Skript zur Vorlesung

Knowledge Discovery in Databases

im Wintersemester 2007/2008

Kapitel 8: DB-Techniken zur Leistungssteigerung

http://www.dbs.ifi.lmu.de/Lehre/KDD

8. DB-Techniken zur Leistungssteigerung

Bisher: (meist) kleine Datenmengen hauptspeicherresident

Jetzt:

- sehr große Datenmengen, die nicht in den Hauptspeicher passen
- Daten auf Sekundärspeicher => Zugriffe viel teurer als im Hauptspeicher
- effiziente Algorithmen erforderlich, d.h. Laufzeitaufwand höchstens O(n log n)

Übersicht über das Kapitel

➡ Hauptaugenmerk auf Clustering Algorithmen

8.1 Datenkompression

Idee:

- DM-Algorithmus auf der gesamten Datenmenge zu teuer
- Komprimiere Daten, so dass sie in den Hauptspeicher passen
- Wende DM-Algorithmus auf komprimierte Daten an

Techniken:

- Sampling (8.1.1)
  Datenbank wird auf eine Stichprobe reduziert

- Micro-Clustering (8.1.2)
  bilde Micro-Cluster, die Teilmengen der Daten möglichst genau repräsentieren; Datenbank wird auf Micro-Cluster reduziert

8.1.1 Sampling

Indexbasiertes Sampling [Ester, Kriegel & Xu 1995]

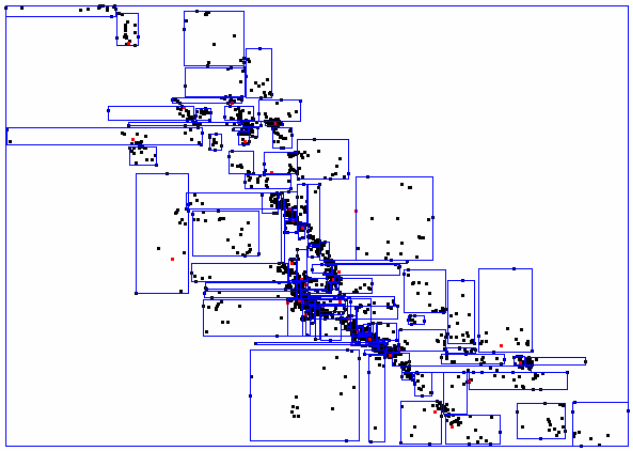
Zufälliges Sampling liefert u.U. schlechte Qualität

Verwendung von räumlichen Indexstrukturen oder verwandten Techniken zur Auswahl des Samplings

- Indexstrukturen liefern ein grobes Vor-Clustering
  räumlich benachbarte Objekte werden möglichst auf der gleichen Seite abgespeichert
- Indexstrukturen sind effizient
  da nur einfache Heuristiken zum Clustering
- schnelle Zugriffsmethoden für verschiedene Ähnlichkeitsanfragen
  z.B. Bereichsanfragen und *k*-Nächste-Nachbarn-Anfragen

Methode

- Aufbau eines R-Baums
- Auswahl von Repräsentanten von den Datenseiten des R-Baums
- Anwendung des Clustering-Verfahrens auf die Repräsentantenmenge
- Übertragung des Clustering auf die gesamte Datenbank



Datenseitenstruktur eines R*-Baums

Auswahl von Repräsentanten

Wieviele Objekte sollen von jeder Datenseite ausgewählt werden?

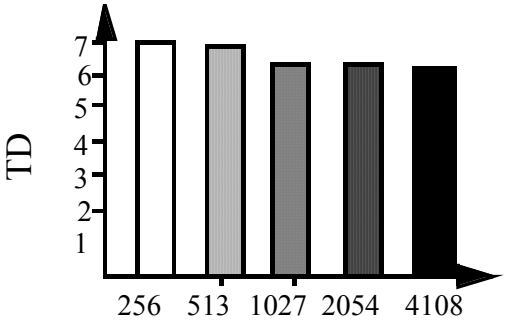
- hängt vom verwendeten Clusteringverfahren ab
- hängt von der Verteilung der Daten ab
- z.B. für CLARANS: ein Objekt pro Datenseite

guter Kompromiss zwischen der Qualität des Clusterings und der Laufzeit

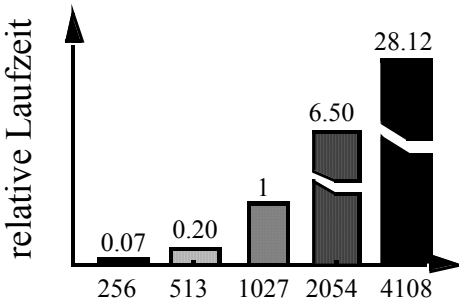
Welche Objekte sollen ausgewählt werden?

- hängt ebenfalls vom Clusteringverfahren und von der Verteilung der Daten ab
- einfache Heuristik: wähle das „zentralste“ Objekt auf der Datenseite

Experimentelle Untersuchung für CLARANS



Anzahl der Repräsentanten



Anzahl der Repräsentanten

- Laufzeit von CLARANS ist etwa $O(n^2)$
- Qualität des Clusterings steigt bei mehr als 1024 Repräsentanten kaum noch

⇒ 1024 Repräsentanten guter Kompromiss zwischen Qualität und Effizienz

BIRCH [Zhang, Ramakrishnan & Linvy 1996]

Methode

- Bildung kompakter Beschreibungen von Teil-Clustern (Clustering Features)
- hierarchische Organisation der Clustering Features
  in einem höhenbalancierten Baum (CF-Baum)
- Anwendung eines Clusteringverfahren wie z.B. CLARANS
  auf die Clustering Features in den Blättern des Baums

CF-Baum

- komprimierte, hierarchische Repräsentation der Daten
- berücksichtigt die Clusterstruktur

Grundbegriffe

Clustering Feature einer Menge C von Punkten X_i: $CF = (N, LS, SS)$

$N = |C|$ „Anzahl der Punkte in C“

$LS = \sum_{i=1}^{N} X_i$ „lineare Summe der N Datenpunkte“

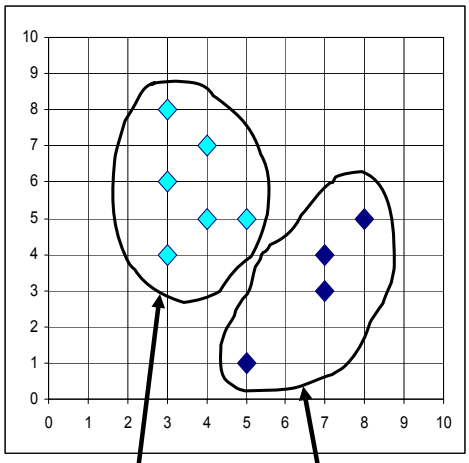
$SS = \sum_{i=1}^{N} X_i^2$ „Quadratsumme der N Datenpunkte“

aus den CF‘s können berechnet werden

- Centroid (Repräsentant)
- Kompaktheitsmaße
- und Distanzmaße für Cluster

Beispiel

(3,4)
(4,5)
(5,5)
(3,6)
(4,7)
(3,8)



(5,1)
(7,3)
(7,4)
(8,5)

$CF_1 = (6, (22,35),(84,215))$

$CF_2 = (4, (27,13),(187,51))$

Grundbegriffe

- Additivitätstheorem

  für CF-Vektoren für zwei disjunkte Cluster C_1 und C_2 gilt

  $$\mathrm{CF}(C_1 \cup C_2) = CF(C_1) + CF(C_2) = (N_1 + N_2,\ LS_1 + LS_2,\ QS_1 + QS_2)$$

  d.h. CF's können inkrementell berechnet werden

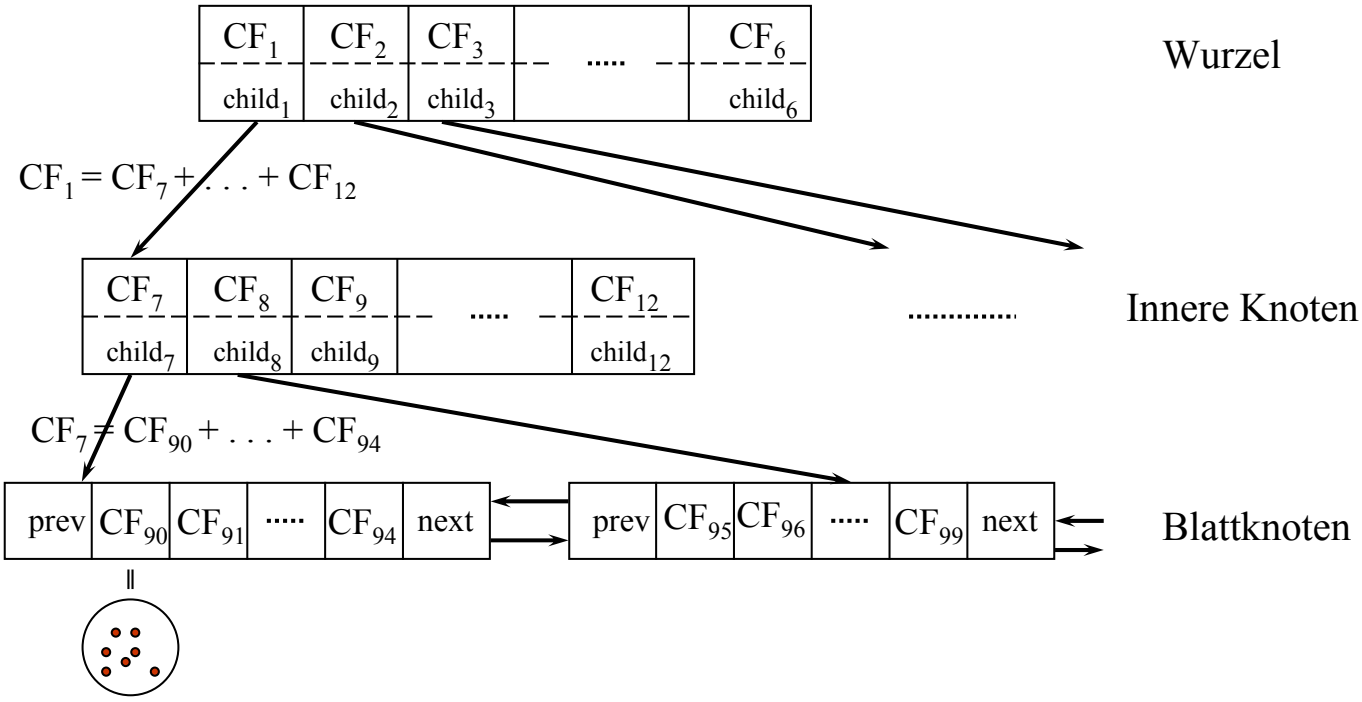
- Definition

  Ein *CF-Baum* ist ein höhenbalancierter Baum zur Abspeicherung von CF's.

