
Kapitel 2

Prinzipien der Anfragebearbeitung in STMM-DBS

Skript zur Vorlesung: Spatial, Temporal, and Multimedia Databases
Sommersemester 2008, LMU München

© 2006 Prof. Dr. Hans-Peter Kriegel, Dr. Peer Kröger, Dr. Peter Kunath, Dr. Matthias Renz, Arthur Zimek

Übersicht

- 2.1 Feature-Räume
- 2.2 Algorithmische Paradigmen zur Anfragebearbeitung
- 2.3 Bereichsanfragen
- 2.4 Nächste Nachbarn Anfragen
- 2.5 Reverse nächste Nachbarn Anfragen
- 2.6 Bewertung von Methoden zur Ähnlichkeitssuche

2.1 Feature-Räume

2.1.1 Das Prinzip der feature-basierten Ähnlichkeit

– Grundidee der Feature-Transformation

- Erwünscht: effiziente Ähnlichkeitssuche in Datenbanken
- Meist ist Effizienz ohne Einsatz von Indexstrukturen nicht zu verwirklichen
- Entwickle nicht für jedes der einzelnen Anwendungsgebiete/ Ähnlichkeitsmaße spezielle Indexstrukturen
- Versuche mit wenigen Arten von Indexstrukturen möglichst viele Anwendungsgebiete abzudecken
- Indexstrukturen für:
 - Multidimensionale Vektoren
 - Allgemein metrische Daten (beliebige Objekte, auf denen eine metrische Distanzfunktion definiert ist)

- Extrahiere charakteristische (numerische) Eigenschaften („Features“) aus den Objekten



- Wichtigste Eigenschaft der Feature-Transformation:
 - Ähnlichkeit der Objekte entspricht geringem Abstand der Feature-Vektoren
 - => Ähnlichkeitsanfragen im Objektraum entsprechen Nachbarschaftsanfragen im Feature-Raum
 - => Unterstützung durch geeignete multidimensionale Indexstrukturen

– Erweiterungen

- Oft reichen die Mächtigkeit von Vektoren für die Modellierung nicht aus (vergleiche z.B. Relationales und OO-Modell)
- Transformation in andere Räume, die die Definition einer Distanzfunktion mit Metrik-Eigenschaften erlauben
 - Graphen
 - Punktmengen
 - ...

– Fazit:

- Das Prinzip der Feature-Transformation ist ein mächtiges Werkzeug zur Modellierung der Ähnlichkeit von komplexen STMM-Objekten
- Herausforderung: Finden geeigneter Feature-Transformationen

2.1.2 Feature-Räume und Distanzen

– Allgemeiner Feature-Raum

Ein Feature-Raum ist ein Tupel $\Phi = (\text{Dom}, \text{dist})$ mit

- Dom ist ein Wertebereich (Domain)
- dist ist eine Distanzfunktion, d.h. es gilt
 - Reflexivität $\forall x, y \in \text{Dom}: \text{dist}(x, y) = 0 \Leftrightarrow x = y$
 - Positiv-Definitheit $\forall x, y \in \text{Dom}, x \neq y: \text{dist}(x, y) > 0$
 - Symmetrie $\forall x, y \in \text{Dom}: \text{dist}(x, y) = \text{dist}(y, x)$

– Metrischer Raum

Ein metrischer Raum ist ein Tupel $\Phi_M = (\text{Dom}, \text{dist})$ mit

- $(\text{Dom}, \text{dist})$ ist ein allgemeiner Feature-Raum
- dist erfüllt zusätzlich die
 - Dreiecksungleichung $\forall x, y, z \in \text{Dom}: \text{dist}(x, z) \leq \text{dist}(x, y) + \text{dist}(y, z)$

– Vektorraum

Ein (euklidischer) Vektorraum der Dimension d (d -dimensionaler Vektorraum) ist ein Tupel $\Phi_E = (\text{Dom}, \text{dist})$ mit

- $(\text{Dom}, \text{dist})$ ist ein metrischer Raum
- $\text{Dom} = \mathbb{R}^d$

– Feature-Transformation

Eine Feature-Transformation ist eine Abbildung

$$T: \text{OBJ} \rightarrow (\text{Dom}, \text{dist})$$

die jedem Objekt $o \in \text{OBJ}$ aus dem Objektraum ein Objekt aus dem Wertebereich Dom zuordnet.

