UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ADAPTACIÓN DE ALGORITMOS PARA INDEXAMIENTO DE ESPACIOS MULTIMÉTRICOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SEBASTIÁN ANDRÉS KREFT CARREÑO

PROFESOR GUÍA:
SR. BENJAMIN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:
SR. PABLO BARCELÓ BAEZA
SR. JENS HARDINGS PERL

SANTIAGO DE CHILE
ENERO 2009

ESTE TRABAJO HA SIDO FINANCIADO POR EL PROYECTO FONDECYT 11070037.

Resumen

Una de las operaciones más importantes en datos multimedia es la de buscar objetos similares entre sí. Para realizar esta búsqueda, es que se recurre al concepto de espacio métrico, el cual permite modelar la relación de similitud por medio de una función de distancia, que cumple la desigualdad triangular, entre otras propiedades. Esta distancia, resulta, generalmente, costosa de calcular, por lo que es necesario la construcción de índices para resolver las búsquedas de manera eficiente.

El tema de la eficacia es también un aspecto muy importante, cuando se trabaja con búsquedas por similitud, ya que no solamente es necesario poder responder las consultas rápidamente, sino que también es necesario entregar resultados relevantes. Para mejorar este aspecto, es que se utiliza un espacio multimétrico, el que define dinámicamente la distancia a utilizar, ponderando en mayor medida aquellas características que sean más relevantes para la consulta. El problema de esta estrategia es que existen pocos índices que permitan trabajar con espacios multimétricos y los índices de espacios métricos no pueden ser usados directamente, pues la distancia de éstos es fija.

Es por esta razón que en esta memoria se busca contribuir con nuevas técnicas de indexamiento para espacios multimétricos. Para esto se estudia y propone una metodología que permite adaptar índices métricos para ser utilizados en un contexto de espacios multimétricos. Se muestra también cómo esta técnica puede ser utilizada para modificar las estructuras *List of Clusters (LC)* [8] y *GNAT* [2], así como también el hecho que las estructuras previamente existentes también resultan de utilizar la metodología propuesta. Finalmente se realiza una evaluación experimental, comparando los índices propuestos con los ya existentes, obteniendo que unos de los índices propuestos, MMGNAT, muestra un mejor desempeño que el estado del arte.

A mis padres y hermanos

Agradecimientos

Quisiera agradecer, en primer lugar, a mi familia, que me apoyó durante todos estos años de estudio, ayudándome a seguir adelante y a afrontar las decisiones tomadas. A los amigos y compañeros con los cuales tuve la suerte de compartir y estudiar.

También quiero agradecer al proyecto Fondecyt 11070037 por el apoyo económico entregado durante la realización de esta investigación, a mi profesor guía, Benjamin Bustos, por toda su orientación y apoyo durante el desarrollo de esta memoria.

Por último, agradezco a los profesores que me instruyeron durante estos seis años, ya que sin ellos probablemente la historia hubiese sido diferente.

Sebastián Kreft Carreño

Santiago, Enero de 2009

Índice General

Índice de figuras

Capítulo 1

Introducción

En la actualidad nos encontramos con diversas aplicaciones que buscan datos dentro de una gran colección. Por ejemplo, cuando realizamos una transacción bancaria, consultamos por la dirección de una persona, etc. Lo que tienen en común todas estas operaciones es el hecho que la información se puede representar y almacenar de una manera estructurada. Las búsquedas pueden realizarse de forma exacta, es decir, se buscan los elementos que calcen completamente con el criterio de búsqueda, o bien es posible buscar todas aquellas entradas en que la llave de búsqueda esté dentro de un rango, como puede ser un rango de tiempo.

En el último tiempo la cantidad de archivos multimedia (vídeos, imágenes, modelos 3D, documentos de texto, etc.) ha tenido un crecimiento significativo. Este crecimiento se debe al avance que ha mostrado la tecnología, desde los equipos de captura hasta la forma de almacenarlos y transmitirlos. Sin embargo, a pesar de este desarrollo tecnológico, todavía es necesario seguir estudiando y analizando el problema de cómo almacenar de manera eficiente estos objetos multimedia, de manera tal que se puedan buscar elementos dentro de una gran base de datos de manera eficiente.

El gran problema al que se enfrentan las bases de datos multimedia es que las búsquedas realizadas no son exactas, sino que son por *similitud*. Esto quiere decir que no se buscan los elementos que sean iguales en su totalidad a una consulta dada, sino aquellos que sean muy parecidos. Esto debido a que la probabilidad que dos objetos multimedia sean completamente iguales bit a bit es prácticamente nula, a no ser que sean copias digitales. Pensemos, por ejemplo, en dos huellas dactilares de una misma persona leídas con un escáner, la presión

con la que se realizó cada una, la sudoración del dedo, entre otros factores, le darán características únicas a cada huella que las harán diferentes entre sí. Sin embargo, seguirán siendo lo suficientemente similares como para identificar a sus dueños.

La búsqueda por similitud tiene muchas aplicaciones en distintas áreas como lo son la de contenidos multimedia, donde se pueden reconocer voces, rostros, huellas dactilares, etc. También es posible comparar documentos, encontrando aquellos que son similares, de manera tal de poder combatir el plagio y resguardar los derechos de autor. Incluso en biología, las búsquedas por similitud son de importancia al estudiar las cadenas de ADN, ya que entre diferentes generaciones de individuos existen mutaciones, por lo que buscar las cadenas parecidas permite identificar y estudiar nuevas mutaciones genéticas.

Para poder trabajar con estos tipos de datos y poder buscar los objetos parecidos es que se recurre al concepto de espacio métrico, que en resumen provee al conjunto de datos de una distancia que permite saber cuán cercanos o parecidos son dos elementos entre sí.

El problema de encontrar los elementos similares a otro en una base de datos puede sonar simple, ya que bastaría con calcular la distancia a todos los elementos y ver cuáles son los más cercanos. Sin embargo, si consideramos que el costo de calcular la distancia entre dos objetos es T_δ, tendríamos que el costo en tiempo sería $O(nT_\delta)$, el cual es muy alto para muchas aplicaciones interesantes, debido a que en general calcular la distancia se considera computacionalmente costoso. Sin embargo, este enfoque es la única forma de solucionar el problema si es que no se permite analizar los datos previamente.

Puesto que el costo de comparar dos objetos entre sí es elevado, es necesario disminuir el número de cálculos de distancia para realizar las búsquedas por similitud eficientemente. Para lograr esto es que se procesan los datos antes de realizar consultas sobre ellos, creando una estructura llamada *índice*, la cual permite realizar búsquedas por similitud de una manera más eficiente.

Existen muchas estrategias para resolver el problema del indexamiento en espacios métri-

cos, entre ellas podemos encontrar a AESA, GNAT, VPT, LC, entre otros [9]. Dentro de estos índices encontramos algunos que consideran la distancias entre los objetos como una caja negra, como por ejemplo, en LC [8], GNAT [2], etc. También existen aplicaciones en las que se trabaja con un grupo particular de espacios métricos, los llamados espacios vectoriales, donde los índices pueden explotar las propiedades geométricas del espacio, como por ejemplo los *R-trees* [16]. Existen dos tipos de índices, unos estáticos y otros dinámicos, estos últimos permiten realizar operaciones de actualización de la base de datos, siendo algunos más adecuados para cierto tipo de aplicaciones.

Otro obstáculo al tratar con espacios métricos es que el desempeño de las distintas técnicas dependerá de los datos con que se esté trabajando. Por ejemplo, en el caso de espacios vectoriales, en que cada objeto es modelado como un vector característico, se tiene que a medida que el número de elementos del vector (la dimensión) aumenta todos los elementos se encuentran aproximadamente a la misma distancia entre sí. Lo anterior se conoce como *maldición de la dimensionalidad*, ya que hace muy difícil distinguir entre elementos relevantes al momento de realizar búsquedas, por lo que el desempeño de los algoritmos de búsqueda disminuye considerablemente. Esta noción también se puede extender a espacios métricos genéricos por medio de la llamada *dimensionalidad intrínseca* [9].

El tema de la eficacia de la búsqueda es también un aspecto muy importante, ya que se necesita que los resultados de las consultas sean relevantes al contexto en que se trabaja. Además, una dificultad adicional respecto de la eficacia es el hecho que no puede ser mejorada con la incorporación de mejor *hardware*, como en el caso de la eficiencia, sino que tiene que ser por medio de *software*, es decir a través de la inclusión de nuevos algoritmos y estrategias de búsqueda. Es por esta razón que desde hace un tiempo se comenzó a estudiar cómo mejorar la calidad de los resultados de las consultas por similitud. Una forma de lograr esto es utilizando una combinación de vectores característicos o de métricas. Este enfoque mostró tener una mayor efectividad en búsquedas por similitud [4]. A partir de esto se introdujo el concepto de espacio multimétrico (definido en la Sección 2.5), en el cual dado el objeto de consulta se define dinámicamente una métrica a partir de una combinación ponderada de otras métricas, ponderando mayormente aquellas características que beneficien más la búsqueda.

El problema de este enfoque, es que el indexamiento de los espacios multimétricos todavía es un área en desarrollo, existiendo pocas alternativas. Entre estas podemos encontrar los *M^3-trees* [7] y la estrategia *Pivot Based* [6]. Además, los índices para espacios métricos no pueden ser utilizados directamente, ya que estos usan una métrica fija. En cambio, en los espacios multimétricos con cada nueva consulta se cambia la métrica a utilizar.

Es por estas razones que es importante el estudio de nuevas alternativas de indexamiento sobre espacios multimétricos, para así tener una mayor variedad de métodos al momento de elegir el adecuado para una aplicación.

En esta memoria se pretende aportar nuevas técnicas de indexamiento para este tipo de espacios. Para ello, se estudia y propone una metodología que permite adaptar índices métricos para ser utilizados en un contexto de espacios multimétricos. Se muestra también cómo esta técnica puede ser utilizada para modificar las estructuras *List of Clusters (LC)* [8] y *GNAT* [2]. Se eligieron estos índices ya que *LC* es uno de los que funcionan mejor en la práctica para espacios con dimensionalidad intrínseca alta y *GNAT* es un algoritmo basado en particiones de Voronoi, que captura la geometría intrínseca del espacio.

En el Capítulo 2 de esta memoria se introducen los conceptos básicos necesarios para comprender el trabajo a realizar. En el Capítulo 3 se describen tres índices para espacios métricos, estos son *List of Clusters*, *GNAT* y *M-Tree*. En el Capítulo 4 se describen los índices existentes para espacios multimétricos, así como también los trabajos relacionados, de manera tal de mostrar el estado del arte respecto de los espacios multimétricos. En el Capítulo 5 se presenta la metodología para adaptar índices métricos. En este capítulo se muestra también cómo utilizar la metodología para modificar los índices *List of Clusters* y *GNAT*, y como los índices *M-Tree* y *Pivot Based* pueden ser vistos como una aplicación de la metodología presentada. Luego en el Capítulo 6 se presentan los resultados obtenidos luego de experimentar con 3 bases de datos reales. Finalmente en el Capítulo 7 se muestran las conclusiones obtenidas en esta memoria.

Capítulo 2

Conceptos Básicos

En esta sección se describen algunos conceptos que son importantes para entender el presente trabajo. Entre ellos se encuentran la definición de espacio métrico, algunas funciones de distancia utilizadas en espacios vectoriales, los tipos de búsqueda por similitud y finalmente el concepto de espacio multimétrico.

2.1. Espacios métricos

Un espacio métrico corresponde al par ordenado $(\mathbb{X}, \delta)$, donde $\mathbb{X} \neq \emptyset$ es el conjunto universo y δ es una métrica, es decir, es una función $\delta : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+$ que satisface las siguientes propiedades:

1. **Simetría** $\forall x, y \in \mathbb{X},\ \delta(x, y) = \delta(y, x)$
2. **Reflexividad** $\forall x \in \mathbb{X},\ \delta(x, x) = 0$
3. **Positividad Estricta** $\forall x, y \in \mathbb{X},\ x \neq y \Longrightarrow \delta(x, y) > 0$
4. **Desigualdad Triangular** $\forall x, y, z \in \mathbb{X},\ \delta(x, z) \leq \delta(x, y) + \delta(y, z)$

2.1.1. Espacio vectorial

Un espacio vectorial es un espacio métrico que cumple además con propiedades adicionales que le otorgan una geometría al espacio. Dentro de estos espacios encontramos los espacios vectoriales reales o $\mathbb{R}^d$, donde cada elemento es una d-tupla de coordenadas reales.

2.1.2. Distancias de Minkowski

Un conjunto especial de métricas para espacios vectoriales corresponde a las llamadas distancias de Minkowski. Estas métricas están definidas por la siguiente fórmula

$$L_p(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}} \quad p \geq 1 \tag{2.1}$$

donde d corresponde a la dimensión del espacio y x_i corresponde a la coordenada i-ésima del elemento x.

Cuando $p = 1$ nos encontramos con la distancia Manhattan, cuando $p = 2$ tenemos la tradicional distancia Euclidiana, y por último cuando se toma el límite de $p \to \infty$ nos encontramos con la distancia de Chebyshev, o distancia del máximo.

$$L_\infty(x, y) = \lim_{p \to \infty} \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}} = \max \left(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_d - y_d| \right) \tag{2.2}$$

La Figura 2.1 a continuación muestra los puntos que están a la misma distancia del centro bajo distintas distancias de Minkowski.

