

# Seminar Komplexe Objekte in Datenbanken

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](mailto: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>

## OPTICS: Ordering Points To Identify the Clustering Structure

### 1. Einführung

Die Analyse von Datenbanken auf Cluster ist eine primäre Methode beim Database Mining. Sie dient dazu, einen Überblick über die Verteilung der Daten zu erhalten und ggf. auch als Vorbereitungsschritt für weitere Algorithmen, die die erkannten Cluster genauer analysieren.

Algorithmen zum Finden von Clustern benötigen meist mehrere Eingabeparameter. Die Qualität der Ergebnisse variiert sehr stark bei verschiedenen Werten für die Parameter. Es ist besonders bei großen Datenmengen mit hochdimensionalen Objekten schwierig, sinnvolle Werte für diese Parameter zu erkennen. Daher müssen solche Algorithmen mehrfach mit verschiedenen Werten laufen und sind somit sehr ineffizient. Der hier vorgestellte Algorithmus OPTICS erkennt nicht Cluster in Datenbanken, sondern erzeugt eine Ordnung der Datenbankobjekte, die eine auf Dichten basierte Clusterstruktur (density-based cluster) repräsentiert. Anhand dieser Ordnung können Clusterstrukturen mit weiteren automatischen und visuellen Techniken erkannt werden. Der Algorithmus OPTICS verhält sehr robust seinen Eingabeparametern gegenüber und liefert bei vielen Werten der Parameter gute Ergebnisse.

### 2. Ordnung der Datenbankobjekte in Bezug auf die Clusterstruktur

#### 2.1 Motivation

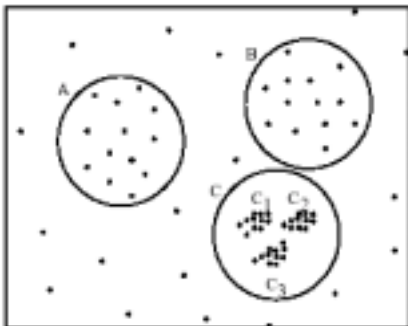


Abbildung 1: Cluster mit verschiedenen Dichteparametern

In realen Datensätzen können Clusterstrukturen nicht durch globale Dichteparameter erkannt werden, da meist Cluster mit den verschiedensten Dichten existieren. Je nach verwendetem Dichteparameter würden in Abbildung 1 nur die Cluster A, B und C oder nur die Cluster C1, C2 und C3 entdeckt, aber nicht A, B, C1, C2 und C3. Daher müsste ein Algorithmus mit einem globalen Dichteparameter mit fast unendlich vielen verschiedenen Werten angewendet werden, um die verschiedenen Cluster zu finden.

Ein Algorithmus (wie OPTICS), der eine Ordnung der Datenbankobjekte in Bezug auf die dichtenbasierte Clusterstruktur erzeugt und dabei Informationen über die Objekte in den verschiedenen Clusterleveln bzw. -größen speichert, liefert Informationen, mit denen eine Analyse der Clusterstruktur sehr einfach ist.

#### 2.2 Definitionen

In einem auf Dichten basierenden Cluster hat jedes Objekt eines Clusters innerhalb einer Umgebung, die durch einen Radius der Größe  $\epsilon$  gegeben ist, eine Mindestanzahl (*MinPts*) von Nachbarobjekten. Die Größe der Nachbarschaft muss also eine Mindestgröße haben. Aufgrund dieser Idee kann folgendes definiert werden:

##### Definition 1: core object

$p$  ist ein core object  $\Leftrightarrow$  In der durch den Radius  $\epsilon$  gegebenen Umgebung ( $\epsilon$ -Umgebung) um  $p$  liegen min. *MinPts* Objekte (incl.  $p$  selbst).

##### Definition 2: directly density-reachable

Objekt  $p$  ist directly density-reachable von Objekt  $q$  aus bzgl. der Parameter  $\epsilon$  und *MinPts*

$\Leftrightarrow$

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

##### Definition 3: density-reachable

Objekt  $p$  ist density-reachable von Objekt  $q$  aus bzgl. der Parameter  $\epsilon$  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. der Parameter  $\epsilon$  und *MinPts*.

Density-reachability ist die transitive Hülle von directly density-reachability. Nur core objects sind in beide Richtungen density-reachable. Ein Beispiel ist in Abbildung 2 dargestellt.