Die Distanz im Objektraum wird durch die Distanz im Feature-Raum repräsentiert, d.h.

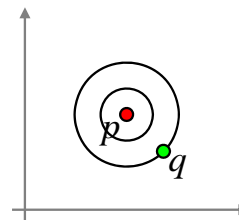
$$\forall x, y \in \text{OBJ}: \text{dist}_{\text{OBJ}}(x, y) \equiv \text{dist}_{\text{Dom}}(T(x), T(y))$$

– Distanzmaße in Vektorräumen

- Euklidische Norm (L_2):

$$\text{dist} = ((p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots)^{1/2}$$

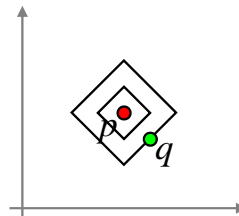
Natürliches Distanzmaß



- Manhattan-Norm (L_1):

$$\text{dist} = |p_1 - q_1| + |p_2 - q_2| + \dots$$

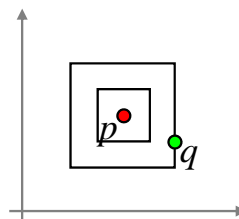
Die Unähnlichkeiten der einzelnen Merkmale werden direkt addiert



- Maximums-Norm (L_∞):

$$\text{dist} = \max\{|p_1 - q_1|, |p_2 - q_2|, \dots\}$$

Die Unähnlichkeit des am wenigsten ähnlichen Merkmals zählt



- Verallgemeinerung L_p -Abstand: $\text{dist}_p = (|p_1 - q_1|^p + |p_2 - q_2|^p + \dots)^{1/p}$

- Gewichtete Euklidische Norm:

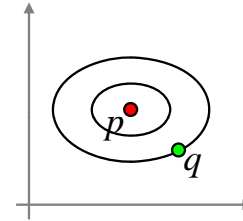
$$\text{dist} = (w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + \dots)^{1/2}$$

Häufig sind die Wertebereiche der Merkmale deutlich unterschiedlich

Beispiel: Merkmal $M_1 \in [0.01 \dots 0.05]$

Merkmal $M_2 \in [3.07 \dots 22.2]$

Damit M_1 überhaupt berücksichtigt wird muss es höher gewichtet werden



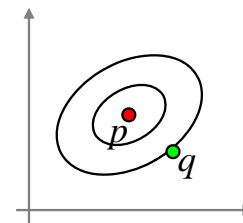
- Quadratische Form:

$$\text{dist} = ((p - q) \mathbf{M} (p - q)^T)^{1/2}$$

Bisherige Abstandsmasse gewichten

Merkmale nur getrennt

Besonders bei Farbhistogrammen müssen verschiedene Merkmale gemeinsam gewichtet werden



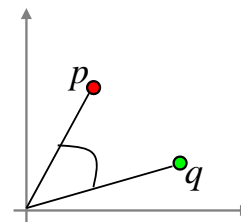
- Cosinus-Distanz

$$\text{dist} = \cos(\text{winkel}(p, q))$$

Berechnet den cosinus des Winkels

Meist für sehr hochdimensionalen

Featurevektoren (z.B. Texten)



– Bemerkungen

- Jeder Vektorraum ist ein metrischer Raum, jeder metrische Raum ein allgemeiner Feature-Raum
- Sprechweise meist: „Feature-Raum“ statt (euklidischer) Vektorraum
- Transformation komplexer Objekte meist immer in metrische Räume wegen der Dreiecksungleichung (Performanz!!!)