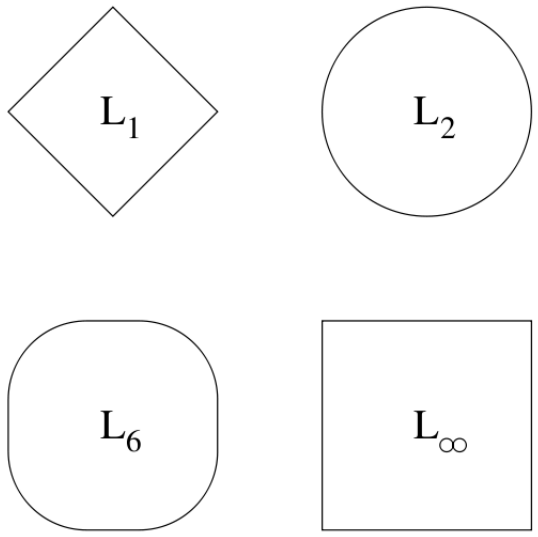

Figura 2.1: Puntos a la misma distancia del centro con distancias de Minkowski

2.2. Búsqueda por similitud

Dado un elemento $q \in \mathbb{X}$, es decir, en el espacio de posibles objetos, y una base de datos $\mathbb{U} \subseteq \mathbb{X}$, se quieren encontrar todos los elementos de la base de datos que sean similares a

q. Para esto existen básicamente dos tipos de consultas: la búsqueda por rango y la de los k-vecinos más cercanos.

2.2.1. Búsqueda por rango

Dados $q \in \mathbb{X}$ y $r \in \mathbb{R}$, una búsqueda por rango en el espacio métrico $(\mathbb{U}, \delta)$, corresponde al conjunto:

$$(q, r)_\delta = \{x \in \mathbb{U},\, \delta(x, q) \leq r\} \tag{2.3}$$

Es decir, entrega todos los elementos de la base de datos que están a una distancia r o menor del elemento q.

En la Figura 2.2 se observa una consulta por rango en $\mathbb{R}^2$, en la cual los elementos q_6, q_{10} y q_{14} pertenecen al rango.

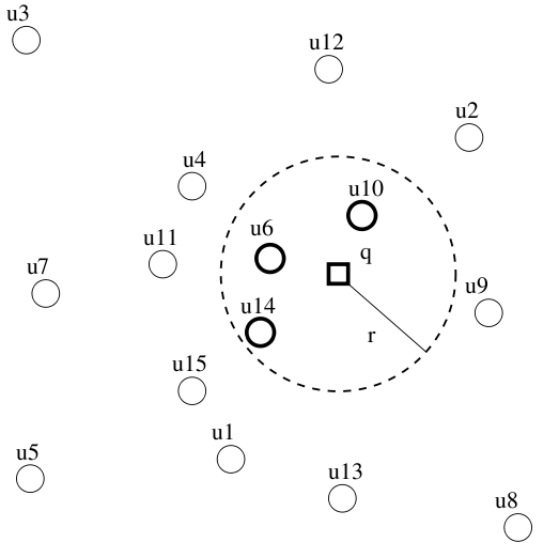

Figura 2.2: Consulta por rango en $\mathbb{R}^2$

2.2.2. Vecinos más cercanos

La consulta de los k-vecinos más cercanos entrega los k elementos en $\mathbb{U}$ más cercanos al elemento q. Esto es $NN_k(q)$ entrega un conjunto $A \subseteq \mathbb{U}$ tal que

$$|A| = k \,\wedge\, \forall u \in A, v \in \mathbb{U} - A,\ \delta(q, u) \leq \delta(q, v) \tag{2.4}$$

Notar que el conjunto A no necesariamente es único.

2.3. Tipos de índices

Según Chávez et al. [9] los índices para espacios métricos se pueden clasificar de dos maneras, estas son basado en pivotes o basado en particiones compactas.

- Un algoritmo está basado en particiones compactas si divide a la base de datos en zonas espaciales lo más compactas posibles, y es capaz de descartar zonas completas realizando pocos cálculos de distancias.
- Un algoritmo está basado en pivotes si selecciona un número de "pivotes" (objetos distinguidos de la BD) y clasifica al resto de los elementos de acuerdo a la distancia a cada uno de los pivotes.

Generalmente los algoritmos basados en pivotes mejoran el rendimiento si se añaden más pivotes, con lo que el espacio necesario para almacenar el índice también aumenta. En cambio, los índices basados en particiones compactas usan una cantidad fija de memoria y generalmente tienen mejor desempeño que los algoritmos basados en pivotes en espacios de dimensionalidad alta.

2.4. Criterios de exclusión

A continuación se presentan los criterios utilizados para descartar elementos al realizas consultas por rango [3].

2.4.1. Pivotes

Cuando se trabaja con pivotes, se pueden descartar todos los objetos que no pertenezcan al anillo centrado en p definido por los radios $\delta(p,q) - r$ y $\delta(p,q) + r$. Esto se puede ver en la Figura 2.3, donde a la izquierda se tiene un solo pivote y pueden ser descartados todos los elementos que quedan fuera del anillo, y en la derecha se observa que se pueden descartar todos los elementos que no pertenezcan a la intrersección de los dos anillos. Es importante notar que a medida que se agregan pivotes, la intersección de los anillos acotará de mejor manera la bola de consulta.

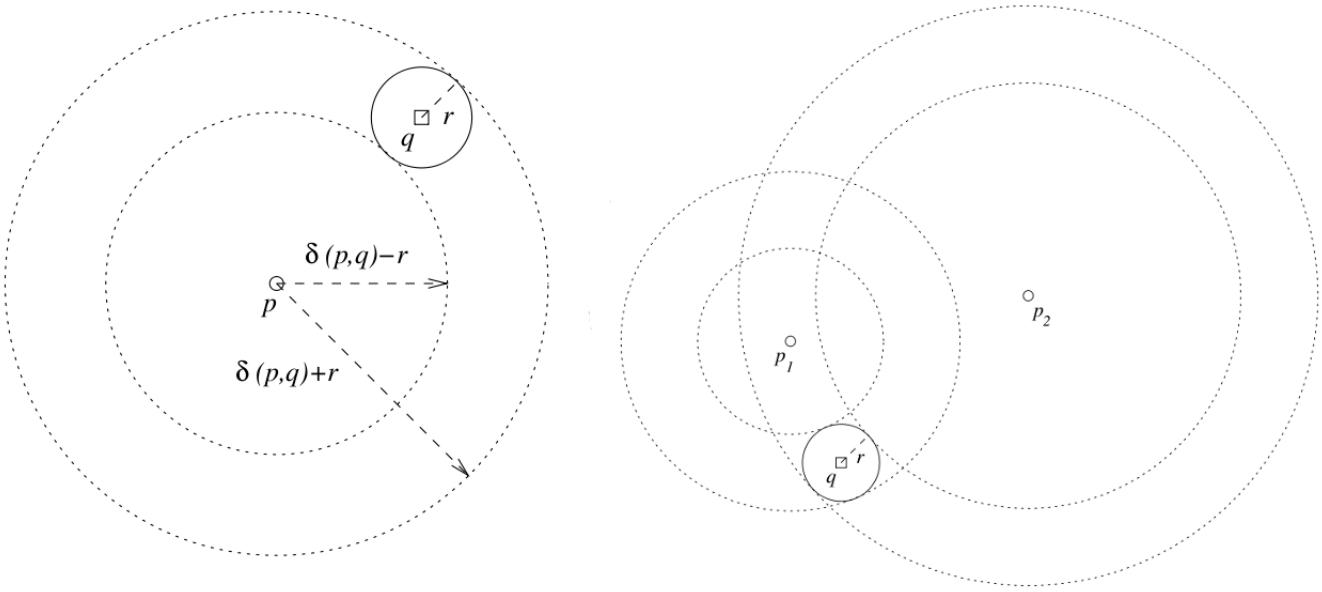

Figura 2.3: Criterio de exclusión con pivotes

2.4.2. Radio Cobertor

El radio cobertor r_c corresponde a la máxima distancia entre un centro c y los miembros de la zona que define. En la Figura 2.4 se aprecia que en caso que el objeto de consulta sea q_1 o q_2 es necesario seguir buscando elementos en la zona, pues puede contener objetos relevantes. En cambio si el objeto de consulta fuera q_3, entonces la zona puede ser descartada, ya que se cumple que $\delta(c, q) - r > r_c$.

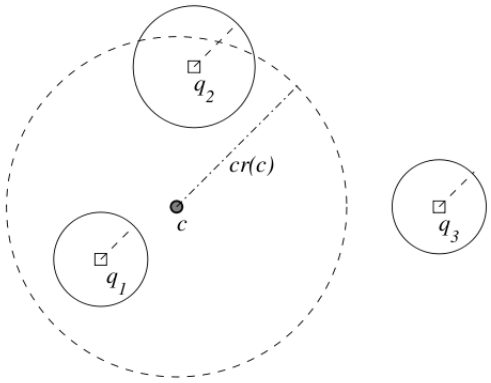

Figura 2.4: Criterio de exclusión del radio cobertor

2.4.3. Partición de Voronoi

En una partición de Voronoi se escogen centros y el resto de los elementos se asocia al centro más cercano, formando así zonas. Luego en una consulta por rango se calcula la distancia entre q y todos los centros. Sea c el centro más cercano a q, entonces se pueden podar todas las zonas definidas por $c_i \neq c$, tales que $\delta(q, c_i) > \delta(q, c) + 2r$, puesto que esas zonas no pueden intersectar la bola de consulta. . En la Figura 2.5 se muestra como funciona este criterio. Si se considera el objeto de consulta q_1 entonces se puede descartar la zona definida por c_4, y en el caso de q_2 solo es necesario revisar la zona definida por c_4.

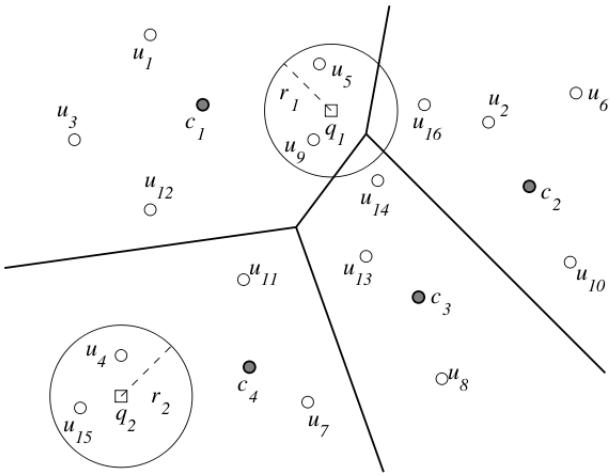

Figura 2.5: Criterio de exclusión con particiones de voronoi

2.5. Espacios multimétricos

Sean $\mathcal{M} = \{(\mathbb{X}_i, \delta_i),\ 1 \leq i \leq n\}$ un conjunto de espacios métricos, se define el *espacio multimétrico* asociado como el par ordenado $(\prod_{i=1}^{n} \mathbb{X}_i, \Delta_{\mathbb{W}})$, donde $\Delta_{\mathbb{W}}$ es una multimétrica lineal, es decir

$$\Delta_{\mathbb{W}}(x, y) = \sum_{i=1}^{n} w_i \delta_i(x, y) \tag{2.5}$$

En la definición anterior el vector de pesos $\mathbb{W} = \langle w_i \rangle$ no está fijo, sino que es un parámetro de Δ. Cabe destacar que si $\forall i\, w_i \in [0, 1] \wedge \exists i\, w_i > 0$, se tiene que $\Delta_{\mathbb{W}}$ es también una métrica.

Capítulo 3

Índices para Espacios Métricos

A continuación se describen los dos algoritmos de indexamiento de espacios métricos que serán adaptados durante la memoria para ser utilizados en un contexto de espacios multimétricos. También se describe el M-Tree, que es un índice dinámico que es la base del índice multimétrico M^3-tree.

3.1. GNAT

Geometric Nearest-Neighbor Access Tree o *GNAT* [2] es una estructura basada en particiones compactas así como también en pivotes. *GNAT* trata que la estructura represente la "geometría intrínseca" del espacio. Para esto utiliza una estructura jerárquica basada en dominios de Dirichlet, llamados también celdas de Voronoi.

3.1.1. Construcción

Dados los puntos $x_1, x_2, \ldots, x_k$ el dominio de Dirichlet asociado a x_i corresponde a todos los puntos que están más cerca de x_i que de cualquier otro x_j. En la raíz de GNAT se elige un conjunto de puntos llamados *split points*, y el espacio es particionado en los dominios de Dirichlet asociados a cada *split point*. Luego, se calcula la distancia máxima y mínima de cada *split point* a cada zona (ver Figura 3.2). Finalmente, cada dominio es estructurado recursivamente.

El algoritmo de construcción de la estructura se detalla en el Algoritmo 1. Los *split points* son seleccionados de manera tal que sean aleatorios, pero que también estén lo suficientemente

alejados entre ellos. Esto se debe a que si dos *split points* están muy cerca no van a aportar demasiada información, además puede resultar que un cluster sea particionado a un gran nivel de detalle.

```
1 function Build (𝕌)
2     let P=p_1, ..., p_k, conjunto de k split points
3     foreach x ∈ 𝕌 − P do asociar x con split point más cercano
4     let D_{p_i} = conjunto asociado a p_i
5     foreach (p_i, p_j) ∈ P × P do Calcular
      range(p_i, D_{p_j}) = [min_d(p_i, D_{p_j}), max_d(p_i, D_{p_j})]
6     foreach p_i ∈ P do Build (D_{p_i})
7 end
```

Algoritmo 1: Construcción de GNAT

En la Figura 3.1 se puede ver un ejemplo de la estructura de GNAT con $k = 4$. A la izquierda se observa la descomposición del espacio en los dominios de Dirichlet y a la derecha el árbol que representa la estructura.

