Kapitel 5: Räumliche Anfragebearbeitung

Skript zur Vorlesung
Geo-Informationssysteme
Wintersemester 2011/12
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]

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) =

$\{o \in DB \mid \forall o' \in DB: dist(point, o) \leq dist(point, o')\}$

DATABASE SYSTEMS GROUP

5.1 Räumliche Anfragen

LMU

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) =

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

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

5.1 Fensteranfragen

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;
```


DATABASE SYSTEMS GROUP

5.1 Fensteranfragen

LMU

Methode

window_index_query(IndexPage, Window, Predicate, Candidates)

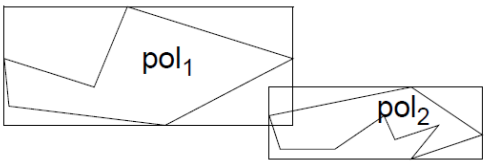
```
FOR ALL Entry ∈ Partitionen in IndexPage DO
    IF Predicate(Window, Entry.Rectangle) THEN
      IF Page = DataPage THEN
        Candidates := Candidates ∪ {Entry};
      ELSE
        window_index_query (Entry.Subtree^, Window, Predicate, Candidates);
```

5.1 Fensteranfragen

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

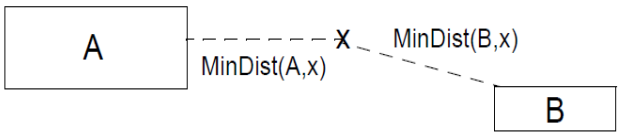
- SpatialIndex (R-Baum, Quadtree, etc.) → DB
- QueryPoint → Obj

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

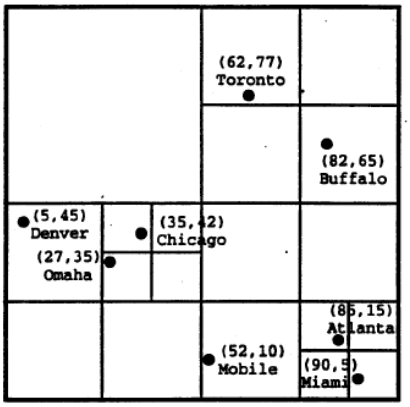
- MinDist(Partition,Point) = minimale Distanz, die theoretisch zwischen dem Anfragepunkt und einem Eintrag der Partition vorkommen kann.



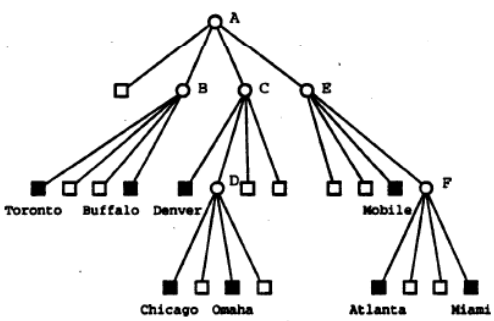
Algorithmus