2.2 Algorithmische Paradigmen zur Anfragebearbeitung

2.2.1 Übersicht

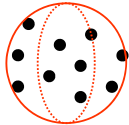
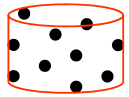
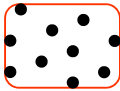
- Typen von Ähnlichkeitsanfragen
 - Bereichsanfragen
 - Nächste Nachbarn Anfragen
 - Reverse Nächste Nachbarn
- Algorithmische Paradigmen
 - Naive (sequentielle) Suche
 - Für alle n Objekte der Datenbank wird das Anfrage-Prädikat (d.h. meist eine Distanzberechnung zum Anfrageobjekt) ausgewertet
 - Kosten: $O(n \cdot \text{Kosten für das Anfrage-Prädikat})$
 - Indexbasierte Suche
 - Mehrstufige Anfragebearbeitung

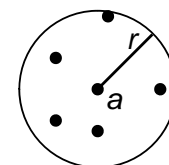
2.2.2 Indexstrukturen

- Prinzip [Böhm, Berchtold, Keim. ACM Computing Surveys, 2001]
 - Organisiere Objekte der Datenbank so, dass während einer Ähnlichkeitsanfrage nur auf „relevante“ Objekte zugegriffen werden muss
 - Baumartige Organisation; jedem Knoten des Baumes ist zugeordnet
 - Seite des Hintergrundspeichers
 - Region des Datenraums
 - Typen von Knoten (= Seiten)
 - Blattknoten sind Datenseiten, speichern Objekte
 - Innere Knoten sind Directoryseiten, speichern Directory-Einträge
 - » Verweis zur Kindseite (Adresse auf dem Hintergrundspeicher)
 - » Beschreibung der Region der Kindseite

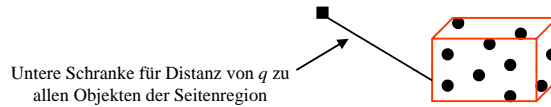
- Physische vs. logische Seiten
 - ursprünglich: eine physische Seite des Hintergrundspeichers wird verwendet
 - » kleinste Informationseinheit, die zwischen Plattenspeicher und RAM übertragen werden kann
 - » ABER: physische Seiten meist zu klein
 - daher: logische Seiten fassen aufeinanderfolgende physische Seiten zusammen
 - Meistens: einheitliche Seitengröße für alle Seiten eines Indexes (um komplizierte Freispeicherverwaltung zu vermeiden)
 - Kapazität der Directory-/Datenseiten (max. Anzahl der Einträge)

$$c_{\text{Data}} = \left\lfloor \frac{\text{Seitengröße} - \text{Verwaltungsoverhead}}{\text{Größe eines Datensatzes}} \right\rfloor \quad c_{\text{Directory}} = \left\lfloor \frac{\text{Seitengröße} - \text{Verwaltungsoverhead}}{\text{Größe eines Directoryeintrags}} \right\rfloor$$
 - Meist keine 100% Füllung der Seiten (Platz für neue Datensätze) aber minimale Füllung (z.B. 40%) wegen Speicherauslastung
 - Speicherauslastung ist die durchschnittliche Anzahl besetzter Einträge
- Jedes Objekt wird in genau einer Datenseite gespeichert
- Regionen gewährleisten, dass ähnliche Objekte möglichst auf den selben Datenseiten (oder Teilbäumen) gespeichert werden

- Gestalt der Seitenregionen
 - Vektordaten:
 - » Achsenparallel Rechtecke, die minimal um die Punktmenge gespannt werden (MUR=minimal umgebendes Rechteck/ MBR=minimum bounding rectangle) (R-Tree, R*-Tree, X-Tree, ...)
 - » Kugeln (SS-Tree) 
 - » Zylinder (TV-Tree) 
 - » Kombinationskörper (Kugel+MBR, SR-Tree) 
 - Allgemein metrische Daten
 - » Kein „Raum“ im Euklidischen Sinne, keine Geometrie
 - » Ankerobjekt a (anchor object) + Hüllradius r (covering radius)
Für alle Objekte o der Seitenregion gilt: $\text{dist}(o, a) \leq r$



