CC52V
Bases de datos multimedia

Prof. Benjamin Bustos

Capítulo 5
Índices métricos

5.1 Conceptos básicos

- Para ciertas aplicaciones, es posible definir una función de distancia entre objetos multimedia, pero no pueden ser descritos en forma razonable como vectores
- También es posible que calcular la similitud directamente entre objetos sea más simple que transformarlos en vectores
- Índices métricos: función de distancia es una métrica

5.1 Conceptos básicos

- Definición de espacio métrico
  - Universo de objetos válidos: $\mathbb{X}$
  - Función de distancia: $\delta : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+$
  - $(\mathbb{X}, \delta)$ representa un espacio métrico ssi δ cumple con las siguientes propiedades:
    i. Positividad estricta $\forall x, y \in \mathbb{X},\ x \neq y \Rightarrow \delta(x, y) > 0$
    ii. Simetría $\forall x, y \in \mathbb{X},\ \delta(x, y) = \delta(y, x)$
    iii. Reflexividad $\forall x \in \mathbb{X},\ \delta(x, x) = 0$
    iv. Desigualdad triangular $\forall x, y, z \in \mathbb{X},\ \delta(x, z) \leq \delta(x, y) + \delta(y, z)$

3

5.1 Conceptos básicos

- Objetos de $\mathbb{X}$ son directamente comparados utilizando δ
- δ indica el grado de disimilitud entre dos objetos
- Ejemplo: strings + distancia de edición
  - *String*: secuencia de caracteres
  - *Distancia de edición*: mínimo # de inserciones, borrados o substituciones para transformar un string en otro

4

5.2 Búsqueda en espacios métricos

- Base de datos: $\mathbb{U} \subset \mathbb{X}$
- Algoritmo ingenuo: búsqueda secuencial
- En general, en espacios métricos se considera que la función de distancia es computacionalmente costosa de calcular
- Tiempo total para evaluar una consulta

$T =$ cómputos de distancia$\cdot$complejidad de $\delta +$ tiempo de CPU $+$ tiempo de E/S

5

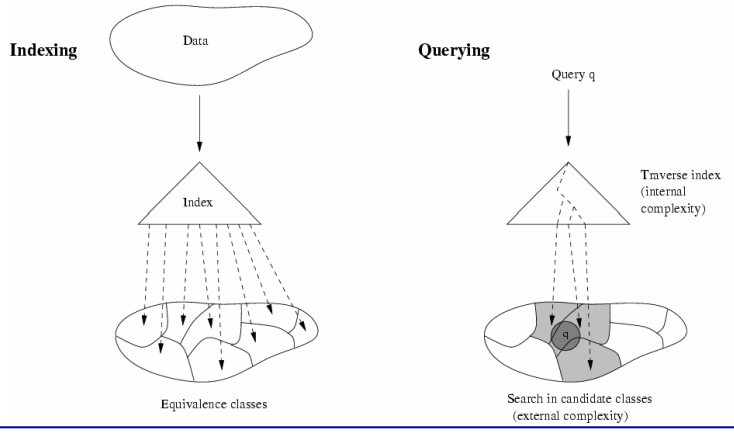
5.2 Búsqueda en espacios métricos

- Índices métricos tratan de minimizar el número de cómputos de distancia necesarios para responder búsquedas por similitud
- Otros componentes del costo pueden ser obviados (depende de la aplicación)
- Índice métrico particiona el espacio en clases de equivalencia
  - Durante la búsqueda se descartan clases
  - Clases no descartadas deben ser examinadas

6

5.2 Búsqueda en espacios métricos

- Indexamiento y búsqueda



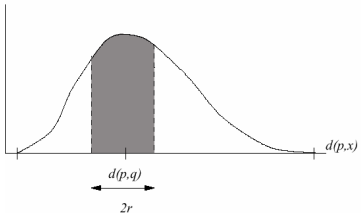
7

5.2 Búsqueda en espacios métricos

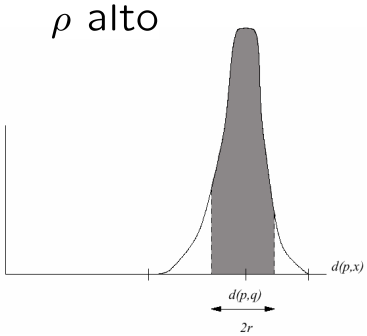
- Efectos en espacios métricos [CNB+01]

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

ρ bajo



ρ alto



8

5.3 Índices basados en pivotes

- Un pivote es un objeto distinguido de la BD
- Puede ser utilizado para descartar objetos durante una búsqueda
- Dada una consulta por rango *(q,r)* y un pivote *p*, se tiene que

$\delta(p,u) \leq \delta(p,q)+\delta(q,u)$ y $\delta(p,q) \leq \delta(p,u)+\delta(u,q)$

