UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL DE CONCEPCIÓN DEL URUGUAY

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN CON ORIENTACIÓN EN BASES DE DATOS

Búsquedas Métrico-Espaciales

Autor: Ing. Planas, Adrián Nicolás

Director: M.Cs. Pascal, Andrés Jorge

Concepción del Uruguay

ENTRE RÍOS - ARGENTINA

AGRADECIMIENTOS

Agradezco de corazón a mi familia por ser base de quien soy y por ser ejemplos de guía,
mi padre Sergio Andrés Planas (Responsabilidad y Orden);
mi madre Ester Grandolio (Comprensión e Intuición)
y a mi hermano Horacio Planas (Energía Cardinal y Desapego).

Por el gran Aporte Científico, Ayuda y Amistad agradezco a Andrés Jorge Pascal quien fué el director de este trabajo.

Siempre estaré agradecido a Anabella De Battista, por su eterna Paciencia y Diligencia, cuando necesité ayuda.

A mi pareja Daiana Belén Chiappella, quien fué de gran Compañía y me ofreció su Paz, en los momentos en los que los resultados no eran los esperados.

A mis Hermanos de SOA y GFU, acompañandome y contribuyendo con mejoras en mi Ser día a día.

Y a todas las personas que estuvieron presentes, directa e indirectamente,
en este proceso lleno de altos y bajos.

A pesar de lo ya dicho en relación a él, aprovecho este espacio para ofrecerle un especial agradecimiento eterno a mi Padre.
Gracias por estar siempre, por brindarme todo lo que un hijo espera y más...

ÍNDICE

Índice de figuras

Capítulo I

INTRODUCCIÓN

1.1. Introducción

Con el avance de las ciencias y tecnologías de la información, ha surgido la necesidad cada vez más importante de almacenar grandes cantidades de datos y de realizar búsquedas para obtener información para la toma de decisiones. Hasta hace unas décadas atrás, los tipos de datos mayormente utilizados eran datos relativamente simples, de tamaño fijo y sobre los cuales se realizaban búsqueda exactas, por rango o en el caso de cadenas de caracteres, por prefijo. Estos datos se almacenaban como registros organizados en campos que contenían valores completamente comparables (*datos estructurados*) y de tamaño fijo. Estas bases de datos estructurados se conocen en la actualidad con el nombre *bases de datos clásicas*.

Si bien las bases de datos clásicas aún son de interés, resultan insuficientes para afrontar todas las necesidades actuales de almacenamiento y recuperación de información. Hoy en día existen requerimientos importantes de almacenamiento y recuperación de datos semi-estructurados o no estructurados, tales como imágenes (huellas digitales, rostros, paisajes, pinturas, logos, etc), sonido (música, voces, etc), video (filmaciones de cámaras de seguridad, películas, etc), texto (secuencias de ADN, secuencias de proteínas,textos en lenguaje natural, etc), documentos XML, entre otros. Estos datos tienen la característica de que no pueden ser

organizados como datos estructurados, y aun cuando pudiera hacerse, las búsquedas exactas sobre ellos no tienen sentido, por lo que requieren otras formas de comparación. En muchos casos, debido al tamaño de estas bases de datos, no es práctico realizar un recorrido exhaustivo de las mismas para resolver una consulta. Para ello es necesario generar Métodos de Acceso (índices) que agilicen las búsquedas.

Otro aspecto importante a considerar es que las bases de datos clásicas mantienen los datos sin preocuparse por su ubicación. Desde hace años se han realizado avances importantes para superar esta limitación, dando origen a las bases de datos espaciales, estas permiten gestionar la distribución de objetos en el espacio.

Estas necesidades han dado origen a nuevos modelos de bases de datos, estos permiten ampliar el campo de aplicación de los modelos tradicionales, pero estos nuevos modelos representan sólo partes de la realidad y deben ser integrados tanto a los modelos clásicos como entre sí, para obtener un modelo más fiel y abarcativo de la misma.

Existen diversos casos particulares de este problema, tales como la búsqueda de documentos geográficos, que han sido estudiados previamente dando lugar a trabajos como el IR-Tree [42], este propone un índice para la búsqueda de documentos asociados a una región geográfica. En [73] se utilizan documentos web geoetiquetados con el fin de realizar búsquedas basadas tanto en la distribución espacial de los documentos como en la textualidad de los mismos. El trabajo descripto en [37] propone almacenar las firmas para las ROI (Regiones de Intersección) y para la consulta, obteniendo candidatos cuyas firmas son similares a la del elemento consultado. Luego verifican los candidatos e identifican las respuestas con técnicas finales de poda. En [82] capturan y representan características semánticas y espaciales de POIs (Puntos de Interés) relacionados al transporte, para medir la similitud semántica y la cercanía a consultas determinadas. Por otra parte, en el trabajo [45] se discute un enfoque para un marco de visualización y búsqueda de datos geoespaciales enriquecidos semánticamente, utilizando un subconjunto de lugares de DBpedia. Observando en detalle el trabajo realizado en [36] se nota el uso de datos

reales obtenidos de Twitter y Flickr para realizar experimentos de búsquedas espacio-textuales. En [10] se realiza un análisis lingüístico de documentos, extrayendo expresiones espaciales (es decir, expresiones que denotan localizaciones geográficas) y luego se realiza la búsqueda conjunta de las características lingüisticas y espaciales. El trabajo [46] tiene como objetivo predecir catástrofes climáticas en determinadas zonas. La información de la base de datos y las consultas poseen información espacial y textual de dichas zonas, y sus problemáticas climáticas. Todas estas técnicas están diseñadas específicamente para resolver consultas por similitud y espaciales donde los objetos de búsqueda son documentos. En este trabajo se propone generalizar el modelo para que incluya todo tipo de objetos que pueda ser comparado por similitud (imágenes, texto, cadenas, etc.). Si bien estas consultas se pueden resolver utilizando índices espaciales y métricos por separado para luego encontrar la respuesta final, es mucho más eficiente contar con métodos de acceso que estén diseñados específicamente para resolverlas.

En esta Tesis se presenta un modelo que integra los modelos Métrico y Espacial, para responder consultas compuestas por un aspecto que hace referencia a la similtud de los elementos, y otro que indica ubicación espacial.

El *Modelo Métrico* se basa en los Espacios Métricos, estos facilitan la resolución eficente de consultas por similitud sobre datos no estructurados o semi-estructurados. Este modelo caracteriza a la realidad como un universo de objetos U y una función métrica de distancia o similitud d la cual mide el grado de similitud (estrictamente, de diferencia) entre los objetos del universo. Las consultas por similitud usuales en espacios métricos son la *búsqueda por rango*, la cual toma como entrada un objeto de consulta y un radio y recupera todos aquellos objetos de la base de datos con distancia al objeto menor al radio, y la *búsqueda de los k vecinos más cercanos*, donde se devuelven los k elementos con menor distancia al objeto de consulta. Por ejemplo, algunas consultas por similitud son: *devolver todas las huellas digitales similares a una dada, con un radio de tolerancia r*, o *devolver las n huellas digitales más parecidas a una dada*.

El *Modelo Espacial* permite almacenar y recuperar elementos que se encuentran distribuidos en el espacio tomando como consulta, alguna especificación espacial. Mientras que las bases de datos tradicionales tratan al espacio como otro tipo de dato más, este tipo de base de datos incorpora la descripción espacial como otra dimensión. Este modelo soporta tres tipos de datos principales: punto, línea y región; y, a su vez, posee diferentes tipos de búsquedas posibles como la búsqueda por intersección, contención o la de adyacencia, entre otras. Por ejemplo, algunas de las búsquedas son: *devolver todos los elementos que se intersectan con un área a dada*, o *devolver todos los elementos que se encuentren más cerca de un punto p determinado*.

Si bien existen modelos aislados los cuales permiten abordar situaciones que las bases de datos clásicas no pueden manejar, en ciertos problemas es de interés combinar el aspecto métrico con el espacial, por ejemplo:

- Dada una base de datos geográfica con fotos asociadas a puntos de interés, *se desea encontrar los edificios similares a una foto de consulta, dentro de un radio dado*.

- Una empresa de seguridad desea rastrear una persona que llevó a acabo un delito, teniendo un registro de su rostro y acceso a cámaras de seguridad distribuidas en toda una ciudad. *Devolver todos los puntos donde fué identificado su rostro por las cámaras dentro del área donde fue llevado a cabo el delito, para comenzar con la investigación de su accionar.*

- Dada una base de datos de pinturas de arte y su información actualizada de su paradero, *se desea obtener las pinturas más parecidas a una determinada imágen, las cuales se encuentren dentro de un perímetro determinado*.

- Dada una base de datos de logos de marcas de negocios de un país y la ubicación de dichas empresas, *devolver 5 negocios más cercanos a un punto dado, los cuales posean logos más similares al consultado.*

En todos estos ejemplos es necesario buscar objetos similares a uno dado, pero dentro en

una determinada ubicación especificada. Tanto para las consultas métricas como para las espaciales existen índices que hacen más eficiente la recuperación de la información, pero ante situaciones como las anteriormente planteadas, la única solución hasta hace un tiempo era realizar la búsqueda de los objetos utilizando un índice métrico y otro espacial por separado, para luego calcular la intersección de los conjuntos resultantes. Este método es mejor en lugar de realizar un recorrido secuencial, pero dista de ser una buena solución. En consecuencia, se considera conveniente el desarrollo de índices los cuales permitan resolver con eficiencia este tipo de consultas combinadas.

En esta Tesis se estudia el modelo *Métrico-Espacial*, el cual permite la representación de situaciones en las que se requiere responder a consultas donde se consideran tanto el aspecto métrico como el espacial, y se presenta el diseño de estructuras que hacen más eficiente la resolución de estos tipos de búsquedas y además permiten realizar consultas métricas puras y consultas espaciales puras.

1.2. Aportes de la Tesis

Los principales aportes de esta Tesis se resumen en los siguientes puntos:

- Definición del *Modelo Métrico-Espacial*. Se define formalmente este nuevo modelo y se caracterizan los problemas resueltos por el mismo. También se describen los tipos de consultas asociados al modelo y se define la forma de cálculo de costos de las búsquedas.

- Diseño del índice métrico-espacial *MeTree*. Se presenta un índice métrico-espacial diseñado en base a una variante del índice métrico FHQT (que permite distancias continuas) y a la familia de índices espaciales R-Tree. En el aspecto espacial, este método permite indexar objetos geométricos de distintos tipos.

- Diseño del índice métrico-espacial *QuadFQTree*. Se presenta una estructura de datos basada en el índice métrico FHQT al cual se le añade el aspecto espacial mediante una

estructura similar al Quadtree para realizar consultas por similitud y espaciales combinadas. Desde el punto de vista espacial, este índice está orientado a la indexación de puntos.

- Diseño de los algoritmos de inserción y de consulta para estas nuevas estructuras. En el desarrollo de los algoritmos de inserción se definieron criterios para determinar el modo de construcción de las estructuras, para obtener una buena integración de los aspectos métrico y espacial

- Evaluación experimental de los nuevos métodos y comparación de su performance con la solución trivial, utilizando un fórmula de costos que integra ambos aspectos.

Los resultados logrados han sido publicados en actas de congresos y en revistas durante el proceso de elaboración de esta Tesis, quedando aún algunos avances sin publicar. El detalle de estas publicaciones es el siguiente:

- A. Planas, A. Pascal, N. Herrera, Consultas Metrico-Espaciales, En *Actas del 25vo Congreso Argentino de Ciencias de la Computación*, CACIC 2019, 14 al 18 de Octubre, Universidad Nacional de Río Cuarto, Córdoba, Argentina. [56]

- A. Planas, A. Pascal, N. Herrera, QuadFQTree - Un Indice Metrico-Espacial, En *Actas del VII Congreso Nacional de Ingeniería Informática – Sistemas de Información, CoNaIISI 2019*, 14 y 15 de Noviembre de 2019, Universidad Nacional de La Matanza, San Justo, Buenos Aires, Argentina. [57]

- Planas A., Pascal A., Herrera N. (2020) MeTree: A Metric Spatial Index. In: Pesado P., Arroyo M. (eds). Communications in Computer and Information Science, vol 1184. Springer, Cham.Switzerland.[58]

1.3. Organización del Informe

Este informe está organizado en seis capítulos: en los primeros tres capítulos se desarrolla el marco teórico y en los tres restantes se presentan los aportes de esta tesis, la evaluación experimental y las conclusiones.

En el Capítulo 2 se introduce el Modelo Métrico, incluyendo la definición, características y usos y limitaciones de los Espacios Métricos como base para resolver consultas por similitud de manera eficiente. También se presentan los métodos de acceso más importantes para las consultas por similitud y en especial se describe con mayor detalle el *FHQT*, que fue utilizado como base para el diseño de los dos nuevos índices métrico-espaciales presentados en esta tesis.

En el Capítulo 3 se brinda una introducción al Modelo Espacial, su definción, representación de datos, operaciones espaciales y tipos de consultas. Se describen métodos de acceso espaciales y en particular, el índice *R-Tree*, del cual se tomaron ideas para desarrollar el índice *MeTree*, y el *QuadTree*, utilizado como base para el diseño del índice *QuadFQTree*.

En el Capítulo 4 se realizan los aportes principales de esta Tesis: definición del Modelo Métrico-Espacial, caracterización de los problemas a resolver, los tipos de consultas básicos y presentación de las nuevas estructuras de acceso: el *MeTree* y el *QuadFQTree*, para las búsquedas por similitud y espacial con eficiencia.

Posteriormente, en el Capítulo 5 se realiza la evaluación experimental de ambos índices sobre dos bases de datos; la primera con cadenas como punto de interés, y la segunda con imágenes.

Finalmente, en el Capítulo 6 se presentan las conclusiones del informe y se proponen trabajos futuros.

Capítulo II

ESPACIOS MÉTRICOS

2.1. Introducción

Las bases de datos tradicionales están orientadas a trabajar con datos simples, usualmente números, fechas o cadenas, que se estructuran para formar construcciones de mayor complejidad. En el caso del modelo relacional, que sigue siendo el de mayor uso en la actualidad a pesar de las nuevas bases NoSQL, estos datos se componen en conjuntos de *n-uplas*. Estas *n-uplas* mantienen una estructura bien definida y permiten el uso de índices eficientes para realizar las consultas requeridas. Para que estos índices funcionen, se debe poder establecer un orden total entre los valores a indexar; orden que es utilizado para descartar elementos utilizando la propiedad transitiva sin necesidad de compararlos con el objeto de búsqueda. Además, las bases de datos tradicionales están orientadas al concepto de búsqueda exacta, búsqueda por rango de valores, o a lo sumo, en el caso de las cadenas, por prefijo. En estos tipos de consulta se pueden utilizar índices tales como los de la familia del *B-Tree*, para disminuir considerablemente el costo (en tiempo) de la búsqueda. Otros tipos de consulta como la búsqueda por subcadena o patrón, no pueden ser indexadas de esta manera por lo cual es necesario recorrer el conjunto de datos completo.

En las últimas dos décadas han surgido otras clases de objetos no contemplados en estos

modelos, como los nombrados anteriormente: imágenes (con todas sus variantes), sonido, video, texto, documentos XML, datos geométricos, secuencias de ADN, series temporales, etc. Hasta hace unos años, estos objetos solo se podían almacenar, pero no existían formas realizar consultas que los involucren, ya que además de no tener una estructura fija, no tiene sentido compararlos por igualdad y no se puede establecer un orden total entre ellos. Por ejemplo, en el caso de fotos de pinturas, no sirve comparar pixel a pixel dos de estas imágenes para saber si corresponden al mismo objeto. Sus resoluciones pueden ser distintas, el ángulo con que fueron tomadas las fotos seguramente es diferente y tampoco tendrían el mismo nivel de luminosidad. Algunas consultas usuales de interés son sobre estas clases de objetos son:

- Encontrar todos los objetos que sean similares (con cierto nivel de tolerancia) a uno dado.
- Encontrar los k objetos con mayor similitud a uno dado.

Este tipo de búsqueda se denomina *Búsqueda por Similitud* y posee un campo de aplicación cada vez más amplio. Algunos ejemplos de aplicación son:

Consultas por contenido Cuando se requiere mantener datos compuestos no estructurados, la solución trivial para poder consultarlos es asociarles un nombre, descripción o identificador y realizar la consulta sobre estos campos, sin embargo esta solución no es aplicable cuando la consulta es el objeto mismo, como por ejemplo, una huella digital. En estos casos surge la necesidad de recuperar la información *por contenido*. Ejemplos de consultas por contenido son:

- Búsqueda de personas a través de una fotografía de sus rostros.
- Búsqueda de huellas digitales para identificar sospechosos de un delito.
- Búsqueda de pinturas para identificar y recuperar información sobre las mismas.
- Búsqueda de logotipos, para asegurar que no haya otro similar ya registrado.
- Búsqueda de canciones o fragmentos de canciones similares a una dada.

- Búsqueda de videos, en base a un fragmento o imagen.

Recuperación de texto La recuperación de texto constituye un problema de características similares a los planteados anteriormente. Esto es debido a que los objetos textuales no están estructurados como para facilitar la ejecución de consultas que son de interés, como por ejemplo, búsquedas por conceptos semánticos. Este problema ha sido estudiado desde hace ya largo tiempo y uno de los métodos propuestos que resuelve este problema consiste en la definición de una función de distancia específica para comparar texto que es utilizada para realizar consultas por similitud.

Reconocimiento de patrones y aproximación de funciones En ciertas situaciones es necesario contar con un método automático de clasificación de nuevos elementos en base a un conjunto ya clasificado. Esta es una forma posible de reconocimiento de patrones. Para ello se cuenta con una base de datos de objetos asociados a sus clases y cuando se presenta un nuevo objeto se utiliza un *aproximador* que determina a cual clase pertenece. Existen distintas estrategias para construir este aproximador, algunas basadas en redes neuronales y funciones difusas, pero otra opción válida y que además no requiere entrenamiento, es determinar el vecino más cercano del nuevo elemento (con el que posee mayor similitud) y asignarle la misma clase.

Biología computacional Las secuencias de proteínas y el ADN constituyen el objeto de estudio de la biología molecular. Estas secuencias usualmente se modelan como texto y se presenta el problema de encontrar una subsecuencia que sea similar a una secuencia dada, ya que casi siempre existen diferencias menores en los fragmentos que describen funciones similares correspondientes a distintas especies o inclusive a distintos individuos de la misma especie. Como en los casos anteriores, una solución posible a este problema es el empleo de espacios métricos como base para realizar consultas por similitud.

Todas estas situaciones se pueden representar a través de un formalismo matemático deno-

minado *Espacio Métrico*, que tiene como característica la existencia de un universo de objetos y de una función, llamada *función de distancia*, que calcula el grado de similitud (o diferencia) entre los objetos del universo.

Formalmente, un *Espacio Métrico* es un par (U, d) donde U es el universo de objetos válidos del espacio y $d : U \times U \longrightarrow R^+$ es una función métrica definida entre los elementos de U que mide la similitud (estrictamente hablando, su diferencia) entre los mismos. Cuanto menor es el valor de la función para dos objetos dados, más cercanos o similares son éstos.

Para que el espacio sea métrico, la función de distancia d debe poseer las propiedades características de una métrica:

- $\forall x, y \in U,\ d(x, y) \geq 0$ (positividad);
- $\forall x, y \in U,\ d(x, y) = d(y, x)$ (simetría);
- $\forall x \in U,\ d(x, x) = 0$ (reflexividad) y
- $\forall x, y, z \in U,\ d(x, y) \leq d(x, z) + d(z, y)$ (desigualdad triangular).

Bajo esta definición, se considera que dos elementos son iguales si su distancia es 0, por lo tanto es natural que las distancias sean positivas ya que una distancia negativa indicaría que hay dos elementos con mayor similitud que dos elementos iguales. La reflexividad indica que el mayor nivel de similitud posible se alcanza entre dos elementos iguales. La simetría expresa que la diferencia o similitud es única para todo par de elementos, y por último, la desigualdad triangular determina que la diferencia directa entre dos elementos siempre será menor o igual que la diferencia indirecta entre los mismos, es decir, pasando por un tercero. Es esta última propiedad una de las más importantes durante la búsqueda en espacios métricos dado que permite descartar elementos sin necesidad de compararlos con la consulta, como veremos más adelante.

Si d no satisface la positividad estricta, se habla de un espacio *pseudo métrico* y se puede adaptar para que mantenga las propiedades relevantes para este tipo de aplicaciones. En

caso de que la función de distancia no cumpla con la simetría (*espacio cuasi-métrico*), se puede definir una nueva función d', que sea simétrica, haciendo: $d'(x, y) = d(x, y) + d(y, x)$. También se permite relajar la desigualdad triangular haciendo que solo cumpla $d(x, y) \leq \alpha d(x, z) + \beta d(z, y) + \gamma$. En estos casos es posible seguir usando los mismos algoritmos de espacios métricos previo escalamiento.

2.2. Consultas por Similitud

Existen tres tipos genéricos de consultas que resultan de interés en espacios métricos [25]. Sea $X \subseteq U$ un subconjunto del universo al cual llamaremos *base de datos* y $n = |X|$ su cardinalidad, se define:

- *Búsqueda por rango* $(q, r)_d$: donde $q \in U$ es el objeto de consulta o *query* y r es un número real que representa el radio de tolerancia, devuelve el conjunto de elementos que se encuentran a distancia de q menor o a lo sumo igual a r. Note que esta consulta podría retornar un conjunto vacío. Formalmente:

$$(q, r)_d = \{x \in X : d(q, x) \leq r\} \tag{2.1}$$

- *Búsqueda del vecino más cercano* $NN(q)$: donde $q \in U$ es el objeto de consulta, devuelve el elemento más cercano a q. En caso de que haya más de un elemento a la misma distancia, podría retornar un conjunto de elementos. Si la base de datos no es vacía, esta consulta siempre tiene al menos un objeto resultante. Formalmente:

$$NN(q) = \{x \in X : \forall y \in X, d(q, x) \leq d(q, y)\} \tag{2.2}$$

- *Búsqueda de los* k *vecinos más cercanos* $NN_k(q)$: esta consulta es similar a la anterior, pero generalizada a los primeros k elementos más cercanos. Formalmente, la consulta

devuelve el conjunto $A \subseteq X$ tal que:

$$|A| = k \wedge \forall x \in A, y \in (X - A) : d(q, x) \leq d(q, y) \tag{2.3}$$

Para medir la eficiencia en tiempo de las consultas, hay tres factores a tener en cuenta. El primero en importancia suele ser el costo de cálculo de la función de distancia, ya que las funciones que calculan similitud entre objetos requieren muchos recursos para su procesamiento. El segundo es la cantidad de accesos a almacenamiento secundario (tiempo de E/S), que puede ser importante debido a que estos accesos son significativamente más lentos que el acceso a memoria principal. Por último, hay que considerar también el tiempo extra de CPU, es decir, el resto de las instrucciones del algoritmo de búsqueda. Por lo tanto, podemos calcular el costo en tiempo T de la siguiente manera:

T = # evaluaciones de d $\times$ complejidad (d) + tiempo de E/S + tiempo extra de CPU

El tiempo extra de CPU se puede descartar cuando se mide la eficiencia de la búsqueda porque normalmente es de un orden computacional mucho menor que los otros dos factores. Cuando los algoritmos de búsqueda realizan su procesamiento en memoria principal, el tiempo de E/S es nulo, por lo cual en estas situaciones para analizar la eficiencia de los algoritmos de consulta, solo se tiene en cuenta la cantidad de evaluaciones de la función de distancia. Este es el caso de los índices propuestos en este trabajo, por lo cual todas las mediciones sobre los índices propuestos están calculadas, en cuanto al especto métrico, en cantidad de evaluaciones de d.

Dado el tamaño creciente de las bases de datos actuales y teniendo en cuenta el alto costo de cálculo de la función de distancia, no es operativo realizar una búsqueda exhaustiva sobre todos los elementos del conjunto de datos. Para ello se deben utilizar estructuras auxiliares (*índices o métodos de acceso*) que descarten elementos sin necesidad de compararlos con el objeto de consulta.

2.3. Funciones de Distancia

Las medidas de distancia en los espacios métricos representan el grado de proximidad entre dos elementos en un dominio dado. Cuando se utilizan funciones de distancia como medida de similitud, los objetos a comparar no son datos simples, sino estructuras con cierta complejidad. Por ejemplo, secuencias de caracteres, vectores n-dimensionales, matrices, árboles o conjuntos. Los vectores n-dimensionales o vectores de características, contienen valores de propiedades, atributos o características de los objetos que representan. Una imagen, por ejemplo, se puede modelar a través de un vector que constituye el histograma de colores de la imagen; si la imagen contiene 16 colores, se puede obtener un vector con 16 elementos cuyos valores definen la cantidad de pixels de cada color contenidos en la imagen. En este caso el espacio métrico es también un *espacio vectorial* y existen varias funciones de distancia disponibles aplicables a este modelo. Si bien las medidas de distancia se definen específicamente para resolver cada problema particular, existen algunas funciones y familias de funciones conocidas que se pueden utilizar en varios dominios. Dependiendo del conjunto de valores que retornan, las funciones de distancia se clasifican en *discretas*, que retornan un conjunto predefinido y normalmente pequeño de valores, y *continuas*, que devuelven un conjunto potencialmente infinito o muy grande de valores. Se presentan algunas funciones importantes a continuación.

2.3.1. Distancias de Minkowski

La familia de funciones más conocida y utilizada sobre espacios vectoriales, es la familia L_p de *Distancias de Minkowski* [76], definida sobre vectores de números reales como:

$$L_p[(x_1, ..., x_n), (y_1, ..., y_n)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

Algunos casos particulares de suma utilidad de esta familia son:

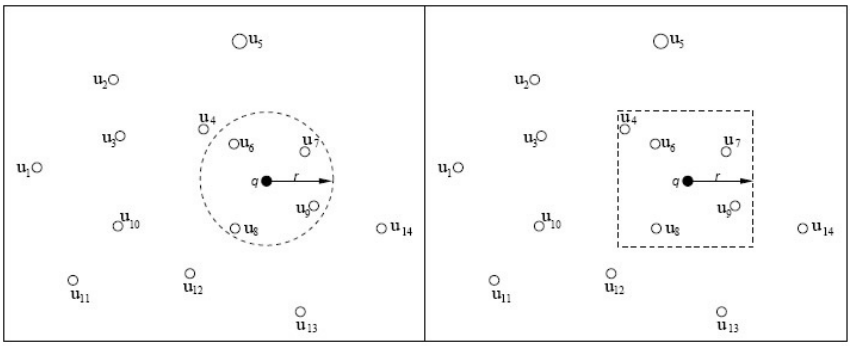

Figura 2.1: *Ejemplos de búsquedas por rango* $(q, r)_d$*, con d=*L_2 *(izquierda) y d =* L_∞ *(derecha)*

- L_1, conocida como *Distancia de Manhattan*.

$$L_1[(x_1, ..., x_n), (y_1, ..., y_n)] = \sum_{i=1}^{n} |x_i - y_i|$$

- L_2, conocida como *Distancia Euclidiana*

$$L_2[(x_1, ..., x_n), (y_1, ..., y_n)] = \sqrt{(\sum_{i=1}^{n} |x_i - y_i|^2)}$$

- L_∞, conocida como *Distancia Máxima*, que corresponde al límite de p tendiendo a infinito

$$L_\infty[(x_1, ..., x_n), (y_1, ..., y_n)] = \max_{i=1}^{n}(|x_i - y_i|)$$

La Figura 2.1 muestra un ejemplo de búsqueda por rango $(q, r)_d$ sobre un conjunto de puntos en R^2, utilizando como función de distancia L_2 y L_∞. La línea punteada identifica aquellos puntos que están a distancia r de q. Como consecuencia, todos los puntos que caen dentro de estas líneas conforman la respuesta a la búsqueda. La distancia L_2, se corresponde con nuestra noción de distancia espacial (en un espacio euclidiano). Las búsquedas que utilizan L_∞ se corresponden con la búsqueda por rango clásica, donde el rango es un hiper-rectángulo k-dimensional.

2.3.2. Distancia de Forma Cuadrática

En muchas aplicaciones que utilizan vectores de datos, los valores de las distintas dimensiones no son totalmente independientes. En estos casos las distancias de Minskowski no son aplicables ya que no tienen en cuenta la existencia de relaciones entre dimensiones. La *Distancia de Forma Cuadrática* [43] utiliza una matriz de pesos que representa el grado de relación entre dos componentes x e y de los vectores X e Y respectivamente. Sean x e y dos vectores de n dimensiones, la siguiente expresión representa la *Distancia de Forma Cuadrática* generalizada d_M.

$$d_M(x, y) = \sqrt{((x - y)^T.M.(x - y))}$$

donde el superíndice T denota la transposición de vectores y

$$M = [m_{i,j}]$$

es la matriz de pesos. Si M es la matriz identidad, esta distancia se transforma en la distancia euclidiana.

2.3.3. Distancia de Hamming

Esta medida se utiliza para determinar la similitud de dos secuencias de símbolos de igual tamaño, y es igual a la cantidad de posiciones en las cuales los símbolos son diferentes. En otra palabras, mide la cantidad mínima de sustituciones que hay que realizar en una de las secuencias para transformarla en la otra. Una generalización de la Distancia de Hamming [68], es la Distancia de Levenshtein, que se describe a continuación.

2.3.4. Distancia de Levenshtein

El grado de similitud entre dos cadenas de símbolos w y v, se puede medir efectivamente mediante la *Distancia de Edición de Cadenas* o *Distancia de Levenshtein* [9], definida como la menor cantidad de operaciones de edición necesarias para transformar la cadena w en v (o viceversa). Las operaciones consideradas son:

- Inserción(w, s, i): agrega el símbolo s en la posición i de la cadena w.
- Eliminación(w, i): borra el elemento que se encuentra en la posición i de la cadena w.
- Reemplazo(w, s, i): reemplaza el elemento que se encuentra en la posición i de la cadena w, por el símbolo s

Esta función se puede generalizar asignando pesos (números reales positivos) a cada tipo de operación. Por ejemplo, es común que el costo de la operación de reemplazo sea mayor que el de las operaciones de inserción y eliminación, ya que el reemplazo se puede simular mediante dos de estas. Otra variante es la asignación de pesos a operaciones sobre símbolos específicos, por ejemplo, la eliminación de la letra *h*, puede tener menor peso que la eliminación de la letra *a* cuando se trata del lenguaje natural español o castellano. Lo mismo sucede con el reemplazo de la letra *c* por la letra *s*, o *v* por *b*, ya que se pronuncian con fonética similar. En cualquier caso, para que se mantenga el espacio métrico, se debe asegurar que estas variantes preserven las propiedades de toda métrica.

