

Seminar Komplexe Objekte in Datenbanken

OPTICS: Ordering Points To Identify the Clustering Structure

Lehrstuhl für Informatik IX - Univ.-Prof. Dr. Thomas Seidl, RWTH-Aachen

<http://www-i9.informatik.rwth-aachen.de>

11. Februar 2003, Alexander Lay

alex@nwadmin.de

Quelle:

Ankerst M., Breunig M. M., Kriegel H.-P., Sander J.: OPTICS: Ordering Points To Identify the Clustering Structure, Proc. ACM SIGMOD Int. Conf. on Management of Data 1999: 49-60
[ABKS 99] <http://www.cs.ualberta.ca/~joerg/papers/OPTICS-final.pdf>

1. Einführung
2. Ordnung der Datenbankobjekte in Bezug auf die Clusterstruktur
 - 2.1 Motivation
 - 2.2 Definitionen
 - 2.3 Algorithmus OPTICS
3. Identifizierung von Clusterstrukturen
 - 3.1 visuelle Methoden
 - 3.2 automatische Methoden
4. Zusammenfassung

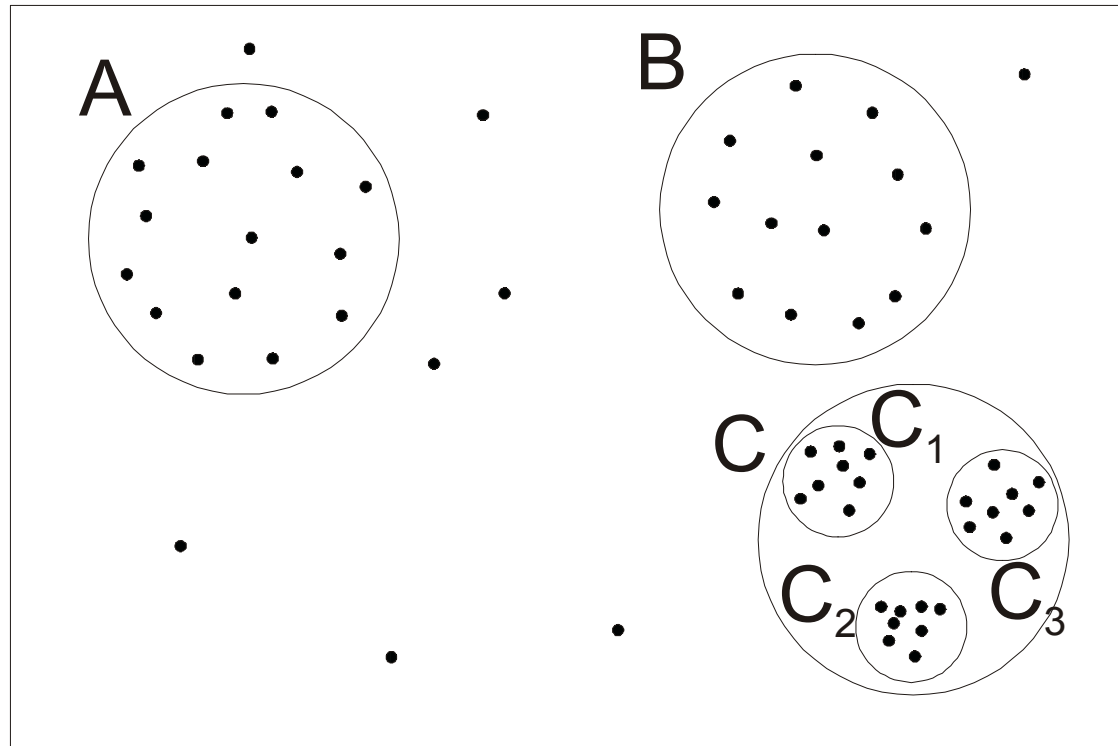
1. Einführung

- Analyse von Datenbanken auf Cluster ist eine primäre Methode beim Database Mining
- Qualität der Ergebnisse viele Algorithmen variiert stark von den Werten der Eingabeparameter
- Besonders bei unbekanntem Daten sind gute Werte schwer zu schätzen
- OPTICS erkennt keine Cluster sondern erzeugt eine Ordnung der Objekte, diese repräsentiert eine auf Dichten basierende Clusterstruktur

2. Ordnung der Datenbankobjekte in Bezug auf die Clusterstruktur

2.1 Motivation

Cluster mit verschiedenen Dichten



Algorithmus mit Dichteparameter erkennt:

- A, B und C oder
- C_1 , C_2 und C_3

2.2 Definitionen

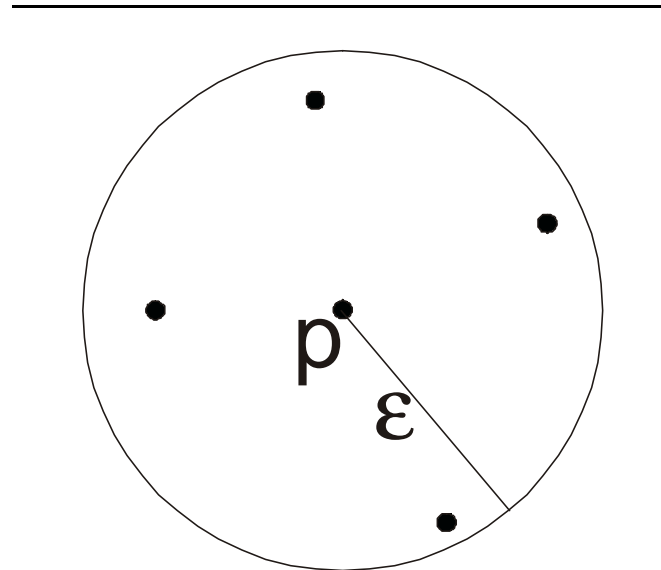
Definition 1: core object

p ist ein core object

\Leftrightarrow

In der ε -Umgebung von p liegen min. $MinPts$ Objekte (incl. p selbst).

(Die Nachbarschaft von p hat eine Mindestgröße von $MinPts$.)



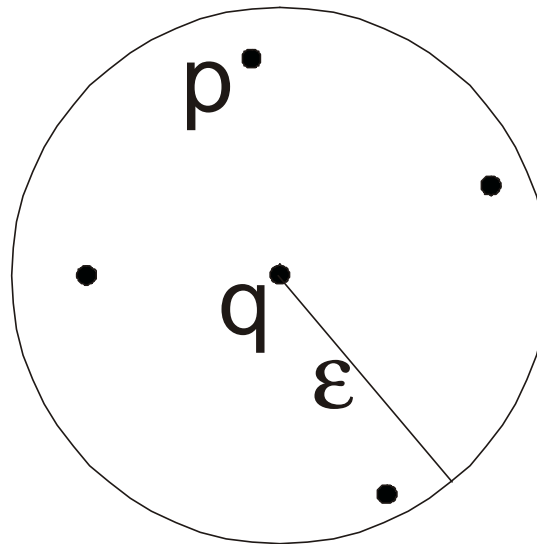
p ist core object mit dem Radius ε und für $MinPts \leq 5$

Definition 2: directly density-reachable

p ist directly density-reachable von q aus bzgl. ε und $MinPts$

\Leftrightarrow

- 1) p liegt in der ε -Umgebung von q
- 2) q ist ein core object.



