

## 1. Hochdimensionale Indexstrukturen

- (a) Anfragearten
- (b) Baumverfahren
- (c) Komplexe Distanzfunktionen
- (d) Fluch der hohen Dimensionen
- (e) Signaturverfahren
- (f) Weitere Indexverfahren

## 7 Effiziente Algorithmen und Datenstrukturen (2) 2. Algorithmen zur Aggregation von Ähnlichkeitswerten

- (a) Container-Algorithmen
- (b) Kondensator-Algorithmus
- (c) Indexaggregation

## Einführung

Algorithmen und Datenstrukturen für  
effiziente Ergebnisberechnung bzgl. der Anfrage

## Einführung

Algorithmen und Datenstrukturen für  
effiziente Ergebnisberechnung bzgl. der Anfrage

Erweiterung von Verfahren klassischer DBMS um  
Behandlung von Ähnlichkeits- bzw.  
Unähnlichkeitswerten  
→ Übergang von Mengensemantik zu  
Listensemantik

- ♦ hochdimensionale Indexstrukturen zur effizienten  
Suche im hochdimensionalen Raum
- ♦ Aggregation von Ähnlichkeitswerten:  
für komplexe Anfragen

Strukturierung der Daten zur Unterstützung einer effizienten Suche

klassische Datenstrukturen in DBMS:

B-Baum und dessen Varianten

- ♦ exakte Suche mit logarithmischem Aufwand
- ♦ aber Einschränkung auf eine Dimension  
→ ungeeignet zur Ähnlichkeitssuche im hochdimensionalen Raum

## Anforderungen an hochdimensionale Datenstruktur und Algorithmen

Korrektheit und Vollständigkeit

skalierbar bzgl. Dimensionsanzahl

räumliche Ausdehnung der Objekte:

- ♦ 0 Dimensionen: Punkt
- ♦ 1 Dimension: Linie
- ♦ 2 Dimensionen: Fläche
- ♦ n Dimensionen: etwa Hyperwürfel

## Anforderungen an hochdimensionale Datenstruktur und Algorithmen (2)

Sucheffizienz, also Anzahl Seitenzugriffe muss besser als bei sequentiell durchlauf sein

viele Anfragearten (siehe nächste Folie)

effiziente Update-Operationen

verschiedene Distanzfunktionen

speicherplatzsparend

## Anfragearten

- ♦ Nächste-Nachbarsuche
- ♦ Approximative Nächste-Nachbarsuche
- ♦ Reverse-Nächste-Nachbarsuche
- ♦ Bereichssuche
- ♦ Punktsuche
- ♦ Partial-Match-Suche
- ♦ Ähnlichkeitsverbund

Feature-Daten eines Anfrageobjekts:  $o_q$

Menge von Feature-Daten:  $FO$

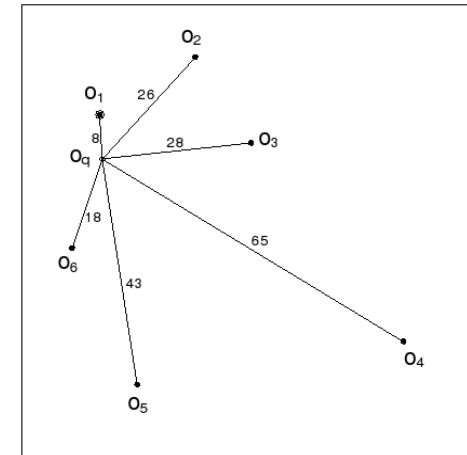
binäre Distanzfunktion  $d()$

Finden des ähnlichsten Medienobjekts (das nächste Feature-Objekt)

mehrere nächste Nachbarn möglich:

$$nn(o_q) \subseteq FO \text{ mit } \forall o_i \in FO : \forall o \in nn(o_q) : d(o_q, o) \leq d(o_q, o_i)$$

Nächste-Nachbarsuche graphisch



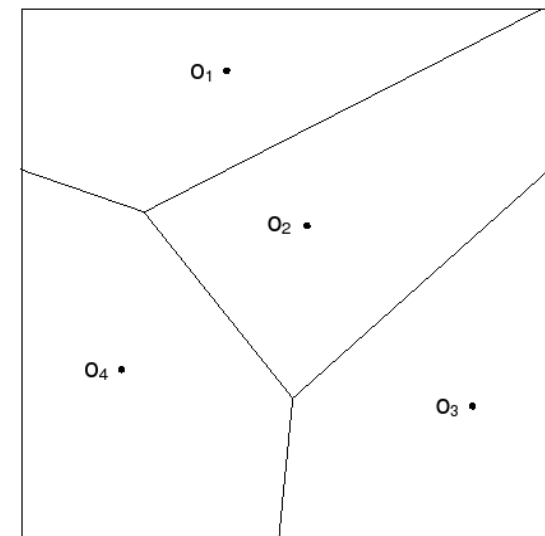
NN-Suche auf punktförmigen Feature-Daten ist äquivalent zum Enthaltenseinstest in Voronoi-Zelle