La *Distancia de Damerau-Levenshtein* es una variante de la distancia de edición, que agrega al conjunto de operaciones básicas anteriormente nombradas la transposición, es decir, el intercambio posicional de dos símbolos adyacentes de la cadena.

2.3.5. Distancia de Edición de Árboles

La distancia de edición anteriormente nombrada se utiliza sobre secuencias o listas, y se puede adaptar para estructuras no-lineales como los árboles. La *Distancia de Edición de Árboles* se

define como el mínimo costo necesario para convertir una estructura de árbol en otra, utilizando un conjunto de operaciones básicas predefinidas, tales como la inserción o eliminación de un nodo. Ya que los documentos XML se pueden ver como estructuras de árboles etiquetados, esta función es apropiada para medir la similitud estructural entre documentos XML.

2.3.6. Coeficiente de Jaccard

El *Coeficiente de Jaccard* [38] es una medida sencilla del grado de similitud entre dos conjuntos. Dados dos conjunto A y B, se define como:

$$d(A, B) = 1 - \frac{(A \cap B)}{(A \cup B)}$$

Esta función expresa la razón entre los elementos que tienen en común los conjuntos, sobre el total de elementos que forman parte de los mismos.

2.4. Métodos de Acceso

Las búsquedas por similitud se pueden resolver recorriendo y comparando el objeto consultado con cada uno de los elementos de la base de datos. Sin embargo, en grandes colecciones de objetos este procedimiento no es adecuado debido al alto costo (en tiempo) que tendrá la consulta, por lo cual, es necesario hacer uso de estructuras auxiliares para descartar elementos sin tener que recorrer la base completa. Estas estructuras auxiliares se denominan *índices* o también *Métodos de Acceso*.

En el caso particular de los Espacios Vectoriales, en principio existen varias estructuras que facilitan las búsquedas, tales como el *KD-Tree* [6, 7], *R-Tree* [39], *Quad-Tree* [66] o el *X-Tree* [8]. Estos métodos utilizan las dimensiones de los vectores como coordenadas para clasificar y agrupar los objetos en el espacio. Sin embargo, su eficiencia se degrada significativamente cuando crece la cantidad de dimensiones del espacio vectorial: el costo de las búsquedas por

similitud depende exponencialmente de la dimensionalidad del espacio métrico [27]. Estrictamente hablando, el factor de mayor influencia en la performance de la consulta no es la cantidad de dimensiones, sino la *Dimensionalidad Intrínseca* del espacio, como se verá más adelante.

Por esta razón, se han diseñado índices sobre espacios métricos generales, que superan la eficiencia de las estructuras anterioremente nombradas cuando la dimensionalidad intrínseca del espacio crece. Además, los métodos orientados a espacios vectoriales no permiten indexar objetos que no son representados a través de vectores. En [32], [73] se muestra que un índice para espacios métricos como el *M-tree* puede ser más eficiente que una de las estructura más utilizadas en espacios vectoriales de baja dimensionalidad, el *R*-Tree*. Existen muchas variantes del *M-Tree*, tales como *Slim-Tree* [70], *DBM-Tree* [75] y *CM-Tree* [2]. El *D-index* [70] es una estructura que utiliza una función hash para mapear objetos en buckets. *LC* [33] utiliza una lista de clusters que mejora la eficiencia en la búsqueda a costa de hacer menos eficiente su construcción. *BP* [1] es un árbol no-balanceado pensado para espacios métricos de alta dimensionalidad.

2.5. Modelo Unificado

Existen varias estrategias de diseño de índices métricos y cada una de ellas posee sus particularidades, sin embargo, en un mayor nivel de abstracción se puede decir que todos los métodos de acceso constan de dos etapas: preprocesamiento de la base de datos y procesamiento de las consultas. En la primer etapa se realiza la construcción del índice mediante el recorrido de la base de datos y la extracción de información sobre sus elementos y en la segunda parte del proceso se responde las consultas utilizando dicho índice para descartar objetos sin necesidad de compararlos con el objeto que se busca. Por otro lado, se debe notar que todos los índices se construyen particionando el conjunto de objetos de la base de datos X en clases de equivalencia X_i. Durante una búsqueda se utilizan una o más propiedades de los espacios métricos para descartar clases sin necesidad de comparar todos sus elementos con la consulta y así hallar el

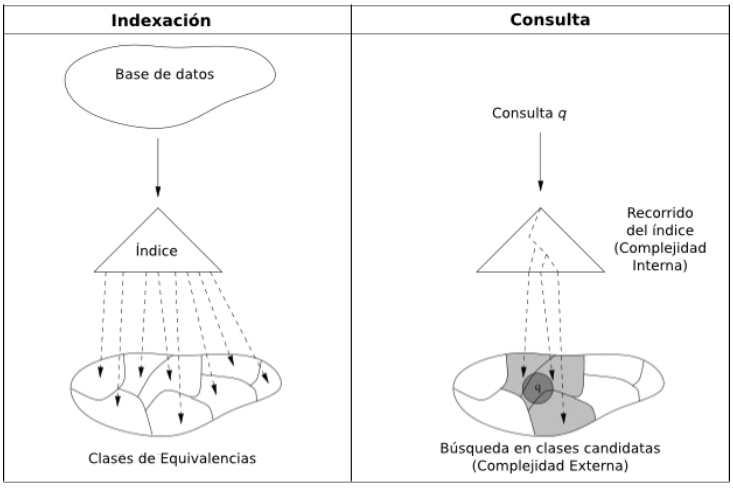

Figura 2.2: *Modelo Unificado de los métodos de acceso*

conjunto de los objetos candidatos, es decir, los objetos que podrían formar parte de la respuesta [25]. Al final de esta segunda etapa, se compara la consulta con cada uno de los candidatos y se obtiene el conjunto respuesta. Es decir que la etapa de búsqueda consta a su vez, de dos partes:

1. Recorrer el índice y utilizar propiedades para descartar clases de tal manera de obtener el conjunto C de candidatos. El costo de este proceso se denomina *complejidad interna*.

2. Recorrer el conjunto C de candidatos y comparar sus elementos con la consulta para hallar los elementos que constituirán la respuesta. El costo de este proceso se denomina *complejidad externa*.

Esta abstracción constituye un *modelo unificado* [25] del proceso realizado por los métodos de acceso en espacio métricos (Figura 2.2).

Los índices métricos se pueden agrupar en dos clases de acuerdo a la estrategia de diseño que utilizan: *Métodos basados en Particiones Compactas* [24], [33], [34], [70] y *Métodos basados en Pivotes* [15], [71], [47], [79]. A continuación se presentan estas estrategias y se describen algunos índices representativos de cada clase. Posteriormente se describe en detalle el índice FHQT, utilizado en la construcción de los nuevos índices métrico-espaciales aquí presentados.

2.5.1. Métodos de acceso basados en Particiones Compactas

Consiste en seleccionar un conjunto de puntos llamados *centros* y utilizar los *Diagramas de Voronoi* para dividir el espacio en particiones asociadas a dichos puntos. Los *Diagramas de Voronoi* (Figura 2.3) constituyen una importante estructura en Geometría Computacional, que facilita el tratamiento de problemas relacionados al problema de hallar el *punto más cercano* a uno determinado. Este diagrama se define sobre un conjunto de objetos asociando un área de influencia o alcance de cada uno de ellos. De esta manera, todos los puntos que se encuentran más cerca de un centro que de los demás, estarán ubicados en la partición correspondiente a dicho centro.

A pesar de que esta noción fue desarrollada para espacios vectoriales, donde los vectores representan información sobre coordenadas, su generalización a espacios métricos es posible.

En los métodos de acceso basados en este concepto se definen clases de equivalencia en función de la proximidad de los elementos de la base de datos al conjunto de centros. Sea $C = \{c_1, c_2, ..., c_k\}$ el conjunto de centros, la relación de equivalencia (que denotaremos $\sim_C$) se define de la siguiente manera:

$$x \sim_C y \Leftrightarrow NN_C(x) = NN_C(y)$$

$$\text{donde } NN_C(z) = \{c \in C \ / \ \forall h \in C : d(z, c) \leq d(z, h)\}$$

La partición que esta relación genera en el espacio se conoce con el nombre de *partición compacta* y resulta de dividir el espacio asociando a cada c_i, la clase formada por el conjunto de puntos que tiene a c_i como su centro más cercano. Dado un elemento cualquiera x de la base de datos, la clase a la que pertenece x se denota $[x]$. Además, se asume que la partición usa un subconjunto de la base de datos X como el conjunto de centros, aunque estrictamente hablando, estos elementos podrían ser externos.

Existen distintos criterios para determinar si una clase completa se puede descartar sin ne-

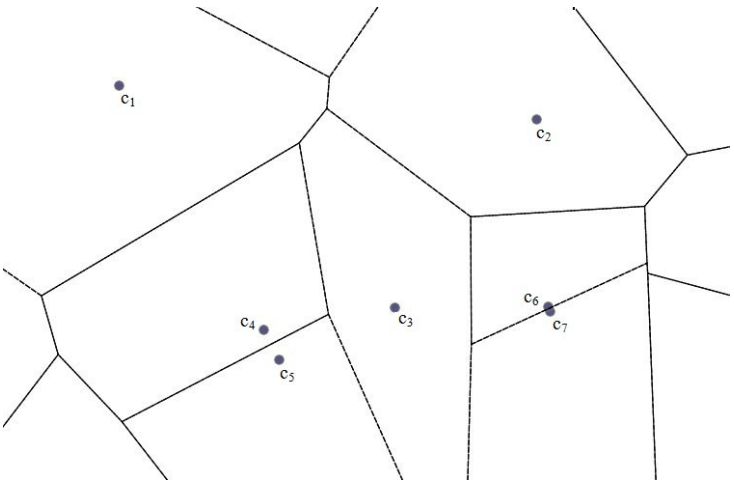

Figura 2.3: *Diagrama de Voronoi (líneas punteadas) para un conjunto de puntos en R^2 con distancia L_2*

cesidad de comparar sus elementos con la consulta. Los más utilizados son:

- **Criterio del hiperplano:** consiste en determinar si la esfera de la consulta intersecta o no, a la partición que se quiere descartar. Obviamente, la partición en que se encuentra la consulta q, no se puede descartar. Si c_j es el centro de la clase $[q]$, entonces la esfera con centro q no intersecta $[c_i]$ si $d(q, c_j) + r < d(q, c_i) - r$ (Figura 2.4). La expresión $d(q, c_j) + r$ representa el punto más lejano al centro c_j, que pertenece a la esfera de la consulta, mientras que $d(q, c_i) - r$ es la distancia mas cercana de c_i a la esfera. En particiones adyacentes, estos valores son iguales en el límite de las mismas.

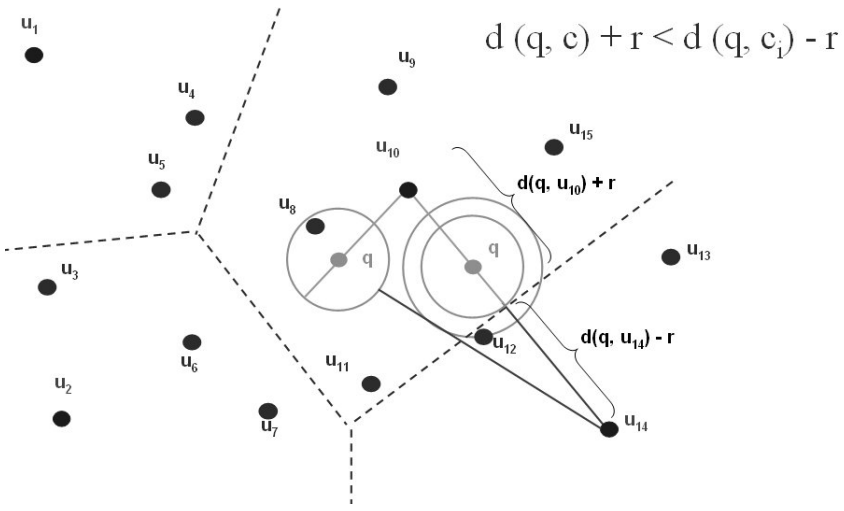

Figura 2.4: *Particiones Compactas - Criterio del hiperplano*

- **Criterio del radio de cobertura:** otra manera de determinar si una clase $[c_i]$ se puede descartar es considerar la esfera centrada en c_i que contiene a todos los elementos de X pertenecientes a dicha clase, y determinar si se intersecta con la esfera de la consulta. Definimos el radio de cobertura de un centro c de la siguiente manera:

$$cr(c) = \max_{x \in [c]} d(c, x)$$

La clase $[c_i]$ se puede descartar sin comparar sus elementos si $d(q, c_i) - r > cr(c_i)$ (Figura 2.5). El valor de $d(q, c_i) - r$ es la mínima distancia de c_i a la esfera de la consulta, mientras que el radio de cobertura define el área en la cual están los elementos de la clase.

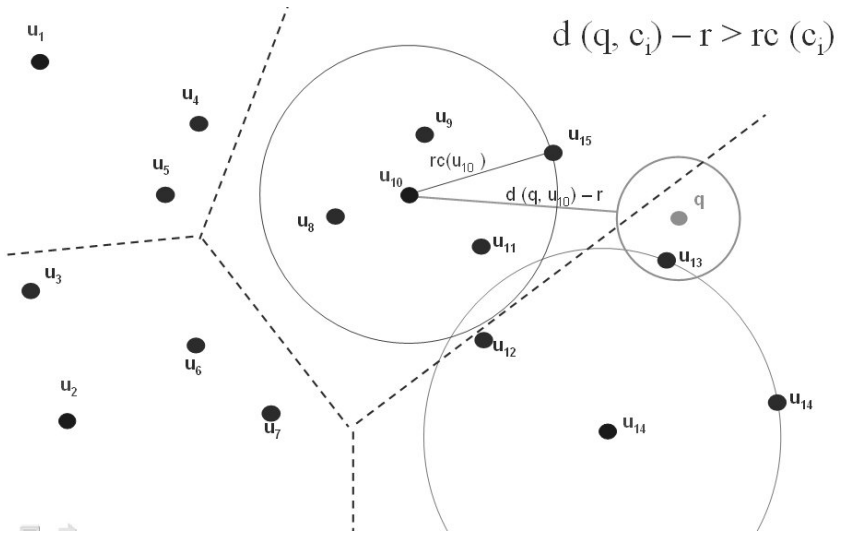

Figura 2.5: *Particiones Compactas - Criterio del radio de cobertura*

Algunos de los índices más importantes basados en particiones compactas son:

- **Bisector Tree (BST):** Es un árbol binario donde cada nodo contiene dos centros. Los elementos de la base de datos se asocian al centro más cercano y este proceso se repite en cada subárbol hasta alcanzar las hojas. Para cada centro c_i se determina y almacena su radio de cobertura y ante una consulta (q, r) se descarta aquellos subárboles donde se cumpla que $d(q, c_i) - rc(c_i) > r$ [40, 53].

- **Generalized-Hyperplane Tree (GHT):** El GHT se construye de la misma manera que el BST, pero no utiliza el radio de cobertura como criterio de descarte, sino el criterio del hiperplano. Sean c_1 y c_2 los centros correspondiente a un nodo y (q, r) una consulta, se puede descartar el subárbol de c_1 si $d(q, c_1) > d(q, c_2) + 2r$ y viceversa [72]. GANT [12] es una generalización del GHT. Divide el espacio utilizando particiones de Voronoi. Existe una versión dinámica de este último, llamada EGANT [13].

- **Geometric Near-neighbor Access Tree (GNAT):** El GNAT es un árbol m-ario, es decir que contiene m centros en cada nodo. Junto a los centros se almacena una tabla de tamaño $O(m^2)$ que contiene la distancia de cada centro c_i hacia el elemento más cercano y más

lejano de cada clase $[c_j]$. A este intervalo lo llamaremos $p(c_i, c_j)$. Luego, sea (q, r) una consulta por rango, se descarta el subárbol correspondiente al centro c_j si para algún otro centro c_i se cumple que la intersección del intervalo $[d(c_i, q) - r, d(c_i, q) + r]$ con el intervalo $p(c_i, c_j)$ es vacía [12].

- **Spatial Approximation Tree (SAT):** El SAT o sa-tree utiliza un enfoque distinto a los anteriores. Se basa en la *Triangulación de Delaunay* para definir un grafo donde los nodos son elementos de la base de datos y las aristas unen cada objeto con un conjunto de vecinos. Durante la búsqueda del vecino más cercano, se parte de un nodo arbitrario y se calcula la distancia de la consulta a dicho objeto y a sus vecinos. Si la consulta se encuentra más cerca del nodo considerado, éste es el resultado buscado. En caso contrario, se realiza un movimiento al vecino de menor distancia a la consulta y se repite el proceso [49, 55]. Existen extensiones dinámicas de SAT, y también para memoria secundaria [14, 50]. En [22] se presenta una mejora al algoritmo de búsqueda de SAT.

2.5.2. Métodos de acceso basados en Pivotes

Estos métodos utilizan una estrategia distinta a la de las particiones compactas, basada en la selección de un conjunto de objetos del universo denominados *pivotes* que se utilizan para definir las clases de equivalencia. Así, una clase estará formada por todos los objetos de la base de datos con igual distancia a cada uno de los pivotes. Formalmente, sea $P = p_1, p_2, ..., p_k$ el conjunto de pivotes y x e y dos elementos de la base de datos, la relación de equivalencia se define como:

$$x \sim_P y \Leftrightarrow d(x, p_i) = d(y, p_i), \forall i = 1 \ldots k$$

Gráficamente, cada clase de equivalencia quedará definida por la intersección de varias capas de esferas o anillos centrados en los pivotes p_i. En la Figura 2.6 se puede ver una relación

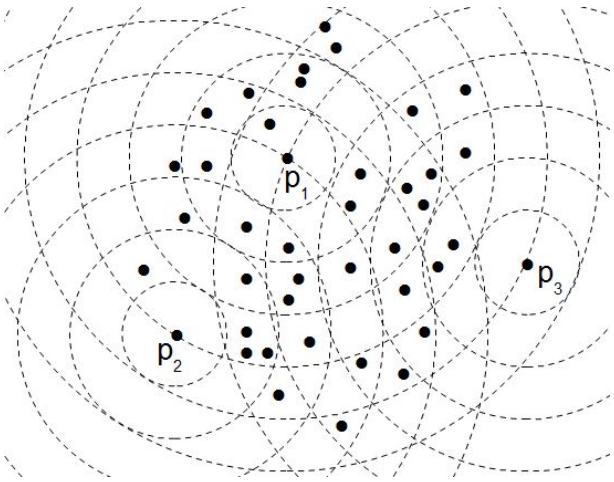

Figura 2.6: *Ejemplo de clases de equivalencia inducida por la intersección de anillos centrados en los pivotes p_1, p_2 y p_3*

.

de equivalencia inducida por los pivotes p_1, p_2 y p_3.

Durante la indexación, estos algoritmos asocian a cada elemento a de la base de datos, el vector $\phi(a) = (d(a, p_1), d(a, p_2), ..., d(a, p_k))$, denominado *firma* de a. Posteriormente, durante la búsqueda, se utiliza la desigualdad triangular junto con la firma de cada elemento para eliminar clases de equivalencia sin medir su distancia a q, es decir:

Dada una consulta por rango $(q, r)_d$, en primer lugar se computa la firma de la query q,

$$\phi(q) = (d(q, p_1), d(q, p_2), ..., d(q, p_k))$$

Luego, suponiendo que existe un pivote p_i tal que para algún elemento a de la base de datos se cumple:

$$r < d(q, p_i) - d(a, p_i)$$

Por la desigualdad triangular se sabe que $d(q, p_i) \leq d(q, a) + d(a, p_i)$, por lo tanto reemplazando en la fórmula anterior:

$$r < d(q, p_i) - d(a, p_i) < d(q, a) + d(a, p_i) - d(a, p_i) = d(q, a)$$

En consecuencia, el elemento a puede eliminarse sin necesidad de evaluar su distancia real a la query q. Análogamente se demuestra que si $d(a, p_i) - d(q, p_i) > r$ entonces $d(a, q) > r$. Resumiendo, durante la búsqueda se pueden descartar todos aquellos elementos a tales que para algún pivote p_i se cumple que

$$|d(q, p_i) - d(a, p_i)| > r \tag{2.4}$$

Los elementos no descartados forman parte de una lista de candidatos, que son los únicos que tienen la posibilidad de formar parte de la respuesta (Figura 2.7). Estos candidatos posteriormente se comparan directamente con la query q para determinar si su distancia es menor que el radio de tolerancia.

Si la base de datos X posee n elementos, se construye un índice con las $n.k$ distancias $d(x, p_i)$, por lo cual al momento de resolver la consulta (q, r) solo es necesario calcular las k distancias $d(q, p_i)$ para obtener la lista de candidatos. Es fácil ver que mientras mayor sea el conjunto de pivotes, mayor será la complejidad interna de la consulta.

Los elementos x no descartados por la condición de la expresión 2.4 deberán ser comparados directamente con q y comprobar si cumplen con la restricción del rango. A este cálculo de distancias adicionales se le denomina complejidad externa de la consulta (q, r).

A continuación se presenta una breve descripción de los algoritmos más conocidos basados en pivotes:

- **Burkhard-Keller Tree (BKT):** Este método asume una función de distancia discreta y se construye recursivamente de la siguiente manera: se elige un elemento arbitrario de la base de datos como pivote para la raíz del árbol y se agrupan los demás elementos de tal manera de que todos los que tienen la misma distancia al pivote, se agrupan en una misma rama cuya etiqueta es dicha distancia. Este mismo proceso se realiza para cada grupo hasta que las hojas del árbol contengan solo un elemento. Todos los nodos internos actúan como pivotes. Durante una búsqueda por rango $(q, r)_d$, para el nodo p, si $d(q, p) <= r$ se devuelve p como parte del resultado. Posteriormente se realiza el mismo

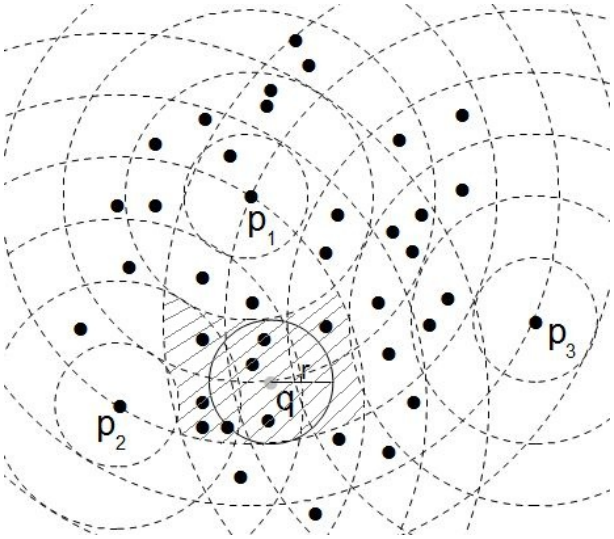

Figura 2.7: *El área resaltada indica las clases que no se pueden descartar*

.

proceso con las ramas cuya etiqueta se encuentra en el intervalo $[d(q,p) - r, d(q,p) + r]$ y se descartan las demás [15].

- **Fixed Queries Tree (FQT):** es una modificación del *BKT* en la cual para cada nivel del árbol se define un solo pivote. Esto disminuye la cantidad de cálculos de la función de distancia requerida por el método anterior, consecuencia de la comparación de la consulta q con cada nodo interno del árbol. Además se define una cantidad mínima necesaria de elementos para que se proceda a subdividir un nodo. Durante la búsqueda se utiliza un procedimiento similar al anterior hasta alcanzar las hojas del árbol. Todos los elementos de las hojas seleccionadas constituirán el conjunto de *candidatos*, los cuales deberán ser comparados con la consulta para obtener el resultado final [4].

- **Fixed Height Fixed Queries Tree (FHQT):** el *FHQT* o *FHFQT* [3] es una variante del *FQT* en la cual todas las hojas se encuentran al mismo nivel. Más adelante se explica su funcionamiento con mayor detalle ya que se utilizó como base para los índices métrico-espaciales desarrollados en esta tesis .

- **Fixed Queries Array (FQA):** está estrechamente relacionado al *FHQT*. Es un arreglo

de tamaño $n.h$ donde h es la altura del *FHQT* asociado (o cantidad de pivotes) y n la cantidad de firmas diferentes. Las firmas se encuentran ordenadas, por lo cual es posible utilizar una búsqueda de complejidad $O(log(n))$ para determinar cuales firmas definirán el conjunto de candidatos a ser comparados con la consulta [31].

- **Vantage Point Tree (VPT):** los métodos anteriores se utilizan fundamentalmente para distancias discretas. El *VPT* fue diseñado explícitamente para distancias continuas y es un árbol binario en el cual cada nodo interno contiene un pivote y se calcula la mediana de las distancias del pivote al conjunto de elementos considerados para subdividir dicho conjunto en dos grupos: el de los menores a la mediana como hijo izquierdo y el de los mayores o iguales a dicho valor como hijo derecho. Mediante este proceso el árbol queda balanceado. Durante una búsqueda $(q, r)_d$, para cada nodo p, si $d(q, p) <= r$ este elemento se incluye en el resultado y luego se decide cuales subárboles se deben visitar. Sea m la mediana correspondiente a los elementos descendientes del nodo p, si $d(q, p) + r < m$ entonces se descarta el hijo derecho, mientras que si $d(q, p) - r > m$ se descarta el hijo izquierdo. Note que es posible que sea necesario recorrer ambos subárboles [79]. Existen variantes de este árbol, tales como el *Multi-way Vantage Point Tree (mw-VPT)* [11] que sigue la misma estrategia pero utiliza percentiles para generar un árbol r-ario en lugar de binario y el *Excluded Middle Vantage Point Forest (VPF)* que excluye los elementos centrales (cercanos a la mediana) para generar otro árbol VPT, dando como resultado un bosque. De esta manera se reduce la probabilidad de que ante una consulta se requiera visitar tanto el subárbol izquierdo como el derecho [78].

- **Approximating Eliminating Search Algorithm (AESA):** esta estructura es una matriz donde se registran todas las distancias entre los elementos de la base de datos. En este caso cada elemento juega el rol de pivote respecto a los demás. Durante una búsqueda se elige un elemento de partida al azar y se utiliza la desigualdad triangular para descartar elementos sin compararlos con la consulta. Este proceso se realiza recursivamente

pero tomando solo los elementos no descartados. El método es muy eficiente en tiempo, pero su costo espacial $O(n^2)$ es demasiado alto como para ser útil en grandes bases de datos [74]. En [48] se presenta una variante denominada **Linear AESA (LAESA)** que disminuye el costo espacial a través de la reducción de la cantidad de pivotes.

Fixed Height Queries Tree (FHQT)

En esta sección se describe con mayor profundidad el *FHQT* [3], método que fue utilizado en esta tesis como base para el diseño de índices métrico-espaciales. Una característica importante de este índice es que es dinámico, es decir que si la cantidad de elementos de la base de datos crece o disminuye significativamente, se pueden agregar o eliminar pivotes (niveles) sin necesidad de modificar el resto de la estructura.

Este índice es una variante del Fixed Queries Tree (FQT) [4] en donde todas las hojas se encuentran a la misma altura. En otras palabras, los caminos más cortos se extienden hasta el último nivel de profundidad del árbol. Esta extensión tiene como consecuencia que la capacidad de descarte durante la búsqueda sea mayor, ya que se utilizarán la totalidad de los pivotes para tratar de eliminar cualquier elemento de la base de datos.

El *FHQT* se construye a partir de k pivotes que pueden ser elegido al azar o mediante algún procedimiento de selección de pivotes [17] de la base de datos X. También es posible que los pivotes se obtengan generándolos especialmente para un conjunto de datos, aunque no se tiene conocimiento de estudios que desarrollen esta posibilidad. Sea p un pivote, para cada distancia i de p hacia algún elemento de la base de datos, se crea el conjunto C_i formado por todos aquellos elementos de la base de datos que están a distancia i de p. Luego, para cada C_i se crea un hijo del nodo correspondiente a p, con rótulo i, y se construye recursivamente el árbol teniendo en cuenta que todos los subárboles del mismo nivel usarán el mismo pivote como raíz. Este proceso se continúa hasta lograr que todas las hojas tengan menos de b (tamaño del $bucket$) elementos y estén en un mismo nivel. La Figura 2.8 muestra un ejemplo de un FHQT con dos pivotes. Ante

una consulta $(q, r)_d$, se comienza por la raíz y se descartan todas aquellas ramas con rótulo i tal que $i \notin [d(p, q) - r, d(p, q) + r]$ siendo p el pivote utilizado en la raíz. La búsqueda continúa recursivamente en todos aquellos subárboles no descartados, utilizando el mismo criterio. Cuando se alcanzan las hojas se obtiene al conjunto de los elementos *candidatos*, que deberán ser comparados con la consulta para determinar si forman parte del resultado.

Ejemplo de Búsqueda

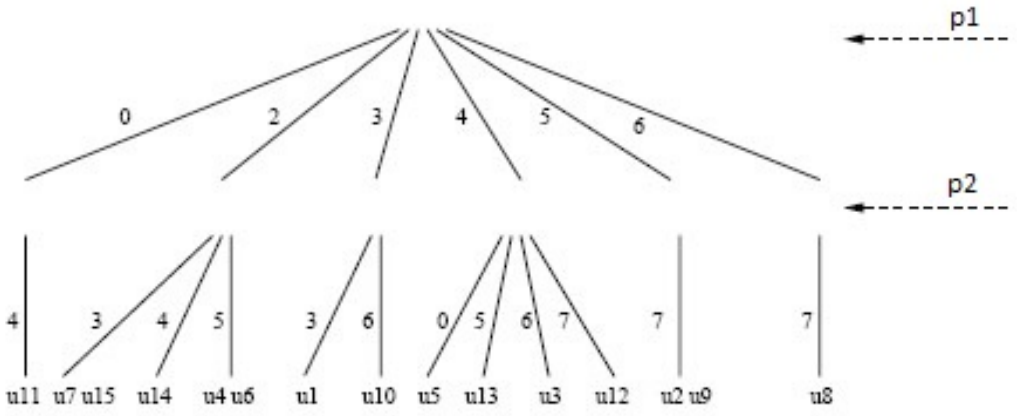

Figura 2.8: *FHQT con pivotes p_1 y p_2*

Por ejemplo, para la consulta $(q, 1)_d$, siendo $(4, 6)$ la firma de q, el FHQT de la Figura 2.8 se comporta de la siguiente manera: como el radio de búsqueda es 1 y la distancia de q al primer pivote es 4, aplicando la condición de exclusión 2.4, se seleccionan solo las ramas etiquetadas con las distancias del intervalo $[4 - 1, 4 + 1]$, es decir 3, 4 y 5 y se descartan las ramas 0, 2 y 6. En el siguiente nivel, se realiza el mismo proceso considerando en este caso solo las ramas $[6 - 1, 6 + 1]$ de los subárboles seleccionados en el nivel 1. Por lo tanto el conjunto resultante está formado por los siguientes objetos: $u_{10}, u_{13}, u_3, u_{12}, u_2, u_9$. En la Figura 2.9 se muestra el pseudocódigo del algoritmo de consulta del $FHQT$.