- Eigenschaften eines CF-Baums
  - Jeder innere Knoten enthält höchstens B Einträge der Form $[CF_i, child_i]$ und CF_i ist der CF-Vektor des Subclusters des i-ten Sohnknotens.
  - Ein Blattknoten enthält höchstens L Einträge der Form $[CF_i]$.
  - Jeder Blattknoten besitzt zwei Zeiger *prev* und *next*.
  - Für jeden Eintrag eines Blattknotens ist der Durchmesser kleiner als T.
- Aufbau eines CF-Baums (analog B^+-Baum)
  - Transformation eines Datensatzes p in einen CF-Vektor $CF_p=(1, p, p^2)$
  - Einfügen von CF_p in den Teilbaum des CF-Vektors mit kleinster Distanz
  - bei Verletzung des Schwellwerts T wird ein neuer Eintrag CF_p eingefügt, sonst absorbiert der nächste Eintrag im Blatt CF_p
  - bei Verletzung des Schwellenwerts B oder L wird Knoten gesplittet:
    - die entferntesten CF's bilden die beiden neuen Knoten
    - die restlichen CF's werden dem neuen Knoten mit geringster Distanz zugeordnet

Beispiel

B = 7, L = 5



Verfahren

Phase 1

- ein Scan über die gesamte Datenbank
- Aufbau eines CF-Baums B_1 bzgl. T_1 durch sukzessives Einfügen der Datensätze

Phase 2

- falls der CF-Baum B_1 noch zu groß ist, wähle ein $T_2 > T_1$
- Aufbau eines CF-Baums B_2 bzgl. T_2 durch Einfügen der CF‘s der Blätter von B_1

Phase 3

- Anwendung eines Clusteringalgorithmus auf die Blatteinträge des CF-Baums
- Clusteringalgorithmus muß evtl. an Clustering Features angepaßt werden

Diskussion

+ Komprimierungsfaktor frei wählbar

+ Effizienz

  Aufbau eines sekundärspeicherresidenten CF-Baums: O(n log n)

  Aufbau eines hauptspeicherresidenten CF-Baums: O(n)

zusätzlich: Aufwand des Clusteringalgorithmus
(wenn CF-Baum im Hauptspeicher, ist dieser Aufwand vernachlässigbar)

- nur für numerische Daten
  euklidischer Vektorraum

- abhängig von der Reihenfolge der Daten

BIRCH*

[Ganti,Ramakrishnan,Gehrke,Powell,French 99]

- Erweiterung auf metrische Daten
- CF* = (n, C, r)
  - n Anzahl Objekte o, die von CF* repräsentiert werden ($o \in$ CF*)
  - C „Clusteroid“ der Objekte $o \in$ CF*
    C ist das Objekt, das die paarweise quadrierten Distanzen zu allen $o \in$ CF* minimiert

$$C = \min_{o \in CF^*} \sum_{o_i \in CF^*} dist(o, o_i)$$

    Clusteroid kann bei Einfügen/Löschen mit Hilfe von Heuristiken sehr effizient inkrementell berechnet werden
  - r „Radius“ der Objekte, die von CF* repräsentiert werden

$$r = \sqrt{\frac{\sum_{o_i \in CF^*} dist^2(C, o_i)}{n}}$$

- Inkrementelle Neuberechnung der Features von CF*=(n,C,r) beim Einfügen eines neuen Objektes u
- Neuberechnung des Clusteroid
  - Problem: CF* enthält viele Objekte (evtl. zu groß für den Hauptspeicher) und Distanzberechnungen sind teuer
  - Sei $RowSum(o) = \sum_{o_i \in CF^*} dist(o, o_i)$
  - Halte die p Objekte aus CF* mit der kleinsten *RowSum* im Hauptspeicher (wenn CF* weniger als p Objekte enthält, sind alle $o \in$ CF* im Hauptspeicher)
  - *RowSum*(u) kann wie folgt approximiert werden

$$RowSum(u) \approx n \cdot r^2 + n \cdot dist^2(C, u)$$

    Dabei wird nur der Radius und der Clusteroid von CF* benötigt (keine teurern Distanzberechnungen mit anderen Objekten aus CF*, kein DB-Scan)
  - Wenn *RowSum*(u) kleiner als die größte *RowSum* der p Objekte von CF* im Hauptspeicher, lösche das Objekt mit der höchsten *RowSum* und füge u ein
  - Wenn u die kleinste *RowSum* insgesamt hat, ist u der neue Clusteroid

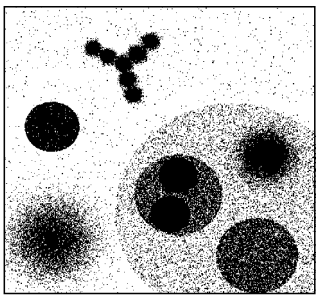
BIRCH*

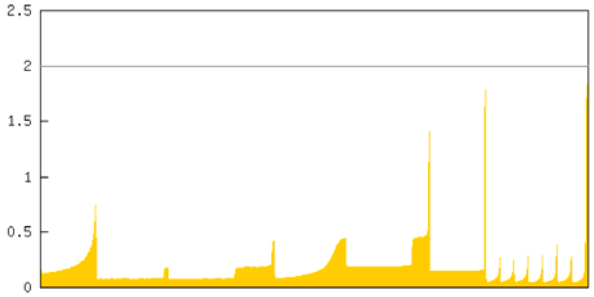
- Verfahren analog zu BIRCH auf euklidischen Daten, d.h.
  - Vorclusterung durch Aufbau des CF*-Baumes
  - optionales Clustering mit einem beliebigen Clusterverfahren der Blattknoten des CF*-Baumes
- Basiert auf Heuristiken beim CF*-Baum-Aufbau (Neuberechnung des Clusteroids, der *RowSum*-Werte, etc.)
- Ähnliche Eigenschaften wie BIRCH, insbesondere Reihenfolgeabhängig

Data Bubbles [Breunig, Kriegel, Kröger, Sander 2001]

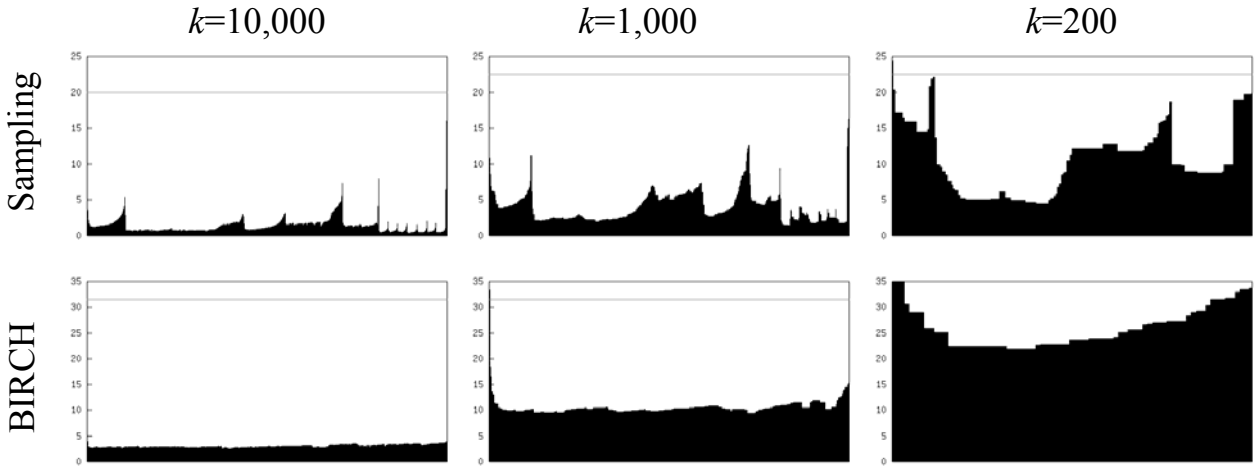
Original DB und OPTICS-Plot

1 Mio. Datenpunkte





OPTICS-Plots auf Komprimierten Daten

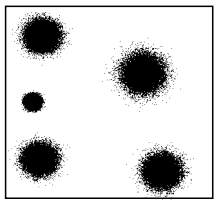


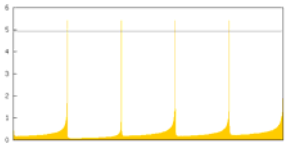
Drei Hauptprobleme bei BIRCH und Sampling für hierarchisches Clustering:

1. Verlorengegangene Objekte (Lost Objects)
   - Viele Objekte der DB fehlen im Plot/Dendrogram
2. Größenverzerrung (Size Distortions)
   - Cluster sind gequetscht und gestreckt
3. Strukturelle Verzerrung (Structural Distortions)
   - Hierarchische Cluster Struktur ist zerstört

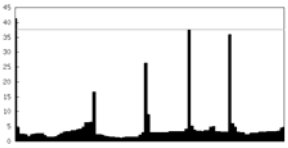
Lösungen

- Post-Processing für Problem 1 und 2 (Lost Objects, Size Distortions)
  - nn-Klassifikation, ersetze repräsentative Objekte durch die Menge der repräsentierten Objekte
- Data Bubbles für Problem 3 (Structural Distortions)





OPTICS original

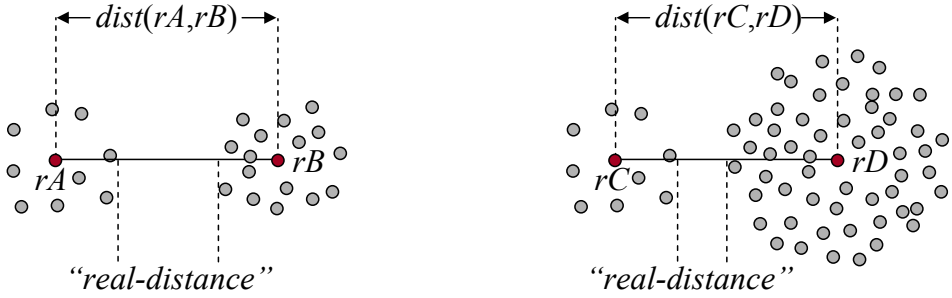


Sampling 100 Obj.

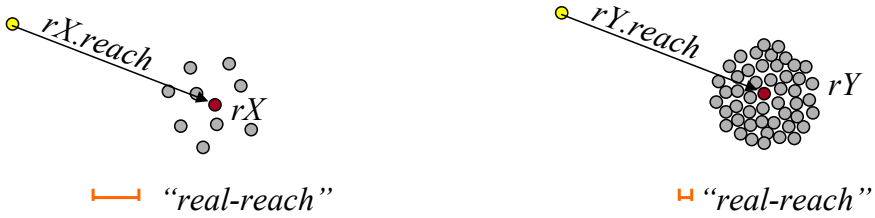
CF 100 Obj.

Gründe für strukturelle Verzerrungen:

- Distanz zwischen den Originalobjekten wird schlecht durch die Distanz zwischen Repräsentanten approximiert



- Erreichbarkeitsdistanz die den Repräsentanten zugeordnet werden, approximieren die Erreichbarkeitsdistanz der repräsentierten Objekte sehr schlecht



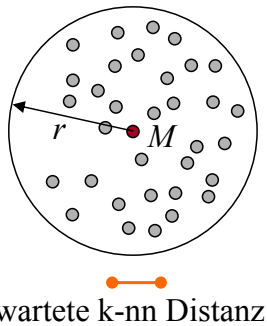
Data Bubble: reichere Abstraktion

Definition: **Data Bubble**

- Data Bubble $B=(n, M, r)$ für eine Menge von n Objekten $X=\{X_i\}$

$$M = \left(\sum_{i=1}^{n} X_i \right) / n$$ ist der *Mittelpunkt* von X und

$$r = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(X_i - X_j)^2}{n \cdot (n-1)}}$$ ist der *Radius* von X.

