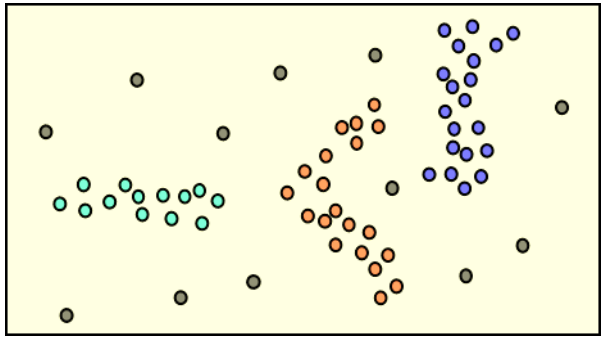
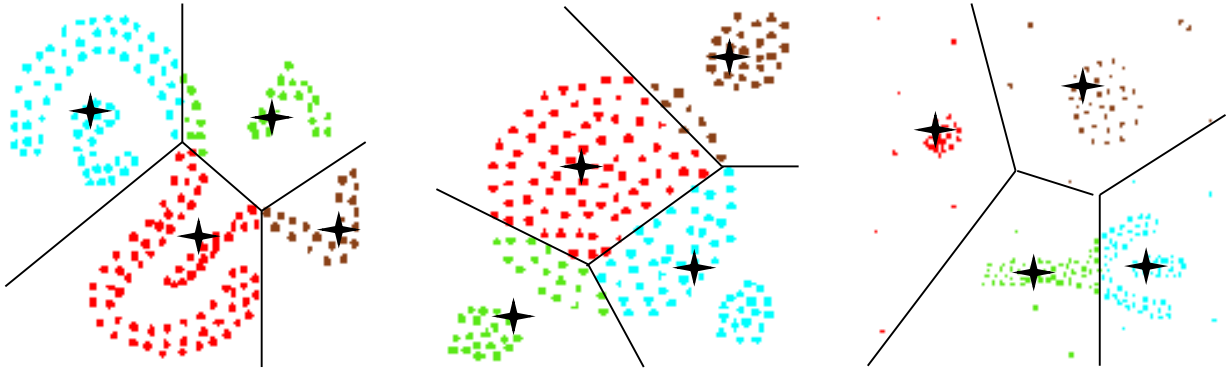
Contents

Density-Based Clustering

- Basic Idea:
  - Clusters are dense regions in the data space, separated by regions of lower object density
- Why Density-Based Clustering?





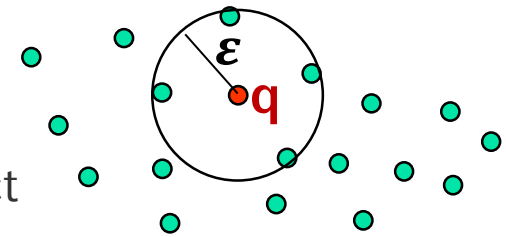
Results of a *k*-medoid algorithm for *k*=4

- Different density-based approaches exist (see Textbook & Papers) Here we discuss the ideas underlying the DBSCAN algorithm and one of its variants

Density-Based Clustering: Basic Concept

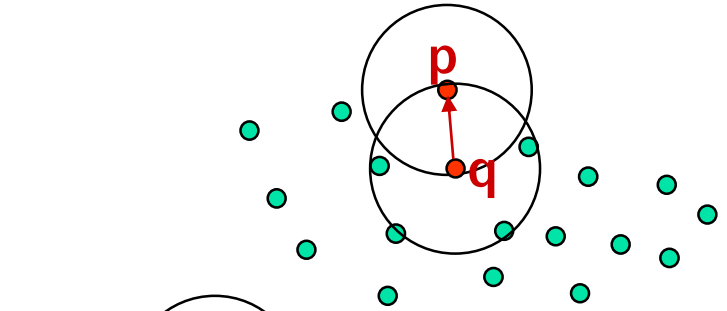
- Intuition for the formalization of the basic idea
  - For any point in a cluster, the local point density around that point has to exceed some threshold
  - The set of points from one cluster is spatially connected
- Local point density at a point q defined by two parameters
  - ε–radius for the neighborhood of point q:
    $N_\varepsilon(q) := \{p \in D | dist(p,q) \leq \varepsilon\}$ *! contains q itself !*
  - ***MinPts*** – minimum number of points in the given neighbourhood $N_\varepsilon(q)$
- q is called a ***core object*** (or core point) w.r.t. ε, *MinPts* if $| N_\varepsilon(q) | \geq MinPts$

MinPts = 5 → **q** is a core object

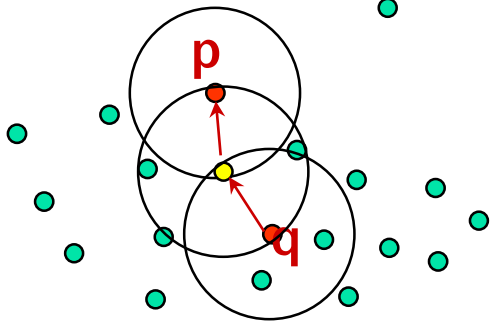


Density-Based Clustering: Basic Definitions

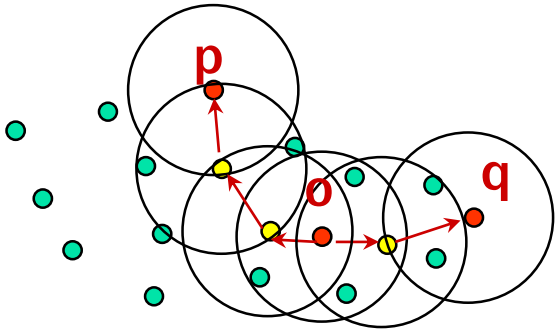
- *p* ***directly density-reachable*** from *q* w.r.t. ε, *MinPts* if
  1) $p \in N_\varepsilon(q)$ and
  2) *q* is a core object w.r.t. ε, *MinPts*



- ***density-reachable***: transitive closure of *directly* density-reachable



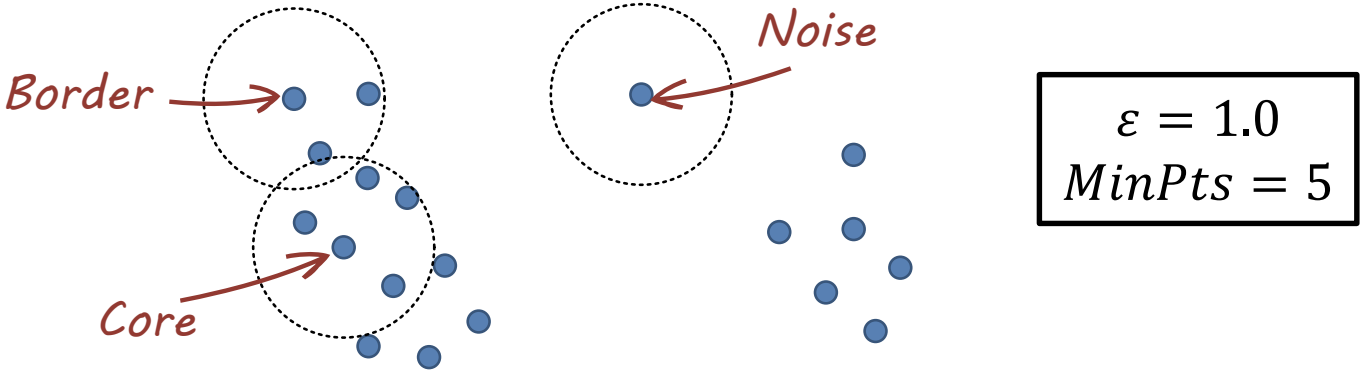
- *p* is ***density-connected*** to a point *q* w.r.t. ε, *MinPts* if there is a point *o* such that both, *p* and *q* are density-reachable from *o* w.r.t. ε, *MinPts*.
  (density-connected is symmetric)



Density-Based Clustering: Basic Definitions

- ***Density-Based Cluster:*** non-empty subset S of database D satisfying:
  1) *Maximality*: if p is in S and q is density-reachable from p then q is in S
  2) *Connectivity*: each object in S is density-connected to all other objects in S

- ***Density-Based Clustering*** of a database D : $\{S_1, ..., S_n; N\}$ where
  - $S_1, ..., S_n$: all density-based clusters in the database D
  - $N = D \setminus \{S_1 \cup ... \cup S_n\}$ is called the ***noise*** (objects not in any cluster)



$\varepsilon = 1.0$
$MinPts = 5$

Density-Based Clustering: DBSCAN Algorithm

- Density Based Spatial Clustering of Applications with Noise
- Basic Theorem:
  - Each object in a density-based cluster C is density-reachable from any of its core-objects
  - Nothing else is density-reachable from core objects.

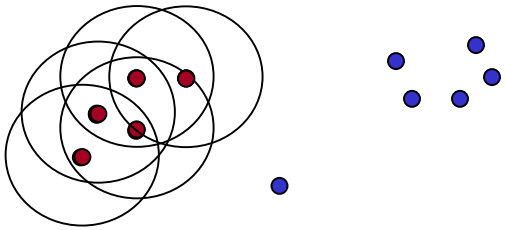
```
for each o ∈ D do
    if o is not yet classified then
        if o is a core-object then
            collect all objects density-reachable from o
            and assign them to a new cluster.
        else
            assign o to NOISE
```

  - density-reachable objects are collected by performing successive ε-neighborhood queries.

Ester M., Kriegel H.-P., Sander J., Xu X.: „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise“, *In KDD 1996*, pp. 226—231.

DBSCAN Algorithm: Example

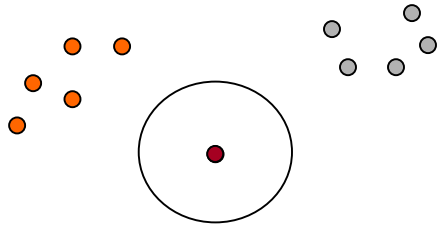
- Parameter
  - $\varepsilon = 2.0$
  - *MinPts* = 3



for each $o \in D$ **do**
  if o is not yet classified **then**
    if o is a core-object **then**
      collect all objects density-reachable from o and assign them to a new cluster.
    else
      assign o to NOISE

DBSCAN Algorithm: Example

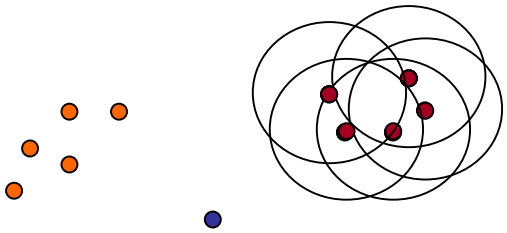
- Parameter
  - $\varepsilon = 2.0$
  - $MinPts = 3$



for each $o \in D$ **do**
  if o is not yet classified **then**
    if o is a core-object **then**
      collect all objects density-reachable from o
      and assign them to a new cluster.
    else
      assign o to NOISE

DBSCAN Algorithm: Example

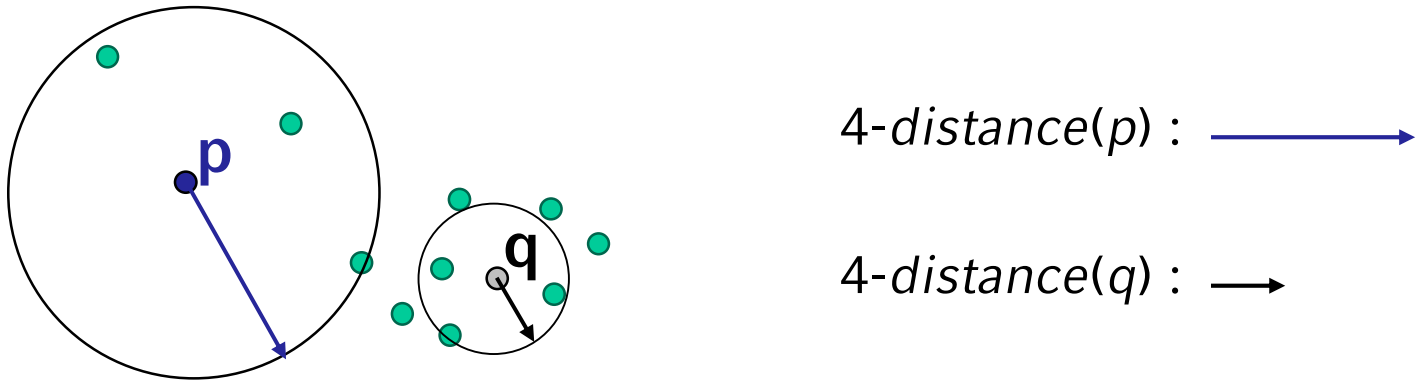
- Parameter
  - $\varepsilon = 2.0$
  - *MinPts* = 3



for each $o \in D$ **do**
  if o is not yet classified **then**
    if o is a core-object **then**
      collect all objects density-reachable from o and assign them to a new cluster.
    else
      assign o to NOISE