La complejidad espacial del *FHQT* está entre $O(n)$ y $O(nh)$, donde h el la altura del árbol y n la cantidad de elementos de la base de datos. Originalmente estas estructuras fueron pro-

FHQT $(q, r)_d$
1. calcular f de q – f es la *firma de* q
2. resultado := Consultar(q, r, 1, f, $raiz$)
3. **return** resultado

donde **Consultar** se define recursivamente como:

Consultar(q, r, n, f, x) – n es el nivel del nodo actual x
1. resultado:=$\emptyset$
2. **if** esHoja(x) **then**
3. **for all** objeto ($o \in x$)
4. **if** ($d(q, o) <= r$) **then**
5. resultado :=resultado $\cup$ $\{o\}$
6. **else**
7. **for all** hijo ($h_i \in x$)
8. **if** $|f_n - d_i| \leq r$ **then**
9. resultado :=resultado $\cup$ **Consultar**$(q, r, n + 1, f, h_i)$
10. **return** resultado

Figura 2.9: *Pseudocódigo de consulta del FHQT*

puestas para funciones de distancias discretas, pero se pueden adaptar a distancias continuas discretizando los valores de las mismas [65, 21] o utilizando rangos de valores en cada rama. Se considera que la cantidad óptima de pivotes es $\log_r(n)$, siendo r la cantidad máxima de hijos por nodo (cantidad de valores posibles para las distancias discretas, o cantidad de rangos definidos para las distancias continuas).

2.6. Maldición de la Dimensionalidad

Uno de los grandes obstáculos en el diseño de algoritmos de búsqueda por similitud es la llamada *maldición de la dimensionalidad* [23, 25, 26]. Si se representa el conjunto de elementos de un espacio métrico como puntos en un espacio vectorial, su dimensión corresponde al número de coordenadas que tienen los puntos que lo componen. Todos los algoritmos de búsqueda por similitud disminuyen su rendimiento cuando aumenta la dimensión del conjunto, hecho conocido como *maldición de la dimensionalidad*. En [25] se muestra que el concepto de dimensión de un espacio vectorial se puede extender a espacios métricos arbitrarios utilizando el concepto

de *dimensionalidad intrínseca*, y que la maldición de la dimensionalidad tiene consecuencias similares en dichos espacios.

Se define la *dimensionalidad intrínseca* de un espacio métrico como:

$$\rho = \frac{\mu^2}{2\sigma^2} \tag{2.5}$$

Los parámetros μ y σ de la Ecuación 2.5 son la media y la varianza del histograma de distancias de los puntos que conforman dicho espacio métrico. A medida que aumenta la dimensionalidad intrínseca del espacio métrico, los algoritmos basados en pivotes necesitan una mayor cantidad de pivotes para mantener, en cierta medida, su rendimiento, hasta un punto donde ya no hay mejora posible.

Cuando se realiza una consulta por rango, se quiere recuperar los elementos del espacio métrico que se encuentran a una distancia menor que un radio de tolerancia dado. Ante esta consulta, los algoritmos basados en pivotes buscan descartar la mayor cantidad de elementos del espacio métrico antes de realizar una búsqueda exhaustiva, es decir, generan una lista de elementos candidatos y luego verifican si los mismos cumplen con la condición de búsqueda comparándolos con el objeto consultado.

Considerando el histograma de distancias de un espacio métrico (U, d) es fácil ver que según la condición de exclusión de elementos (Ecuación 2.4), dada una consulta (q, r) y un conjunto de k pivotes p_i, se pueden descartar todos los elementos $x \in E$ que no cumplan para algún $i \in 1...k$ la condición de la Ecuación 2.6:

$$d(p_i, x) \in [d(p_i, q) - r, d(p_i, q) + r] \tag{2.6}$$

La dimensionalidad intrínseca del espacio métrico crece cuando la media del histograma aumenta y/o su varianza disminuye. Una disminución de la varianza del histograma tiene como consecuencia que la cantidad de elementos que se encuentran dentro del rango $[d(p_i, q) - r, d(p_i, q) + r]$ sea mayor, es decir, cada vez son menos los elementos que se pueden descartar.

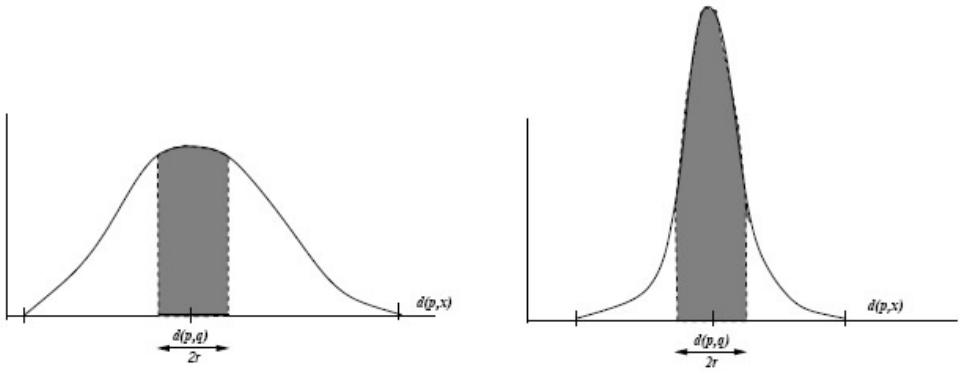

Figura 2.10: *Histogramas de distancias de baja dimensionalidad (izquierda) y de alta dimensionalidad (derecha)*

Por otro lado, si la media crece entonces se necesita un r mayor para que la búsqueda devuelva algún elemento, por lo que nuevamente, la capacidad de descarte se ve disminuida. En espacios de alta dimensionalidad, prácticamente todos los elementos se transforman en candidatos a ser verificados exhaustivamente (complejidad externa). A esto se le denomina *maldición de la dimensionalidad*, y es independiente de la naturaleza del espacio métrico al que pertenecen los elementos.

La Figura 2.10 muestra intuitivamente la maldición de la dimensionalidad. El histograma de distancias de la izquierda representa un espacio métrico de dimensión baja, y el de la derecha de dimensión alta. Se puede apreciar que en el caso del espacio con dimensión alta casi ningún elemento puede excluirse de la lista de candidatos, por lo que debe realizarse prácticamente una búsqueda exhaustiva para responder la consulta. Para poder descartar más elementos es necesario utilizar más pivotes, pero esto aumenta la complejidad interna de la consulta.

No obstante, respecto al crecimiento de la media, se debe hacer una consideración. Es posible que todos los elementos de la base de datos sean muy distintos entre sí (valor alto de la media), pero que el elemento que se consulta sea similar (cercano) a alguno de ellos. En este caso, el radio de búsqueda puede ser pequeño y aún así devolver un resultado, por lo cual el hecho de que la media de las distancias sea un valor grande, no significa necesariamente que el radio de búsqueda deba ser un número grande también.

En general, para espacios de muy alta dimensionalidad éstos algoritmos necesitan una gran

cantidad de pivotes, lo que implica enormes cantidades de memoria ya que el tamaño de los árboles crece exponencialmente. En la práctica, los espacios con dimensionalidad intrínseca menor a 20 se consideran tratables.

En [25] se muestra que el número óptimo de pivotes es $O(\log(n))$ si se eligen al azar, donde n es el número de elementos de la base de datos. A través de una selección adecuada de los pivotes, el costo de la búsqueda por similitud disminuye considerablemente. Los pivotes tienen que tener poder de discriminación (dividir los elementos en varios grupos de distancias) y ser complementarios, es decir, no discriminar los mismos elementos ya separados por otro pivote. La descripción detallada de los métodos de selección de pivotes queda fuera del alcance de este informe.

El desarrollo de índices métricos es un área activa de investigación. En los últimos años se han generado nuevos métodos de acceso tales como los que se muestran en [1], [5], [16], [19], [28], [51], [54], [59] y [60].

Capítulo III

MODELO ESPACIAL

En este capítulo se presenta una introducción al *Modelo Espacial*, el cual constituye el segundo aspecto en el cual se basa el modelo métrico-espacial. El procesamiento de consultas espaciales implica la ejecución de operaciones geométricas complejas y costosas. Las bases de datos espaciales suelen contener grandes cantidades de objetos geométricos, realizar un recorrido secuencial para resolver una consulta espacial no es una solución práctica en la mayoría de los casos, por lo cual en las aplicaciones reales es necesario el uso de índices espaciales. Como en el captítulo anterior, en éste se muestran además los métodos de acceso espacial (SAM) relevantes para este trabajo, incluyendo el índice *QuadTree* y los índices de la familia del *R-Tree*, utilizados como bases para el diseño del *QuadFQTree* y del *MeTree* respectivamente.

3.1. Conceptos Básicos

3.1.1. Representación de los Objetos Espaciales

Los SAM también pueden clasificarse en dos categorías [64]: estructuras dirigidas por el espacio y estructuras dirigidas por los datos. Las primeras están basadas en la partición de un espacio 2D en regiones rectangulares. Los objetos se mapean en las regiones de acuerdo a

algún criterio geométrico. En el segundo caso, las particiones están basadas en la distribución del conjunto de objetos que se está indexando.

A continuación se detalla cada una de las categorías antes mencionadas.

Estructuras Dirigidas por los Datos

La interpretación del espacio depende de la semántica asociada al territorio geográfico. Considerando, por ejemplo, el territorio de Argentina y adoptando un punto de vista administrativo, Argentina está dividida en varias provincias. En cambio, desde el punto de vista de un geólogo, se obtiene una organización del espacio completamente diferente en áreas geológicas; o si estamos interesados en el control del tráfico, la atención se centra en la red de rutas y caminos. En cada caso, elegimos una nueva interpretación del espacio y definimos una nueva colección de entidades que lo describen. Además, si bien la definición de un objeto espacial como un conjunto de puntos es bastante general, en la práctica se utilizan los siguientes tipos de objetos espaciales:

- Objetos o puntos de dimensión cero:lLos puntos se utilizan para representar la ubicación de entidades cuya forma no se considera útil, o cuando el área es bastante pequeña con respecto al tamaño del espacio en el que se encuentra. Las ciudades, los comercios y los cruces son ejemplos de entidades cuya extensión espacial podría reducirse a un punto en un mapa a gran escala, dependiendo de la problemática a representar.

- Objetos unidimensionales u objetos lineales: estos objetos se utilizan comúnmente para representar redes (carreteras, hidrografía, etc.). El tipo geométrico básico considerado usualmente en estos casos es la polilínea.

  Una polilínea se define como un conjunto finito de segmentos o aristas de línea, de modo que cada punto final de segmento (llamado vértice) es compartido exactamente por dos segmentos, excepto por los dos puntos finales (llamados puntos extremos), que pertenecen a un solo segmento. Ocasionalmente se consideran las siguientes variantes:

- Una polilínea está cerrada si los dos puntos extremos son idénticos.
- Una polilínea es simple, siempre que ningún par de aristas no consecutivas se crucen entre sí.
- Una polilínea es monótona con respecto a una línea $\mathcal{L}$ si cada línea $\mathcal{L}$' ortogonal a $\mathcal{L}$ se encuentra con la polilínea en un punto como máximo.

La última polilínea de la Figura 3.1 no es monótona. Se utiliza a menudo el término línea en lugar de polilínea cuando no hay ambigüedad.

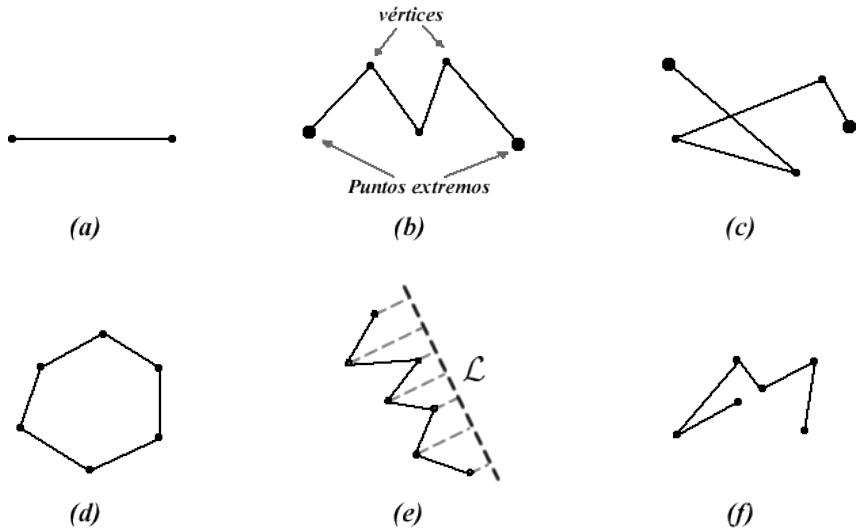

Figura 3.1: *Ejemplo de Objetos Unidimensionales: (a) Segmento de Línea; (b) Polilínea; (c) Polilínea no simple; (d) Polilínea simple cerrada; (e) Polilínea monótona; (f) Polilínea no monótona.*

- Los objetos bidimensionales u objetos de superficie se utilizan principalmente para representar entidades con grandes áreas, como parcelas o provincias. Los *polígonos* constituyen el principal tipo geométrico de tales objetos. Un polígono es una región del plano delimitada por una polilínea cerrada, llamada *límite*. Es habitual distinguir los siguientes tipos de polígonos:
  - Un polígono es *simple* si su límite es una polilínea simple.

- Un polígono *convexo* P es tal que para cualquier par de puntos A y B en P, el segmento AB está completamente incluido en P.
- Un polígono *monótono* es un polígono simple siempre que su límite L se pueda dividir en exactamente dos polilíneas monótonas. La monotonicidad se expresa generalmente con respecto a los ejes.

Los objetos bidimensionales no están necesariamente conectados, por ejemplo, un país y sus islas se representan mediante una *región* conformada por un conjunto de polígonos (Figura 3.2 (f)).

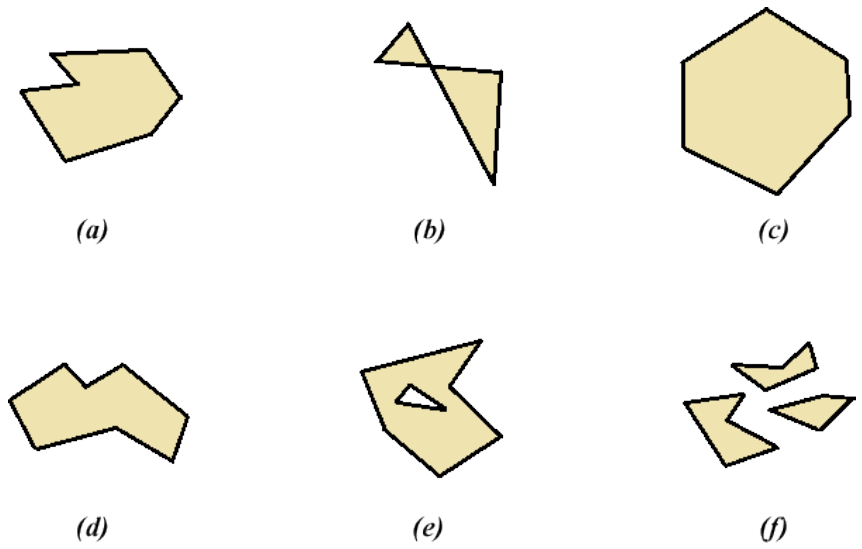

Figura 3.2: *Ejemplo de Objetos Bidimensionales: (a) Polígono simple; (b) Polígono no simple; (c) Polígono convexo; (d) Polígono monótono; (e) Polígono con hueco; (f) Región.*

Es importante mencionar dos observaciones sobre la representación espacial: primero, la elección de tipos geométricos es arbitraria. Más precisamente, depende del uso de la colección de entidades. Muchos factores pueden influir en esta elección, siendo uno importante la escala de interés para las aplicaciones previstas. Por ejemplo, un aeropuerto puede verse como un punto (si está interesado en enlaces aéreos) o como un área, si el enfoque es la organización interna del aeropuerto. Puede ser interesante almacenar la representación de objetos a varias escalas

y obtener la representación adecuada dependiendo de lo solicitado. Una segunda observación sucede en la representación de objetos lineales y superficiales, las cual se basa en segmentos de línea. En otras palabras, usamos sólo una aproximación lineal de entidades, esta podría representarse de manera más precisa con polinomios de orden superior en x e y. Esta aproximación simplifica el diseño de bases de datos espaciales y conduce a formas eficientes de modelar y consultar información espacial. Sin embargo, existe una compensación entre una aproximación fiel y el número de segmentos para representar la curva.

Estructuras Dirigidas por el Espacio

En un enfoque basado en el espacio, con cada punto del espacio se asocia uno o varios valores de atributo, definidos como funciones continuas en x e y. La altitud sobre el nivel del mar es un ejemplo de función definida sobre x e y, cuyo resultado es el valor de una variable h para cualquier punto del espacio 2D. Las medidas para varios fenómenos se pueden recopilar como valores de atributo, los cuales varían con la ubicación en el plano. Estos son, por ejemplo, precipitación, temperatura, etc. Esta visión del espacio como un campo continuo contrasta con el modelo basado en entidades, donde se identifica un conjunto de puntos (región, línea) como una entidad u objeto.

3.1.2. Tipos de Datos Abstractos Espaciales

Los TDA se introdujeron en las bases de datos como una forma de superar la falta de poder de modelado del modelo relacional cuando hay necesidad de representar objetos no estructurados. Un TDA es una vista funcional abstracta de objetos definida a través de un conjunto de operaciones sobre un tipo de datos. La idea detrás de este concepto es ocultar la estructura del tipo de datos al usuario, que puede acceder a él sólo a través de las operaciones definidas en él, por lo que la interfaz de un TDA es una lista de operaciones (encapsulamiento). Gracias a esto, es posible definir una lista de tipos de datos espaciales que proporcionan una interfaz

conveniente y sencilla para el usuario. En la Figura 3.3 se muestran algunos TDA espaciales.

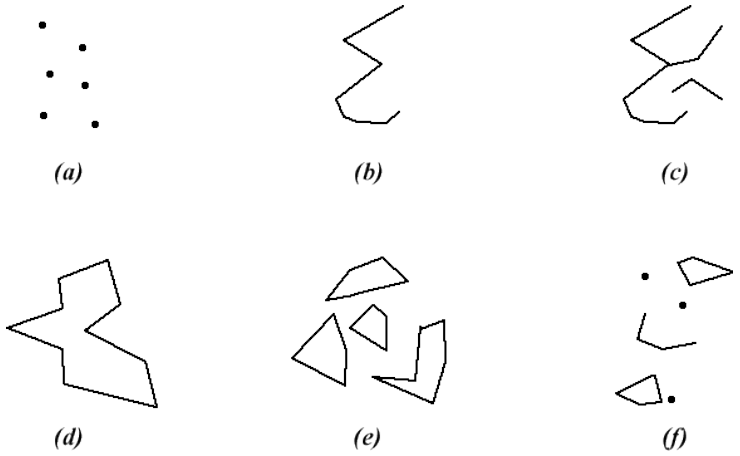

Figura 3.3: *Tipo de Datos Abstractos Espaciales: (a) Puntos; (b y c) Polilíneas; (d y e) Regiones y (f) Conjunción de todos los tipos.*

Los tipos de datos abstractos espaciales tomados en cuenta en este trabajo de tesis fueron:

- Tipo Punto. Un punto representado por dos coordenadas es una instancia de este tipo.
- Tipo Polilínea. Una instancia de este tipo es una lista de segmentos conectados de a pares (un segmento es parte de una línea limitada por dos puntos). Agregamos la restricción de que un punto final puede ser compartido por dos segmentos como máximo.
- Tipo Región. Una instancia de este tipo es cualquier conjunto de polígonos que no se superponen. En algunas situaciones, elegir un tipo de polígono para objetos de superficie puede ser demasiado restrictivo, ya que algunos objetos geográficos (por ejemplo ríos anchos como el Rio de la Plata) pueden ser muy complejos de representar de esta manera.

3.1.3. Operaciones Espaciales

Elegir un conjunto de operaciones asociados a un TDA no es sencillo. En particular, el resultado de la operación puede ser cualquier tipo atómico (por ejemplo, real o entero) proporcionado por el sistema de gestión de bases de datos, o uno de los tres tipos abstractos espaciales.

Una de las operaciones espaciales más importantes es la Intersección de dos polígonos (Figura 3.4). Cada polígono representa el conjunto infinito de puntos que se encuentran dentro del mismo, incluyendo su frontera. La Intersección de dos polígonos estará definida por todos los puntos comunes a ambos polígonos. El resultado podría ser, por ejemplo: (a) dos polígonos, en cuyo caso se puede representar como un objeto de tipo región; (b) dos polígonos y una línea; o (c) dos polígonos, una línea y dos puntos. En estos últimos casos, no hay un tipo que represente estas combinaciones.

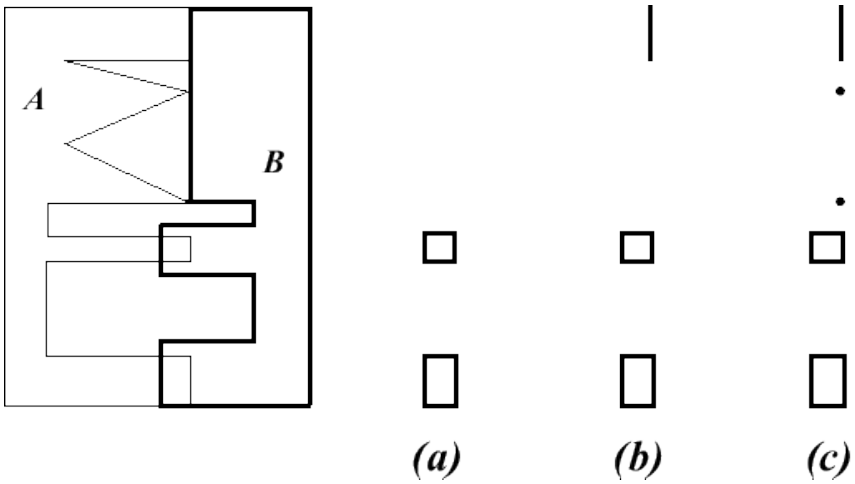

Figura 3.4: *Resultado de una intersección: (a) $A \cap B$ puede ser dos polígonos, en cuyo caso se puede representar como un objeto de tipo región; (b) dos polígonos y una línea; (c) o dos polígonos, una línea y dos puntos.*

En aras de la simplicidad, adoptamos aquí la primera semántica. El resultado es un conjunto de polígonos e ignoramos los puntos y líneas. Observe que esto no coincide exactamente con la semántica de intersección de los conjuntos de puntos. Sin embargo, el resultado se ve obligado a ser uno de los tipos elegidos.

A continuación se describen otras operaciones espaciales sobre estos tipos de datos. Cada operación tiene un nombre y una firma. La firma de operación indica el tipo de cada uno de sus argumentos (Dominio) y de su resultado (Codominio). Además se describe brevemente su semántica.

El TDA Región incluye las siguientes operaciones:

- PuntoEnRegion: región × punto → bool (Comprueba si un punto se encuentra en una región).
- Superposicion: región × región → bool (Comprueba si dos regiones se cruzan).
- Intersección: región × región → región (Devuelve la intersección de dos regiones: una región, posiblemente vacía).
- Adyacencia: región × región → bool (Prueba la adyacencia de dos regiones).
- Área: región → real (Devuelve el tamaño del área de una región).
- Union: región* → región (Toma un conjunto de regiones y devuelve una región, la unión de las regiones de entrada. Esta función se puede ver como una función agregada en terminología SQL, similar a min, max y sum, entre otras funciones.)

EL TDA Línea posee las siguientes operaciones:

- PuntoEnLinea: línea × punto → bool (Prueba la intersección entre un punto y una línea).
- Longitud: línea → real (Calcula la longitud de una línea).
- InterseccionLR: línea × región → bool (Prueba la intersección entre una línea y una región).
- InterseccionLL: línea × línea → punto (Devuelve la intersección entre dos líneas: punto, posiblemente vacío).

El TDA Punto posee la siguiente operación.

- Distancia: región × punto → real (Calcula la distancia entre un punto y el límite de una región).

Aunque esta lista claramente no es exhaustiva, puede brindarnos un conjunto simple de operaciones suficientes para responder a diversas consultas.

3.2. Índices Espaciales

En esta sección se presentan métodos de acceso espacial haciendo énfasis y detallando con mayor profundidad los índices *QuadTree* y *R-Tree*, que fueron tomados como base para el diseño del aspecto espacial de los índices propuestos en este trabajo.

3.2.1. Fixed Grid

El Fixed Grid divide el espacio en celdas fijas para indexar puntos. Cada celda está asociada a una página de disco donde se almacenan secuencialmente los objetos contenidos en dicha celda. Si la celda se llena, se divide en dos. Existen variantes del mismo para indexar rectángulos.

El espacio de búsqueda se descompone en $celdas$ rectangulares. La cuadrícula resultante es una matriz de $n_x \times n_y$ de celdas de igual tamaño. Cada celda c está asociada con una página de disco. El punto P se asigna a una celda c si el rectángulo $c.rect$ asociado con la celda c contiene a P.

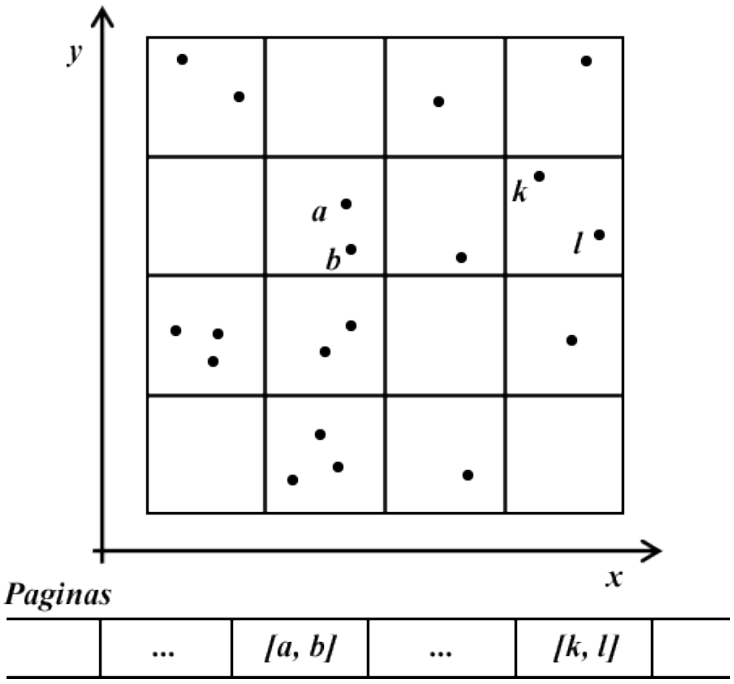

Figura 3.5: *Fixed Grid*

Los objetos asignados a una celda c se almacenan secuencialmente en la página asociada con c. La Figura 3.5 muestra una cuadrícula fija que indexa una colección de 19 puntos. El espacio de búsqueda tiene como origen el punto con coordenadas (x_0, y_0).

El índice requiere una matriz 2D $DIR_{[1:n_x,1:n_y]}$ como $directorio$. Cada elemento $DIR_{[i,j]}$ del directorio contiene la dirección $paginaID$ de la página que almacena los puntos asignados a la celda $c_{i,j}$. Si [S_x, S_y] es el tamaño 2D del espacio de búsqueda, el tamaño del rectángulo de cada celda es [S_x/n_x, S_y/n_y]. A continuación se describen los algoritmos para la inserción de puntos, consulta punto y consulta de tipo ventana:

- Insersión de $P_{(a,b)}$: calcular $i = (a - x_0)/(S_x/n_x) + 1$ y $j = (b - y_0)/(S_y/n_y) + 1$, y luego obtener la página $DIR_{[i,j]}.paginaID$ e insertar P.

- Consulta punto: dado como argumento un punto $P_{(a,b)}$, obtener la página correspondiente (de la misma manera que para la inserción), leer dicha página, y verificar si alguno de los elementos es P.

- Consulta ventana: calcular el conjunto S de celdas c tales que $c.rectangulo$ se superponga a la ventana de consulta V; para cada celda $c_{i,j}$ en S, hallar la página $DIR_{[i,j]}.paginaID$ y devolver los puntos de la página contenidos en la ventana V.

La consulta por puntos es eficiente en este contexto. Suponiendo que el directorio reside en la memoria principal, una consulta puntual requiere una única E/S. El número de las E/S para una consulta de ventana dependen del número de celdas que intersecta la ventana; es decir, es proporcional al área de la ventana.

La resolución de la cuadrícula depende del número N de puntos a indexar. Dada una capacidad de página de M puntos, se puede crear una cuadrícula fija con al menos N/M celdas. Cada celda contiene, en promedio, menos de M objetos. Una celda a la que se asignan más de M puntos se desborda. Entonces, las páginas desbordadas son encadenadas a la página inicial.