p ist directly density-reachable von q aus

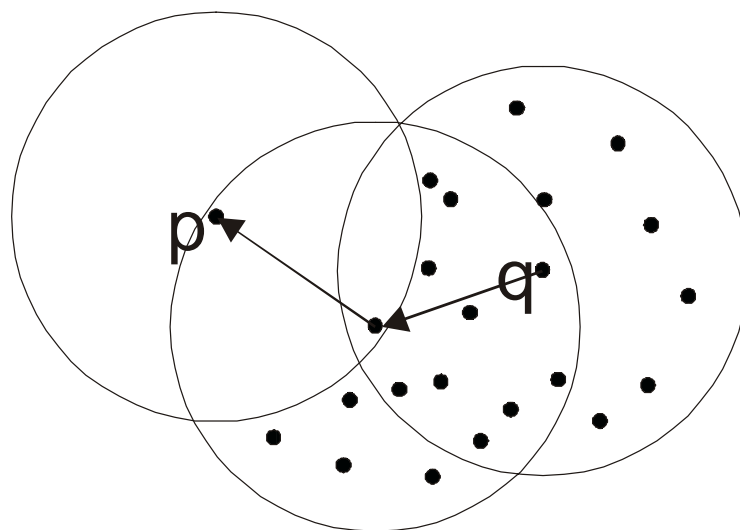
Definition 3: density-reachable

p ist density-reachable von q aus bzgl. ϵ und $MinPts$

\Leftrightarrow

- 1) Es existieren Objekte p_1, \dots, p_n mit $p_1 = q, p_n = p$
- 2) jedes p_{i+1} ist directly density-reachable von p_i aus bzgl. ϵ und $MinPts$.

Nur core objects sind in beide Richtungen density-reachable.



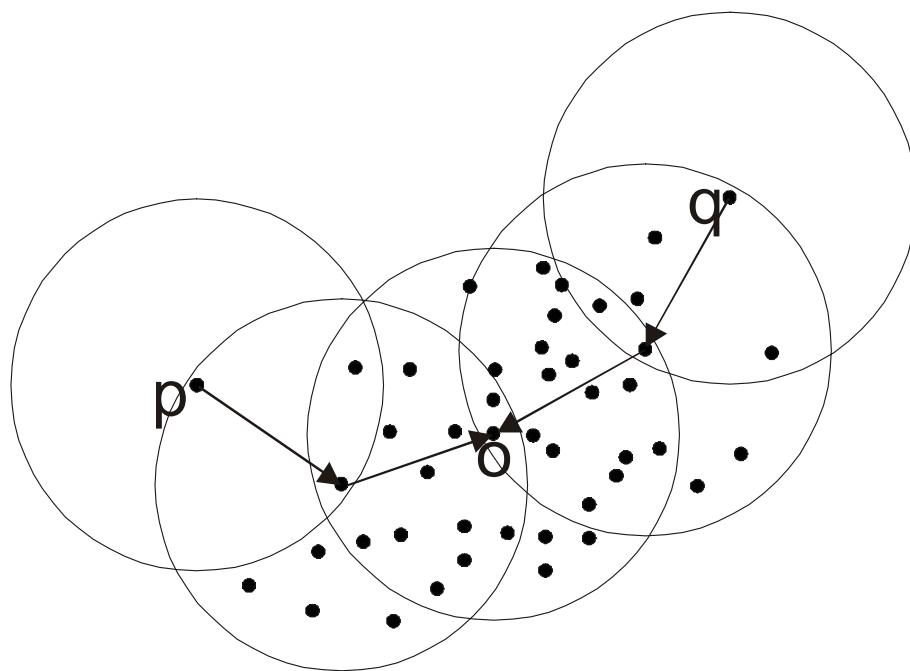
p ist density-reachable von q

Definition 4: density-connected

p ist density-connected mit q bzgl. ϵ und $MinPts$

\Leftrightarrow

- 1) Es existiert ein Objekt o
 - 2) p und q sind density-reachable von o aus bzgl. ϵ und $MinPts$.
- q ist also ein core object, density-connected ist eine symmetrische Eigenschaft.



p ist über o density-connected mit q

Definition 5: cluster und noise

C ist ein Cluster bzgl. ϵ und $MinPts$

\Leftrightarrow

1) **Maximality:**

für alle Objekte p, q gilt:

Wenn $p \in C$ und q ist density-reachable von p
aus bzgl. ϵ und $MinPts$

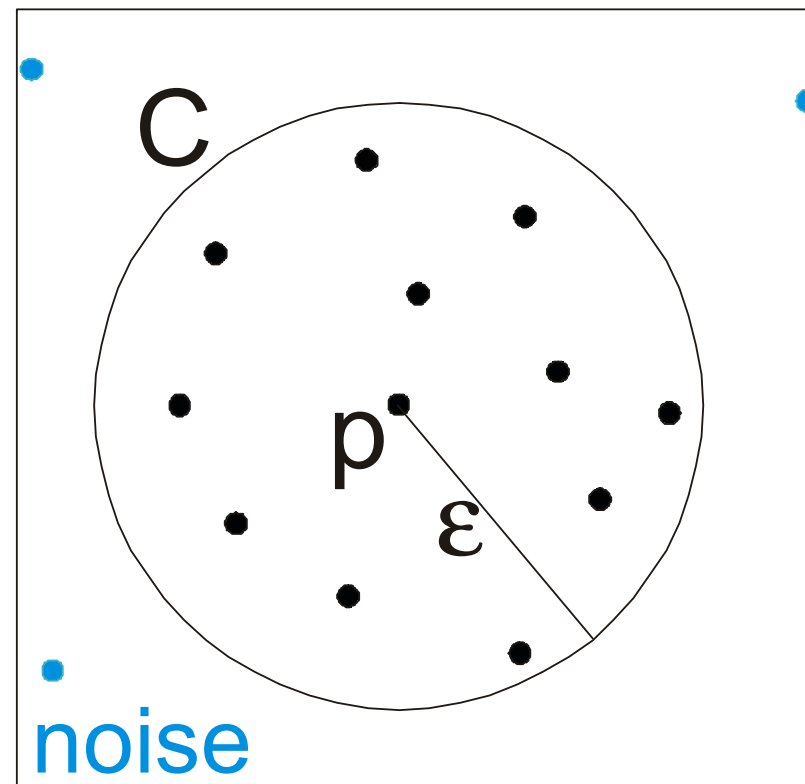
$\Rightarrow q \in C$

2) **Connectivity:**

für alle Objekte p, q in C gilt:

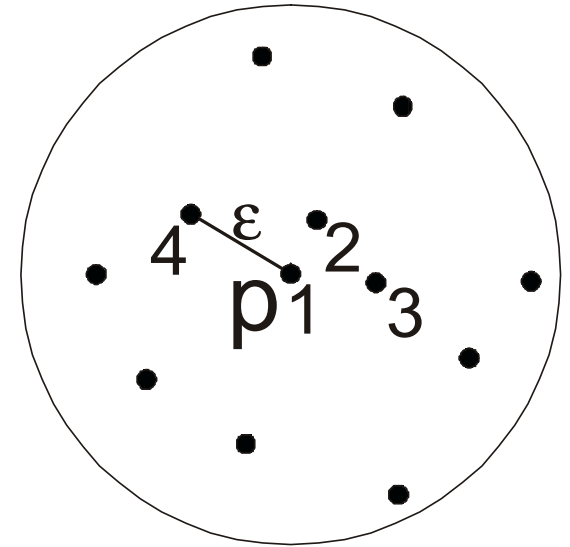
p ist density-connected von q aus bzgl. ϵ und
 $MinPts$

Jedes Objekt, dass nicht in einem Cluster ist, ist
noise.