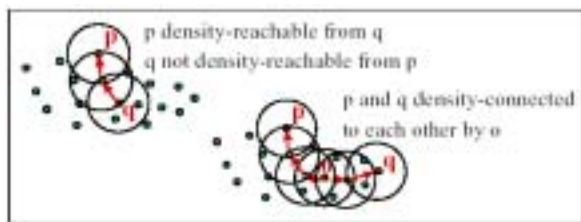


Abbildung 2: density-reachable und density-connected

**Definition 4: density-connected**

Objekt p ist density-connected mit Objekt q bzgl. der Parameter  $\epsilon$  und  $MinPts$

$\Leftrightarrow$

- 1) Es existiert ein Objekt o
- 2) p und q sind density-reachable von o aus bzgl. der Parameter  $\epsilon$  und  $MinPts$ .

q ist also ein core object, density-connected ist eine sym-

metrische Eigenschaft.

**Definition 5: cluster und noise**

C ist ein Cluster bzgl. der Parameter  $\epsilon$  und  $MinPts$

$\Leftrightarrow$

- 1) Es existiert min. ein Objekt in C
- 2) (Maximality:) für alle Objekte p, q gilt:

Wenn  $p \in C$  und q ist density-reachable von p aus bzgl. der Parameter  $\epsilon$  und  $MinPts$   
 $\Rightarrow q \in C$

- 3) (Connectivity:) für alle Objekte p, q in C gilt:

p ist density-connected von q aus bzgl. der Parameter  $\epsilon$  und  $MinPts$

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

Ein Cluster enthält nicht nur core objects. Alle nicht core objects heißen „border objects“ des Clusters und sind directly density-reachable von einem core object aus dem Cluster.

**Definition 6:  $MinPts$ -distance(p)**

$MinPts$ -distance des Objektes p ist die Entfernung bis zum  $MinPts$ -ten Objekt. Dabei wird p mitgezählt. Es wird immer bei dem am nächsten zu p liegende Objekt weitergezählt, das noch nicht gezählt wurde (vgl. Abb. 3, core-distance).

**Definition 7: core-distance eines Objekts p**

$$core-distance_{\epsilon, MinPts}(p) =$$

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

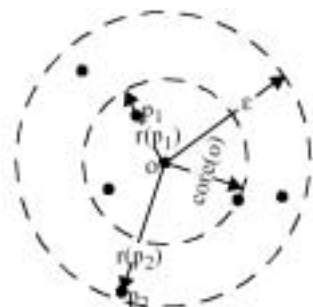


Abbildung 3: Beispiele für core-/reachability-distance

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

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

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

Liegt o außerhalb der  $\epsilon$ -Umgebung von p, so ist die  $reachability-distance_{\epsilon, MinPts}(p, o)$  gleich der Distanz zwischen p und o. Wenn p außerhalb der  $\epsilon$ -Umgebung von o liegt, dann ist sie gleich der  $core-distance(o)$ . Ist o kein core object, ist sie immer UNDEFINED.

In Abbildung 3 ist daher die  $core-distance_{\epsilon, MinPts}(o)$  ( $core(o)$ ) gleich der  $reachability-distance_{\epsilon, MinPts}(p_1, o)$  ( $r(p_1)$ ). Die  $reachability-distance_{\epsilon, MinPts}(p_2, o)$  ( $r(p_2)$ ) ist aber gleich der Distanz der Objekte p und o.

**2.3 Algorithmus OPTICS**

Der Algorithmus OPTICS erzeugt eine Ordnung der Datenbankobjekte und speichert dabei für jedes Objekt die  $core-distance$  und die  $reachability-distance$  zum nächsten core object des Clusters.

```

1 OPTICS (SetofObjects,  $\epsilon$ , 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,  $\epsilon$ ,
                           MinPts, OrderedFile)
7   OrderedFile.close();
8 END; // OPTICS

```

Abbildung 4: Algorithmus OPTICS

Hauptsächlich übergibt der Algorithmus jedes Objekt der Datenbank, welches noch nicht behandelt wurde (Zeile 5), der Prozedur ExpandClusterOrder (Zeile 8). Außerdem werden alle Objekte der Datenbank ( $SetOfObjects$ ), der Radius  $\epsilon$ ,  $MinPts$  und die Datei ( $OrderedFile$ ) übergeben, in der die Ordnung gespeichert wird.

```

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