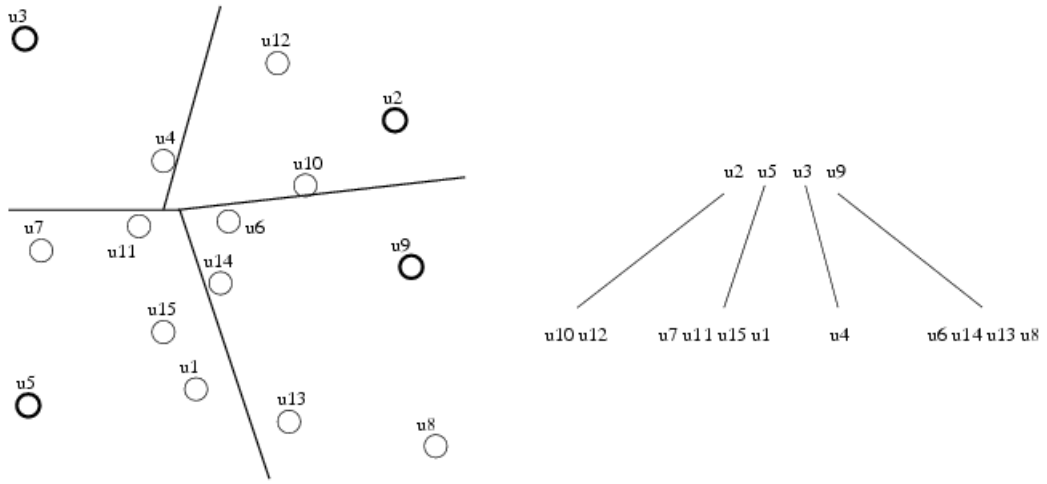

Figura 3.1: Estructura de GNAT.

3.1.2. Búsqueda por Rango

Para realizar una búsqueda por rango se calcula la distancia de cada uno de los *split points* al punto de consulta y se verifica si pertenece al rango o no, verificando que los puntos pertenecientes a la zona definida por este *split point* pertenezcan a la bola de consulta. En

caso contrario, se poda esa rama del árbol. La poda se puede ver en la línea 7, del Algoritmo 2, donde la condición se justifica a raíz de la desigualdad triangular. Gráficamente esta condición puede ser apreciada en la Figura 3.2.

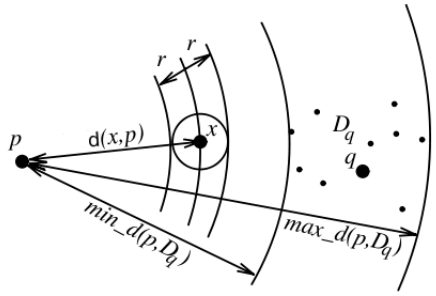

Figura 3.2: Poda de ramas en GNAT usando rangos.

```
1 function Search (N,x,r)
2     let P=split points(N)
3     foreach p ∈ P do
4         Calcular d(p, x)
5         if d(p, x) ≤ r then reportar p
6         foreach q ∈ P do
7             if [d(x, p) − r, d(x, p) + r] ∩ range(p, D_q) = ∅ then remover q de P
8     foreach p ∈ P do Search (D_p,x,r)
9 end
```

Algoritmo 2: Búsqueda por Rango en GNAT

3.1.3. k-Vecinos más cercanos

Para realizar una búsqueda de k-vecinos más cercanos se utiliza un algoritmo desarrollado en base a la técnica presentada por Hjaltason y Samet en [18], ya que en la publicación donde fue presentada esta estructura, no se explicaba como realizar este tipo de consultas.

Para construir el algoritmo es necesario poder estimar la distancia desde el punto de consulta a un conjunto, que corresponderá a alguna de las zonas de Voronoi. Para esto se utilizan los rangos calculados entre cada par de *splits points*.

Para estimar la distancia entre el *split point* q y el punto x, se utilizan los rangos de cada *split point* a D_q. Si consideramos el *split point* p, el punto x tiene 3 lugares en donde puede

estar. Estos son, dentro de la zona D_q, en cuyo caso la distancia es 0, más alejado de p que de D_q, o más cerca de p que de D_q. Los últimos dos casos se muestran en la Figura 3.3. En el segundo caso la distancia puede ser acotada superiormente como $d(p,x) - max_d(p, D_q)$, y en el último caso la distancia puede ser acotada superiormente por $min_d(p, D_q) - d(p,x)$. Luego para calcular la estimación de la distancia se toma el máximo de todas las estimaciones realizadas con cada *split point*.

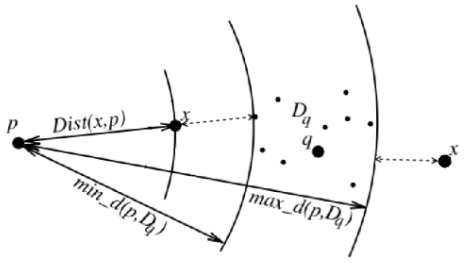

Figura 3.3: Distancia de un punto a uno zona en GNAT.

El algoritmo obtenido para k-NN se puede ver en el Algoritmo 3.

```
1  function Knn (N,x,k)
2      let Q=priorityQueue()
3      let cnt = 0
4      enqueue((root,0),Q)
5      while Q ≠ ∅ do
6          let element = dequeue(Q)
7          if element is Spatial object then
8              report element
9              cnt++
10             if cnt = k then return
11         else
12             foreach p ∈ P do
13                 Calcular d(p,x)
14                 enqueue((p,d(p,x)),Q)
15             foreach p ∈ P do
16                 let dist = máx_{q∈P}{d(q,x) − min_d(q,D_p), max_d(q,D_p) − d(q,x), 0}
17                 enqueue((p,dist),Q)
18 end
```

Algoritmo 3: k-Vecinos más cercanos en GNAT

3.2. List of Clusters

List of Clusters [8] es un índice para espacios métricos basado en particiones compactas, que muestra un buen desempeño en espacios de dimensión alta. Además, debido a su estructura funciona bien en memoria secundaria.

3.2.1. Construcción

Para construir la estructura se elige un elemento c de la base de datos y un radio r_c, luego todos los elementos que estén a una distancia r_c o menor de c se agrupan en I_c, es decir, este conjunto corresponde a

$$I_c = \{u \in \mathbb{U} - \{c\}, d(c,u) \leq r_c\} \tag{3.1}$$

El resto de los elementos se agrupa en E_c, es decir

$$E_c = \{u \in \mathbb{U}, d(c,u) > r_c\} = \mathbb{U} - I_c \tag{3.2}$$

Luego el proceso se realiza recursivamente en E_c. El algoritmo se puede ver en detalle en el Algoritmo 4. En la Figura 3.4 se muestra un ejemplo de como se organiza la estructura

```
1 function Build(𝕌)
2     if 𝕌 = ∅ then return lista vacía
3     Seleccionar c ∈ 𝕌
4     Seleccionar radio r_c
5     I_c ← {u ∈ 𝕌 − {c}, d(c,u) ≤ r_c}
6     E_c ← 𝕌 − I_c
7     return (c, r_c, I_c) :Build (E_c)
8 end
```

Algoritmo 4: Construcción de List of Clusters

en $\mathbb{R}^2$ y como queda la estructura en forma de lista. En la Figura 3.4 se muestra también una característica muy importante de esta estructura, que es el hecho de que un elemento pertenece necesariamente a la primera partición que lo contenga. Es decir, en caso que dos particiones se superpongan los elementos que puedan caer en la intersección de ambas, necesariamente pertenecerán a la que fue creada primera. En el caso de la Figura 3.4, el elemento u pertenece a la partición c_1, aunque también cae dentro de la partición c_2.

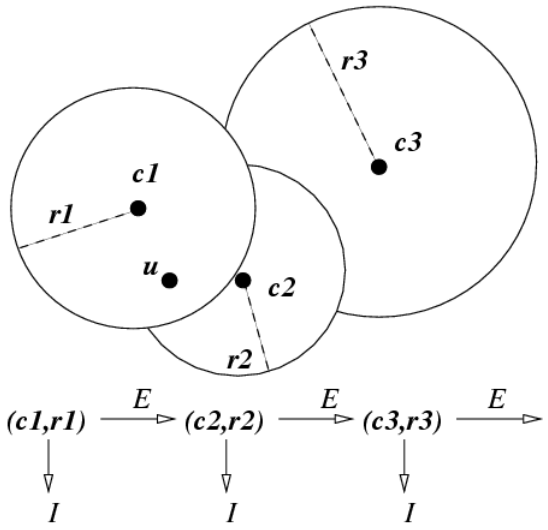

Figura 3.4: Ejemplo de List of Cluster en $\mathbb{R}^2$.

En el algoritmo de construcción antes presentado hay dos partes que se pueden realizar de distintas maneras. Estas son la elección del centro de la partición y la elección del radio. Luego de realizar algunos experimentos se obtuvo que la mejor estrategia para la elección del centro es: *(p5) El elemento que maximiza la suma de las distancias a los centros previos* [8]. Esta estrategia trata que las particiones no se superpongan, eligiendo centros que estén alejados entre sí, tal como se hace en GNAT.

Para el caso del radio, existen dos alternativas, particiones de radio fijo o particiones de tamaño fijo. La primera alternativa implica que a medida que avanzamos en la lista las particiones van quedando vacías. En la segunda alternativa a medida que se avanza en la lista las particiones son de mayor tamaño, además tiene la ventaja que la lista queda de tamaño fijo. Luego de resultados experimentales se optó por trabajar con particiones de tamaño fijo, ya que en el caso de particiones de radio fijo una pequeña variación en la elección de radio cambiaba el rendimiento del índice notablemente.

El tiempo de construcción del algoritmo es $O(n^2/m^*)$, donde m^* es el número de elementos promedios de cada partición. A pesar que este costo en principio es independiente de la dimensión, en la práctica aumenta debido a que en dimensiones altas es necesario reducir m^* de manera de conseguir mejores resultados.

3.2.2. Búsqueda por Rango

```
1 function Search (L,q,r)
2     if L = ∅ then return
3     let L = (c, r_c, I) : E
4     Calcular d(c, q)
5     if d(c, q) ≤ r then agregar c a la lista de resultados
6     if d(c, q) ≤ r_c + r then buscar exhaustivamente en I
7     if d(c, q) > r_c − r then Search (E,q,r)
8 end
```

Algoritmo 5: Búsqueda por Rango de List of Clusters

El algoritmo de búsqueda está esquematizado en el Algoritmo 5. La idea es calcular la distancia del objeto de consulta q al primer centro de la lista c y se añade c a los resultados si corresponde. Luego existen 3 casos que se presentan en la Figura 3.5, el primero (q_1) es si existe intersección entre el *bucket* I_c y la bola de consulta, caso en que se revisa exhaustivamente I_c. El segundo (q_2) ocurre cuando la bola de consulta está totalmente contenida en I_c, caso en que la búsqueda puede ser terminada, ya que por construcción ningún elemento de la bola de consulta puede estar en otro *bucket*. Este último caso es una característica esencial de la estructura que no se encuentra en otros algoritmos. Finalmente, el tercer caso (q_3), ocurre cuando la bola de consulta no intersecta a I_c, caso en que no se revisa I_c. Tanto en el primer, como en el último caso la búsqueda continúa recursivamente en el resto E_c de la lista.

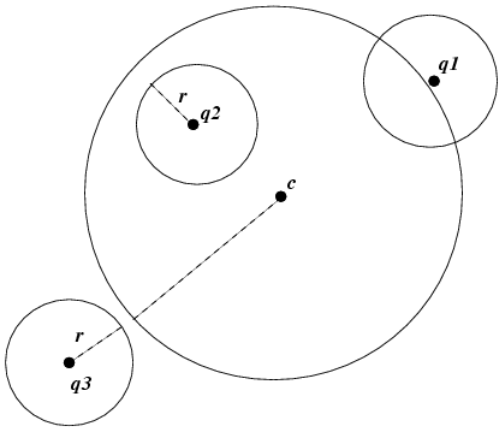

Figura 3.5: Casos posibles en búsqueda por rango en List of Clusters

3.2.3. k-Vecinos más cercanos

El algoritmo para realizar una búsqueda de k-vecinos más cercanos aquí presentado fue obtenido de la librería SISAP [15].

La idea de este algoritmo es ir revisando los *buckets* que puedan tener mayor posibilidad de contener a los vecinos más cercanos, manteniendo una lista con los k elementos más cercanos revisados hasta el momento. Esto permite omitir la búsqueda de algunos *buckets*, ya que a medida que se va construyendo la solución se van teniendo estimaciones del radio máximo que puede tener la solución. Para esto se mantiene una cola de prioridad que almacenará como máximo k elementos, permitiendo así conocer el elemento más lejano encontrado hasta el momento.

En el Algoritmo 6 se presenta el método en detalle.

```
1  function Knn (cluster,k,q,res)
2      if cluster > size(list) then return
3      Calcular d(cluster.c, q)
4      if dist ≤ cluster.r then
5          searchBucketKnn(cluster.bucket,k,q,res)
6          if size(res) < k or d(cluster.c, q) + res.maxdist ≥ cluster.r then Knn
           (cluster.next,k,q,res)
7      else
8          Knn (cluster.next,k,q,res)
9          if size(res) < k or d(cluster.c, q) − res.maxdist ≤ cluster.r then
           searchBucketKnn(cluster.bucket,k,q,res)
10 end
```

Algoritmo 6: Búsqueda de k-Vecinos más cercanos en List of Clusters

3.3. M-Tree

M-Tree [12] es un índice métrico dinámico y balanceado que presenta un buen comportamiento en memoria secundaria.

3.3.1. Construcción

En las hojas del árbol se almacenan los objetos indexados de la base de datos y en los nodos internos se almacenan los *routing objects*.

Un *routing object* O_r almacena la siguiente información:

- El vector característico del objeto multimedia, no una referencia. Esto es importante para asegurar el buen comportamiento en memoria secundaria.
- Un puntero $ptr(T(O_r))$ que referencia a la raíz del árbol cobertor de O_r, llamado $T(O_r)$.
- Un radio $r(O_r)$. Todos los elementos del árbol $T(O_r)$ están a una distancia menor que $r(O_r)$ de $T(O_r)$.
- La distancia del objeto al padre $\delta(O_r, P(O_r))$.

Las hojas O_j almacenan la siguiente información:

- El vector característico del objeto multimedia, no una referencia. Esto es importante para asegurar el buen comportamiento en memoria secundaria.
- Un puntero $oid(O_j))$ que referencia al objeto multimedia.
- La distancia del objeto al padre $\delta(O_r, P(O_r))$.

Para construir el árbol se van insertando los elementos uno a uno. Esto es posible gracias a que la estructura es dinámica. Para insertar un elemento se revisa desde el tope del árbol hacia abajo cuál es el mejor nodo para insertar el elemento, este nodo será aquel cuyo radio cobertor tenga que aumentar lo menos posible de manera tal de poder alojar el nuevo elemento.