Definition 6: *MinPts-distance(p)*

$MinPts\text{-distance}(p) =$
Entfernung von p bis zum $MinPts$ -ten Objekt,
 p wird mitgezählt



$$MinPts\text{-distance}(p) = \epsilon$$
$$MinPts = 4$$

Definition 7: core-distance eines Objekts p

$$core\text{-distance}_{\epsilon, MinPts}(p) =$$

$$\begin{cases} \text{UNDEFINED, wenn in der } \epsilon\text{-Umgebung von } p \text{ weniger als } MinPts \text{ Objekte liegen} \\ MinPts\text{-distance}(p), \text{ sonst} \end{cases}$$

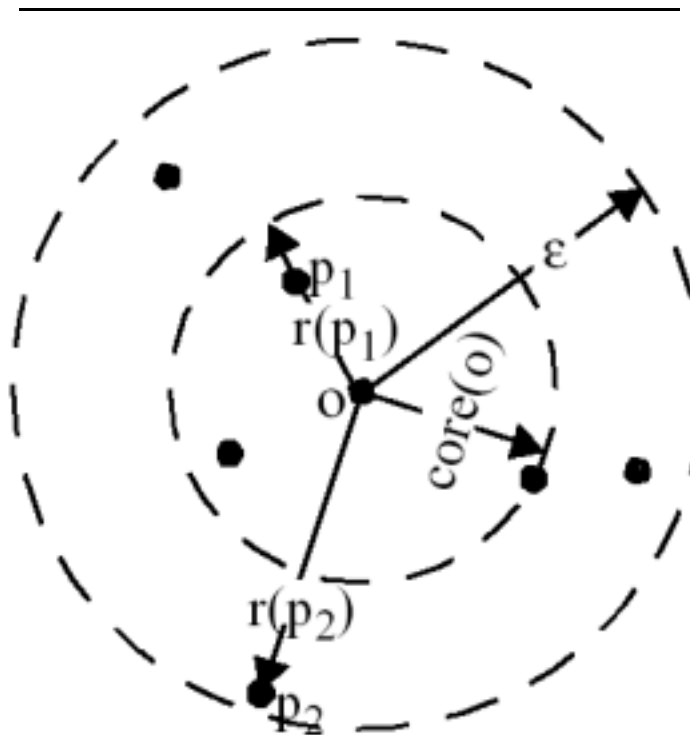
Ist p kein core object, dann ist $core\text{-distance}(p) = \text{UNDEFINED}$.

Definition 8: *reachability-distance* eines Objekts p bzgl. eines Objekts o

$$reachability\text{-}distance_{\varepsilon, MinPts}(p, o) =$$

$$\begin{cases} UNDEFINED, & \text{wenn in der } \varepsilon\text{-Umgebung von } o \text{ weniger als } MinPts \text{ Objekte liegen} \\ \max(core\text{-}distance(o), distance(o, p)), & \text{sonst} \end{cases}$$

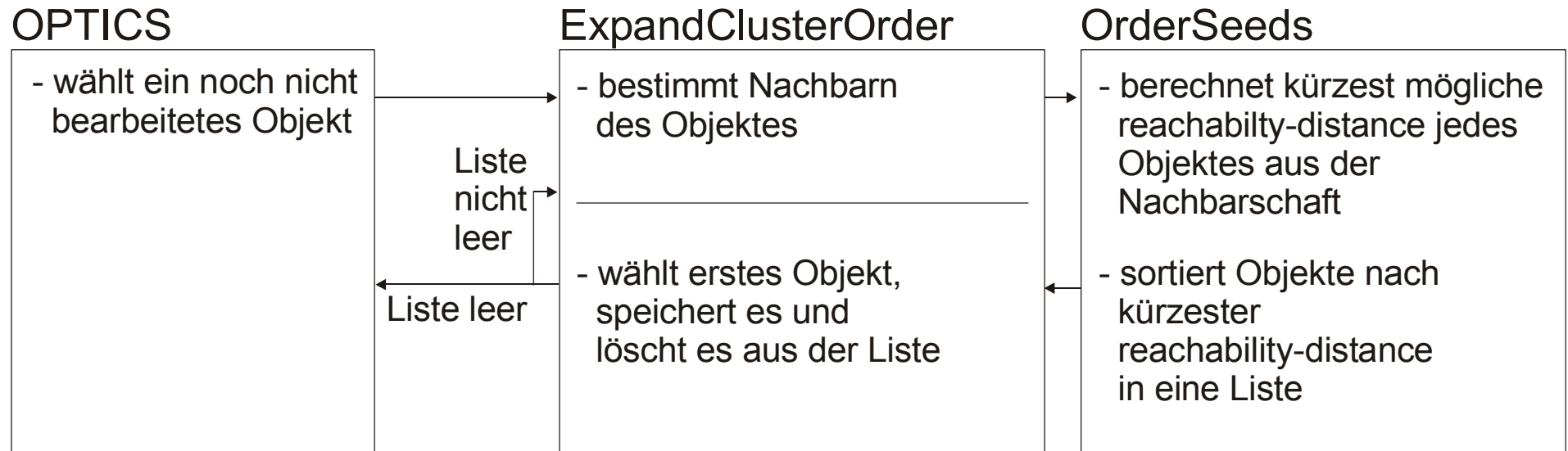
Ist o kein core object, dann ist $reachability\text{-}distance(p, o) = UNDEFINED$.



2.3 Algorithmus OPTICS

- OPTICS erzeugt eine Ordnung der Objekte, die einer auf Dichten basierten Clusterstruktur zugrunde liegt
- Algorithmus besteht aus:
 - Hauptprogramm:
 - Startet das Verfahren für jeden noch nicht bearbeiteten Cluster
 - Prozedur *ExpandClusterOrder*:
 - Bestimmt die Reihenfolge der Objekte des Clusters und speichert diese mit reachability-distance und core-distance
 - Prozedur *OrderSeeds*:
 - Berechnet kürzeste reachability-distance eines Objekts zum nächsten anderen und sortiert die Objekte nach kürzester reachability-distance

Vereinfachte Darstellung des Algorithmus



- OPTICS kann durch entsprechende Indexe stark beschleunigt werden
- Laufzeit liegt grob zwischen $O(n^2)$ und $O(n \log n)$

Hauptprogramm (vereinfacht):

1. Wählt ein noch nicht bearbeitetes **Objekt** aus und übergibt es *ExpandClusterOrder*
2. Falls noch nicht bearbeitete Objekte existieren, weiter bei 1.
→ Alle Objekte der Datenbank werden so bearbeitet

```
1 OPTICS (SetofObjects, ε, MinPts, OrderedFile)
2   OrderedFile.open();
3   FOR i FROM 1 TO SetOfObjects.size DO
4     Object := SetOfObjects.get(i)
5     IF NOT Object.Processed THEN
6       ExpandClusterOrder(SetOfObjects, Object, ε,
                           MinPts, OrderedFile)
7   OrderedFile.close();
8 END; // OPTICS
```