Determining the Parameters ε and *MinPts*

- Cluster: Point density higher than specified by ε and *MinPts*
- Idea: use the point density of the least dense cluster in the data set as parameters – but how to determine this?
- Heuristic: look at the distances to the *k*-nearest neighbors (typically computed for a sample of the objects)

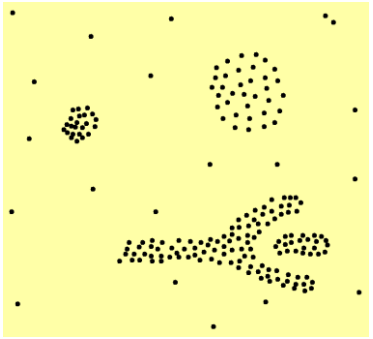


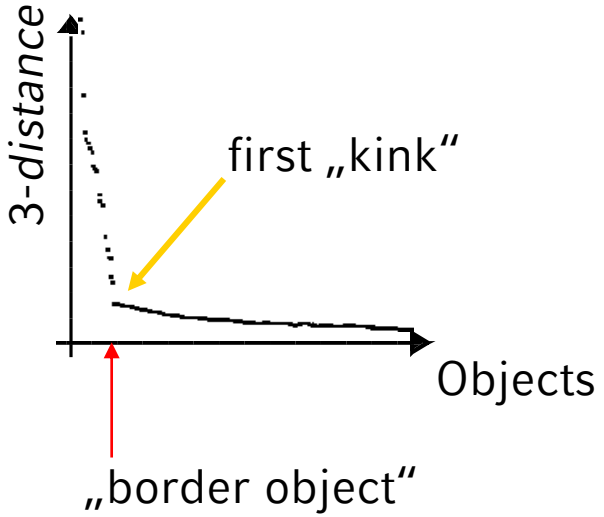
- Function *k-distance*(*p*): distance from *p* to the its *k*-nearest neighbor
- *k-distance plot*: *k*-distances of all objects, sorted in decreasing order

Determining the Parameters ε and *MinPts*

- Heuristic method:
  - Fix a value for *MinPts*
  - (default: $2 \times d - 1$, d = dimension of data space)
  - User selects "border object" *o* from the *MinPts-distance* plot; ε is set to *MinPts-distance*(o)

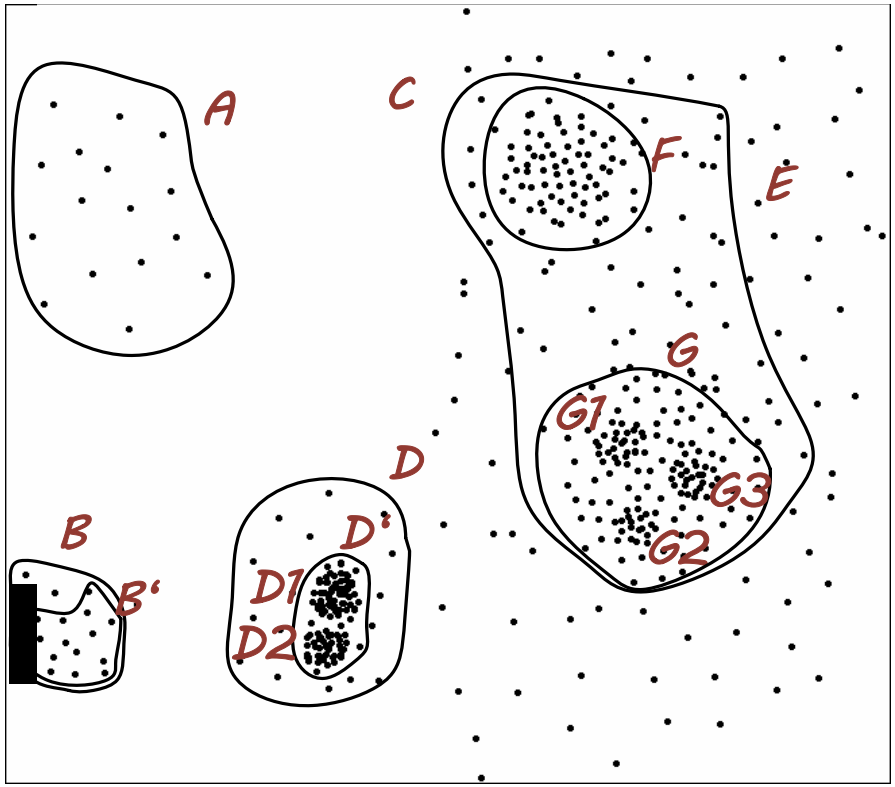
- Example *k*-distance plot
  1. dim = 2 → MinPts = 3
  2. Identify border object (kink)
  3. Set ε

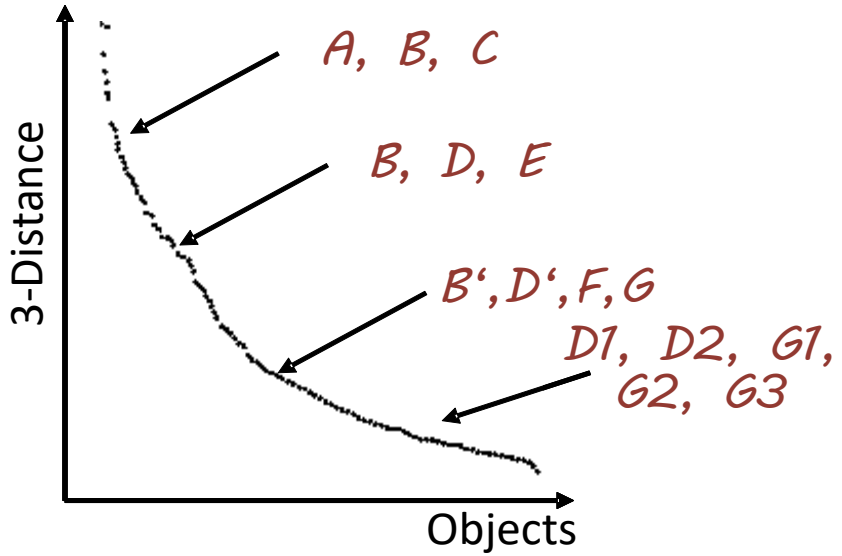




Determining the Parameters ε and *MinPts*

Problematic example





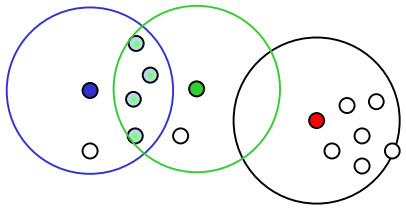
Density-Based Clustering: Discussion

- Advantages
  - Clusters can have arbitrary shape and size, i.e. clusters are not restricted to have convex shapes
  - Number of clusters is determined automatically
  - Can separate clusters from surrounding noise
  - Can be supported by spatial index structures
  - Complexity: N_ε-query: O(n) DBSCAN: O(n^2) (without index)
- Disadvantages
  - Input parameters may be difficult to determine
  - In some situations very sensitive to input parameter setting

Shared Nearest Neighbor (SNN) Clustering

- DBSCAN
  - Erkennt Cluster unterschiedlicher Form und Größe
  - Hat Probleme bei Clustern mit unterschiedlicher Dichte
- Verbesserung: anderer Ähnlichkeitsbegriff
  - Ähnlichkeit zwischen zwei Objekten, wenn sie beide sehr nahe zu einer Referenzmenge R sind
  - Ähnlichkeit wird durch die Referenzmenge R "bestätigt"
  - Ähnlichkeit z.B. durch die Anzahl gemeinsamer nächster Nachbarn definieren (d.h. R ist die Menge der nächsten Nachbarn)
  - Shared Nearest Neighbor (SNN) Ähnlichkeit:
    - $SNN_k\text{-similarity}(p,q) = |NN(p, k) \cap NN(q, k)|$
    - $NN(o, k)$ = Menge der k-nächsten Nachbarn von o

$k = 6$



$SNN_6\text{-similarity}(\bullet, \bullet) = 4$

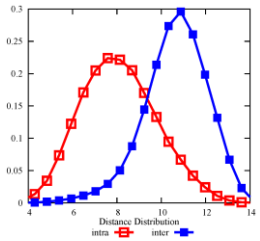
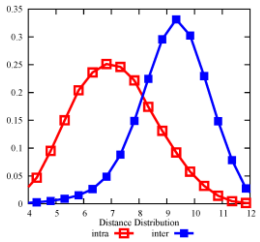
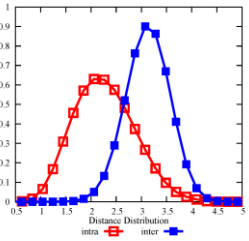
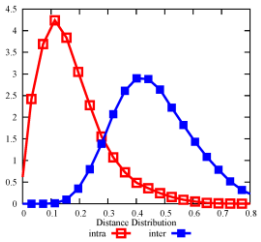
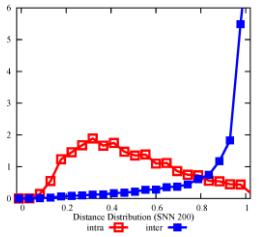
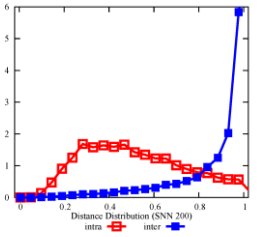
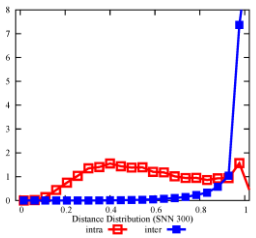
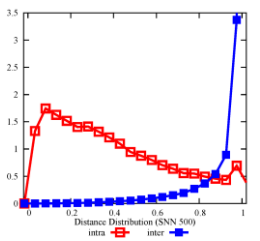
$SNN_6\text{-similarity}(\bullet, \bullet) = 0$

Dichtebasiertes Clustering

Studie: Stabilität von SNN [Houle, Kriegel, Kröger, Schubert, Zimek 2010]

- Verteilungen Euklidischer Distanz vs. SNN-Distanz: intra-cluster (rot) vs. inter-cluster (blau) Distanzen
- Vier reale Datensätze (unterschiedliche Multimedia-Features aus Bildern) mit hoher Dimensionalität (zwischen 5000 und 7000)

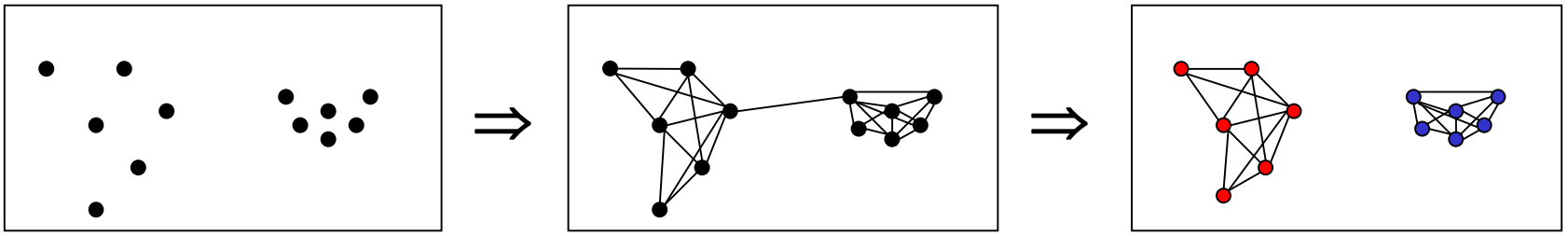
  (Basis-Datensatz: ALOI = Amsterdam Library of Images)

	Datensatz 1	Datensatz 2	Datensatz 3	Datensatz 4
Euklidische Distanz				
SNN Distanz				

Dichtebasiertes Clustering

Einfaches SNN-Clustering [Jarvis, Patrick 73]:

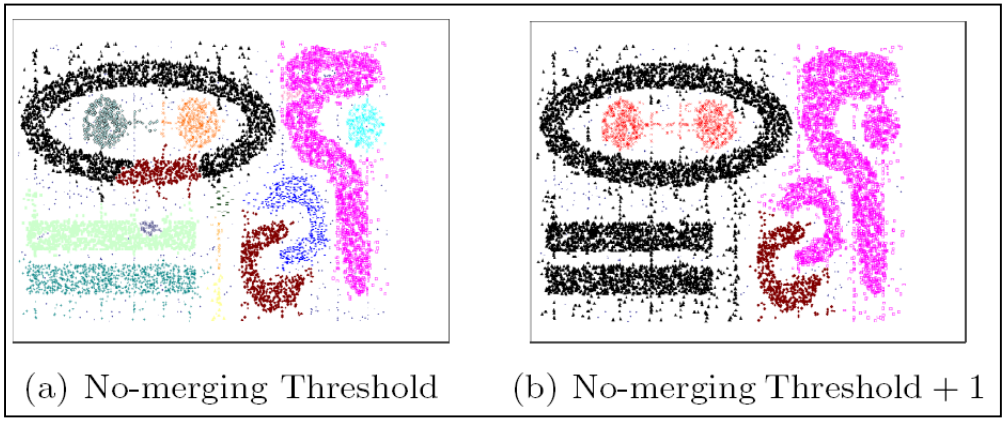
1. Berechnung der Ähnlichkeitsmatrix und des Ähnlichkeitsgraphen
   - für alle Objekt-Paare $p,q \in DB$: berechne SNN_k-similarity(p,q)
   - SNN_k-Ähnlichkeitsgraph:
     - Knoten = Objekte
     - Kante zwischen jedem Objektpaar p,q mit Gewicht SNN_k-similarity(p,q)
   - Keine Kanten mit Gewicht 0
2. Generiere Cluster
   - Lösche alle Kanten, deren Gewicht unterhalb eines Grenzwerts τ liegen
   - Cluster = verbundene Komponenten im resultierenden Graphen



Dichtebasiertes Clustering

Problem:

- Threshold τ schwer zu bestimmen
- kleine Variationen führen zu stark unterschiedlichen Ergebnissen



(a) No-merging Threshold (b) No-merging Threshold + 1

Bild aus:

[Ertöz, Steinbach, Kumar 03]

Lösung [Ertöz, Steinbach, Kumar 03]

- Kombiniere SNN-Ähnlichkeit mit dichtebasierten Konzepten
- SNN-Dichte:

  Anzahl der Punkte innerhalb eines spezifizierten Radius ε bzgl. SNN-Ähnlichkeit

  $SNN_k\text{-density}(p,\varepsilon) = |\{q \mid SNN_k\text{-similarity}(p,q) \geq \varepsilon\}|$

Dichtebasiertes Clustering

SNN-Dichte

- Beispiel:
  - 10 000 Daten (Bild (a))
    $k = 50$, $\varepsilon = 20$
  - Bild (b): "Kernpunkte"
    alle Punkte mit SNN-Dichte ≥ 34
  - Bild (c): "Randpunkte"
    alle Punkte mit SNN-Dichte ≥ 17
  - Bild (d): "Rauschen"
    alle Punkte mit SNN-Dichte < 17

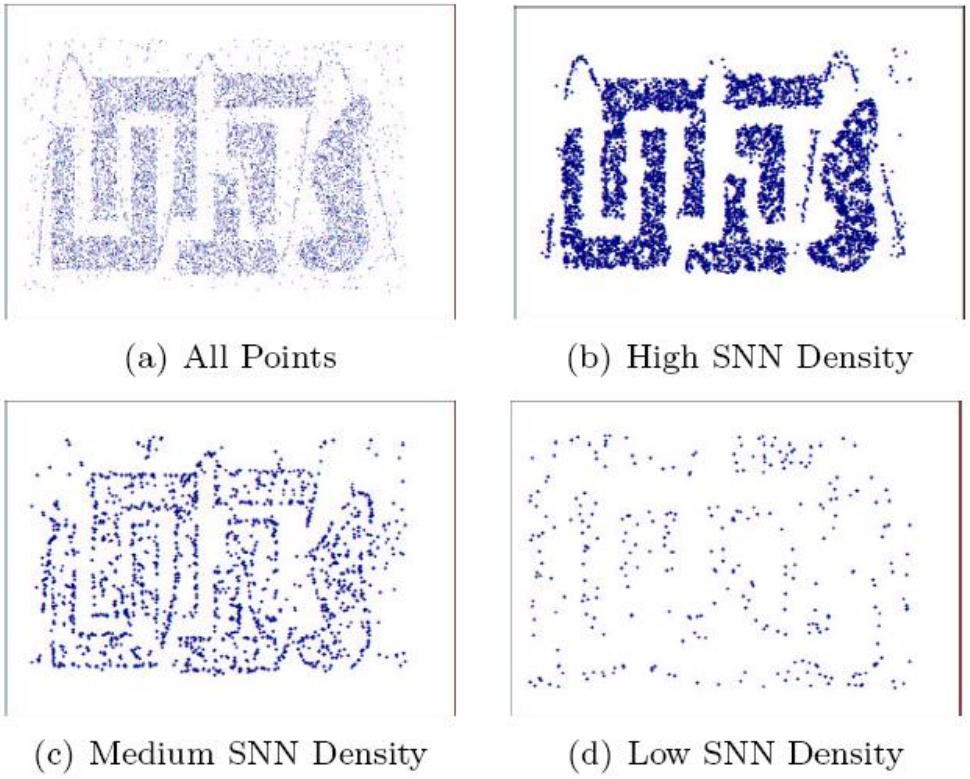


Bild aus: [Ertöz, Steinbach, Kumar 03]

- Analogie zu DBSCAN: $\varepsilon = 20$, *minPts* $= 34$
- Kernpunkt *p*: mehr als *minPts* Punkte haben 20 oder mehr der 50 nächsten Nachbarn mit *p* gemeinsam

SNN-Clustering Algorithmus [Ertöz, Steinbach, Kumar 03]

Eingabe: k, ε, *minPts*

1. Berechne Ähnlichkeitsmatrix und –graph
   (siehe einfaches SNN-Clustering)
2. Berechne die SNN_k-Dichte für jeden Punkt bzgl. ε
3. Bestimme Kernpunkte bzgl. *minPts*
   (alle Punkte mit einer SNN-Dichte $\geq$ *minPts*)
4. Vereinige Kernpunkte p,q, wenn SNN_k-similarity(p,q) $\geq \varepsilon$
5. Ordne Nicht-Kernpunkt p einem Cluster zu, wenn es einen Kernpunkt q gibt, mit SNN_k-similarity(q,p) $\geq \varepsilon$
6. Alle anderen Nicht-Kernpunkte sind Rauschen

DBSCAN mit SNN-Ähnlichkeit

Diskussion

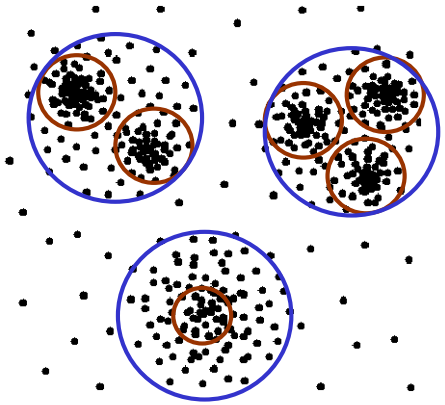
- Unterschied zu DBSCAN
  - DBSCAN mit Euklidischer Distanz: nur Cluster, die dichter sind als der Grenzwert (spezifiziert durch *minPts* und ε)
  - SNN-Dichte eines Punktes *p*: Anzahl der Punkte, die mind. ε nächste Nachbarn mit *p* gemeinsam haben

    $\Rightarrow$ unabhängig von der eigentlichen Dichte, daher adaptiver
- Parametrisierung
  - Wahl von *k* ist kritisch:
    - Zu klein: auch relativ gleichverteilte Cluster werden wegen lokalen Variationen gesplittet $\Rightarrow$ viele kleine Cluster
    - Zu groß: wenige große, gut-separierte Cluster
  - *minPts*, $\varepsilon < k$

Contents

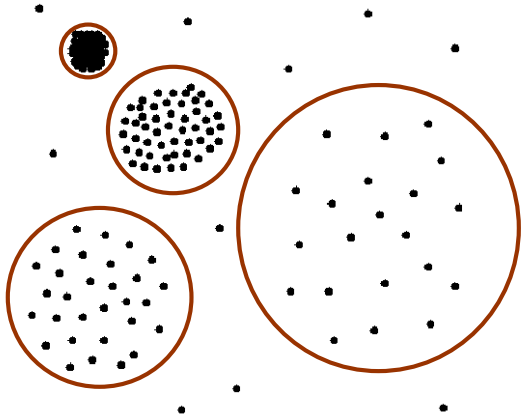
From Partitioning to Hierarchical Clustering

- Global parameters to separate all clusters with a partitioning clustering method may not exist