```

Abbildung 5: Prozedur ExpandClusterOrder

Die Prozedur ExpandClusterOrder arbeitet wie folgt:

- 1) Alle Objekte in der  $\epsilon$ -Umgebung von *Object* werden in *neighbors* gespeichert (Zeile 2), das *Object* wird als bearbeitet markiert (Zeile 3).
- 2) Die reachability-distance des *Objects* wird auf UNDEFINED gesetzt (Zeile 4) und die core-distance wird bestimmt.
- 3) Das *Object* wird mit der core-distance und reachability-distance gespeichert.
- 4) Falls *Object* kein core object ist, endet ExpandClusterOrder und OPTICS wahlt das nachste Objekt aus (Zeile 8). Andernfalls:
  - a. Die Objekte aus der  $\epsilon$ -Umgebung von *Object* werden in die Seedliste (Prozedur OrderSeeds.update) aufgenommen (Zeile 9). Diese verwaltet und sortiert die Seedliste anhand der reachability-distances der Objekte.
  - b. Solange die Seedliste nicht leer ist (Zeile 10):
    - i. Das erste Objekt in der Seedliste wird das aktuelle Objekt (currentObject). (Zeile 11).
    - ii. Die Objekte aus der  $\epsilon$ -Umgebung werden in neighbors gespeichert (Zeile 12).
    - iii. currentObject wird als bearbeitet markiert. (Zeile 13).
    - iv. Die core-distance des currentObject wird ermittelt (Zeile 14)
    - v. currentObject wird zusammen mit seiner core-distance und der durch OrderSeeds.update ermittelten reachability-distance in der Liste gespeichert (Zeile 15).
    - vi. Ist currentObject ein core object werden in die Seedliste erneut die Objekte aus der  $\epsilon$ -Umgebung von currentObject aufgenommen (Zeile 16 und 17)

OrderSeed.update verwaltet und sortiert die Seedliste, dabei werden die Objekte in der Reihenfolge der reachability-distance zum jeweils nachsten core object sortiert.

```

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

```

Abbildung 6: Prozedur OrderSeeds::update

Der Algorithmus berechnet dabei zuerst die core-distance des gegebenen *CenterObject*. Danach wird fur jedes Objekt in der  $\epsilon$ -Umgebung des *CenterObject* (Zeile 3), das noch nicht betrachtet wurde (Zeile 4), die reachability-distance zum *CenterObject* bestimmt (Zeile 5). Wurde eine reachability-distance dieses Objekts noch nicht bestimmt (UNDEFI-

NED) oder war UNDEFINED (Zeile 6), so wird das Objekt an der aktuellen Position der Seedliste eingefugt (Zeile 7). Wurde eine reachability-distance (zu einem anderen core object) bereits berechnet (Zeile 9), so wird die reachability-distance zum aktuellen *centerObject* fur das Objekt verwendet, wenn sie kleiner ist als die bisher vermerkte reachability-distance zu einem anderen core object (Zeile 10). Ist dies der Fall, wird das Objekt entsprechend weiter an den Anfang der Seedliste verschoben (Zeile 12).

OPTICS beginnt also bei einem Objekt in der Datenbank und betrachtet nach und nach immer die Objekte, die am nachsten zusammenliegen und noch nicht betrachtet wurden. Der Algorithmus wandert dabei zunachst in der aktuellen Region, in die Cluster mit besonders hohen Dichten und bearbeitet dann die Gegend darum herum, bis die Objekte zu

weit entfernt sind, um noch zu einem großen Cluster zu gehören. Die Seedsliste ist dann leer, die WHILE-Schleife in ExpandClusterOrder wird beendet. Danach wird ein beliebiges noch nicht betrachtetes Objekt ausgewählt und die Suche nach nahe liegenden Objekten und Clustern in der Gegend wird fortgesetzt. Dies wird so lange fortgesetzt, bis alle Objekte betrachtet wurden. Jedes Mal, wenn ein Objekt betrachtet wurde, wird es mit der reachability-distance und der core-distance gespeichert. Somit sind Objekte, die besonders dicht zusammenliegen auch in der Ordnung besonders dicht zusammen.

