Kapitel 2
Prinzipien der Anfragebearbeitung in STMM-DBS

Skript zur Vorlesung: Spatial, Temporal, and Multimedia Databases
Sommersemester 2009, LMU München

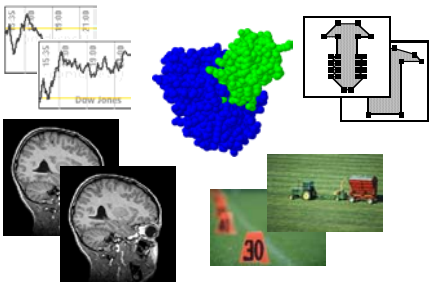
Übersicht

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)

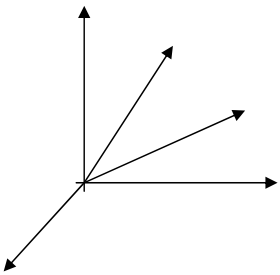
- Extrahiere charakteristische (nummerische) Eigenschaften („Features“) aus den Objekten



Anwendungsgebiete (Objektraum)

Feature-Transformation

Histogramm-Bildung
Moment-Invarianten
Rechteck-Abdeckung
Sektoren-Ermittlung
Fourier-Transformation Kurvatur
usw.



Featurevektor-Raum

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

– Metrischer Raum

Ein metrischer Raum ist ein Tupel Φ_M = (Dom, dist) mit
  - (Dom,dist) ist ein allgemeiner Feature-Raum
  - dist erfüllt zusätzlich die
    – Dreiecksungleichung $\forall x,y,z \in Dom: dist(x,z) \leq dist(x,y) + dist(x,z)$

– Vektorraum

Ein (euklidischer) Vektorraum der Dimension *d* (*d*-dimensionaler Vektorraum) ist ein Tupel Φ_E = (Dom, dist) mit

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

– Feature-Transformation

Eine Feature-Transformation ist eine Abbildung

T: OBJ $\rightarrow$ (Dom,dist)

die jedem Objekt o$\in$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.

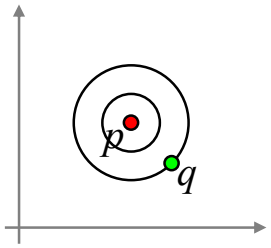
$\forall x,y \in OBJ: dist_{OBJ}(x,y) \equiv dist_{Dom}(T(x),T(y))$

– Distanzmaße in Vektorräumen

- Euklidische Norm (L_2):

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

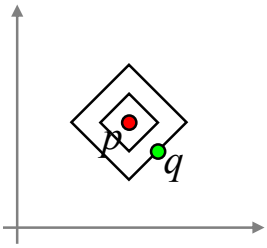
  Natürliches Distanzmaß



- Manhattan-Norm (L_1):

  dist = $|p_1-q_1|+|p_2-q_2|+...$

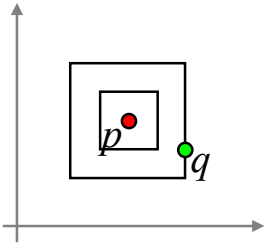
  Die Unähnlichkeiten der einzelnen Merkmale werden direkt addiert



- Maximums-Norm (L_∞):

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

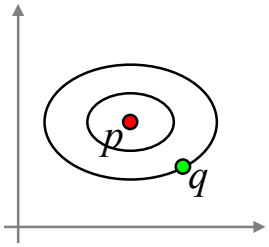
  Die Unähnlichkeit des am wenigsten ähnlichen Merkmals zählt



- Verallgemeinerung L_p-Abstand: $dist_p = (|p_1-q_1|^p + |p_2-q_2|^p + ...)^{1/p}$
- Gewichtete Euklidische Norm:

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

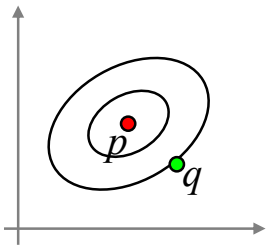
  Häufig sind die Wertebereiche der Merkmale deutliche unterschiedlich
  Beispiel: Merkmal $M_1 \in [0.01 \ldots 0.05]$
  Merkmal $M_2 \in [3.07 \ldots 22.2]$
  Damit M1 überhaupt berücksichtigt wird muss es höher gewichtet werden



- Quadratische Form:

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

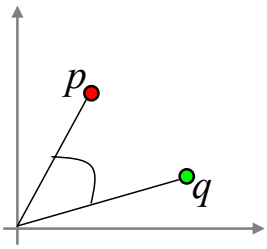
  Bisherige Abstandsmasse gewichten Merkmale nur getrennt
  Besonders bei Farbhistogrammen müssen verschiedene Merkmale gemeinsam gewichtet werden



- Cosinus-Distanz

  $dist$ = cos(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* · Kosten für das Anfrage-Prädikat)
  - Indexbasierte Suche
  - Mehrstufige Anfragebearbeitung

2.2.2 Indexstrukturen

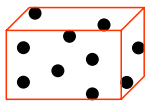
- Prinzip [Böhm, Berchtold, Keim. ACM Computing Surveys, 2001]
  - Organisiere Objekte der Datenbank so, dass während einer Ähnlichkeitsanfrage nur auf „relevante“ Objekte zugegriffen werden muss
  - Baumartige Organisation; jedem Knoten des Baumes ist zugeordnet
    - Seite des Hintergrundspeichers
    - 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)

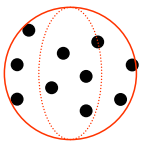
$$c_{\text{Data}} = \left\lfloor \frac{\text{Seitengröße} - \text{Verwaltungsoverhead}}{\text{Größe eines Datensatzes}} \right\rfloor \qquad c_{\text{Directory}} = \left\lfloor \frac{\text{Seitengröße} - \text{Verwaltungsoverhead}}{\text{Größe eines Directoryeintrags}} \right\rfloor$$

  - 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

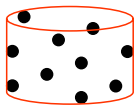
- Gestalt der Seitenregionen
  - Vektordaten:
    - » Achsenparallel Rechtecke, die minimal um die Punktmenge gespannt werden (MUR=minimal umgebendes Rechteck/ MBR=minimum bounding rectangle) (R-Tree, R*-Tree, X-Tree, …)



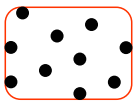
    - » Kugeln (SS-Tree)



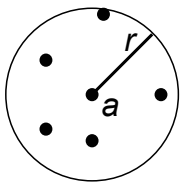
    - » Zylinder (TV-Tree)