Luego de ingresado el elemento en su correspondiente nodo, puede que la hoja este llena, por lo que puede ser necesario realizar una división de la hoja en dos nodos. Al momento de realizar esta división se eligen dos elementos que se promueven de nivel y el resto de

los elementos se asocian a cada uno de los elementos escogidos. En esta etapa se trata de minimizar le volumen de las regiones así como el traslape de ambas.

3.3.2. Algoritmos de búsqueda

El algoritmo de búsqueda de k-vecinos más cercanos está basado en la técnica propuesta por Hjaltason y Samet en [18]. Utilizando como distancia del objeto de consulta q a un nodo O_r el valor máx$(\delta(q, O_r) - r(O_r), 0{,}0)$.

Para realizar una búsqueda por rango se recorre el árbol desde la raíz hasta las hojas eliminando en el camino aquellos nodos en que se cumpla $\delta(O_r, q) > r + r(O_r)$. La condición anterior se debe a que todos los elementos en el subárbol definido por O_r, están a una distancia menor que $r(O_r)$ de O_r, luego por la desigualdad triangular se cumple que todos los elementos en el subárbol de O_r están a una distancia mayor a r, siendo posible la poda de ese subárbol.

En ambas consultas se utiliza la distancia al padre para disminuir el número de distancias calculadas. Cada vez que se cumpla $|\delta(P(O_r), q) - \delta(O_r, P(O_r))| > r + r(O_r)$, entonces se cumple que $\delta(O_r, q) > r + r(O_r)$, que es justamente la condición para podar una rama. Esta propiedad es consecuencia directa de la desigualdad triangular y puede ser apreciada en la Figura 3.6.

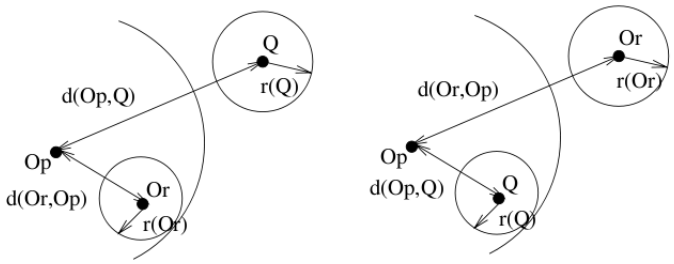

Figura 3.6: Distancias en M-Tree

Capítulo 4

Índices para Espacios Multimétricos

A continuación se presentan las estructuras y algoritmos existentes para tratar con búsquedas por similitud en espacios multimétricos, así como también estructuras que se relacionan con la indexación de espacios multimétricos.

4.1. Pivot Based

En "A Pivot-Based Index Structure for Combination of Feature Vectors" [6] B. Bustos et al. presentan un índice basado en pivotes para espacios multimétricos. Este índice consiste de un conjunto de índices basados en pivotes, uno por cada métrica, los que son usados posteriormente para calcular la tabla de pivotes al momento de la consulta, que es cuando los pesos de las distintas métricas son conocidos. Los autores presentan también un algoritmo para responder consultas del vecino más cercano, tanto con pesos fijos como con pesos dinámicos.

Resultados experimentales con datos reales mostraron que la estructura propuesta mejora hasta en un factor de 3x el rendimiento en comparación con un análisis lineal de la base de datos. Los resultados muestran también que utilizar un R*-tree [1], posible solamente dadas ciertas restricciones, en la práctica no funciona para nada bien debido a la alta dimensionalidad que presenta el espacio resultante. La mayor desventaja de este índice es que funciona en memoria principal y no es claro como implementarlo en memoria secundaria.

4.2. M^3-tree

Multi-metric M-tree (M^3-tree) [7] es un índice dinámico para espacios multimétricos basado en M-tree [12].

El índice se construye con una métrica estática ($\Delta_{1,0}$) en que todos los pesos son iguales a 1, por lo que el proceso de construcción es muy similar al del M-tree. Lo único que cambia es que cada vez que se almacenan las distancias al padre o el radio se almacenan además las componentes de estos valores. Esta información adicional permite generar una cota bastante ajustada del radio de la región cuando la métrica es cambiada por una con pesos dinámicos. La estructura modificada del M^3-tree se aprecia en la Figura 4.1, donde los campos en gris corresponden a los campos originales de M-tree, y los campos en negro son los campos adicionales de M^3-tree.

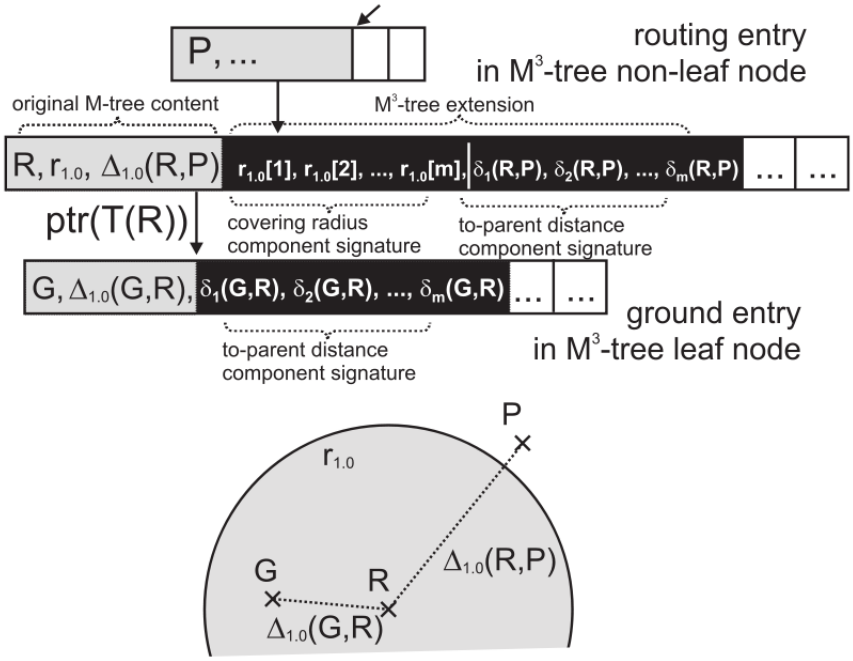

Figura 4.1: Estructura de M^3-tree

En el artículo se presenta también una forma de adaptar el M-tree para que pueda ser utilizado en consultas por similitud en espacios multimétricos sin la necesidad de agregarle información adicional a la estructura.

En la evaluación experimental se obtuvo que el M^3-tree se comporta casi tan bien como si se tuviera un M-tree por cada combinación de métricas utilizadas, lo que es el resultado óptimo que se puede esperar de la estructura. También se muestra que el M-tree adaptado funciona relativamente bien sólo con coeficientes altos, aunque de todas formas no supera al M^3-tree. Respecto a los accesos a disco, M^3-tree realiza casi los mismos accesos que se harían si se tuviera un índice por cada métrica, y el M-tree adaptado realiza más accesos a discos incluso que el análisis secuencial de los datos.

4.3. M^2-tree

M^2-tree [10] es un índice que permite indexar múltiples espacios métricos y realizar sobre ellos consultas complejas, como pueden ser una combinación de características, lo que correspondería a una búsqueda por similitud en un espacio multimétrico.

La estructura básicamente es una unión de varios M-trees [12], uno por cada espacio que está siendo indexado. Para responder las consultas, el algoritmo calcula una cota inferior de la distancia. Para esto necesita que la función que representa a la consulta sea una función monótona en todas sus características. Luego, logra la cota acotando cada característica por el valor que puede obtener del M-tree asociado. El problema de este enfoque es el hecho que se considera la combinación de características como una caja negra, lo que impide obtener una cota ajustada.

En el artículo los autores no describen mucho su estructura ni tampoco los algoritmos utilizados. En los resultados se presenta solo una mejora del 30 % en el cálculo de distancias en comparación con un análisis secuencial de los datos.

4.4. QIC-M-tree

QIC-M-Tree [11] es un índice que permite realizar consultas con distancias definidas por el usuario. Para esto se construye el índice con una métrica fija que tiene que ser una cota inferior de las distancias que serán utilizadas posteriormente en las consultas. Esta cota inferior representa el hecho que existe un s tal que para cada par de datos se cumple que

$d_i(a,b) \leq s \cdot d_q(a,b)$. El problema con este enfoque es que al momento de descartar elementos las estimaciones de las distancias pueden resultar poco ajustadas, lo que influye directamente en el rendimiento del índice.

4.5. BOND

La técnica *Branch and bound on Decomposed Data* (BOND) [14] es un método de acceso espacial (SAM por su sigla en inglés) que permite realizar consultas con una combinación de características.

El índice almacena tablas con los coeficientes de cada dimensión para cada vector de la base de datos, y luego las analiza secuencialmente calculando cotas inferiores y superiores de las distancias entre los objetos. Para calcular las cotas son necesarias dos cosas, que los datos estén acotados en el espacio y derivar las cotas acorde a cada métrica utilizada. En el algoritmo sólo derivan las cotas para dos tipos de métricas, una modificación de la distancia euclidiana y una métrica para histogramas.

A pesar de lo anterior, los mayores problemas de esta técnica son la escalabilidad, ya que en el peor caso se puede requerir espacio adicional proporcional al tamaño de la base de datos, y que sólo funciona con espacios vectoriales, restringiendo la aplicabilidad de la técnica.

Capítulo 5

Adaptación de algoritmos para espacios multimétricos

En este capítulo se presenta una metodología estándar para adaptar una estructura de indexamiento de espacios métricos para indexar espacios multimétricos. Luego se explica cómo este método es aplicado sobre las estructuras GNAT y List of Clusters. Se muestra también cómo la estructura M^3-Tree y el índice basado en pivotes encajan dentro de este modelo de transformación.

5.1. Método de Adaptación

5.1.1. Cotas

Las estructuras de indexamiento de espacios métricos, ya sean basadas en pivotes o en particiones compactas (ver 2.3), almacenan distancias entre objetos, las que posteriormente son utilizadas para analizar si un objeto pertenece a la consulta.

Las distancias almacenadas corresponden a uno de los siguientes tipos:

1. Distancia entre dos objetos $d = \delta(x, y)\ x, y \in \mathbb{U}$
2. Distancia máxima de un objeto a un conjunto $d = \text{máx}_{x \in C}\, \delta(x, y)\ y \in \mathbb{U}, x \in C \subseteq \mathbb{U}$
3. Distancia mínima de un objeto a un conjunto $d = \text{mín}_{x \in C}\, \delta(x, y)\ y \in \mathbb{U}, x \in C \subseteq \mathbb{U}$

Notar que el caso 1 es un caso particular de 2 y 3.

Cuando se trabaja con espacios multimétricos, la función de distancia no se conoce de antemano, solo se conoce al momento de realizar la consulta, momento en el cual se definen los pesos. Es por esta razón que un índice multimétrico generalmente tiene que estimar las distancias.

Cuando la distancia es de tipo 1, la distancia puede ser calculada de manera exacta. Para esto es necesario almacenar las componentes de la distancia entre ambos objetos, y posteriormente ponderarlas por los pesos y sumarlos.

Si la distancia es de tipo 2 ó 3, es necesario estimar la distancia. A continuación se presentan dos lemas que permiten estimar estas distancias.

Sean $\mathbb{X}$ un espacio multimétrico, $C \subseteq \mathbb{X}$ un conjunto cualquiera, y $y \in \mathbb{X}$, y sean

$$r^{\mathbb{W}}_{max} = \max_{x \in C} \Delta_{\mathbb{W}}(x, y) = \max_{x \in C} \sum w_i \delta_i(x, y) \tag{5.1}$$

$$r^{\mathbb{W}}_{min} = \min_{x \in C} \Delta_{\mathbb{W}}(x, y) = \min_{x \in C} \sum w_i \delta_i(x, y) \tag{5.2}$$

donde $\mathbb{W}$ es un vector de pesos con valores en [0,0, 1,0] y donde al menos un peso es no nulo. Se considera también la notación $\mathbb{W} = 1{,}0$, que significa que todos los pesos del vector son iguales a 1,0.

Lema 1. *Cota basada en pesos*

$$r^{\mathbb{W}}_{max} \leq r^{\mathbb{W}}_{cs1} = \max w_i \, r^{1,0}_{max} \tag{5.3}$$

$$r^{\mathbb{W}}_{min} \geq r^{\mathbb{W}}_{ci1} = \min w_i \, r^{1,0}_{min} \tag{5.4}$$

Demostración.

$$r^{\mathbb{W}}_{max} = \max_{x \in C} \sum w_i \delta_i(x, y) \leq \max_{x \in C} \sum \max w_i \delta_i(x, y) = \max w_i \max_{x \in C} \sum \delta_i(x, y) = \max w_i \, r^{1,0}_{max}$$

□

Lema 2. *Cota basada en componentes de distancia*

$$r^{\mathbb{W}}_{max} \leq r^{\mathbb{W}}_{cs2} = \sum w_i \max_{x \in C} \delta_i(x, y) \tag{5.5}$$

$$r^{\mathbb{W}}_{min} \geq r^{\mathbb{W}}_{ci2} = \sum w_i \min_{x \in C} \delta_i(x, y) \tag{5.6}$$

Demostración.

$$r^{\mathbb{W}}_{max} = \max_{x \in C} \sum w_i \delta_i(x, y) \leq \max_{x \in C} \sum w_i \max_{z \in C} \delta_i(z, y) = \sum w_i \max_{z \in C} \delta_i(z, y)$$

□

Corolario 1. *Combinando los lemas 1 y 2 se tiene que*

$$r^{\mathbb{W}}_{max} \leq r^{\mathbb{W}}_{cs} = \text{mín}(r^{\mathbb{W}}_{cs1}, r^{\mathbb{W}}_{cs2}) \tag{5.7}$$
$$r^{\mathbb{W}}_{min} \geq r^{\mathbb{W}}_{ci} = \text{máx}(r^{\mathbb{W}}_{ci1}, r^{\mathbb{W}}_{ci2}) \tag{5.8}$$

En conclusión, para estimar una distancia del tipo 2 ó 3 es necesario contar con el valor de la distancia para la métrica $\Delta_{1,0}$, así como también los valores máximos o mínimos por componentes de la distancia.