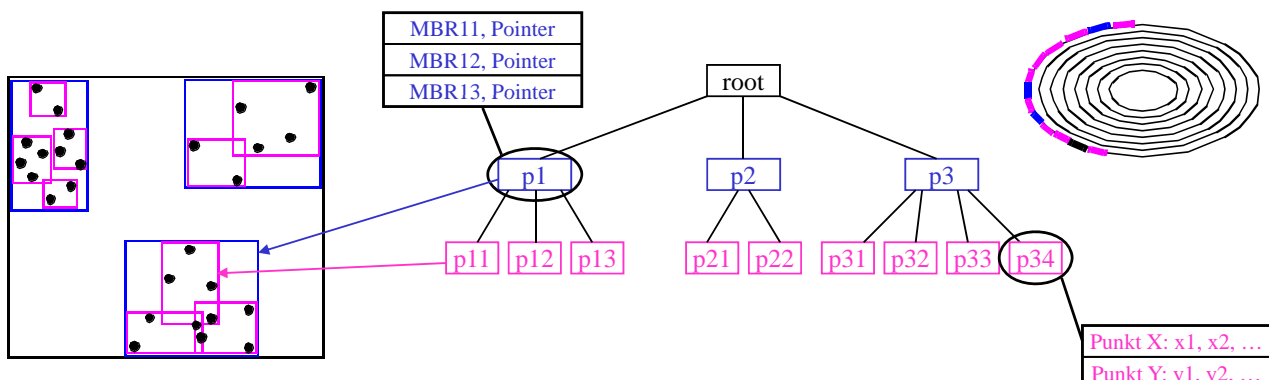
- Seitenregion umfasst immer alle Objekte der Datenseite bzw. des Teilbaums der Directoryseite
 - Konservative Approximation, lower-bounding property: die Distanz von einem Objekt zu einer Seitenregion kann immer mit einer unteren Schranke abgeschätzt werden



- Damit kann man Vollständigkeit der Anfrageergebnisse gewährleisten
- Bäume sind meist balanciert (alle Blätter auf selbem Level)
- Indexstrukturen sind dynamisch, d.h. Insert/Delete sind effizient
 - Tiefensuche nach entsprechender Datenseite (auf einen Pfad beschränkt)
 - Einfügen/Löschen des Objektes
 - Überlauf/Unterlauf-Behandlung durch Split/Merge
 - » Verschiedene Kriterien (minimale Überlappung, toter Raum, ...)
 - » Verschiedene Algorithmen (linear, quadratisch, ...)
 - » Entsprechendes Einfügen/Löschen im Elternknoten (rekursiv, notfalls bis Wurzel)