```
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;
```


Beispiel

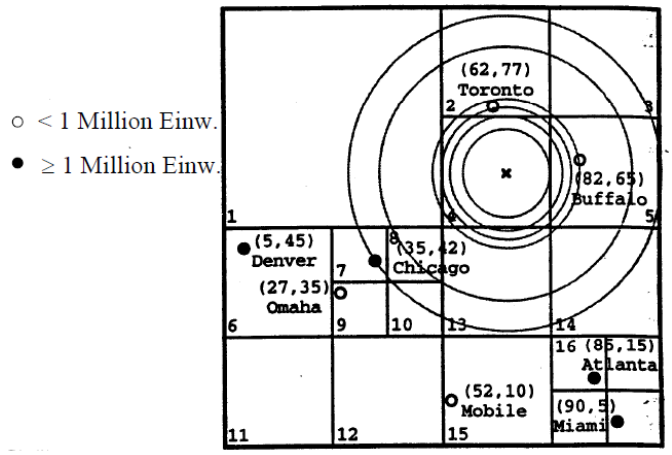


PR-Quadtree
mit Blattkapazität 1



5.1 Nächste-Nachbarn-Anfragen

Beispiel



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

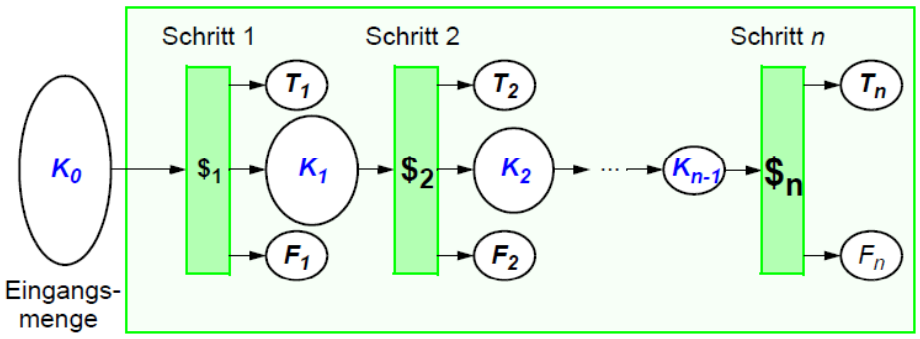
z.B. Wurzel = 0/1, Quadrant NE = ½

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)

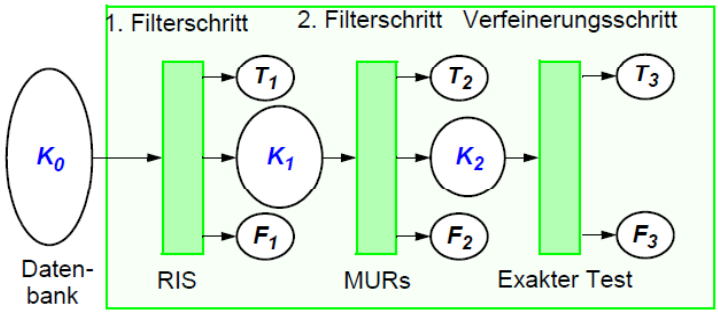
Überblick



- 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. 2Bestimmung 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)

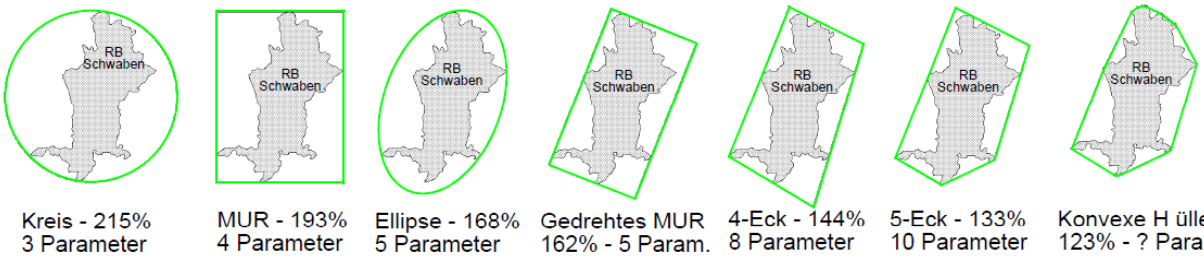
Weitere Verbesserungen

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

5.2 Approximationen (I)

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



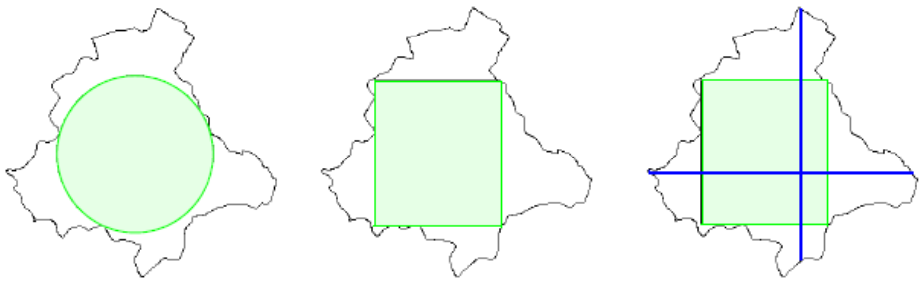
- (Es sind die durchschnittlichen Flächen der Approximationen in Prozent zur Objektfläche (=100%) angegeben (BKS 93))

  ⇒ 5-Eck: guter Kompromiß zwischen Genauigkeit und Speicherplatzbedarf

5.2 Approximationen (II)

Progressive Approximationen

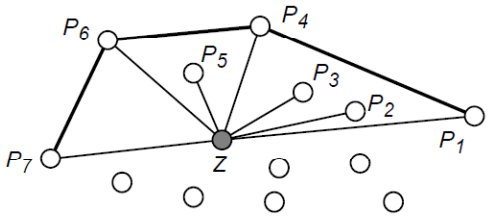
- sind vollständig im zu approximierenden Objekt enthalten
- dienen insbesondere zur Bestimmung von Treffern (Beispiel: (a.prog_appr ∩ b.prog_appr) ⇒ (a ∩ b))
- Berechnung schwierig (insbesondere für maximale progressive Approximationen)



5.2 Berechnung der konvexen Hülle

Fächermethode von Graham (1972)

- Bestimme einen Zentrumspunkt Z
- Sortiere sämtliche Punkte P_i nach aufsteigendem Winkel bezüglich Z
- Durchlaufe die Punkte P_i gegen den Uhrzeigersinn
- Betrachte dabei immer aufeinanderfolgende Punkte P_k, P_{k+1} *und* P_{k+2}*:*
  - 1. Fall: P_{k+2} *erzwingt Linksdrehung: Weiterlaufen*