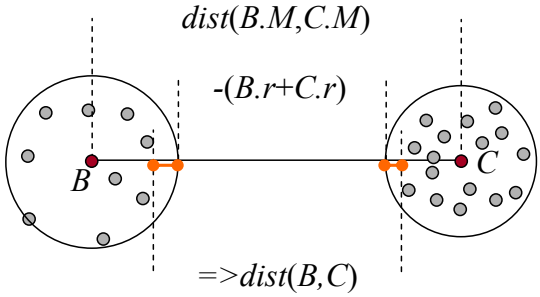


- Erwartete k-nn Distanz der Objekte X_i im Data Bubble (bei Gleichverteilung)

$$nnDist(k, B) = r \cdot \left(\frac{k}{n} \right)^d$$

- Data Bubbles können entweder aus einem Sample oder aus CFs berechnet werden

Definition: Distanz zwischen Data Bubbles



Definition: Kern- und Erreichbarkeitsdistanz für Data Bubbles

- analog zur Kern- und Erreichbarkeitsdistanz von Punkten

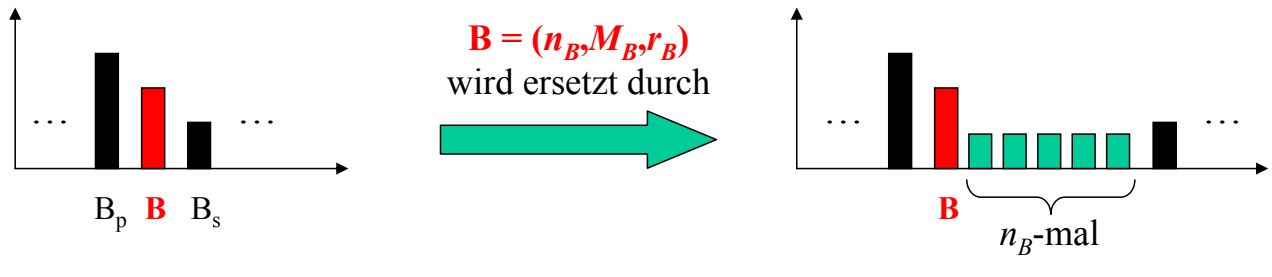
Definition: virtuelle Erreichbarkeitsdistanz eines Data Bubble

- Erwartete *k*-nn-Distanz innerhalb des Data Bubble
- bessere Approximation der Erreichbarkeitsdistanz der repräsentierten Objekte

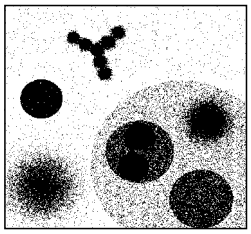
Clustering mit Data Bubbles:

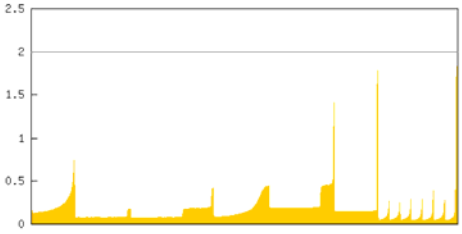
- Generiere *m* Data Bubbles aus *m* Sampleobjekten oder CF-Features
- Cluster die Menge der Data Bubbles mit OPTICS
- Generiere das Erreichbarkeitsdiagramm:

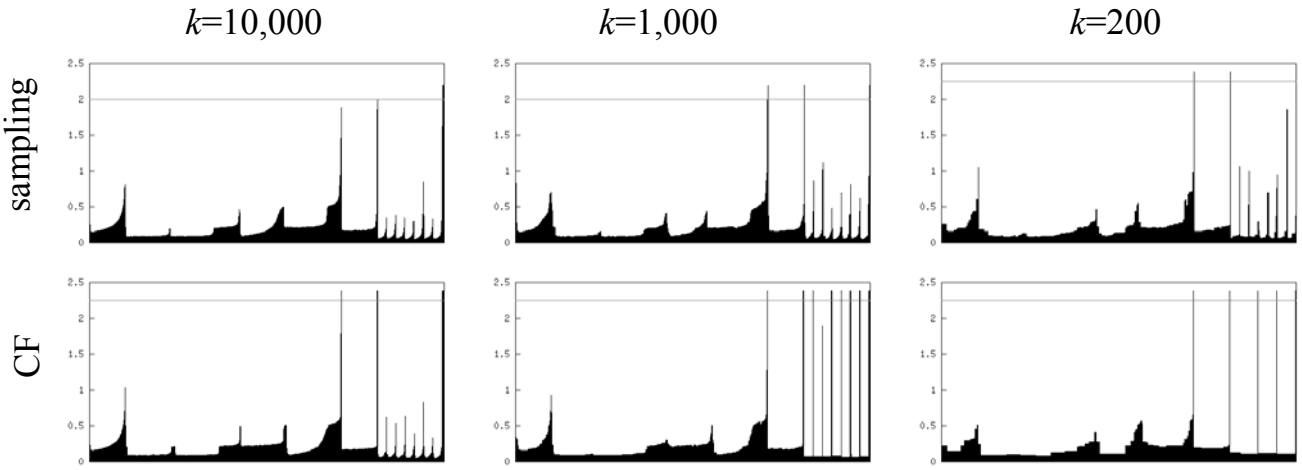
  Für jedes Data Bubble *B*:
  - „Plotte“ die Erreichbarkeitsdistanz *B.reach* (vom OPTICS-Lauf auf den Data Bubbles erzeugt)
  - „Plotte“ alle Punkte, die von *B* repräsentiert werden mit der virtuellen Erreichbarkeitsdistanz von *B*



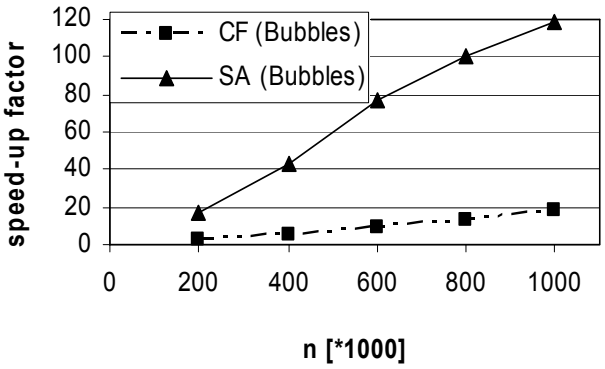
Ergebnisse (Kompression auf *k* Objekte)



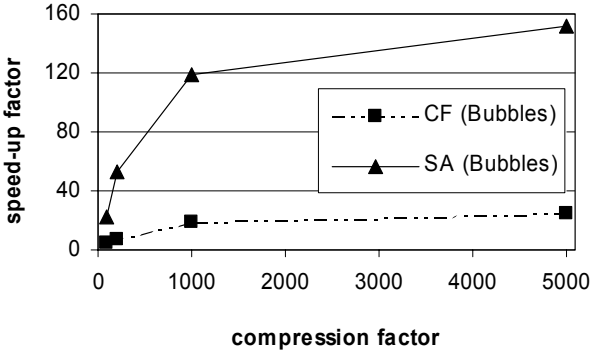




Speed-Up-Faktoren



Speed-Up-Faktoren



bzgl. Datenbank Größe bzgl. Kompressions Faktor

für Datenbank mit (1 Mio. Objekte)

Diskussion:

- Löst die Probleme Lost Objects, Size Distortions und Structual Distortions
- Data Bubbles aus zufällig gewählten Reerenzobjekten (Sample) erzielt exprimentell bessere Ergebnisse als mit BIRCH erzeugte Data Bubbles
- Nur auf Feature-Räume anwendbar

Erweiterung für allgemein metrische Objekte [Zhou, Sander 2003]

- Repräsentant *M*: Medoid statt Mittelwert
- Distanz zwischen Data Bubbles deutlich komplizierter, da die Berechnung der paarweisen *k*-nn-Distanzen komplex ist

GRID-Clustering [Schikuta 1996]

Idee: Grob-Clustering durch räumliche Indexstruktur

das Volumen des durch eine Datenseite repräsentierten Datenraums ist um so kleiner, je höher die Punktdichte in diesem Gebiet des Raums ist

Nachbearbeitung durch Verschmelzen von Seitenregionen

Seitenregionen mit hoher Punktdichte werden als Clusterzentren angesehen und rekursiv mit benachbarten, weniger dichten Seitenregionen verschmolzen

⇨ Micro-Cluster ≡ Datenseite der Indexstruktur

Verwendete Indexstruktur

Gridfile

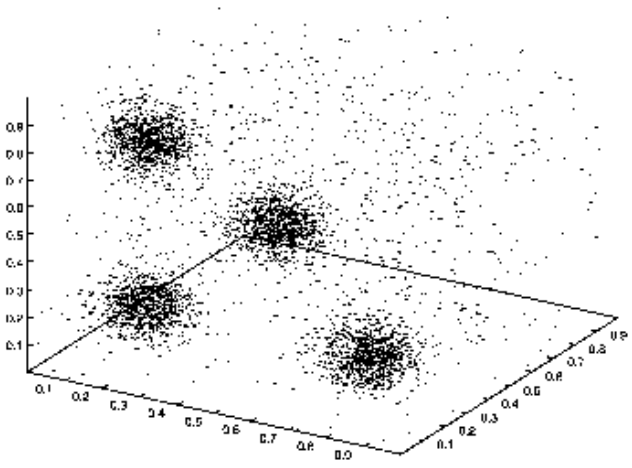
Methode

- beginne mit der Datenseite *S*, die die höchste Punktdichte hat
- die Seite *S* wird dann mit allen (direkten und indirekten) Nachbarseiten *R* verschmolzen, deren Punktdichte kleiner oder gleich der Punktdichte von *S* ist
- wenn es nur noch Nachbarseiten mit höherer Punktdichte gibt:

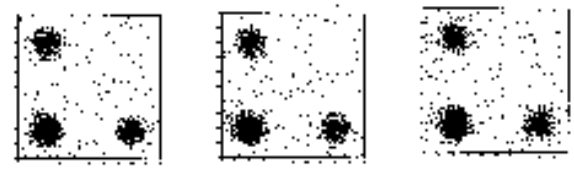
beginne einen neuen Cluster mit der Seite, die nun die höchste Punktdichte unter den noch nicht betrachteten Datenseiten hat

mit der zusätzlichen Information über die Verschmelzungsreihenfolge lässt sich das Ergebnis des Algorithmus als Dendrogramm darstellen!

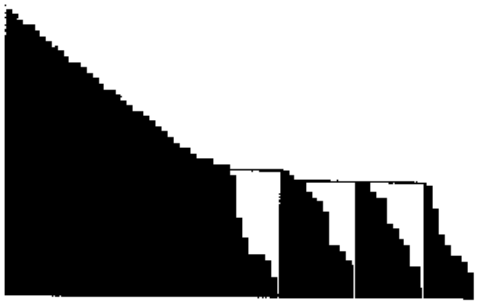
Beispiel



3-dimensionale Punktdaten



Dim1:Dim2 Dim1:Dim3 Dim2:Dim3



Resultierendes Dendrogramm

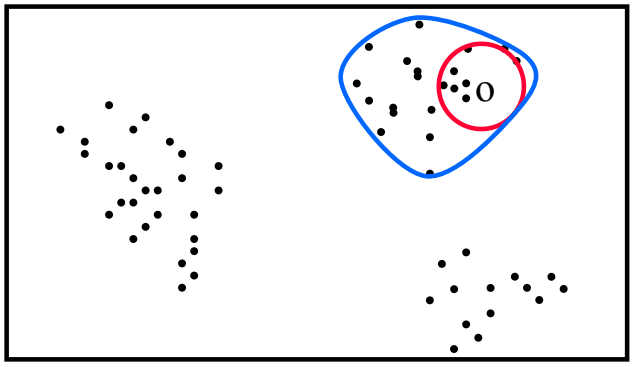
Fazit Micro-Clustering-Verfahren

- BIRCH gut geeignet für partitionierende Verfahren
- Data Bubbles geeignet für hierarchische Verfahren (speziell Single-Link und OPTICS)
- GridClustering geeignet für hierarchische Verfahren (speziell Single-Link und OPTICS)
- Die Qualität der gefundenen Cluster(-Struktur) hängt bei allen Verfahren ab von der Güte der Microcluster und daher indirekt von der Kompressionsrate
  - D.h. Anzahl der Punkte pro CF-Blatt (bzw. Die Wahl der Schwellwerte B,L und T)
  - Von der Anzahl der Punkte pro Data Bubble
  - Größe der Datenseite der verwendeten Indexstruktur beim Grid-Clustering

Inkrementelles DBSCAN

[Ester, Kriegel, Sander, Wimmer & Xu 1998]

- nicht die ganze aktualisierte Datenbank erneut clustern
- nur die alten Cluster und die eingefügten / gelöschten Objekte betrachten
- DBSCAN: nur die Nachbarschaft eines eingefügten / gelöschten Objekts und die davon dichte-erreichbaren Objekte sind *betroffen*



o Einfügung / Löschung

RQ(o, ε)

