LMU LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
DEPARTMENT INSTITUTE FOR INFORMATICS
DATABASE SYSTEMS GROUP

Kapitel 5: Räumliche Anfragebearbeitung

Skript zur Vorlesung

Geo-Informationssysteme

Wintersemester 2014/15

Ludwig-Maximilians-Universität München

5. Räumliche Anfragebearbeitung

1. Algorithmen für räumliche Anfragen
2. Mehrstufige Anfragebearbeitung

Beispiel-Anfragen in Relationaler Algebra

(1) cities select [center inside Bavaria]
"Bavaria" sei eine Konstante des Typs *region*

(2) rivers select [route intersects Window]

(3) cities select [dist(center,Hagen) < 100 and population > 500.000]

(4) cities states join [center inside area]

(5) cities rivers join [dist(center,route) < 50]

5.1 Räumliche Anfragen

Notationen

- GEO = {***points, lines, regions***}
- OBJ = {**cities, highways, . . .**} ***(anwendungsspezifische Typen mit einem GEO-Attribut)***
- ∀ *obj in OBJ*, ∀ *geo, geo1, geo2 in GEO*

Fensteranfrage

set(obj) × *(obj* → *geo1)* × *geo2* × *(geo1* × *geo2* → *bool)* → *set(obj)*

WindowQuery(DB, window, predicate) = {*o* ∈ *DB* | predicate(*o*, window)}

predicate z.B. inside, intersect

5.1 Räumliche Anfragen

Notationen

- GEO = {***points, lines, regions***}
- OBJ = {**cities, highways, . . .**} ***(anwendungsspezifische Typen mit einem GEO-Attribut)***
- ∀ *obj in OBJ,* ∀ *geo, geo1, geo2 in GEO*

Nächste-Nachbarn-Anfrage

set(obj) × *(obj* → *geo1)* × *geo2* → *set(obj)*

NearestNeighborQuery(DB, point) =

{*o* ∈ DB | ∀ o′ ∈ DB: dist(point, *o*) ≤ dist(point, *o*′)}

5.1 Räumliche Anfragen

Notationen

- GEO = {***points, lines, regions***}
- OBJ = {**cities, highways, . . .**} ***(anwendungsspezifische Typen mit einem GEO-Attribut)***
- ∀ *obj in OBJ,* ∀ *geo, geo1, geo2 in GEO*

Räumlicher Verbund

set(obj) × set(obj) × (obj → geo1) × (obj → geo2) × (geo1 x geo2 → bool) → set(obj × obj)

spatial_join(DB1, DB2, predicate) =

$$\{(o_1,o_2) \mid o_1 \in DB_1,\ o_2 \in DB2,\ \text{predicate}(o_1,o_2)\}$$

predicate z.B. “dist ≤ d”, “intersect”, “north” etc.

Algorithmus

WindowQuery(DB, Window, Predicate)

```
Candidates := ∅; // Kandidatenmenge
window_index_query(DB.SpatialIndex.Root, Window, Predicate, Candidates);
                                // siehe Kap. 4 bzw. nächste Seite
Result := ∅; // Ergebnismenge
FOR ALL Candidate IN Candidates
    Object := Candidate.RetrieveExactGeometry();
    IF Predicate(Window, Object) THEN
      Result := Result ∪ {Object};
return Result;
```

5.1 Fensteranfragen

Methode

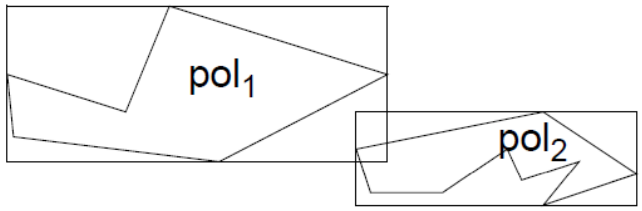
window_index_query(IndexPage, Window, Predicate, Candidates)

```
FOR ALL Entry ∈ Partitionen in IndexPage DO
    IF Predicate(Window, Entry.Rectangle) THEN
        IF Page = DataPage THEN
            Candidates := Candidates ∪ {Entry};
        ELSE
            window_index_query (Entry.Subtree^, Window, Predicate, Candidates);
```

Algorithmus

- Wenn predicate(MUR(pol_1), MUR(pol_2)) gilt,
  dann muss nicht unbedingt predicate(pol_1, pol_2) gelten.

z.B. intersect



- Aber: Wenn NOT predicate(MUR(pol_1), MUR(pol_2)) gilt,
  dann gilt auch NOT predicate(pol1, pol_2).
- Frage: Unter welchen Umständen kann man schon echte Treffer (Polygone, die die Anfrage erfüllen) an der Beziehung der MUR's erkennen?
  - beim Prädikat intersect?
  - Bei anderen Prädikaten? (siehe Übung)