### 3. Identifizierung von Clusterstrukturen

#### 3.1 visuelle Methoden

Da OPTICS eine Ordnung der Objekte erzeugt und zu jedem Objekt u.a. die reachability-distance zum nächsten core object speichert, können die Objekte in der gegebenen Reihenfolge auf der x-Achse und mit ihrer reachability-distance auf der y-Achse in einem Graphen dargestellt werden. Ein solcher reachability plot ist in der Abbildung 7 dargestellt. Die Abbildung 8 zeigt, wie sich die Anzahl der Cluster der verschiedenen Clusterleveln/-dichten bei verschiedenen Radien  $\epsilon$  verändert, je größer  $\epsilon$ , um so mehr Cluster mit geringen Dichten werden erkannt. Im unteren Graph ist erkennbar, dass der Graph gezackt aussieht, wenn  $MinPts$  kleiner gewählt wird. Bei größeren  $MinPts$  ist der Graph glatter. Abbildung 9 zeigt den reachability plot einer Datensammlung mit Clustern verschiedener Dichten und Formen.

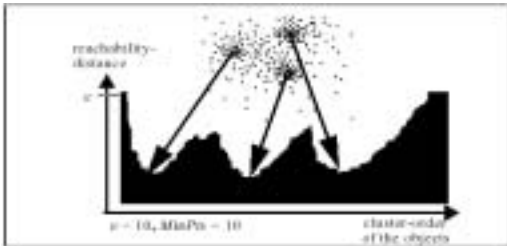


Abbildung 7: reachability plot

Formen.

Reachability plots sind eine sehr intuitive Methode um Clusterstrukturen in Datenbanken zu erkennen. Die Form der Graphen ist recht unempfindlich gegenüber den Parametern  $\epsilon$  und  $MinPts$ .  $\epsilon$  muss nur groß genug gewählt werden um



Abbildung 9: reachability plot mit Clustern verschiedener Dichten und Formen

auch weniger dichte Cluster zu erkennen, in Experimenten haben Werte zwischen 10 und 20 für  $MinPts$  gute Ergebnisse geliefert.

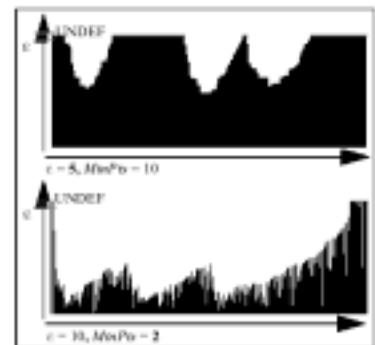


Abbildung 8: reachability plots mit verschiedenen Parametern

#### Reachability

plots sind bei der Darstellung in der Anzahl von Objekten und Dimensionen beschränkt, sehr lange Graphen sind schließlich schwer auswertbar. Um große Datenmengen mit hohen Dimensionen visuell zu untersuchen, kann z.B. die extended circle segment technique angewendet werden. Bei dieser Methode wird in einem Kreis je eine Dimension in einem Segment des Kreises abgebildet. Diese Segmente teilen den Kreis wie einen Kuchen auf. In einem zusätzlichen Segment kann die reachability-distance dargestellt werden. Innerhalb der Segmente stellen Pixel die jeweilige Dimension eines Objekts dar. Jedes Objekt ist in allen Segmenten relativ an der gleichen Stelle. D.h. ein Objekt, welches im Segment 1 ganz links außen in der ersten Dimension dargestellt wird, wird im Segment  $i$  ganz links außen in der  $i$ . Dimension dargestellt. Den möglichen Werten der einzelnen Dimensionen werden dabei Farben zugeordnet. Experimente haben ergeben, dass sich Grautöne am Besten eignen, um hierarchische Cluster zu erkennen. In der Abbildung 10 wurden 30.000 16-dimensionale Objekte dargestellt. Die Objekte wurden in Kreisen von innen nach außen angeordnet. Eines der Segmente zeigt die reachability-distance der Objekte. Hohe Werten wurden helle Farben und niedrigen Werten dunkle Farben zugeordnet. Der sehr dunkle äußere Bereich stellt also einen dichten Cluster dar, die reachability-distance der Objekte sind sehr klein. Um den Cluster herum existieren einige Objekte, die noise sind. Sie sind durch die weiße Linie unter dem schwarzen Bereich dargestellt. Die einzelnen schwarzen Bereiche zur Kreismitte repräsentieren kleinere aber dichte Cluster. In den anderen Segmenten kann nach ähnlichen Werten in verschiedenen Dimensionen bei Objekten in und außerhalb von Clustern gesucht werden.