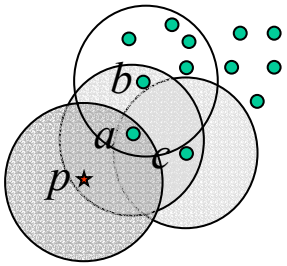
Vom Update betroffen

Grundlagen

- Kernpunkteigenschaft muss *inkrementell auswertbar* sein
- *Randobjekt*: gehört zum Cluster, ist aber kein Kernobjekt
- Potentielle Konsequenzen der Einfügung oder Löschung eines Objekts p

  In RQ(p, ε): Kernobjekte ↔ Randobjekte ↔ Rauschen

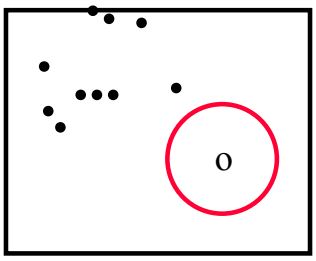
  In RQ(q, ε) mit $q \in$ RQ(p, ε): Randobjekte ↔ Rauschen



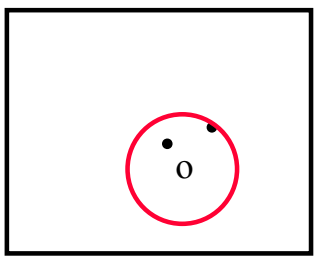
MinPts = 4, ε wie gezeigt

a: Randobjekt ↔ Kernobjekt
c: Rauschen ↔ Randobjekt

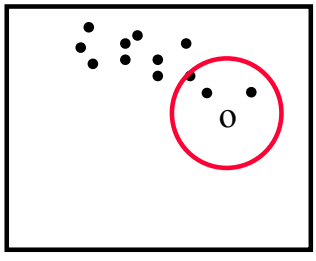
Einfügen



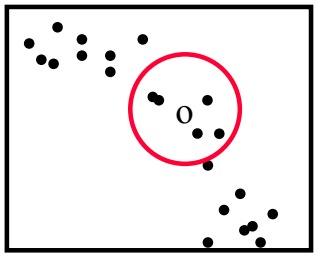
Rauschen



Neues Cluster



Erweiterung



Verschmelzen

o Einfügung

MinPts = 3, ε wie gezeigt

Virtuelle Cluster IDs

- speichere die Information, welche Cluster verschmolzen wurden
- Verschmelzen erfordert keinen Zugriff auf die betroffenen Cluster

Algorithmus

Einfügen von *p*:

```
Für alle "neuen" Kernobjekte o:
   Verbinde die Objekte in RQ(o,ε) mit dem Cluster, dem o zugehört
   Wenn Kernobjekte verschiedener Cluster nun miteinander verbunden sind,
   Verschmelze die entsprechenden Cluster
```

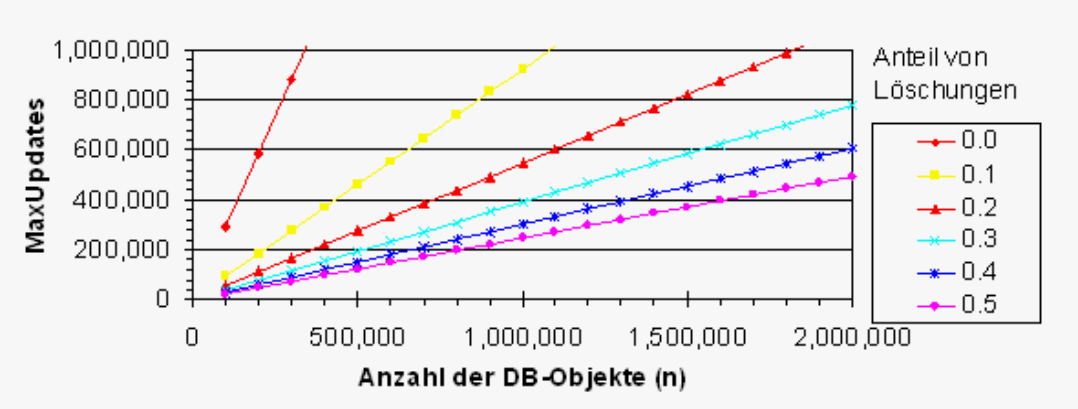
Löschen von *p*:

```
Für alle "zerstörten" Kernobjekte o:
   Setze die Cluster_Id aller Nicht-Kernobjekte aus RQ(o,ε) auf Noise;
   Füge alle Kernobjekte aus RQ(o,ε) in eine Menge UpdSeed ein;
Wende eine Variante von DBSCAN an, die jedoch eine Clusterexpansion
   nur mit Objekten aus der Menge UpdSeed beginnt;
```


Experimentelle Untersuchung

MaxUpdates: Zahl von Updates, bis zu denen Inkrementelles DBSCAN effizienter ist als DBSCAN angewendet auf die ganze aktualisierte Datenbank

Löschen ist üblicherweise teurer



sogar für 50 % Löschungen/Einfügung (worste-case Verteilung im Realfall):
MaxUpdates = 25 % der Datenbank Größe

Inkrementelles OPTICS

[Achtert, Böhm, Kriegel, Kröger 2005]

Analog wie DBSCAN:

- Nicht gesamte Cluster Ordnung neu berechnen
- Nur den Teil der CO neu berechnen in dem sich etwas verändert hat
  => Was kann sich verändern?
- In einem Lauf gleich für ein gesamtes Update-Set U (Bulk-Updates)

Beobachtung:

- Kerndistanz von Objekten können sich durch Update verändern
  - Einfügen: Kerndistanz kann schrumpfen
  - Löschen: Kerndistanz kann anwachsen
- Erreichbarkeitsdistanz kann sich in Folge ebenfalls ändern

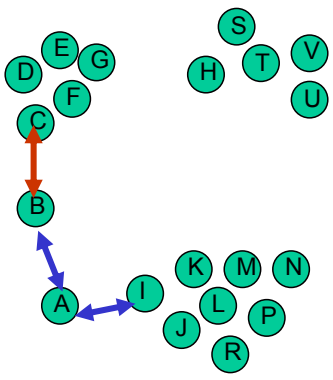
$$\text{Erreichbarkeitsdistanz}_{\varepsilon,MinPts}(p,o) = \max(\text{Kerndistanz}_{\varepsilon,MinPts}(o), \text{dist}(p,o))$$

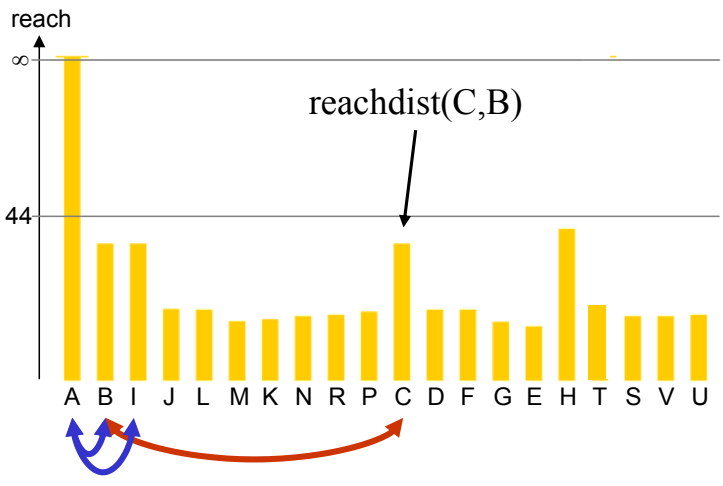
Definition: *Vorgänger* eines Objekts pre(o)

Objekt p von dem aus o im OPTICS-Lauf erreicht wurde

Definition: *Nachfolger* eines Objekts succ(o)

Alle Objekte p deren Vorgänger o ist, also pre(p) = o





z.B. C wurde von B erreicht => pre(C) = B

z.B. pre(I) = A und pre(B) = A => succ(A) = {B, I}

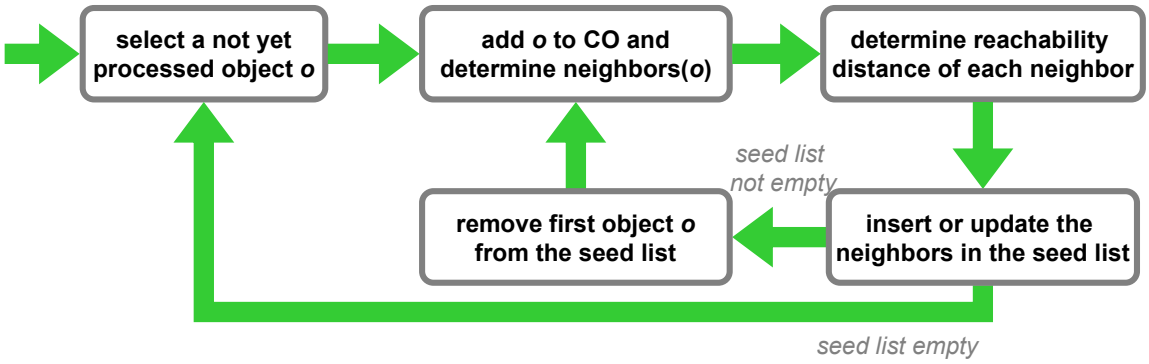
Definition: *Clusterordnung*

Seien $\mu \in \aleph, \varepsilon \in \Re$ und CO eine Permutation der Objekte in DB. Jedes Objekt $o \in DB$ hat drei zusätzliche Attribute $o.P, o.C$ und $o.R$, wobei $o.P \in \{1,...,n\}$ die Position von o in CO sybolisiert. CO ist eine Clusterordnung wenn die folgenden Bedingungen erfüllt sind :

(1) $\forall o \in CO: o.C = \text{Kerndistanz}_{\varepsilon,\mu}(o)$

(2) $\forall o \in CO: o.P = i \Rightarrow o.R = \min\{\text{Erreichbarkeitsdistanz}_{\varepsilon,\mu}(x,y) \mid x.P < i \wedge y.P \geq i\}$

Algorithmus OPTICS



- Aufgrund einer Update Operation können sich die Kerndistanzen und Erreichbarkeitsdistanzen einiger Objekte ändern
  → diese Objekte haben möglicherweise eine falsche Position in der Clusterordnung *CO*

- Nur für diese *betroffenen* Objekte eine neue Position in der neuen Clusterordnung CO_{new} berechnen

- Alle anderen Objekte werden aus der alten Clusterordnung CO_{old} übernommen

- Zwei Arten von betroffenen Objekten („affected objects“)
  - Direkt betroffene Objekte
  - Indirekt betroffene Objekte

Direkt betroffenen Objekte

- Objekte, deren Kerndistanz sich ändert
- Definition: *Direkt betroffene Objekte*

$$DAff\ (U) = \{o \in \mathsf{RNN}(u,k) \mid u \in U \wedge dist(o,u) \leq \varepsilon\}$$

wobei $x \in \mathrm{RNN}(y,k) \Leftrightarrow y \in \mathrm{NN}(x,k)$ die k-invers nächsten Nachbarn bezeichnet

- Intuitiv:

$o \in DAff(U)$ wenn $\exists u \in U: u \in \mathrm{RQ}(o,\varepsilon)$

- Direkt betroffene Objekte können die Erreichbarkeitsdistanzen (und damit die Positionen in *CO*) anderer Objekte verändern

→ Reorganisation dieser Objekte („*indirekt betroffene*“ Objekte)

Indirekt betroffenen Objekte

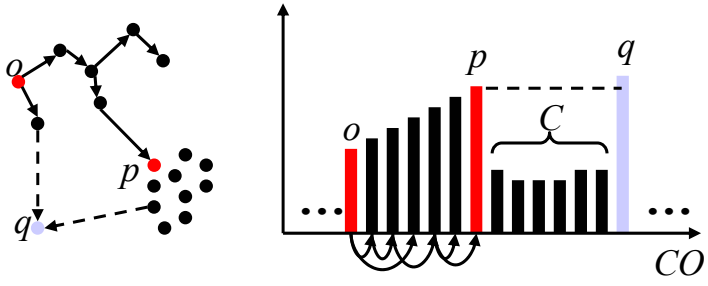
- Objekte, deren Erreichbarkeitsdistanz sich aufgrund veränderter Kerndistanzen verändert haben
- So nicht einfach bestimmbar
- Obermenge: Potentielle Nachfolger („Potential Successors“)