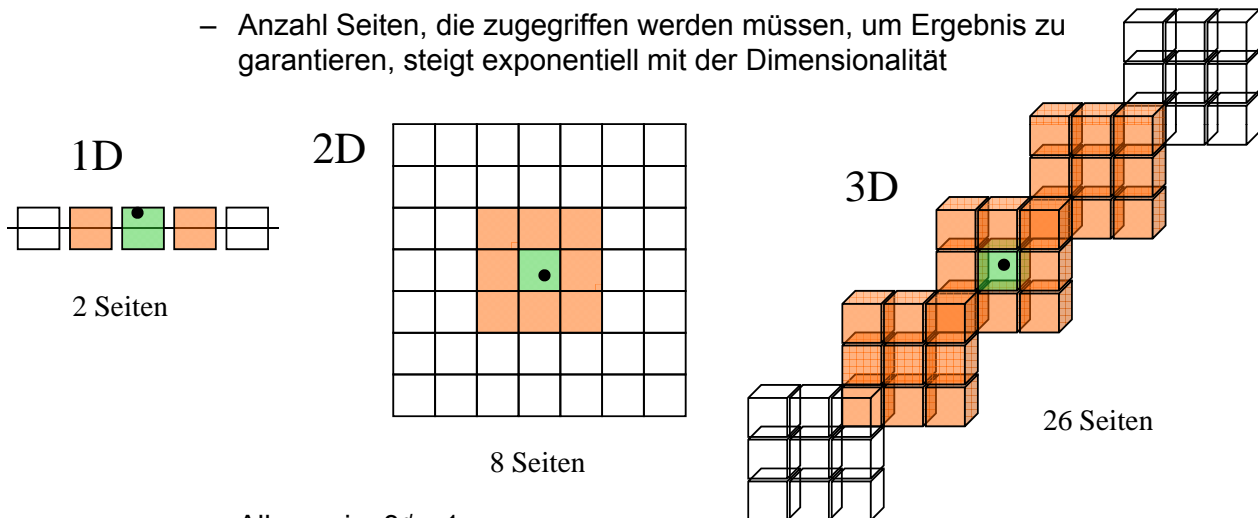
– Beispiel: R*-tree

- Entworfen für 2D Rechtecksdaten (Punkte sind spezielle Rechtecke), oft verwendet für hochdimensionale Vektordaten
- Design
 - Balancierter Index mit einheitlich großen Daten-/Directoryseiten
 - Seitenregionen: MBRs
 - Insert-Strategie: Overlap, Volumen
 - Überlaufbehandlung: Forced Re-Insert-Konzept
 - Splitkriterien: Umfang/Oberfläche, Überlappungsvolumen, (toter Raum)



– Curse of Dimensionality (Vektordaten)

- Anfrageleistung von Indexstrukturen verschlechtern sich mit zunehmender Dimension
 - Anzahl Seiten, die zugegriffen werden müssen, um Ergebnis zu garantieren, steigt exponentiell mit der Dimensionalität

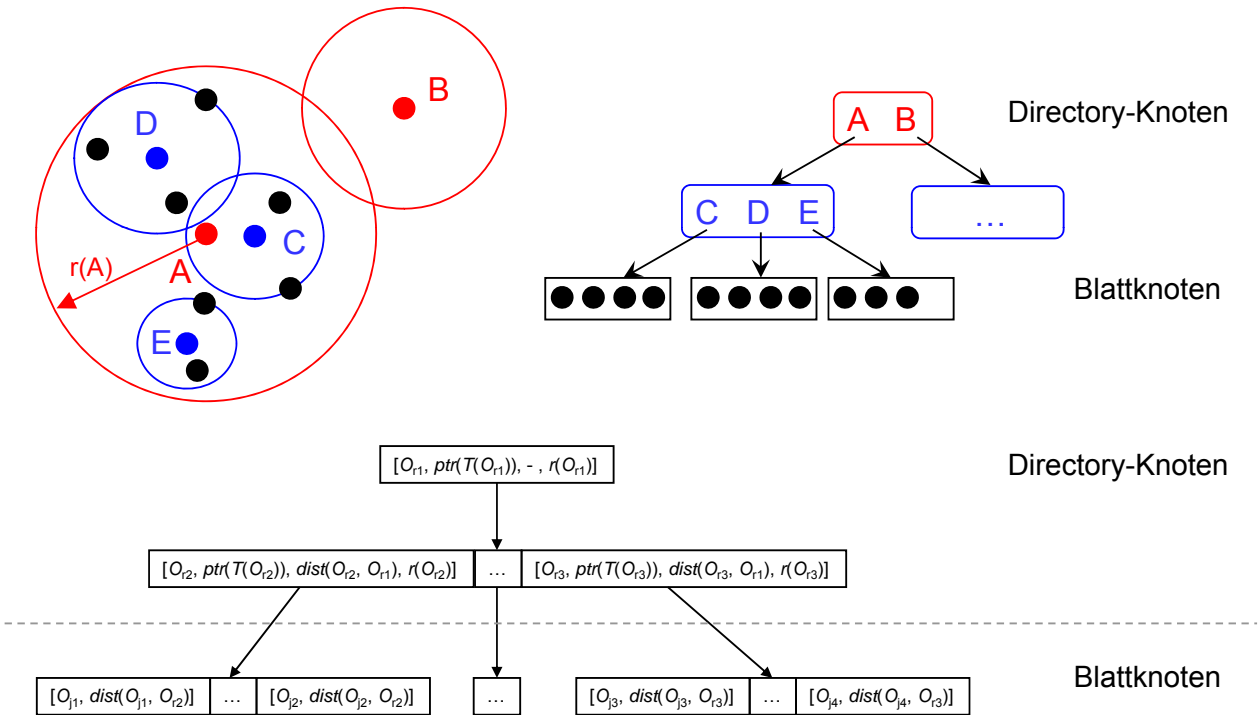


- Allgemein: $3^d - 1$
- Seitenregionen überlappen sich stärker
- Folge: oft muss komplettes Directory gelesen werden

