Un-Supervised Learning

Clustering Techniques

Credits : Some of the slides are taken from:

Data Analytics (CS40003) by Dr. Debasis Samanta

CS 578: Statistical Machine Learning by Yexiang Xue

Introduction to data mining, by Tan, Steinbach, and Kumar

University at buffalo NY state University

CSE 185 Introduction to Computer Vision
University of California MERCED

Usman RoshanQ   Assaf Gottlieb

Topics to be covered…

- Introduction to clustering
- Similarity and dissimilarity measures
- Clustering techniques
  - Hierarchical algorithms
  - **Partitioning algorithms (Kmedoids, Clara Calarans)**
  - **Density-based algorithm** (**DBScan, Optics , Denclue**).
  - Model Based algorithms (GMM)
  - Spectral Clustering

The k-Medoids algorithm

Now, we shall study a variant of partitioning algorithm called k-Medoids algorithm.

Motivation: We have learnt that the k-Means algorithm is sensitive to outliers because an object with an "extremely large value" may substantially distort the distribution. The effect is particularly exacerbated due to the use of the SSE (sum-of-squared error) objective function. The k-Medoids algorithm aims to diminish the effect of outliers.

Basic concepts:

- The basic concepts of this algorithm is to select an object as a cluster center (one representative object per cluster) instead of taking the mean value of the objects in a cluster (as in k-Means algorithm).
- We call this cluster representative as a cluster medoid or simply medoid.

1. Initially, it selects a random set of *k* objects as the set of medoids.
2. Then at each step, all objects from the set of objects, which are not currently medoids are examined one by one to see if they should be medoids.

The k-Medoids algorithm

- That is, the k-Medoids algorithm determines whether there is an object that should replace one of the current medoids.
- This is accomplished by looking all pair of medoid, non-medoid objects, and then choosing a pair that improves the objective function of clustering the best and exchange them.
- The sum-of-absolute error (SAE) function is used as the objective function.

$$SAE = \sum_{i=1}^{k} \sum_{x \in \boldsymbol{C}_i, x \notin M \text{ and } c_m \in M} |x - c_m|$$

Where c_m denotes a medoid

M is the set of all medoids at any instant

x is an object belongs to set of non-medoid object, that is, x belongs to some cluster and is not a medoid. i.e. $x \in \boldsymbol{C}_i , x \notin M$

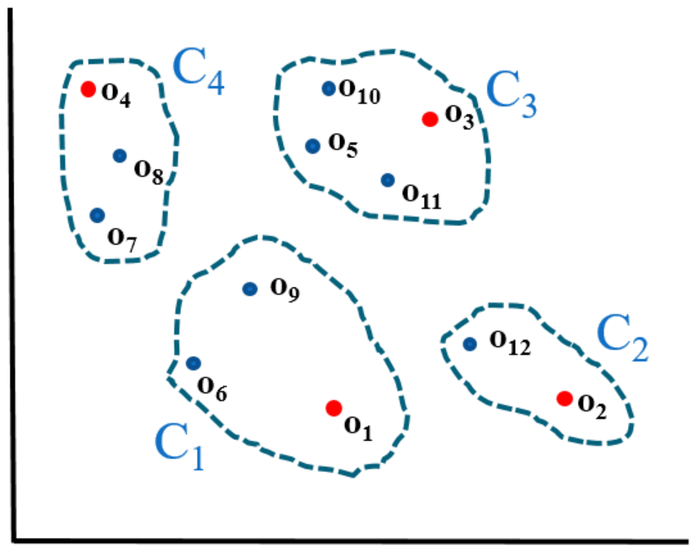
PAM (Partitioning around Medoids)

- For a given set of medoids, at any iteration, it select that exchange which has minimum SAE.
- The procedure terminates, if there is no any change in SAE in syuccessive iteration (i.e. there is no change in medoid).
- This k-Medoids algorithm is also known as PAM (Partitioning around Medoids).