- Bei ausreichend großem ϵ ist die gesamte Datenbank ebenfalls ein großer Cluster:
 - FOR-Schleife läuft nur ein Mal
 - *ExpandClusterOrder* bearbeitet dann alle Objekte
 - Algorithmus läuft länger

Prozedur *ExpandClusterOrder* (vereinfacht):

1. Bestimmt Nachbarn des **Objekt**
2. Terminiert falls **Objekt** kein core object, sonst übergibt **Objekt** und Nachbarn *OrderSeeds*
3. Terminiert falls *OrderSeeds*-Liste leer ist, sonst wählt aus der *OrderSeeds*-Liste das erste Element (**Objekt**) aus
 - a) Bestimmt Nachbarn von **Objekt**
 - b) Speichert **Objekt** mit reachability-distance und core-distance
 - c) Falls **Objekt** core object ist, übergibt **Objekt** und Nachbarn *OrderSeeds*, *OrderSeeds*-Liste wird ggf. erweitert und neu geordnet
 - d) Weiter bei 3.

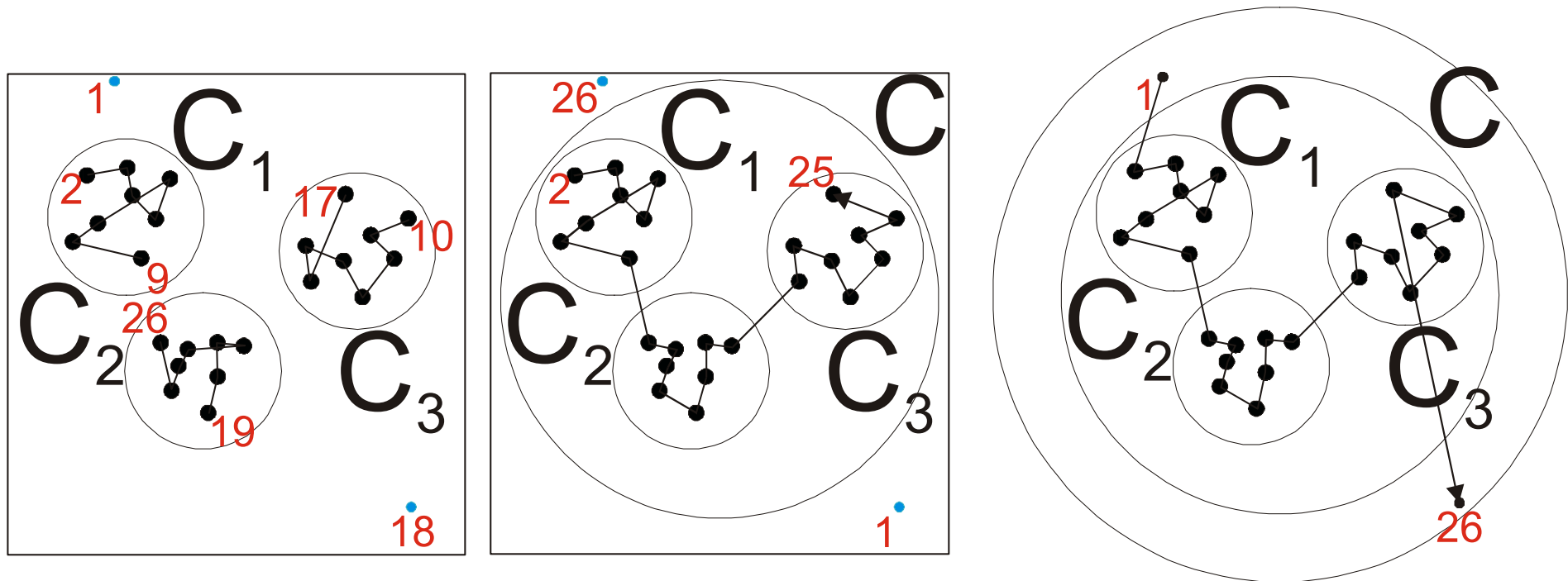
```
1 ExpandClusterOrder(SetOfObjects, Object, ε, MinPts, OrderedFile);
2 neighbors := SetOfObjects.neighbors(Object, ε);
3 Object.Processed := TRUE;
4 Object.reachability_distance := UNDEFINED;
5 Object.setCoreDistance(neighbors, ε, MinPts);
6 OrderedFile.write(Object);
7 IF Object.core_distance <> UNDEFINED THEN
8   OrderSeeds.update(neighbors, Object);
9   WHILE NOT OrderSeed.empty() DO
10    current.Object := OrderSeeds.next();
11    neighbors := SetOfObjects.neighbors(currentObject, ε);
12    currentObject.Processed := TRUE;
13    currentObject.setCoreDistance(neighbors, ε, MinPts);
14    OrderedFile.write(currentObject);
15    IF currentObject.core_distance <> UNDEFINED THEN
16      OrderSeeds.update(neighbors, currentObject);
17 END; // ExpandClusterOrder
```

Prozedur *OrderSeeds* (vereinfacht):

1. Für alle Nachbarn von **Objekt**
2. Bestimmt die reachability-distance zum **Objekt**
 - a. Falls das Objekt noch nicht bearbeitet wurde
 - i. Falls reachability-distance = UNDEFINED:
reachability-distance wird berechnet und Objekt passend in die *OrderSeeds*-Liste eingefügt
 - b. Falls das Objekt bereits bearbeitet wurde
 - i. Falls reachability-distance zum **Objekt** < gespeicherte reachability-distance:
reachability-distance zum **Objekt** wird gespeichert
 - ii. Objekt entsprechend neu in die *OrderSeeds*-Liste einsortiert

```
1 OrderSeeds::update(neighbors, CenterObject)
2   c_dist := CenterObject.core_distance;
3   FORALL Object FROM neighbors DO
4     IF NOT Object.Processed THEN
5       new_r_dist := max(c_dist, CenterObject.dist(Object));
6       IF Object.reachability_distance = UNDEFINED then
7         Object.reachability_distance = new_r_dist;
8         insert(Object, new_r_dist)
9     ELSE // Object already in OrderSeeds
10      IF new_r_dist < Object.reachability_distance THEN
11        Object.reachability_distance = new_r_dist;
12        decrease(Object, new_r_dist);
13  END; //OrderSeeds::update
```