- Cota inferior para distancia entre *q* y *x*

$$|\delta(p,q) - \delta(p,u)| \leq \delta(q,u)$$

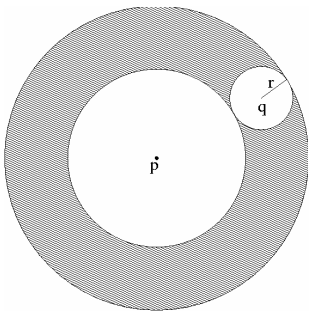
9

5.3 Índices basados en pivotes

- Criterio de exclusión: $|\delta(p,q) - \delta(p,u)| > r$

$r < |\delta(p,q)-\delta(p,u)| \leq \delta(q,u) \Rightarrow u$ no puede ser relevante

- Gráficamente



10

5.3 Índices basados en pivotes

- Algoritmo de búsqueda canónico basado en pivotes
  - Elegir *k* pivotes entre los objetos de la BD
  - Índice consiste en *kn* distancias precalculadas entre pivotes y objetos de la BD

	p_1	...	p_k
u_1	$\delta(p_1, u_1)$	...	$\delta(p_k, u_1)$
...	...	...	...
u_n	$\delta(p_1, u_n)$	...	$\delta(p_k, u_n)$

11

5.3 Índices basados en pivotes

- Consultas por rango
  - Calcular *k* distancias entre *q* y pivotes
  - Descartar objetos usando el criterio de exclusión (basta que con un pivote se descarte un objeto)

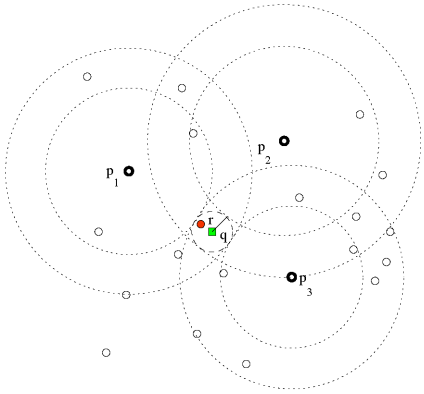
$$|\delta(p_i, q) - \delta(p_i, u)| > r \text{ para algún } p_i$$

  - Lista de objetos candidatos (los que no se pudieron descartar) deben ser comparados directamente con *q*

12

5.3 Índices basados en pivotes

- Ejemplo



$$\begin{bmatrix} \delta(p_1,u_1) & \delta(p_2,u_1) & \delta(p_3,u_1) \\ \delta(p_1,u_2) & \delta(p_2,u_2) & \delta(p_3,u_2) \\ \vdots & \vdots & \vdots \\ \delta(p_1,u_n) & \delta(p_2,u_n) & \delta(p_3,u_n) \end{bmatrix}$$

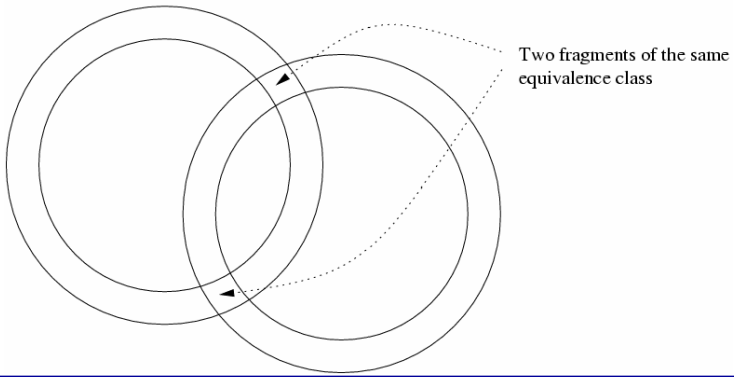
Criterio de exclusión:

$$|\delta(p_i,q) - \delta(p_i,u)| > r$$

13

5.3 Índices basados en pivotes

- Clase de equivalencia con respecto a un conjunto de pivotes



14

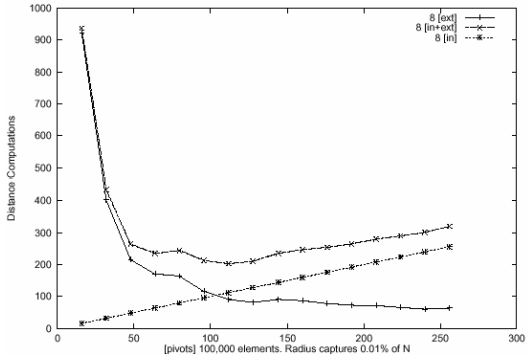
5.3 Índices basados en pivotes

- Complejidad de la búsqueda
  - Complejidad interna (*k*): cómputos de distancia entre pivotes y *q*
  - Complejidad externa (*m*): cómputos de distancia entre objetos no descartados y *q*
  - Complejidad total: *k*+*m*
  - Dado que la complejidad interna crece con el número de pivotes y la complejidad interna decrece con el número de pivotes: existe un número óptimo de pivotes a utilizar