  Objekte, die durch veränderte Kerndistanzen oder reorganisierte Objekte ebenfalls reorganisiert werden müssen
- Definition: *Potential Successors* eines Objekts *o*, $SUC^{pot}(o)$

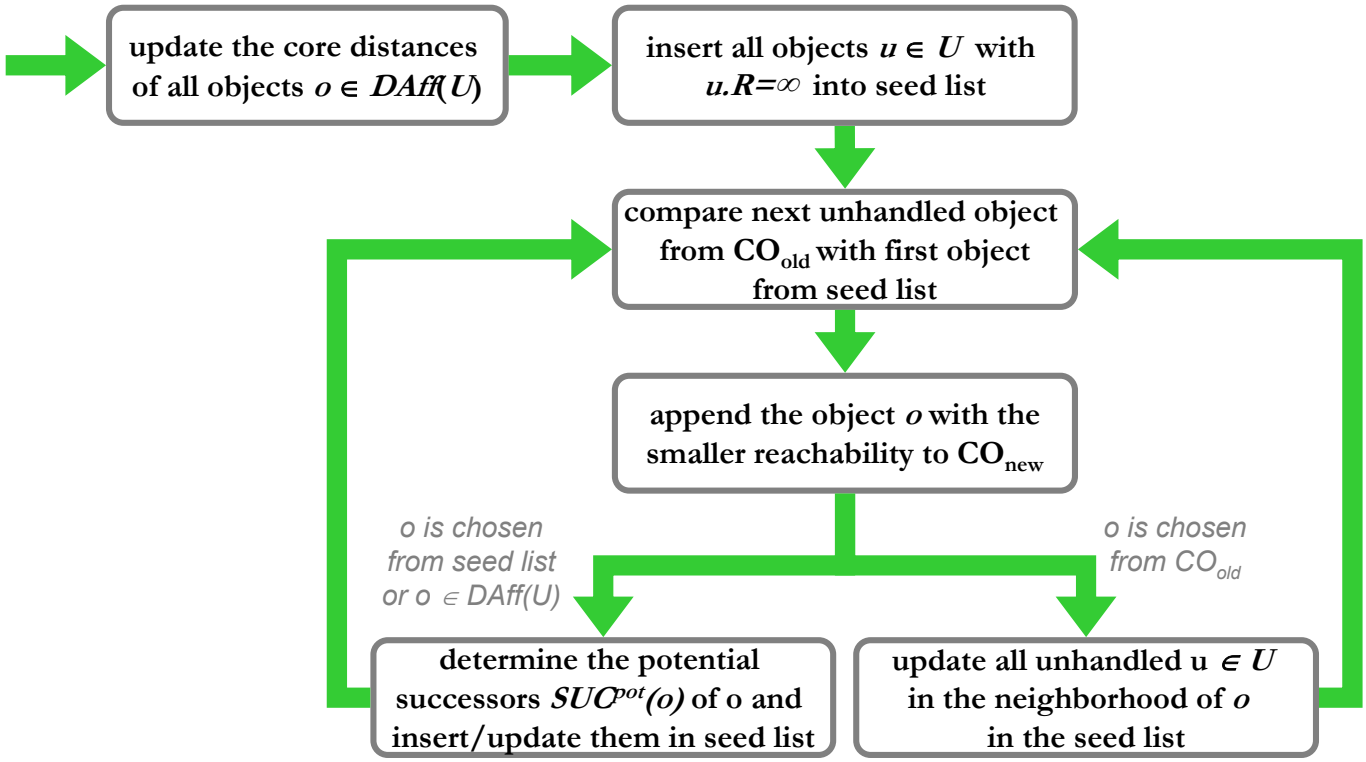
(1) $p \in SUC(o) \Rightarrow p \in SUC^{pot}(o)$

(2) $q \in SUC^{pot}(o) \wedge p \in SUC(q) \wedge x \in SUC^{pot}(o) \wedge (x.P = p.P - 1 \Rightarrow x.R \leq p.R)$