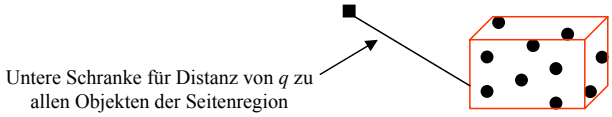
    - » Kombinationskörper (Kugel+MBR, SR-Tree)



  - Algemein metrische Daten
    - » Kein „Raum“ im Euklidischen Sinne, keine Geometrie
    - » Ankerobjekt *a* (anchor object) + Hüllradius *r* (covering radius)
      Für alle Objekte *o* der Seitenregion gilt: dist(*o*,*a*) ≤ *r*



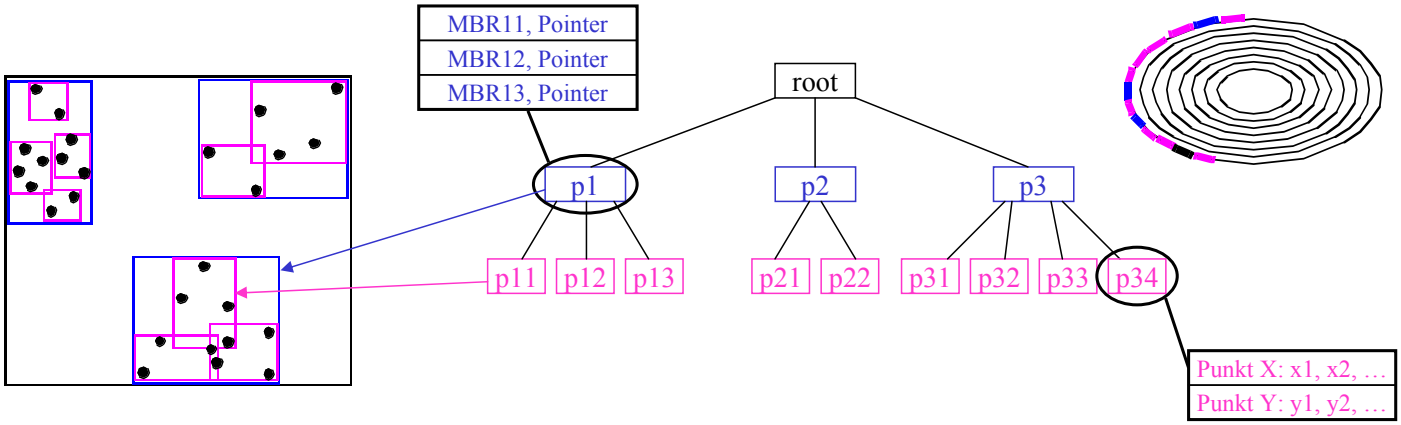
- Seitenregion umfasst immer alle Objekte der Datenseite bzw. des Teilbaums der Directoryseite
  - Konservative Approximation, lower-bounding property: die Distanz von einem Objekt zu einer Seitenregion kann immer mit einer unteren Schranke abgeschätzt werden



  - 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, …)
    - » Ensprechendes Einfügen/Löschen im Elternknoten (rekursiv, notfalls bis Wurzel)