15

5.3 Índices basados en pivotes

- Complejidad de la búsqueda vs. # pivotes



  - *k** muy alto, se utiliza toda la memoria disponible

16

5.3.1 Vantage Point Tree [Yia93]

- VPT es un árbol binario
- Algoritmo de construcción
  - Usar cualquier objeto *p* como raíz
  - Calcular la mediana de todas las distancias a *p*

$$M = \text{mediana}\{\delta(p,u),\ u \in \mathbb{U}\}$$

  - Subárbol izquierdo: $\delta(p,u) \leq M$
  - Subárbol derecho: $\delta(p,u) > M$
  - Proceder recursivamente hasta que sólo haya 1 objeto en subárbol

17

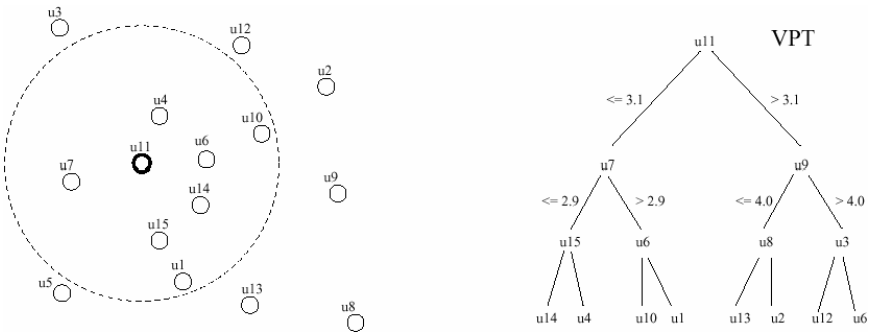
5.3.1 Vantage Point Tree [Yia93]

- Heurística propuesta para elegir *p*: escoger objetos “lejos” del resto
- Algoritmo de búsqueda, consultas por rango
  - Calcular $d = \delta(q,p)$
  - Si $d \leq r$ añadir *p* al resultado
  - Si $d - r \leq M$ buscar recursivamente en subárbol izquierdo
  - Si $d + r > M$ buscar recursivamente en subárbol derecho

18

5.3.1 Vantage Point Tree [Yia93]

- Ejemplo de un VPT



19

5.3.2 Técnicas de selección de pivotes

- Selección de los pivotes afecta el rendimiento del algoritmo de búsqueda
  - 2 pivotes muy cercanos no aumentan mucho el poder discriminativo del índice
- En general, pivotes se eligen aleatoriamente
- Existen técnicas de selección de pivotes [BNC03]

20

5.3.2 Técnicas de selección de pivotes

- Espacio de pivotes
  - Espacio *k*-dimensional (cada coordenada es la distancia entre *i*-ésimo pivote y el objeto)

$$[x] = (\delta(x, p_1), \ldots, \delta(x, p_k)) \in \mathbb{R}^k$$

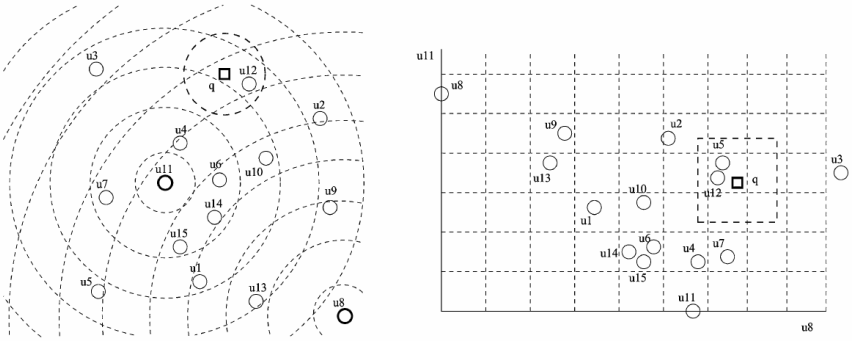
  - Distancia utilizada: distancia del máximo

$$\Delta_{p_1,\ldots,p_k}([x],[y]) = \max_{i=1}^{k}(|\delta(x,p_i) - \delta(y,p_i)|)$$

21

5.3.2 Técnicas de selección de pivotes

- Mapeando objetos al espacio de pivotes



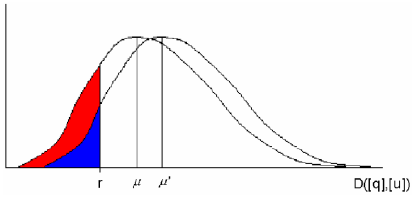
22

5.3.2 Técnicas de selección de pivotes

- Condición de exclusión en espacio de pivotes

$$|\delta(p_i, u) - \delta(p_i, q)| > r \Rightarrow \Delta_{p_1,\ldots,p_k}([q],[u]) > r$$

