qué sucede con los costos de construcción y por ende con los costos de selección de la raíz. Los espacios métricos considerados son los siguientes:

Espacio de Palabras: formado por un diccionario de 69.069 palabras del Inglés y la *distancia de edición*, que es el mínimo número de inserciones, supresiones y/o reemplazos de caracteres necesarios para hacer una palabra igual a otra. Esta distancia es útil para tratar con deletreo o reconocimiento de palabras, recuperación de texto, escritura y errores en reconocimiento óptico de caracteres (OCR).

Espacio de Vectores de Imágenes de la NASA: un conjunto de 40.700 vectores de características de 20 componentes, generados desde imágenes descargadas desde el sitio de la NASA [1]. La *distancia Euclídea* es utilizada como la función de distancia sobre este espacio.

Espacio de Histogramas de Imágenes: un conjunto de 112.682 vectores, formado por histogramas de color 8-D de 112 componentes para cada imagen [2]. La distancia debía ser de la familia de Minkowski, así se eligió la *distancia Euclídea* como la alternativa más simple y significativa.

Espacio de Documentos: formado por un conjunto de 1.265 documentos y la función de *distancia coseno*, muy utilizada en recuperación de la información [1]. Los documentos son obtenidos de la colección TREC- 3 [3] y son vistos como vectores (cada término de los documentos es una coordenada).

Espacio de Vectores de Coordenadas de Dimensión 15: es un espacio integrado por 100.000 vectores de dimensión 15, con componentes reales en el cubo unitario y *distancia Euclídea*. Para este espacio hemos generado 100.000 puntos al azar con distribución uniforme en el espacio $[0,1]^{15}$. Lo trataremos como a un espacio métrico, desechando la información que brindan las coordenadas. Esto nos permite controlar la dimensionalidad exacta con la que trabajaremos, lo que no es fácil en un espacio métrico general o si los vectores provienen desde una situación real.

Los experimentos consisten en ejecutar la construcción del *SAT* para los diversos algoritmos de selección de raíces y en cada espacio métrico considerado, y luego buscar en ellos. Para la construcción

[1] Disponible en: `http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html`

[2] Disponible en `http://www.dbs.informatik.uni-muenchen.de/~seidl/DATA/histo112.112682.gz`

[3] Disponible en: `http://trec.nist.gov`

del índice se usa el $90\,\%$ de los elementos del espacio métrico. El $10\,\%$ restante se utilizan como objetos de búsqueda. Realizamos varias ejecuciones de cada experimento sobre permutaciones distintas de la base de datos; por lo tanto, los resultados exhibidos corresponden a valores medios obtenidos.

Para el Espacio de Palabras en Inglés, con función de distancia discreta, los radios de búsqueda utilizados en los experimentos fueron de 1 a 4, que recuperan en promedio aproximadamente el 0.00003 %, el 0.00037 %, el 0.00326 % y el 0.01757 % respectivamente. Los radios de búsqueda utilizados en los espacios con distancia continua, son los siguientes:

a) Espacio de Vectores de Imágenes de la NASA: 0.605740, 0.78 y 1.009.

b) Espacio de Histogramas de Imágenes: 0.051768, 0.082514 y 0.131163.

c) Espacio de Documentos: 0.189441, 0.222466 y 0.600048.

d) Espacio de Vectores de Dimensión 15: 0.686576, 0.833130 y 1.019767.

Estos radios recuperan en promedio aproximadamente el 0.01 %, 0.1 %, 1 % elementos de la base de datos, respectivamente.

4.1. Comparación General de los Métodos

Comparamos todas las técnicas en cada espacio particular, para ver cuáles son las que mostraron menor número de evaluaciones de distancia en las consultas. Se agregan en las gráficas resultados de métodos de Mínimas y Máximas distancias que no fueron considerados como métodos alternativos en este trabajo por ser demasiado costosos, pero que sirven como referencia para los demás métodos.

4.1.1. Espacio de Palabras

De la comparación se observa que aquí, de los métodos considerados, los que tienen menor costo para las búsquedas son CSA y Sampling. Cabe destacar que el método de selección de la raíz Aleatorio no parece obtener un buen árbol, desde el punto de vista de las búsquedas, dado que casi todas las demás maneras de elegir la raíz, logran superarlo para los cuatro radios considerados. La Figura 7 muestra estos resultados.