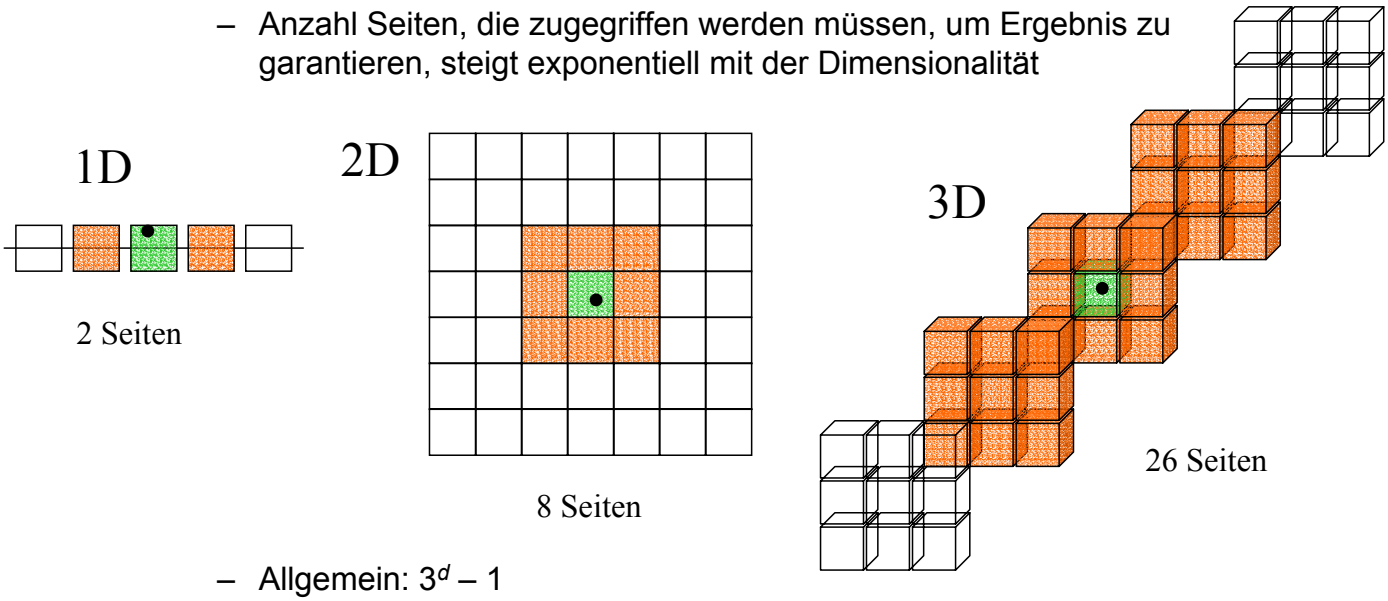
– Beispiel: R*-tree

- 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

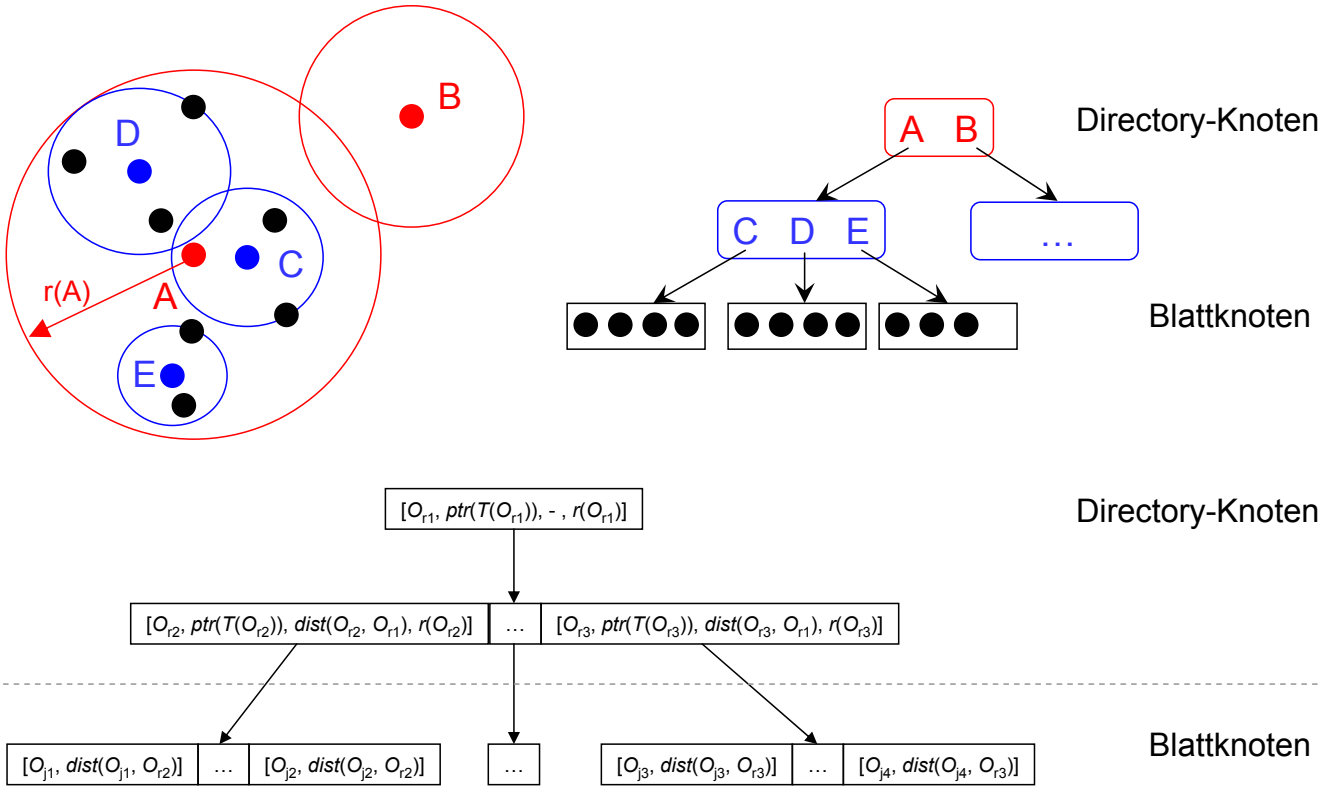


  - Allgemein: $3^d - 1$
  - Seitenregionen überlappen sich stärker
  - Folge: oft muss komplettes Directory gelesen werden

– Beispiel: M-tree

- Dynamische Indexstruktur für allgemeine metrische Räume
- Die Distanzfunktion zur Berechung 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 Databank-Objekten, denen eine *Routing Rolle* zugewiesen wurde
  - Wenn ein Knoten überläuft und geplittet werden muß, vergibt der Splitalgoritmus 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

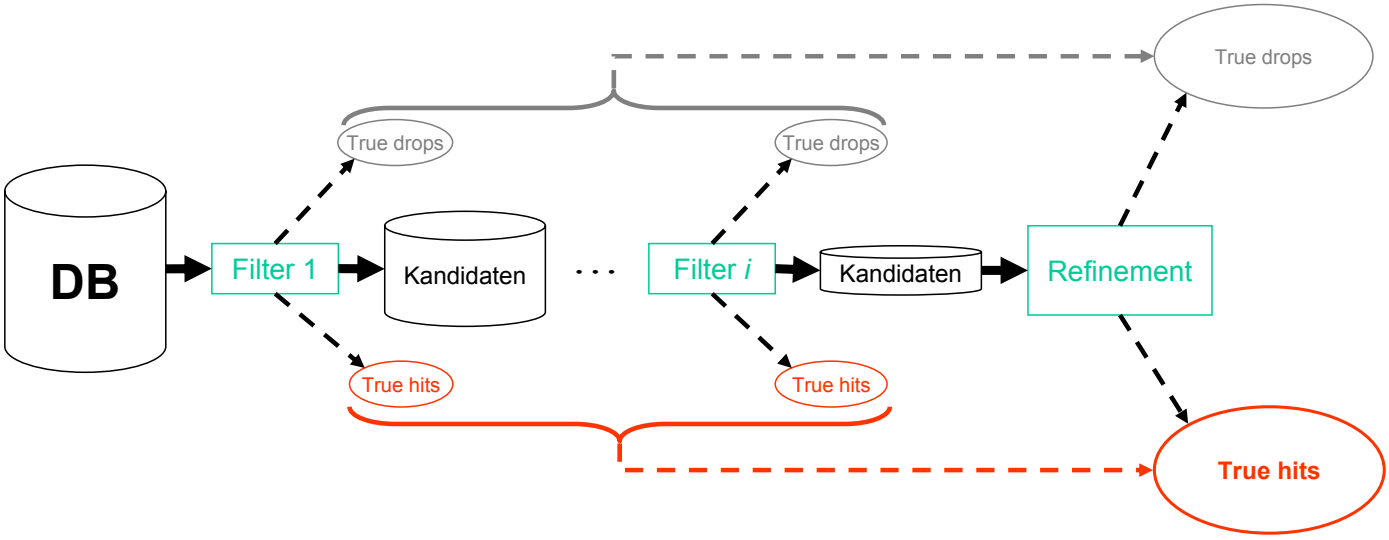
– Idee:

- 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
  - Verfeinerungschritt testet auf Korrektheit der Kandidaten



– Zusammenpassen von Filter und Refinement
  - Idealfall: Filterdistanz ist obere oder unter 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

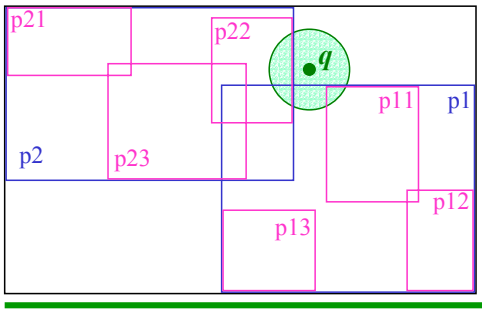
$$RQ(q,\varepsilon) = \{o \in DB \mid dist(q,o) \leq \varepsilon\}$$