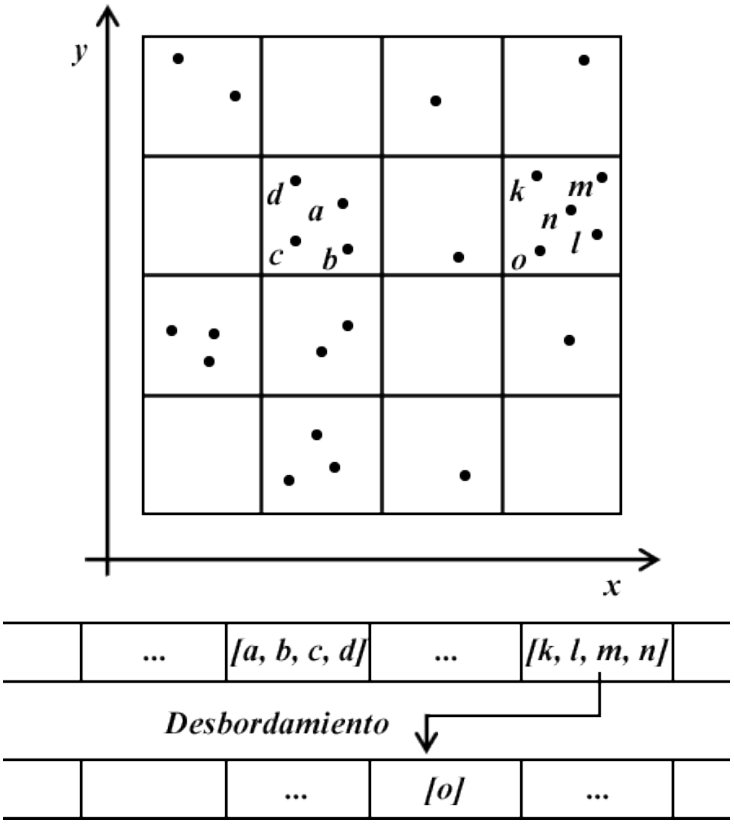

Figura 3.6: *Desbordamiento (Overflow) en Fixed Grid*

Por ejemplo, asumiendo que cada página puede almacenar como máximo cuatro entradas, los puntos concentrados en exceso dentro de un área reducida se indexarán como se ilustra en la Figura 3.6, en la cual los elementos k, l, m, n y o generan que una celda se desborde. En este caso, k, l, m y n se almacenan en la página principal p, mientras que o se asigna a una página de desbordamiento vinculada a p. Entonces, una consulta por punto puede requerir hasta r E/S si el punto se almacena en la r-ésima página de una cadena de desbordamiento.

Si los puntos se distribuyen uniformemente en el espacio de búsqueda, el fenómeno descrito anteriormente rara vez ocurre y las cadenas de desbordamiento no son largas, a menos que el tamaño del conjunto de datos aumente significativamente. En este caso incluso si la distribución de puntos permanece uniforme, las inserciones de nuevos puntos producir un gran número de páginas de desbordamiento. Por el contrario, las eliminaciones pueden resultar en páginas casi

vacías. En resumen, este índice es útil para distribuciones uniformes de puntos y con tamaño de la base de datos previamente conocido.

3.2.2. Grid File

En el *Grid File* [52, 61], como en *Fixed Grid*, se asocia una página con cada cuadrante, pero cuando un cuadrante se desborda, se divide en dos cuadrantes y los puntos se asignan al nuevo cuadrante en el que se encuentran. Entonces los cuadrantes son de diferente tamaño y la partición se adapta a la distribución de puntos. Para ello son necesarias dos estructuras de datos:

- El directorio DIR es una matriz 2D que hace referencia a páginas asociadas a las celdas. La estructura es similar a la de la *Fixed Grid*, aunque difiere en dos cuadrantes adyacentes pueden hacer referencia a la misma página.
- Escalas lineales: arreglos unidimensionales con los puntos de corte, los cuales inducen las particiones sobre cada uno de los ejes.

La construcción de la estructura se describe mejor con un ejemplo. La Figura 3.7, ilustra los cuatro pasos en la construcción de un *Grid File* de una colección de puntos. Para simplificar, asumimos que M, la capacidad de una página, es 4. En el paso (a), la estructura contiene sólo cuatro objetos. El directorio es un solo cuadrante asociado a la página p_1. En el paso (b), se insertan sucesivamente los puntos a, b y c. Al insertar a, p_1 se desborda y entonces se subdivide. Para ello se asigna una nueva página de datos p_2 y los cinco puntos se distribuyen entre p_1 y p_2. La división vertical se representa en el eje x como valor x_1. a y b se insertan en p_1 y c en p_2.

En el paso (c), se inserta el punto d. Éste debe almacenarse en la página p_1, como p_1 está lleno, se realiza una división horizontal con valor y_1 sobre el eje y. Luego, los cinco puntos de esa partición se distribuyen entre las páginas p_1 y p_3. Cada nueva división también implica dividir todos los cuadrantes impactados por la misma, aunque existan cuadrantes donde no

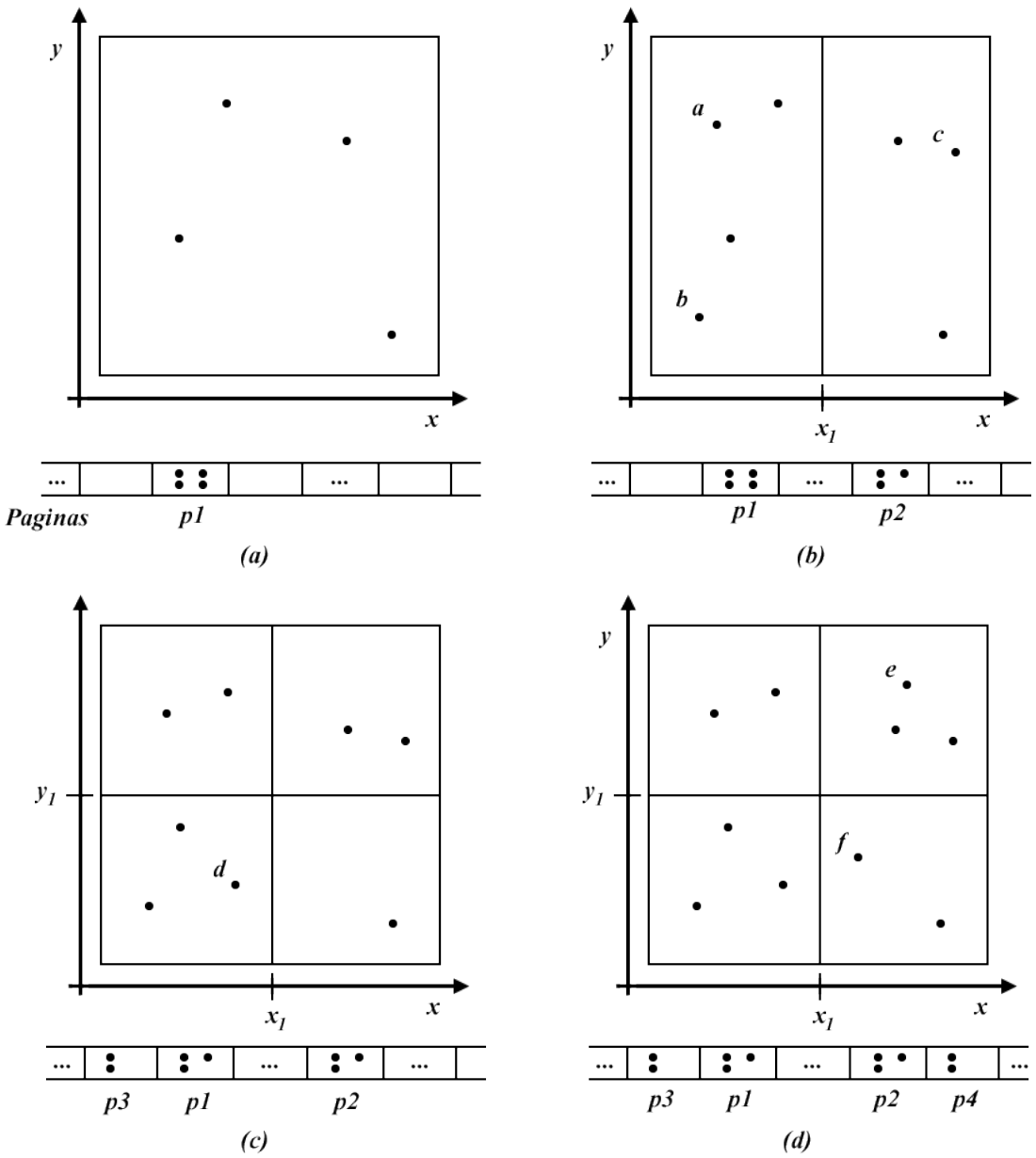

Figura 3.7: *Inserciones en Grid File*

exista desborde. Sin embargo, los cuadrantes divididos sin desbordamiento siguen haciendo referencia a las mismas páginas. En el ejemplo, el cuadrante de la derecha se reemplaza por dos nuevos cuadrantes, los cuales apuntan a p_2 (Figura 3.7 (c)), donde no se generó desbordamiento de página en ninguno de los cuadrantes generados.

Finalmente, en el paso (d), se insertan e y f. Esto genera el desbordamiento de p_2 y una

nueva página (p_4) es asignada. En este caso, no es necesaria una división de eje, porque ya se definió una división en la estructura DIR en el valor y_1. Los puntos de los cuadrantes se almacenan en p_2 y p_4 donde sí se generó un desbordamiento por poseer más elementos que lo permitido por M.

En resumen, al insertar un punto P, se deben considerar tres casos:

- *Sin divisiones de cuadrantes*. Este es el caso más simple: el punto P cae en una página que no está llena. Luego se accede a la página y se inserta el punto en ella.
- *División de cuadrantes sin división de directorio*. El punto P que se va a insertar cae en el cuadrante c, y la página p asociada con c está llena pero referenciada por varios (al menos dos) cuadrantes distintos. En ese caso, se crea una nueva página p' y se asigna a c. Los objetos en p contenidos en c, así como el punto a insertar, se mueven a p'. La entrada de directorio correspondiente a c se actualiza con p' (ver paso (d)).
- *División de cuadrante y división de directorio*. Este es el caso más complejo. La página p a la cual hace referencia el cuadrante c en la que P cae está llena, y no hay otros cuadrantes que hagan referencia a p. Entonces c se divide sobre el eje x o sobre el eje y.

El *Grid File* supera los límites principales de *Fixed Grid*. Una búsqueda por puntos siempre se puede realizar con dos accesos al disco (uno para acceder a la página p referenciada en el directorio y otro para la página de datos asociada con una entrada en p). Además, la estructura es dinámica y presenta un comportamiento razonable con respecto a la utilización del espacio. Sin embargo, una deficiencia del método es que para conjuntos numerosos de datos, el número de cuadrantes en el directorio puede ser tan grande que el directorio podría no caber en la memoria principal.

3.2.3. K-D-Tree

El K-D-Tree (árbol K-Dimensional) [6, 29, 62] fué diseñado como Método de Acceso a Puntos y sentó las bases para varios Métodos de Acceso Espacial desarrollados posteriormente.

Es un árbol de búsqueda binario donde cada nodo representa un punto K-dimensional en el espacio. Cada nodo interior del árbol genera implícitamente un hiperplano que divide el espacio en dos partes, conocidas como espacios medios. Los puntos a la izquierda de este hiperplano estarán representados por el subárbol izquierdo de ese nodo y los puntos a la derecha del hiperplano por el subárbol derecho. Por ejemplo, si se indexan puntos bidimensionales, la raíz podría dividir el espacio en forma perpendicular al eje x y luego en el siguiente nivel la división será perpendicular al eje y y así alternadamente hasta indexar todos los elementos de la base de datos o alcanzar subconjuntos de cardinalidad menor o igual a un tamaño de bucket predefinido.

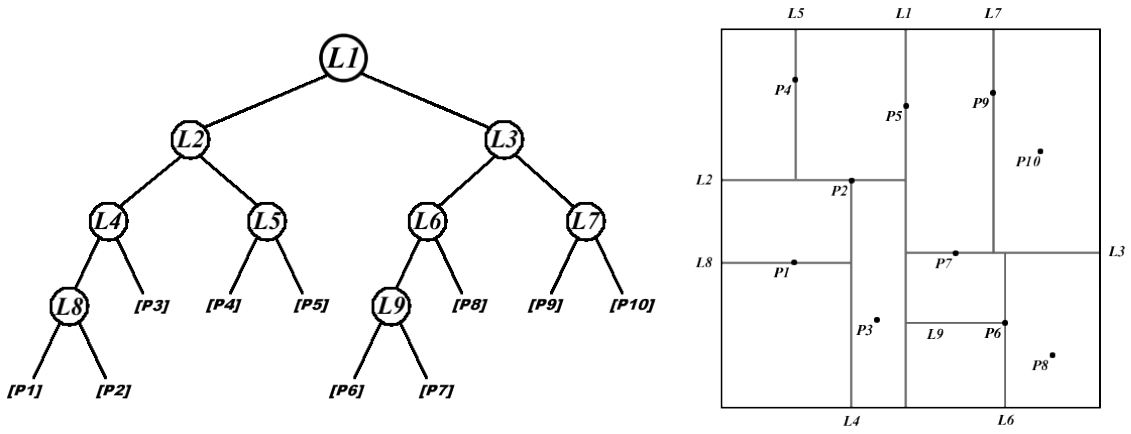

Figura 3.8: *K-D-Tree*

Como se muestra en la Figura 3.8, para representar esta estructura se genera un nodo por cada corte realizado. Todos los puntos menores que el punto de corte actual p, de acuerdo al eje utilizado, se guardan en el hijo de la izquierda, y los mayores en el derecho (si la coordenada correspondiente a un punto es igual a la de p, se puede insertar en cualquiera de los dos hijos, para tratar de balancear los subárboles). Cuando se alcanzan todos los puntos o sólo resta un número pequeño de puntos, se almacenan en las hojas. El tamaño de bucket se obtiene como

parámetro de inicio e indica la cantidad mínima de elementos por nodo para continuar subdividiendo el espacio o detenerse. No es necesario almacenar el eje de corte, esta información se obtiene al recorrer el árbol.

Para un mejor rendimiento, la división del espacio debe ser lo más balanceada posible. Para ello, el método más común utilizado es seleccionar el valor de corte basado en la mediana del eje de corte del subconjunto de elementos correspondientes. Esto produce un árbol de altura $O(log(n))$. Con un procedimiento recursivo se puede construir el árbol en $O(n.log(n))$. El mayor costo está dado por el cálculo de la mediana del eje.

Consulta de los k Vecinos Más Cercanos (*kNN*)

Para realizar una consulta del *Vecino Más Cercano (NN: Nearest Neighbor)* en el *K-D-Tree*, se procede de la siguiente manera. En primer lugar se recorre el árbol desde la raíz hasta la hoja correspondiente al objeto que se consulta (como si fuese una inserción), se calculan las distancias hacia cada elemento de la hoja y se toma el mínimo. Este elemento es el primer candidato a ser el *NN*. La distancia entre la consulta y dicho elemento se utiliza como radio inicial para descartar particiones. Entonces se regresa el nodo padre de la hoja y se verifica si el rectángulo correspondiente al otro hijo se intersecta o no con la esfera actual de la consulta, definida por el radio calculado. Si no hay intersección, el subárbol se descarta. En caso de tener puntos en común, se recorre y se actualiza el NN si se encuentra un elemento a menor distancia. Este proceso se realiza hasta haber recorrido o descartado todos los subárboles.

Para obtener los *kNN* sólo es necesario que en lugar de calcular un solo valor como elemento a menor distancia, se mantengan los kNN calculados hasta el momento a través de una lista ordenada de menor a mayor por distancia a la consulta q, y se devuelve esta lista como resultado.

3.2.4. QuadTree

Esta estructura de datos orientada a memoria principal se utiliza comúnmente para acelerar el acceso a objetos 2D debido a su simplicidad [18, 35, 63, 80, 83] . Es especialmente eficiente cuando los elementos son puntos, aunque se puede utilizar también para indexar líneas y polígonos. El espacio de búsqueda se descompone de forma recursiva en cuadrantes hasta que el número de objetos contenidos en cada cuadrante sea menor que la capacidad de la página. Los cuadrantes se denominan Noroeste (*NW*), Noreste (*NE*), Suroeste (*SW*) y Sureste (*SE*). El índice se representa como un árbol cuaternario (cada nodo interno tiene cuatro hijos, uno por cuadrante). Cada hoja se asocia a la página de disco que almacena las entradas correspondientes. Como en el caso del *Grid File*, el mismo objeto aparecerá en todos los cuadrantes con los cuales haya superposición. La Figura 3.9 muestra una partición de la colección de búsqueda para una colección de polígonos representados a través de sus MBBs, y el árbol asociado. Las hojas están etiquetadas con sus rectángulos correspondientes. Se supone en este ejemplo una capacidad de página de cuatro entradas.

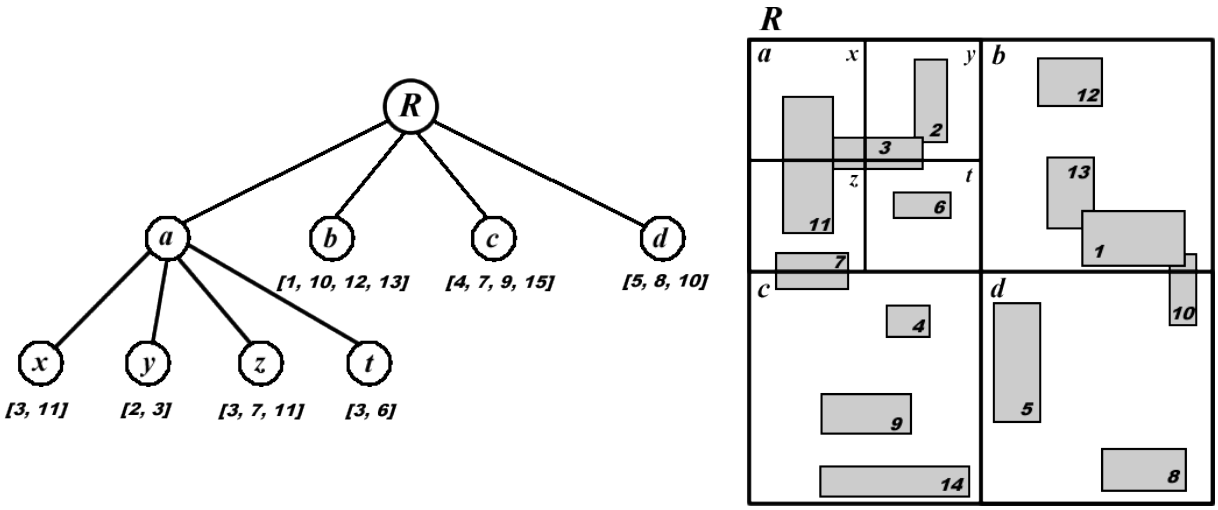

Figura 3.9: *QuadTree*

La consulta de puntos en el *QuadTree* es simple. Se sigue un único camino desde la raíz del árbol hasta la hoja. En cada nivel, se elige entre los cuatro cuadrantes el que contiene es-

pacialmente al punto de consulta. La Figura 3.10 ilustra una consulta puntual y la ruta seguida desde la raíz hasta la hoja, con el rectángulo 11 como resultado. En cambio, para una consulta ventana (rectángulo) es necesario ingresar en todos los cuadrantes hijos que se intersecten con el rectángulo consulta.

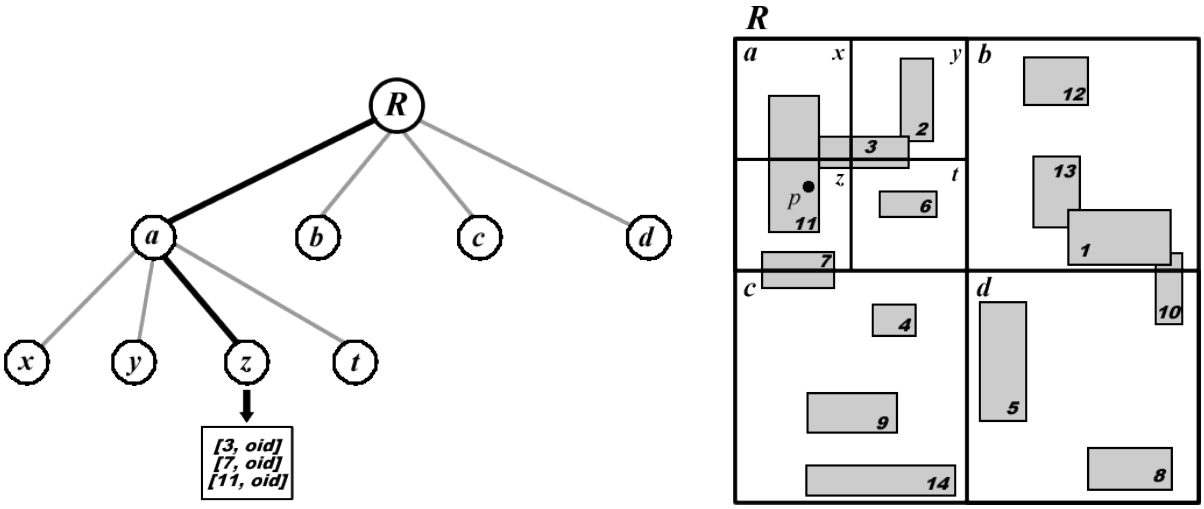

Figura 3.10: *Consulta por puntos en QuadTree*

Inserción en el QuadTree

En el caso de que el QuadTree indexe elementos geométricos que no sean puntos, primero se calcula el *MMB* del objeto y luego se inserta este rectángulo en cada cuadrante con los cuales haya superposición. Esta operación se repite hasta llegar a las hojas. Luego se lee la página p asociada a cada hoja. Entonces pueden ocurrir dos casos: p tiene lugar (en este caso se inserta una nueva entrada) o está llena. En este último caso el cuadrante se subdivide en cuatro nuevos cuadrantes. Uno de ellos mantiene asociada la página actual y a los otros tres se les asignan nuevas páginas. Las entradas de la página antigua y el nuevo elemento se distribuyen entre las cuatro páginas. Se agrega una entrada e a cada una de las cuatro páginas cuyos cuadrantes se intersectan con el *MBB* de e. La Figura 3.11 muestra el índice obtenido después de la inserción de los rectángulos 15 y 16. La inserción de 15 no da lugar a ninguna división, por lo tanto sólo

se añade a la hoja d. Pero la inserción de 16 conduce a la división de c en los cuadrantes e, f, g y h. Se agrega 16 a las hojas t y f.

El tiempo de consulta de *quadtree* está relacionado con la profundidad del árbol, que puede ser importante. En el peor de los casos, cada nodo interno del árbol está en una página separada y entonces el número de E/S es igual a la profundidad del árbol. Si la colección es estática, existen empaquetamientos eficientes de los nodos del árbol en las páginas, pero el rendimiento se degrada en el caso dinámico.

Al igual que el *Grid File*, cuando se indexan objetos geométricos con extensión, el *QuadTree* sufre de duplicación de objetos en varias hojas. Cuando el tamaño de la colección es muy grande, el tamaño del cuadrante es del orden del tamaño de los rectángulos indexados, la tasa de duplicación (y por lo tanto el tamaño del índice) aumenta exponencialmente y este SAM deja de ser efectivo. Por esta razón el *QuadTree* se utiliza principalmente para indexar puntos.

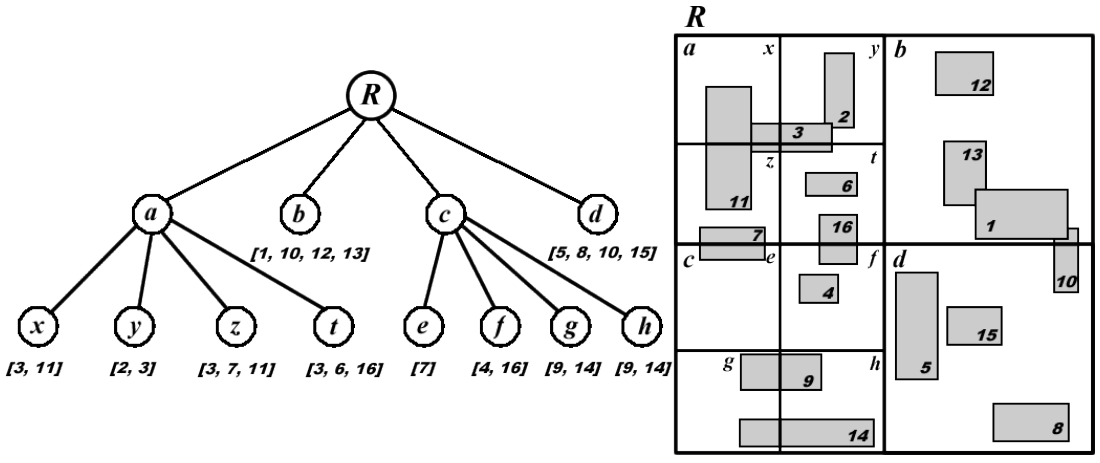

Figura 3.11: *Inserción en QuadTree*

Bitplane Quadtree

El Bitplane Quadtree (BQ-Tree) [41] es una variante diseñada para datos geoespaciales de gran escala. Posee múltiples utilizaciones en distintos tipos de aplicaciones [44] [81].

Ejemplo del Quadtree

En la Figura 3.12 se muestra una parte del *QuadTree* utilizado en este trabajo, obtenido en base a la distribución espacial de los elementos con los cuales se realizaron los experimentos.

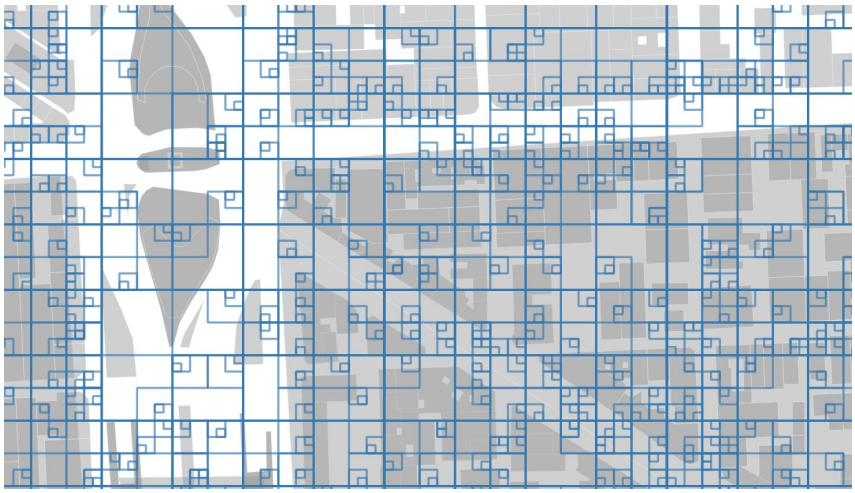

Figura 3.12: *QuadTree desplegado en el microcentro de Bs. As.*

3.2.5. Space Filling Curves

Una ”Space Filling Curve” o *Curva de Recubrimiento del Espacio* [20] es un SAM que define un orden total en las celdas de una cuadrícula 2D. Este orden es útil porque conserva parcialmente la proximidad, es decir, es probable que dos celdas cercanas en el espacio estén cercanas en el orden total. Aunque este no es siempre el caso, algunas curvas proporcionan una aproximación razonable de esta propiedad de proximidad. El orden también debe ser estable con respecto a la resolución de la cuadrícula.

A continuación describimos dos *Curvas de Recubrimiento del Espacio* populares utilizando como cuadrícula subyacente una matriz de celdas de NxN.

La primera variedad (ver 3.13 a), conocida como *Orden-Z*, se puede generar a partir de un QuadTree de la siguiente manera: se asocia una etiqueta a cada nodo del QuadTree completo, compuesta por símbolos del alfabeto {0, 1, 2, 3}. La raíz tiene como etiqueta la cadena vacía.

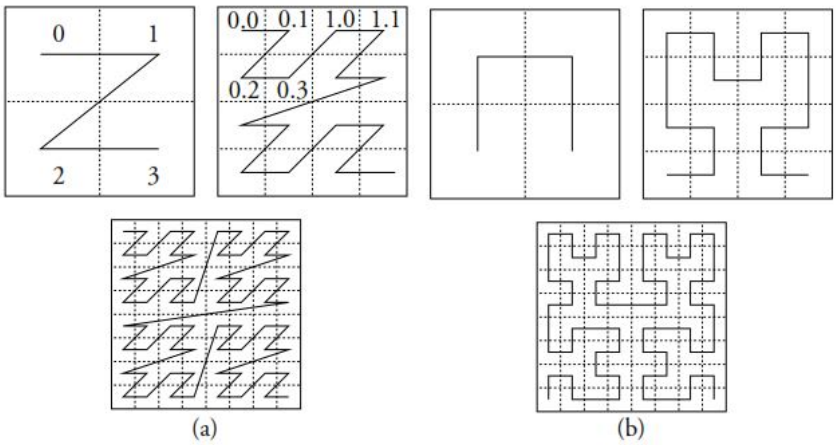

Figura 3.13: *Curvas que recubren el espacio: Orden z (a) y curva de Hilbert (b)*

El hijo NW (respectivamente, NE, SW, SE) de un nodo interno cuya etiqueta es k tendrá como etiqueta $k0$ (respectivamente, $k1$, $k2$, $k3$). Por ejemplo, una etiqueta de nivel 3 podría ser 102. Luego las celdas se pueden ordenar lexicográficamente de acuerdo a sus etiquetas. Si elegimos una profundidad $d = 3$ y un orden ascendente, la celda 212 estará antes que la celda 300 y después que la celda 21. El orden NW, NE, SW, SE dá origen al nombre *Orden-Z*. Este orden también se denomina código Morton.

La curva de Hilbert [20] también es una curva unidimensional, que visita cada celda dentro de una cuadrícula 2D. La forma no es de una Z sino de una Π. A diferencia del *Orden-Z*, la curva de Hilbert consta de segmentos de longitud uniforme, es decir, nunca se saltará a una ubicación distante mientras se recorren las celdas (Figura 3.13 (b)). Cómo fácilmente se observa, en ambos casos existen algunas situaciones inevitables en las que dos objetos están cerca en el espacio 2D, pero lejos uno del otro en la curva que recubre el espacio.

3.2.6. R-Tree

Un *R-Tree* es un árbol $M - ario$ balanceado [30, 39, 67] en el que cada nodo corresponde a una página de disco. Un nodo hoja contiene una lista de pares (MBB, oid), donde MBB es el (hiper) rectángulo mínimo que contiene al objeto representado por oid. Los MBB tienen

la forma (x_1, y_1, x_2, y_2, ..., x_d, y_d), donde d es la cantidad de dimensiones del espacio de búsqueda. Aunque la estructura de *R-Tree* puede manejar datos con dimensiones arbitrarias, lo más habitual es trabajar con dos o tres dimensiones. Un nodo intermedio (no hoja) contiene un arreglo de entradas. La estructura satisface las siguientes propiedades:

- Para todos los nodos del árbol (excepto la raíz), el número de entradas ocupadas está entre m y M, donde $m \in [0, M/2]$.
- Para cada entrada (dr, $nodo_{id}$) de un nodo interior N, dr es el rectángulo mínimo que contiene espacialemente a todos los elementos del subárbol de N cuya dirección de página es $nodo_{id}$.
- Para cada par (MBB, oid) de una hoja, MBB es el (hiper) rectángulo mínimo que contiene al objeto almacenado en la dirección oid.
- La raíz tiene al menos dos entradas ocupadas (a menos que sea una hoja).
- Todas las hojas están al mismo nivel.