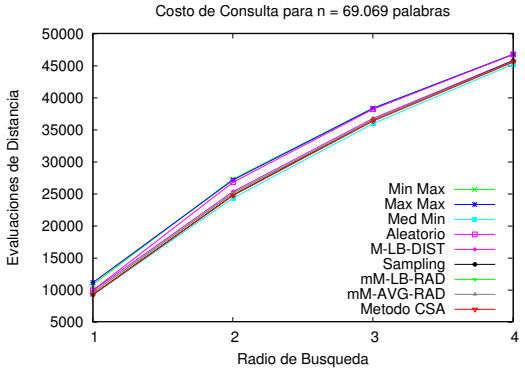


Figura 7: Comparación de los costos de búsqueda para todos los métodos sobre el Espacio de Palabras.

4.1.2. Espacio Vectores de Imágenes de la NASA

En este espacio el método que tuvo mejor desempeño en las búsquedas es el M-LB-DIST. Los métodos mM-LB-RAD y mM-AVG-RAD obtuvieron idénticos desempeños. Nuevamente aquí se ve que el método Aleatorio no produce el mejor árbol para las búsquedas. En la Figura 8 podemos apreciar este analisis.

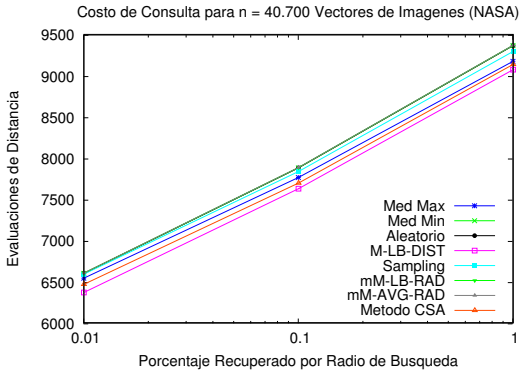


Figura 8: Comparación de los costos de búsqueda para todos los métodos sobre el Espacio de Vectores de Imágenes de la NASA.

4.1.3. Espacio de Histogramas de Imágenes

En este espacio métrico se comprobó que el método más económico en las búsquedas fue el Aleatorio y a continuación, con muy poca diferencia, la técnica CSA. Es llamativo que sea el método Aleatorio el que produce el menor costo en cantidad de evaluaciones en las búsquedas, siendo éste un algoritmo de costo $O(1)$ en cantidad de cálculos de distancia, y sumamente simple en su elaboración. Sin duda las características de este espacio hacen que resulte infructuoso usar nuestras técnicas, pero claramente no ha sido el caso en la mayoría de los espacios utilizados. La Figura 9 refleja estos hechos.

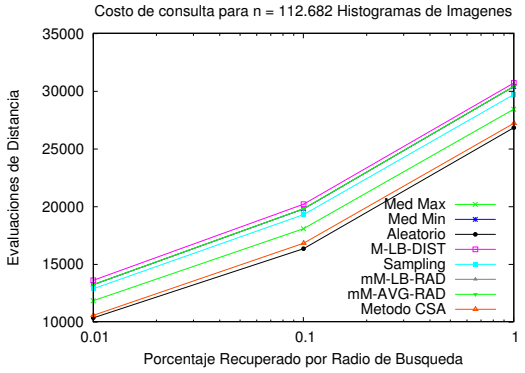


Figura 9: Comparación de los costos de búsqueda para todos los métodos sobre el Espacio de Histogramas de Imágenes.

4.1.4. Espacio de Documentos

En este espacio, el mejor método en las búsquedas para todos los radios y que además obtuvo un buen costo en la construcción es el método CSA, aunque las diferencias entre los distintos métodos no son demasiado significativas, ya que entre el mejor y el peor costo de búsqueda para cada radio la diferencia es cercana a 50 evaluaciones de distancia. En la Figura 10 es posible observar esta situación.

4.1.5. Espacio Vectores de Dimensión 15

En los experimentos se observa que existe una gran similitud entre los costos de las búsquedas de casi todos los métodos, no existiendo prácticamente diferenciación entre ellos. El único método que se destaca como menos costoso, por escasa diferencia de evaluaciones de los demás métodos, es CSA en todos los radios considerados. Sin embargo, aunque CSA haya mostrado mejor desempeño en las búsquedas, resultó ser el peor en cuanto a costos de construcción. Si no se está dispuesto a pagar tanto