Distribución de Δ



Criterio de eficiencia para comparar dos conjuntos de pivotes:

$$\mu'_{\Delta_{p'_1,\ldots,p'_k}} > \mu_{\Delta_{p_1,\ldots,p_k}}$$

5.3.2 Técnicas de selección de pivotes

- Estimación de μ_Δ
  - Elegir un conjunto de *A* pares de puntos
  - Calcular Δ_i para el *i*-ésimo par
  - Media de distancias en Δ se estima como

$$\mu_\Delta = \frac{1}{A}\sum_{i=1}^{A} \Delta_i$$

  - Costo para estimar μ_Δ: $2kA$ cálculos de δ

5.3.2 Técnicas de selección de pivotes

- Selección de *N* grupos aleatorios
  - Elegir *N* grupos de *k* pivotes aleatorios
  - Estimar para cada grupo
  - Escoger el grupo que maximiza μ_Δ
  - Costo de optimización: $2kAN$ cálculos de δ

25

5.3.2 Técnicas de selección de pivotes

- Selección incremental
  - Elegir un pivote de una muestra de *N* objetos aleatorios, que maximice la media de distancias de Δ
  - Elegir un segundo pivote de una muestra de *N* objetos aleatorios, tal que el conjunto maximice la media de distancias de Δ
  - Repetir hasta que se elijan *k* pivotes
  - Costo de optimización: $2kAN$ cálculos de δ
- Ventajas
  - Permite agregar pivotes sin tener que rehacer todo el trabajo de optimización

26

5.3.2 Técnicas de selección de pivotes

- Selección de óptimo local
  - Elegir *k* pivotes aleatorios
  - Calcular matriz *M*

$$M_{A\times k} = \Delta_{p_j}([a_r],[a'_r]),\ 1 \le r \le A,\ 1 \le j \le k$$

  - Se sigue que

$$\Delta([a_r],[a'_r]) = \max_{j=1}^{k} M[r,j]$$

27

5.3.2 Técnicas de selección de pivotes

- Selección de óptimo local
  - Contribución del pivote p_j: suma sobre las *A* filas de cuanto ayuda p_j a incrementar el valor de Δ
    - r_{max}: índice del pivote con el valor máximo en esa fila
    - r_{max2}: índice del pivote con el segundo valor máximo

$$\text{contribución} = \begin{cases} M[r,r_{max}] - M[r,r_{max2}] & \text{si } j = r_{max} \\ 0 & \text{si no} \end{cases}$$

28

5.3.2 Técnicas de selección de pivotes

- Selección de óptimo local
  - Pivote con contribución mínima es la *víctima*
  - Se reemplaza, si es posible, por un pivote elegido de una muestra de *X* objetos
  - Proceso se repite *N'* veces
  - Costo de optimización:
    - Construcción de M: $2Ak$ cálculos de δ
    - Calcular víctima: 0
    - Reemplazar víctima: $2AX$ cálculos de δ
    - Costo total: $2A(k+N'X)$ cálculos de δ

29

5.3.2 Técnicas de selección de pivotes

- Selección de óptimo local
  - Si $kN=k+N'X$ (i.e., $N'X=k(N-1)$), costo = $2AkN$ cálculos de d
  - Notar que se pueden intercambiar los valores de *N'* y *X* manteniendo el costo total de optimización
    - Óptimo local A: $N'=k$ y $X=N-1$
    - Óptimo local B: $N'=N-1$ y $X=k$

30

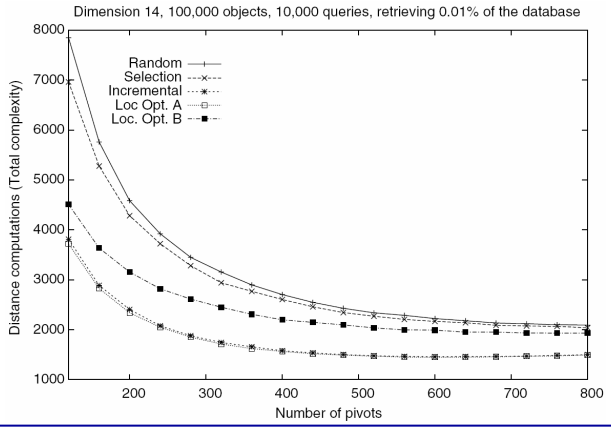
5.3.2 Técnicas de selección de pivotes

- Experimentalmente se encontró que
  - Es mejor tener buena estimación de (i.e., *A* debe ser grande)
  - Muestras pequeñas de objetos son suficientes para obtener buenos resultados (i.e., *N* puede ser pequeño)
- Pruebas experimentales
  - Con datos sintéticos
  - Métodos de selección encuentran conjuntos de "buenos pivotes"