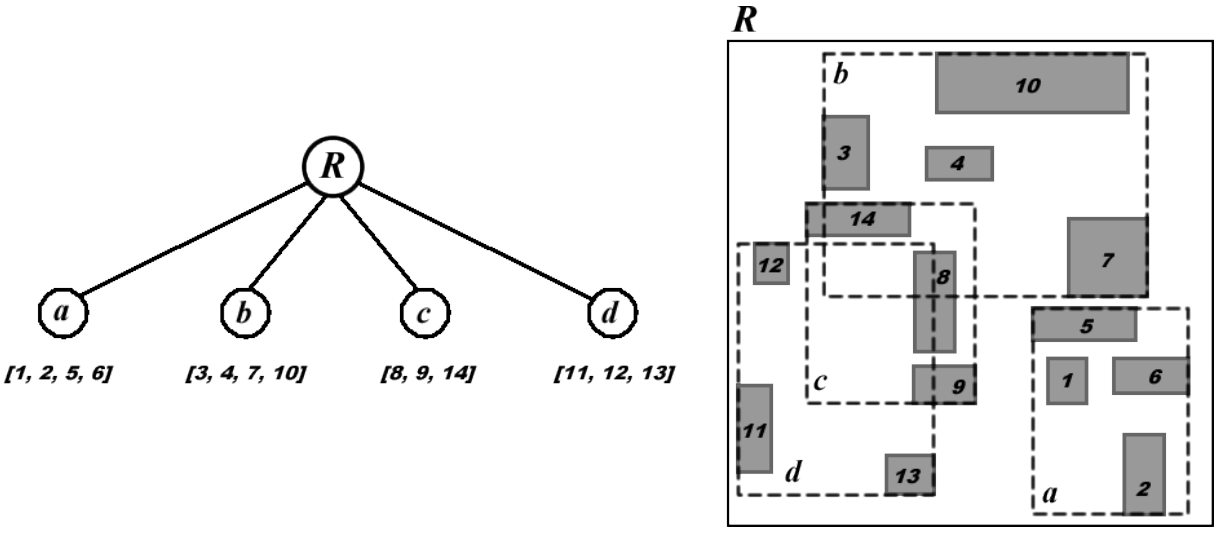

Figura 3.14: *R-Tree*

La Figura 3.14 muestra un *R-Tree* con $m = 2$ y $M = 4$. La BD indexada C contiene 14 objetos. Los rectángulos correspondientes a las hojas a, b, c y d están representados en el gráfico de la derecha mediante líneas de puntos. Las propiedades enumeradas anteriormente se conservan ante cualquier inserción o eliminación. Cabe resaltar que la estructura se mantiene balanceada aún cuando la distribución espacial de los objetos no sea uniforme. Este no es el caso de árboles con partición del espacio, como el *QuadTree*, donde algunas ramas del árbol pueden ser profundas, correspondientes a regiones con una alta densidad de rectángulos, mientras que otras podrían ser superficiales cuando el área es poco poblada.

El número máximo de entradas M en un nodo, depende del tamaño de las entradas a registrar $size(E)$ y de la capacidad de la página del disco $size(P)$: $M = size(P) / size(E)$. La cantidad mínima de entradas por nodo m, varía entre 0 y $M/2$ y se define en función de la estrategia de división de nodos a seguir.

Un *R-Tree* de aridad M y profundidad d, indexa como máximo M^{d+1} objetos. A su vez, para N elementos la profundidad de un *R-Tree* estará entre $log_m(N) - 1$ y $log_M(N) - 1$ niveles. El valor exacto depende de la capacidad ocupada de cada nodo.

Para ilustrar la eficiencia en tiempo del *R-Tree*, asumamos que el tamaño de la página es 4KBytes, el tamaño de cada entrada es 20 bytes (16 bytes para el MBB y 4 bytes para el oid) y m se establece en el 40 % de la capacidad de la página. Por tanto, $M = 4096\ div\ 20 = 204$ y $m = 81$. Entonces si el *R-Tree* tiene profundidad 1, podrá indexar hasta 41.616 objetos, mientras que un *R-Tree* de profundidad 2 tendrá capacidad para indexar hasta 8.489.664 objetos. Entonces, para una colección de un millón de objetos, sólo se necesitan tres accesos a páginas de disco para atravesar el árbol y un acceso complementario para obtener el registro donde se almacena el objeto. Este ejemplo muestra que incluso para colecciones muy grandes, sólo se necesitan unas pocas lecturas a disco para recorrer del árbol y acceder a los objetos.

A diferencia de los SAM de partición del espacio vistos anteriormente, un objeto aparece en una, y sólo una, de las hojas del árbol. Sin embargo, los rectángulos asociados con los nodos

internos no constituyen una partición exacta del espacio y por lo tanto, pueden superponerse. En la Figura 3.14, se muestra la superposición de los rectángulos b, c y d (que corresponden a hojas distintas).

Consultas al R-Tree

En esta sección se presenta el algoritmo de consulta por punto. El algoritmo para consultas de ventana es casi idéntico. La función de búsqueda por punto se realiza en dos pasos.

Primero se visitan todos los hijos de la raíz cuyos (hiper) rectángulos asociados contienen al punto P. Debido a que varios rectángulos pueden superponerse, es posible que el punto consultado esté contenido en la intersección de varios rectángulos. En ese caso todos los subárboles correspondientes a dichos rectángulos deberán ser visitados.

El proceso se repite en cada nivel del árbol hasta que se alcanzan las hojas. Para cada nodo N visitado, pueden ocurrir dos situaciones:

- En un nodo dado, ningún rectángulo de entrada contiene el punto y la búsqueda en ese subárbol termina. Esto puede ocurrir incluso si P cae en el rectángulo asociado al nodo. En este caso se dice que P cae en el llamado espacio muerto del nodo.
- P está contenido en los rectángulos de una o varias entradas. En este otro caso, hay que visitar cada subárbol asociado.

A diferencia de los SAM que dividen el espacio, durante una búsqueda sobre un $R - Tree$, puede ser necesario recorrer varios caminos desde la raíz hasta las hojas.

En el segundo y último paso, el conjunto de hojas producidas en el primer paso se escanea secuencialmente y se leen las páginas correspondientes a las entradas de cada hoja cuyo MBB contiene a P.

Inserciones y eliminaciones en el R-Tree

Para insertar un objeto, primero se recorre el árbol desde la raíz hascia las hojas, de forma similar a una búsqueda. En cada nivel, puede suceder que se encuentre una entrada cuyo rectángulo asociado contenga el MBB del objeto y entonces se continúa por el subárbol correspondiente. O podría ser que no exista tal entrada. Entonces se elige de las entradas la que requiera la mínima ampliación de su dr, se modifica su rectángulo y se visita el subárbol asociado siguiendo el mismo procedimiento. Cuando se llega al último nivel, si la hoja no está llena se agrega una nueva entrada [MBB, oid] a la página asociada a la hoja. Si el rectángulo de la hoja tiene que agrandarse, el nodo debe actualizarse con el nuevo valor de dr. Esto podría implicar la propagación de la ampliación de los rectángulos hacia arriba en el árbol; en el peor de los casos hasta la raíz.

Si la hoja l en la que se debe insertar el objeto está llena, se produce una *división*. Entonces se crea una nueva hoja l' y las entradas $M + 1$ se distribuyen entre l y l'. Cuando finaliza la división, se actualiza la entrada de la hoja antigua l en su nodo padre f, y se inserta una nueva entrada en f correspondiente a la nueva hoja l'. Si f está lleno, se aplica el mismo mecanismo de división sobre el nodo f. En el peor de los casos, este proceso de división se propaga por el árbol hasta la raíz. La profundidad del árbol se incrementa en 1 cuando se produce una división de la raíz.

La eliminación de una entrada e de un ***R-Tree*** se lleva a cabo en tres pasos:

1. encontrar la hoja H que contiene e,
2. eliminar e de H, y
3. reorganizar el árbol si es necesario.

La reorganización del árbol es la parte difícil del algoritmo. Ocurre cuando, después de eliminar una entrada de un nodo N, dicho nodo queda con menos de m entradas. Un solución

simple es eliminar el nodo y volver a insertar los elementos correspondientes a las $m - 1$ entradas.

Ejemplo del R-Tree aplicado en los experimentos

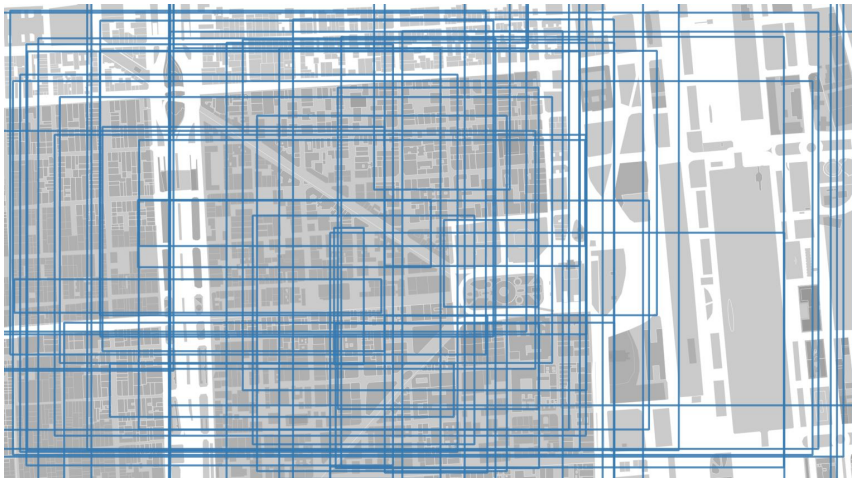

Figura 3.15: *R-Tree desplegado en el microcentro de Bs. As.*

En la Figura 3.15 se muestra el gráfico de las divisiones creadas por los rectángulos de los distintos niveles del *R-Tree* utilizado en este trabajo ($MeTree$, en realidad), obtenido en base a la distribución espacial de puntos de interés utilizados en los experimentos, correspondientes al microcentro de Buenos Aires.

Si bien el área de los índices espaciales está más estabilizada debido al muy buen funcionamiento de los métodos ya desarrollados, se siguen generando nuevos aportes tales como se muestra en [69] y [77].

Capítulo IV

MODELO MÉTRICO-ESPACIAL

4.1. Introducción

El modelo espacial ha aumentado su uso significativamente desde la década de los 90', principalmente como soporte para los Sistemas de Información Geográfica (SIG). Con el crecimiento de Internet, la cantidad de objetos asociados a una ubicación espacial se ha incrementado exponencialmente. Los dispositivos móviles han permitido la generación de cantidades enormes de datos georeferenciados que se comparten a través de la Web; principalmente imágenes, cadenas o textos con ubicación. Por ejemplo, fotos de personas, paisajes, animales y contrucciones, nombres de lugares o personas, comentarios o descripciones de lugares y eventos. Muchos de ellos se registran en algún tipo de sistema de información geográfica, o simplemente se guardan asociados a una ubicación espacial. Además, la cámaras de video de vigilancia (video+ubicación) son cada vez más comunes en las ciudades. En todos estos casos existen objetos que para su consulta requieren ser comparados por similitud y para los cuales tiene sentido también hacer referencia a su ubicación espacial. Por ejemplo, en un mapa de un país con fotos de edificios y monumentos importantes, sería de interés, dada una imagen de una construcción, encontrar las imágenes similares dentro de un área particular. También, en lugar de una imagen, simplemente buscar por similitud el nombre de un punto de interés, espacialmente más cercano

al lugar donde uno se encuentra. Estas consultas implican una búsqueda donde se tiene en cuenta tanto un aspecto espacial como la similitud con el elemento consultado, y son cada vez más comunes.

Como se expuso anteriormente, existen algunos índices que combinan similitud y espacio, pero diseñados exclusivamente para resolver búsquedas de documentos. En este capítulo se define el modelo *Métrico-Espacial*, este incluye todo tipo de objetos que pueda ser comparado por similitud (imágenes, texto, cadenas, etc.), y se presentan dos métodos de acceso, uno orientado a puntos y el otro a cualquier tipo de objeto geométrico, los cuales permiten realizar consultas métrico-espaciales con mayor eficiencia.

Se analiza con mayor profundidad el ejemplo en el cual se registran los logos de los negocios de una ciudad junto a su ubicación espacial. Algunas consultas métrico-espaciales que tienen sentido son:

1. Encontrar todos los negocios cuyo logo es similar a uno dado, dentro de una zona determinada de la ciudad (consulta por similitud por rango y por intersección espacial).

2. Encontrar el negocio más cercano a un punto, cuyo logo es similar a uno dado (consulta por similitud por rango y por vecino más cercano espacial).

3. Hallar el negocio cuyo logo tiene mayor parecido a uno dado y se encuentra en un área determinada (consulta del vecino más cercano respecto a la similitud y por intersección espacial).

4. Hallar los diez negocios con logo con mayor parecido a uno dado, que se encuentran en un área determinada (consulta de los 10 vecinos más cercano respecto a la similitud, y por intersección espacial).

Estas consultas no pueden ser resueltas eficientemente con índices métricos porque no tienen en cuenta el aspecto espacial, ni con índices espaciales, porque es necesaria la comparación por similitud. Así como se ha hecho anteriomente con la integración de los modelos temporal y

espacial dando lugar a los índices *Espacio-Temporales*, es necesario en este caso definir un modelo donde se combinen los modelos métrico y espacial, permitiendo utilizar métodos de acceso adecuados para realizar este tipo de consultas eficientemente.

4.2. Modelo Métrico-Espacial

Sea S el conjunto de objetos válidos a ser consultados por similtud y U el universo de objetos con sus ubicaciones, el modelo Métrico-Espacial se define mediante el par (U, d), donde todo $o \in U$, es un objeto compuesto por dos atributos: $s(o) \in S$, componente métrico del objeto, y $e(o)$, su aspecto espacial. La función métrica d, es la medida de disimilitud y está definida como $d : SxS \rightarrow R^{+}$. Por ejemplo, si $o \in U$ es un punto de interés en un mapa, $s(o)$ podría ser una cadena representando nombre o una foto del mismo, mientras que $e(o)$ será el punto (u objeto geométrico) indicando su ubicación (o forma espacial).

Es importante definir las propiedades que caracterízan los problemas que resuelve el *Modelo Métrico-Espacial*.

4.2.1. Caracterización de un problema Métrico-Espacial

Las situaciones problemáticas que este nuevo modelo de bases de datos permite resolver, se caracterizan a través de los siguientes puntos:

1. No tiene sentido realizar búsquedas exactas sobre los objetos, es decir, los elementos de la base de datos no tienen una clave que se pueda utilizar en la búsqueda, o la clave existe pero no está almacenada en la base de datos al momento de consultar, o existe y se encuentra registrada, pero la consulta en sí no contiene la clave. Por ejemplo, en una base de datos de música, cada canción puede contener una clave que la identifica, pero se puede requerir buscar una canción que sea similar a un fragmento de canción dado, justamente porque no se conoce su identificador.

2. Los objetos poseen información de ubicación o forma espacial asociada. Ésta representa dónde se encuentra el objeto en el espacio geográfico de búsqueda o cual es su forma, por ejemplo, las coordenadas geográficas de un punto de interés, o un terreno representado por un polígono.

3. Existen consultas en las cuales se requiere combinar las búsquedas por similitud con el aspecto espacial. Además podría ser requerido también realizar consultas por similitud puras o consultas espaciales puras.

4. La base de datos contiene una cantidad suficientemente grande de objetos, o bien, el tiempo de respuesta ante una consulta debe ser suficientemente reducido como para que no tenga sentido realizar una búsqueda secuencial.

Cuando se cumplen estas cuatro propiedades, la aplicación de este modelo conduce a una solución eficiente.

4.3. Consultas Métrico-Espaciales

Las consultas métrico-espaciales resultan de combinar los tipos de consultas por similitud con los tipos de consultas espaciales. Las clases de consulta por similtud más importantes son la búsqueda por rango y la de los *k* vecinos más cercanos. Respecto al aspecto espacial, existen muchas clases de consulta: intersección, inclusión, *k* vecinos más cercanos, adyacencia, etc. De todos éstos, la intersección es la más importante porque permite resolver casi todas las demás, por lo cual, es el tipo elegido en el desarrollo de esta tesis.

Una consulta por rango métrico e intersección espacial, se denota a través de la 3-upla $(q, r, g)_d$ y se define formalmente de la siguiente manera:

$$(q, r, g)_d = \{o \in X / d(s(o), q) \leq r \wedge intersects(e(o), g)\} \tag{4.1}$$

siendo $X \subseteq U$, la base de datos, q el aspecto métrico de la consulta, r el radio de búsqueda que representa el valor máximo de disimilitud aceptada, y g su aspecto geométrico (punto, polilínea o polígono).

Por otro lado, si la consulta es de los k vecinos más cercanos respecto a su similitud e intersección espacial, se denota como $NN_k(q, g)$. Esta devolverá el mayor conjunto $A \subseteq X$ tal que:

$$\begin{aligned} &(|A| \leq k) \wedge \\ &(\forall o \in A, y \in (X - A) : intersects(e(o), g) \wedge \\ &(intersects(e(y), g)) \implies d(q, s(o)) \leq d(q, s(y))) \end{aligned}$$

Una forma trivial de resolver una consulta métrico-espacial sin necesidad de comparar el objeto de la consulta con todos los elementos de la base de datos, es mediante el uso de dos índices, uno métrico y el otro espacial. Luego, ante una consulta por rango/intersección $(q, r, g)_d$, se procede de la siguiente manera:

1. Realizar la búsqueda por similitud $(q, r)_d$ sobre el índice métrico, devolviendo el conjunto L como resultado,

2. Realizar una búsqueda de los elementos que se intersectan con g, sobre el índice espacial, devolviendo como resultado el conjunto M,

3. Por último, realizar la intersección $L \cap M$ para obtener el resultado final.

La desventaja de esta solución es que no aprovecha la información métrica y espacial al mismo tiempo para el descarte de elementos. Una mejor estrategia es el diseño de índices que integren ambos aspectos y permitan consultas métricas, espaciales y métrico-espaciales.

4.4. Métodos de Acceso Métrico-Espaciales

Como parte de esta de tesis, se desarrollaron los índices métrico-espaciales $a)$ *MeTree*, que combina las estructuras *FHQT* y *R-Tree*, permitiendo indexar objetos con aspectos geométricos de cualquier tipo, y $b)$ *QuadFQTree*, que utiliza un *FHQT* como soporte del aspecto métrico y un *QuadTree* para el manejo de puntos en el espacio.

A continuación se describen las estructuras, métodos de construcción y algoritmos de consultas de ambos índices.

4.4.1. MeTree

El *MeTree* es un árbol r-ario en el cual cada nodo posee un rectángulo que contiene espacialmente a todos sus hijos, tal como el *R-Tree*; además, cada nodo tiene un intervalo métrico que representa el valor mínimo y máximo de las distancias de todos los elementos del subárbol hacia el pivote correspondiente al nivel del nodo (como un *FHQT*). La altura del árbol en principio es fija, y está determinada por la cantidad de pivotes con la cual se decide indexar. Sin embargo, esta cantidad puede ser extendida en cualquier momento, agregando nuevos pivotes y por lo tanto, nuevos niveles. Una hoja del árbol contendrá apuntadores a objetos similares, y además, cercanos espacialmente.

Un problema importante a resolver es que la base de datos podría contener elementos similares muy alejados espacialmente. Esta situación produciría en los niveles inferiores del árbol rectángulos demasiado grandes, afectando considerablemente a la eficiencia del índice. Para resolverlo se establecieron restricciones en los elementos para pertenecer a un mismo nodo, y se decidió permitir que tanto los intervalos métricos como los rectángulos de nodos hermanos, no sean excluyentes.

Este índice permite resolver consultas por similitud puras, espaciales puras y métrico-espaciales, soportendo tanto funciones de distancia discretas como continuas.

Estructura

El *MeTree* es un árbol r-ario donde cada nodo integra intervalos métricos con rectángulos espaciales. Formalmente se lo define de la siguiente manera:

- Todo nodo interior es una 4-upla $(i_m, f_m, rect, hijos)$, donde $i_m..f_m$ es el intervalo métrico definido a través de las distancias del pivote de su nivel hacia todos los objetos correspondientes a ese subárbol, $rect$ es el rectángulo que contiene espacialmente a todos los hijos de dicho nodo, e $hijos$ es una lista con punteros a sus nodos hijos.
- Salvo la raíz, todos los demás nodos interiores tienen un pivote asociado.
- Las hojas tienen una estructura similar a los nodos interiores, pero en lugar de tener una lista de $hijos$, poseen una lista de apuntadores a los elementos de la base de datos.

En la Figura 4.1 se presenta un ejemplo de la estructura de un *MeTree*. En la parte superior se muestran los nodos del árbol y los intervalos métricos de cada nodo, de acuerdo a las distancias de los elementos al pivote de cada nivel. Por razones de legibilidad, los rectángulos de cada nodo se muestran por separado, en la parte inferior de la figura. Como se puede ver, puede haber superposición tanto entre intervalos métricos de los nodos hermanos, como en los rectángulos que representan el espacio ocupado. La estructura es dinámica, permitiendo tanto altas como bajas y soportando espacialmente el indexado de puntos, polilíneas y polígonos. Respecto a las consultas por similitud, soporta cualquier objeto que pueda ser comparado a través de una función de distancia métrica.

Construcción del índice

Una vez seleccionados los pivotes, ya sea al azar o mediante algún algoritmo particular, se registran en el índice y se construye el árbol insertando uno a uno los elementos de la base de datos. El procedimiento es parecido al utilizado por el *FHQT* pero en lugar de distancias discretas fijas, utiliza intervalos de distancias permitiendo también funciones continuas. Los intervalos

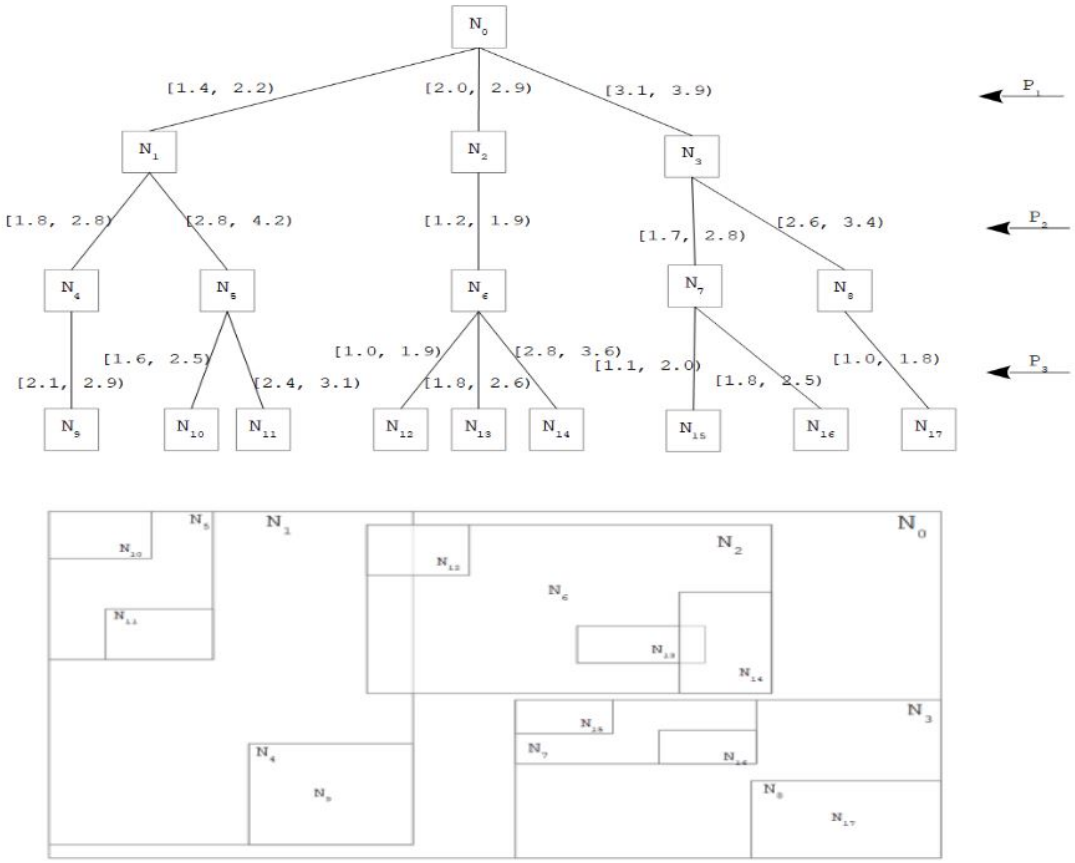

Figura 4.1: *Esquema de un MeTree*

son dinámicos porque pueden variar su tamaño cuando se agregan o eliminan elementos. Estos intervalos pueden tener algún nivel de superposición, es decir, no son excluyentes. Además, ante una inserción se actualizan los MBB (Minimum Bounding Box) de los nodos correspondientes a la rama en la cual se ubica el nuevo objeto. Es decir, si el rectángulo del nodo elegido para la inserción del nuevo elemento no contiene espacialmente al mismo, su MBB se amplía hasta incluirlo.

En las Figuras 4.2 y 4.3 se presentan los procedimientos correspondientes al algoritmo de inserción que utiliza el *MeTree*. Este algoritmo está organizado en dos partes. La primera calcula la firma del nuevo objeto y la diagonal máxima correspondiente a la raíz y la segunda realiza la inserción.

MeTree-Insercion $(o, raiz)$
1. calcular $firma$ de $s(o)$
2. $maxDiag$:=diagonal($raiz.rect$)
3. $nivel$:=1
4. **Insertar**(o,$raiz$,$firma$,$nivel$,tmi,$maxDiag$)

Figura 4.2: *MeTree: pseudocódigo del procedimiento principal de inserción*

El algoritmo posee cinco hiperparámetros que determinan algunos aspectos de su comportamiento:

1. prr: porcentaje de reducción del tamaño máximo de un rectángulo entre un nivel y el siguiente.

2. prm: porcentaje de reducción del tamaño máximo del intervalo métrico entre un nivel y el siguiente.

3. tmi: tamaño máximo del intervalo métrico inicial.

4. k: cantidad de pivotes (igual a la cantidad de niveles).

```
Insertar(o, nodo, firma, nivel, maxIntMetrico, maxDiag)
 1. minCosto:=∞
 2. for all (h ∈ nodo.hijos)
 3.   – cálculo de costo métrico
 4.   if firma_nivel Between h.i_m And h.f_m then
 5.     costoMetrico := 0
 6.   elsif firma_nivel <h.i_m then
 7.     if h.f_m - firma_nivel >MaxIntMetrico then
 8.       costoMetrico:=∞
 9.     else
10.       costoMetrico:= (h.i_m-firma_nivel)*100/MaxIntMetrico
11.   elsif firma_nivel - h.i_m >MaxIntMetrico then
12.     costoMetrico:=∞
13.   else
14.     costoMetrico:=(firma_nivel - h.f_m) * 100 / MaxIntMetrico
15.   – cálculo de costo espacial
16.   if costoMetrico <∞ then
17.     if contiene(h.rect,e(o)) then
18.       costoEspacial := 0
19.     else
20.       nuevoRect:=extenderMBB(h.rect,e(o))
21.       nuevaDiag:=diagonal(nuevoRect)
22.       if nuevaDiag >maxDiag then
23.         costoEspacial := ∞
24.       else
25.         costoEspacial := nuevaDiag*100/maxDiag
26.   costo:=(costoMetrico*α) + costoEspacial*(1-α)
27.   – actualizar el mínimo
28.   if costo<minCosto then
29.     minCosto:=costo
30.     hijoElegido := h
31. end for all
32. if minCosto < ∞ then
33.   actualizarIntervaloMetrico(hijoElegido,o)
34.   actualizarRectangulo(hijoElegido,o)
35. else
36.   hijoElegido:=agregarHijo(nodo,o)
37. if nivel=k then
38.   agregarHijo(hijoElegido,punteroBD(o))
39. else
40.   – continuar con el nodo del nivel siguiente
41.   maxIntMetrico:=maxIntMetrico*(1-prm/100)
42.   maxDiag:=maxDiag*(1-prr/100)
43.   Insertar(o, hijoElegido, firma, nivel+1, maxIntMetrico, maxDiag)
```

Figura 4.3: *MeTree: pseudocódigo del procedimiento* ***Insertar***

5. α : valor entre 0 y 1 que indica la importancia del costo métrico en relación al costo espacial, en el cálculo del costo total de inserción de un nuevo objeto en un nodo.

Ante una inserción se procede de la siguiente manera: Sea o el objeto a insertar, inicialmente se computa la $firma$ de o y luego para cada nodo hijo del nodo actual, se calcula el costo de agregar o en dicho nodo y se obtiene el costo mínimo. Si dicho costo es menor que $infinito$, significa que se encontró un nodo que puede contener a o sin violar las restricciones de tamaño máximo del intervalo métrico y del tamaño máximo del rectángulo. Entonces el objeto se añade al hijo de costo mínimo si es que existe alguno con costo menor a $infinito$, caso contrario se agrega un nuevo hijo.

El costo se calcula como la suma ponderada (a través del hiperparámetro α) del costo métrico y el costo espacial. El costo métrico es cero si $i_m \leq firma_{nivel} \leq f_m$, es decir, si la distancia del objeto al pivote del nivel se encuentra dentro del intervalo métrico del nodo. En caso contrario es igual al menor incremento del intervalo, expresado como porcentaje, necesario para incluir a $firma_{nivel}$. Si el intervalo aumentado es mayor que el tamaño máximo permitido, el costo es $infinito$.

Un procedimiento similar se utiliza para calcular el costo espacial. Si el aspecto espacial del objeto a insertar se encuentra ubicado dentro del rectángulo del nodo, el costo es cero. Para calcular el costo de aumentar el rectángulo para incluir al objeto se calcula el incremento necesario de su diagonal, y si éste es mayor al valor máximo de la diagonal para el nivel, nuevamente el costo será $infinito$.