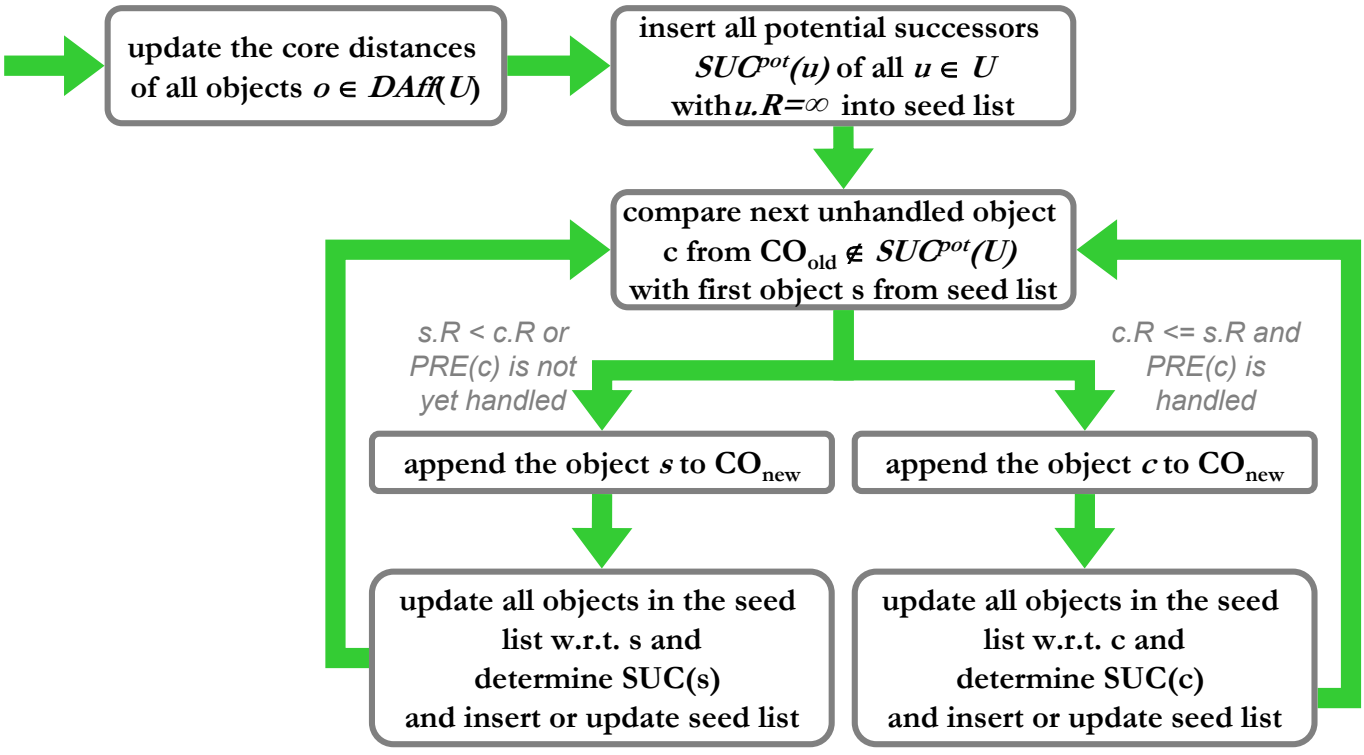
$\Rightarrow p \in SUC^{pot}(o)$



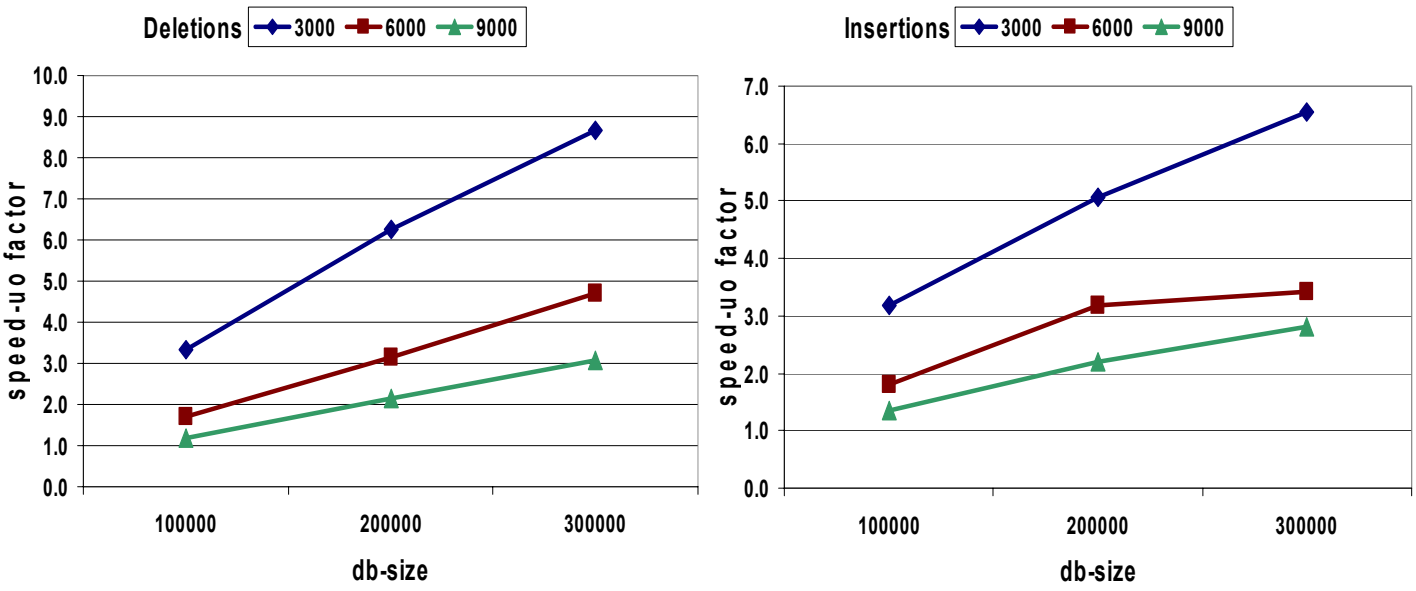
Online Bulk Insertions



Online Bulk Deletions

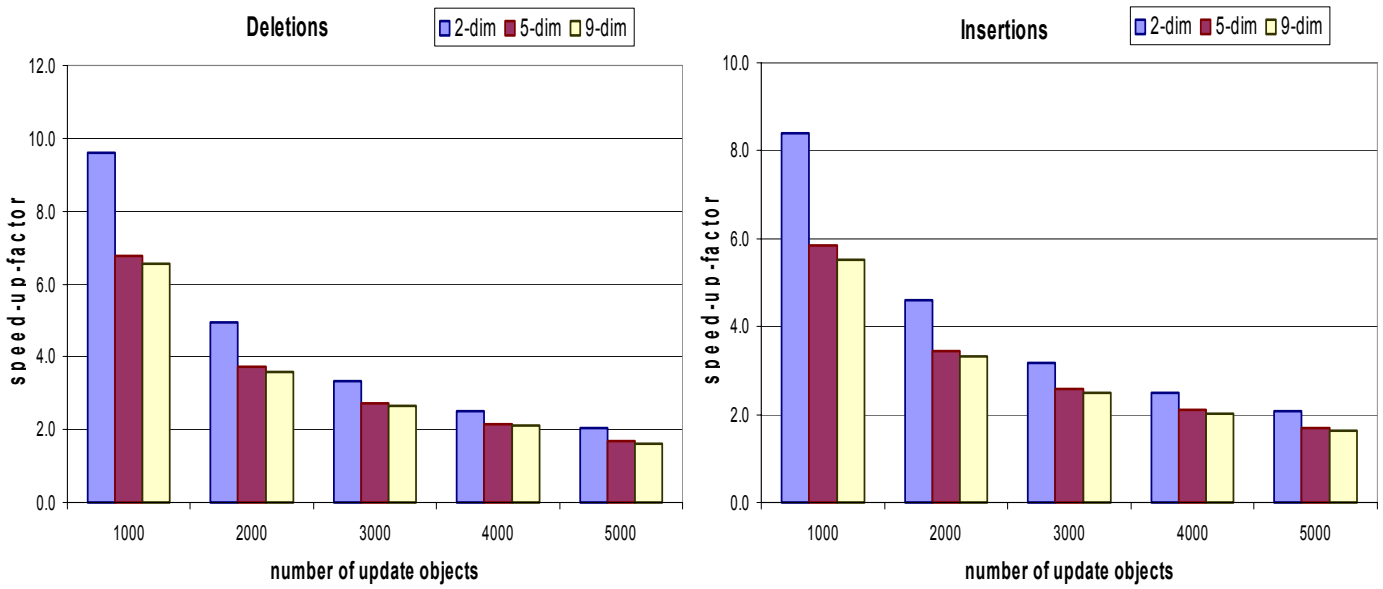


Speed-Up Faktoren



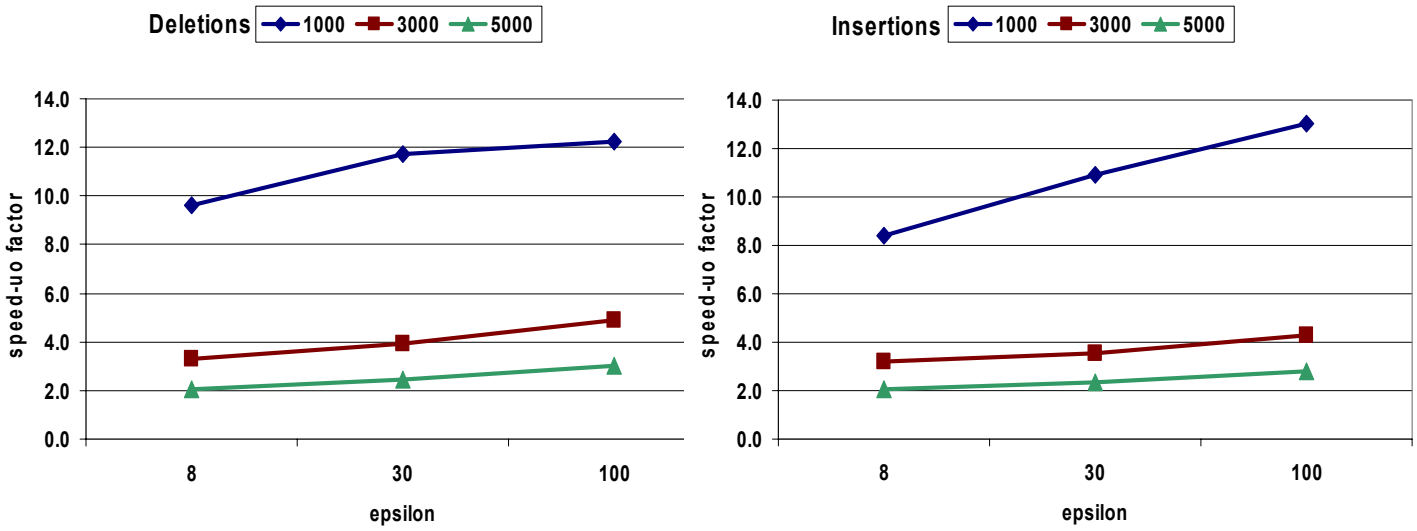
8.2 Online Data Mining

Speed-Up Faktoren

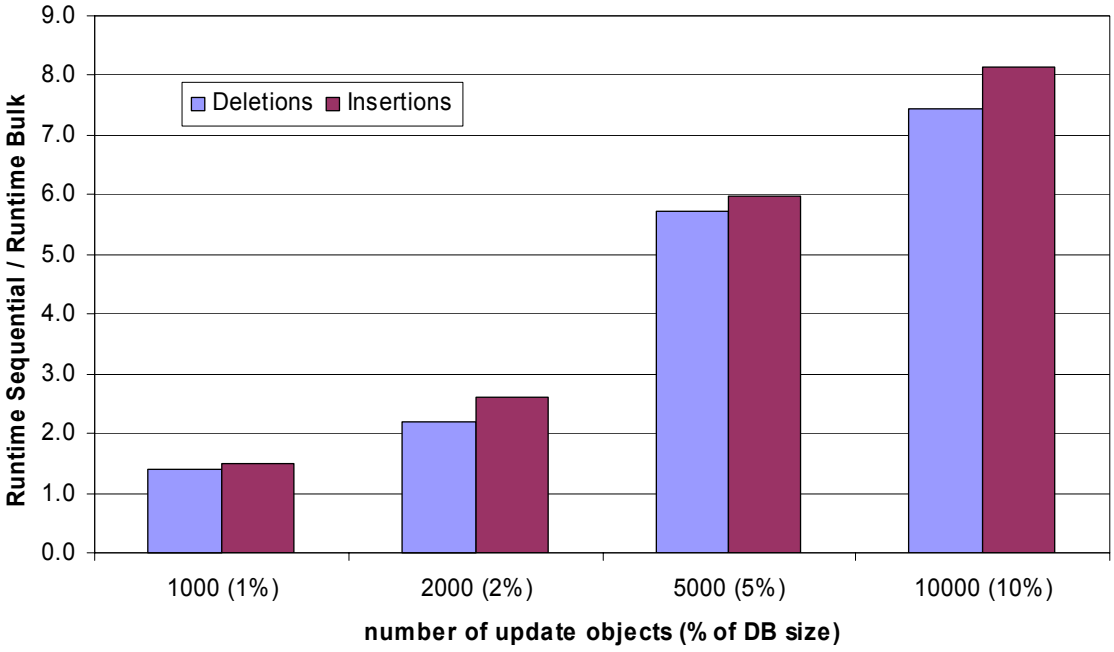


(synthetic data sets of 100.000 objects)

Speed-Up Faktoren



Speed-Up Faktoren: Sequentieller Update / Bulk Update



Bemerkungen zum Online Data Mining:

- Inkrementelle / Online DM Algorithmen in großen Data Warehouses unverzichtbar
- Ähnliche (wichtige) Anwendung: Data Mining in Datenströme

  Datenströme (Data Streams): es kommen ständig neue Datenobjekte zur DB hinzu, die geclustert/klassifiziert/etc. werden müssen
- Unterschied zum Stream-Mining:

  Online DM: Bulk-Updates möglich, d.h. während der Reorganisation der Muster sind alle Objekte des Update-Sets *U* bekannt

  DM in Streams: kein Bulkupdate möglich, d.h. ein Update-Objekt (typischerweise eine Einfügung) pro Reorganisation

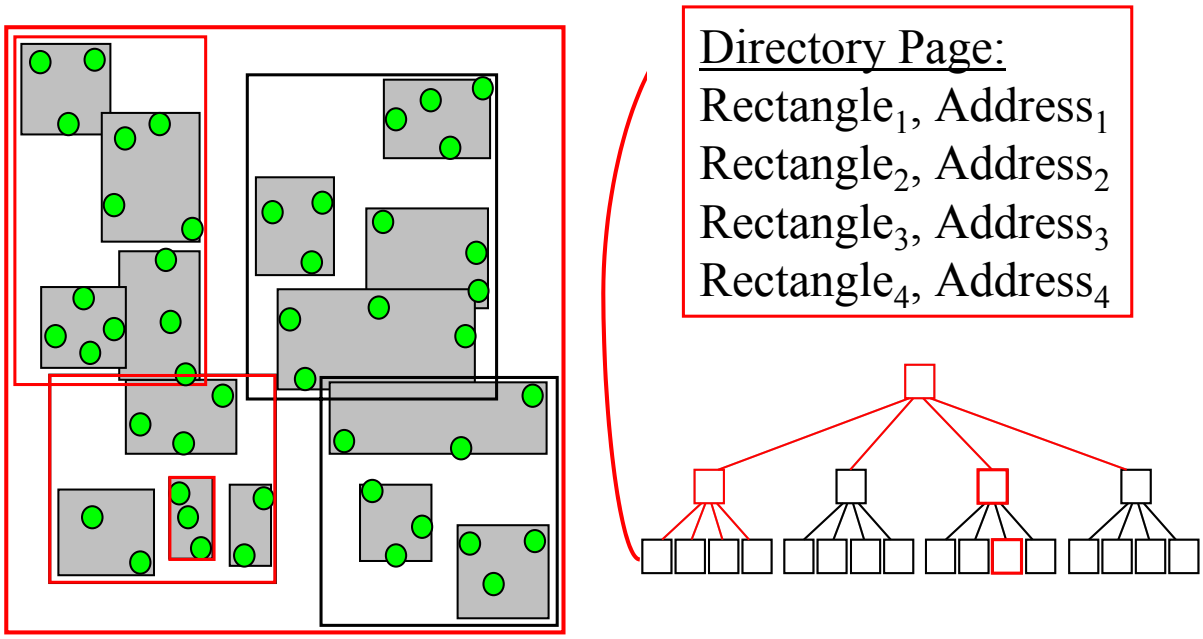
8.3 Indexstrukturen

Motivation

- DM-Algorithmen benötigen Distanzberechnungen und verwenden Ähnlichkeitsanfragen
- Sequentieller Scan für Ähnlichkeitsanfragen über die Datenbank auf dem Sekundärspeicher sehr teuer
- Indexstrukturen versuchen die Datenobjekte „clever“ auf dem Hintergrundspeicher zu organisieren
- Ziel: Lade für eine Anfrage nicht alle Datenseiten sondern möglichst nur die „relevanten“ Datenseiten (Datenseiten, auf denen sich Treffer befinden)

  => Minimierung der I/O-Kosten (meist mit einem vernachlässigbaren Mehraufwand bei CPU-Kosten verbunden)
- Räumliche Indexstrukturen meist baumartig organisiert
  - für mehrdimensionale Punktdaten (R-Baum, R*-Baum, X-Baum, …)
  - Für allgemein metrische Objekte (M-Baum)

Prinzip von Indexstrukturen



Bei allgemein metrischen Daten: keine MURs

Daher meist (z.B. M-Baum): Anchor-Objekt + Covering-Radius

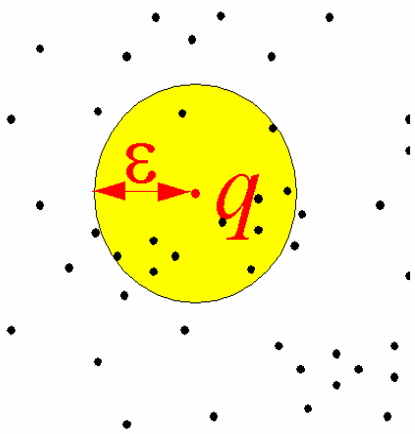
Beispiel: Range Queries

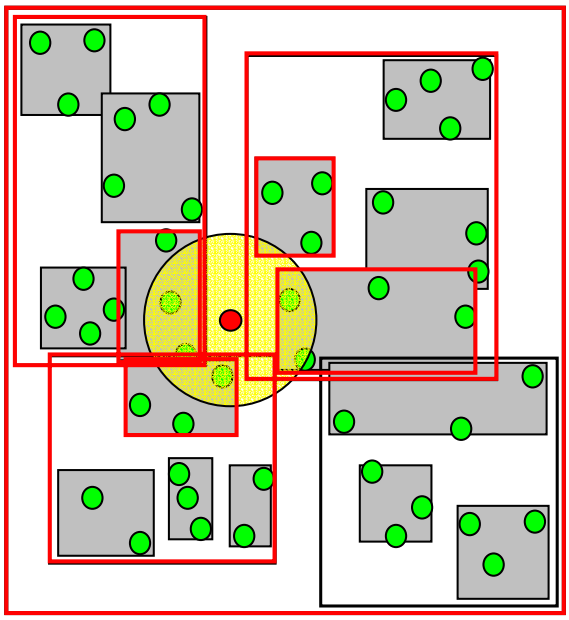
Gegeben: Anfragepunkt q
Anfragedistanz ε

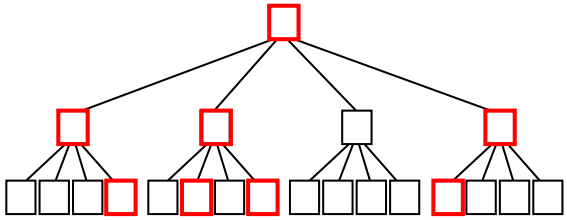
Formale Definition:

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

Kardinalität des Ergebnisses ist schwer zu kontrollieren:
- ε zu klein ➔ keine Ergebnisse
- ε zu groß ➔ komplette DB







Indexstrukturen für dichtebasiertes Clustering

- Basisoperation für DBSCAN und für OPTICS:
  Berechnung der ε-Nachbarschaft jedes Objekts *o* in der Datenbank

- effiziente Unterstützung von Bereichsanfragen durch räumliche Indexstrukturen

  R-Baum, X-Baum, M-Baum, . . .