31

5.3.2 Técnicas de selección de pivotes

- Resultados



32

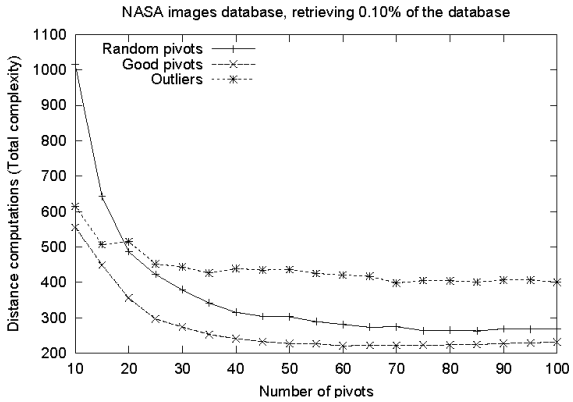
5.3.2 Técnicas de selección de pivotes

- Características de los “buenos pivotes”
  - Son objetos alejados del resto de la BD
  - Son objetos alejados entre sí
- Objetos con estas características se denominan *outliers*
  - Método alternativo de selección: elegir outliers
  - Producen mejores resultados que “buenos pivotes” en espacios sintéticos (distribución uniforme), funcionan mal con datos reales

33

5.3.2 Técnicas de selección de pivotes

- Resultados experimentales



Observationes:

• “Buenos pivotes” elegidos considerablemente mejores que pivotes aleatorios
• “Buenos pivotes” alcanzan la eficiencia óptima de los pivotes aleatorios con muchos menos pivotes
• Outliers usados como pivotes obtienen peores resultados que pivotes aleatorios

34

5.3.2 Técnicas de selección de pivotes

- Conclusiones
  - "Buenos pivotes" son outliers
  - No todo outlier es un buen pivote
  - Existen conjuntos de "buenos pivotes"
    - Se necesita una buena estimación de la media de distancias de Δ
    - No es necesario tener una muestra muy grande para escoger los pivotes
  - Tema de investigación en la actualidad

35

5.3.3 Otros índices basados en pivotes

- Árboles métricos basados en pivotes
  - Burkhard-Keller Tree [BK73], Fixed Queries Tree [BCM+94], Fixed-Height Queries Tree [BCM+94], Multi Vantage Point Tree [BO97]
- Representación del árbol en arreglos
  - Spaghettis [CMB99], Fixed Queries Array [CMN01]
- Otras estructuras
  - Approximating and Eliminating Search Algorithm (AESA) [Vid86], Linear AESA [MOV94]

36

5.4 Índices basados en particiones compactas

- Dividen el espacio en zonas o particiones lo más compactas posible
- Cada zona tiene un punto representativo *c* denominado “centro”
- Cada zona puede ser particionada recursivamente en más zonas, forando una jerarquía de búsqueda
- Hay dos criterios generales: partición de Voronoi y radio cobertor

37

5.4 Índices basados en particiones compactas

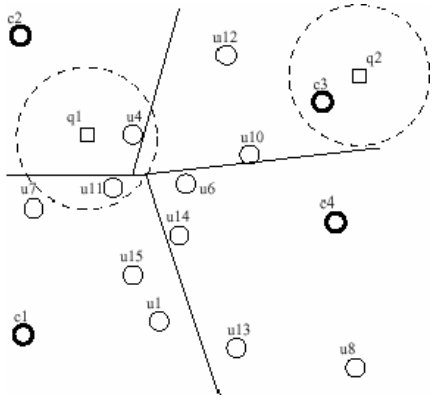
- Partición de Voronoi
  - Se eligen m centros (objetos de la BD)
  - El resto de los objetos se asigna a su centro más cercano (partición de Voronoi)
  - Dada una consulta por rango *(q,r)*, se calculan las distancias entre *q* y los *m* centros
  - Sea *c* el centro más cercano a *q*. Se pueden descartar las zonas *i* cuyo centro cumpla con:

$$\delta(q, c_i) > \delta(q, c) + 2r$$

38

5.4 Índices basados en particiones compactas

- Ejemplo de partición de Voronoi



39

5.4 Índices basados en particiones compactas

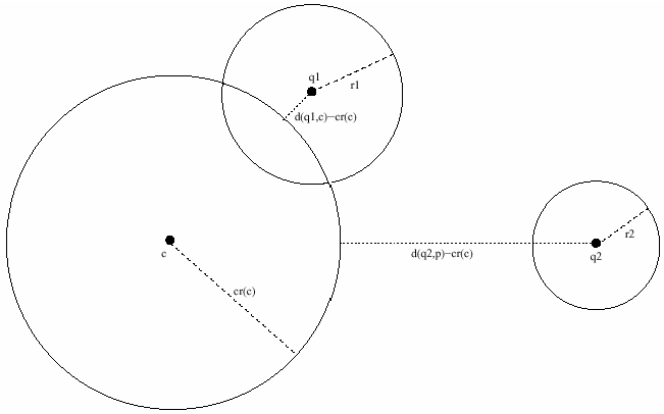
- Criterio de radio cobertor
  - El radio cobertor *cr(c)* es la distancia máxima entre un centro c y algún objeto en su zona
  - Dada la consulta *(q,r)*, la zona de centro *c* no puede tener intersección con la bola de consulta si

$$\delta(q,c) - cr(c) > r$$

40

5.4 Índices basados en particiones compactas

- Ejemplo de radio cobertor



41

5.4 Índices basados en particiones compactas

- Índices basados en particiones compactas
  - Partición de Voronoi
    - Generalized-Hyperplane Tree [Uhl91]
  - Radio cobertor
    - Bisector Tree (BST) [KM83], Voronoi Tree [DN87], Monotonous BST [NVZ92], M-tree [CPZ97], List of Clusters [CN05]
  - Ambos criterios
    - Geometric Near-neighbor Access Tree [Bri95], Spatial Approximation Tree [Nav02]

42

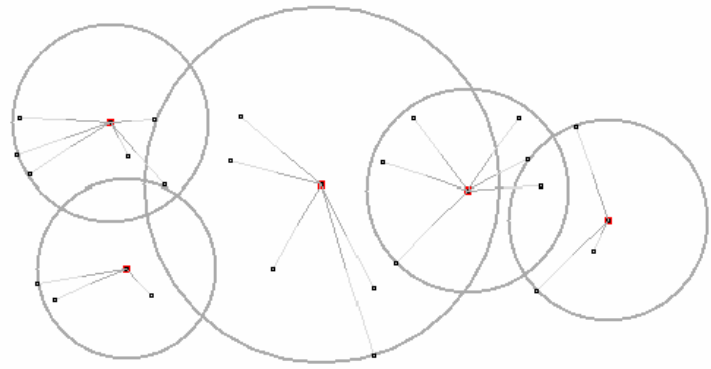
5.4.1 M-tree [CPZ97]

- M-tree [CPZ97]
  - Índice basado en el criterio de radio cobertor
  - Árbol balanceado
  - Objetivos
    - Índice dinámico (permite inserciones, borrados, updates)
    - Buen desempeño de E/S
    - Pocos cómputos de distancia por consulta

43

5.4.1 M-tree [CPZ97]

- Ejemplo de M-tree



44

5.4.1 M-tree [CPZ97]

- Estructura del M-tree
  - Hojas (nodos externos)
    - Almacenan los objetos indexados
  - Nodos internos
    - Almacenan los *routing objects*
    - Por cada routing object O_r
      - *ptr(T(O_r))* apunta a la raíz del subárbol
      - *T(O_r)* árbol cubridor de O_r
      - *r(O_r)* radio cobertor de O_r
      - *P(O_r)* padre de O_r (indefinido para la raíz)

5.4.1 M-tree [CPZ97]

- Nodos del M-tree

Nodos internos:

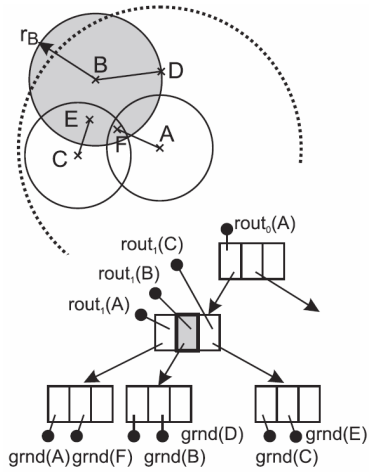
O_r	(feature value of the) routing object
$ptr(T(O_r))$	pointer to the root of $T(O_r)$
$r(O_r)$	covering radius of O_r
$d(O_r, P(O_r))$	distance of O_r from its parent

Nodos externos:

O_j	(feature value of the) DB object
$oid(O_j)$	object identifier
$d(O_j, P(O_j))$	distance of O_j from its parent

5.4.1 M-tree [CPZ97]

- Ejemplo



47

5.4.1 M-tree [CPZ97]

- Búsquedas: consulta por rango

```
RS(N:node, Q:query_object, r(Q):search_radius)
{ let O_p be the parent object of node N;
  if N is not a leaf
  then { ∀ O_r in N do:
         if |d(O_p,Q) − d(O_r,O_p)| ≤ r(Q) + r(O_r)
         then { Compute d(O_r,Q);
                if d(O_r,Q) ≤ r(Q) + r(O_r)
                then RS(*ptr(T(O_r)),Q,r(Q)); }}
  else { ∀ O_j in N do:
         if |d(O_p,Q) − d(O_j,O_p)| ≤ r(Q)
         then { Compute d(O_j,Q);
                if d(O_j,Q) ≤ r(Q)
                then add oid(O_j) to the result;}}}
```