– Beispiel: M-tree

- Dynamische Indexstruktur für allgemeine metrische Räume
- Die Distanzfunktion zur Berechnung der Ähnlichkeit zweier Objekte muss die Eigenschaften einer Metrik erfüllen
- Design
 - Balancierter Index mit einheitlich großen Daten-/Directoryseiten
 - Die indexierten Datenbank-Objekte werden in den Blattknoten abgespeichert
 - Die Directory-Knoten enthalten sog. *Routing Objekte*
 - Routing Objekte entsprechen Datenbank-Objekten, denen eine *Routing Rolle* zugewiesen wurde
 - Wenn ein Knoten überläuft und geplittet werden muß, vergibt der Splitalgorithmus eine Routing Rolle an ein Objekt
 - Zusätzlich zur Objektbeschreibung enthält ein Routing Objekt einen Zeiger auf seinen zugehörigen Unterbaum und den Radius, in dem sich alle Objekte des Unterbaums befinden
 - Wahl der Routing Objekte: die beiden am weitesten voneinander entfernt liegenden Objekte der übergelaufenen Seite

- M-tree Struktur



2.2.3 Mehrstufige Anfragebearbeitung

- Idee:

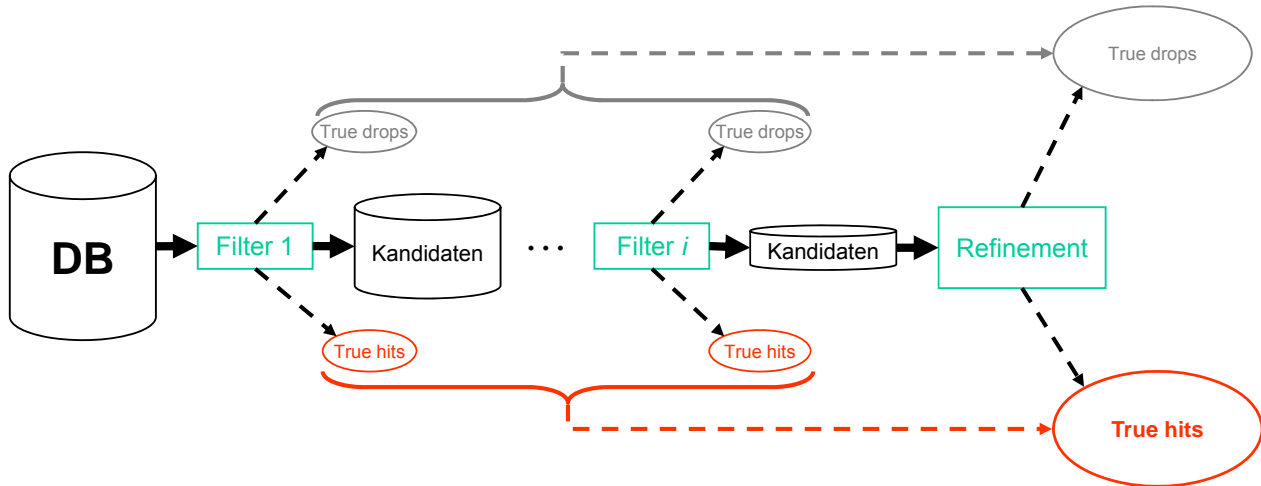
- Verwendetes Distanzmaß ist sehr teuer (z.B. Edit-Distanz) oder nicht feature-basiert (z.B. Überlappungsfläche von Polygonen)
- Vektorraum sehr hochdimensional (Curse of Dimensionality)
- Benutze ein feature-basiertes (meist niedrig-dimensionaler Vektorraum) Distanzmaß als Filterschritt (Filterdistanz)
 - Filterdistanz sollte billiger sein als exakte Distanz (entsprechend niedrig-dimensional => Dimensionsreduktion?)
 - Werte Anfrage-Prädikat (Distanzberechnung) mit Filterdistanz aus
 - Ergebnisse sind noch keine exakten Treffer sondern Kandidaten
 - Kandidatenmenge sollte möglichst klein sein (Filterselektivität)
 - Filterselektivität

$$\sigma_F = \frac{\text{\#Kandidaten}}{n}$$

- Verfeinerung: für die Kandidaten wird das eigentliche Distanzmaß berechnet, was i.A. teurer dafür selektiver als der Filterschritt ist