5.1 Nächste-Nachbarn-Anfragen

Parameter

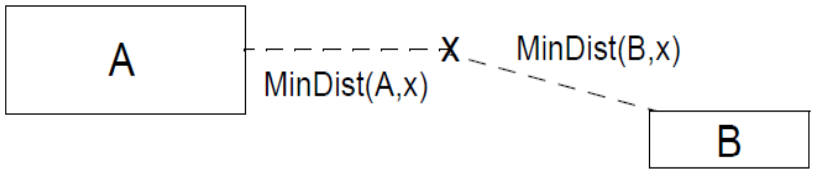
- SpatialIndex (R-Baum, Quadtree, etc.) → DB
- QueryPoint → Obj

Variablen

- PartitionList: Eine Liste von Partitionen des Datenraums, der durch SpatialIndex verwaltet wird. Eine *Partition* ist z.B. durch ein MUR oder durch einen Quadranten definiert. PartitionList wird nach MinDist zum QueryPoint aufsteigend sortiert.
- NN: der Nächste Nachbar von QueryPoint in den bisher gelesenen Datenseiten

Hilfsfunktion

- MinDist(Partition,Point) = minimale Distanz, die theoretisch zwischen dem Anfragepunkt und einem Eintrag der Partition vorkommen kann.

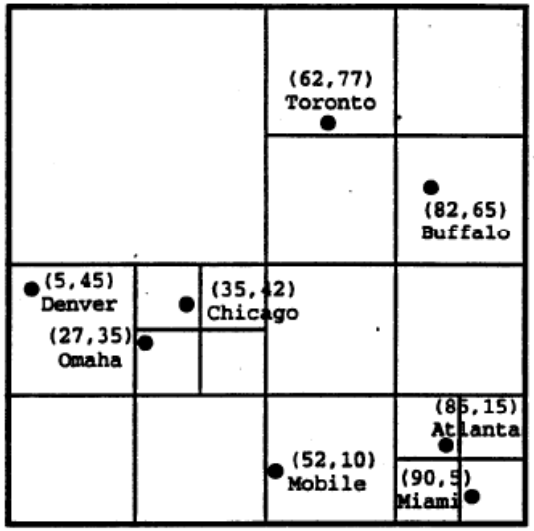


Algorithmus

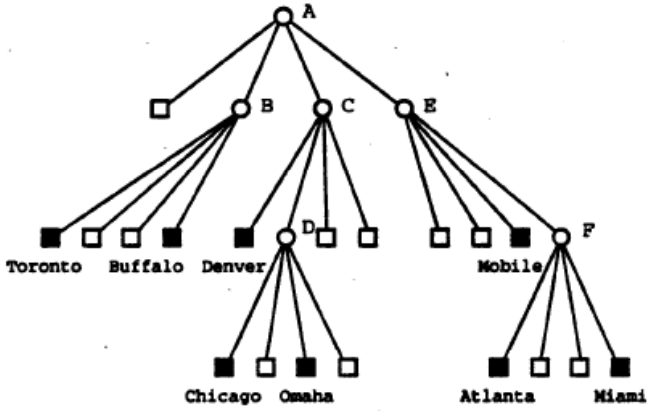
```
Initialisiere PartitionList mit den Root-Partitionen des SpatialIndex;
Sortiere die Einträge p der PartitionList nach MinDist(p, QueryPoint);
NNdist := MAXREAL;
WHILE PartitionList ≠ ∅ DO
  Entferne erstes Element TopPart aus PartitionList;
  IF TopPart ist ein Blatt des SpatialIndex THEN
    FOR EACH Entry ∈ TopPart DO
      NNC := Entry.RetrieveExactGeometry();
      IF dist(QueryPoint, NNC) < NNdist THEN
        NN := NNC; NNdist := dist(QueryPoint, NNC);
      END IF;
    END FOR
    Entferne alle Elemente q aus der PartitionList für die gilt: MinDist(q, QueryPoint) > NNdist;
  ELSIF TopPart liegt in einem inneren Knoten des SpatialIndex THEN
    ersetze TopPart durch seine Sohn-Partitionen;
    Sortiere die PartitionList erneut nach MinDist(part,QueryPoint);
  END IF;
END WHILE;
RETURN NN;
```