Vereinfachte Darstellung der Ordnung der Objekte



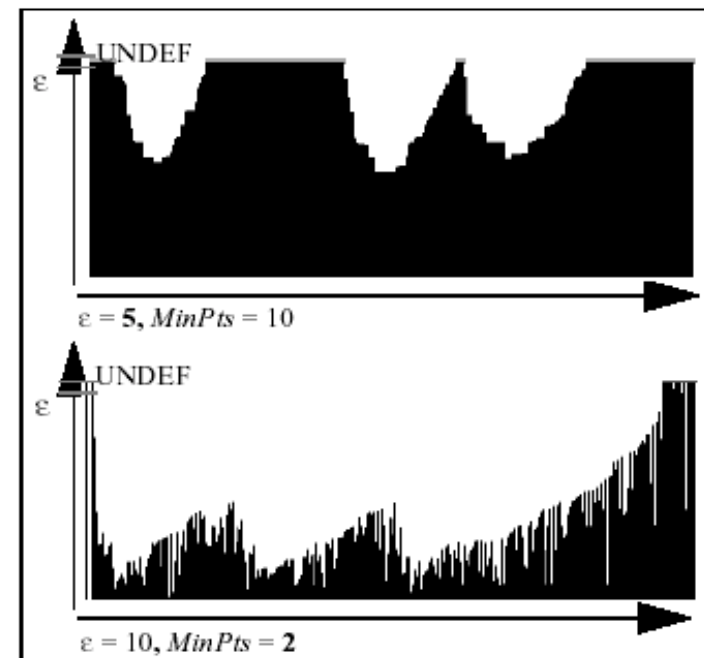
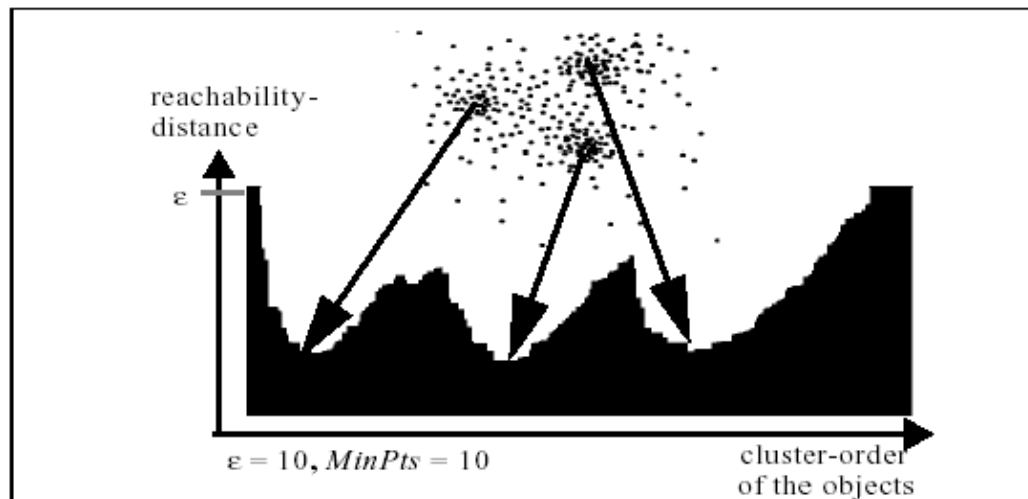
- Bei zu kleinem ε werden große, nicht sehr dichte Cluster nicht erkannt, *ExpandClusterOrder* wird oft verlassen
 - ➔ Ordnung der Objekte ist unterschiedlich und repräsentiert die Clusterstruktur nicht korrekt, Reihenfolge der Cluster stimmt nicht mehr

3. Identifizierung von Clusterstrukturen

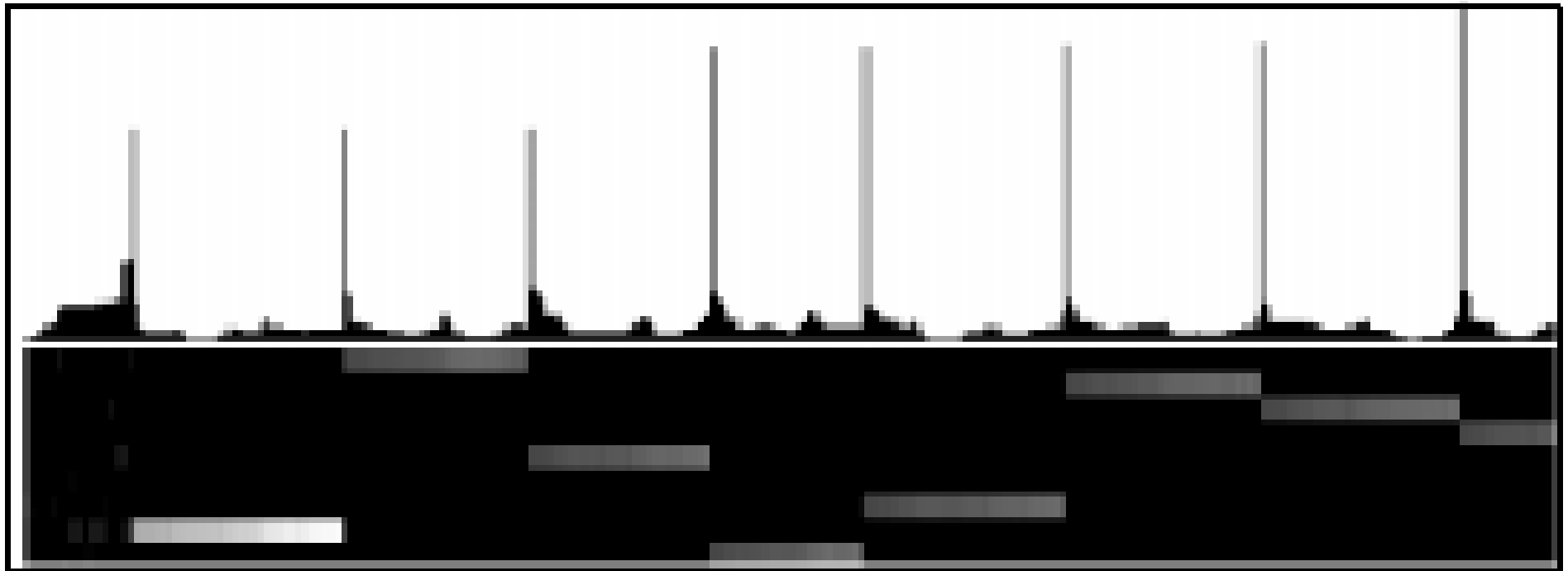
3.1 visuelle Methoden

Reachability plot:

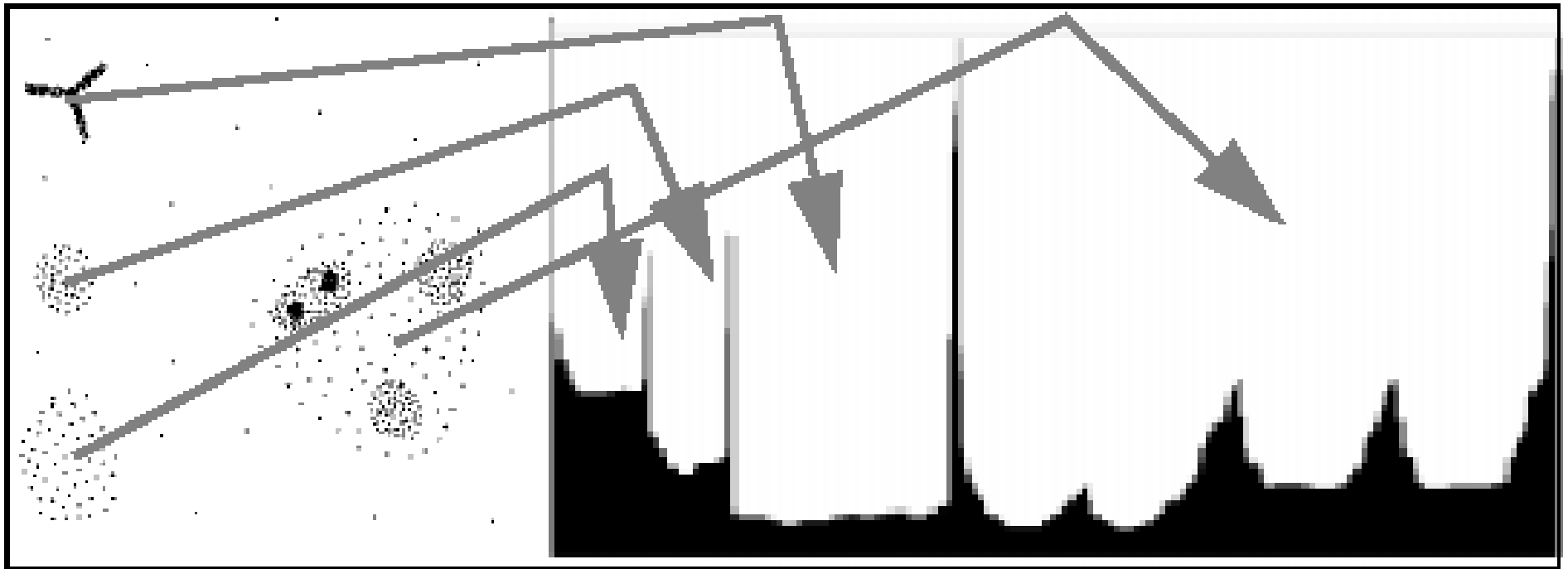
- Objekte werden in der von OPTICS gegebenen Reihenfolge mit ihrer reachability-distance in einem Graphen eingetragen



- Bei zu kleinem ϵ
 - werden große Cluster nicht erkannt
 - wird *ExpandClusterOrder* oft verlassen, OPTICS wählt einen beliebigen neuen Punkt, die Ordnung stimmt nicht mehr (bei UNDEF wird neu angesetzt)
- Größere *MinPts* glätten den Graphen (größere *core-distances* überwiegen kleine *distances*), Werte zwischen 10-20 sind gut (zeigten Experimente)



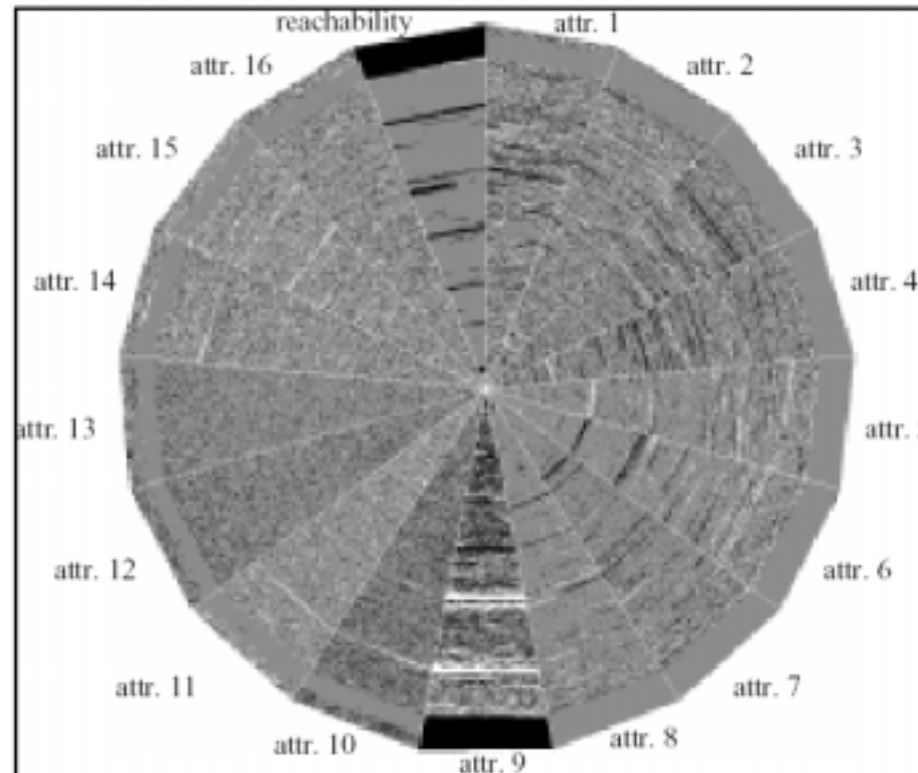
- 9 dimensional Daten von Wetterstationen
- Werte der einzelnen Dimensionen der Objekte sind unter dem Reachability Plot in graustufen dargestellt
- Zusammenhänge zwischen Werten in bestimmten Dimensionen und Clustern können erkannt werden



- Bei großen Datenmengen sind diese mit reachability plots schwer auswertbar der Graph wird
 - sehr lang
 - unübersichtlich

Extended circle segment technique:

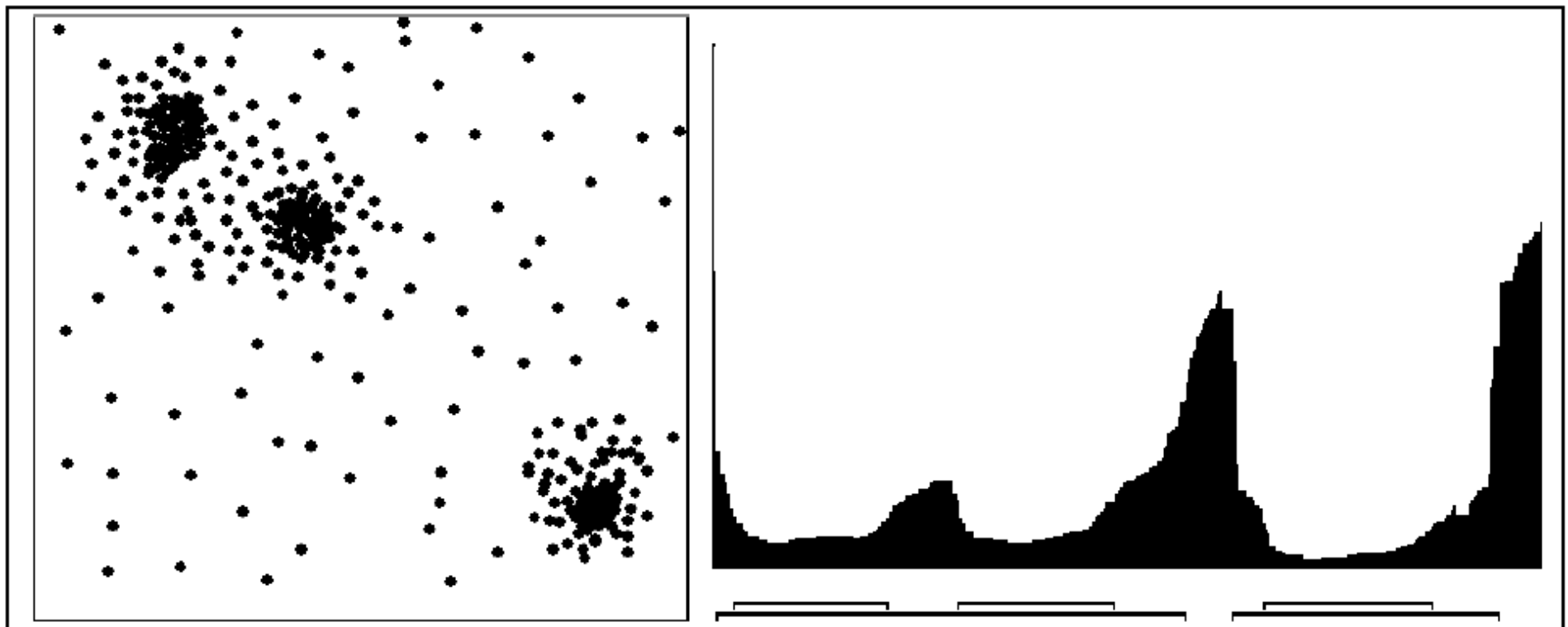
- Kreis wird in Segmente geteilt
- In jedem Segment wird eine Dimension dargestellt
- Jedes Objekt liegt in jedem Segment relativ an der selben Position