– Mehrstufige Anfragebearbeitung:

- Ein oder mehrere (kaskadierende) Filterschritte schränken die Kandidatenmenge sukzessive ein
- Verfeinerungsschritt testet auf Korrektheit der Kandidaten



– Zusammenpassen von Filter und Refinement

- Idealfall: Filterdistanz ist obere oder untere Schranke (upper/lower bound) der exakten Distanz => es kann garantiert werden, dass keine exakten Treffer verloren gehen (no false dismissals/drops)
- Sonst: Ergebnisse u.U. nicht vollständig!!!
- Lower Bounding Filter F_{LB}

$$\forall x, y \in DB : dist_{F_{LB}}(F_{LB}(x), F_{LB}(y)) \leq dist(x, y)$$

- Konservative Approximation (d.h. Obermenge) der Ergebnismenge
- Objekte können evtl. bereits aufgrund der Filterdistanz als exakte Fehltreffer (true drops) identifiziert werden

- Upper Bounding Filter F_{UB}

$$\forall x, y \in DB : dist_{F_{UB}}(F_{UB}(x), F_{UB}(y)) \geq dist(x, y)$$

- Progressive Approximation (d.h. Untermenge) der Ergebnismenge
- Objekte können evtl. bereits aufgrund der Filterdistanz als exakte Treffer (true hits) identifiziert werden

2.3 Bereichsanfragen

– Allgemeines

- Eigenschaften

- Benutzer gibt Anfrageobjekt q und maximale Distanz ε vor
- Ergebnis enthält alle Objekte, die höchstens eine Distanz von ε zu q haben

- Formal

$$RQ(q, \varepsilon) = \{o \in DB \mid dist(q, o) \leq \varepsilon\}$$

– Basisalgorithmus (sequential scan)

RQ-SeqScan(DB, q , ε)

result = \emptyset ;

FOR $i=1$ **TO** n **DO**

IF $dist(q, DB.getObject(i)) \leq \varepsilon$ **THEN**

 result := result \cup getObject(i);

RETURN result;

– Algorithmus mit Index: Tiefensuche

RQ-Index(pa, q , ε) // pa = Diskadress z.B. der Wurzel des Indexes

result = \emptyset ;

p := pa.loadPage();

IF p.isDataPage() **THEN**

FOR $i=0$ **TO** p.size() **DO**

IF $dist(q, p.getObject(i)) \leq \varepsilon$ **THEN**

 result := result \cup getObject(i);

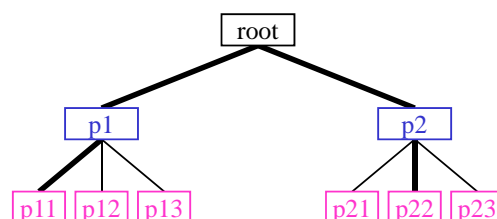
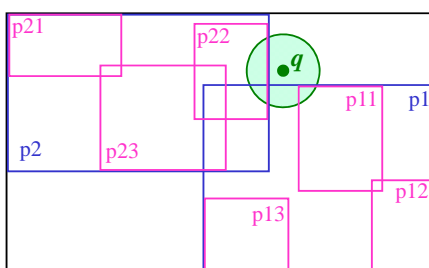
ELSE // p ist Directoryseite

FOR $i=0$ **TO** p.size() **DO**

IF MINDIST($q, p.getRegion(i)$) $\leq \varepsilon$ **THEN**

 result := result \cup RQ-Index(p.childPage(i), q , ε)