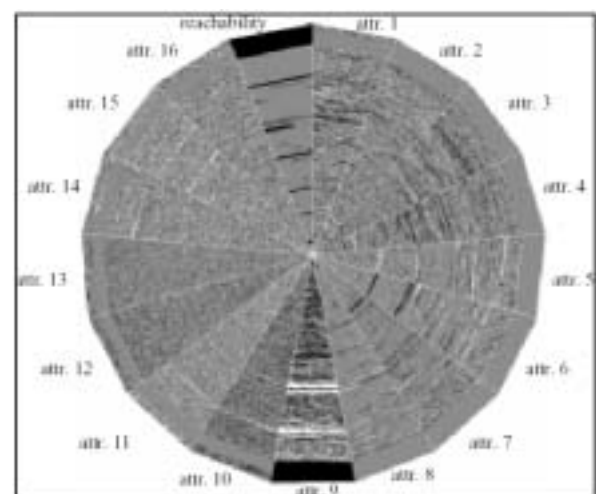


Abbildung 10: Clusterstruktur von 30.000 16-dim Objekten

### 3.2 automatische Methoden

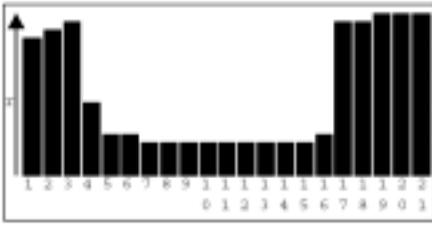


Abbildung 11: Cluster

Um Cluster mit automatischen Methoden erkennen zu können, wird eine Definition von Clustern benötigt, die auf den Ergebnissen der Algorithmus OPTICS basiert. Abbildung 11 zeigt einen Cluster, der bei Punkt 3 beginnt und bei Punkt 16 endet. Aufgrund der Funktionsweise von OPTICS hat Punkt 3 eine hohe reachability-distance in Bezug zu Punkt 1 und 2 aber Punkt 4 hat eine kurze reachability-distance zu Punkt 3, daher ist der Punkt 3 im Cluster enthalten. Punkt 17 ist nicht im Cluster enthalten, da er eine hohe reachability-distance zu den vorigen Punkten hat. Cluster in realen Daten beginnen und enden oft nicht mit so extremen Stufen.

Aufgrund der folgenden Definition können verschiedene Stufenhöhen bestimmt werden:

**Definition 9:  $\xi$ -steep points**

Ein Punkt  $p \in \{1 \dots n-1\}$  heißt  $\xi$ -steep upward point, wenn er  $\xi\%$  niedriger ist als sein Nachfolger:

$$UpPoint_{\xi}(p) \Leftrightarrow r(p) \leq r(p+1) * (1-\xi)$$

Analog ist der  $\xi$ -steep downward point definiert:

$$DownPoint_{\xi}(p) \Leftrightarrow r(p) * (1-\xi) \leq r(p+1)$$

Alle Cluster beginnen mit einigen downward points und enden mit einigen upward points. Diese Regionen (steep areas) beginnen mit  $\xi$ -steep points auf die einige Punkte folgen, die keine  $\xi$ -steep points sind, deren reachability-distance aber auch steigt bzw. sinkt oder auf gleichem Level bleibt. Eine  $\xi$ -steep upward area beginnt und endet mit einem  $\xi$ -steep upward point. Dazwischen steigen die Punkte ebenfalls oder bleiben auf dem selben Level. Eine  $\xi$ -steep area enthält maximal  $MinPts$  aufeinander folgende Punkte, die keine  $\xi$ -steep points sind. Andernfalls wäre diese Region ein eigener Cluster. Außerdem sind steep areas maximal. Formal sind steep areas folgendermaßen definiert:

**Definition 10:  $\xi$ -steep areas**

Ein Intervall  $I = [s, e]$  heißt  $\xi$ -steep upward area ( $UpArea_{\xi}(I)$ )

$\Leftrightarrow$

1.  $UpPoint_{\xi}(s)$
2.  $UpPoint_{\xi}(e)$
3. jeder Punkt zwischen  $s$  und  $e$  ist mindestens so hoch wie sein Vorgänger
4. es existieren nicht mehr als  $MinPts$  aufeinanderfolgende nicht  $\xi$ -steep upward points
5.  $I$  ist maximal, d.h.:  
 $\forall J: (I \subseteq J, UpArea_{\xi}(J) \Rightarrow I = J)$

$\xi$ -steep downward area ( $DownArea_{\xi}(I)$ ) ist analog definiert.