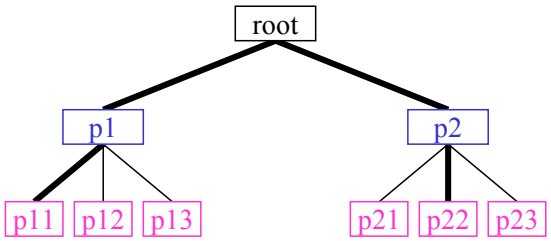
– Basisalgorithmus (sequential scan)

```
RQ-SeqScan(DB, q, ε)
    result = ∅;
    FOR i=1 TO n DO
        IF dist(q, DB.getObject(i)) ≤ ε THEN
            result := result ∪ getObject(i);
    RETURN result;
```


– Algorithmus mit Index: Tiefensuche

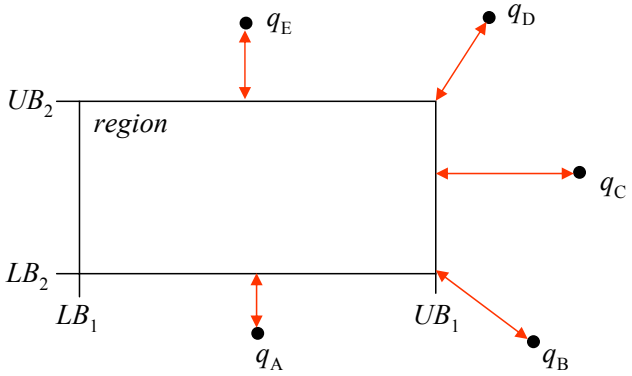
```
RQ-Index(pa, q, ε)          // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i)) ≤ ε THEN
                result := result ∪ getObject(i);

    ELSE                        // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ ε THEN
                result := result ∪ RQ-Index(p.childPage(i), q, ε)
    RETURN result;
```





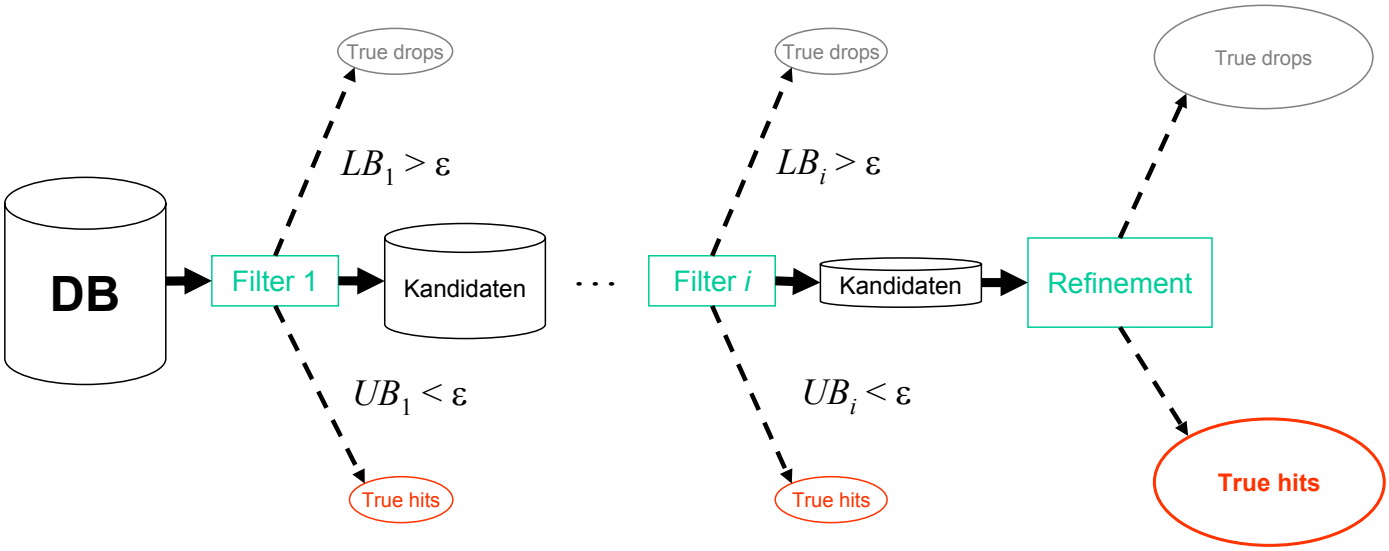
- MINDIST
  - Test ob Queryregion sich mit Seitenregion schneidet
  - Minimale Distanz zwischen Anfragepunkt und allen Punkten der Seitenregion (=> Lower Bound!!!)
  - Beispiel: Berechnung der MINDIST für L_2-Norm

$$\text{MINDIST}(region, q) = \sqrt{\sum_{0<i\le d} \begin{cases} (region.LB_i - q_i)^2 & \text{if} \quad q_i \le region.LB_i \\ 0 & \text{if} \quad region.LB_i \le q_i \le region.UB_i \\ (q_i - region.UB_i)^2 & \text{if} \quad region.UB_i \le q_i \end{cases}}$$



- Mehrstufiger Algorithmus: Filter-/Refinement
  - Lower Bounding Filterdistanz LB
  - Upper Bounding Filterdistanz UB

```
RQ-MultiStep(DB, q, ε)
    result = ∅;
    candidates = ∅;

    // Filter
    FOR i=1 TO n DO
        IF UB(q, DB.getObject(i)) ≤ ε THEN
            result := result ∪ getObject(i);
        ELSE IF LB(q, DB.getObject(i)) ≤ ε THEN
            candidates := candidates ∪ getObject(i);

    // Refinement
    FOR i=1 TO candiates.size() DO
        IF dist(q, candidates.getObject(i)) ≤ ε THEN
            result := result ∪ candidates.getObject(i);
    RETURN result;
```