5.1 Nächste-Nachbarn-Anfragen

Beispiel



PR-Quadtree
mit Blattkapazität 1

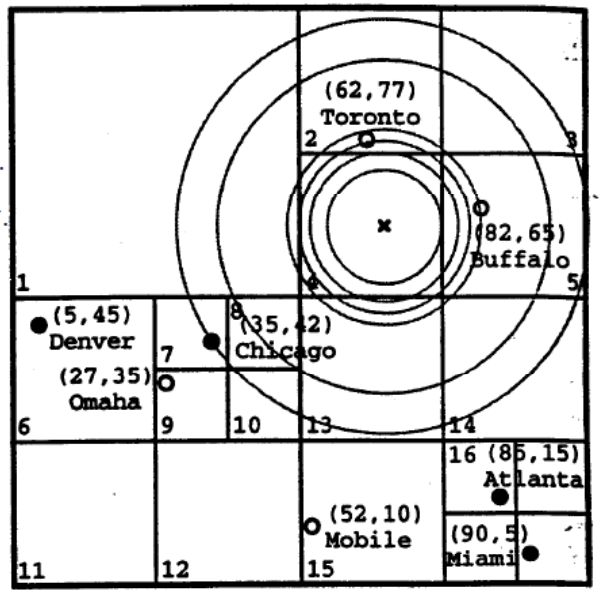


5.1 Nächste-Nachbarn-Anfragen

Beispiel

○ < 1 Million Einw.

● ≥ 1 Million Einw.



Vorausgesetzt wird eine Abzählung der Blätter des PR-Quadtrees

Partitionen des PR-Quadtrees werden codiert als a/b

a = Tiefe

b = *für innere Knoten:* Nummer des nordwestlichsten Blatts im Teilbaum
für Blattknoten: Nummer des Blattes

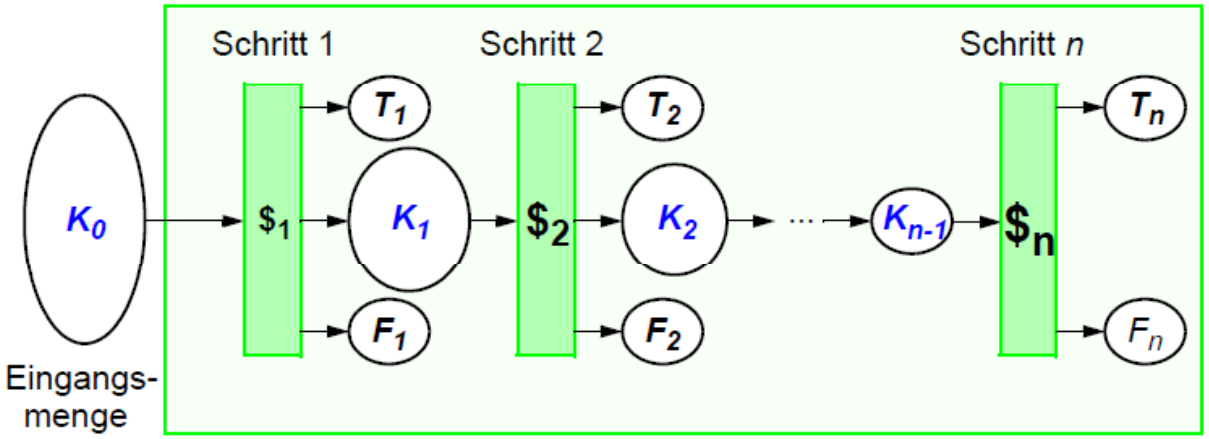
z.B. Wurzel = 0/1, Quadrant NE = ½

Suche die zum Punkt x nächstgelegene Millionenstadt!

- PartitionList = 1. [1/2, 1/13, 1/1, 1/6], 2.[2/4, 2/5, 1/13, 2/2, 1/1, 2/3, 1/6], 3. [2/5, 1/13, 2/2, 1/1, 2/3, 1/6], 4. [1/13, 2/2, 1/1, 2/3, 1/6], 5. [2/13, 2/2, 1/1, 2/3, 2/14, 1/6, 2/15, 2/16], 6. [2/2, 1/1, 2/3, 2/14, 1/6, 2/15, 2/16], 7. [1/1, 2/3, 2/14, 1/6, 2/15, 2/16], . . .
- NN = nach 4.: Buffalo (zu klein), nach 7.: Toronto (zu klein), . . . , Chicago (Millionenstadt)
  → Chicago