Basierend auf diesen Definitionen können nun  $\xi$ -cluster wie folgt definiert werden:

**Definition 11:  $\xi$ -cluster**

Ein Intervall  $C = [s, e] \subseteq [1..n]$  ist ein  $\xi$ -cluster, wenn die folgenden 4 Bedingungen erfüllt sind:

- cluster $_{\xi}(c) \Leftrightarrow \exists D = [s_D, e_D], U = [s_U, e_U]$  mit
  1.  $DownArea_{\xi}(D) \wedge s \in D$
  2.  $UpArea_{\xi}(U) \wedge e \in U$
  3. a)  $e - s \geq MinPts$   
 b)  $\forall x, s_D < x < e_U : (r(x) \leq \min(r(s_D), r(e_U)) * (1-\xi))$
4.  $(s, e) = \begin{cases} (\max\{x \in D \mid r(x) > r(e_U + 1)\}, e_U), & \text{falls } r(s_D) * (1-\xi) \geq r(e_U + 1) \text{ (b)} \\ (s_D, \min\{x \in U \mid r(x) < r(s_D)\}), & \text{falls } r(e_U + 1) * (1-\xi) \geq r(s_D) \text{ (c)} \\ (s_D, e_U), & \text{sonst (a)} \end{cases}$

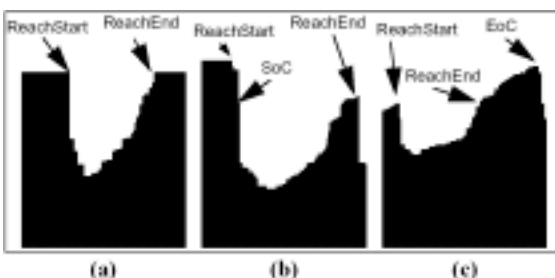


Abbildung 12: Drei verschiedene Cluster, ReachStart =  $s_D$ , ReachEnd =  $e_U$ , SoC = Clusterbeginn, EoC = Clusterende

Ein  $\xi$ -cluster beginnt also innerhalb einer DownArea (1.), endet innerhalb einer UpArea (2.) und enthält mehr als  $MinPts$  Objekte (3. a). Die reachability-distance aller Objekte im Cluster ist mindestens  $\xi\%$  kleiner, als die des ersten Objekts des Cluster und die des ersten Objekts nach dem Cluster (3. b). Die vierte Bedingung bestimmt Start und Ende des Clusters:

- (a) Liegen  $s_D$  und  $e_U$  nicht mehr als  $\xi\%$  auseinander, dann beginnt der Cluster bei  $s_D$  und endet bei  $e_U$ .
- (b) Liegt  $r(s_D)$   $\xi\%$  höher als  $r(e_U)$ , dann endet der Cluster bei  $e_U$  und beginnt bei dem Objekt in  $D$ , dass annähernd den selben Wert wie  $e_U$  hat.
- (c) Analog falls  $r(e_U)$   $\xi\%$  höher als  $r(s_D)$  liegt.

Mit diesen Definitionen kann ein zunächst ineffizienter, aber einfacher Algorithmus die  $\xi$ -cluster finden:

Der Algorithmus geht alle Objekte in der von OPTICS gegebenen Reihenfolge durch und speichert dabei alle gefundenen steep down areas mit einem Startindex und einem Endindex in *SDASet*:

1.  $index := 0$
2.  $SDASet := \text{emptySet}$
3. WHILE  $index < n$  DO
  - a. Startet eine neue steep down area bei  $index$ , füge sie zu *SDASet* hinzu und fahre rechts davon fort.
  - b. Startet eine neue steep up area bei  $index$ , kombiniere sie mit allen steep down areas aus *SDASet* und prüfe, ob die  $\xi$ -cluster-Bedingungen erfüllt sind. Falls ja, berechne  $s$  und  $e$  (4. Bedingung der Definition) und speichere den Cluster mit  $(s, e)$ .
  - c.  $Inc(index)$