– Oft nur Lower Bounding Distanzen

=> Anzahl der Kandidaten größer, da keine true hits

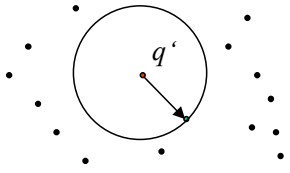
2.4 Nächste Nachbarn Anfragen

2.4.1 Nächste Nachbarn Anfrage

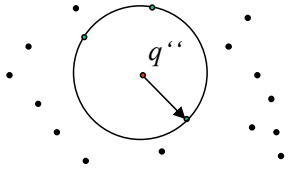
– Allgemeines

- Eigenschaften
  - Benutzer gibt Anfrageobjekt *q* vor
  - Ergebnis enthält das Objekt, das die geringste Distanz zu *q* hat
  - Mehrdeutigkeiten müssen sinnvoll behandelt werden (mehrere nächste Nachbarn, oder nichtdeterministisch ein Objekt)
- Formal

$$NN(q) = \{o \in DB \mid \forall x \in DB : dist(q,o) \leq dist(q,x)\}$$



Eindeutiges Ergebnis



Mehrdeutiges Ergebnis

– Basisalgorithmus (sequential scan): nichtdeterministisch

```
NN-SeqScan(DB, q)
    result = ∅;
    stopdist = +∞;
    FOR i=1 TO n DO
        IF dist(q, DB.getObject(i)) ≤ stopdist THEN
            result := getObject(i);
            stopdist = dist(q, DB.getObject(i));
    RETURN result;
```

– Algorithmus mit Index: Einfache Tiefensuche

- Unterschied zur Range-Query
  - Nächste Nachbar kann beliebig weit vom Anfragepunkt weg liegen
  - Gestalt der Query zunächst unbekannt
  - Es kann zunächst nicht anhand der Seitenregion entschieden werden, ob eine Seite gebraucht wird
  - Ob eine Seite gebraucht wird, hängt auch von dem Inhalt der anderen Seiten ab

  - Kennt man NN-Distanz, würde Range Query ausreichen
  - Kennt man einen beliebigen Objekte, kann man dessen Abstand als obere Schranke für die NN-Distanz nutzen
  - Kennt man mehrere Objekte, kann man den geringsten Abstand als obere Schranke für die NN-Distanz nutzen
- Umformulierung des RQ-Algorithmus:
  - Verwende als ε die kleinste Distanz zu den bisher gefunden Nachbarn

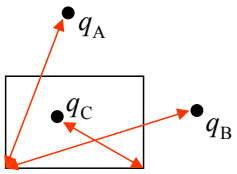
```
Globale Variable: stopdist = +∞;

NN-Index-Simple-TS(pa, q)          // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i)) ≤ stopdist THEN
                result := getObject(i);
                stopdist = dist(q, p.getObject(i));
    ELSE                           // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ stopdist THEN
                result := NN-Index-Simple-TS(p.childPage(i), q)
    RETURN result;
```

- Nachteil des einfachen Tiefensuch-Algorithmus
  - Initialisierung: stopdist = +∞
  - Dadurch: Start mit beliebigem Pfad
  - Folge: die ersten gefundenen Objekte sind meist sehr weit vom Anfrageobjekt entfernt => stopdist ist wenig selektiv
  - Verbesserung: beginne Pfad, der möglichst nah zum Anfrageobjekt liegt

– Algorithmus mit Index: Tiefensuche nach [RKV 95]

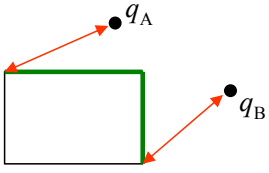
[Roussopoulos, Kelley, Vincent. Proc. ACM Int. Conf. Management of Data (SIGMOD), 1995]

- Vermeidet langsame Einschränkung des Suchraums durch
  - Verwendung der Seitenregionen zur Abschätzung der NN-Distanz
  - Priorisierung der Tiefensuche nach Distanz der Seitenregion zur Query
- Neben MINDIST weitere Abschätzungen der NN-Distanz durch:
  - MAXDIST
    - » Maximale Distanz zwischen Query und allen Punkten der Seitenregion
    - » NN-Distanz kann nicht schlechter als MAXDIST werden



$$\mathrm{MAXDIST}(region, q) = \sqrt{\sum_{0<i\le d} \max\{(q_i - region.UB_i)^2, (q_i - region.LB_i)^2\}}$$

  - MINMAXDIST
    - » MBRs als Seitenregionen: maximale NN-Distanz noch besser abzuschätzen
    - » Auf jeder Kante des MBR muss ein Punkt liegen (sonst ist MBR nicht minimal)
    - » Intuition: „nächstliegende Kante, weitester Punkt“



$$MINMAXDIST(region, q) = \sqrt{\min_{0<k\le d}(|q_i - rm_i|^2 + \sum_{\substack{i\ne k \\ 0<i\le d}} |q_i - rM_i|^2)}$$

wobei $$rm_i = \begin{cases} region.LB_i & \text{if } q_i \le \dfrac{region.LB_i + region.UB_i}{2} \\ regionUB_i & \text{else} \end{cases}$$

$$rM_i = \begin{cases} region.LB_i & \text{if } q_i \ge \dfrac{region.LB_i + region.UB_i}{2} \\ regionUB_i & \text{else} \end{cases}$$

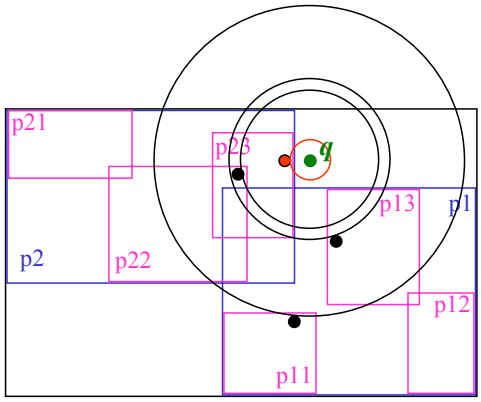
  - Für andere Geometrien (nicht MBRs) sind MINDIST und MAXDIST analog definierbar; MINMAXDIST allerdings nicht
  - Abschätzung von stopdist durch Minimum aus stopdist und MINMAXDIST (bzw. MAXDIST) aller bisher bekannten Seitenregionen (pruningdist)
  - Vor dem rekursiven Abstieg: sortieren der Kindseiten nach MINDIST (experimentell als bestes Prioritätsmaß ermittelt)

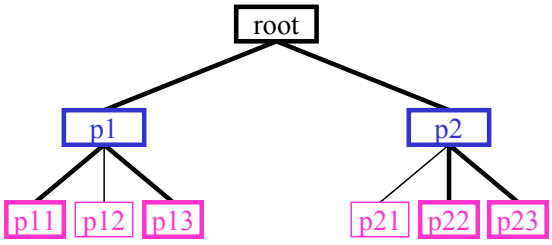
– Algorithmus:

Globale Variablen: stopdist = +∞; pruningdist = +∞;