48

5.4.1 M-tree [CPZ97]

- Consulta por *k* vecinos más cercanos
  - Hjaltason y Samet [HS95]
  - Dos colas de prioridad
    - *k* mejores candidatos hasta el momento
    - *Active Page List* (APL)
      - Nodos cuyo padre ya fue visitado, pero ellos no han sido visitados aún
      - Ordenados por cota inferior de la distancia al objeto de consulta

5.4.1 M-tree [CPZ97]

- Consulta por *k* vecinos más cercanos
  - Inicialmente: lista de candidatos vacío, APL sólo contiene la raíz del árbol
  - Se saca un nodo de la APL
    - Medir distancia al centro y agregarlo a la lista de candidatos si es necesario (menos de *k* candidatos o distancia al centro menor que al *k*-ésimo candidato)
    - Insertar todos los hijos a la APL
  - Terminar si APL vacía o mínima cota inferior de un nodo mayor que distancia a *k*-ésimo candidato, sino iterar

5.4.1 M-tree [CPZ97]

- Algoritmo de inserción

```
Insert(N:node, entry(O_n):M-tree_entry)
{ let 𝒩 be the set of entries in node N;
  if N is not a leaf then
  { let 𝒩_in = entries such that d(O_r, O_n) ≤ r(O_r);
    if 𝒩_in ≠ ∅
    then let entry(O*_r) ∈ 𝒩_in : d(O*_r, O_n) is minimum;
    else { let entry(O*_r) ∈ 𝒩:
                    d(O*_r, O_n) − r(O*_r) is minimum;
           let r(O*_r) = d(O*_r, O_n); }
    Insert(*ptr(T(O*_r)), entry(O_n)); }
  else /* N is a leaf */
  { if N is not full
    then store entry(O_n) in N
    else Split(N, entry(O_n)); }}
```


5.4.1 M-tree [CPZ97]

- Split de nodos

```
Split(N:node; E:M-tree_entry)
{ let 𝒩 = entries of node N ∪ {E};
  ifN is not the root then
    let O_p be the parent of N, stored in N_p node;
  Allocate a new node N';
  Promote(𝒩, O_p1, O_p2);
  Partition(𝒩, O_p1, O_p2, 𝒩_1, 𝒩_2);
  Store 𝒩_1's entries in N and 𝒩_2's entries in N';
  if N is the current root
  then { Allocate a new root node, N_p;
         Store entry(O_p1) and entry(O_p2) in N_p; }
  else { Replace entry(O_p) with entry(O_p1) in N_p;
         if node N_p is full
         then Split(N_p, entry(O_p2))
         else store entry(O_p2) in N_p; }}
```

Promote escoge dos routing objects para ser insertados en el Nodo padre

Partition divide las entradas en el nodo que hizo overflow en dos subconjuntos disjuntos

5.4.1 M-tree [CPZ97]

- Nuevos radios cobertores después de un split
  - Si es una hoja

$$r(O_{p_1}) = max\{d(O_j, O_{p_1}) | O_j \in \mathcal{N}_1\}$$

  - Si es un nodo interno

$$r(O_{p_1}) = \max\{d(O_r, O_{p_1}) + r(O_r) | O_r \in \mathcal{N}_1\}$$

5.4.1 M-tree [CPZ97]

- Métodos *Promote* y *Partition*
  - Definen una política de split
  - Objetivos
    - Minimizar “volumen”
    - Minimizar “solapamiento”
  - *Promote*
    - Elegir el padre del nodo como uno de los routing objects (*confirmed split policy*)
    - Aleatoriamente
    - Minimizar suma de radios cobertores resultantes

5.4.1 M-tree [CPZ97]

- Métodos *Promote* y *Partition*
  - *Partition*
    - *Generlized Hyperplane*: asignar cada objeto al routing object más cercano
    - Balanceado: repartir la mitad de los objetos a cada routing objects (criterio: distancia a los routing objects)
  - Generalized Hyperplane funciona mejor en la práctica

5.4.1 M-tree [CPZ97]

- Conclusiones
  - Índice métrico dinámico
  - Performance depende de la política de split utilizada
  - Si se conocen los objetos de la BD de antemano: bulk loading
  - Se sigue realizando investigación para extender capacidades de este índice

5.4.2 List of Clusters [CN05]

- Basado en el criterio de radio cobertor
- Árbol binario desbalanceado
  - Hijo izquierdo siempre contiene datos
  - Hijo derecho contiene árbol con resto de los objetos
  - Último nodo sólo tiene hijo izquierdo