5.1.2. Algoritmo de Adaptación

Los pasos a seguir para modificar un índice métrico en multimétrico son los siguientes:

1. Identificar los tipos de distancias involucradas en el índice.

2. Verificar que los invariantes de la estructura se mantengan al cambiar los pesos.
   En este punto hay que verificar si dado un índice construido con una multimétrica definida (i.e., los pesos están fijos), este índice mantiene los invariantes de la estructura al cambiar los pesos. Este paso es crucial para asegurar la correctitud de los resultados de las consultas.

3. Modificar construcción de la estructura.
   La construcción del índice se realiza con la multimétrica $\Delta_{1,0}$, es decir, todos los pesos son iguales a 1,0. Cada vez que se almacena una distancia, se almacenan además las distancias por componentes, dependiendo del tipo de distancia que sea. En este paso puede ser necesario almacenar nuevas distancias que aseguren que se mantenga el invariante de la estructura.

4. Modificar algoritmos de consultas usando las cotas.
   Partiendo de los algoritmos de consulta del índice métrico es necesario modificar las condiciones que involucran distancias.

   - Distancia Tipo 1: se calcula exactamente la distancia y se mantiene la condición
   - Distancia Tipo 2 ó 3: en estos tipos de distancia la forma en que cambia la condición dependerá de la acción que sea realizada en caso que esta sea verdadera.

La explicación que sigue corresponde a una distancia de tipo 2, pero es análogo para una de tipo 3. Se pueden identificar dos tipos de acciones básicas a realizar, la primera es continuar con la búsqueda en una porción de la estructura y la segunda corresponde a podar la búsqueda.

En el primer caso necesitamos asegurar que cada vez que se cumpla la condición con los nuevos pesos, se cumpla también para la estimación. Es decir, necesitamos que $cond(d) \Rightarrow cond(d_{cs})$, ya que es necesario asegurar que todas las "ramas" necesarias de la estructura sean revisadas. En el segundo caso, necesitamos asegurar que cada vez que se cumpla la condición para la estimación de las distancias entonces se cumpla también para la distancia exacta, esto ya que no es correcto terminar una búsqueda antes de tiempo. La relación que debe cumplirse es $cond(d_{cs}) \Rightarrow cond(d)$.

Generalmente se tiene que las distancias de tipo 2 con una acción de búsqueda corresponden a una condición del tipo $d \leq a$, y las de tipo 2 con acción de poda, con una condición del tipo $d \geq a$. Se observa que las condiciones anteriores satisfacen las propiedades requeridas para asegurar la correctitud del resultado, esto es $d \geq a \Rightarrow d_{cs} \geq a$ (búsqueda) y $d_{cs} \leq a \Rightarrow d \leq a$ (poda).

En caso que las condiciones del algoritmo sean las descritas anteriormente se pueden aplicar directamente las cotas obtenidas en el Corolario 1, quedando así que la primera condición se cambia por $d_{cs} \geq a$, y la segunda por $d_{cs} \leq a$.

En este paso puede ser necesario agregar nuevas condiciones que aseguren que se mantenga el invariante de la estructura.

5.2. Multimetric GNAT (MMGNAT)

Para adaptar la estructura se seguirán los pasos propuestos en la sección anterior.

Las distancias almacenadas en GNAT corresponden a los rangos de cada *split point* a las zonas. El rango es de la forma $[min_d(p, D_q), max_d(p, D_q)]$, donde la distancia mínima es de

tipo 2 y la máxima es de tipo 3.

Es importante notar que a pesar que en la construcción de la estructura se asocia cada punto con el *split point* más cercano, está condición no es utilizada de manera alguna en las búsquedas. Es por esta razón que la estructura de GNAT no tiene ningún invariante relevante, por lo que se puede modificar directamente para indexar espacios multimétricos.

5.2.1. Construcción

Ya que no fue necesario modificar el invariante de la estructura, el nuevo algoritmo de construcción es bastante similar al original. Se realiza la construcción con la multimétrica $\Delta_{1,0}$, y cada vez que se almacene la distancia máxima de un rango se almacena además la máxima distancia de cada componente a la zona. Lo mismo se realiza para la distancia mínima. Esto se debe a que son distancias de tipo 2 y 3.

5.2.2. Búsqueda por Rango

Se puede ver en el Algoritmo 2 que la única condición que depende de las distancias almacenadas es la que aparece en la línea 7. Esta condición equivale a lo siguiente:

7 **if** $max_d(p, D_q) < dist(x, p) - r \; or \; min_d(p, D_q) > dist(x, p) + r$ **then** remover q de P

Es decir, se tiene que las distancias máximas están involucradas en condiciones del tipo $d < a$ y las distancias mínimas en condiciones de la forma $d > a$, y como la acción realizada en este caso corresponde a una poda, podemos reemplazar los valores de las distancias directamente por las respectivas cotas, a raíz de lo explicado en la sección anterior.

Con esto, el algoritmo de búsqueda queda como sigue:

En la línea 7, se utiliza el corolario 1 para estimar la distancia máxima y mínima del rango.

```
1  function Search (N,x,r)
2      let P=split points(N)
3      foreach p ∈ P do
4          Calcular d(p, x)
5          if d(p, x) ≤ r then reportar p
6          foreach q ∈ P do
7              let range_e(p, D_q) = Estimar range(p, D_q)
8              if [d(x, p) − r, d(x, p) + r] ∩ range_e(p, D_q) = ∅ then remover q de P
9      foreach p ∈ P do Search (D_p,x,r)
10 end
```

Algoritmo 8: Búsqueda por Rango en MMGNAT

5.2.3. k-Vecinos más cercanos

Se puede ver que en este algoritmo las distancias almacenadas son utilizadas solamente para estimar la distancia del punto de consulta a una zona (ver líneas 19 y 20 del Algoritmo 3). Si las distancias de un rango son reemplazadas por las respectivas cotas, lo que se obtiene es una estimación del rango real. Esta estimación contiene el rango exacto, por lo que todos los puntos de una zona estarán incluidos en el rango estimado. Con esto, se tiene que la estimación de la distancia de un punto a una zona va a ser incluso menos ajustada que la estimación que se tenía para GNAT, pero seguirá siendo una estimación válida de la distancia.

El algoritmo se puede ver en el Algoritmo 9.

5.3. Multimetric List of Clusters (MMLCluster)

Para adaptar la estructura se seguirán los pasos propuestos en la Sección 1 del presente capítulo.

Las distancias almacenadas en List of Clusters corresponden al radio de cada cluster. Es decir, corresponde a una distancia de tipo 3, ya que esta distancia se calcula como la máxima distancia del centro a todos los puntos del cluster.

5.3.1. Invariante

El invariante de List of Clusters es que si un punto puede pertenecer a dos clusters, este necesariamente estará incluido en el primero que fue creado. Este invariante permite que

```
1  function Knn (N,x,k)
2      let Q=priorityQueue()
3      let cnt = 0
4      enqueue((root,0),Q)
5      while Q ≠ ∅ do
6          let element = dequeue(Q)
7          if element is Spatial object then
8              report element
9              cnt++
10             if cnt = k then return
11         else
12             foreach p ∈ P do
13                 Calcular d(p, x)
14                 enqueue((p,d(p,x)),Q)
15             foreach p ∈ P do
16                 let dist = 0
17                 foreach q ∈ P do
18                     let range_e(p, D_q) = Estimar range(p, D_q)
19                     if d(q, x) − min_ci(q, D_p) > dist then dist = d(q, x) − min_ci(q, D_p)
20                     if max_cs(q, D_p) − d(q, x) > dist then dist = max_cs(q, D_p) − d(q, x)
21                 enqueue((p,dist),Q)
22 end
```

Algoritmo 9: k-Vecinos más cercanos en MMGNAT

la búsqueda por rango pueda ser podada, disminuyendo el número de comparaciones. Si se construye la estructura, digamos con la multimétrica $\Delta_{1,0}$, y luego se cambian los pesos, entonces la estructura construida no cumplirá el invariante para la nueva multimétrica, ya que los puntos pueden cambiar de cluster. En la Figura 5.1 se puede ver gráficamente como los elementos pueden cambiar de cluster al cambiar los pesos. Se tiene una base de datos que combina dos espacios vectoriales de una dimensión. En la imagen de la izquierda se muestra el cluster definido por q_1 cuando la multimétrica es $\Delta_{1,0}$, y a la derecha se observa como debiese quedar el cluster cuando se la multimétrica se cambia por $\Delta_{\{0,0,1,0\}}$. En la imagen de la derecha se aprecia que ahora el punto q_4 pertenece al cluster.

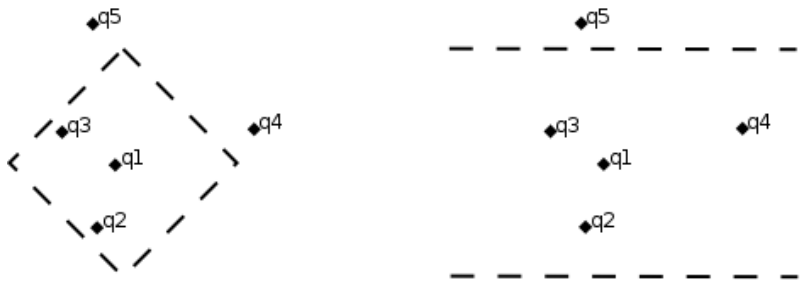

Figura 5.1: Puntos cambian de cluster cuando se cambian los pesos.

En la estructura List of Clusters original, se realiza la poda cada vez que la bola de consulta está completamente contenida en algún cluster. Esta poda puede ser realizada debido a que, por el invariante, todos los puntos que siguen en la lista no pueden pertenecer al cluster.

Para asegurar que el invariante sea conservado al cambiar los pesos, se reinterpreta la forma en que se realiza la poda. Si introducimos una nuevo radio $r_{externo}$, que indique que todos los puntos que siguen en la lista están a una distancia $d \geq r_{externo}$ del centro del cluster, la poda puede ser reinterpretada de manera que el invariante sea conservado. Con este nuevo radio la poda será realizada cada vez que la bola de consulta esté completamente contenida en la bola definida por el centro del cluster y $r_{externo}$. La reinterpretación propuesta de la poda no se contradice con el invariante original de la estructura, ya que en el caso de pesos fijos se tiene que el radio del cluster es menor que el radio externo, es decir, se cumple que $r \leq r_{externo}$. Pero está modificación permite que, al cambiar los pesos, el invariante se siga cumpliendo, ya que el radio externo corresponde a una distancia de tipo 2, lo que permite

obtener una cota inferior de su valor.

En la Figura 5.2 se observa como queda la estructura con el nuevo radio ingresado. Los puntos p pertenecen al *cluster* definido por c_1 y los puntos o pertenecen al cluster definido por c_2. Notar que o_1 y o_3 pertenecen al cluster c_2 a pesar que su distancia a c_1 es menor que r, esto se debe a que su distancia es mayor que $r_{externo}$, por lo que el invariante no asegura que pertenezcan al primer cluster creado.

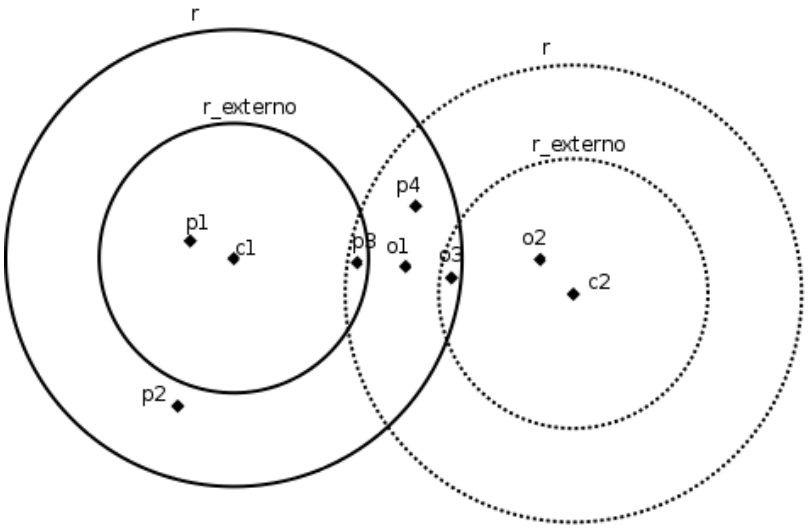

Figura 5.2: Nuevas distancias para MMLCluster.

En la figura 5.3 se observan los criterios para buscar o realizar una poda en caso de una consulta por rango. Para la bola de consulta q_1 no es necesario buscar en el cluster actual, para q_2 como la bola está completamente contenida en el radio externo se termina la búsqueda luego de revisado el *cluster* y para q_3 y q_4 es necesario revisar el *cluster* y seguir revisando la lista. Es importante notar que para q_4 en el caso de List of Clusters la búsqueda terminaría por estar completamente dentro del radio, sin embargo, el nuevo invariante no permite realizar esa poda.

Lema 3. *El nuevo invariante se mantiene si los pesos se cambian.*
Sea C el conjunto de todos los objetos que ya han sido agregados a un cluster *y c el último* cluster *creado, se cumple que*

$$p \notin C \Rightarrow \Delta_{\mathbb{W}}(p, c) \geq r^{\mathbb{W}}_{externo} \tag{5.9}$$

es decir, todos los objetos que aún no han sido agregados a un cluster *están a una distancia mayor o igual que $r^{\mathbb{W}}_{externo}$.*

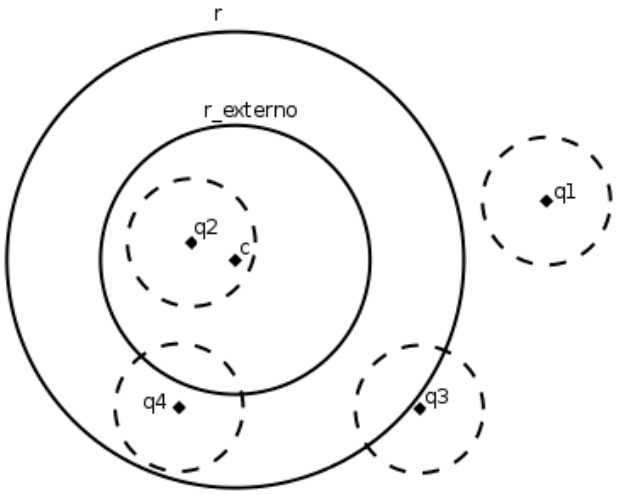

Figura 5.3: Búsqueda por rango en MMLCluster.