5.2 Mehrstufige Anfragebearbeitung (I)

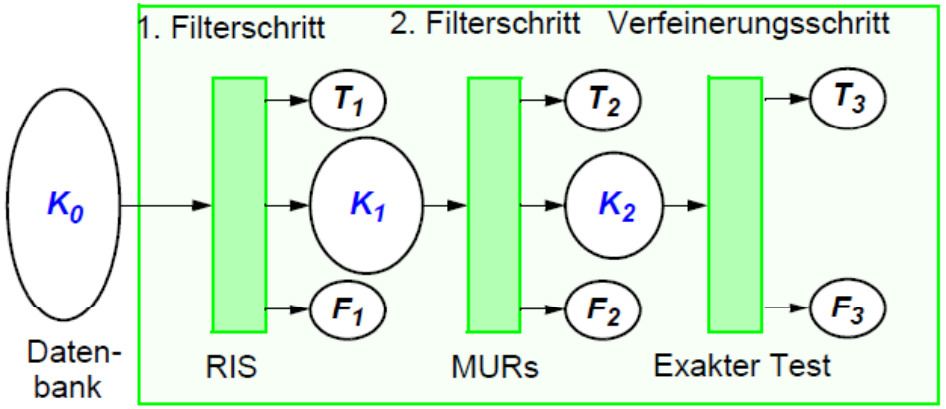
Überblick



- T_i Menge der in Schritt *i* identifizierten *Treffer* (Antworten)
- F_i Menge der in Schritt *i* ausgefilterten *Fehltreffer* (keine Antworten)
- K_i Menge der nach Schritt *i* verbliebenen *Kandidaten* (potentielle Antworten)
- $\$_i$ Kosten des Schritts *i* für einen *Kandidaten*
- Ziel: Minimierung der Gesamtkosten
  ⇒ Minimierung von K_i, d.h. Maximierung von T_i und F_i, mit möglichst geringem $\$_i$

5.2 Mehrstufige Anfragebearbeitung (II)

Konkretisierung



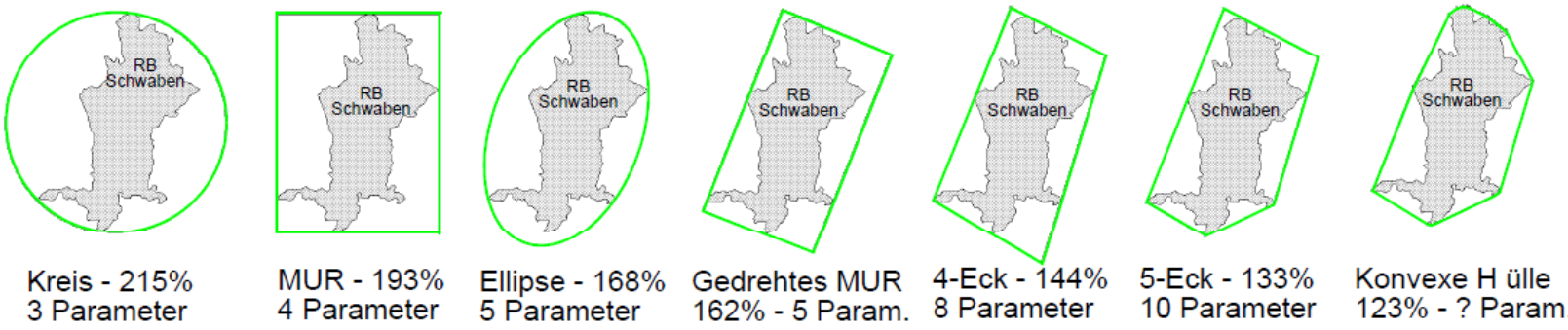
1. Bestimmung der Datenseiten, die Treffer und Kandidaten enthalten, durch RIS (z.B. R-Baum, ...)
2. Bestimmung der Objekte auf den gefundenen Datenseiten, die aufgrund ihrer MURs als Treffer in Frage kommen
3. Einlesen und Test der exakten Geometrie bezüglich der Anfragebedingung (z.B. Punkt-in-Polygon-Test für Point Query)

Weitere Verbesserungen

- Weitere Approximationen für zusätzliche Filterschritte
- Zerlegung der Geo-Objekte und exakter Test nur auf relevanten Komponenten

Konservative Approximationen

- enthalten das zu approximierende Objekt vollständig
- dienen insbesondere zur Bestimmung von Fehltreffern
  (Beispiel: ¬ (a.kons_appr ∩ b.kons_appr) ⇒ ¬ (a ∩ b))
- Vergleich verschiedener konservativer Approximationen



- (Es sind die durchschnittlichen Flächen der Approximationen in Prozent zur Objektfläche (=100%) angegeben (BKS 93))

  ⇒ 5-Eck: guter Kompromiß zwischen Genauigkeit und Speicherplatzbedarf

5.2 Approximationen (II)

Progressive Approximationen

- sind vollständig im zu approximierenden Objekt enthalten
- dienen insbesondere zur Bestimmung von Treffern (Beispiel: (a.prog_appr ∩ b.prog_appr) ⇒ (a ∩ b))
- Berechnung schwierig (insbesondere für maximale progressive Approximationen)