  - 2. Fall: P_{k+2} *erzwingt Rechtsdrehung:*
    - Lösche P_{k+1}
    - Betrachte wiederholt P_{k-j}, P_{k+1-j} *und* P_{k+2}, *bis kein Punkt mehr* gelöscht wird



⇒ Die konvexe Hülle dient als Grundlage zur Berechnung des minimal umgebenden n-Ecks und des gedrehten MURs

5.2 Strukturelle Zerlegungsrepräsentation (I)

- Das Rastermodell kann als flächiges Modell bezeichnet werden
  - Flächen werden durch flächige Basiselemente (explizit) dargestellt
  - Basisoperationen: einfach und unabhängig voneinander ausführbar
- Das Vektormodell kann als Linienmodell bezeichnet werden
  - Flächen werden durch Beschreibung des Randes (implizit) dargestellt
  - Geringer Speicherplatzbedarf
- *Strukturelle Zerlegungsrepräsentation* ist eine flächige Vektorrepräsentation
  - Zerlegung in Basiskomponenten
    - vollständig
    - disjunkt
  - Basiskomponenten sind im Vektormodell beschrieben
  - Explizite Repräsentation von Flächen
  - Basisoperationen unabhängig voneinander ausführbar
  - Geringer Speicherplatzbedarf

5.2 Strukt. Zerlegungsrepräsentation (II)

Motivation

- Beobachtung: komplexe und zeitaufwendige Algorithmen dominieren die Anfragebearbeitung
- Prinzip: Divide-and-Conquer
  Zerlegung in einfache Komponenten
- Prinzip: Einsatz geeigneter geometrischer Datenstrukturen
- Prinzip: Vorverarbeitung

  Investition von Zeit und Speicherplatz bei der Repräsentation

  ⇒ höherer Aufwand zum Einfügezeitpunkt

  ⇒ geringerer Aufwand bei der Anfrage

Vorteile

+ Vereinfachung der algorithmischen Komplexität von Anfragen und Operationen

+ Lokalität von Anfragen und Operationen kann ausgenutzt werden

5.2 Strukt. Zerlegungsrepräsentation (III)