Demostración. Por definición, $r^{\mathbb{W}}_{externo} = \min_{x \in \mathbb{U}-C} \Delta_{\mathbb{W}}(c, x)$ y como $p \notin C$, se cumple entonces que $r^{\mathbb{W}}_{externo} \leq \Delta_{\mathbb{W}}(c, p)$. □

5.3.2. Construcción

Para construir el nuevo índice se realiza un indexamiento del espacio con la multimétrica $\Delta_{1,0}$. Cada vez que se crea un cluster, se almacena, además del radio, la máxima distancia por componente del centro a cada elemento del *bucket*. Para mantener el nuevo invariante es necesario también guardar el radio externo, que corresponde a la mínima distancia del centro a cada uno de los elementos que aún no han sido considerados en ningún cluster, y la mínima distancia por componente del centro a todos los elementos que faltan por ser añadidos al cluster. Esto se debe a que las distancias almacenadas son del tipo 2 y 3.

5.3.3. Búsqueda por rango

La búsqueda por rango debe ser adaptada de manera que utilice el nuevo invariante propuesto, cambiando la condición que se relacionaba con la poda. Las condiciones presentes son $r_c \geq a$ y $r_{externo} \leq a$, y en ambas se sigue con la búsqueda en caso de ser verdaderas, por lo que no hay problema si reemplazamos r_c y $r_{externo}$ por sus respectivas cotas superiores e inferiores.

Tomando en consideración lo anterior el algoritmo queda como se muestra en el Algoritmo 10.

```
1  function Search (L,q,r)
2      if L = ∅ then return
3      let L = (c, r_c, I) : E
4      Calcular d(c, q)
5      let r_c = Estimar r_c
6      let r_e = Estimar r_externo
7      if d(c, q) ≤ r then agregar c a la lista de resultados
8      if d(c, q) ≤ r_c + r then buscar exhaustivamente en I
9      if d(c, q) > r_e − r then Search (E,q,r)
10 end
```

Algoritmo 10: Búsqueda por Rango de MMLCluster

5.3.4. k-Vecinos más cercanos

Si nos fijamos en la línea 6 del Algoritmo 6, notamos que es la misma condición que la línea 7 del Algoritmo 5, y la línea 9 del Algoritmo 6 tiene la misma condición que la línea 6 del Algoritmo 5, por lo que las transformaciones son iguales a las realizadas en la búsqueda por rango.

Con esto el algoritmo queda como se muestra en el Algoritmo 11.

```
1  function Knn (cluster,k,q,res)
2      if cluster > size(list) then return
3      Calcular d(cluster.c, q)
4      if dist ≤ cluster.r then
5          searchBucketKnn(cluster.bucket,k,q,res)
6          let r_e = Estimar cluster.r_externo
7          if size(res) < k or d(cluster.c, q) + res.maxdist ≥ r_e then Knn
           (cluster.next,k,q,res)
8      else
9          Knn (cluster.next,k,q,res)
10         let r_c = Estimar cluster.r_c
11         if size(res) < k or d(cluster.c, q) − res.maxdist ≤ r_c then
           searchBucketKnn(cluster.bucket,k,q,res)
12 end
```

Algoritmo 11: Búsqueda de K-Vecinos más cercanos en MMLClusters

5.4. M^3-Tree

La estructura M^3-Tree [7] resulta de aplicar el método antes presentado al M-Tree.

En el M-Tree se tienen dos tipos de distancias, la distancia al padre que es una distancia del tipo 1, y la máxima distancia de un nodo a cada uno de sus descendientes, que es de tipo 2. Luego en la construcción del M^3-Tree se indexa el espacio con la multimétrica $\Delta^{1,0}$, y se almacenan estas distancias así como las respectivas componentes. Ningún cambio adicional es necesario en la estructura, debido a la ausencia de invariante.

La adaptación de los algoritmos de búsqueda es similar a como se hizo para GNAT o List of Clusters.

La única diferencia existente entre M^3-Tree y la estructura resultante de aplicar el método al M-Tree es el hecho que en el M^3-Tree en vez de almacenar directamente las componentes de la distancia se guarda una estimación de la componente (i.e., se utilizan menos bits), lo que permite reducir la memoria requerida por la estructura.

5.5. Pivot Based

El índice multimétrico basado en pivotes [6], también puede ser visto como la aplicación directa del método antes presentado.

La versión métrica del índice basado en pivotes almacena la distancia de todos los puntos a cada uno de los pivotes. Es decir, corresponde a una distancia del tipo 1. Por lo que para transformarlo a un índice multimétrico solo basta con almacenar las componentes de las distancias.

Capítulo 6

Evaluación Experimental

En el presente capítulo se presentan los resultados de los distintos experimentos realizados. En la primera sección se presenta la metodología y en las secciones siguientes se presentan los resultados para las distintas bases de datos utilizadas. Finalmente se muestra un análisis de los datos obtenidos.

6.1. Metodología

Las pruebas serán realizadas en 3 bases de datos reales: Objetos 3D de la Universidad de Konstanz [5], Corel de la colección UCI-KDD [17] y las características de la imágenes de Flickr obtenidas por CoPhIR [13]. La primera corresponde a una base de datos de objetos 3D y las dos últimas corresponden a bases de datos de imágenes. Para todas las bases de datos la métrica utilizada corresponde a las distancia L_1 o Manhattan (ver Sección 2.1.2).

En primer lugar se reduce la dimensión de los vectores característicos, en caso de ser necesario. Luego se normalizan todas las características de modo tal que la distancia máxima entre dos objetos cualquiera de la base de datos sea 1,0, de este modo se asegura que no exista preferencia por sobre una característica en particular. Esta normalización toma tiempo $O(n^2)$ en caso de ser realizada exactamente, por lo que en este caso se realiza de forma aproximada, considerando un número fijo de objetos por cada elemento de la base de datos. Finalmente se combinan las características y se extrae un porcentaje que será usado como objetos de consulta y el resto se considera la base de datos a consultar.

Las pruebas a realizar pueden ser dividas en dos grupos: el primero en el que sólo se

experimenta con las versiones multimétricas de los índices y el segundo, en que además se construye el índice métrico óptimo dados los pesos. Por lo costoso que resulta construir un índice por cada peso, para esta última prueba, se trabajó solamente con 5 vectores de pesos diferentes en cada consulta.

Para ambos grupos se realizan las siguientes pruebas:

1. Consultas de 10-NN con pesos aleatorios en el intervalo $[w, w + 0{,}1]$.
2. Consultas de k-NN con pesos en el rango $[0, 0{,}1]$, variando k entre 1 y 64.

Además, para el caso en que se trabaja tanto con las versiones métricas como multimétricas se realiza la siguiente prueba:

3. Consultas de k-NN con un peso igual a 1,0 y todo el resto igual a 0,0, variando k entre 1 y 10 .

Para cada consulta se contabilizarán el número de distancias calculadas que realiza cada uno de los índices.

6.1.1. Pesos

En las pruebas, los pesos pertenecen a rangos del tipo $[w, w + 0{,}1]$. Se eligieron estos rangos, debido a que en una consulta de k-NN el resultado es invariante si se escalan todos los pesos. Es por esto que el rango $[w, w + 0{,}1]$ es equivalente a $[\frac{w}{w+0{,}1}, 1{,}0]$. Con esto la "complejidad" de la consulta para un índice multimétrico aumenta a medida que el valor de w disminuye, ya que los pesos se van alejando cada vez más de los pesos originales.

Es importante notar que una consulta con pesos en el rango $[0{,}0, 0{,}1]$ es equivalente a una con los pesos en el rango $[0{,}0, 1{,}0]$, que es el caso más general que se puede tener.

6.1.2. Reducción dimensional

En algunos casos, debido a la alta dimensionalidad del espacio multimétrico resultante, puede ser necesario disminuir el número de dimensiones del espacio. Para lograr este objetivo

se utiliza el método de análisis de componentes principales (PCA por su sigla en inglés) [19].

El método se detalla a continuación:

- Se calcula el promedio de cada coordenada y se centra cada coordenada en el promedio.
- Se calcula la matriz de covarianza.
- Se calculan los vectores y valores propios.
- Se eligen aquellas coordenadas con mayor valor propio.

6.2. Objetos 3D

Esta es una base de datos de objetos 3D que posee 16 características, cuya dimensión va desde 32D hasta 510D. Esta base de datos tiene 1.838 elementos, entre los que se encuentran modelos de plantas, automóviles, tazas, sillas, mesas, etc. Del total de elementos se consideraron 1.654 para la base de datos de prueba y los 184 (10 %) elementos restantes se dejaron como elementos de consulta.

Para las pruebas se eligieron 6 características, estas son: 3DDFT (365D), CPX (484D), GRAY (496D), H3D (112D), SIL (510D), VOX (343D). Dada la alta dimensionalidad de estas características es que se les disminuyó la dimensión utilizando PCA. La dimensión de cada característica fue reducida a 8D y 16D, obteniendo así una dimensión combinada de 48D y 96D respectivamente.

En estas pruebas GNAT se construyó con aridad 5, M-Tree con aridad 2, y List of Clusters con 10 elementos por cluster.