hierarchical cluster structure

and/or

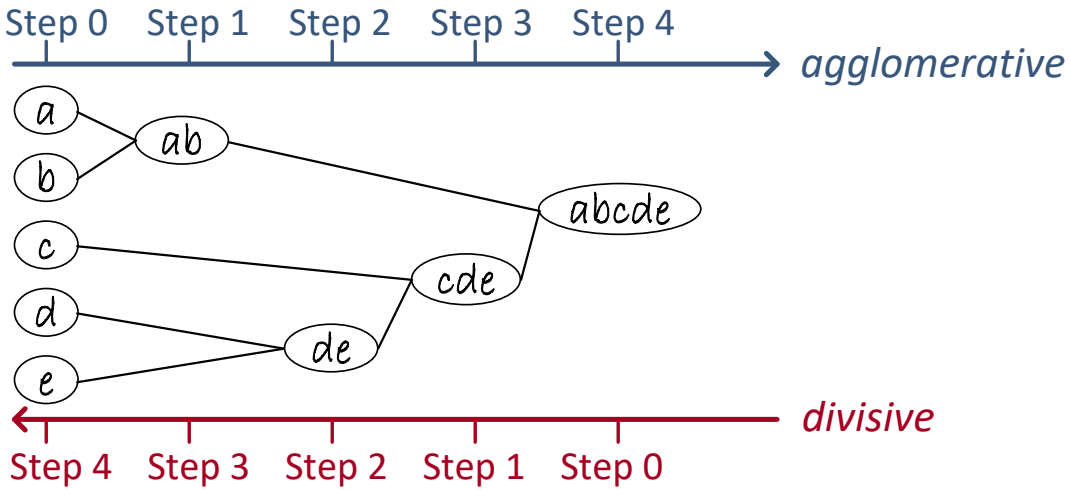


largely differing densities and sizes

- Need a hierarchical clustering algorithm in these situations

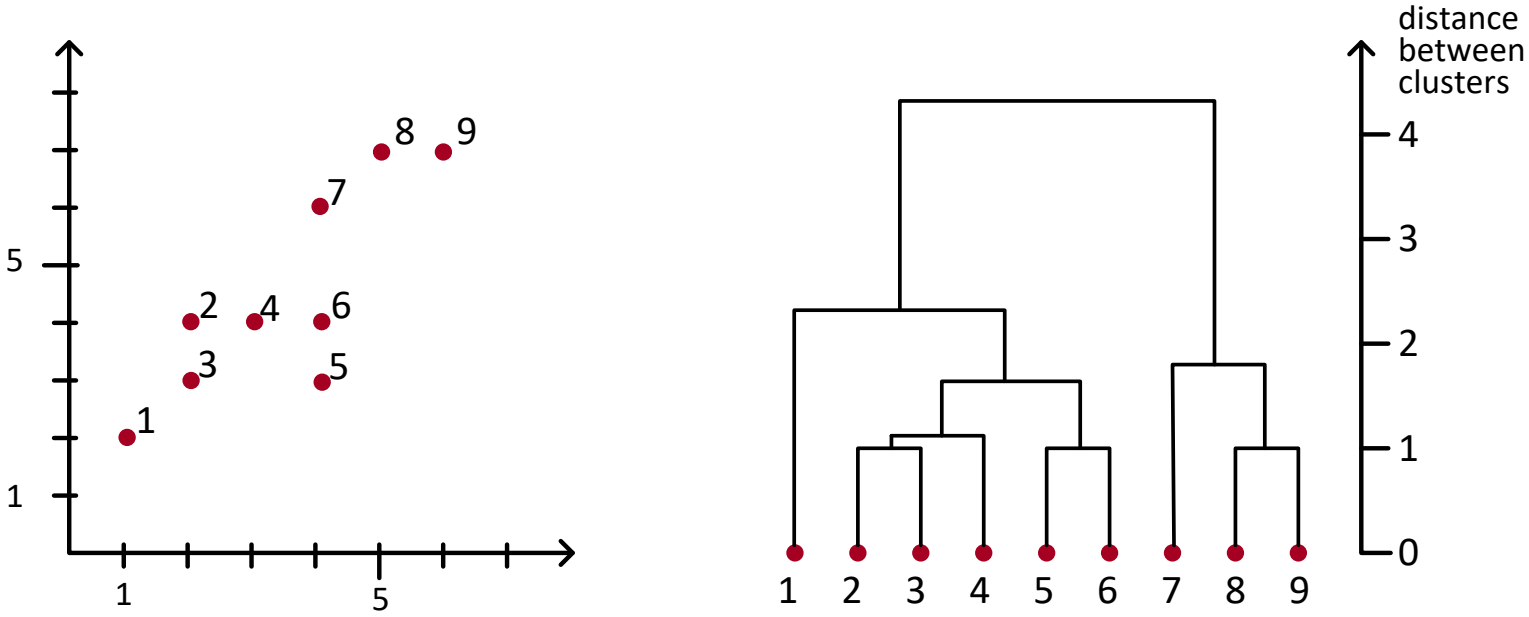
Hierarchical Clustering: Basic Notions

- Hierarchical decomposition of the data set (with respect to a given similarity measure) into a set of nested clusters
- Result represented by a so called *dendrogram* (greek δενδρο = tree)
  - Nodes in the dendrogram represent possible clusters
  - can be constructed bottom-up (agglomerative approach) or top down (divisive approach)



Hierarchical Clustering: Example

- Interpretation of the dendrogram
  - The root represents the whole data set
  - A leaf represents a single object in the data set
  - An internal node represents the union of all objects in its sub-tree
  - The height of an internal node represents the distance between its two child nodes



Contents

Agglomerative Hierarchical Clustering

1. Initially, each object forms its own cluster
2. Consider all pairwise distances between the initial clusters (objects)
3. Merge the closest pair (A, B) in the set of the current clusters into a new cluster $C = A \cup B$
4. Remove A and B from the set of current clusters; insert C into the set of current clusters
5. If the set of current clusters contains only C (i.e., if C represents all objects from the database): STOP
6. Else: determine the distance between the new cluster C and all other clusters in the set of current clusters; go to step 3.

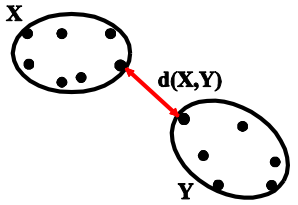
=> Requires a ***distance function for clusters*** (sets of objects)

Single Link Method and Variants

Distance Functions for clusters:

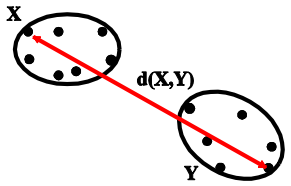
- Given: a distance function *dist*(*p*, *q*) for database objects
- The following distance functions for clusters (i.e., sets of objects) *X* and *Y* are commonly used for hierarchical clustering:

Single-Link: $dist\_sl(X,Y) = \min_{x \in X, y \in Y} dist(x,y)$



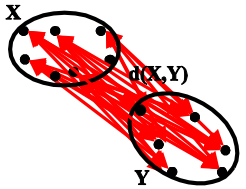
Single-Link

Complete-Link: $dist\_cl(X,Y) = \max_{x \in X, y \in Y} dist(x,y)$



Complete-Link

Average-Link: $dist\_al(X,Y) = \frac{1}{|X| \cdot |Y|} \cdot \sum_{x \in X, y \in Y} dist(x,y)$



Average-Link

Divisive Hierarchical Clustering

General approach: Top Down

- Initially, all objects form one cluster
- Repeat until all objects are singletons
  - Choose a cluster to split ⇐ **how ?**
  - Replace the chosen cluster with the sub-clusters ⇐ **how to split ?** e.g., 'reversing' agglomerative approach and split into two

Example solution: DIANA

- Select the cluster C with largest diameter for splitting
- Search the most disparate observation o in C (highest average dissimilarity)
  - $SplinterGroup := \{o\}$
  - Iteratively assign the $o' \in C \backslash SplinterGroup$ with the highest $D(o') > 0$ to the splinter group until for all $o' \in C \backslash SplinterGroup$: $D(o') \leq 0$

$$D(o') = \sum_{o_j \in C \backslash SplinterGroup} \frac{d(o', o_j)}{|C \backslash SplinterGroup|} - \sum_{o_i \in SplinterGroup} \frac{d(o', o_i)}{|SplinterGroup|}$$

Discussion Agglomerative vs. Divisive HC

Divisive HC and Agglomerative HC need n-1 steps

- Agglomerative HC has to consider $\frac{n \cdot (n-1)}{2} = \binom{n}{2}$ combinations in the first step ($O(n^2)$ implementations exist for SL, CL, and AL)
- Divisive HC has $2^{n-1} - 1$ many possibilities to split the data in its first step
  →Not every possibility has to be considered (DIANA)

Divisive HC is conceptually more complex since it needs a second 'flat' clustering algorithm (splitting procedure)

Agglomerative HC decides based on local patterns

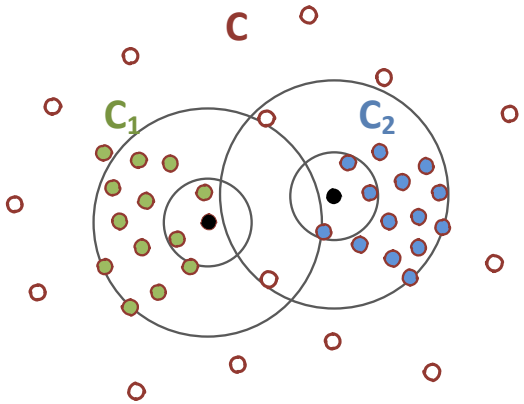
Divisive HC uses complete information about the global data distribution
→more accurate than Agglomerative HC?!

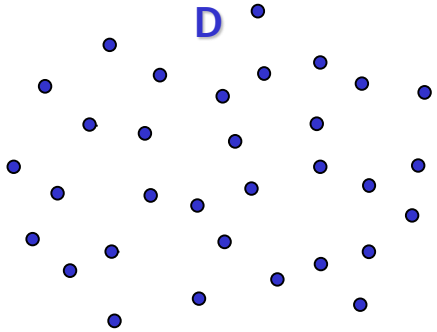
Contents

Density-Based Hierarchical Clustering

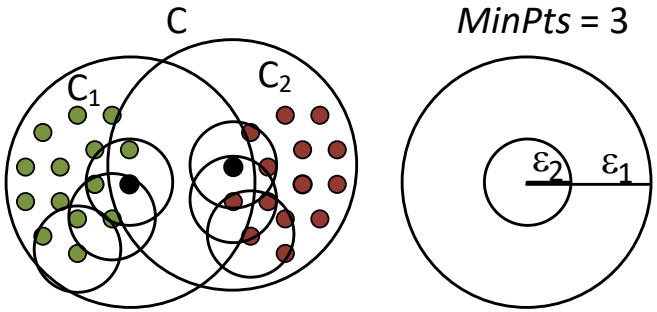
- *Observation*: Dense clusters are completely contained by less dense clusters

  => could we obtain clusters of ***different*** density threshold in one run?





- *Idea*: Process objects in the "right" order and keep track of point density in their neighborhood

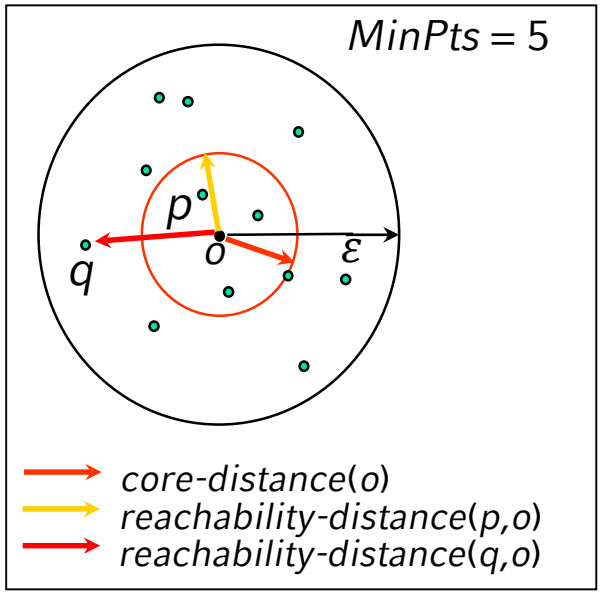


Core Distance and Reachability Distance

- Parameters: "maximal" ε, fixed value *MinPts*
- Goal: Find all clusters with threshold $\varepsilon' \leq \varepsilon$
- ***core-distance***$_{\varepsilon,MinPts}$***(o)***

  "smallest distance such that *o* is a core object"
  (if core-distance > ε: "?")
- ***reachability-distance***$_{\varepsilon,MinPts}$***(p, o)***

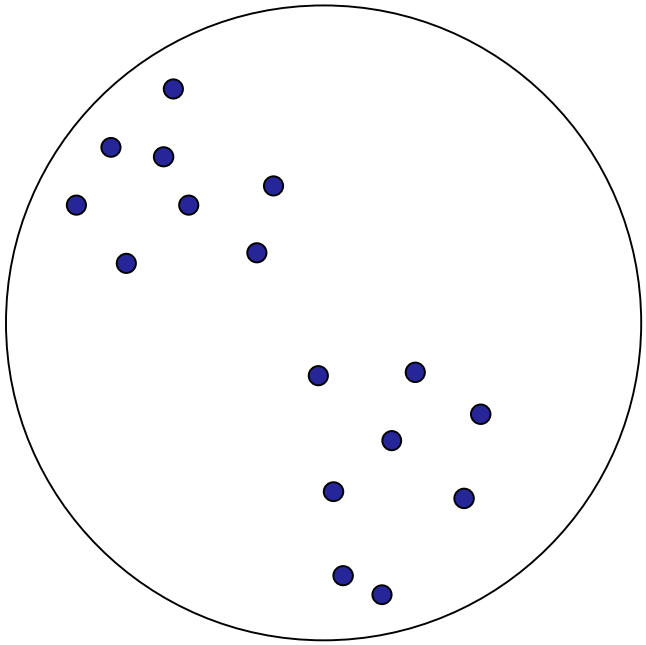
  "smallest distance such that *p* is *directly* density-reachable from *o*"
  (if reachability-distance > ε: ∞)



$$reach\_dist(p,o) = \begin{cases} dist(p,o) & ,dist(p,o) \geq core\_dist(o) \\ core\_dist(o) & ,dist(p,o) < core\_dist(o) \\ \infty & ,dist(p,o) > \epsilon \end{cases}$$

The Algorithm OPTICS

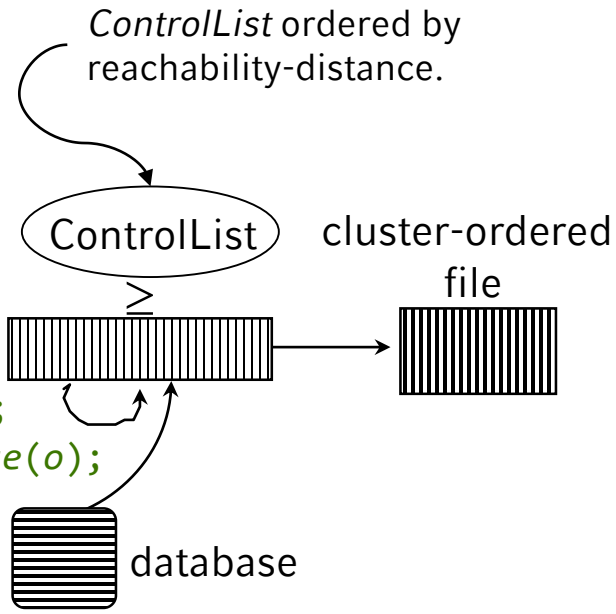
- OPTICS: **O**rdering **P**oints **T**o **I**dentify the **C**lustering **S**tructure
- Basic data structure: controlList
  - Memorize shortest reachability distances seen so far ("distance of a jump to that point")
- Visit each point
  - Make always a shortest jump
- Output:
  - order of points
  - core-distance of points
  - reachability-distance of points



Ankerst M., Breunig M., Kriegel H.-P., Sander J.: „OPTICS: Ordering Points To Identify the Clustering Structure", *In SIGMOD 1999,* pp. 49-60.