Offensichtlich findet der Algorithmus alle Cluster, ist aber noch sehr ineffizient. Durch Umformungen der Bedingung 3. b) der Definition können die meisten Clusterkandidaten herausgefiltert werden, die keine Cluster werden und die Schleife über alle Objekte eines Clusters kann entfernt werden:

Um die neuen Bedingungen (sc1\*) und (sc2\*) anzuwenden, speichert der zu ExtractClusters veränderte Algorithmus jeweils den maximalen Wert (maximum in between,  $mib$ ) zwischen dem Ende jeder steep down area und dem aktuellen  $index$  mit in *SDASet*. Weiterhin wird ein globaler  $mib$ -Wert (der maximale Wert) zwischen dem Ende der letzten steep down oder up area und dem aktuellen  $index$  in  $mib$  gespeichert. Sobald beim Durchlaufen der Objekte eine neue steep down oder up area gefunden wird, werden alle steep down areas aus *SDASet*, für die gilt „Startwert\*( $1-\xi$ ) < globaler  $mib$ -Wert“, gefiltert. Somit wird die Anzahl der möglichen Cluster erheblich reduziert und (sc1\*) erfüllt (Zeile 7). In Zeile 16 wird der „Endwert der steep up area\*( $1-\xi$ )“ mit dem  $mib$ -Wert ( $D.mib$ ) der aktuellen steep down area  $D$  verglichen. Somit wird die Bedingung (sc2\*) erfüllt.

$$\forall x, s_D < x < e_U : (r(x) \leq \min(r(s_D), r(e_U)) * (1-\xi))$$

$$\Leftrightarrow$$

$$(sc1) \forall x, s_D < x < e_U : (r(x) \leq r(s_D) * (1-\xi)) \wedge$$

$$(sc2) \forall x, s_D < x < e_U : (r(x) \leq r(e_U) * (1-\xi))$$

$$\Leftrightarrow$$

$$(sc1^*) \max \{x \mid s_D < x < e_U\} \leq r(s_D) * (1-\xi) \wedge$$

$$(sc2^*) \max \{x \mid s_D < x < e_U\} \leq r(e_U) * (1-\xi)$$

```

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);

```

Abbildung 14 zeigt die Laufzeit des Algorithmus ExtractCluster auf 64-dimensionalen Farbhistogrammen, die von Fernsehbildern extrahiert wurden. Der Algorithmus lief auf einem 180 MHz Pentium Pro mit 96MB RAM unter Windows NT 4.0 und wurde in Java mit dem Sun JDK 1.1.6 implementiert.

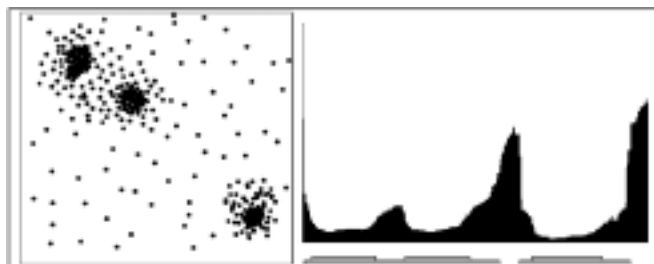


Abbildung 15: Daten (links), reachability plot (rechts oben) und  $\xi=0.09$  Clusterstruktur darunter

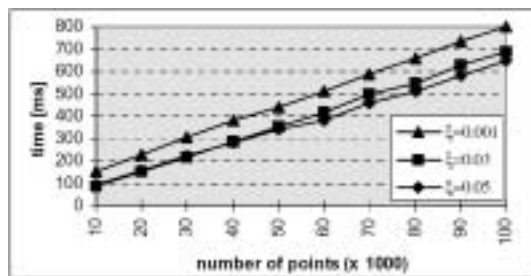


Abbildung 14: Laufzeit von ExtractClusters

Abbildung 15 zeigt links synthetische 2-dimensionale Daten und rechts den reachability plot und im Vergleich dazu darunter die von ExtractClusters markierten Cluster.

## 4. Zusammenfassung

Der Algorithmus OPTICS erzeugt eine Ordnung der Objekte einer Datenbank und speichert zu jedem Objekt seine core-distance und reachability-distance. Mit diesen Informationen können Clusterstrukturen mit visuellen oder automatischen Methoden in den Daten gefunden werden. OPTICS verhält sich dabei sehr robust seinen Eingabewerten gegenüber und liefert für sehr 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.