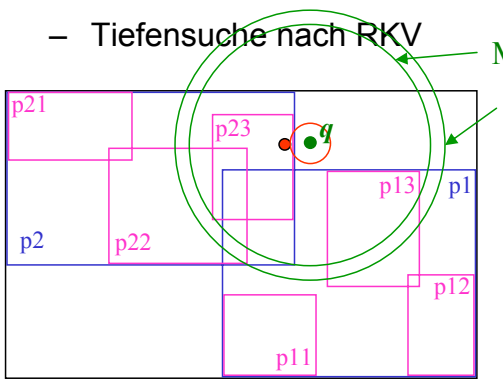
```
NN-Index-RKV-TS(pa, q)          // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i)) ≤ stopdist THEN
                result := getObject(i);
                stopdist = dist(q, p.getObject(i));
            IF stopdist < pruningdist THEN
                pruningdist = stopdist;
    ELSE                        // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINMAXDIST(q, p.getRegion(i)) < pruningdist THEN
                pruningdist = MINMAXDIST(q, p.getRegion(i));
        quicksort(p.getObjectArray(), MINDIST);
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ pruningdist THEN
                result := NN-Index-RKV-TS(p.childPage(i), q);
    RETURN result;
```

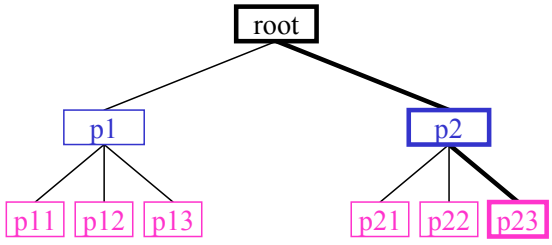

- Ablaufbeispiel
  - Einfache Tiefensuche



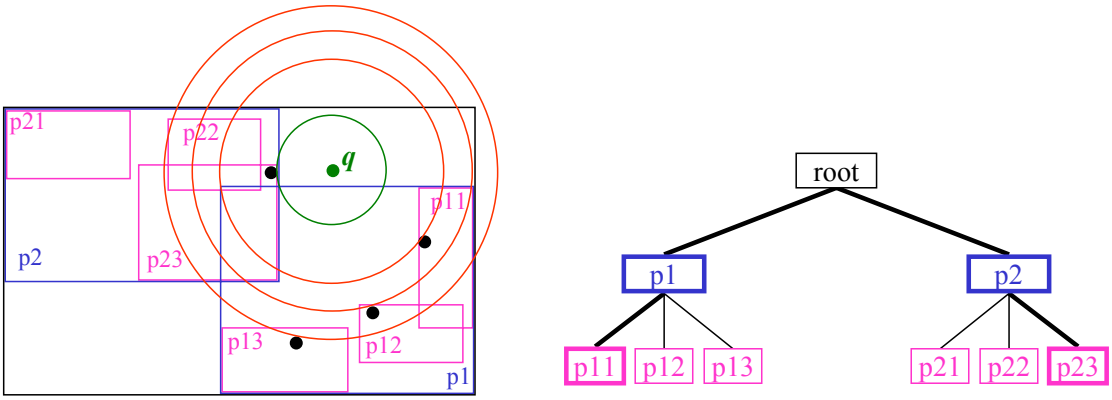


  - Tiefensuche nach RKV



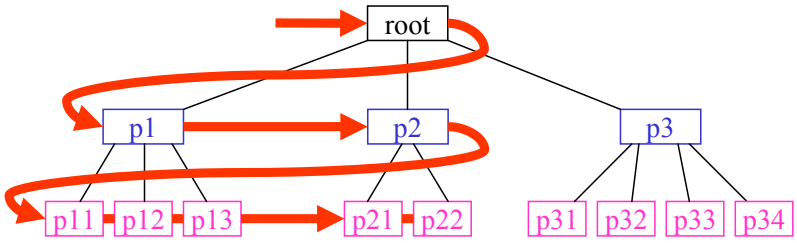


- Fazit:
  - Priorisierung mit MINDIST bewirkt Reduktion der Seitenzugriffe von 7 auf 3
  - MINMAXDIST verbessert Pruning-Distanz, verhindert hier aber keine Seitenzugriffe
  - Trotz Priorisierung: Tiefendurchlauf kann prinzipiell stark fehlgeleitet werden wenn z.B. eine Seite auf dem ersten Level sehr nah am Queryobjekt liegt, ihre Kindseiten aber relativ weit weg



  - Bei Start mit p_2 hätte keine der Kindseiten von p_1 geladen werden müssen

– Algorithmus mit Index: Breitensuche



- Abgesehen von MINMAXDIST-Abschätzung stehen Punktdistanzen erst auf Blatt-Ebene zur Verfügung
  - Viele Zugriffe auf Directoryseiten
  - Die erste Punktdistanz hätte viele dieser Zugriffe schon verhindern können
- Speicherintensiv
  - Worst-case: gesamte letzte Directory-Ebene muss im RAM gehalten werden

– Algorithmus mit Index: Prioritätssuche nach [HS 95]

[Hjaltason, Samet. Proc. Int. Symp. on Large Spatial Databases (SSD), 1995]

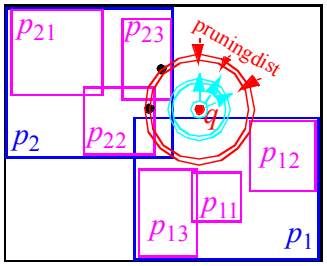
- Statt rekursivem Durchlauf: Liste der aktiven Seiten (active page list APL)
  - Seite p ist aktiv genau dann wenn folgende Bedingungen erfüllt sind:
    - » p wurde noch nicht geladen
    - » Elternseite von p wurde bereits geladen
    - » MINDIST(q, p.getRegion()) ≤ pruningdist
  - APL wird mit Wurzel des Indexes initialisiert
  - Seiten in APL nach MINDIST zum Anfrageobjekt aufsteigend sortiert
  - Algorithmus entnimmt immer die erste Seite aus APL (mit kleinster MINDIST)
  - Entnommene Seite wird geladen und verarbeitet: („verfeinert“)
    - » Datenseiten werden wie bisher verarbeitet
    - » Directoryseiten: Kindseiten mit MINDIST ≤ pruningdist in APL einfügen
  - Ändert sich pruningdist werden Seiten mit MINDIST > pruningdist alternativ:
    - » aus APL entfernt
    - » als gelöscht markiert
    - » ohne explizite Markierung später ignoriert

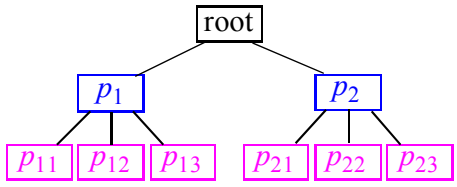
– Algorithmus:

Globale Variablen: stopdist = +∞; pruningdist = +∞;

```
NN-Index-HS(pa, q)              // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    apl = LIST OF (dist:Real, da:DiskAdress) ORDERED BY dist ASCENDING
    apl = [(0.0, pa)]
    WHILE NOT apl.isEmpty() AND apl.first().dist ≤ pruningdist DO
        p := apl.getFirst().da.loadPage();
        apl.deleteFirst();
        IF p.isDataPage() THEN

            (* wie bisher *)

        ELSE                        // p ist Directoryseite
            FOR i=0 TO p.size() DO
                IF MINDIST(q, p.getRegion(i)) ≤ pruningdist THEN
                    apl.insert(MINDIST(q, p.getRegion(i)), p.childPage(i));
    RETURN result;
```

- Beispiel





APL:

1. root
2. p_1 p_2
3. p_2 p_{12} p_{13} p_{11}
4. p_{23} p_{22} p_{12} p_{13} p_{11} p_{21}
5. p_{22} p_{12} p_{13}

- Eigenschaften
  - Allgemein
    - » Seiten werden nach aufsteigendem Abstand geordnet zugegriffen (blaue Kreise)
    - » pruningdist wird kleiner, sobald nähergelegenes Objekt gefunden (rote Kreise)
    - » Anfragebearbeitung stoppt, wenn beide Kreise sich treffen

  - Speicherbedarf
    - » Wie bei Breitensuche kann gesamter unterste Directorylevel in APL stehen
    - » Dieser Fall is allerdings unwahrscheinlicher als bei Breitensuche
    - » Speicherkomplexität O(n) (Tiefensuche O(log n)

- Optimalität des Verfahrens

[Berchtold, Böhm, Keim, Kriegel. ACM Smp. Principles of database Systems (PODS), 1997]

  - Prioritätssuche nach [HS 95] ist optimal bzgl. der Anzahl der Seitenzugriffe
  - Beweis (Überblick):
    - » Lemma 1: jeder korrekte Algorithmus muss mind. die Seiten laden, die von der NN-Kugel um q berührt werden
    - » Lemma 2: das Verfahren greift auf Seiten in aufsteigendem Abstand von q zu
    - » Lemma 3: keine Seite s wird zugegriffen, mit MINDIST(q, s) > NN-Distanz(q)

  - **Lemma 1:** Ein korrekter NN-Algorithmus muss mind. die Seiten s laden, die MINDIST(q, s) ≤ NN-Distanz(q) erfüllen.

    Beweis: Angenommen eine Seite s mit MINDIST(q, s) ≤ NN-Distanz(q) wird nicht geladen. Dann kann diese Seite Punkte enthalten (als Datenseite; Directoryseiten können im entspr. Teilbaum Punkte speichern), die näher am Anfragepunkt liegen als der nächste Nachbar. Der nächste Nachbar ist also nicht als solcher validiert, da über Punkte in einem Teilbaum keine Infos bekannt sind, außer dass sie in der entsprechenden Region liegen.

□

– **Lemma 2:** Das Verfahren greift auf die Seiten des Index aufsteigend sortiert nach MINDIST zu.

Beweis: Die Seiten werden in aufsteigender Reihenfolge aus der APL entnommen. Es muss also nur sichergestellt werden, dass nach Entnahme von Seite *s* keine Seiten *s'* mehr in APL eingefügt werden, mit MINDIST(*q*,*s'*) < *r* := MINDIST(*q*,*s*). Alle Seiten, die nach Entnahme von s in APL eingefügt werden, sind entweder Kindseiten von *s* oder Kindseiten von Seiten *s''* mit MINDIST(*q*,*s''*) ≥ *r*. Da die Region einer Kindseite in der Region der Elternseite vollständig eingeschlossen ist, ist die MINDIST einer Kindseite nie kleiner als die der Elternseite. Daher haben alle später eingefügten Seiten eine MINDIST ≥ *r*.

□

– **Lemma 3:** Das Verfahren greift auf keine Seite *s* zu, mit MINDIST(*q*,*s*) > NN-Distanz(*q*).

Beweis: Nach Lemma 2 können nach Zugriff auf Seite *s* nur Punkte *p* gefunden werden, mit dist(*q*, *p)* > MINDIST(*q*, *s*). Wäre vor Zugriff auf *s* ein Punkt *p* mit dist(*q*, *p)* < MINDIST(*q*, *s*) gefunden worden, dann wäre s aus der APL gelöscht worden bzw. der Algorithmus hätte vor der Bearbeitung von *p* angehalten.

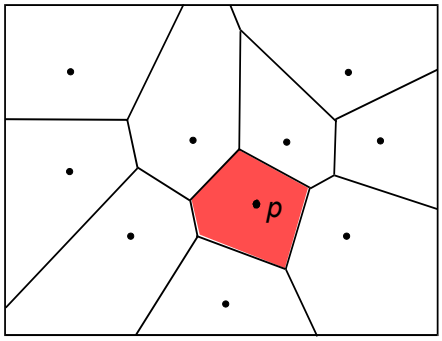
□

– Aus Lemma 1-3 ergibt sich, dass der Algorithmus nach [HS 95] optimal bzgl. der Anzahl der Seitenzugriffe ist.

– Hybrider Algorithmus: Voronoi-Diagramme

[Berchtold, Ertl, Keim, Kriegel, Seidl. Proc. Int. Conf. Data Engineeering (ICDE), 1998]