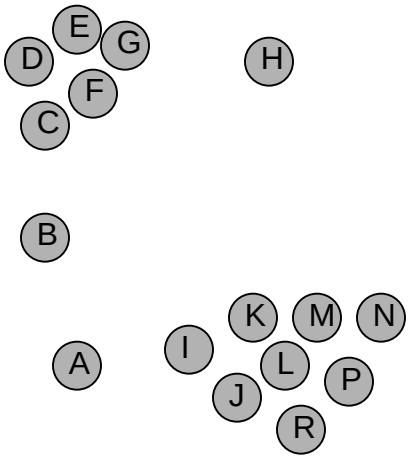
The Algorithm OPTICS

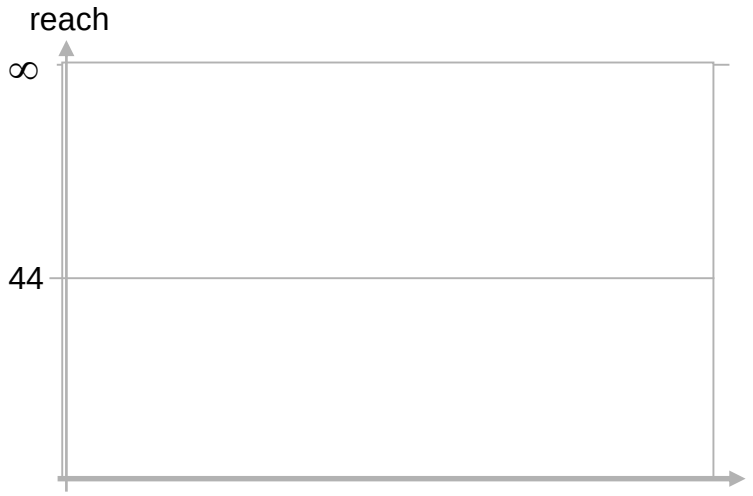

```
foreach o ∈ Database
  // initially, o.processed = false for all objects o

  if o.processed = false;
     insert (o, ∞) into ControlList;

  while ControlList contains objects not yet processed
        select first element (o, r-dist) from ControlList;
        retrieve N_e(o) and determine c_dist = core-distance(o);
        set o.processed = true;
        write (o, r_dist, c_dist) to file;
        if o is a core object at any distance ≤ ε
          foreach p ∈ N_e(o) not yet processed;
                determine r_dist_p =  reachability-distance(p, o);
                if (p, _) ∉ ControlList
                    insert (p, r_dist_p) in ControlList;
                else if (p, old_r_dist) ∈ ControlList and r_dist_p < old_r_dist
                    update (p, r_dist_p) in ControlList;
```



The Algorithm OPTICS - Example

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$



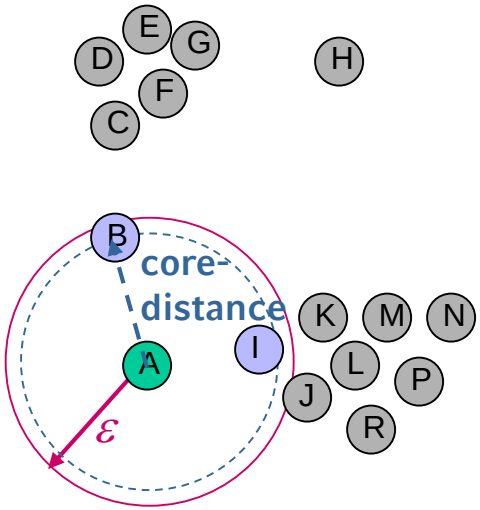


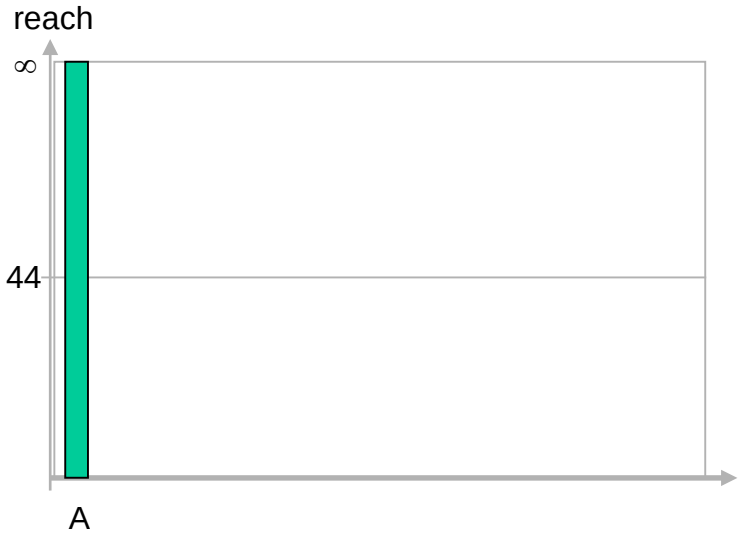
seed list:

(ControlList)

The Algorithm OPTICS – Example (2)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

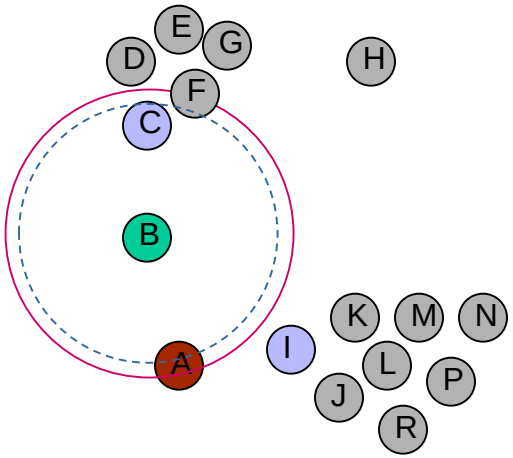


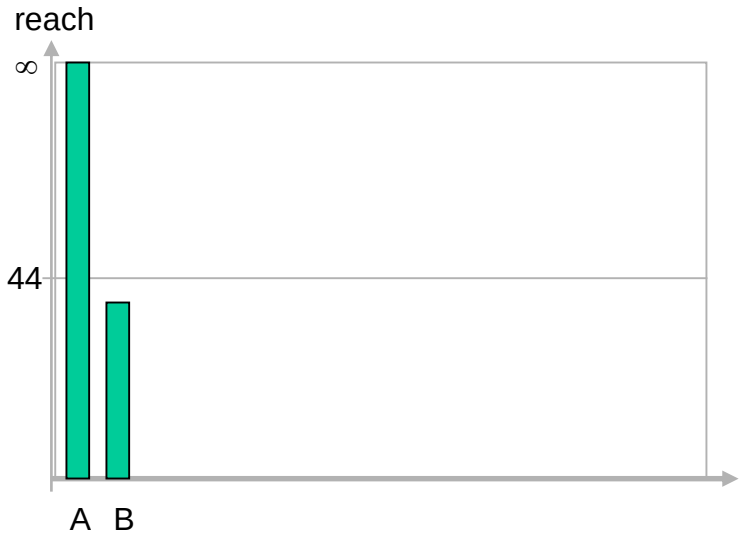


seed list: (B,40) (I, 40)

The Algorithm OPTICS – Example (3)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

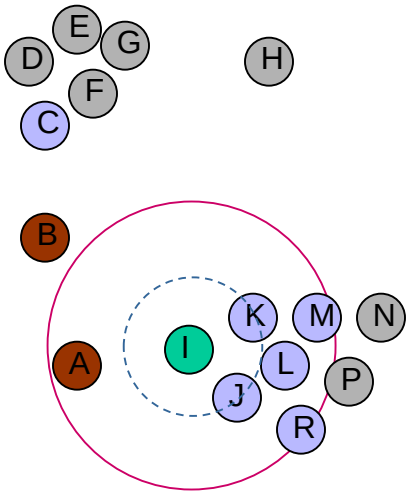




seed list: (I, 40) (C, 40)

The Algorithm OPTICS – Example (4)

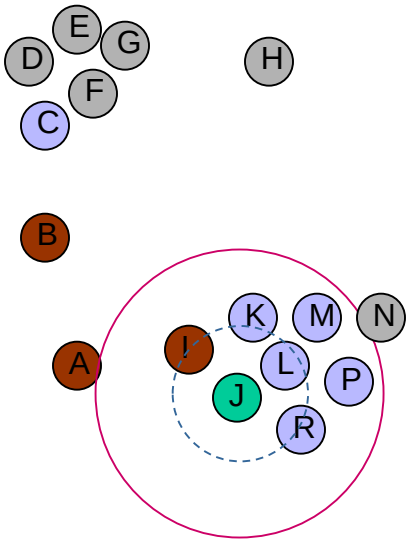
- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$



seed list: (J, 20) (K, 20) (L, 31) (C, 40) (M, 40) (R, 43)

The Algorithm OPTICS – Example (5)

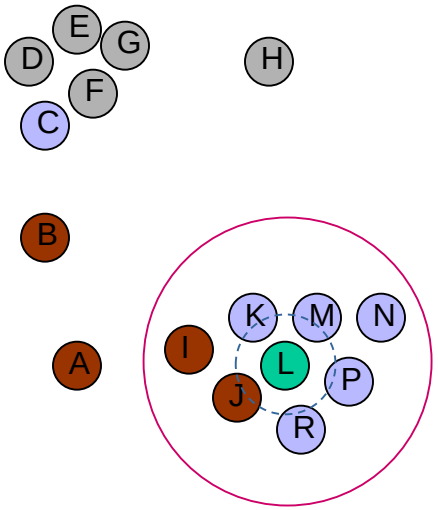
- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

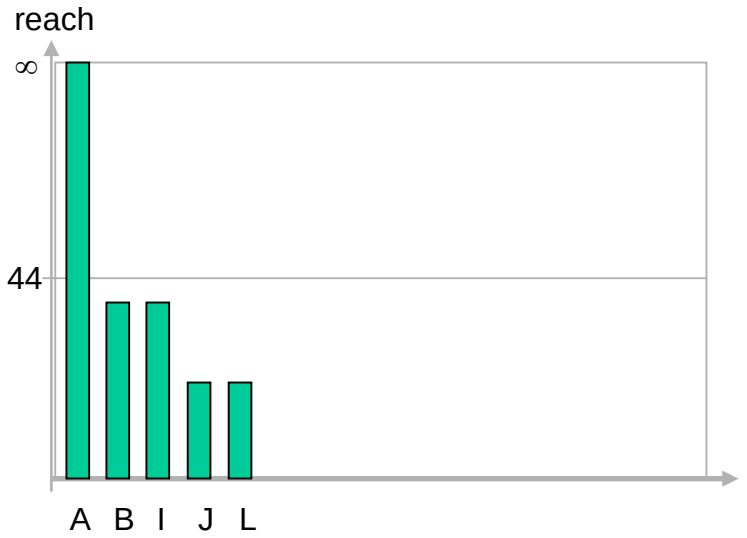


seed list: (L, 19) (K, 20) (R, 21) (M, 30) (P, 31) (C, 40)

The Algorithm OPTICS – Example (6)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