Por ello, tanto el intervalo métrico como el rectángulo de cada nodo pueden aumentar de tamaño. El tamaño máximo inicial de los intervalos métricos es el hiperparámetro tmi, fijado de antemano, y los intervalos van disminuyendo su tamaño máximo de un nivel al siguiente, de acuerdo al porcentaje de reducción prm. En el caso de los rectángulos, la diagonal máxima también depende del nivel del nodo. La raíz contiene el espacio completo donde pueden estar los elementos. La diagonal máxima de cada nivel disminuye de a cuerdo a un porcentaje (prr) del valor máximo del nivel anterior.

La cantidad de niveles del árbol es igual a la cantidad de pivotes k. Se consideró la posibili-

dad de agregar niveles que sólo tomen en cuenta el aspecto espacial, pero no se tuvo en cuenta en este trabajo.

Consulta

Cuando se realiza una consulta métrico-espacial, se procede de la siguiente manera: en cada nivel del árbol, se seleccionan los subárboles hijos del nodo que se está procesando cuyos rectángulos se intersectan con el objeto espacial de la consulta, y donde el valor de su firma para dicho nivel se encuentra en el intervalo definido por el inicio del intervalo métrico menos el radio, y el fin métrico más el radio. Este procedimiento se repite hasta llegar al último nivel.

Para cada hoja no descartada, se comprueba primero (para cada elemento que contiene) si su distancia métrica a la consulta es menor o igual al radio, y posteriormente si posee superposición espacial. Estas dos verificaciones se realizan en forma secuencial, de tal manera de evitar realizar ambas si la primera ya no satisface la consulta. Si bien en esta tesis se optó por comparar primero por similitud, el algoritmo se podría configurar para tomar esta decisión a través de un hiperparámetro. En general las funciones de distancia métrica son más costosas que la intersección espacial, pero suelen tener mayor probabilidad de descarte, por lo tanto la decisión de cual verificación realizar primero debe ser considerada para cada caso particular.

El algoritmo de consulta por rango de similitud e intersección espacial se muestra en la Figura 4.4 y también está organizado en dos partes. La primera función calcula la firma del objeto que se consulta –para que la firma se compute una sola vez– y luego invoca a la función *Consultar*, que es la que realiza la búsqueda en sí. Esta segunda función recorre el árbol descartando las ramas que no cumplen con las restricciones espaciales o métricas y procesando las demás. Cuando se alcanzan las hojas, se realiza una búsqueda secuencial sobre los objetos contenidos en las mismas, devolviendo aquellos que cumplen ambas condiciones.

```
MeTree (q, r, g)_d
 1. calcular firma de q
 2. nivel := 1
 3. return Consultar(q, r, g, raiz, firma, nivel)

donde Consultar se define recursivamente como:

Consultar(q, r, g, nodo, firma, nivel)
 1. resultado:=∅
 2. if esHoja(nodo) then
 3.     for all (o ∈ nodo.hijos)
 4.         if (d(q, s(o)) <= r) then
 5.             if intersecta(e(o), g) then
 6.                 resultado :=resultado ∪ {o}
 7. else
 8.     for all (h ∈ nodo.hijos)
 9.         if (firma_nivel ∈ [h.i_m − r, h.f_m + r]) ∧ intersecta(h.rect, g) then
10.             resultado :=resultado ∪ Consultar(q, r, g, h, firma, n + 1)
11. return resultado
```

Figura 4.4: *MeTree: pseudocódigo de consulta por similitud por rango e intersección espacial*

Análisis del MeTree

El *MeTree* permite indexar cualquier tipo de objeto que pueda ser comparado a través de una función métrica para determinar su similitud y que posea una ubicación espacial (punto) o cualquier otra forma (polilínea, polígono o colecciones de elementos geométricos). Permite resolver consultas métrico espaciales y también consultas puras por similitud o espaciales puras.

En comparación con la solución trivial planteada en la Sección 4.3, su eficiencia es mucho mayor, como se observa en el siguiente capítulo, ya que descarta elementos por ambos criterios a la vez, mientras que en la solución trivial se realiza la búsqueda primero por uno de los criterios y luego por el otro y además se requiere realizar la intersección final entre los conjuntos resultantes. El *MeTree* posee complejidad computacional sublineal, entre $O(n)$ y $O(\log_r(n))$, donde r es la cantidad máxima de hijos que puede tener cada nodo. Sin embargo el cálculo de esta cantidad no es trivial, ya que depende de la distribución de distancias métricas, de la distribución espacial, y de la distribución resultante de combinar las distancias métricas con las ubicaciones espaciales para cada elemento. Esto es así porque dos elementos cercanos en

cuanto a similitud pero lejanos en cuanto a ubicación, no pueden agruparse en el mismo nodo. De la misma manera, dos elementos cercanos espacialmente pero muy distintos en cuanto a su similitud, tampoco estarán juntos, por lo cual al distribución de los elementos impacta sobre la cantidad de hijos que tiene cada nodo. Actualmente dicha cantidad no está limitada, y se puede aumentar o reducir indirectamente modificando los hiperparámetros.

Respecto al costo espacial, el *MeTree* tiene prácticamente los mismos requisitos a la variante para distancias continuas del *FHQT*, porque solo se debe agregar en cada nodo el rectángulo asociado. Como todo árbol (debe poseer más de un hijo por nodo), su crecimiento es exponencial en función de la cantidad de niveles, por lo cual es importante seleccionar la mínima cantidad de pivotes donde se brinde una eficiencia aceptable para las consultas.

4.4.2. QuadFQTree

El segundo método de acceso presentado en esta tesis se denomina *QuadFQTree*. Está diseñado para realizar consultas métrico-espaciales sobre una base de datos en la cual el aspecto espacial de los elementos representado a través de puntos. El *QuadFQTree* combina la estructura de un *QuadTree* con una variante del índice métrico *FHQT* permitiendo distancias continuas. Es un árbol r-ario en el cual cada nodo divide el espacio en cuatro cuadrantes conteniendo espacialmente a todos sus hijos y además, se encuentran dentro de un rango de distancias al pivote de cada nivel (como en un *FHQT*). Sin embargo la cantidad de hijos máxima de un nodo no es cuatro, donde puede haber cuadrantes duplicados para distintos rangos de distancias métricas debido a la existencia de elementos cercanos espacialmente, pero no similares. La altura del árbol es fija en principio, y está determinada por la cantidad de pivotes con la cual se decide indexar. Esta cantidad puede ser extendida en cualquier momento mediante el agregado de nuevos pivotes. Cada hoja del *QuadFQTree* contendrá apuntadores a objetos similares y además, cercanos espacialmente.

Estructura

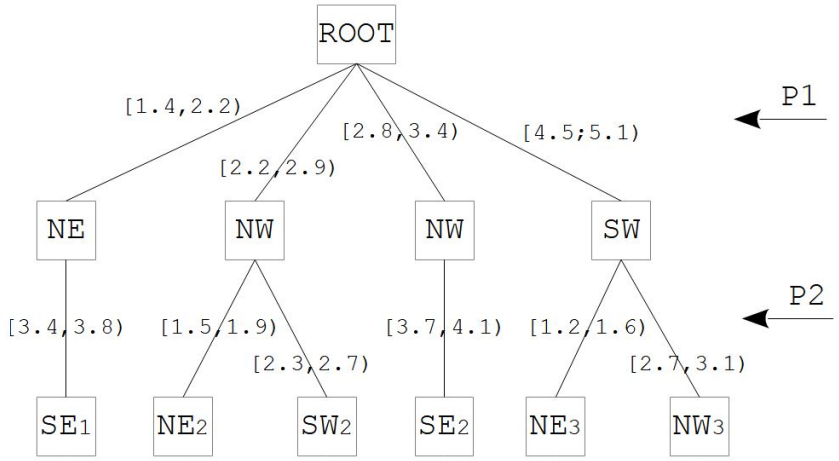

Figura 4.5: *Estructura de un QuadFQTree.*

Un nodo interior del QuadFQTree es una 5-upla $(i_m, f_m, c, rect, hijos)$, donde $i_m..f_m$ es el intervalo métrico y $rect$ el rectángulo donde se representa al cuadrante c. El atributo c puede tomar los valores NE, NW, SE y SW. A su vez, cada nivel del árbol (salvo la raíz) tiene asociado un pivote. En la Figura 4.5 se presenta un ejemplo de la estructura de un QuadFQTree con dos pivotes. Como se puede ver, la raíz contiene cuatro hijos, de los cuales dos de ellos representan el mismo cuadrante (NW, noroeste), pero con intervalos métricos diferentes (indicados a través de intervalos en los arcos correspondientes).

Construcción del índice

El algoritmo de inserción de un elemento en un *QuadFQTree* es más simple que el del *MeTree*, porque los rectángulos posibles correspondientes a los hijos de un nodo son siempre cuatro y no pueden cambiar de tamaño. Por otro lado, los intervalos métricos sí pueden ampliar su tamaño, siempre dentro del rango permitido definido por los hiperparámetros. Las Figuras 4.6 y 4.7 describen el pseudocódigo de inserción de un nuevo elemento correspondiente a este método de acceso.

El procedimiento de construcción del *QuadFQTree* posee tres hiperparámetros que modifican su funcionamiento:

1. prm: porcentaje de reducción del tamaño máximo del intervalo métrico entre un nivel y el siguiente.

2. tmi: tamaño máximo del intervalo métrico inicial.

3. k: cantidad de pivotes (igual a la cantidad de niveles).

QuadFQTree-Insercion $(o, raiz)$
1. calcular $firma$ de $s(o)$
2. $nivel$:=1
3. **Insertar**(o,$raiz$,$firma$,$nivel$,tmi)

Figura 4.6: *QuadFQTree: pseudocódigo del algoritmo principal de inserción*

Sea o el objeto a insertar, inicialmente se calcula su $firma$ y posteriormente para cada nodo hijo del nodo actual, se calcula el costo de agregar o en dicho nodo. En este caso, el costo considera sólo el aspecto métrico ya que los cuadrantes espaciales son fijos para cada nodo. De los nodos que cumplen con ambas restricciones, se selecciona el de costo mínimo y se agrega o al mismo. Si ninguno de los hijos satisface las restricciones, se agrega un nuevo hijo que pueda contener a o. Este nuevo nodo tiene asociado el cuadrante de o y como rango métrico, el intervalo [$firma_{nivel}$,$firma_{nivel}$]. Es decir, los intervalos métricos comienzan conteniendo sólo el valor de la firma del elemento para el nivel del nodo que se agrega. Cuando se insertan nuevos elementos en el mismo nodo, el intervalo métrico puede aumentar su tamaño.

Como en el *MeTree*, el tamaño máximo inicial de los intervalos métricos es el hiperparámetro tmi, fijado de antemano y los intervalos van disminuyendo su tamaño máximo de un nivel al siguiente, de acuerdo al porcentaje de reducción prm.

```
Insertar(o, nodo, firma, nivel, maxIntMetrico)
 1. cuadrante := calcularCuadrante(nodo.rect,e(o))
 2. minCosto:=∞
 3. for all (h ∈ nodo.hijos)
 4.   if h.c = cuadrante) then
 5.    – cálculo de costo métrico
 6.     if firma_nivel Between h.i_m And h.f_m then
 7.       costo := 0
 8.     elsif firma_nivel <h.i_m then
 7.       if h.f_m - firma_nivel >MaxIntMetrico then
 9.         costo:=∞
10.        else
11.          costo:= (h.i_m-firma_nivel)*100/MaxIntMetrico
12.     elsif firma_nivel - h.i_m >MaxIntMetrico then
13.       costo:=∞
14.     else
15.       costo:=(firma_nivel - h.f_m) * 100 / MaxIntMetrico
16.    – actualizar el mínimo
17.     if costo<minCosto then
18.       minCosto:=costo
19.       hijoElegido := h
20. end for all
21. if minCosto < ∞ then
22.   actualizarIntervaloMetrico(hijoElegido,o)
23. else
24.   hijoElegido:=agregarHijo(nodo,o, cuadrante)
25. if nivel=k then
26.   agregarHijo(hijoElegido,punteroBD(o))
27. else
28.   maxIntMetrico:=maxIntMetrico*(1-prm/100)
29.   Insertar(o, hijoElegido, firma, nivel+1, maxIntMetrico)
```

Figura 4.7: *QuadFQTree: pseudocódigo del algoritmo* ***Insertar***

Consulta

Ante una consulta métrico-espacial, se recorre el árbol seleccionando cada hijo del nodo actual cumpliendo con la restricción métrica en primer lugar, es decir, donde el valor de la firma para el nivel esté en el intervalo $[hijo.i_m - r, hijo.i_f + r]$, y luego cumpliendo con la superposición espacial. Esto se realiza en secuencia para disminuir la cantidad de cálculos espaciales donde la evaluación de la restricción métrica es mucho menos costosa porque no involucra el uso de la función de distancia (la firma se calcula sólo una vez, al principio).

```
QuadFQTree (q, r, g)_d
 1. calcular firma de q
 2. return Consultar(q, r, g, raiz, firma, 1)

donde Consultar se define recursivamente como:

Consultar(q, r, g, nodo, firma, nivel)
 1. resultado:=∅
 2. if not esHoja(nodo) then
 3.     for all (h ∈ nodo.hijos)
 4.         if (firma_nivel ∈ [h.i_m − r, h.f_m + r]) then
 4.             if intersecta(h.rect, g) then
11.                 resultado :=resultado ∪ Consultar(q, r, g, h, firma, n + 1)
 8. else
 4.     for all (o ∈ nodo.hijos)
 5.         if (d(q, s(o)) <= r) then
 6.             if intersecta(e(o), g) then
 7.                 resultado :=resultado ∪ {o}
12. return resultado
```

Figura 4.8: *QuadFQTree: pseudocódigo de consulta por similitud por rango e intersección espacial*

Una estrategia similar se utiliza al llegar a las hojas. Para seleccionar los elementos resultantes a partir de los candidatos, primero se comprueba que el elemento esté a menor distancia métrica que el radio, y luego, la superposición espacial. Nuevamente, estas dos comprobaciones se podrían configurar a través de un hiperparámetro para alterar su orden de acuerdo a cada caso particular. Si bien la intersección espacial tiene siempre un costo parecido, no se puede decir lo mismo de la función de distancia, la cual suele estar entre $O(n)$ y $O(n^2)$.

En la Figura 4.8 se muestra el algoritmo de consulta por intersección espacial y rango de similitud para el *QuadFQTree*.

Análisis del QuadFQTree

El *QuadFQTree* indexa objetos cuya ubicación espacial se representa a través de puntos. Dichos objetos además deben poder ser comparados a través de una función de distancia métrica para determinar su similitud. Este método de acceso también permite resolver consultas métrico espaciales, consultas por similitud puras y espaciales puras.

Si lo comparamos con la solución trivial planteada en la Sección 4.3, su eficiencia es mucho mayor ya que descarta elementos por ambos criterios a la vez, como el primer índice descripto en este capítulo. Respecto a su costo espacial, en la práctica ocupa menos espacio que el *MeTree* porque al trabajar con cuadrantes fijos por cada nodo, la cantidad de hijos por nodo suele ser aproximadamente entre el 50 % y 60 % de lo requerido por el primero.

4.4.3. Eliminación de Elementos

```
Eliminar(elemento)
 1. nodoHoja := obtenerNodoHoja(elemento)
 2. eliminarElemento(nodoHoja.hijos,elemento)
 3. if nodoHoja.hijos = ∅ then
 4.    nodo = obtenerNodoPadre(nodohoja)
 5.    eliminarNodo(nodo.hijos,nodohoja)
 6.    while (nodo.hijos = ∅) ∧ not esRaiz(nodo) do
 7.       nodoPadre = obtenerNodoPadre(nodo)
 8.       eliminarNodo(nodoPadre.hijos,nodo)
 9.       nodo := nodoPadre
```

Figura 4.9: *Algoritmo de Eliminación de elementos para el **MeTree** y el **QuadFQTree***

Cabe resaltar que dichos métodos comparten el mismo algoritmo de eliminación de elementos al ser ambos árboles r-arios con una estructura similar. El proceso consiste en dos partes: quitar el elemento propiamente dicho y recorrer el índice desde la hoja donde se encuentra el elemento hacia la raíz, eliminando todos los nodos que queden vacíos. El algoritmo de eliminación se muestra en la Figura 4.9. El costo de eliminación es logarítmico sobre la cantidad de elementos de la base de datos ya que como máximo consiste en un recorrido desde una hoja hasta la raíz, es decir, equivalente a la altura del árbol.

Capítulo V

EVALUACIÓN EXPERIMENTAL

En este capítulo se presenta la evaluación experimental de los índices diseñados y su comparación con la solución trivial descripta anteriormente. Inicialmente se describen los experimentos realizados y la metodología empleada, luego se muestran los resultados obtenidos y por último se realiza un análisis de los mismos.

5.1. Descripción de los Experimentos

A continuación se especifican las bases de datos utilizadas, la dimensionalidad intrínseca de los datos, la función de costo usada para medir la eficiencia de los índices y los radios de las consultas. Además, se realiza una descripción general de los experimentos llevados a cabo.

Bases de Datos Utilizadas

El primer paso consistió en seleccionar una base de datos para la cual tengan sentido las búsquedas métrico-espaciales. Para ello se exportaron los datos del plano del microcentro de Capital Federal a partir del Sistema de Información Geográfica OpenStreetMap, de libre acceso en la web. La Figura 5.1 muestra una sección del mismo. Los datos que contiene están organizados internamente en polígonos (manzanas, parques, plazas, etc), polilíneas (calles, avenidas,

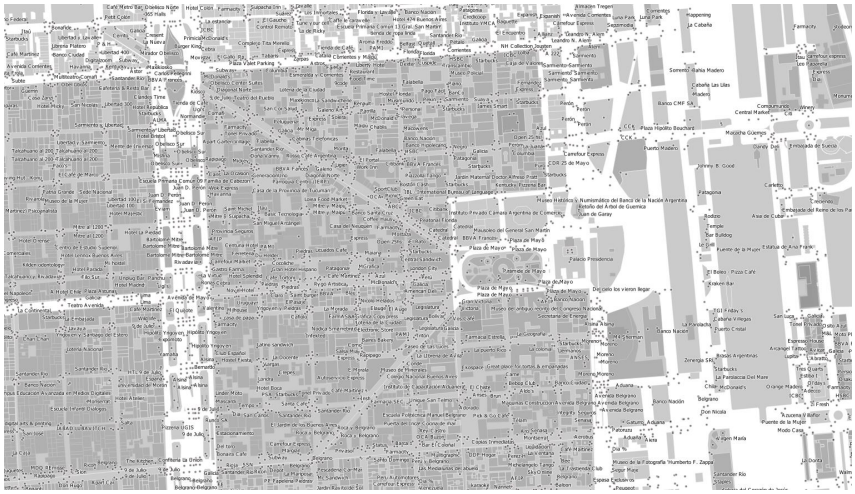

Figura 5.1: *Sección del mapa utilizado en los experimentos.*

etc) y 1.514 puntos de interés, correspodientes a comercios, monumentos, escuelas, estacionamientos, hospitales, etc., que fueron los datos de base para los experimentos.

Con el fin de contar con mayor cantidad de elementos para evaluar la eficiencia de los métodos de acceso desarrollados ante cambios en el tamaño de la base de datos, se generaron elementos adicionales hasta alcanzar los 50.000 puntos de interés, ubicados en forma aleatoria dentro área espacial definida.

En base a esos puntos, se crearon dos bases de datos distintas, *DBstr* y *DBimg*. La primera con cadenas como información métrica asociada a cada elemento y la segunda con vectores de caracteríscas que representan imágenes.

Para *DBstr* se utilizó el nombre de los puntos de interés descargados de OpenStreetMap y se completaron los 50.000 nombres con palabras descargas del diccionario de español del sitio de la Universidad de Michigan (https://umich.edu/), de tal manera de contar con cadenas sin duplicados.

Para completar la base de datos *DBimg*, se descargaron 50.000 imágenes desde los sitios CALTECH (www.vision.caltech.edu) y COCO (cocodataset.org) y se calcularon sus vectores característicos. En la Figura 5.2 se muestran algunos ejemplos de la variedad de imágenes utili-

zadas. En forma previa a la extracción de características se redujo el tamaño de las imágenes de tal manera que la mayor dimensión tenga 150 pixels y sin perder las proporciones, utilizando la herramienta FastStone Resizer 4.3.

Figura 5.2: *Ejemplo de la diversidad de imágenes de la base de datos.*

La paleta de colores fue generada en base a la paleta Amstrad CPC, conformada por 27 colores obtenidos a partir de distintas combinaciones de valores del espacio RGB.

Los vectores fueron normalizados con el fin de obtener resultados más representativos al compararlos mediante una función de distancia. El algoritmo de extracción de características y normalización se muestra la Figura 5.3.

Las funciones métricas utilizadas fueron la distancia de Levenshtein para las cadenas de la base *DBstr* y la distancia Euclidiana para la comparación de los vectores de imágenes en *DBimg*.

```
HistogramaImg(img,paleta)
 1. min := ∞
 2. vector := vectorCeros(tamaño(paleta))
 3. For all pixel (p ∈img)
 4.     For all color (c ∈paleta)
 5.       dist := euclidiana(p, c)
 6.       if dist <min then
 7.         min := dist
 8.         color = c
 9.     vector[pos(c)]:= vector[pos(c)] + 1
10. For i in 1..tamaño(paleta)
11.     vectorNormalizado[i]:= vector[i]*100/tamaño(img)
12. return vectorNormalizado
```

Figura 5.3: *Algoritmo de obtención de vectores característicos de imágenes.*

Dimensionalidad Intrínseca

Como parte de este trabajo se realizó un breve análisis de la Dimensionalidad Intrínseca del conjunto de vectores de la base *DBimg*. Para ello se calcularon las distancias Euclidianas de 500 mil pares de imágenes tomados al azar. En la Figura 5.4 se muestra la distribución de las frecuencias de los valores de distancia obtenidos.

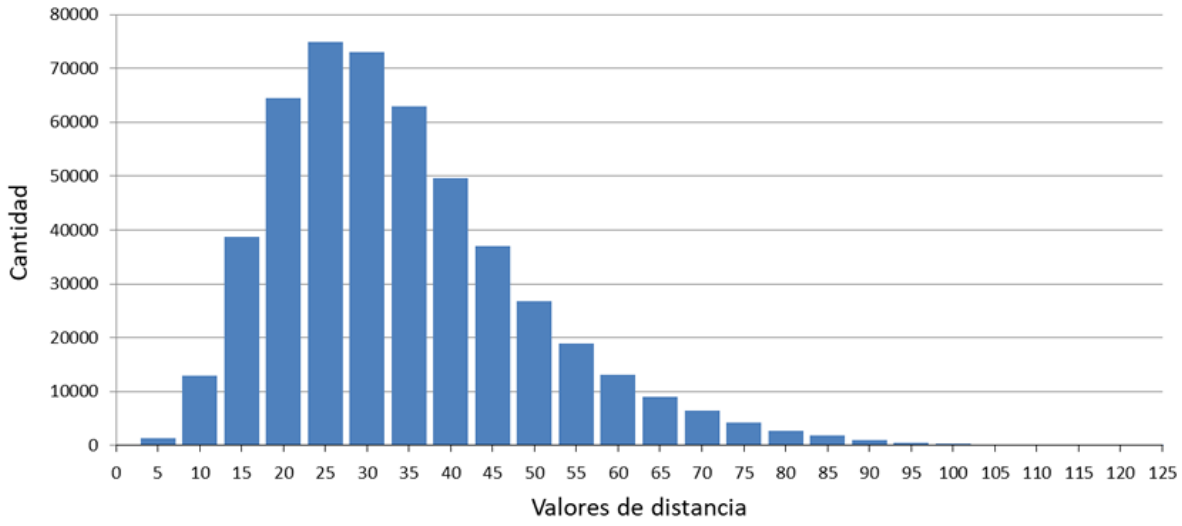

Figura 5.4: *Histograma de distancias para DBimg*

Para el cálculo de la Dimensionalidad Intrínseca, se utilizó la Fórmula 2.5 presentada en el Capítulo 2. A continuación se exponen los valores de algunas de las medidas que describen numéricamente al conjunto de datos y su distribución:

- Cantidad de evaluaciones de la función: 500.000
- Distancia mínima (excluyendo la distancia 0): 3,54
- Distancia máxima: 125,79
- Distancia promedio: 36,5
- Varianza: 227,13

Utilizando estos valores para la media y la varianza, la Dimensionalidad Intrínseca para este conjunto de datos y la distancia Euclidiana, es 2,93. Si bien no hay estudios –en nuestro conocimiento– que determinen analíticamente hasta qué valor de la Dimensionalidad Intrínseca los índices métricos mantienen su nivel de eficiencia, en la práctica se sabe que cuando está por debajo de 15 estas estructuras poseen un buen funcionamiento.

Cálculo del Costo

Para medir la eficiencia de los índices, teniendo en cuenta que están pensados para residir en memoria principal y combinan cálculos de la distancia métrica con computos de intersección espacial, se definió el costo de las consultas como la suma ponderada entre ambas, teniendo en cuenta su costo relativo. Es decir:

$$costo = m\alpha + i(1 - \alpha) \tag{5.1}$$

donde m es la cantidad de evaluaciones de la distancia métrica e i la cantidad de evaluaciones de la intersección espacial.

Luego se calculó el valor α para la distancia métricas de Levenshtein y Euclidiana y para la intersección espacial, realizando 100 mil cómputos de las mismas y estableciendo los pesos en forma proporcional al costo temporal de cada una. Los valores obtenidos fueron:

- Distancia de Levenshtein: α = 0,89
- Distancia Euclidiana: α = 0,96

En nuestros experimentos la distancia Euclidiana es más costosa que la distancia de Levenshtein. En la práctica, estos costos dependen principalmente del tamaño de las estructuras comparadas.

Consultas Realizadas

Para cada una de las bases de datos creadas, se generaron 100 consultas por rango métrico e intersección espacial, que fueron obtenidas seleccionando aleatoriamente 100 objetos de cada una de las bases *DBstr* y *DBimg*, realizándoles modificaciones y asignándole a cada uno de ellos, radios de búsqueda por similtud y polígonos de consulta espacial de diferentes tamaños, siempre dentro del área de trabajo considerada.

Los valores de r utilizados como radios de consultas fueron generados al azar entre los siguientes rangos:

- $r > 0 \wedge r < 10$, para *DBstr*.
- $r > 20 \wedge r < 40$, para *DBimg*.

Para definir los rangos se tuvieron en cuenta los histogramas de distancias, tomando como criterio que las respuestas a las consultas no sean vacías y que tampoco abarquen todos los elementos de las bases de datos.

Experimentos Realizados

Los experimentos fueron realizados utilizando el *MeTree*, el *QuadFQTree* y la Solución Trivial sobre ámbas bases de datos. Las primeras pruebas fueron para comparar los costos de las consultas utilizando los nuevos índices versus la Solución Trivial, planteada en la Sección 4.3.

Posteriormente se realizó un análisis de variación del costo en función de la cantidad de elementos, ejecutando las consultas para 10 mil, 20 mil, 30 mil, 40 mil y 50 mil elementos. Otro experimento realizado tuvo como objetivo analizar el comportamiento de los índices ante el cambio en la cantidad de pivotes, haciendo pruebas con 5, 10, 15 y 20 pivotes. También se experimentó variando el orden de inserción de los elementos en el *MeTree*, ya que se sabe que dicho orden afecta a la familia de índices del R-Tree.

Por último, se realizó una comparación entre los dos nuevos índices.

5.2. Resultados sobre DBstr

A continuación se presentan los resultados obtenidos en los experimentos realizados utilizando los índices *MeTree* y *QuadFQTree* sobre la base de datos *DBstr*, aplicando Levenshtein como distancia métrica y la intersección del rectángulo de consulta con los elementos de la base de datos como función espacial.

5.2.1. MeTree

En esta sección se muestran los resultados específicos de los experimentos con el nuevo índice *MeTree*.

Comparación de costos de *MeTree* con la Solución Trivial en *DBstr*

En la Figura 5.5 se muestra la comparación de costos de 100 consultas utilizando el *MeTree* versus la Solución Trivial, ordenados en forma descendente a partir de la consulta de mayor costo. La cantidad de elementos fué 50 mil y se utilizaron 10 pivotes.

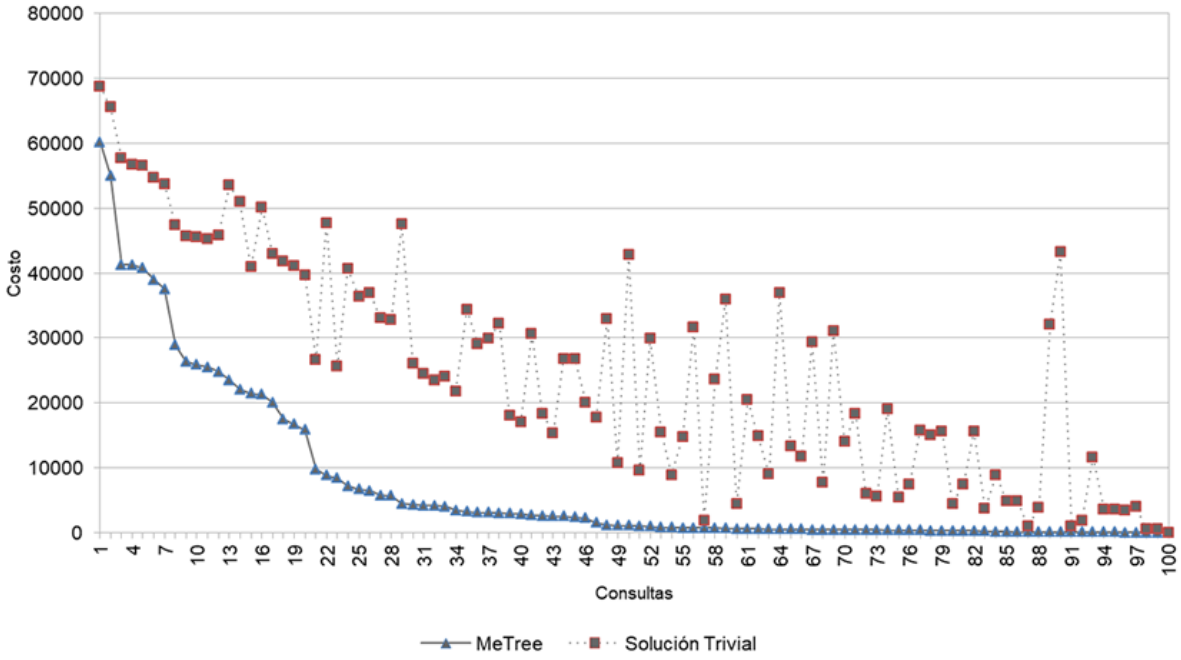

Figura 5.5: *Comparación de costos de MeTree con Solución Trivial, para 100 consultas en DBstr.*

El costo promedio de la solución trivial fué 24.859,67 y del *MeTree* 7.439,84. Esto significa que este nuevo índice resolvió las consultas en promedio en un 29,92 % del tiempo requerido por la solución trivial. Como se ve en la figura, sólo en unas pocas consultas el costo de ambas soluciones es similar, pero en la mayoría de los casos el *MeTree* es significativamente más veloz. Esta mejora en la eficiencia es notable principalmente cuando uno de los aspectos, el métrico o el espacial, descarta una gran cantidad de elementos y el otro tiene escasa capacidad de descarte.