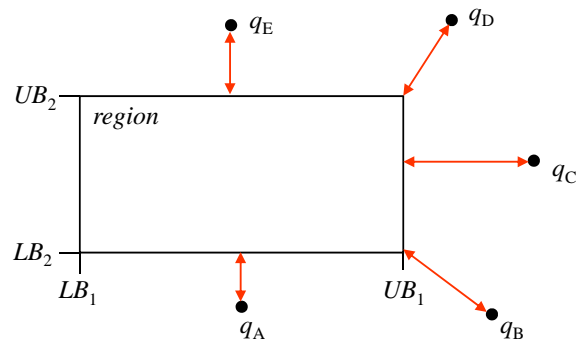
RETURN result;



- MINDIST

- Test ob Queryregion sich mit Seitenregion schneidet
- Minimale Distanz zwischen Anfragepunkt und allen Punkten der Seitenregion (=> Lower Bound!!!)
- Beispiel: Berechnung der MINDIST für L_2 -Norm

$$\text{MINDIST}(\text{region}, q) = \sqrt{\sum_{0 < i \leq d} \begin{cases} (\text{region.LB}_i - q_i)^2 & \text{if } q_i \leq \text{region.LB}_i \\ 0 & \text{if } \text{region.LB}_i \leq q_i \leq \text{region.UB}_i \\ (q_i - \text{region.UB}_i)^2 & \text{if } \text{region.UB}_i \leq q_i \end{cases}}$$



- Mehrstufiger Algorithmus: Filter-/Refinement

- Lower Bounding Filterdistanz LB
- Upper Bounding Filterdistanz UB

RQ-MultiStep(DB, q, ϵ)

result = \emptyset ;

candidates = \emptyset ;

// Filter

FOR $i=1$ **TO** n **DO**

IF UB(q, DB.getObject(i)) $\leq \epsilon$ **THEN**

 result := result \cup getObject(i);

ELSE IF LB(q, DB.getObject(i)) $\leq \epsilon$ **THEN**

 candidates := candidates \cup getObject(i);

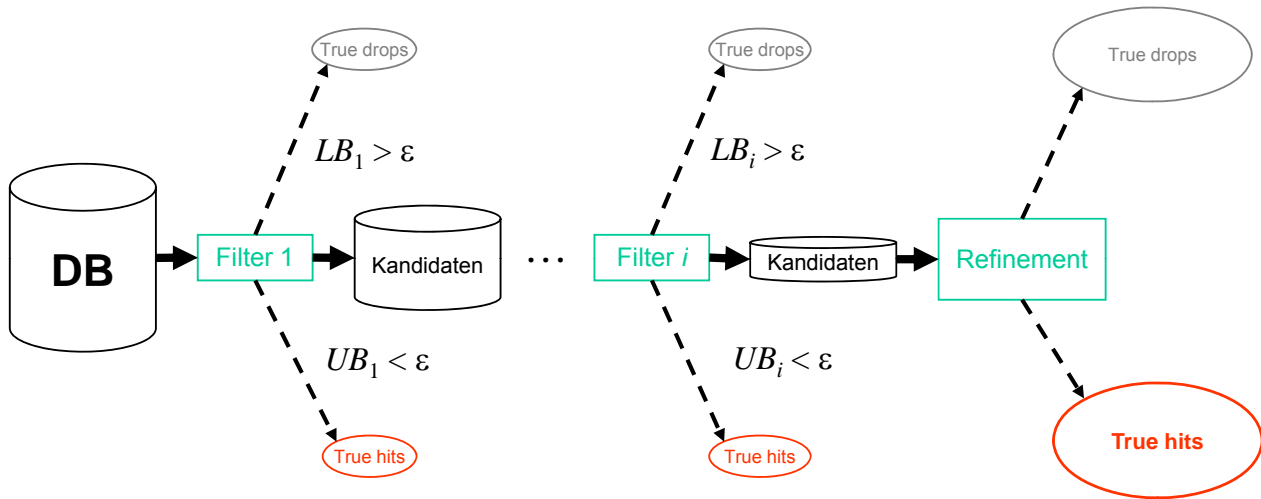
// Refinement

FOR $i=1$ **TO** candidates.size() **DO**

IF dist(q, candidates.getObject(i)) $\leq \epsilon$ **THEN**

 result := result \cup candidates.getObject(i);

RETURN result;



- Oft nur Lower Bounding Distanzen
- => Anzahl der Kandidaten größer, da keine true hits