6.2.1. 8D

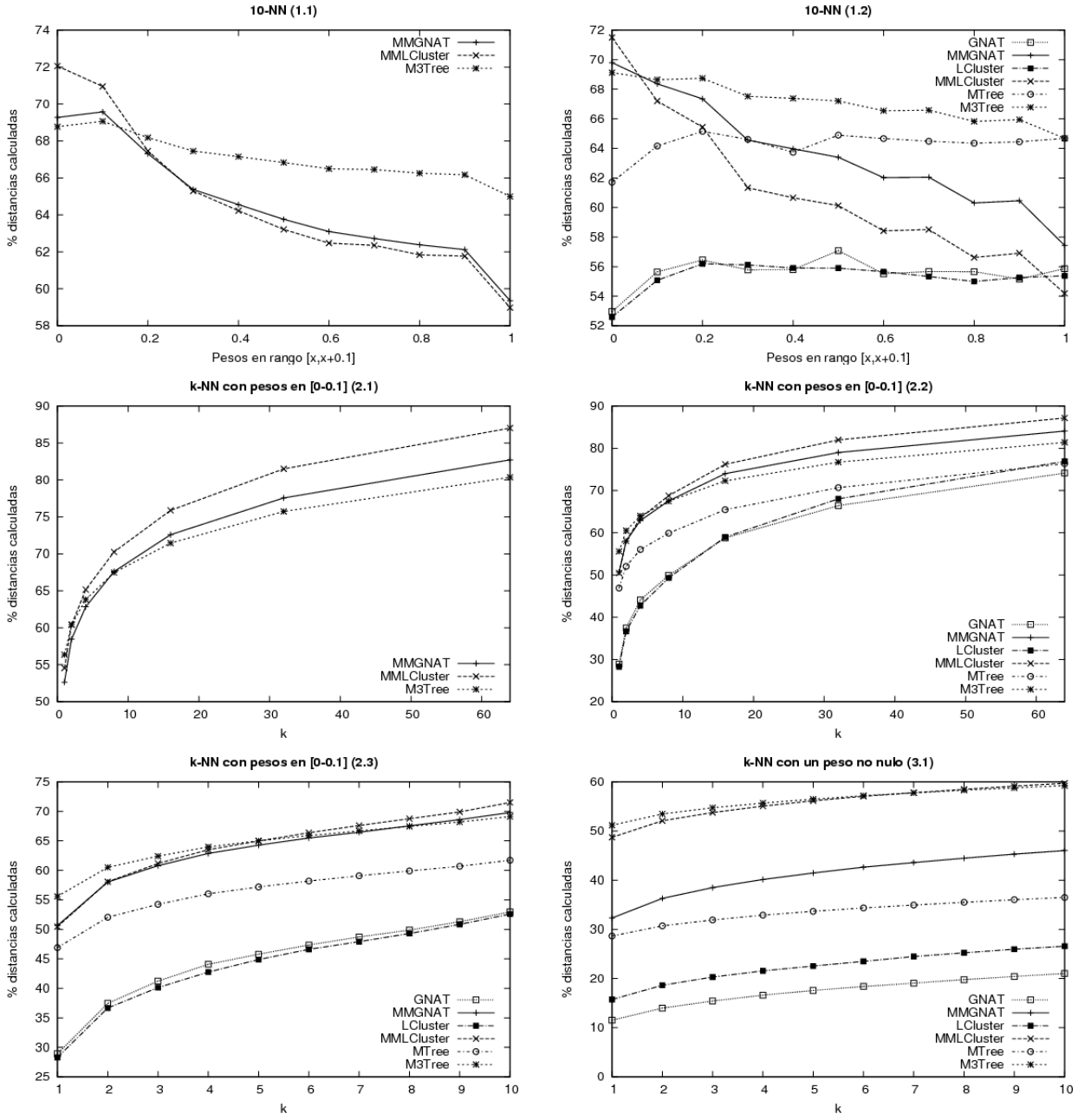

Figura 6.1: Objetos 3D: Resultados 8D

6.2.2. 16D

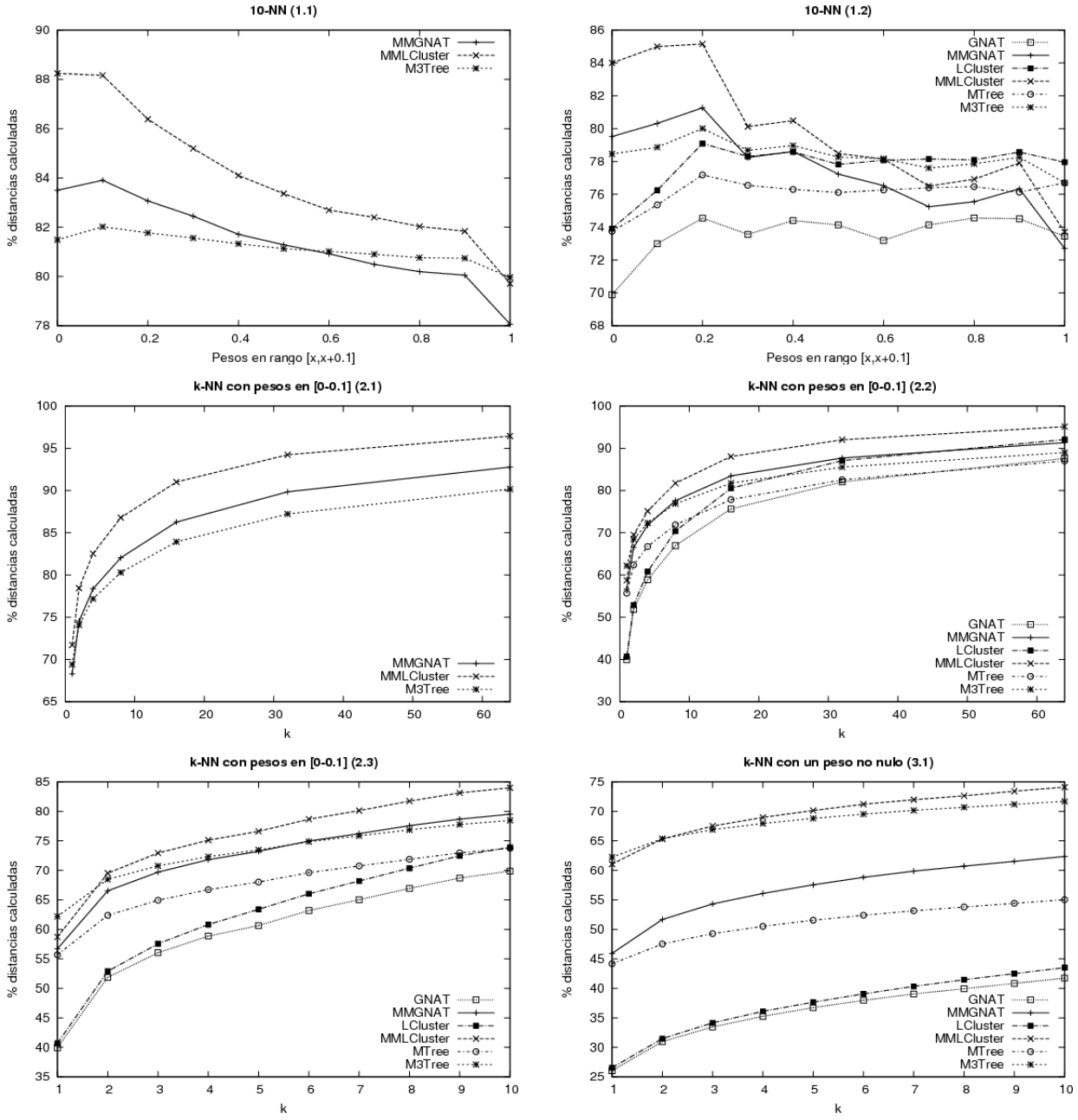

Figura 6.2: Objetos 3D: Resultados 16D

6.3. Corel

Esta base de datos corresponde a una base de datos de imágenes que posee 65.615 elementos y 4 vectores característicos, estos son: *Color Histogram* (32D), *Color Moments* (9D), *Cooc Texture* (16D) y *Layout Histogram* (32D), lo que da un total de 89D al combinar todas las características.

De los 65.615 elementos de la base de datos se extrajo aleatoriamente el 2 % (1.312 elementos), elementos que fueron considerados como objetos de consulta. El resto de los objetos corresponde a la base de datos de prueba.

En esta prueba GNAT se construyó con aridad 5, M-Tree con aridad 20, y List of Clusters con 200 elementos por cluster. Los valores de List of CLusters y M-Tree se aumentaron debido a que el tiempo de construcción era muy alto con los parámetros utilizados en la prueba anterior. Recordemos que el costo de construir List of Clusters es $O(n^2/m)$ (ver Sección 3.2.1).

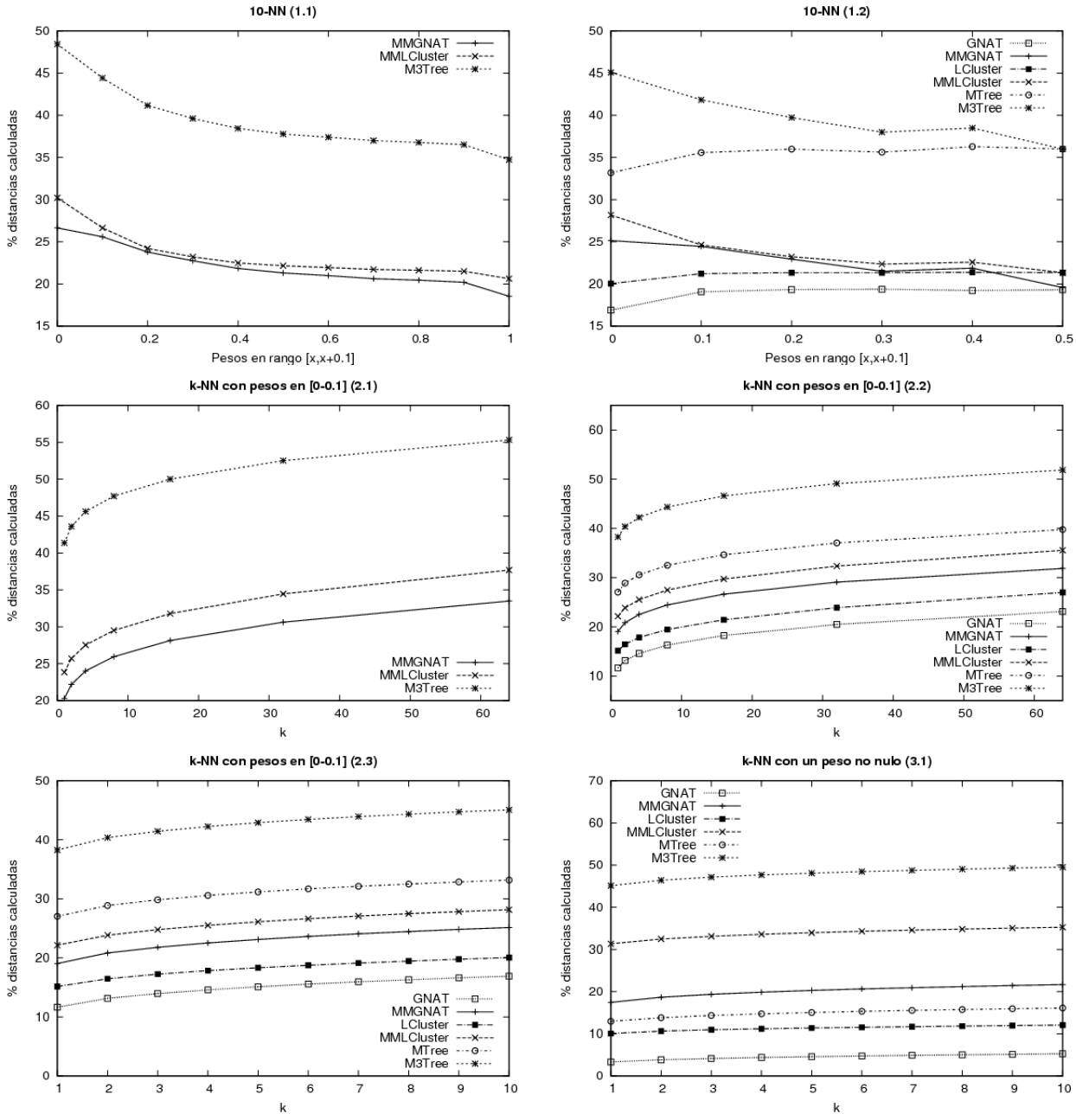

Figura 6.3: Corel: Resultados

6.4. Flickr

El grupo CoPhiR ha extraído 5 características para más de 50 millones de imágenes del sitio web de internet Flickr. Dado que los índices que se están evaluando en este trabajo están pensados para memoria principal, en los experimentos se trabajó solamente con 200 mil imágenes de esta base de datos. Los vectores característicos que posee esta colección son: *Color Layout* (12D), *Edge Histogram* (80D), *Scalable Color* (64D), *Color Structure* (64D) y *Homogeneous Texture* (62D).

Dado que las dimensiones combinadas dan un total de 282D, la pruebas se realizaron también con una versión de la base de datos en que todas las características tienen una dimensión igual a 12D.

De las 200.000 imágenes, 1.000 (0,5 %) fueron consideradas como objetos de consulta y el resto como la base de datos de prueba.

En estas pruebas los índices se construyeron con los mismos parámetros que en la prueba anterior.