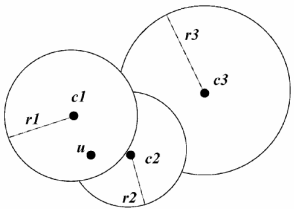
5.4.2 List of Clusters [CN05]

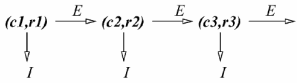
- Estructura
  - Se elige un objeto *c* como centro y un radio cobertor *r*
  - Se insertan en la región de *c* todos los objetos a distancia menor o igual que *r* de *c*
    - Objetos en la región: forman nodo izquierdo
    - Resto del espacio: nodo derecho
  - Se elige otro centro y se repite el proceso hasta que no queden más objetos
  - Orden de las regiones en la lista es importante
    - Si hay traslape, el objeto pertenece al primer nodo de la lista que cubra la posición donde está

5.4.2 List of Clusters [CN05]

- Algoritmo de construcción

```
Build (𝕌)
   if 𝕌 = ∅ then return an empty list
   Select c ∈ 𝕌
   Select a radius r_c
   I  ⟵  {u ∈ 𝕌 − {c},  d(c,u) ≤ r_c}
   E  ⟵  𝕌 − I
   return (c, r_c, I):Build(E)
```



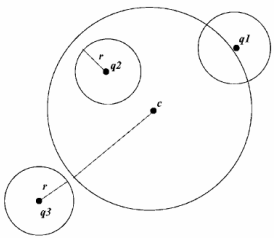


59

5.4.2 List of Clusters [CN05]

- Algoritmo de búsqueda por rango

```
Search (L, q, r)
   if L is empty then return
   Let L = (c, r_c, I) : E
   Compute d(c, q)
   if d(c, q) ≤ r then add c to the list of results
   if d(c, q) ≤ r_c + r then search I exhaustively
   if d(c, q) > r_c − r then Search (E, q, r)
```



60

5.4.2 List of Clusters [CN05]

- Selección del centro
  - Existen distintas opciones
  - Mejores resultados empíricos: objeto que maximice la suma de distancias a los centros previos
- Selección del radio
  - Radio fijo o número de objetos por región fijo
  - Mejores resultados: número de objetos fijo

61

5.4.2 List of Clusters [CN05]

- Conclusiones
  - Índice simple, desbalanceado
  - En la práctica funciona bien en espacios métricos con dimensión intrínseca alta

62

5.5 Referencias

- [BK73] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230—236, 1973
- [BCM+94] R. Baeza-Yates, W. Cunto,U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Annual Symposium on Combinatorial Pattern Matching*, LNCS 807, pages 198—212, 1994
- [BNC03] B. Bustos, G. Navarro, and E. Chavez. Pivot selection techniques for proximity queries in metric spaces. *Pattern Recognittion Letters 24(14):2357-2366*, 2003
- [BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM International Conference on Management of Data (SIGMOD'97)*, pages 357—368. ACM Press, 1997
- [Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Larga Databases (VLDB'95)*, pages 574-584, 1995
- [CMB99] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *Proc. 6th Intl. Symp. on String Processing and Information Retrieval (SPIRE'99)*, pages 38—46. IEEE CS Press, 1999
- [CMN01] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications*, 14(2):113—135, 2001
- [CN05] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. Pattern Recognition Letters, 26(9):1363–1376, 2005.
- [CNB+01] E. Chávez, G. Navarro, J. Marroquín, and R. Baeza-Yates. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273—321, 2001

63

5.5 Referencias

- [CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23rd Conf. on Very Large Databases (VLDB'97)*, pages 426-435, 1997
- [DN87] F. Dehne and H. Noltemeier. Voronoi trees and clustering problems. *Information Systems 12(2):171-175*, 1987
- [KM83] I. Kalantari and G. McDonald. A data structure and an algorithm foe the nearest point problem. *IEEE Transactions on Software Engineering 9(5):631-634*, 1983
- [MOV94] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognnition Letters*, 15:9—17, 1994
- [Nav02] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ) 11(1):28-46*, 2002
- [NVZ92] H. Noltemeier, K. Verbarg, and C. Zirkelbach, C. Monotonous Bisector Trees – a tool for efficient partitioning of complex schemes of geometric objects. In Data Structures and Efficient Algorithms, LNCS 594, pages 186–203. Springer, 1992.
- [Uhl91] J. Uhlmann. Satisfaying general proximity/similarity queries with metric trees. *Information Processing Letters, 40:175-179*, 1991
- [Vid86] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognittion Letters*, 4:145—157, 1986
- [Yia93] P. Yianlos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc.4th ACM/SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311—321, 1993

64