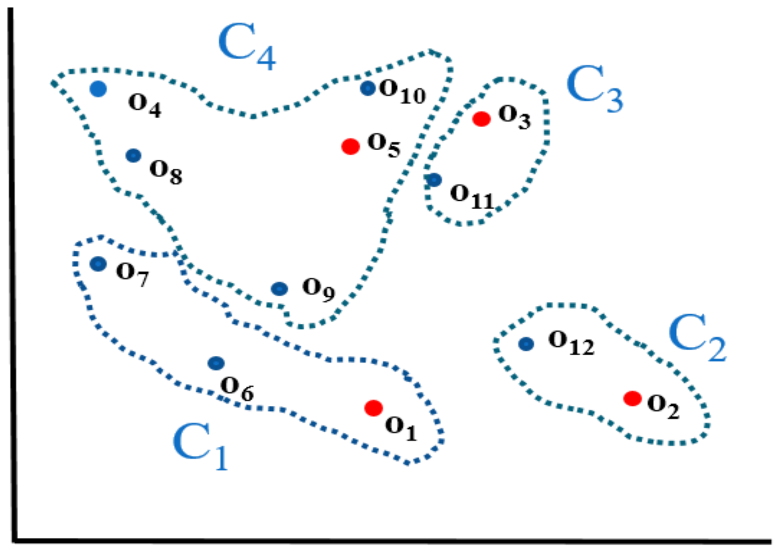
Illustration of PAM

- Suppose, there are set of 12 objects $O(o_1, o_2, \ldots, o_{12})$ and we are to cluster them into four clusters. At any instant, the four cluster C_1, C_2, C_3 *and* C_4 are shown in Fig. 16.7 (a). Also assume that $o_1, o_2, o_3,$ *and* o_4 are the medoids in the clusters C_1, C_2, C_3 *and* C_4, respectively. For this clustering we can calculate SAE.
- There are many ways to choose a non-medoid object to be replaced any one medoid object. Out of these, suppose, if o_5 is considered as candidate medoid instead of $o_4,$ then it gives the lowest SAE. Thus, the new set of medoids would be $o_1, o_2, o_3,$ *and* o_5. The new cluster is shown in Fig 16.7 (b).

PAM (Partitioning around Medoids)



(a) Cluster with $o_1, o_2, o_3, and\ o_4$ as medoids



(b) Cluster after swapping $o_4\ and\ o_5$ (o_5 becomes the new medoid).

Fig 16.7: Illustration of PAM

PAM (Partitioning around Medoids)

PAM algorithm is thus a procedure of iterative selection of medoids and it is precisely stated in Algorithm 16.2.

Algorithm 16.2: PAM

Input: Database of objects D.

k, the number of desired clusters.

Output: Set of k clusters

Steps:

```
1.  Arbitrarily select k medoids from D.
2.  **For** each object $o_i$ not a medoid **do**
3.      **For** each medoid $o_j$ **do**
4.  Let $M = \{o_1, o_2, \ldots, o_{i-1}, o_i, o_{i+1}, o_k\}$      //Set of current medoids
        $M' = \{o_1, o_2, \ldots, o_{j-1}, o_j, o_{j+1}, o_k\}$    //set of medoids but swap with non-medoids $o_j$
5.  Calculate $cost(o_i, o_j) = SAE|_M - SAE_{M'}$
6.      **End** of 2 for loop
```

PAM (Partitioning around Medoids)

Algorithm 16.2: PAM

7. Find o_i, o_j for which the $\text{cost}(o_i, o_j)$ is the smallest.
8. Replace o_i with o_j and accordingly update the set M.
9. Repeat step 2 - step 8 until cost(o_i, o_i) ≤ 0.
10. Return the cluster with M as the set of cluster centers.
11. Stop

Comments on PAM

1. Comparing k-Means with k-Medoids:

- Both algorithms needs to fix k, the number of cluster prior to the algorithms. Also, oth algorithm arbitrarily choose the initial cluster centroids.
- The k-Medoid method is more robust than k-Means in the presence of outliers, because a medoid is less influenced by outliers than a mean.

2. Time complexity of PAM:

- For each iteration, PAM consider $k(n-k)$ pairs of object o_i, o_j for which a cost $cost(o_i, o_j)$ determines. Calculating the cost during each iteration requires that the cost be calculated for all other non-medoids o_j. There are $n-k$ of these. Thus, the total time complexity per iteration is $n(n-k)^2$. The total number of iterations may be quite large.

3. Applicability of PAM:

- PAM does not scale well to large database because of its computation complexity.

Other variants of k-Medoids algorithms

- There are some variants of PAM that are targeted mainly large datasets are CLARA (Clustering LARge Applications) and CLARANS (Clustering Large Applications based upon RANdomized Search), it is an improvement of CLARA.

References:

For PAM and CLARA:

- L. kaufman and P. J. Rousseew, “Finding Groups in Data: An introduction to cluster analysis”, John and Wiley, 1990.

For CLARANS:

- R. Ng and J. Han, “Efficient and effective clustering method for spatial Data mining”, Proceeding very large databases [VLDB-94], 1994.