Idee: Vorausberechnung aller Voronoi-Zellen und anschließend Enthaltenseinstest

jedem Feature-Objekt ist eigene Voronoi-Zelle zugewiesen

Problem: Berechnungskomplexität für Enthaltenseinstest

Voronoi-Zelle enthält alle Raumpunkte, die nächste Nachbarn des entsprechenden Feature-Objekts sind



die  $k$  nächsten Nachbarn werden gesucht

$knn(o_q) \subseteq FO$  mit  $|knn(o_q)| = k$  und  $\forall o \in FO \setminus knn(o_q) :$

$\forall o_{knn} \in knn(o_q) : d(o_q, o) \geq d(o_q, o_{knn})$

bei gleichen Distanzen: nichtdeterministische Auswahl  
Ergebnisobjekte werden aufsteigend ausgegeben

$k$  ist üblicherweise so klein, dass das Ergebnis in den  
Hauptspeicher passt  
→ Hauptspeichersortierung

ansonsten:

Ergebnisobjekt sukzessive abholen  
(getNext-Semantik / ranking-Anfrage)

Effizienzgewinn bzgl. NN-Anfragen, wenn kleine  
Ungenauigkeiten tolerierbar  
 $\epsilon$  als Maß der Ungenauigkeit

$ann(o_q) = o \in FO$  wenn  $d(o_q, o) \leq (1 + \epsilon) \cdot d(o_q, nn(o_q))$

mehrere Feature-Objekte können Bedingung erfüllen  
→ nichtdeterministische Auswahl

Vorsicht:  $ann$ -Ergebnis muss nicht in Nähe des  
 $nn$ -Ergebnisses liegen

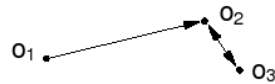
(probably approximately correct)  
weitere Abschwächung einer ANN-Suche

Forderung: Wahrscheinlichkeit der Abweichung von  
 $ann$ -Bedingung muss vorgebbaren Mindestwert überschreiten  
ermöglichte effizientere Suche

Suche nach Feature-Objekten, deren nächster Nachbar der Anfragepunkt ist (etwa Suche nach bestem Ort für neuen Einkaufsmarkt)

Achtung: Ergebnis oft anders als bei NN-Suche, da Nächste-Nachbarrelation nicht symmetrisch ist

$$rnn(o_q) = \{o \in FO \mid o_q \in nn(o)\}$$



Anfrage definiert einen Bereich (Region) im hochdimensionalen Raum

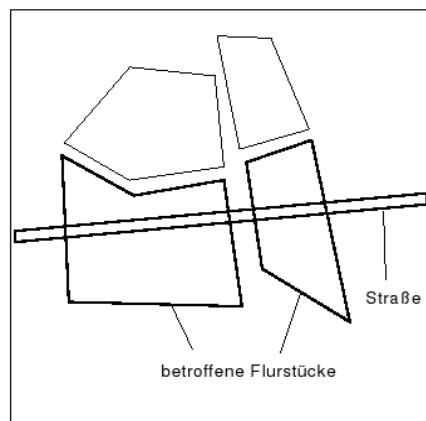
Ergebnis sind alle Feature-Objekte, die Anfragebereich schneiden

$$range(o_q) = \{o \in FO \mid o \cap o_q \neq \emptyset\}$$

Varianten

- ◆ begrenzte versus unbegrenzte Bereiche
- ◆ Spezialfall Hyperkugeln und Hyperrechteck

Straßenplanung im Katasteramt:



Suche anhand eines gegebenen Feature-Objekts  $o_q$

Test auf Enthaltensein  
(exakte Überdeckung)

Punktsuche in MMDB ist relativ selten

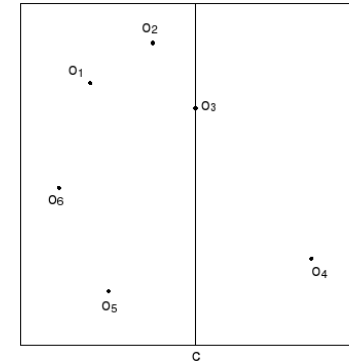
$$punkt(o_q) = \begin{cases} \text{wahr} & : \exists o \in FO : o_q = o \\ \text{falsch} & : \text{sonst} \end{cases}$$

Punktsuche kann als Complete-Match-Suche aufgefasst werden

bei Partial-Match-Suche Übereinstimmung nur in einigen Dimensionen (restliche Dimensionen werden ignoriert)

ist Spezialfall der Bereichsanfrage mit teilweise unbegrenztem Bereich

Suchbereich ist senkrechte Linie (Übereinstimmung in nur einer Dimension)



Operation auf zwei Mengen von Feature-Objekten  
Verbund findet Paare, deren Distanz kleiner als vorgegebener Schwellenwert  $\epsilon$  ist

Selbstverbund: dieselbe Menge zweimal

$$simjoin(FO_1, FO_2, \epsilon) = \{(o_1, o_2) | o_1 \in FO_1 \wedge o_2 \in FO_2 \wedge d(o_1, o_2) \leq \epsilon\}$$