- Laufzeitkomplexitäten für die Algorithmen DBSCAN und OPTICS:

	einzelne Bereichsanfrage	gesamter Algorithmus
ohne Index	$O(n)$	$O(n^2)$
mit Index	$O(\log n)$	$O(n \log n)$
mit direktem Zugriff	$O(1)$	$O(n)$

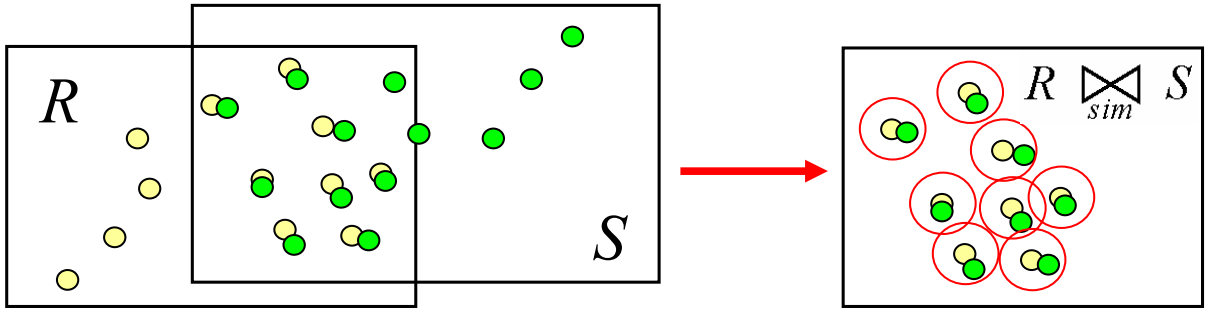
Fazit Indexstrukturen:

- As Performanzsicht: Indexstrukturen bei großen Datenmengen beinahe unumgänglich
- Gut bei niedrig-dimensionalen Daten (2-5 Dimensionen)
- Bei höher-dimensionalen Daten meist starker Leistungsabfall (Curse of Dimensionality)
- „Indexstrukturen und Clustering-Algorithmen haben ähnliche Ziele“

8.4 Similarity Joins

Ähnlichkeitsverbund („Similarity Join“)

- Der Join ist ein DB-Grundoperation (effiziente Algorithmen zur Berechnung)
- Gegeben: zwei Mengen *R*, *S* von DB-Objekten
- Gesucht: Finde alle Paare von Punkten, die der Join-Bedingung genügen



- Verschiedenste Definitionen für Ähnlichkeit in Join-Bedingung möglich

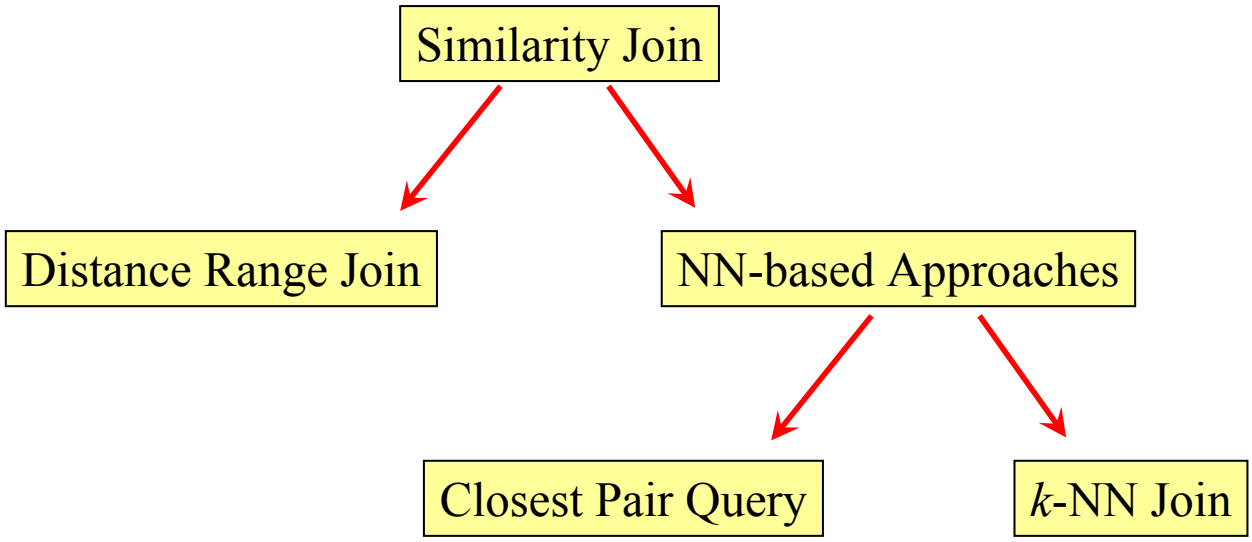
Intuitive Definition: Similarity Join

1. Der Similarity Join ist ein Join im relationalen Sinn:
   Zwei Objektmengen R und S werden so kombiniert, dass die Ergebnismenge alle Paare von Punkten enthält, die eine Join-Bedingung erfüllen

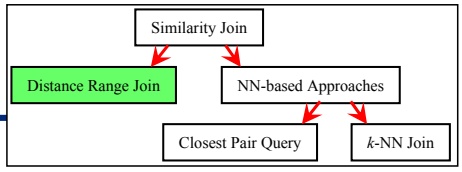
$$R \underset{sim}{\bowtie} S \subseteq R \times S$$

2. Vektordaten oder metrische Daten statt allgemeine Datenbank-Tupel beliebigen Typs
3. Die Join-Bedingung basiert auf einem Ähnlichkeitsprädikat

Arten von Similarity Joins



8.4 Similarity Joins



Distance-Range-Join (ε-Join)

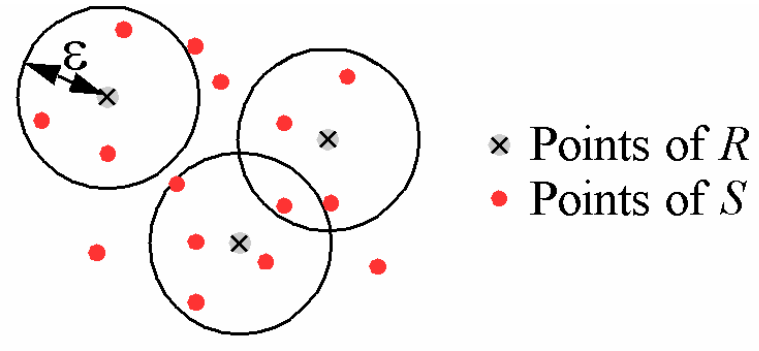
- Gegeben ε
- Definition:

$$R \underset{\varepsilon}{\bowtie} S := \{(r_i, s_j) \in R \times S : \|r_i - s_j\| \leq \varepsilon\}$$

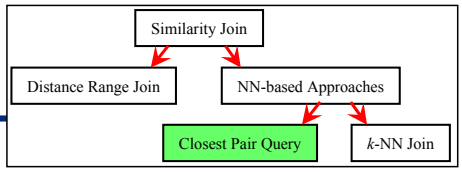
- SQL-Statement:

SELECT * **FROM** R, S **WHERE** $\|R.\text{obj} - S.\text{obj}\| \leq \varepsilon$

- Am Häufigsten verwendet und gut untersucht (algorithmisch)



8.4 Similarity Joins



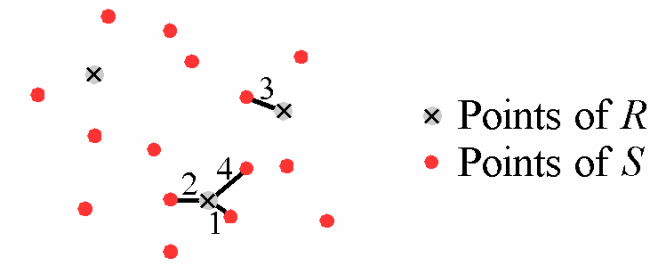
k-Closest Pair Query

- Intuition: Finde die k Paare mit kleinster Distanz
- Definition: $R \underset{k\text{-CP}}{\bowtie} S$ is the smallest subset of $R \times S$ that contains at least k pairs of points and for which the following condition holds:

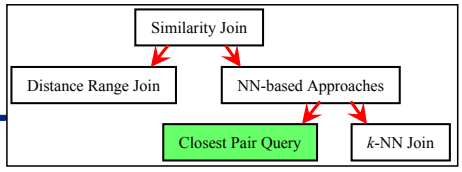
$$\forall\, (r,s) \in R \underset{k\text{-CP}}{\bowtie} S,\ \forall\, (r',s') \in R \times S \setminus R \underset{k\text{-CP}}{\bowtie} S : \|r-s\| < \|r'-s'\|$$

- SQL-Statement:

SELECT * **FROM** R, S
ORDER BY $\|R.\text{obj} - S.\text{obj}\|$
STOP AFTER k



8.4 Similarity Joins

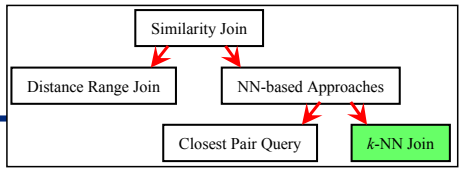


k-Closest Pair Query (cont.)