CLARA & CLARENS

K-Medoids Method

- Minimize the sensitivity of k-means to outliers
- Pick actual objects to represent clusters instead of mean values
- Each remaining object is clustered with the representative object (**Medoid**) to which is the most similar
- The algorithm minimizes the sum of the dissimilarities between each object and its corresponding reference point

$$E = \sum_{i-1}^{k} \sum_{p \in C_i} | p - o_i |$$

  - **E**: the sum of absolute error for all objects in the data set
  - **P**: the data point in the space representing an object
  - **O_i**: is the representative object of cluster C_i

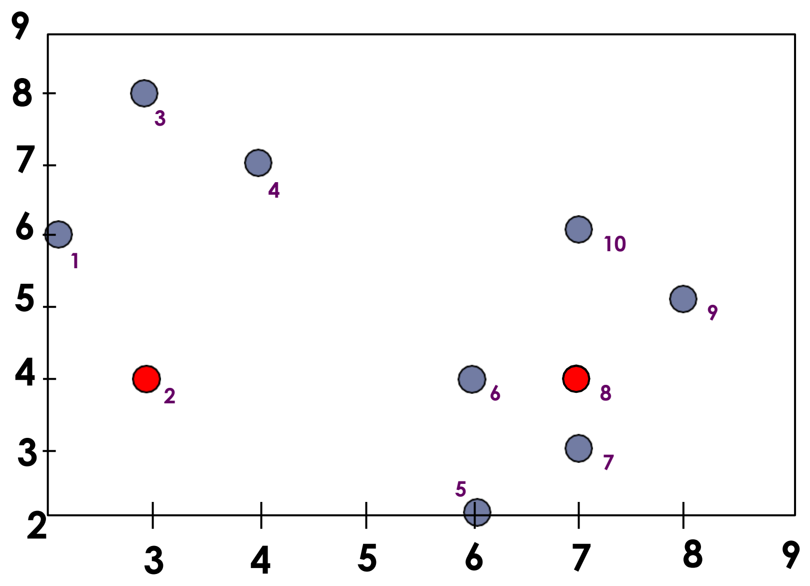
K-Medoids Method: The Idea

- Initial representatives are chosen randomly
- The iterative process of replacing representative objects by no representative objects continues as long as the quality of the clustering is improved
- For each representative Object O
  - For each non-representative object R, swap O and R
- Choose the configuration with the lowest cost
- Cost function is the difference in absolute error-value if a current representative object is replaced by a non-representative object

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



Goal: create two clusters

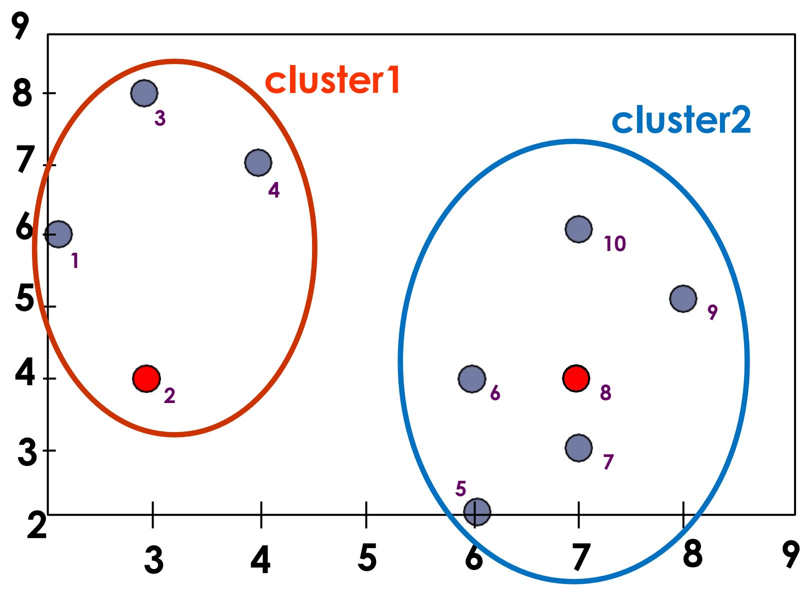
Choose randmly two medoids

$O_2 = (3,4)$
$O_8 = (7,4)$

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



- Assign each object to the closest representative object
- Using L1 Metric (Manhattan), we form the following clusters