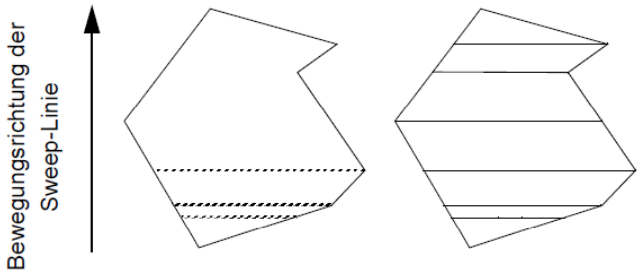
Charakterisierung verschiedener Zerlegungen

- qualitative Aspekte
  - Welche Typen von Komponenten werden erzeugt?
  - Ist die Menge der Komponenten homogen oder heterogen?
  - Ist die Zerlegung eindeutig?
  - Ist die Beschreibungslänge der Komponenten fest oder variabel?
  - Erfüllt die Zerlegung ein Gütekriterium?
  - Wie geeignet ist die Zerlegung für eine Verwaltung durch eine geometrische Datenstruktur?
- quantitative Aspekte
  - Anzahl der erzeugten Komponenten
  - Speicherplatzbedarf der Zerlegungsrepräsentation
  - Zeitkomplexität des Zerlegungsalgorithmus

5.2 Trapezzerlegung

- Zerlegung eines Polygons in (achsenparallele) Trapeze (Asano+Asano 1983)

- Berechnung durch Plane-Sweep-Algorithmus: O(n log n) für n Eckpunkte



- Speicherplatzaufwand
  - Anzahl der Komponenten = *n bei n Eckpunkten*
  - aber Vervielfachung des Speicherplatzes da nun Trapeze statt Punkten verwaltet, d.h. abgespeichert werden

5.2 Verwaltung der Komponenten (I)

Beobachtung

- Operationen auf Zerlegungskomponenten sind einfach
- Anzahl der Zerlegungskomponenten: O(n)

⇒ Kein Gewinn, falls alle Komponenten getestet werden
⇒ Einsatz von Datenstrukturen zur Auswahl "relevanter" Komponenten
⇒ Einsatz von räumlichen Indexstrukturen (RIS)

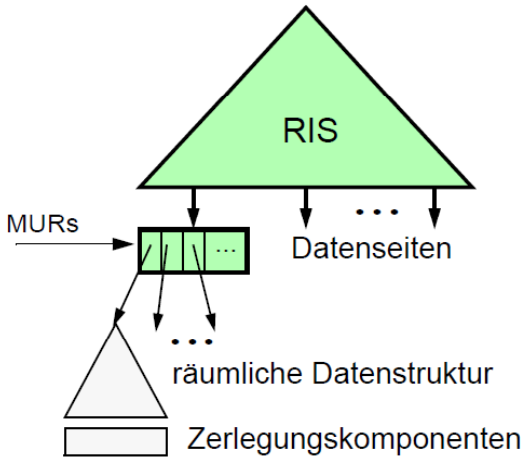
1. Ansatz

- *Eine* RIS verwaltet Zerlegungskomponenten *aller* Objekte
  - Redundanz bei der Anfragebearbeitung (betrifft insbesondere größere Window Queries)

5.2 Verwaltung der Komponenten (II)

2. Ansatz

- Eine RIS verwaltet die Objektapproximationen (MUR) aller Objekte
- Für jedes Objekt verwaltet eine separate räumliche Datenstruktur die Zerlegungskomponenten dieses Objektes