6.4.1. Dimensiones originales

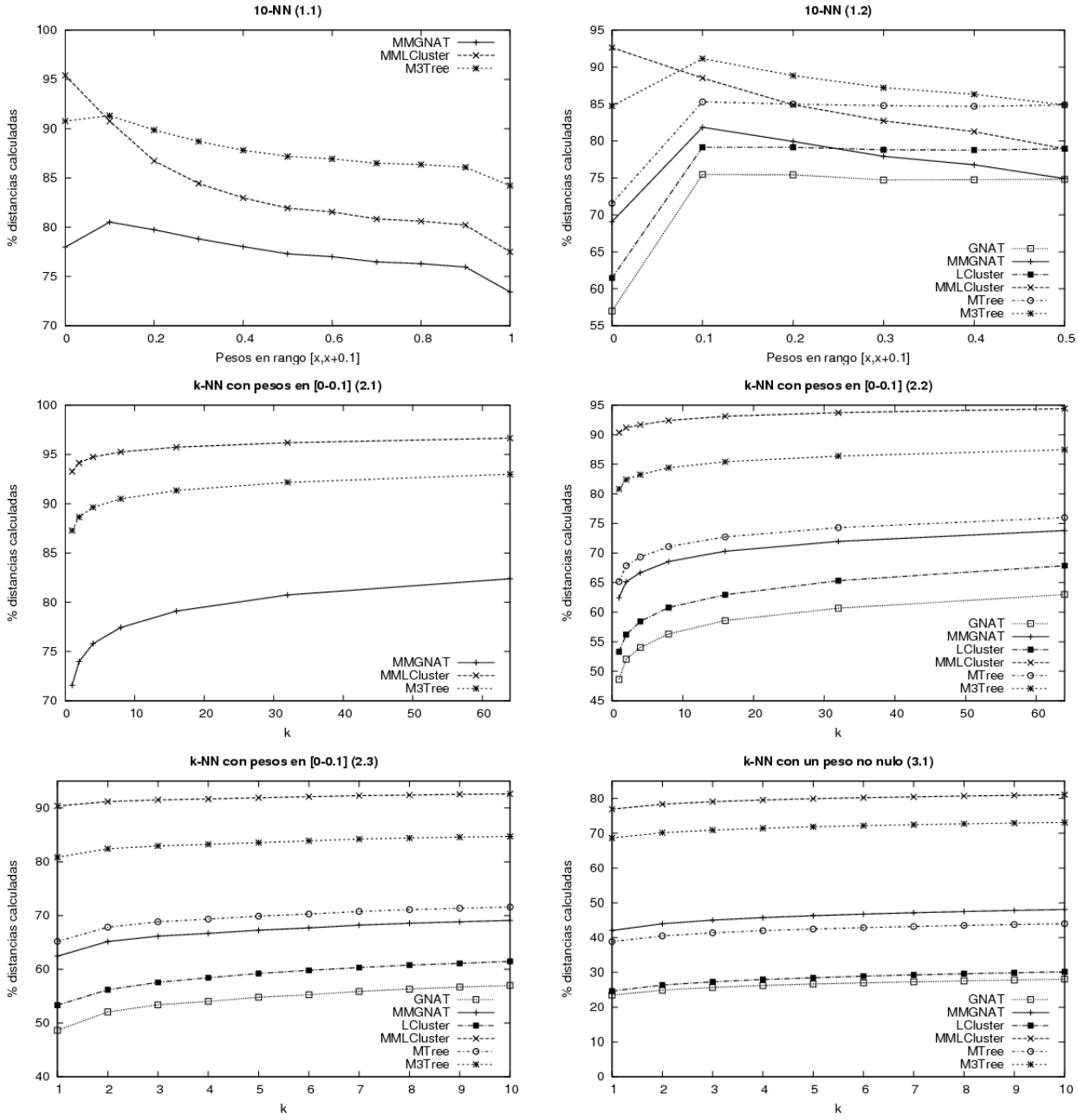

Figura 6.4: Flickr: Resultados

6.4.2. 12D

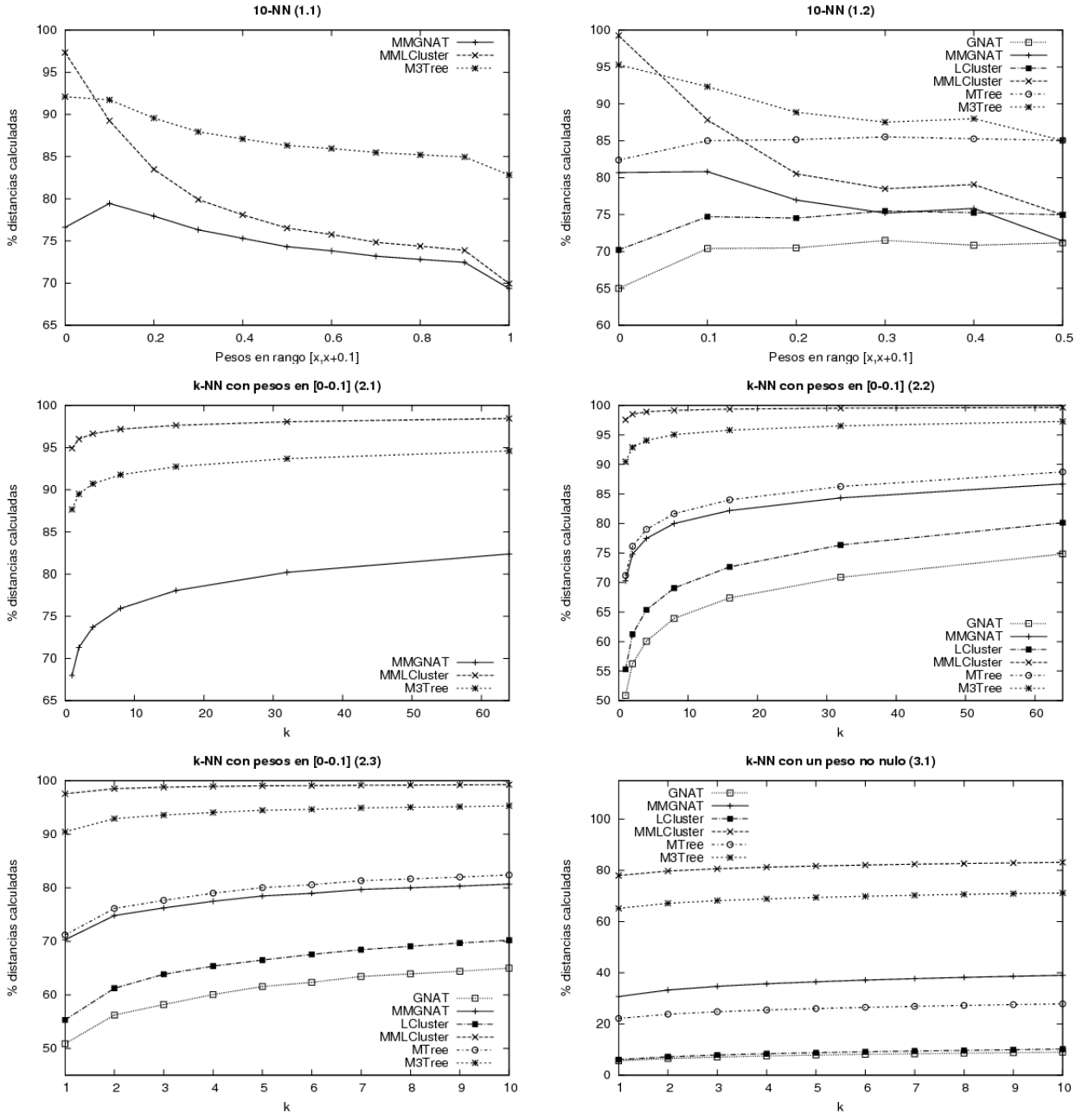

Figura 6.5: Flickr 12D: Resultados

6.5. Análisis

A partir de los resultados obtenidos para cada una de las distintas bases de datos, se obtiene que el índice métrico que mejor funciona es GNAT y MMGNAT en el caso de índice multimétrico. Este resultado es contrario a lo obtenido en [8], sin embargo, dichas pruebas fueron realizadas con datos artificiales, en que las bases de datos correspondían a objetos distribuidos uniformemente con dimensión inferior a 20. Además en esas pruebas solo se trabajó con consultas por rango.

Con respecto a *List of Clusters* se observa que su versión métrica muestra un desempeño bastante similar al de GNAT, aunque un poco inferior. Sin embargo, MMLCluster no se comporta tan bien como LCluster. Esto se debe al hecho que List of Clusters almacena muy poca información, solo una distancia por *cluster* (2 si consideramos MMLCluster). Esto hace que al momento de estimar las distancias en el caso multimétrico, las cotas no queden muy ajustadas. Además, al comparar los resultados obtenidos para la base de modelos 3D con dimensión 8 y 16, se observa que MMLCluster resulta no ser tan resistente al aumento de dimensionalidad como MMGNAT o M^3-Tree. Con dimensión 8 se obtiene que MMLCluster resulta ser el mejor índice, pero al aumentar la dimensión a 16 este termina mostrando el peor desempeño.

En caso de que los pesos sean todos iguales a 1,0, el índice multimétrico se debiese comportar tal como el índice métrico, pues el índice multimétrico se construye con estos pesos (ver Sección 5.1.2). Sin embargo, en la Figura 6.2 (1.2) se observa que MMGNAT realiza menos cálculos de distancias que la versión métrica, esto se debe a que durante la construcción de GNAT y MMGNAT los *split points* son elegidos aleatoriamente, por lo que cada vez que se construye un índice resultará uno diferente. Para el caso de LCluster podemos ver en la Figura 6.1 (1.2) que la versión multimétrica se comporta mejor que su contraparte métrica. La razón de este hecho se debe a que MMLCluster no es exactamente la misma estructura que LCluster. En la Sección 5.3.1, se muestra que es necesario añadir una distancia original a LCluster de modo de poder asegurar el invariante en el caso multimétrico. Es esta distancia adicional la que explica por qué MMLCluster resulta más eficiente que LCluster en el caso señalado, pues permite realizar más podas que en el caso de LCluster.

Se observa también que a medida que disminuye w, las distancias calculadas en la prueba número 1 van en aumento, lo cual es el resultado esperado, a excepción de cuando se llega a $w = 0$. Lo anterior se observa en la Figura 6.4 (1.1). Esto ocurre, ya que, a pesar de que el rango de pesos va disminuyendo, no necesariamente lo hace la distancia al k-ésimo elemento encontrado, medida que indica la dificultad de realizar la búsqueda. Para graficar lo anterior se muestra en la Figura 6.6 la "distancia" al k-ésimo elemento. Como los pesos varían, no se pueden comparar las distancias directamente, ya que éstas están escaladas. Es por esto que en la gráfica se muestra la distancia obtenida en la búsqueda de 10-NN, dividida por $w + 0{,}05$, que corresponde a la esperanza del intervalo de los pesos. De esta manera todas las distancias son ahora comparables. En la imagen se observa como la distancia al k-ésimo es más o menos constante a excepción del intervalo $[0, 0{,}1]$, que es mucho menor, lo que significa menor número de distancias calculadas, ya que es posible realizar un mayor número de podas. Este fenómeno se puede deber a las propiedades de los intervalos elegidos. Como se vio en la

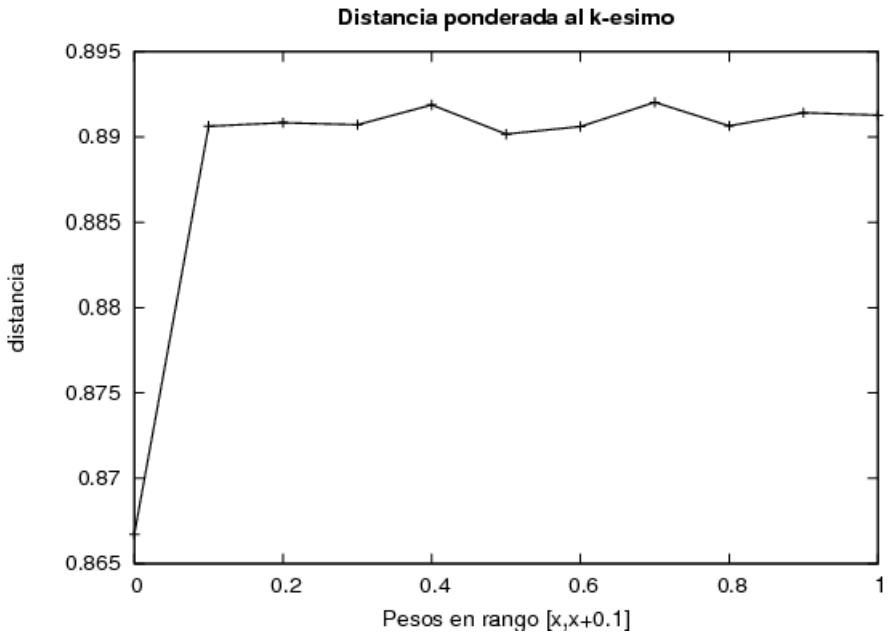

Figura 6.6: Distancia al k-ésimo elemento

Sección 6.1.1, si los pesos pertenecen al intervalo $[0{,}0, 0{,}1]$, es equivalente a que pertenecieran al intervalo $[0{,}0, 1{,}0]$ por lo que es el caso más general. Con estos pesos puede ocurrir que una característica tenga un peso muy bajo por lo que prácticamente no influye en el valor

final de la distancia. En cambio, el intervalo [0,1, 0,2], que es equivalente a [0,5, 1,0] no puede tener pesos muy bajos, por lo que todas las características van a incidir en la distancia final.

En los resultados obtenidos se observa que, en el caso extremo en que sólo una métrica tiene un peso no nulo (ver Figura 6.5 (3.1)), los índices multimétricos muestran un desempeño bastante alejado del observado en sus contrapartes métricas. Sin embargo, es importante destacar que el desempeño de MMGNAT es bastante similar al del M-Tree, lo que muestra la efectividad de este índice multimétrico.

Capítulo 7

Conclusiones

El objetivo de esta memoria fue adaptar dos índices métricos para ser usados con espacios multimétricos, puesto que con la utilización de estos espacios es posible obtener una mejor eficacia al realizar consultas por similitud y hasta el momento existían pocos índices para estos espacios.

Los principales resultados obtenidos en este trabajo se presentan a continuación:

- Se propuso una metodología general que permite adaptar un índice métrico para ser usado como índice de espacios multimétricos. La mayor ventaja de este método es que puede ser usado con cualquier índice métrico, lo que permitiría eventualmente tener la misma cantidad de índices de espacios multimétricos como los existentes para espacios métricos. Esto permitiría además elegir el mejor índice dependiendo de la aplicación, pues este método no se limita a alguna característica en particular del índice. Por ejemplo, puede ser usado tanto en índices que operen en memoria principal o en disco.

- La metodología propuesta muestra buenos resultados, sin embargo, los resultados dependerán en gran medida de la cantidad de información que se disponga para estimar las distancias. Es por esta razón que el índice MMLCluster no mostró muy buen desempeño.

- Se mostró que la metodología puede ser aplicada a 2 estructuras existentes (GNAT y List of Clusters) y que además es un caso general de la construcción de los índices multimétricos existentes (M^3-Tree y Pivot Based).

- Uno de los índices adaptados, MMGNAT, mostró tener un mejor desempeño que el

M^3-Tree, logrando así mejores resultados que los del estado del arte.

- Se mostró también que el índice métrico GNAT muestra un mejor desempeño que lo mostrado anteriormente en otros estudios, siendo robusto a la dimensionalidad del espacio. De hecho, es este índice el que resultó mostrar el mejor desempeño de los tres índices métricos considerados.

- Se contribuyó también con el desarrollo de un algoritmo de k-vecinos más cercanos para GNAT. Si bien el algoritmo fue diseñado en base a la técnica propuesta por Hjaltason y Samet en [18], este algoritmo no había sido mencionado en la literatura.

Luego del trabajo se presentan las siguientes interrogantes que aún no han sido resueltas.

- Al adaptar estructuras basadas en particiones compactas, ¿se comportan estas mejor o peor que las basadas en pivotes?
  Esta pregunta surge del hecho que los índices basados en particiones compactas requieren una menor cantidad de memoria para almacenar el índice, por lo que se dispondrá de menor información para estimar las distancias.

- ¿Es el balanceo de la estructura un factor importante en el rendimiento final de la estructura adaptada?
  Esta pregunta busca explicar si la estructura completamente desbalanceada de List of Clusters fue el motivo que la estructura adaptada mostrara un rendimiento tan bajo.

- ¿Se puede aplicar este método a la mayoría de los índices?
  Existen dos puntos en la metodología que son críticos para adaptar un índice, estos son poder asegurar que se cumple el invariante y adaptar las condiciones involucradas en los algoritmos. Para lo segundo la metodología indica como se modifican algunas condiciones con sus respectivas acciones, pero ¿qué tan general son estas condiciones?, ¿son aplicables al resto de los índices?. Con respecto al invariante, ¿es posible modificar el invariante de un gran número de índices o resulta ser una tarea demasiado compleja?.

Referencias

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

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

[3] B. Bustos. *Index structures for similarity search in multimedia databases.* PhD thesis, Department of Computer and Information Science, University of Konstanz, June 2006.

[4] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Using entropy impurity for improved 3D object similarity search. In *Proc. IEEE International Conference on Multimedia and Expo (ICME'04)*, pages 1303–1306. IEEE, 2004.

[5] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries, Special issue on Multimedia Contents and Management in Digital Libraries*, 6(1):39–54, 2006.

[6] B. Bustos, D. Keim, and T. Schreck. A pivot-based index structure for combination of feature vectors. In *Proc. 20th Annual ACM Symposium on Applied Computing, Multimedia and Visualization Track (SAC-MV'05)*, pages 1180–1184. ACM Press, 2005.

[7] B. Bustos and T. Skopal. Dynamic similarity search in multi-metric spaces. In *Proc. 8th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR'06)*, pages 137–146. ACM Press, 2006.

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

[9] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[10] P. Ciaccia and M. Patella. M2-tree: Processing complex multi-feature queries with just one index, 2000.

[11] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.*, 27(4):398–437, 2002.

[12] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[13] CoPhIR. Content-based photo image retrieval test-collection [http://cophir.isti.cnr.it].

[14] A. P. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient k-NN search on vertically decomposed data. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 322–333, New York, NY, USA, 2002. ACM.

[15] K. Figueroa, G. Navarro, and E. Chávez. Metric spaces library [http://www.sisap.org]. 2007.

[16] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[17] S. Hettich and S. D. Bay. The UCI KDD archive [http://kdd.ics.uci.edu]. 1999.

[18] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD '95: Proceedings of the 4th International Symposium on Advances in Spatial Databases*, pages 83–95, London, UK, 1995. Springer-Verlag.

[19] J. E. Jackson. *A User's Guide to Principal Components*. Wiley-Interscience, 1991.