- Mehr als *k* Objekte mit gleichem Abstand zum Query-Objekt?
  - Ergebnismenge erweitern
  - Nicht-Determinismus (z.B. zufällige Auswahl der Ergebnisse)
- Inkrementelles Ranking
  - Keine **STOP AFTER** Klausel:

    SELECT * **FROM** *R*, *S*
    ORDER BY ||*R*.obj - *S*.obj||
  - Cursor "holt" Ergebnis Paare nacheinander (Fetch-Operation)

8.4 Similarity Joins

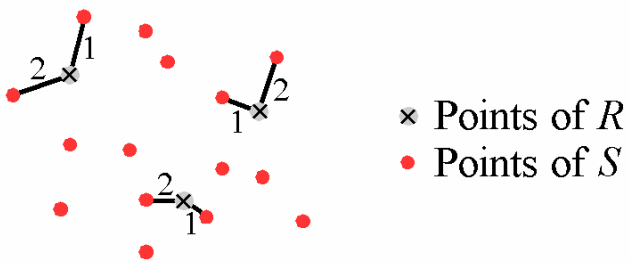


k-nächster Nachbar Join

- Kombiniere jeden Punkt mit seinen *k*-nächsten Nachbarn
- Definition: $R \overset{k\text{-NN}}{\ltimes} S$ is the smallest subset of $R \times S$ that contains for each point of R at least k points of S and for which the following condition holds:

  $\forall\ (r,s) \in R \overset{k\text{-NN}}{\ltimes} S,\ \forall\ (r,s') \in R \times S \setminus R \overset{k\text{-NN}}{\ltimes} S\text{: } \|r-s\| < \|r-s'\|$
- SQL-Statement (nur für k=1):

  SELECT * **FROM** *R*, *S*
  GROUP BY *R*.obj
  ORDER BY ||*R*.obj - *S*.obj||



Anwendungsbeispiele:

- Join-basiertes *k*-means / *k*-medoid
  - Unterstützung von Schritt 2: „DB-Objekte zu Clusterzentren neu zuordnen“
  - NN-Join: DB-Punkte $\ltimes_{1\text{-NN}}$ Clusterzentren

- Join-basierte *k*-NN Klassifikation [Braunmüller, Ester, Kriegel, Sander 2000]
  - Speziell wenn man große Mengen an Objekten gleichzeitig klassifizieren muss
    - Astronomie: 10.000 gesammelte Objekte pro Nacht
    - Online Kundenbewertung: Bewertung aller Kunden, die momentan online sind
  - Unterstützung des Klassifikationsschrittes: „berechne k-nächste Nachbarn für unbekanntes Objekt“
  - *k*-NN-Join: Neue Objekte $\ltimes_{k\text{-NN}}$ Trainingsobjekte

- Join-basierter DBSCAN [Böhm, Braunmüller, Breunig, Kriegel 2000]
  - Schema:

```
foreach Point p ∈ DB
        PointSet Neighborhood := RQ(p, ε);
        if |Neighborhood| >= MinPts
                ExpandCluster(p,Neighborhood);
```

Kernpunkteigenschaft

Clusterexpansion

  - ε-Join:

```
foreach PointPair (p,q) ∈ R ⋈_ε R
    DoSomething (p,q);
```

  - DoSomething (*p*,*q*) :
    - Kernpunkteigenschaft
    - Clusterexpansion

8.4 Similarity Joins

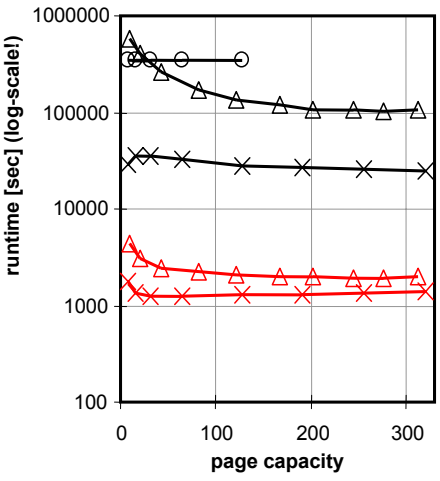
– DoSomething (*p*,*q*) :
  – Kernpunkteigenschaft:

    Zähler *p*.counter und *q*.counter erhöhen (*p* bzw. *q* Kernpunkt, wenn mehr als *MinPts* Paare für *p* gefunden wurden)
  – Clusterexpansion (vgl. Tabelle):
    » ClusterID an alle Punkte, die mit einem bekannten Kernpunkt ein Paar bilden
    » bekannte Cluster vereinigen

p \ *q*		***q*.counter >= MinPts**		***q*.counter < MinPts**	
		***q*.ID gültig**	***q*.ID = NULL**	***q*.ID gültig**	***q*.ID = NULL**
***p*.counter >= MinPts**	***p*.ID gültig**	**IF** *p*.ID<>*q*.ID **THEN** merge	*q*.ID = *p*.ID		*q*.ID = *p*.ID
	***p*.ID = NULL**	*p*.ID = *q*.ID	*p*.ID = *q*.ID = **NEW**(ID)		*q*.ID = *p*.ID = **NEW**(ID)
***p*.counter < MinPts**	***p*.ID gültig**				
	***p*.ID = NULL**	*p*.ID = *q*.ID	*p*.ID = *q*.ID = **NEW**(ID)		

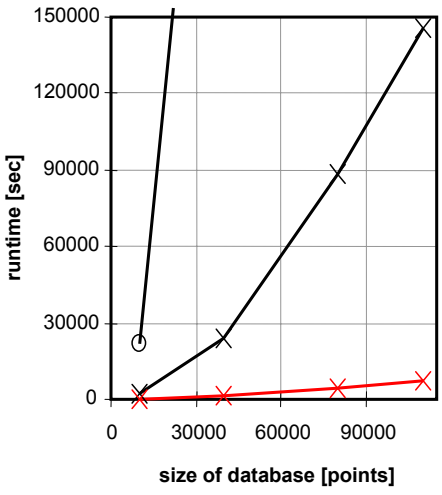
8.4 Similarity Joins

Color image data



Q-DBSCAN (Seq. Scan)
Q-DBSCAN (R*-tree)
Q-DBSCAN (X-tree)
J-DBSCAN (R*-tree)
J-DBSCAN (X-tree)

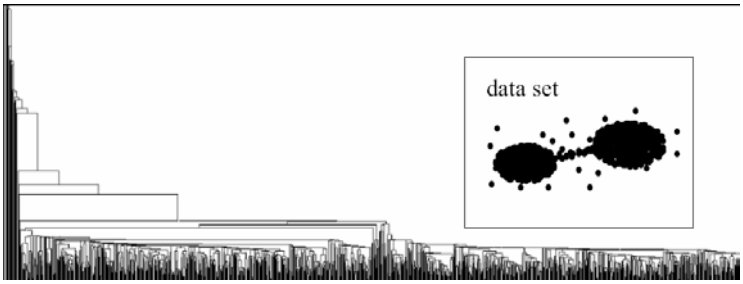
Color image data

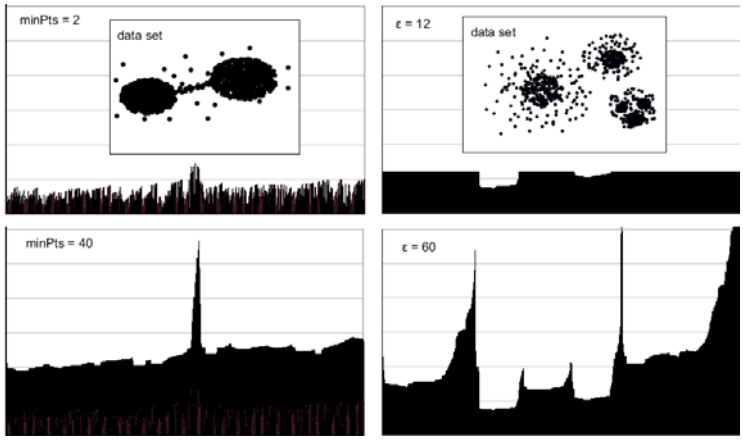


Q-DBSCAN (Seq. Scan)
Q-DBSCAN (X-tree)
J-DBSCAN (X-tree)

8.4 Similarity Joins

- Density-Link Clustering (DeLiClu) [Achtert, Böhm, Kröger 2006]
  - Nachteile von Single-Link:
    - Single-Link Effekt
    - Unübersichtliches Dendrogramm
    - Hohe Laufzeit
  - OPTICS:
    - Einführung von MinPts gegen Single-Link Effekt
    - Nutzung von Indexstrukturen
  - ABER:
    - ε-Parameter: Trade-Off zwischen Performanz und Accuracy





8.4 Similarity Joins

- Lösung: Kombination von OPTICS und Single-Link mit Closest-Pair-Join (Closest-Pair-Ranking, CPR)
- Idee:
  - gegeben:

    R = Menge von Objekten, die bereits in die Clusterstruktur (Dendrogramm/Erreichbarkeitsdiagramm) integriert wurden

    S = Menge noch nicht behandelter Objekte
  - Hauptschleife eines hierarchischen Algorithmus:

    Berechne das „Closest Pair“ $(r,s) \in (R \times S)$

    Lösche s aus S und füge s in R ein

    Füge s am Ende der Clusterstruktur ein
  - Distanz für Closest Pair Ranking:

    $\text{DenDist}(r,s) = \max\{k\text{-nndist}(r), \text{dist}(r,s)\}$ mit $k = MinPts$
  - k-nn-Distanz kann mit k-nn-Join auf $DB \times DB$ vorberechnet werden

8.4 Similarity Joins

Algorithmus DeLiClu(*DB*, *MinPts*)

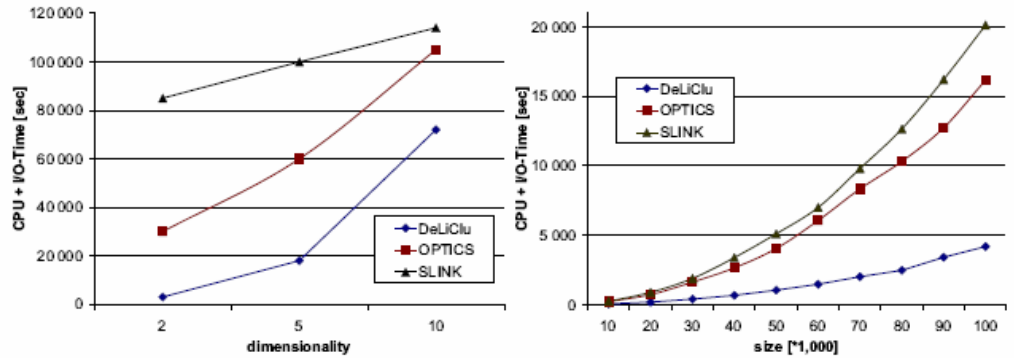
1. Berechne $DB \underset{MinPts\text{-NN}}{\ltimes} DB$
2. Menge *R* enthält ein beliebiges Startobjekt aus *DB*
3. Menge $S := DB - R$
4. Initialisiere CPR über $(R \times S)$ mit DenDist als Distanzfunktion
5. Hole nächstes Paar (r,s) aus CPR
6. Lösche *s* aus *S* und füge *s* in *R* ein
7. Hänge s an das Ende des Erreichbarkeitsdiagramms an
8. Wiederhole ab Schritt 5 solange $S \neq \emptyset$

Eigenschaften:

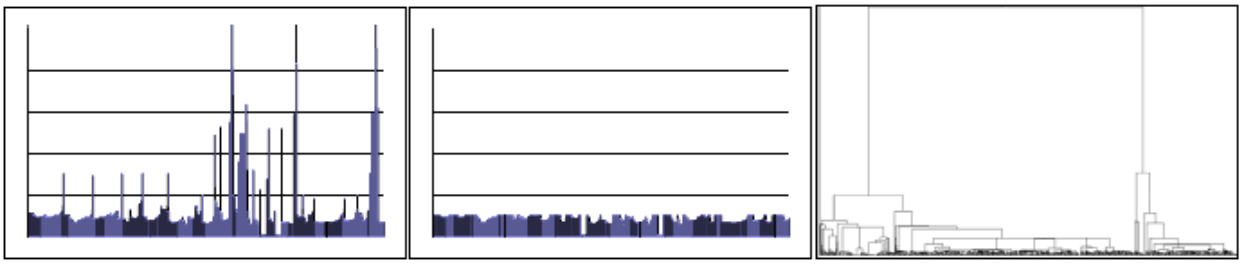
- Bessere Performanz gegenüber SLINK und OPTICS
- Leichtere Parameterwahl (einziger Parameter: *MinPts*) gegenüber OPTICS
- Kein Verlust von Clusterinformationen durch falsche Parameterwahl (ε bei OPTICS)
- Höhere Robustheit gegen Rauschen gegenüber SLINK durch Parameter *MinPts*

8.4 Similarity Joins

Performanz



Cluster Qualität



DeLiClu | OPTICS (ε so gewählt, dass Laufzeit mit DeLiClu vergleichbar) | SLINK

Fazit Similarity-Join:

- Umformulierung der ursprünglichen Algorithmen (nicht immer möglich) ergibt klaren Performanzgewinn
- Integration von DM-Algorithmen in DBMS (Join-Algorithmus, Anfrageoptimierung, etc.)
- Join-Algorithmen verwenden typischerweise Indexstrukturen und haben daher ähnliche Probleme (Curse of Dimensionality)