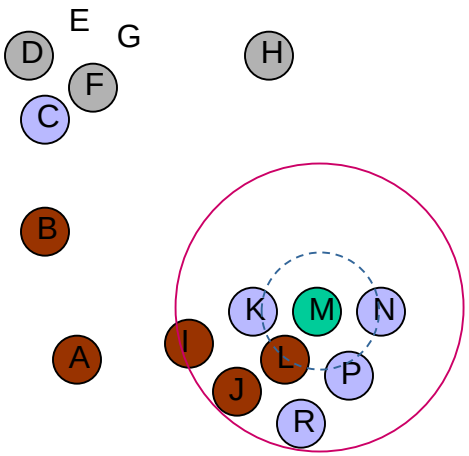


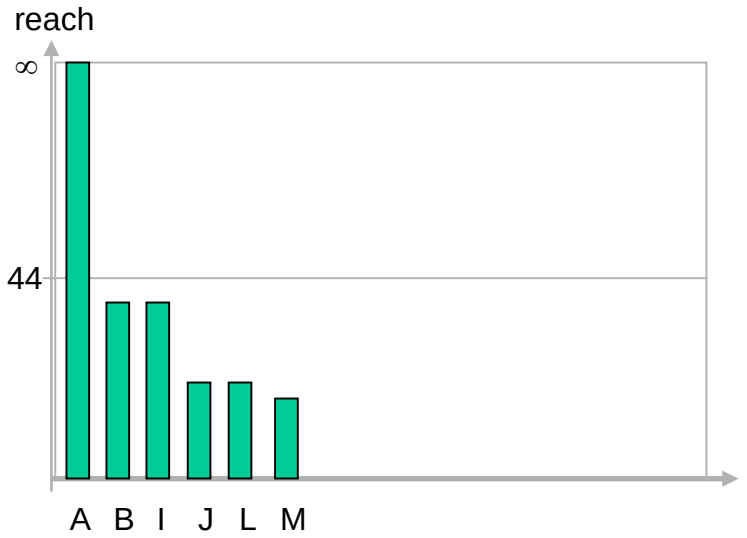


seed list: (M, 18) (K, 18) (R, 20) (P, 21) (N, 35) (C, 40)

The Algorithm OPTICS – Example (7)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

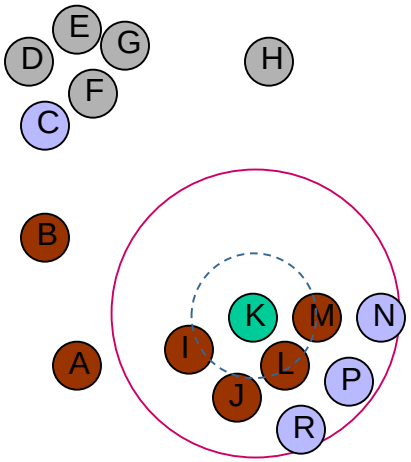


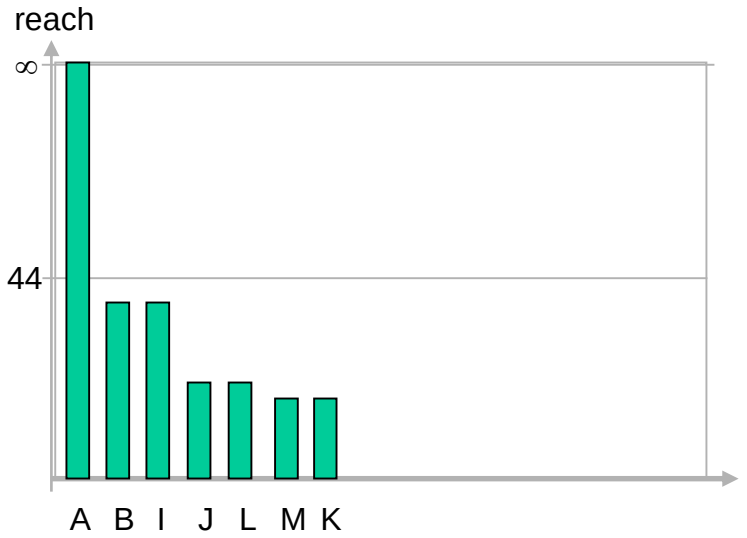


seed list: (K, 18) (N, 19) (R, 20) (P, 21) (C, 40)

The Algorithm OPTICS – Example (8)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

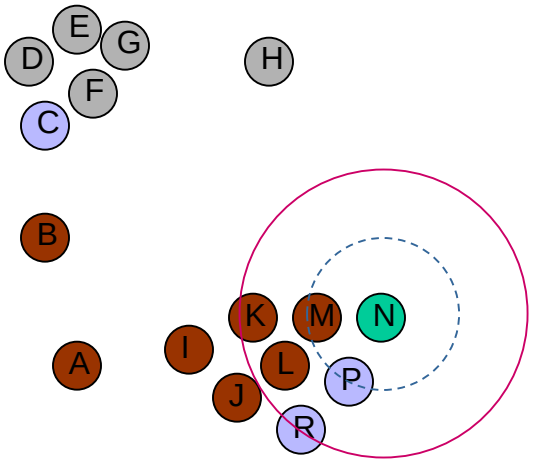


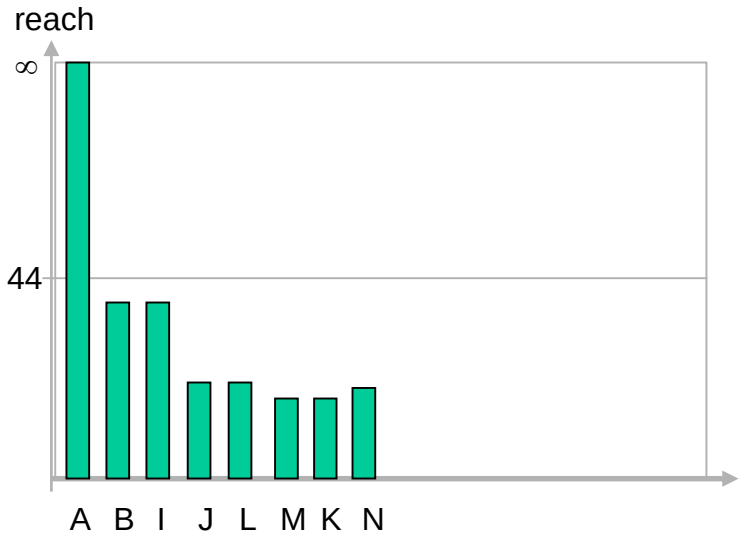


seed list: (N, 19) (R, 20) (P, 21) (C, 40)

The Algorithm OPTICS – Example (9)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

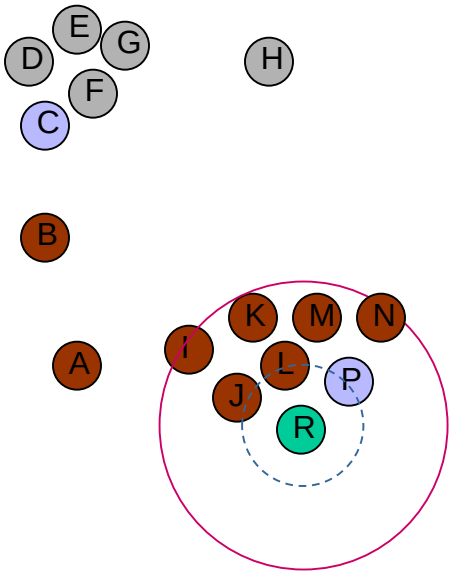


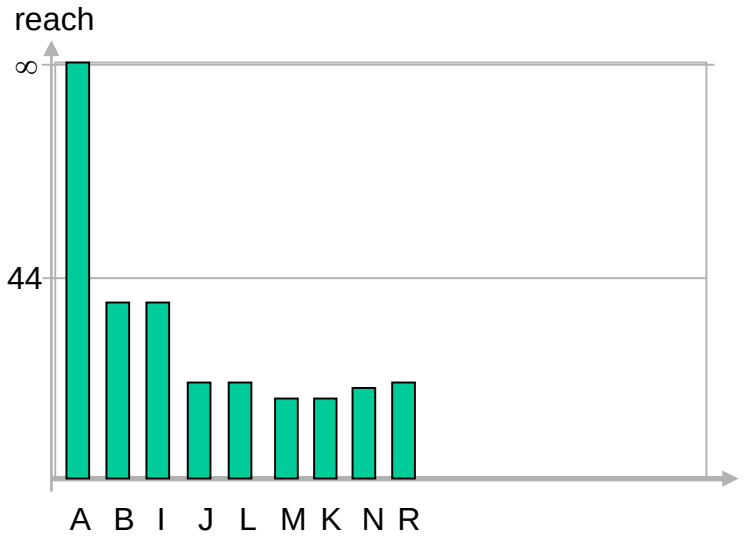


seed list: (R, 20) (P, 21) (C, 40)

The Algorithm OPTICS – Example (10)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

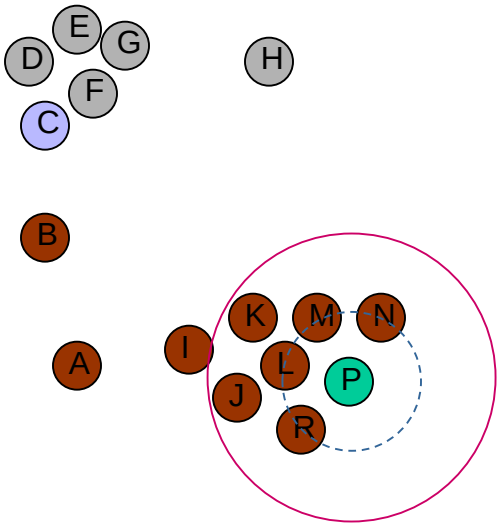




seed list: (P, 21) (C, 40)

The Algorithm OPTICS – Example (11)

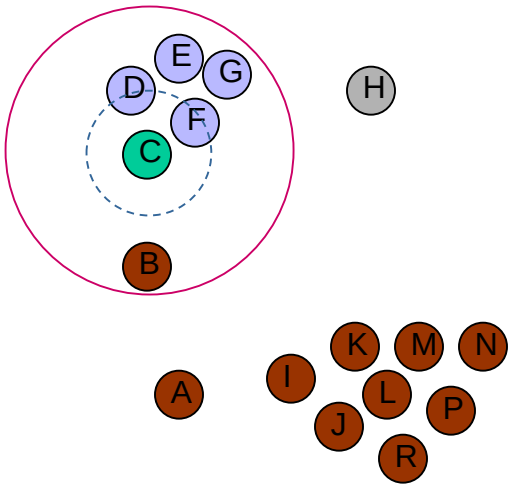
- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

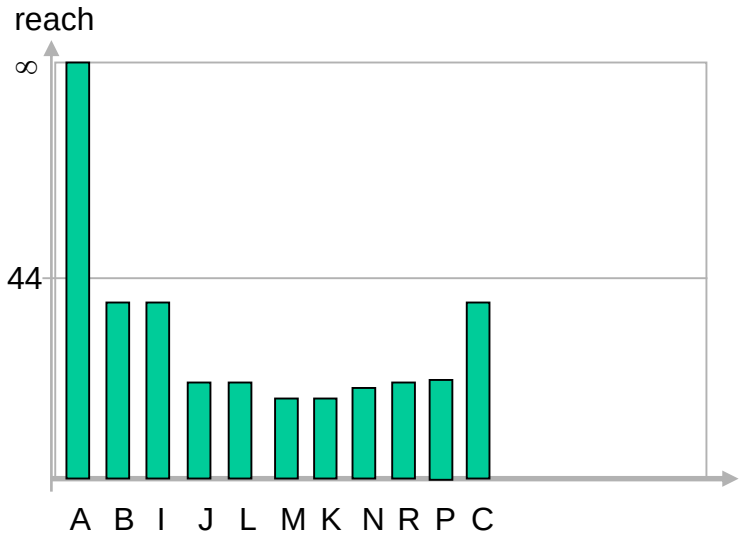


seed list: (C, 40)

The Algorithm OPTICS – Example (12)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

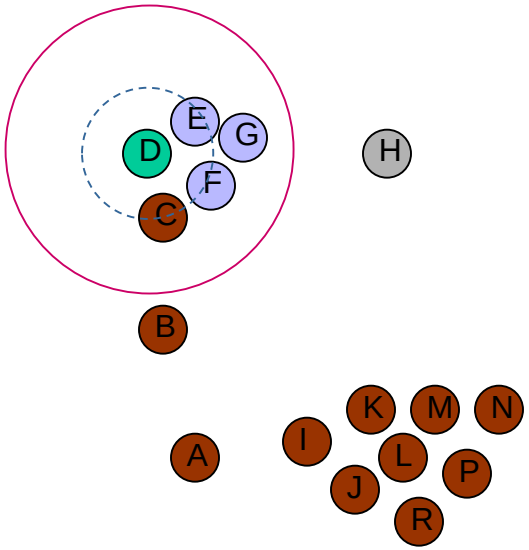


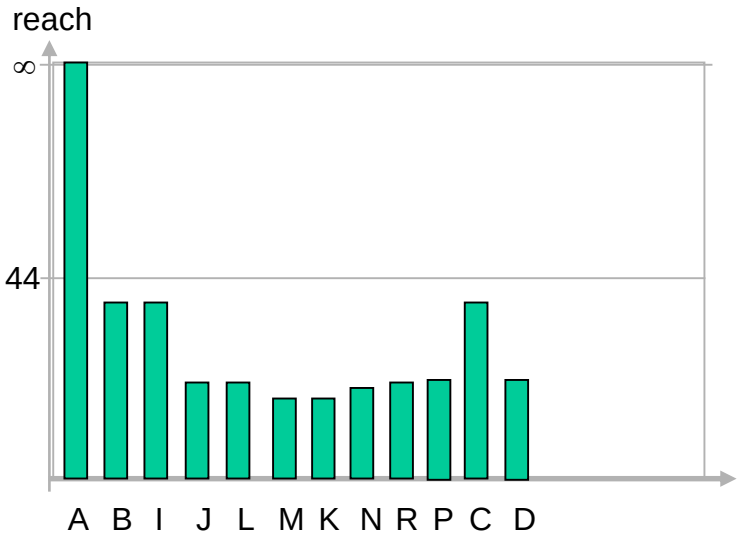


seed list: (D, 22) (F, 22) (E, 30) (G, 35)