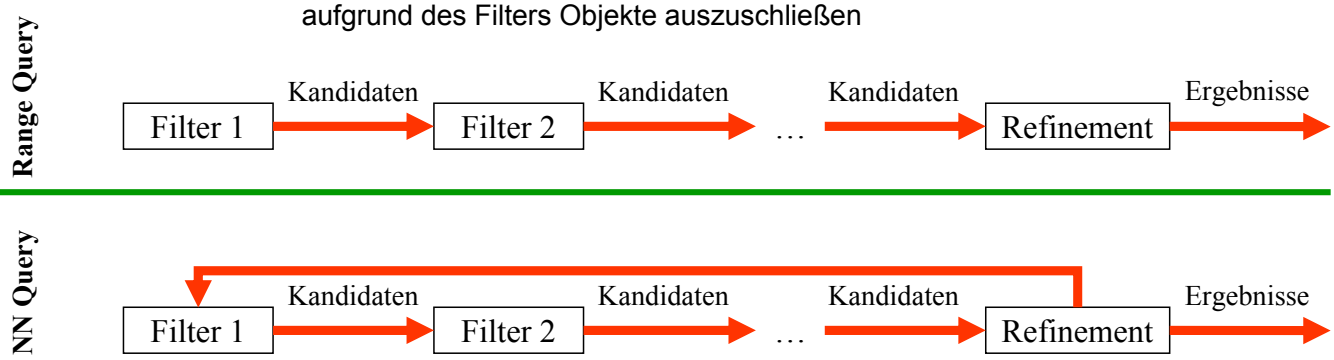
- Nur für Vektordaten!!!
- Idee:
  - Berechne für jeden Punkt *p* den Teil des Datenraumes in dem *p* der nächste Nachbar ist (Voronoi-Zellen)
  - Speichere Voronoi-Zellen in DB
  - NN-Anfrage entspricht Punktanfrage mit *q* auf den Voronoi-Zellen
    => Punkt *p*, in dessen Voronoi-Zelle *q* liegt, ist der NN von *q*



- Problem:
  - Voronoi-Zellen sind konvexe Polygone
  - Ab *d* > 2 sehr komplex (große Anzahl Eckpunkte)
- Lösung: Approximation der Voronoi-Zellen (z.B. mit MBR)
  - Nur Filterschritt, da MBRs sich überlappen können, und q in mehrere dieser MBRs liegen kann
    => Verfeinerung der Kandidaten mit exakten Punktdistanzen

– Mehrstufiger Algorithmus: Filter/Refinement

- Allgemeines
  - Algorithmen verwenden meist nur LB-Filter
  - Bei mehreren Filterschritten: $dist_{Filter1} \leq dist_{Filter2} \leq \ldots$
  - Unterschied zu Bereichsanfragen:
    - » RQs können durch einfache Hintereinanderschaltung der Filterschritte und der Verfeinerung ausgewertet werden
    - » Bei NN-Queries nicht so leicht möglich, da der NN-Kandidat des (ersten) Filters nicht notwendigerweise der exakte NN sein muss
    - » Bei geeigneter Filterdistanz ist es aber wahrscheinlich, dass exakter NN unter den ersten NN-Kandidaten des Filterschritts ist
    - » Rückmeldung der im Refinement ermittelten Distanzen an den Filterschritt um aufgrund des Filters Objekte auszuschließen



- Im folgenden:
  - Verschiedene Auswertungsstrategien
  - Ein Filterschritt (leicht generalisierbar auf mehrere Filterschritte)

- Auswertung mit Bereichsanfrage

  [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. Proc. Int. Conf. Very Large Databases (VLDB), 1996]
  [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. TKDE 10(6), 1998]
  - Idee
    - » Verfeinerungsdistanz ε eines beliebigen Punktes ist obere Schranke für die NN-Distanz
    - » Folge: ist p der NN von q, so gilt $dist(p, q) \leq \varepsilon$ und $LB_{Filter}(p, q) \leq \varepsilon$
    - » Also: $p \in RQ(q, \varepsilon)$
    - » Gutes ε ist z.B. der NN von q bzgl. der Filterdistanz
  - Prinzip
    - » Auf Filterebene NNQ ausführen
    - » Anschließend eine RQ ausführen (mit Index oder mehrstufig)
    - » Auf dem (hoffentlich kleinen) Ergebnis der RQ den exakten Test (Refinement) durchführen

– Algorithmus

```
NN-MultiStep-RQ(DB, q)
    r = NN-Query auf der Filterdistanz;          // beliebig implementierbar
    ε = dist(q, r);
    candidates = RQ-MultiStep(DB, q, ε);
    result = r ;
    stopdist = ε;

    // Refinement
    FOR EACH p∈candidates DO
        IF dist(p,q) ≤ stopdist THEN
            stopdist = dist(q, p)
            result = p;
    RETURN result;
```

– Vorteil
  » Einfacher Algorithmus
– Nachteil
  » Leistung stark von Filterselektivität abhängig: schlechter Filter => großes ε => große Ergebnismenge der RQ => hohe Kosten für Verfeinerung

- Auswertung mit unmittelbarer Verfeinerung
  – Idee
    » Jedes Objekt, das nicht aufgrund des Filters ausgeschlossen werden kann, wird sofort verfeinert
    » Einbau in einen beliebigen NN-Algorithmus, z.B. in NN-Index-Simple-TS (S. 60)
  – Algorithmus

```
NN-MultiStep-Simple(pa, q)      // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist_Filter(q, p.getObject(i)) ≤ stopdist THEN
                IF dist(q, p.getObject(i)) ≤ stopdist THEN
                    result := getObject(i);
                    stopdist = dist(q, p.getObject(i));
    ELSE                        // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ stopdist THEN
                result := NN-MultiStep-Simple(p.childPage(i), q)
    RETURN result;
```

- Vorteil
  - » Gute Speicherplatzkomplexität (je nach NN-Algorithmus!!!), da keine Kandidaten zwischen gespeichert werden müssen
  - » Einfache Erweiterung eines beliebigen NN-Algorithmus
- Nachteil
  - » Hohe Verfeinerungskosten (fast alle Punkte), wenn Filter wenig selektiv ist oder NN-Algorithmus langsam konvergiert

• Auswertung nach Priorität

[Seidl, Kriegel. Proc. ACM Int. Conf. Management of Data (SIGMOD),1998]

- Auf Filterebene läuft „Ranking Query“ ab (siehe Kapitel 2.4.3)
  - » Funktion getNext(): liefert beim ersten Aufruf den 1. Nachbarn, beim zweiten Aufruf den 2. Nachbarn, usw.
  - » Rufe solange getNext() auf, bis das erhaltene Objekt die aktuelle stopdist überschreitet
  - » Verfeinere das erhaltene Objekt und passe ggf. die stopdist an
- Vorteil
  - » Beweisbar: Algorithmus optimal, d.h. eine minimale Anzahl von Kandidaten werden verfeinert
- Nachteil
  - » Komplexität des Ranking-Algorithmus (Speicher und/oder Zeit)

- Algorithmus

```
NN-MultiStep-Optimal(DB, q)
    Ranking = initialisiere Ranking bzgl. q auf Filterdistanz      // Kapitel 2.4.3
    result = ∅;
    stopdist = +∞;
    REPEAT
        p = Ranking.getNext();
        filterdist = dist_Filter(p, q);
        IF dist(q, p) ≤ stopdist THEN
            result = p;
            stopdist = dist(q, p);
    UNTIL filterdist > stopdist;
    RETURN result;
```