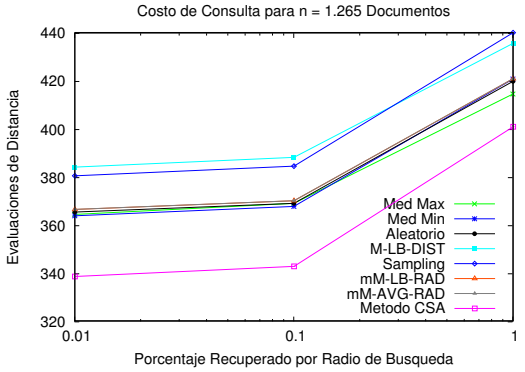


Figura 10: Comparación de los costos de búsqueda para todos los métodos sobre el Espacio de Documentos.

en la construcción de la estructura una muy buena alternativa sería el método de Sampling porque su desempeño en las búsquedas es muy similar y su costo de construcción es mucho menor. La Figura 11 permite observar gráficamente estos resultados.

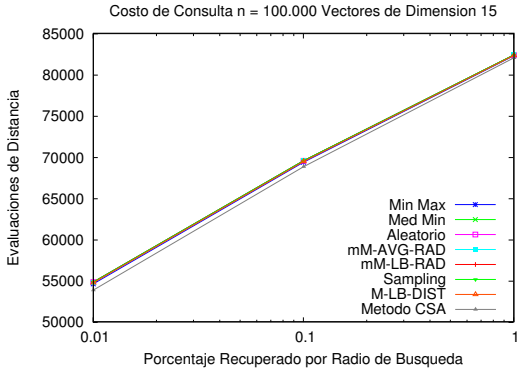


Figura 11: Comparación de los costos de búsqueda para todos los métodos sobre el Espacio de Vectores de Dimensión 15.

5. CONCLUSIONES

El *Árbol de Aproximación Espacial* (*SAT*) ha mostrado ser muy competitivo en espacios métricos de media a alta dimensionalidad o respondiendo a consultas con baja selectividad [12][9]. Un análisis detallado muestra que la selección de la raíz del árbol es uno de los procesos que se podrían optimizar, dado que éste es el que determina completamente al *SAT* y originalmente se realizaba al azar. Otros autores se han dedicado a este tema [13]. Sin embargo creemos que nuestro trabajo brinda más información debido a que se estudiaron y compararon más métodos, la mayoría de los cuales fueron diseñados o adaptados por nosotros para esta estructura, y se realizaron más experimentos.

Hemos conseguido construir un árbol más eficiente para las búsquedas, sólo pagando algunos cálculos de distancia adicionales respecto del *SAT* original y manteniendo la correctitud de la estructura. Demostramos experimentalmente que un mayor conocimiento sobre el espacio métrico particular permite mejorar la estructura, y que los algoritmos de selección de la raíz propuestos logran *mejorar* tanto costos de búsqueda como los costos de construcción del *SAT* original. Aunque nuestros resultados son aplicables principalmente al *SAT*, algunos podrían serlo en otras estructuras arbóreas.

De la misma manera que se mejoró la versión estática de *SAT* eligiendo la raíz de modo diferente al original, se podría intentar adaptar alguna de las técnicas a la versión dinámica del *SAT* [10, 11].

Hasta ahora el *SAT* trabaja en memoria principal; la posibilidad que la selección de una mejor raíz resulte en un árbol más balanceado, podría hacer que su almacenamiento en memoria secundaria fuera más eficiente.

REFERENCIAS

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

[3] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering*, 5(4):333–340, 1979.

[4] E. Chávez and G. Navarro. Measuring the dimensionality of general metric spaces. Technical Report TR/DCC-00-1, Dept. of Computer Science, University of Chile, 2000.

[5] E. Chávez and G. Navarro. Towards measuring the searching complexity of metric spaces. In *Proc. Mexican Computing Meeting*, volume II, pages 969–978, Aguascalientes, México, 2001. Sociedad Mexicana de Ciencias de la Computación.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

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

[8] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[9] G. Hjaltason and H. Samet. Improved search heuristics for the sa-tree. *Pattern Recognition Letters*, 24(15):2785–2795, 2003.

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

[11] G. Navarro and N. Reyes. Dynamic spatial approximation trees. *ACM Journal of Experimental Algorithmics (JEA)*, 2007. To appear.

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

[13] José Peñarrieta, Patricio Morriberón, and Ernesto Cuadros-Vargas. Distributed spatial approximation tree- sat$*$. In M. Marin and G. Acuña, editors, *Actas de la XXXII Conferencia Latinoamericana de Informatica (CD ROM)(CLEI2006)*, 2006.

[14] R.F. Santos Filho, A. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: The omni family of all-purpose access methods. In *Proc. of the 17th International Conference on Data Engineering*, pages 623–630, 2001.