The Algorithm OPTICS – Example (13)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

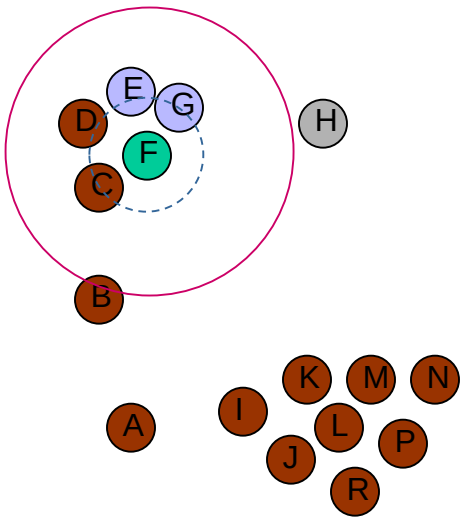


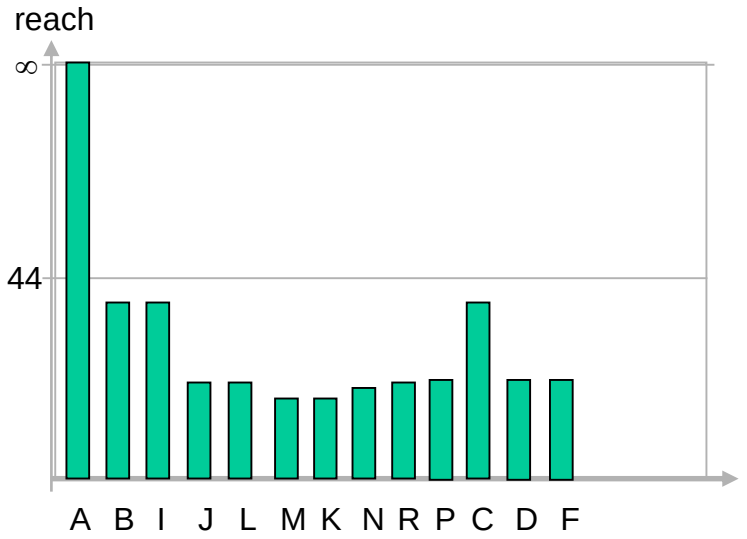


seed list: (F, 22) (E, 22) (G, 32)

The Algorithm OPTICS – Example (14)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

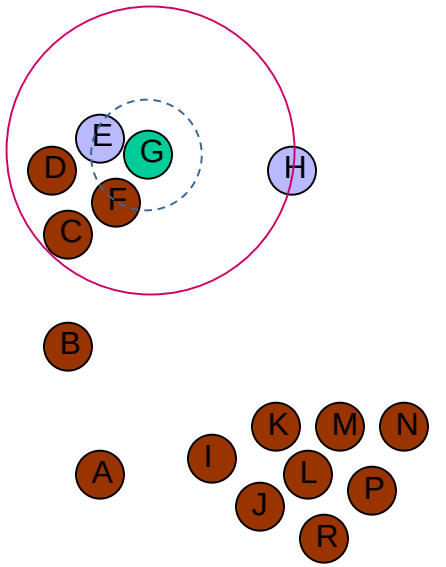


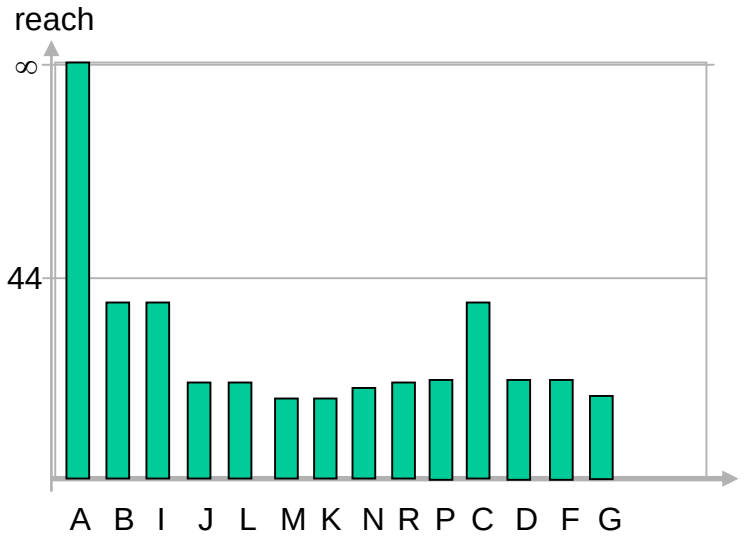


seed list: (G, 17) (E, 22)

The Algorithm OPTICS – Example (15)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

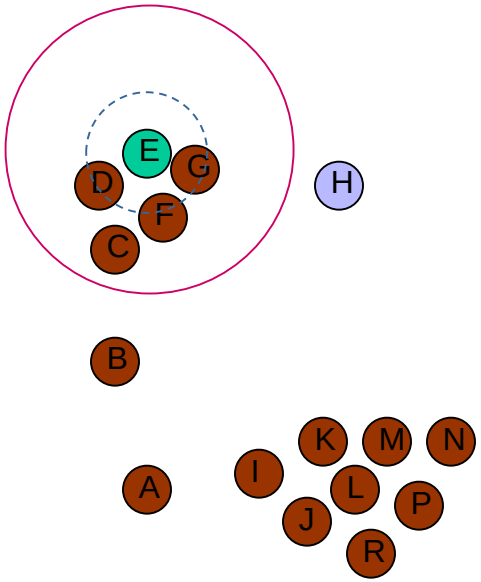


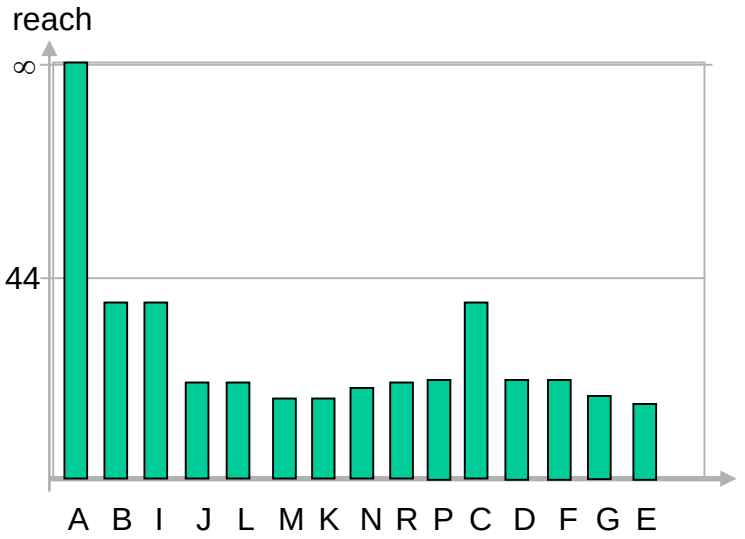


seed list: (E, 15) (H, 43)

The Algorithm OPTICS – Example (16)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

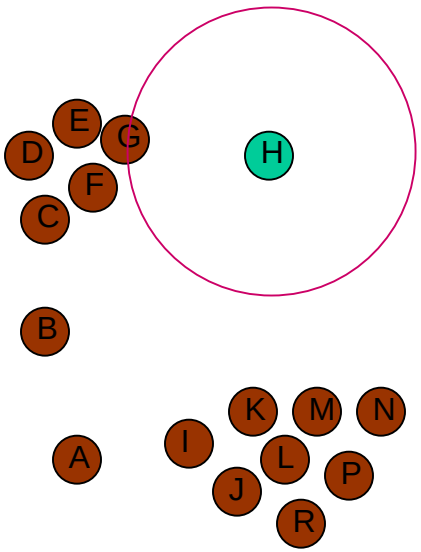


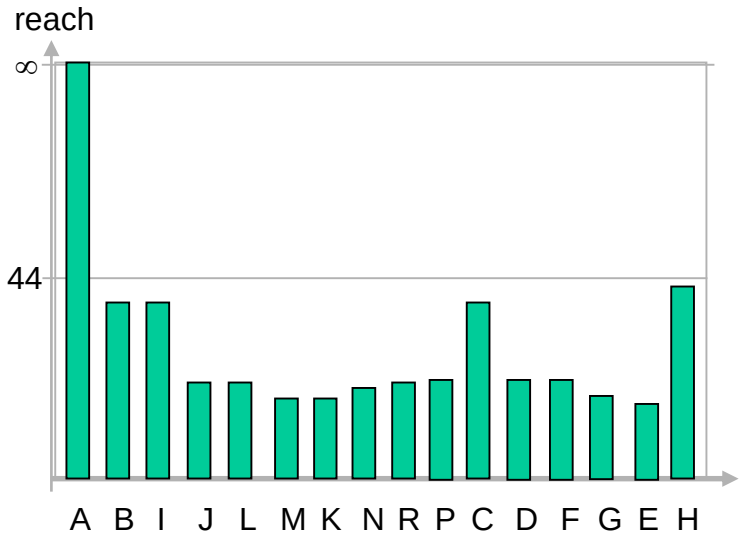


seed list: (H, 43)

The Algorithm OPTICS – Example (17)

- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$

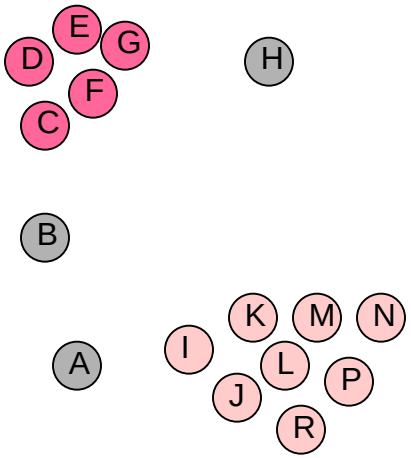


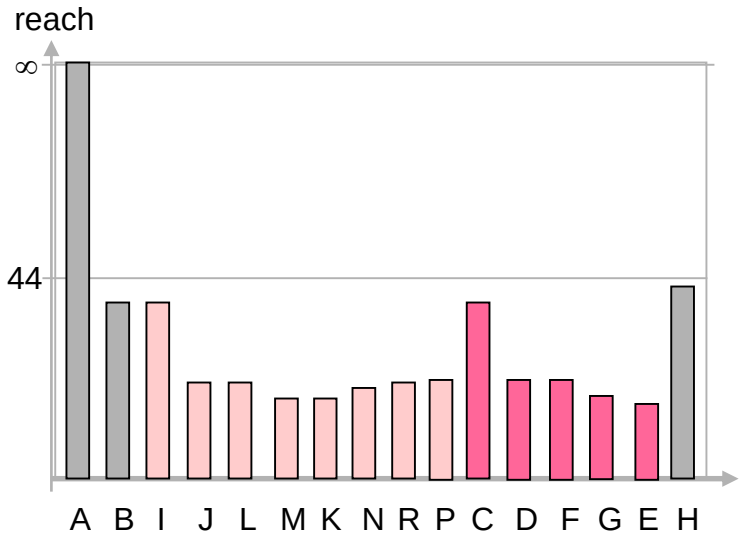


seed list: -

The Algorithm OPTICS – Example (18)