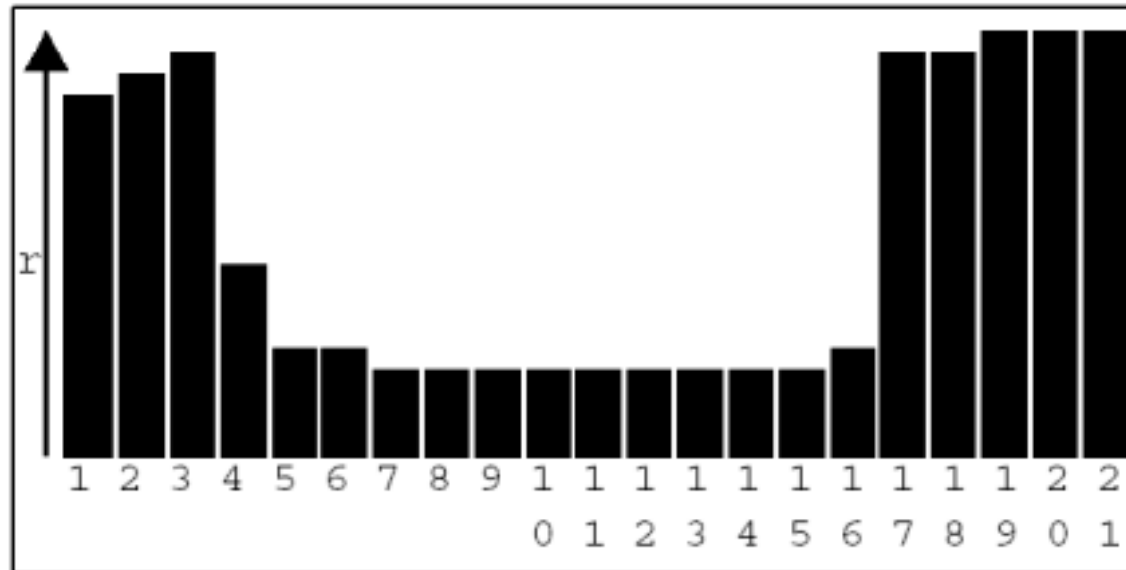
- Kleine Werte haben hier dunkle Werte, z.B. außen befindet sich ein großer Cluster

3.2 automatische Methoden

- Algorithmus *ExtractCluster* ermittelt Intervalle aller erkennbaren Cluster automatisch
- Erspart manuelle Bearbeitung von Reachability plots oder ähnlichem
- Durch eine automatische Auswertung der Ergebnisse von OPTICS können gefundene Cluster direkt mit weiteren Methoden bearbeitet werden

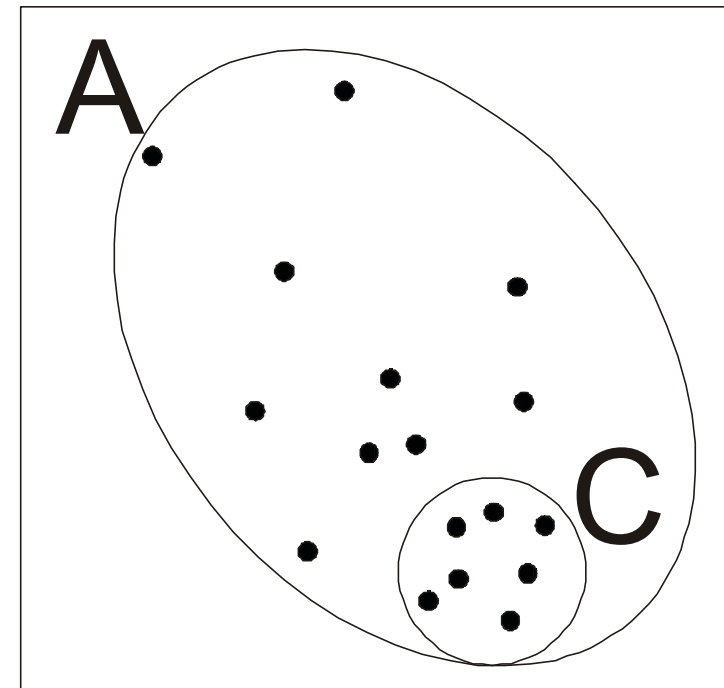
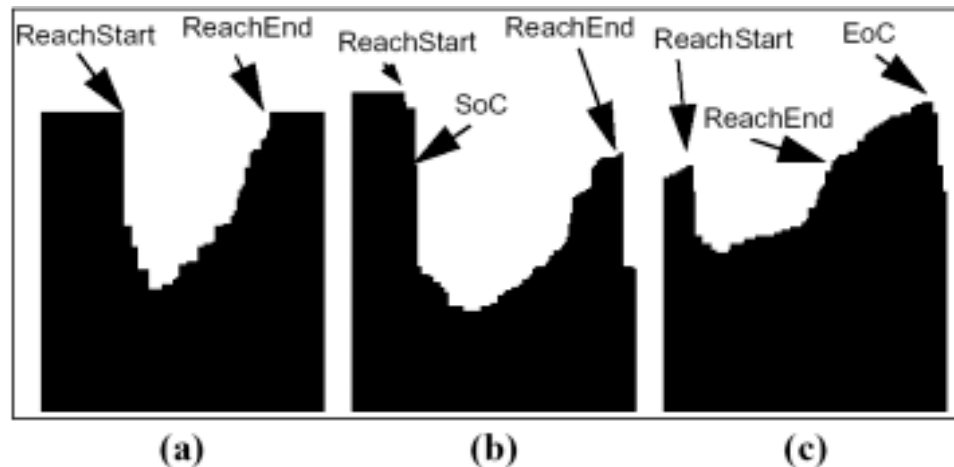


- *ExtractCluster* betrachtet die Objekte in der durch OPTICS gegebenen Ordnung und sucht Objekte mit sinkender / steigender reachability-distance (DownPoint / UpPoint)
- Dabei muss die reachability-distance eines DownPoint bzw. UpPoint min. $\xi\%$ niedriger bzw. höher sein als die des vorherigen Objekts
- Aufeinanderfolgende DownPoints bzw. UpPoints bilden eine „steep down area“ bzw. „steep up area“