Efecto del tamaño de la base de datos para *DBstr* con *MeTree*

Se comparó la solución trivial y el *MeTree* con 10 pivotes variando la cantidad de elementos entre 10 mil, 20 mil, 30 mil, 40 mil y 50 mil.

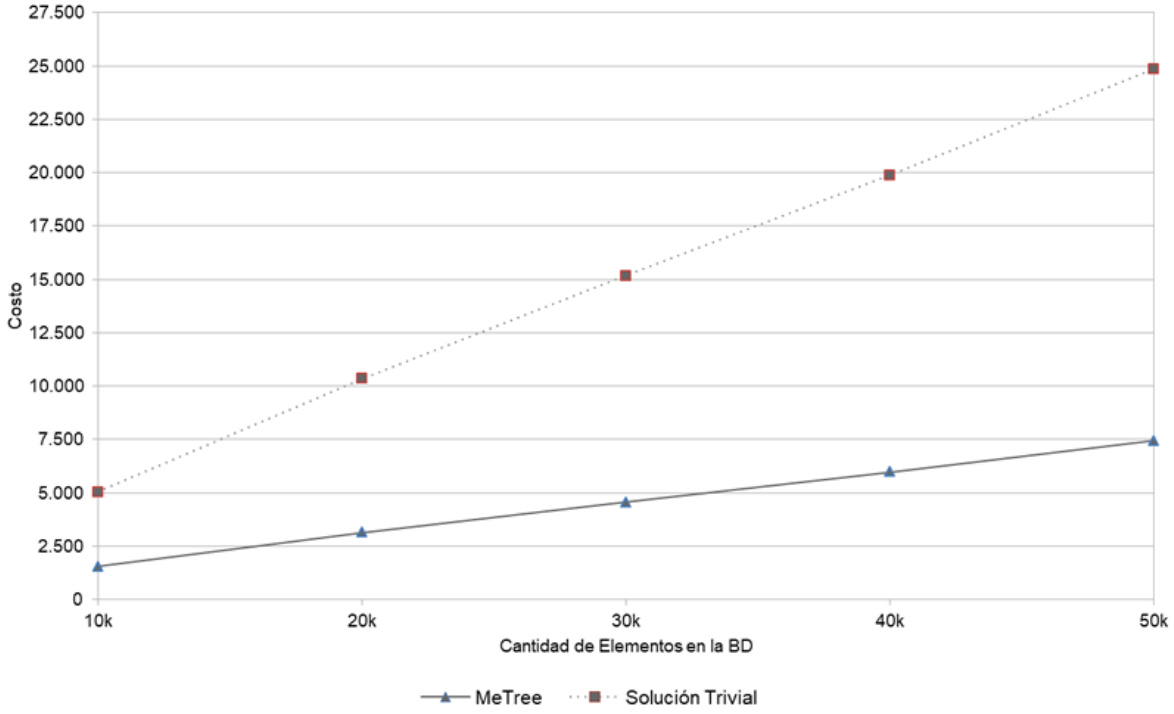

Figura 5.6: *Relación del Costo acorde a la cantidad de elementos de DBstr con MeTree.*

Analizado en detalle los valores que se muestran en la Figura 5.6 se observa que la tasa de crecimiento del costo no es un valor constante sino que disminuye muy lentamente. El orden de complejidad para la búsqueda por rango e intersección es $O(log_r n)$ en el mejor de los casos, donde r es la cantidad promedio de hijos de los nodos del árbol y $O(n)$ en el peor de los casos, ya que si el radio y el rectángulo de búsqueda son suficientemente grandes, la consulta devolverá todos los elementos de la base de datos.

En los experimentos, el costo del *MeTree* representó el 30,67 % del costo de la solución trivial para 10 mil elementos, el 30,38 % para 20 mil elementos, el 30,06 % para 30 mil elementos, el 29,99 % para 40 mil elementos y el 29,92 % para 50 mil elementos. Es decir, a medida

que aumenta la cantidad de elementos en la base de datos las diferencias de costo entre ambas soluciones se hacen cada vez mas grandes, por lo tanto se deduce que la aplicación de índices métrico-espaciales en grandes bases de datos podría producir un aumento significativo en la eficiencia de las consultas.

Efecto de la cantidad de pivotes para *DBstr* con *MeTree*

A continuación se presenta la comparación de costos entre la Solución Trivial y el *MeTree* para distintas cantidades de pivotes/niveles en *DBstr*. Las pruebas se realizaron para 50 mil elementos y utilizando la distancia de Levenshtein como se explicó anteriormente.

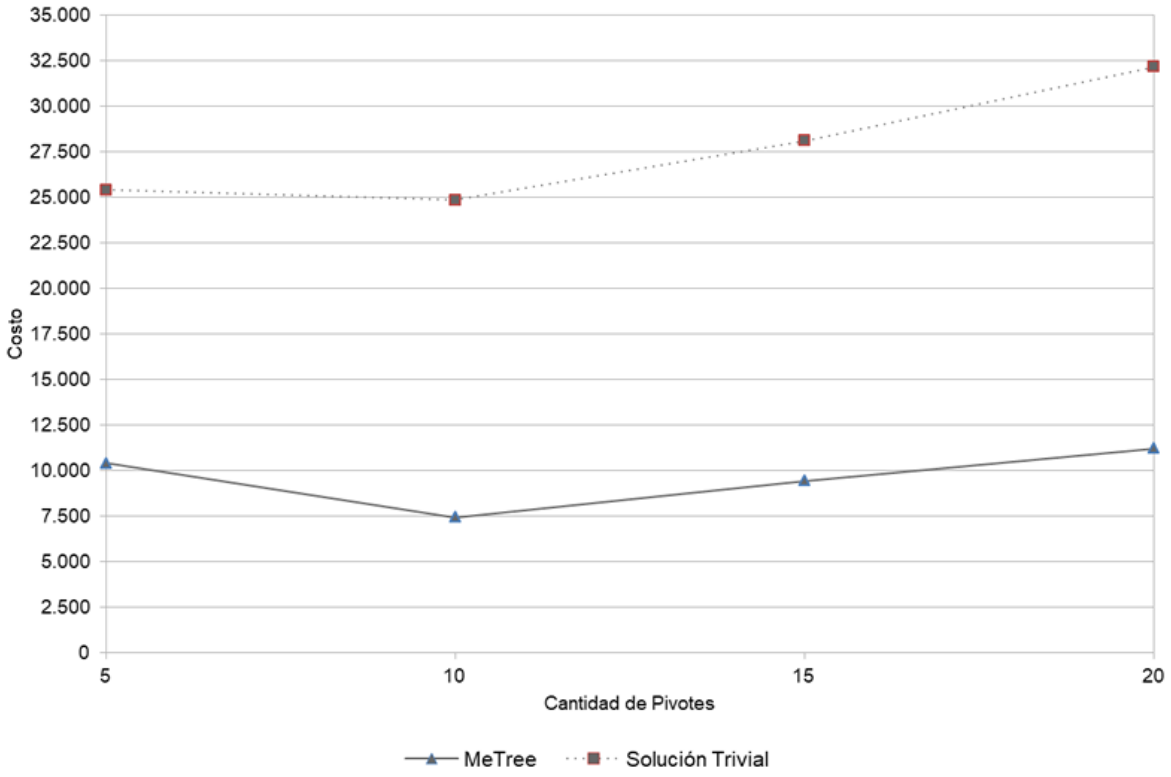

Figura 5.7: *Relación del Costo acorde a la cantidad de pivotes de DBstr con MeTree.*

La Figura 5.7 muestra como varía el costo en función de la cantidad de pivotes utilizada. Los experimentos se realizaron con 5, 10, 15 y 20 pivotes elegidos aleatoriamente y aplicados

al *MeTree* y a la solución trivial.

Inicialmente resultó inesperado que a partir de los 10 pivotes, el aumento de la cantidad de pivotes resulte en un incremento del costo de las consultas. Luego de analizar en profundidad los experimentos, se dedujo que este comportamiento se debe a dos razones. En primer lugar, la selección aleatoria de pivotes no produjo un aumento significativo en la eficiencia de las búsquedas por similtud. Por otro lado, se detectó un aumento importante en el costo espacial. Esto fue debido a que al agregar más niveles la cantidad de nodos del árbol se incrementa considerablemente y en los niveles superiores se produce una cantidad importante de nodos con el mismo rectángulo de extensión, pero con distinto rango métrico. Entonces ante una consulta, en dichos niveles se repite muchas veces la misma comparación con el rectángulo consultado. Este problema se podría solucionar alternando los aspectos métrico y espacial en los niveles. Es decir, los niveles impares sólo tienen en cuenta el aspecto métrico y los pares sólo el espacial, o viceversa. Por razones de espacio y tiempo, no se probó esta posible mejora al *MeTree*.

Efecto del orden de inserción de elementos en el *MeTree*

En la Figura 5.8 se comparan dos *MeTree* de 50k elementos y 10 pivotes insertando los elementos en orden inverso, para verificar si el orden de inserción de los elementos produce un efecto significativo en la eficiencia de este método de acceso. Esta prueba se realizó sabiendo que la familia de índices del *R-Tree* es sensible al orden de inserción de los datos.

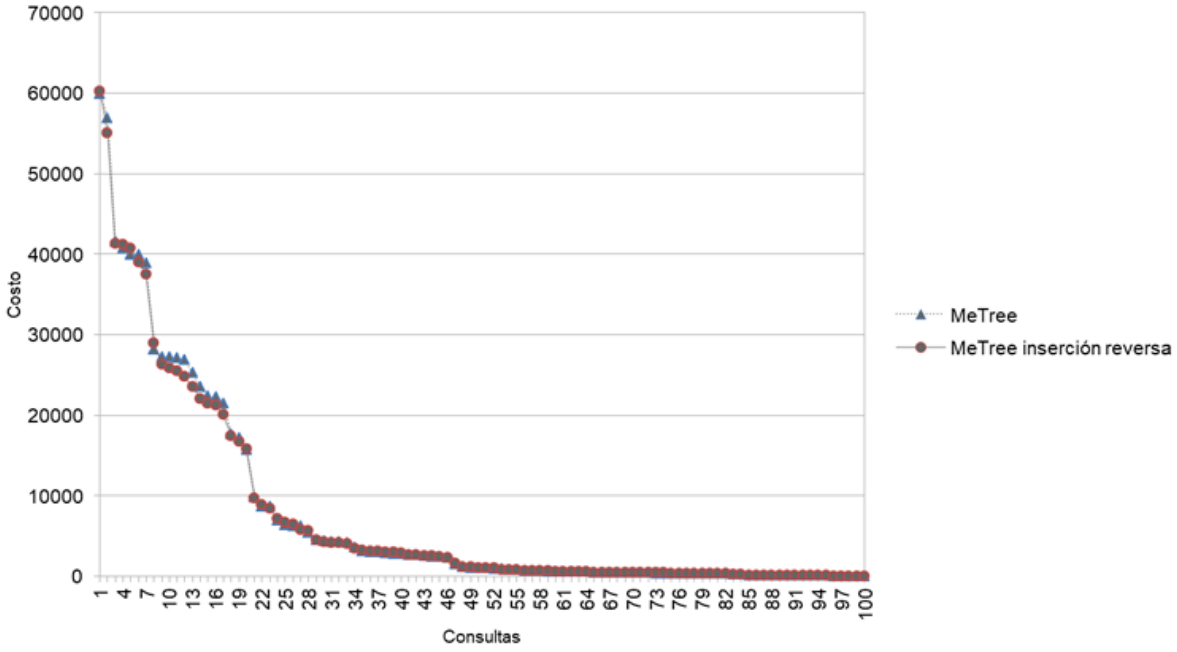

Figura 5.8: *Comparación de costos de MeTree y MeTree con inserción reversa en DBstr.*

Los resultados obtenidos no mostraron diferencias importantes en su performance, por esto se observa que el índice es robusto en este aspecto.

5.2.2. QuadFQTree

En esta sección se muestran los resultados específicos de los experimentos con el nuevo índice *QuadFQTree*.

Comparación de costos de *QuadFQTree* con Solución Trivial en *DBstr*

En la Figura 5.9 se muestra la comparación de costos de 100 consultas, *QuadFQTree* contra la Solución Trivial, ordenados en forma descendente desde la consulta de mayor costo. La cantidad de elementos fué de 50 mil y se utilizaron 10 pivotes.

El costo promedio de la solución trivial fue de 32.482,79 y del *QuadFQTree* de 7.935,64. Es decir, en promedio este índice requiere sólo una cuarta parte del costo de la solución trivial.

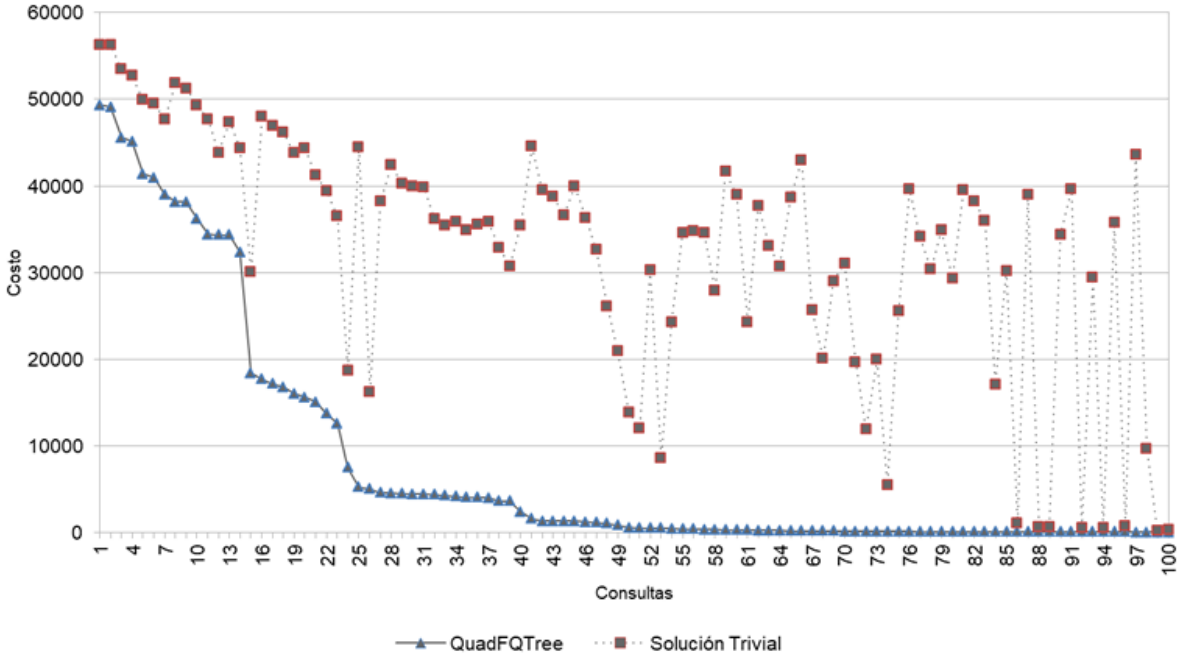

Figura 5.9: *Comparación de costos de QuadFQTree con Solución Trivial, para 100 consultas en DBstr.*

Efecto del tamaño de la base de datos para *DBstr* con *QuadFQTree*

Se realizó la comparación entre la solución trivial y el *QuadFQTree* con 10 pivotes y variando la cantidad de elementos entre los valores 10 mil, 20 mil, 30 mil, 40 mil y 50 mil.

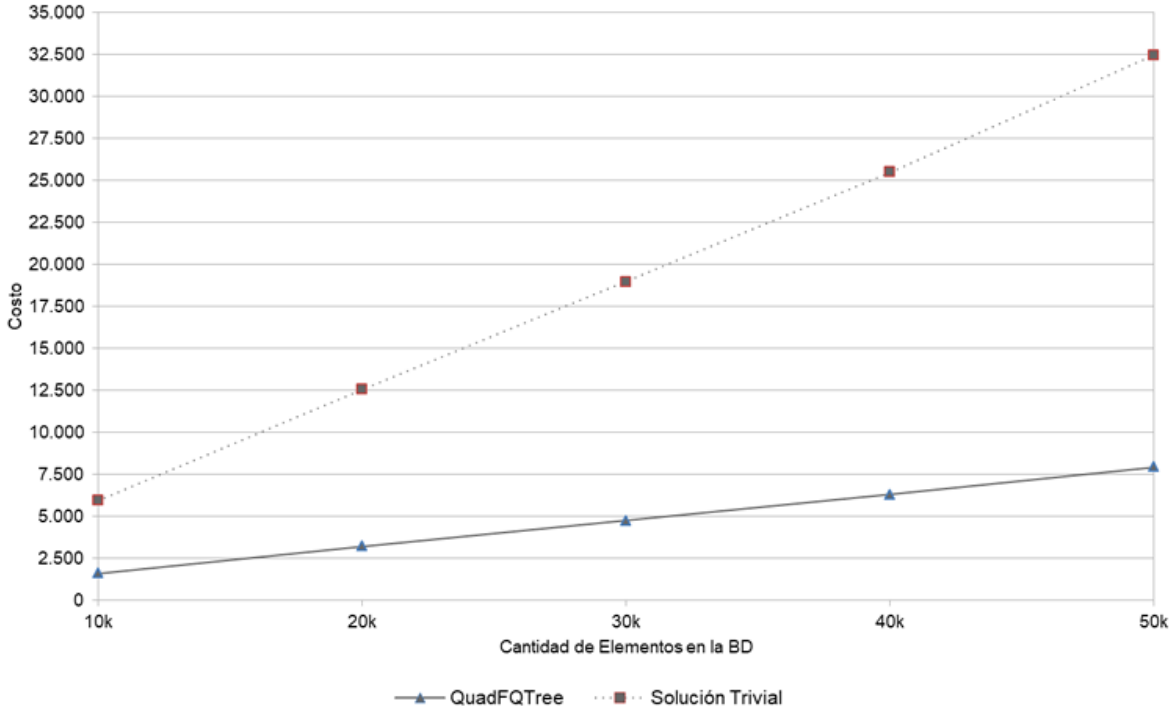

Figura 5.10: *Relación del Costo acorde a la cantidad de elementos de DBstr con QuadFQTree.*

En la Figura 5.10 se observa un factor de crecimiento que, si bien no es constante, dentro del rango considerado es alrededor del $0,16$ para el *QuadFQTree* y $0,66$ para la solución trivial.

De igual manera que el *MeTree*, el *QuadFQTree* produce mejoras sustanciales en las búsquedas métrico-espaciales en todos los casos analizados.

Efecto de la cantidad de pivotes para *DBstr* con *QuadFQTree*

A continuación se presenta la comparación de costos de la Solución Trivial y del *QuadFQTree*, utilizando 5, 10, 15 y 20 pivotes seleccionados al azar a partir de 50 mil elementos de la base de datos *DBstr*. En la Figura 5.11 se muestra como varía el costo para las 100 consultas ejecutadas con el *QuadFQTree*.

Como el *MeTree*, el costo del *QuadFQTree* disminuye al pasar de 5 a 10 pivotes, pero luego aumenta porque se suman niveles a la estructura los cuales no producen mejoras importantes en

el aspecto métrico (porque la selección de pivotes utilizada es aleatoria) y generan una cantidad mayor de comparaciones espaciales. Nuevamente, gran parte de estas comparaciones duplicadas se podrían evitar alternando los aspectos métrico y espacial a través de los distintos niveles del índice.

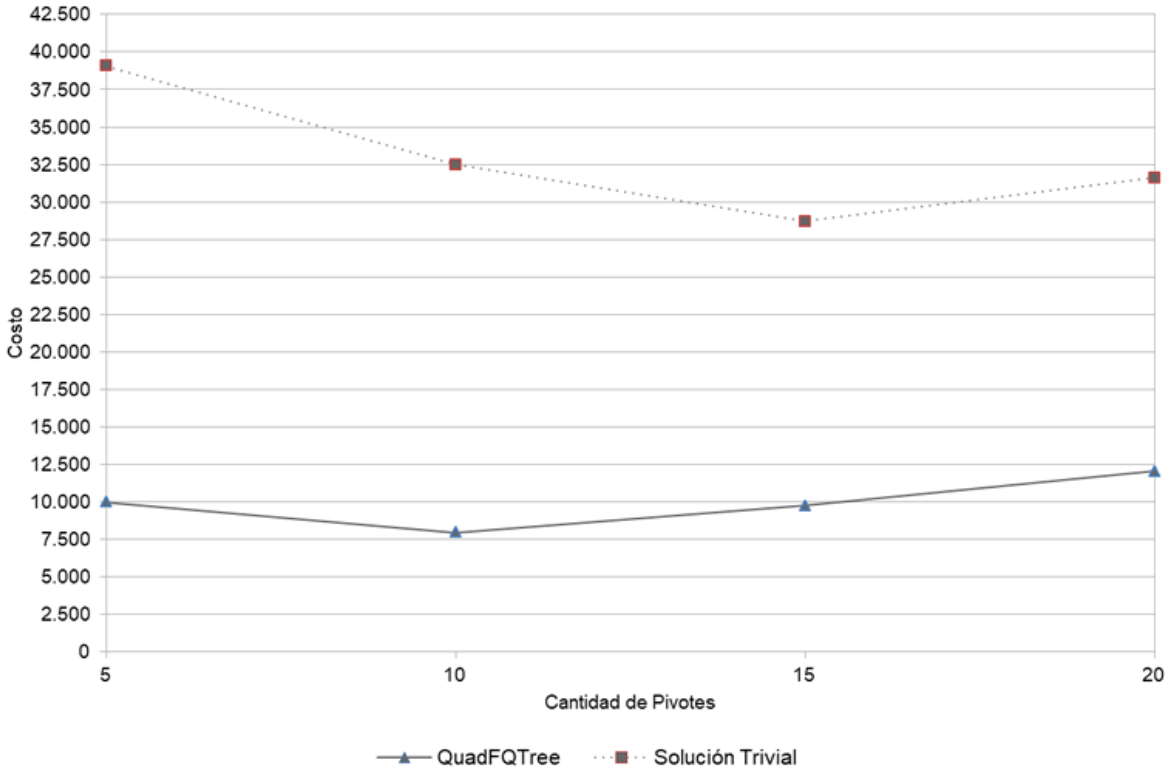

Figura 5.11: *Relación del Costo acorde a la cantidad de pivotes en DBstr con QuadFQTree.*

5.2.3. MeTree vs QuadFQTree

En esta sección se comparan los costos de la resolución de las consultas correspondientes al *MeTree* y *QuadFQTree* sobre la base de datos *DBstr*.

Comparación de costos de *MeTree* con *QuadFQTree* en *DBstr*

En la Figura 5.12 se observa la comparación del costo de *MeTree* con *QuadFQTree* sobre la base de datos *DBstr*. Se ejecutaron 100 consultas sobre 50 mil elementos utilizando 10 pivotes

y la distancia de Levenshtein.

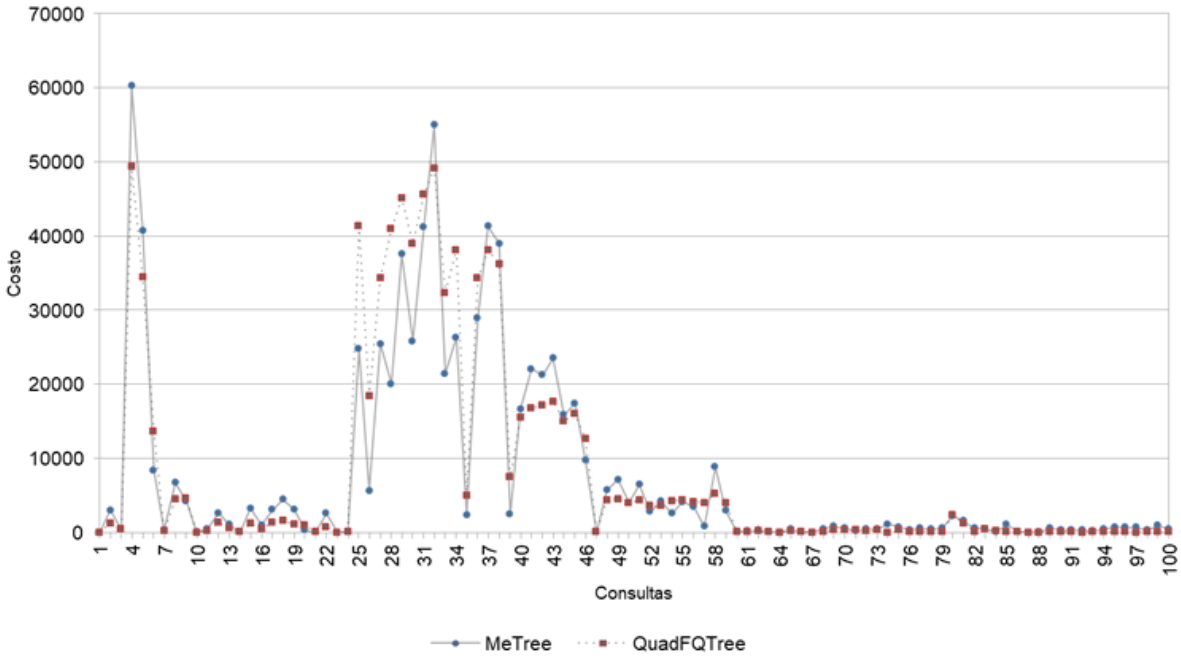

Figura 5.12: *Comparación de costos de MeTree con QuadFQTree en DBstr.*

El costo promedio del *MeTree* fué 7.439,8 y el del *QuadFQTree* 7.935,6. La diferencia de eficiencia entre ambos no es significativa y si bien en este experimento el *MeTree* obtuvo en promedio mejores valores, se observa que para algunas consultas el *QuadFQTree* es más eficiente.

Costo espacial del *MeTree* y del *QuadFQTree* en *DBstr*

Otro aspecto importante a tener en cuenta al comparar los índices es el espacio requerido por cada uno. En la Figura 5.13 se muestran las cantidades de nodos generados para el *MeTree* y el *QuadFQTree* en función de la cantidad de elementos de la base de datos y la cantidad de pivotes utilizada.

Índice	Cantidad de Elementos	Cantidad de Pivotes	Cantidad aprox. De Nodos
MeTree	10.000	10	51.000
QuadFQTree	10.000	10	28.000
MeTree	50.000	10	215.000
QuadFQTree	50.000	10	87.000
MeTree	50.000	15	509.000
QuadFQTree	50.000	15	329.000
MeTree	50.000	20	707.000
QuadFQTree	50.000	20	579.000

Figura 5.13: *Comparación de cantidad de nodos generados entre MeTree y QuadFQTree en DBstr.*

El *QuadFQTree* utilizó entre un $40\,\%$ y un $82\,\%$ del espacio requerido por el *MeTree* para la misma cantidad de elementos y pivotes, lo que le otorga una ventaja en este aspecto.

5.3. Resultados sobre DBimg

A continuación se presentan los resultados obtenidos en los experimentos realizados utilizando los índices *MeTree* y *QuadFQTree* sobre la base de datos *DBimg*, aplicando la distancia Euclideana como métrica para medir similitud, y la intersección del rectángulo de consulta con los elementos de la base de datos como función espacial.

5.3.1. MeTree

En esta sección se muestran los resultados específicos de los experimentos con el nuevo índice *MeTree*.

Comparación de costos de *MeTree* con la Solución Trivial en *DBimg*

En la Figura 5.14 se muestra la comparación de costos de 100 consultas entre el *MeTree* y la Solución Trivial, pero esta vez sobre la base de datos de imágenes *DBimg* representadas

mediantes vectores y utilizando la distancia Euclidiana. Las consultas se ordenaron en forma descendente por costo. La cantidad de elementos fué 50 mil y se utilizaron 10 pivotes.

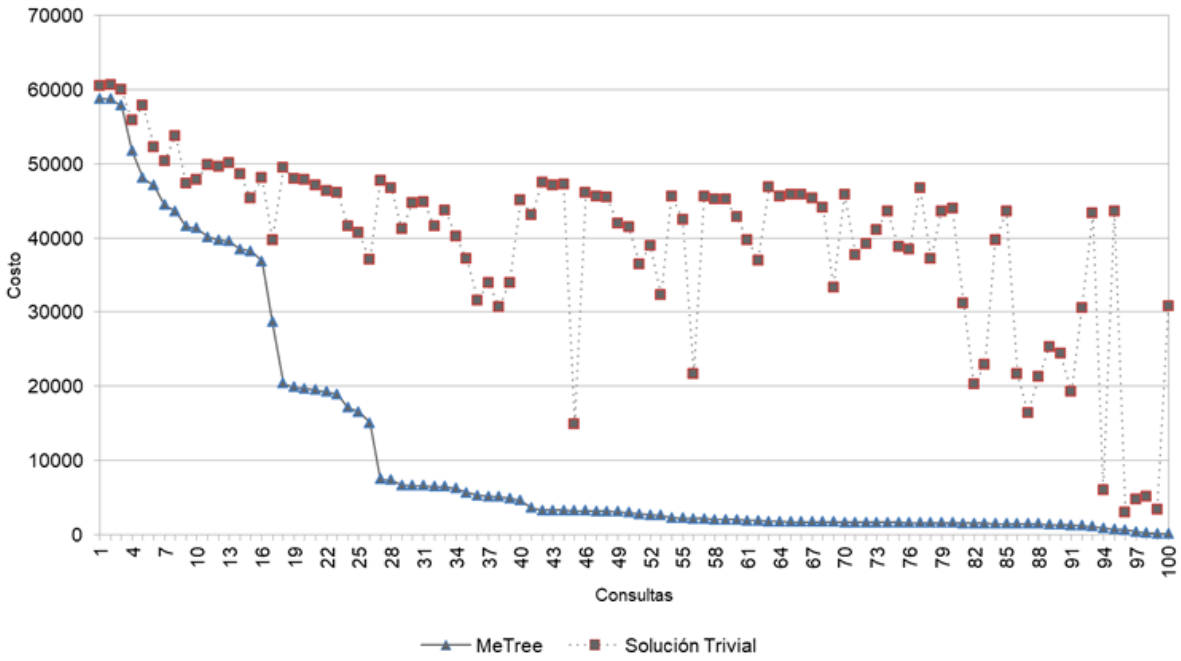

Figura 5.14: *Comparación de costos de MeTree con Solución Trivial, para 100 consultas en DBimg.*

El costo promedio de la solución trivial fué 39.586,73 pero utilizando el *MeTree* se redujo a 11.136,17, es decir 3,5 veces más eficiente aproximadamente. Es importante notar que cuando las consultas poseen un radio amplio y un polígono donde abarca la mayoría de los elementos de la base de datos como respuesta, la diferencia de costo no es importante. Pero ante consultas más selectivas, la eficiencia del *MeTree* es significativamente mayor

5.3.2. QuadFQTree

En esta sección se muestran los resultados específicos de los experimentos con el nuevo índice *QuadFQTree*.