- Wenn die exakte Objektgeometrie untersucht werden muß, werden die Zerlegungskomponenten einschließlich der zugehörigen räumlichen Datenstruktur vom Sekundärspeicher in den Hauptspeicher eingelesen



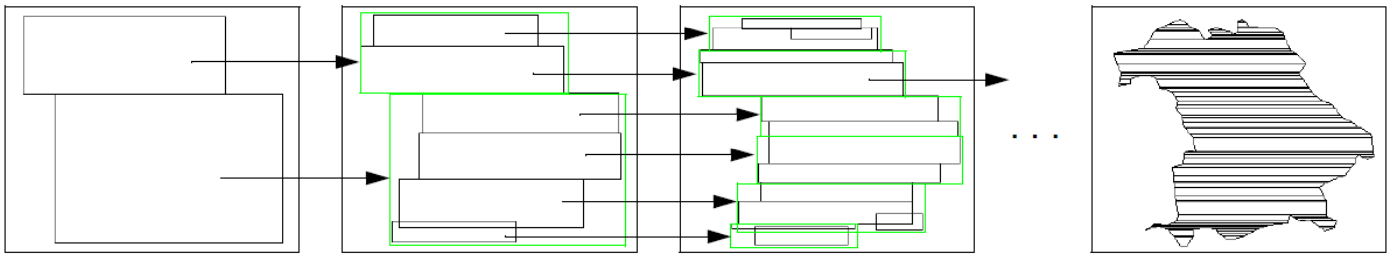
5.2 Verwaltung der Komponenten (III)

TR-Ansatz (Ausprägung des 2. Ansatzes)

- Verwende R-Baum zur Verwaltung der Zerlegungskomponenten

⇒ Anpassung des R-Baums an die neuen Anforderungen *(TR-Baum):*

- TR-Baum soll für den Hauptspeicher ausgelegt werden
  - möglichst kleine Knotengröße
- TR-Baum soll möglichst schnell in den Hauptspeicher eingelesen werden
  - kompakte Speicherung auf dem Plattenspeicher
  - kein dynamischer Aufbau, keine Adressneuberechnungen
- TR-Baum sollte möglichst kompakt gespeichert sein



5.2 Verwaltung der Komponenten (IV)

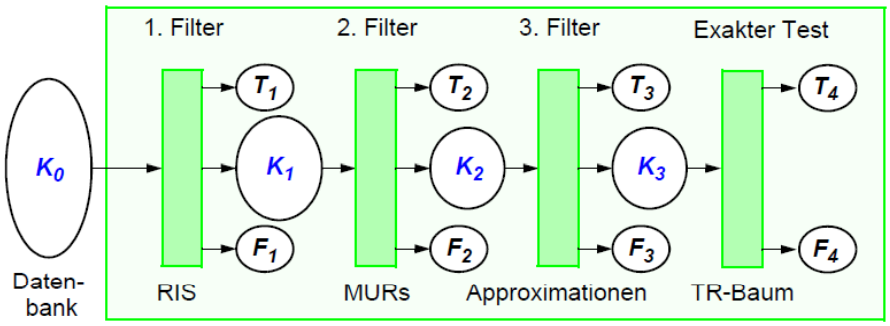
Eigenschaften des TR-Ansatzes

+ sehr schnelle Bearbeitung geometrischer Operationen (z.B. Punkt-In-Polygon-Test)

- erheblich höherer Speicherplatzbedarf (und damit höhere Übertragungskosten beim Einlesen der exakten Geometrie)

5.2 Mehrstufige Anfragebearbeitung (III)

Zusammenfassung



1. Bestimmung der Datenseiten, die Treffer und Kandidaten enthalten, durch RIS
2. Bestimmung der Objekte auf den gefundenen Datenseiten, die aufgrund ihrer MURs als Treffer in Frage kommen
3. Bestimmung weiterer Treffer oder Fehltreffer durch zusätzliche konservative und progressive Approximationen
4. Einlesen der exakten Geometrie und Test der exakten Geometrie bezüglich der Anfragebedingung mit Hilfe des TR-Baum-Ansatzes

⇒ *GENESYS: Prototyp-System, das diese Anfragebearbeitung realisiert*