- 4 ist ein DownPoint, 17 ist ein UpPoint
- 3 gehört zum Cluster (kurze reachability-distance von 4 aus)

- Cluster beginnen in einer steep down area und enden in einer steep up area
- Cluster beginnen bzw. enden nicht unbedingt am Anfang einer steep down area bzw. am Ende einer steep down area



- Objekte in A bilden Teil einer steep up area (vergleichbar mit (c))
- Cluster C endet nicht am Ende dieser steep up area
- Aufgrund dieser Überlegungen werden ξ -cluster definiert

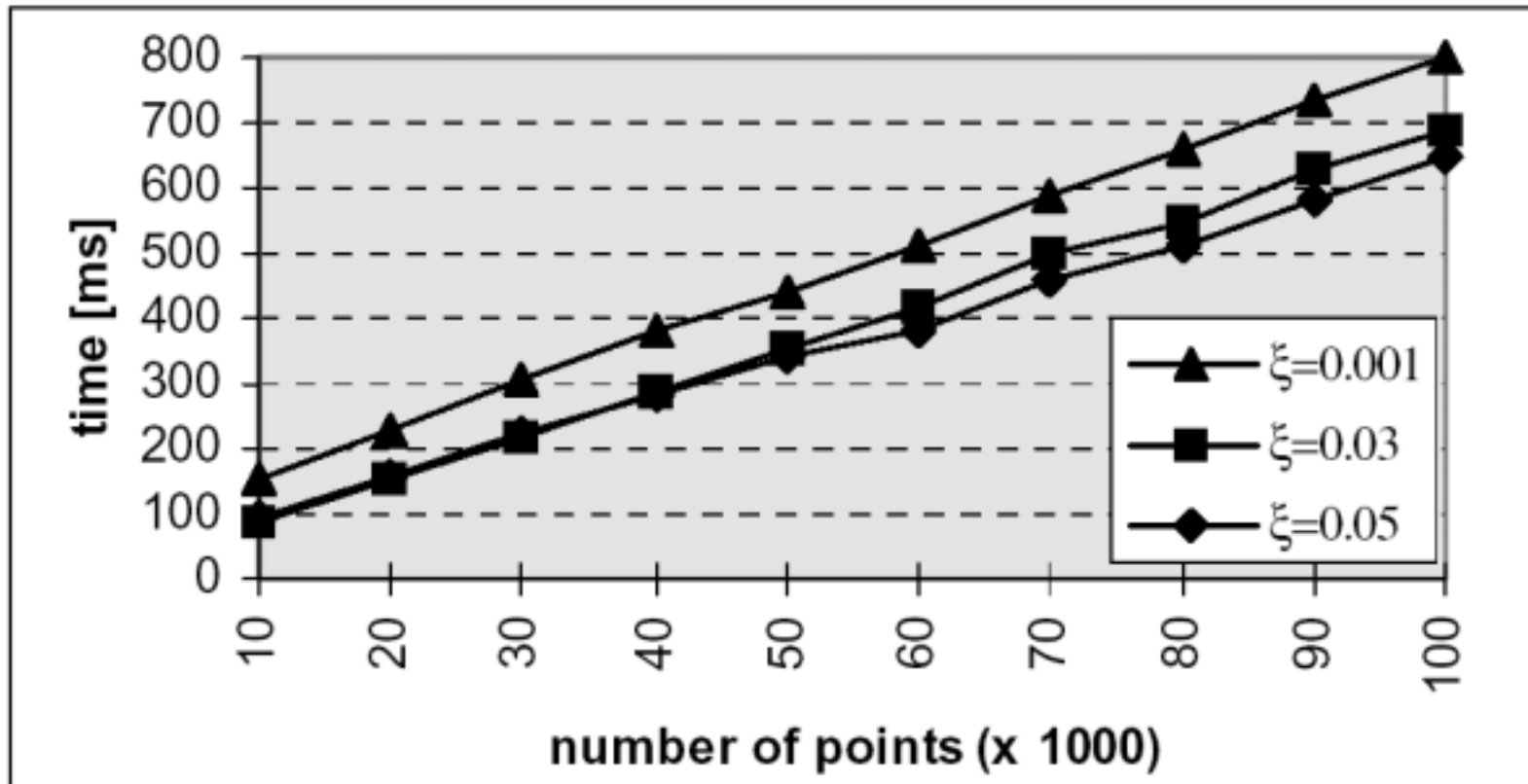
Ineffizienter Algorithmus zum Finden von Clustern:

- Algorithmus geht alle Objekte in der von OPTICS gegebenen Reihenfolge durch und speichert alle gefundenen steep down area mit einem Startindex und einem Endindex in *SDASet*:
 1. *index* := 0
 2. *SDASet* := emptySet
 3. WHILE *index* < *n* DO
 - i. Startet eine neue steep down area bei *index*, füge sie zu *SDASet* hinzu und fahre rechts davon fort
 - ii. Startet eine neue steep up area bei *index*, kombiniere sie mit allen steep down areas aus *SDASet* und prüfe, ob die ξ -cluster-Bedingungen erfüllt sind (u.a. Schleife über alle Objekte des vermuteten Clusters)
Falls ja, berechne Start und Ende und speichere diese Werte des Clusters
 - iii. Inc(*index*)

- Durch Speichern der höchsten reachability-distance seit Ende der steep down areas kann die Schleifen über alle Objekte des Clusters entfernt werden
→ effizienter Algorithmus *ExtractCluster*

```
1  SetOfSteepDownAreas := EmptySet;
2  SetOfClusters := EmptySet;
3  index := 0; mib := 0;
4  WHILE (index < n) DO
5    mib := max(mib, r(index));
6    IF (start of a steep down area D at index)
7      update mib-values and filter SetOfSteepDownAreas(*)
8      set D.mib := 0;
9      add D to the SetOfSteepDownAreas
10   index := end of D + 1; mib := r(index);
11  ELSE
12   IF (start of a steep up area U at index)
13     update mib-values and filter SetOfSteepDownAreas
14     index := end of U + 1; mib := r(index)
15   FOR EACH D in SetOfSteepDownAreas DO
16     IF (combination of D and U is valid AND(**)
17         satisfies cluster conditions 1., 2., 3.a) )
18       compute [s, e] add cluster to SetOfClusters
19   ELSE index := index + 1;
20  RETURN(SetOfClusters);
```

- Laufzeit von *ExtractClusters* auf 64-dimensionalen Farbhistogrammen extrahiert von Fernsehbildern auf einem 180MHz Pentium mit 96MB RAM unter Windows NT 4.0, implementiert mit dem Sun JDK 1.1.6



4. Zusammenfassung

- OPTICS erzeugt eine Ordnung der Objekte einer Datenbank und speichert zu jedem Objekt seine core-distance und reachability-distance
- Damit können Clusterstrukturen mit visuellen oder automatischen Methoden in Datenbanken gefunden werden
- OPTICS verhält sich sehr robust seinen Eingabewerten gegenüber und liefert für viele Werte gute Ergebnisse
- Probleme gibt es jedoch noch bei Datenbanken mit einigen Millionen hochdimensionalen Objekten.
-> Hier existieren (noch) keine Indexstrukturen um OPTICS zu beschleunigen.