Comparación de costos de *QuadFQTree* con Solución Trivial en *DBimg*

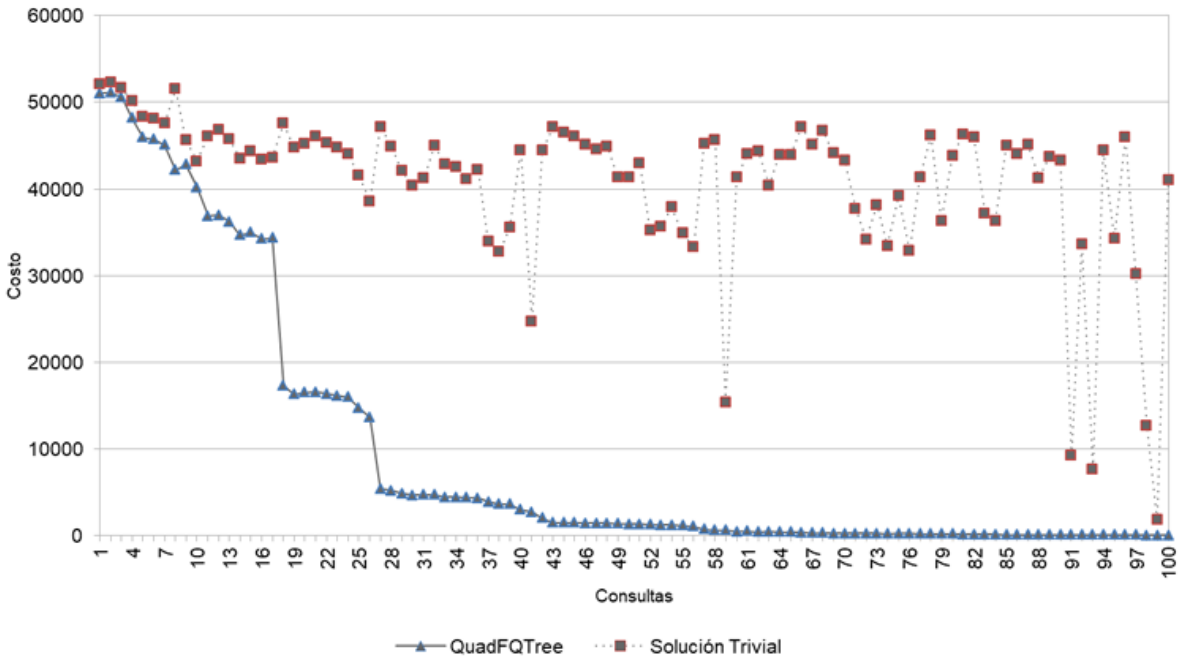

Figura 5.15: *Comparación de costos de QuadFQTree con Solución Trivial, para 100 consultas en DBimg.*

En la Figura 5.15 se muestra la comparación de costos de 100 consultas, *QuadFQTree* contra la Solución Trivial sobre la base de datos *DBimg*, ordenados en forma descendente desde la consulta de mayor costo. La cantidad de elementos fue de 50 mil, se utilizaron 10 pivotes y la función de distancia empleada fue la Euclidiana. El costo promedio de la solución trivial fué de 40.867,83 y del *QuadFQTree* de 9.464,73, es decir, más de cuatro veces más eficiente que la solución trivial.

5.3.3. MeTree vs QuadFQTree

En esta sección se comparan los costos de la resolución de las consultas correspondientes al *MeTree* y *QuadFQTree* sobre la base de datos *DBimg*.

Comparación de costos de *MeTree* con *QuadFQTree* en *DBimg*

En la Figura 5.15 se muestra la comparación de costos entre el *MeTree* y el *QuadFQTree* sobre la base de datos de imágenes. Se ejecutaron 100 consultas sobre 50 mil elementos con 10 pivotes elegidos al azar y utilizando la distancia Euclideana para comparar los verctores característicos de las imágenes.

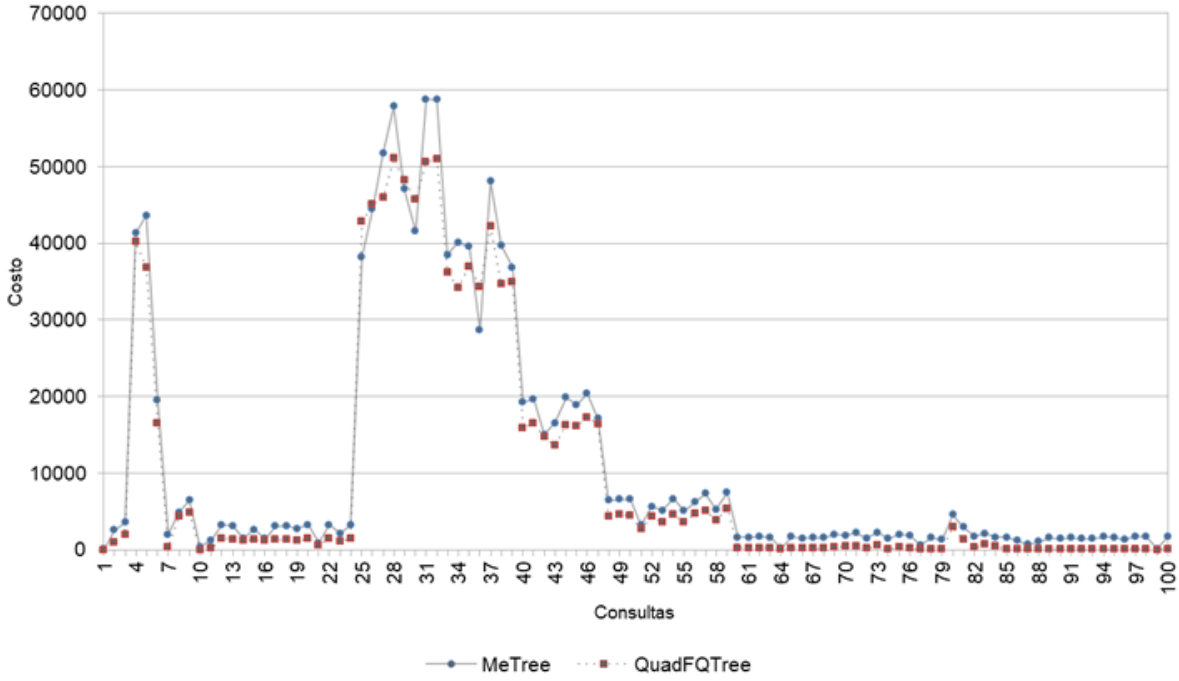

Figura 5.16: *Comparación de costos de MeTree con QuadFQTree en DBimg.*

El costo promedio del *MeTree* fué de 11.136,2 y el del *QuadFQTree* 9.464,7. A diferencia de los experimentos sobre la base de datos de cadenas, en este caso el *QuadFQTree* obtuvo un mejor rendimiento, resolviendo las consultas en un $85\,\%$ del tiempo requerido por el *MeTree*.

Estas diferencias de eficiencia entre ambos métodos parecen depender principalmente de la distribución del aspecto métrico de los objetos de la base de datos y de la relación de costos entre la función de distancia y la intersección espacial.

5.4. Conclusiones

En este Capítulo se presentó la evaluación experimental de los índices métrico-espaciales *MeTree* y *QuadFQTree* sobre las bases de datos *DBstr* y *DBimg* en comparación con la solución trivial, para 50 mil elementos y 10 pivotes. Luego se analizó la evolución del costo en función de la cantidad de elementos y también variando la cantidad de pivotes. Por último se realizaron comparaciones entre ambos índices y además se verificó el comportamiento del *MeTree* ante el cambio en el orden de inserción de los elementos.

En todos los experimentos estos nuevos métodos de acceso obtuvieron mejores resultados que la solución trivial, mostrando ser entre tres y cuatro veces más eficientes. Se verificó también, que esta diferencia se mantiene e incluso se amplía levemente al aumentar la cantidad de elementos de la base de datos.

El aumento de la cantidad de pivotes por otro lado, no siempre redunda en un aumento de la eficiencia. En estos experimentos uno de los factores importantes que tuvo como consecuencia este comportamiento fue la selección de pivotes al azar. Cuando los pivotes que se agregan cumplen una función similar a los ya existentes, no producen mejoras importantes en el aspecto métrico, suman niveles al árbol y aumentan el costo por comparaciones espaciales.

Respecto a la comparación de ambos índices, el *MeTree* obtuvo mejores resultados sobre la base de datos *DBstr* mientras que el *QuadFQTree* hizo lo propio cuando se utilizó *DBimg*, aunque las diferencias no fueron significativas. En cuanto al espacio utilizado, el *QuadFQTree* aventaja al *MeTree* requiriendo sólo entre un $40\,\%$ y un $82\,\%$ del ocupado por este último. Sin embargo, ambos índices tienen su uso específico ya que el *QuadFQTree* está orientado principalmente a puntos mientras que el *MeTree* permite cualquier tipo de objeto geométrico.

En conclusión, los dos nuevos índices muestran sus beneficios en la resolución eficiente de consultas métrico-espaciales.

Capítulo VI

CONCLUSIONES Y TRABAJO FUTURO

Actualmente existe una necesidad creciente de almacenamiento y procesamiento de objetos complejos con una ubicación o forma espacial. Esta necesidad es clara en aplicaciones combinando Sistemas de Información Geográfica con objetos multimedia por ejemplo. Uno de los problemas comunes en estas aplicaciones es el procesamiento de consultas teniendo en cuenta tanto el aspecto espacial como la similtud con el objeto consultado. Esta tesis se centró en el diseño de nuevos métodos de acceso y algoritmos para el procesamiento eficiente de consultas métrico-espaciales. En este capítulo se detallan las conclusiones y aportes de esta tesis, como también los principales problemas a abordar en el futuro.

6.1. Conclusiones

Debido a la necesidad cada vez más frecuente de desarrollar aplicaciones que resuelvan consultas métrico-espaciales, es de interés el uso de métodos de acceso como los desarrollados en esta tesis con el propósito de dar soporte a la implementación de tales aplicaciones. Éste es un campo de investigación relativamente reciente dando lugar a otros tópicos relacionados,

como por ejemplo, lenguajes de consulta o algoritmos de agrupamiento con soporte de tipos de datos métrico-espaciales.

Los métodos de acceso propuestos en esta tesis se basan en espacios métricos. Este enfoque permite generalizar avances anteriores que se limitaban a la indexación por similitud y espacial exclusivamente de documentos. De esta manera, cualquier objeto que pueda ser comparado a través de una función de distancia métrica, será posible de ser indexado para acelerar su búsqueda.

Los principales aportes de esta tesis se resumen a continuación:

- Se formalizó el *Modelo Métrico-Espacial*, se caracterizaron los problemas que resuelve, y se definieron los tipos de consultas asociados al mismo.

- Se diseñó el índice métrico-espacial *MeTree*, el cual en los experimentos realizados obtuvo un costo mucho menor a la solución trivial, resolviendo las consultas entre un 29,92 % (*DBstr*) y un 28,13 % (*DBimg*) del tiempo requerido por la solución trivial. El *MeTree* demostró ser robusto respecto al orden de inserción de los elementos, ya que en las pruebas de construcción del índice con distintos ordenamientos de los datos no se encontraron diferencias de costos significativas ante el mismo lote de consultas.

- Se desarrolló un segundo método de acceso, el *QuadFQTree*, orientado a objetos espaciales representados a través de puntos. Este índice también mostró ser más eficiente que la solución trivial para ambos conjuntos de datos, requiriendo sólo un 24,43 % para *DBstr* y un 23,15 % para *DBimg* del costo de la solución trivial.

- Se desarrollaron los algoritmos de consulta, inserción y eliminación para ambos métodos.

Como conclusión final se hace notar que los métodos de acceso *QuadFQTree* y *MeTree* poseen un nivel de eficiencia similar, pero cada uno es más apropiado a una situación distinta. El *QuadFQTree* funciona mejor con objetos que se representan a través de vectores de características cuya forma geométrica es un punto, mientras que el *MeTree* obtiene mejores resultados

cuando los objetos que se consultan son cadenas, y no tiene restricciones en el aspecto espacial. Cabe destacar que el primero de ellos tiene un requerimento de almacenamiento significativamente menor.

6.2. Trabajo Futuro

En esta sección delineamos una serie de problemas interesantes cuya solución podría resultar en mejoras de las propiedades de nuestros métodos de acceso.

- La selección de pivotes métrico-espaciales para estos índices no ha sido estudiada. Los pivotes juegan un rol fundamental en este tipo de estructuras, y los estudios existentes sólo sirven para índices métricos puros.

- Otro de los factores importantes en la eficiencia de los métodos de acceso, es la dimensionalidad. En este caso es necesario realizar experimentos con conjuntos de datos de mayor dimensionalidad intrínseca y analizar cómo disminuye la performance de ambos índices.

- Es muy importante la adaptación de estos índices a memoria secundaria, para ello se deben utilizar dos soportes distintos: los discos rígidos y los de estado sólido, ya que poseen características distintas pudiendo las mismas influir en el diseño de los mismos.

- Es conveniente realizar experimentos con el *MeTree* sobre conjuntos de datos representados espacialmente a través de polilíneas y/o polígonos, ya que hasta el momento las pruebas han sido sólo sobre puntos.

- También sería conveniente realizar experimentos con otros tipos de objetos métricos tales como sonido o video, como así también usar funciones de distancia.

REFERENCIAS

[1] ALMEIDA, J., DA SILVA TORRES, R., AND LEITE, N. J. Bp-tree: an efficient index for similarity search in high-dimensional metric spaces. In *CIKM* (2010).

[2] ARONOVICH, L., AND SPIEGLER, I. Cm-tree: A dynamic clustered index for similarity search in metric databases. *Data and Knowledge Engineering 63*, 3 (2007), 919 – 946. 25th International Conference on Conceptual Modeling (ER 2006.

[3] BAEZA-YATES, R. *Searching: An algorithmic tour*, vol. 37. Allen Kent and James G. Williams, editors, 1997.

[4] BAEZA-YATES, R. A., CUNTO, W., MANBER, U., AND WU, S. Proximity matching using fixed-queries trees. In *CPM* (1994).

[5] BEECKS, C. *Distance based similarity models for content based multimedia retrieval*. PhD thesis, Aachen, 2013. Zsfassung in dt. und engl. Sprache; Aachen, Techn. Hochsch., Diss., 2013.

[6] BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18* (1975), 509–517.

[7] BENTLEY, J. L. Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Eng. 5*, 4 (1979), 333–340.

[8] BERCHTOLD, S., KEIM, D., AND KRIEGEL, H. The x-tree: An index structure for high-dimensional data. In *VLDB* (1996), pp. 28–39.

[9] BERGER, B., WATERMAN, M. S., AND YU, Y. W. Levenshtein distance, sequence comparison and biological database search. *IEEE Trans. Inf. Theory 67*, 6 (2021), 3287–3294.

[10] BILHAUT, F., CHARNOIS, T., ENJALBERT, P., AND MATHET, Y. Geographic reference analysis for geographic document querying. In *Proceedings of the HLT-NAACL 2003 Workshop on Analysis of Geographic References* (2003), pp. 55–62.

[11] BOZKAYA, T., AND ÖZSOYOGLU, Z. M. Distance-based indexing for high-dimensional metric spaces. In *SIGMOD Conference* (1997).

[12] BRIN, S. Near neighbor search in large metric spaces. In *VLDB* (1995).

[13] BRIN, S. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1995), VLDB '95, Morgan Kaufmann Publishers Inc., pp. 574–584.

[14] BRITOS, L., PRINTISTA, A. M., AND REYES, N. Dsacl+-tree: A dynamic data structure for similarity search in secondary memory. In *SISAP* (2012).

[15] BURKHARD, W. A., AND KELLER, R. M. Some approaches to best-match file searching. *Commun. ACM 16*, 4 (Apr. 1973), 230–236.

[16] BUSTOS, B., KREFT, S., AND SKOPAL, T. Adapting metric indexes for searching in multi-metric spaces. *Multimedia Tools and Applications 58*, 3 (Jun 2012), 467–496.

[17] BUSTOS, B., NAVARRO, G., AND CHÁVEZ, E. Pivot selection techniques for proximity searching in metric spaces. In *XXI Conference of the Chilean Computer Science Society* (2001), pp. 33–40.

[18] CAI, Q., AND ZHOU, Y. A quadtree-based hierarchical clustering method for visualizing large point dataset. In *2016 Sixth International Conference on Information Science and Technology (ICIST)* (May 2016), pp. 372–375.

[19] CARÉLO, C., POLA, I., CIFERRI, R., TRAINA, A., JR, C., AND CIFERRI, C. Slicing the metric space to provide quick indexing of complex data in the main memory. *Inf. Syst. 36* (03 2011), 79–98.

[20] CASTRO, J., AND BURNS, S. Online data visualization of multidimensional databases using the hilbert space-filling curve. In *VIEW* (2006).

[21] CHÁVEZ, E., HERRERA, N., RUANO, C., AND VILLEGAS, A. Una implementación completa del fqtrie. In *VII Workshop de Investigadores de Ciencias de la Computación* (2005), pp. 61–65.

[22] CHÁVEZ, E., LUDUEÑA, V., REYES, N., AND ROGGERO, P. Faster proximity searching with the distal SAT. *Inf. Syst. 59* (2016), 15–47.

[23] CHÁVEZ, E., MARROQUÍN, J., AND NAVARRO, G. Overcoming the curse of dimensionality. *European Workshop on Content-Based Multimedia Indexing (CBMI'99)* (1999), 57–64.

[24] CHAVEZ, E., AND NAVARRO, G. A compact space decomposition for effective metric indexing. vol. 26, pp. 1363—1376.

[25] CHÁVEZ, E., NAVARRO, G., BAEZA-YATES, R., AND MARROQUÍN, J. L. Searching in metric spaces. *ACM Comput. Surv. 33*, 3 (Sept. 2001), 273–321.

[26] CHÁVEZ, E., NAVARRO, G., BAEZA-YATES, R., AND MARROQUÍN, J. Proximity searching in metric spaces. *ACM Computing Surveys* (2001).

[27] CHAZELLE, B. Computational geometry: a retrospective. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1994), ACM, pp. 75–94.

[28] CHEN, L., GAO, Y., LI, X., JENSEN, C. S., AND CHEN, G. Efficient metric indexing for similarity search and similarity joins. *IEEE Trans. on Knowl. and Data Eng. 29*, 3 (Mar. 2017), 556–571.

[29] CHEN, Y., ZHOU, L., TANG, Y., SINGH, J. P., BOUGUILA, N., WANG, C., WANG, H., AND DU, J. Fast neighbor search by using revised k-d tree. *Information Sciences 472* (2019), 145–162.

[30] CHO, S., KIM, W., OH, S., KIM, C., KOH, K., AND NAM, B. Failure-atomic byte-addressable r-tree for persistent memory. *IEEE Transactions on Parallel Distributed Systems 32*, 03 (mar 2021), 601–614.

[31] CHÁVEZ, E., MARROQUÍN, J., AND NAVARRO, G. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP) 14*, 2 (2001), 113–135.

[32] CIACCIA, P., PATELLA, M., RABITTI, F., AND ZEZULA, P. Indexing metric spaces with m-tree, 1997. Sistemi Evolui per Basi di Dati.

[33] CIACCIA, P., PATELLA, M., AND ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1997), VLDB '97, Morgan Kaufmann Publishers Inc., pp. 426–435.

[34] DOHNAL, V., GENNARO, C., SAVINO, P., AND ZEZULA, P. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications 21* (2003), 9–33.

[35] DUAN, K., LIU, P., JIA, K., AND FENG, Z. An adaptive quad-tree depth range prediction mechanism for hevc. *IEEE Access 6* (2018), 54195–54206.

[36] EFSTATHIADES, C., BELESIOTIS, A., SKOUTAS, D., AND PFOSER, D. Similarity search on spatio-textual point sets. In *EDBT* (2016).

[37] FAN, J., LI, G., ZHOU, L., CHEN, S., AND HU, J. Seal: Spatio-textual similarity search. *PVLDB 5* (2012), 824–835.

[38] FORMAN, S., AND SAMANTHULA, B. K. Secure similar document detection: Optimized computation using the jaccard coefficient. In *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)* (2018), pp. 1–4.

[39] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec. 14*, 2 (June 1984), 47–57.

[40] KALANTARI, I., AND MCDONALD, G. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering 9*, 05 (sep 1983), 631–634.

[41] KALIGIRWA, N., LEAL, E., GRUENWALD, L., ZHANG, J., AND YOU, S. Parallel quad-tree encoding of large-scale raster geospatial data on multicore cpus and gpgpus. In *BigSpatial@SIGSPATIAL* (2014).

[42] LEE, W., ZHENG, B., LI, Z., LEE, K. K., WANG, X., AND LEE, D. L. Ir-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering 23*, 04 (apr 2011), 585–599.

[43] LI, L., YANG, M., WANG, C., AND WANG, B. Robust point set registration using signature quadratic form distance. *IEEE Transactions on Cybernetics 50*, 5 (2020), 2097–2109.

[44] LI, S., WANG, J., AND ZHU, Q. Adaptive bit plane quadtree-based block truncation coding for image compression. In *Ninth International Conference on Graphic and Image Processing (ICGIP 2017)* (2018), H. Yu and J. Dong, Eds., vol. 10615, International Society for Optics and Photonics, SPIE, pp. 436 – 445.

[45] MAI, G., JANOWICZ, K., PRASAD, S., AND YAN, B. Visualizing the semantic similarity of geographic features.

[46] MEHROTRA, S., HARIHARAN, R., HORE, B., AND LI, C. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *Scientific and Statistical Database Management, International Conference on* (Los Alamitos, CA, USA, jul 2007), IEEE Computer Society, p. 16.

[47] MICÓ, L., ONCINA, J., AND CARRASCO, R. C. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters 17* (1996), 731–739.

[48] MICÓ, L., ONCINA, J., AND VIDAL, E. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters 15* (1994), 9–17.

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

[50] NAVARRO, G., AND REYES, N. Dynamic spatial approximation trees for massive data. In *Similarity Search and Applications, International Workshop on* (Los Alamitos, CA, USA, aug 2009), IEEE Computer Society, pp. 81–88.

[51] NAVARRO, G., AND URIBE-PAREDES, R. Fully dynamic metric access methods based on hyperplane partitioning. *Inf. Syst. 36* (06 2011), 734–747.

[52] NIEVERGELT, J., HINTERBERGER, H., AND SEVCIK, K. C. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst. 9* (1984), 38–71.

[53] NOLTEMEIER, H., VERBARG, K., AND ZIRKELBACH, C. Monotonous bisector* trees - a tool for efficient partitioning of complex scenes of geometric objects. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative* (Berlin, Heidelberg, 1992), Springer-Verlag, pp. 186–203.

[54] NOVAK, D., BATKO, M., AND ZEZULA, P. Large-scale similarity data management with distributed metric index. *Information Processing and Management 48* (09 2012), 855–872.

[55] PAREDES, R. U., AND NAVARRO, G. Egnat: A fully dynamic metric access method for secondary memory. In *Proceedings of the 2009 Second International Workshop on Similarity Search and Applications* (Washington, DC, USA, 2009), SISAP '09, IEEE Computer Society, pp. 57–64.

[56] PLANAS, A., PASCAL, A., AND HERRERA, N. Consultas metrico-espaciales. *Congreso Argentino de Ciencias de la Computación - CACIC 25* (2019).

[57] PLANAS, A., PASCAL, A., AND HERRERA, N. Quadfqtree - un indice metrico-espacial. *Congreso Nacional de Ingeniería Informática – Sistemas de Información - CoNaIISI 7* (2019).

[58] PLANAS, A., PASCAL, A., AND HERRERA, N. Metree: A metric spatial index. In *Computer Science – CACIC 2019* (Gewerbestrasse 11,Cham, Switzerland, 2020), Springer, Cham, pp. 250–261.

[59] POLA, I., POLA, F., AND ELER, D. Double distance-calculation-pruning for similarity search. *Information 9* (05 2018), 124.

[60] POLA, I. R. V., TRAINA, C., AND TRAINA, A. J. M. The nobh-tree: Improving in-memory metric access methods by using metric hyperplanes with non-overlapping nodes. *Data and Knowledge Engineering 94* (2014), 65 – 88.

[61] QARI, S. M., ALHARTHI, M. A., HOUMAIDAN, L. S., ALQURASHI, R. M., ALMALKI, A. A., AND ALARABI, L. Grid file index in big data crowdsourcing. In *2022 2nd International Conference on Computing and Information Technology (ICCIT)* (2022), pp. 363–369.

[62] RAM, P., AND SINHA, K. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining* (New York, NY, USA, 2019), KDD '19, Association for Computing Machinery, p. 1378–1388.

[63] RIGAUX, P., SCHOLL, M., AND VOISARD, A. Spatial databases with application to gis. *SIGMOD Record* (01 2001).

[64] RIGAUX, P., SCHOLL, M., AND VOISARD, A. 6 - spatial access methods. In *Spatial Databases*, P. Rigaux, M. Scholl, and A. Voisard, Eds., The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, 2002, pp. 201 – 266.

[65] RUANO, C., CHÁVEZ, E., AND HERRERA, N. Discretización binaria del fqtrie. In *Actas del X Congreso Argentino de Ciencias de la Computación* (Buenos Aires, Argentina, 2004), pp. 100–111.

[66] SAMET, H. The quadtree and related hierarchical data structures. *ACM Comput. Surv. 16*, 2 (1984), 187–260.

[67] SHIN, J., WANG, J., AND AREF, W. G. The lsm rum-tree: A log structured merge r-tree for update-intensive spatial workloads. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)* (Los Alamitos, CA, USA, apr 2021), IEEE Computer Society, pp. 2285–2290.

[68] TAHERI, R., GHAHRAMANI, M., JAVIDAN, R., SHOJAFAR, M., POORANIAN, Z., AND CONTI, M. Similarity-based android malware detection using hamming distance of static binary features. *Future Generation Computer Systems 105* (2020), 230–247.

[69] TANG, J., ZHANG, B., ZHOU, Y., AND WANG, L. An energy-aware spatial index tree for multi-region attribute query aggregation processing in wireless sensor networks. *IEEE Access 5* (2017), 2080–2095.

[70] TRAINA, C., TRAINA, A. J. M., SEEGER, B., AND FALOUTSOS, C. Slim-trees: High performance metric trees minimizing overlap between nodes. In *EDBT* (2000).

[71] TRAINA, JR., C., FILHO, R. F., TRAINA, A. J., VIEIRA, M. R., FALOUTSOS, C., AND FALOUTSOS, C. The omni-family of all-purpose access methods: A simple and effective way to make similarity search more efficient. *The VLDB Journal 16*, 4 (Oct. 2007), 483–505.

[72] UHLMANN, J. Implementing metric trees to satisfy general proximity/similarity queries. Manuscrito, 1991.

[73] VAID, S., JONES, C. B., JOHO, H., AND SANDERSON, M. Spatio-textual indexing for geographical search on the web. In *SSTD* (2005).

[74] VIDAL, E. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters 4* (1986), 145–157.

[75] VIEIRA, M. R., TRAINA, C., CHINO, F. J. T., AND TRAINA, A. J. M. Dbm-tree: A dynamic metric access method sensitive to local density data. In *SBBD* (2004).

[76] XU, H., ZENG, W., ZENG, X., AND YEN, G. G. An evolutionary algorithm based on minkowski distance for many-objective optimization. *IEEE Transactions on Cybernetics 49*, 11 (2019), 3968–3979.

[77] YANG, S., CHEEMA, M. A., LIN, X., ZHANG, Y., AND ZHANG, W. Reverse k nearest neighbors queries and spatial reverse top-k queries. *The VLDB Journal 26*, 2 (Apr 2017), 151–176.

[78] YIANILOS, P. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99* (Baltimore, MD, 1999).

[79] YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA* (1993).

[80] ZHANG, C., ZHANG, Y., ZHANG, W., AND LIN, X. Inverted linear quadtree: Efficient top k spatial keyword search. In *2013 29th IEEE International Conference on Data Engineering (ICDE 2013)* (Los Alamitos, CA, USA, apr 2013), IEEE Computer Society, pp. 901–912.

[81] ZHANG, J., YOU, S., AND GRUENWALD, L. Quadtree-based lightweight data compression for large-scale geospatial rasters on multi-core cpus. In *2015 IEEE International Conference on Big Data (Big Data)* (2015), pp. 478–484.

[82] ZHAO, J., PENG, D., WU, C., CHEN, H., YU, M., ZHENG, W., MA, L., CHAI, H., YE, J., AND QIE, X. Incorporating semantic similarity with geographic correlation for query-poi relevance learning. In *AAAI* (2019).

[83] ZHOU, K., TAN, G., AND ZHOU, W. Quadboost: A scalable concurrent quadtree. *IEEE Transactions on Parallel Distributed Systems 29*, 03 (mar 2018), 673–686.