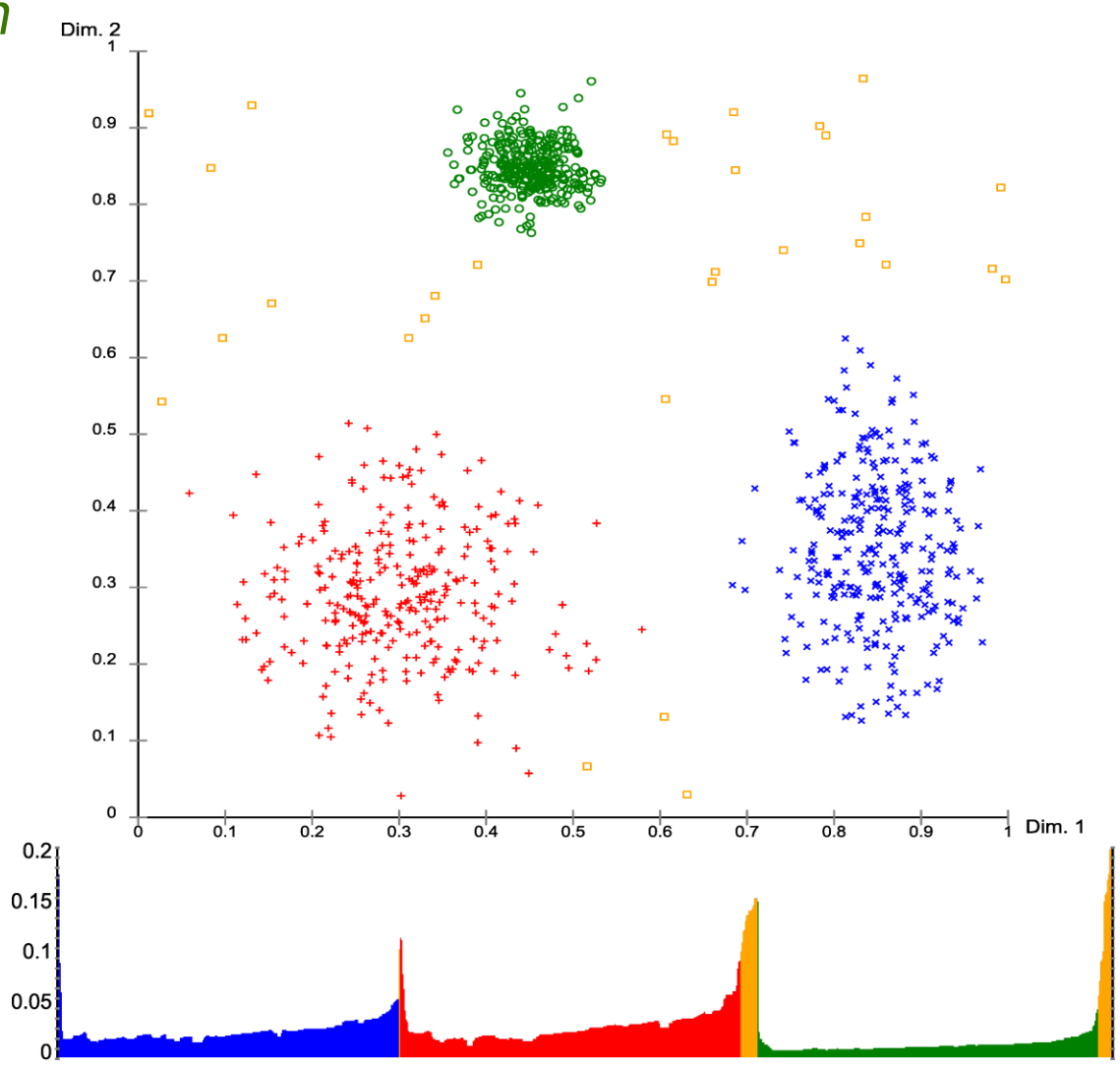
- Example Database (2-dimensional, 16 points)
- $\varepsilon = 44$, $MinPts = 3$





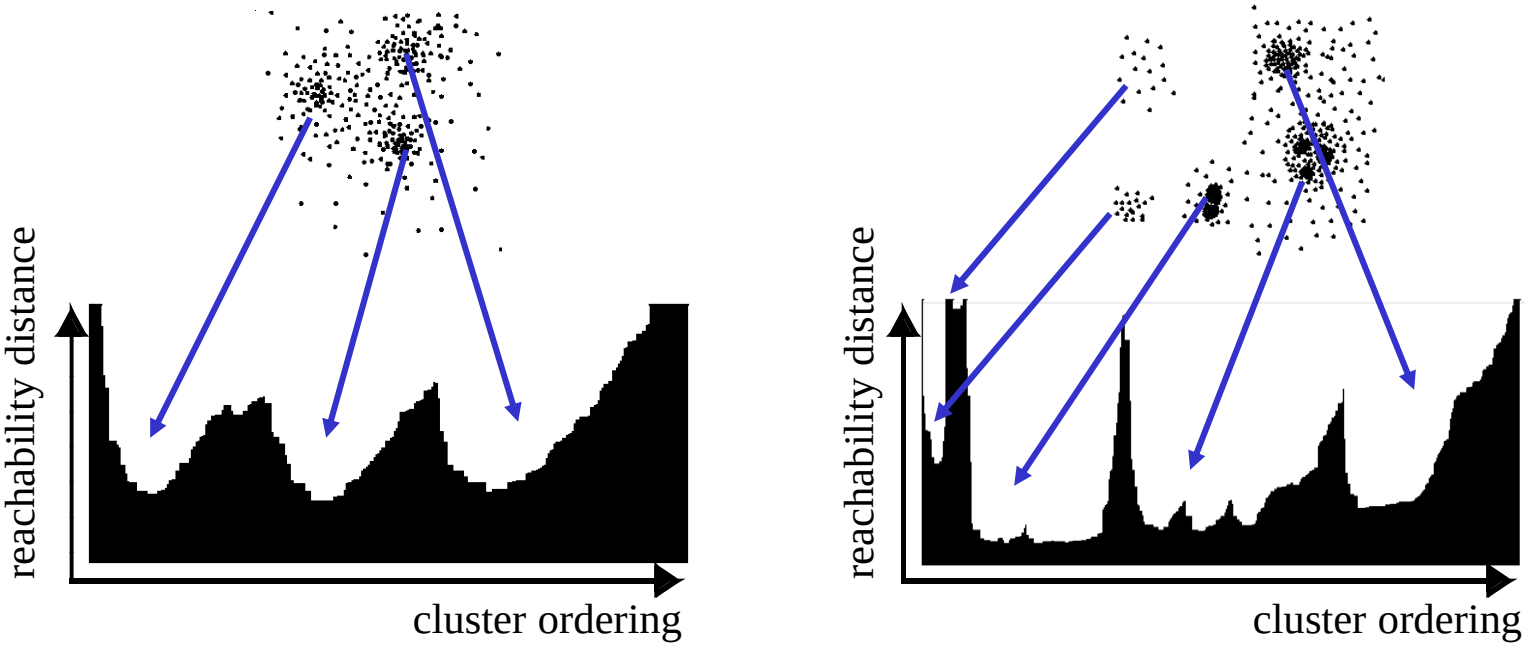
OPTICS: The Reachability Plot

Reachability diagram



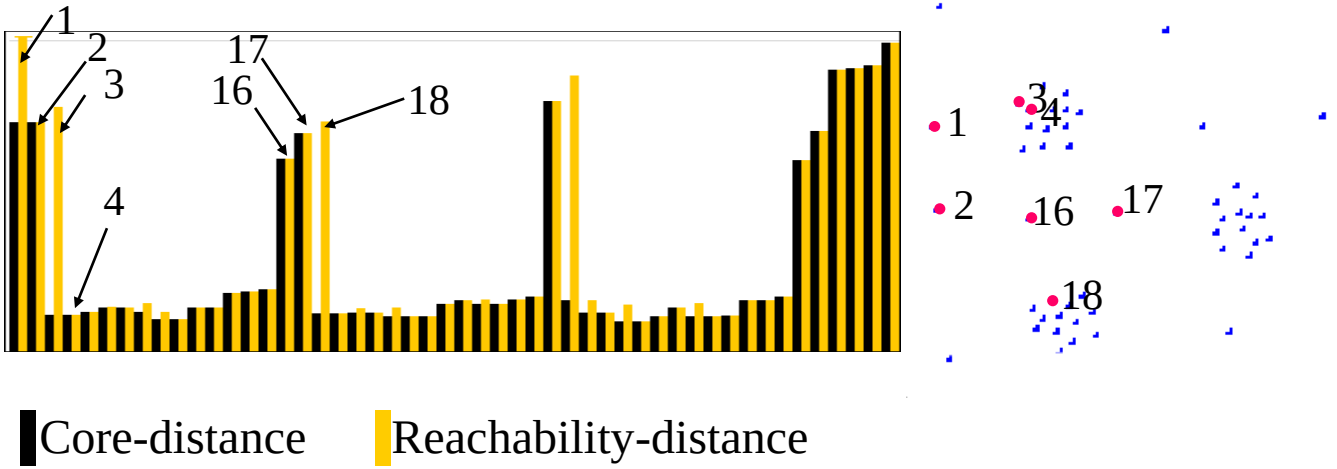
OPTICS: The Reachability Plot

- plot the points together with their reachability-distances. Use the order in which they where returned by the algorithm
  - represents the density-based clustering structure
  - easy to analyze
  - independent of the dimension of the data



OPTICS: Properties

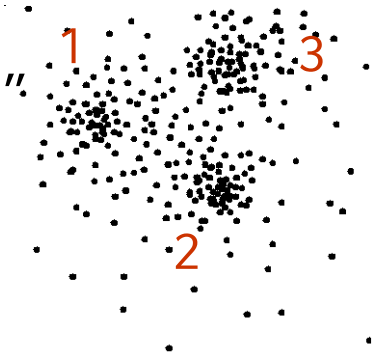
- "Flat" density-based clusters wrt. $\varepsilon^* \leq \varepsilon$ and *MinPts* afterwards:
  - Start with an object *o* where *c-dist*(*o*) $\leq \varepsilon^*$ and *r-dist*(*o*) $> \varepsilon^*$
  - Continue while *r-dist* $\leq \varepsilon^*$



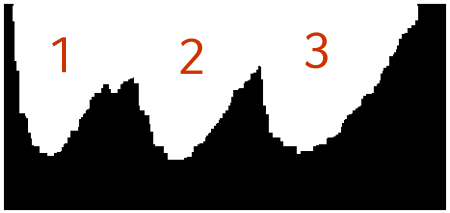
- Performance: approx. runtime(DBSCAN(ε, *MinPts*))
  - O(*n* * runtime(ε-neighborhood-query))
    - without spatial index support (worst case): O(n^2)
    - e.g. tree-based spatial index support: O($n * \log(n)$)

OPTICS: Parameter Sensitivity

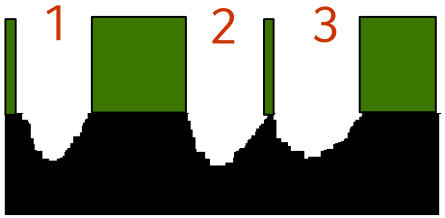
- Relatively insensitive to parameter settings
- Good result if parameters are just "large enough"
- ε is a tuning parameter – WHY???
- Variant DeLiClu[1] gets rid of ε parameter



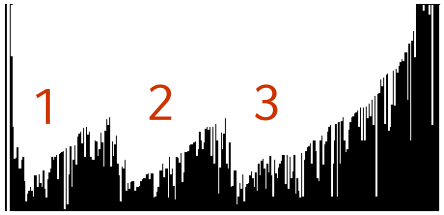
MinPts = 10, ε = 10



MinPts = 10, **ε = 5**



***MinPts* = 2**, ε = 10



[1] E. Achtert, C. Böhm, P. Kröger (2006). DeLiClu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking. In *PAKDD06, pp. 119–128.*

Hierarchical Clustering: Discussion

- Advantages
  - Does not require the number of clusters to be known in advance
  - No (standard methods) or very robust parameters (OPTICS/DeliClu)
  - Computes a complete hierarchy of clusters
  - Good result visualizations integrated into the methods
  - A "flat" partition can be derived afterwards (e.g. via a cut through the dendrogram or the reachability plot)

- Disadvantages
  - May not scale well
    - Runtime for the standard methods: $O(n^2 \log n^2)$
    - Runtime for OPTICS: without index support $O(n^2)$

      => DeliClu uses a closest-pair-join but still in $O(n^2)$ without index
  - User has to choose the final clustering