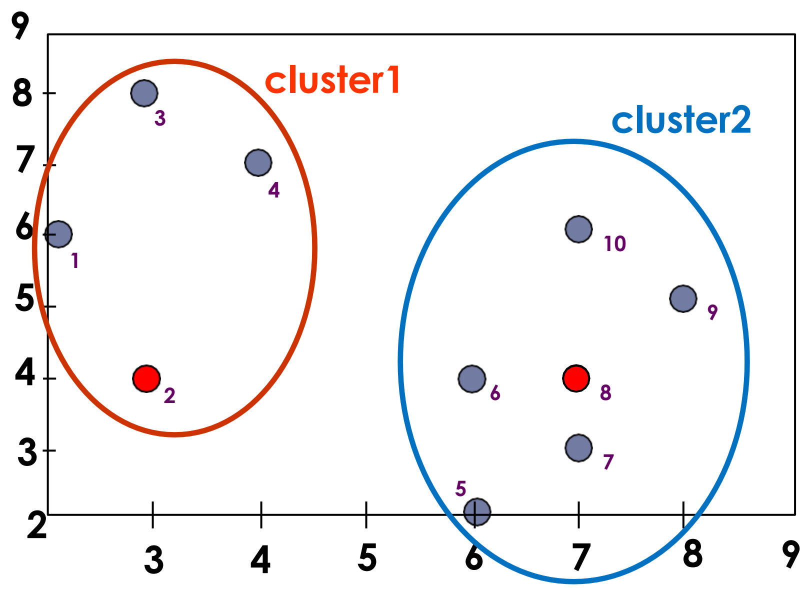
Cluster1 = $\{O_1, O_2, O_3, O_4\}$

Cluster2 = $\{O_5, O_6, O_7, O_8, O_9, O_{10}\}$

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



Compute the absolute error criterion **[for the set of Medoids (O2,O8)]**

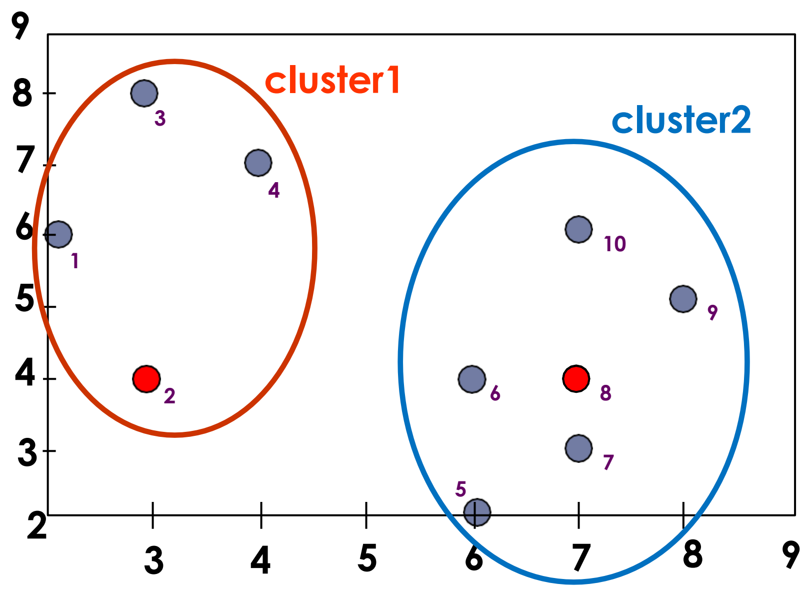
$$E=\sum_{i-1}^{k}\sum_{p\chi C_i}|p-o_i|=|o_1-o_2|+|o_3-o_2|+|o_4-o_2|$$

$$+|o_5-o_8|+|o_6-o_8|+|o_7-o_8|+|o_9-o_8|+|o_{10}-o_8|$$

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



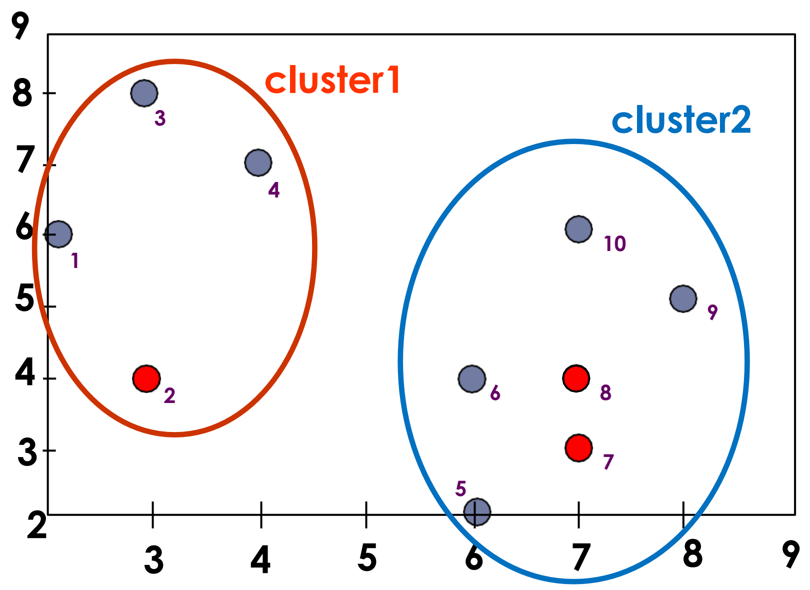
The absolute error criterion **[for the set of Medoids (O2,O8)]**

$$E = (3+4+4)+(3+1+1+2+2) = 20$$

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



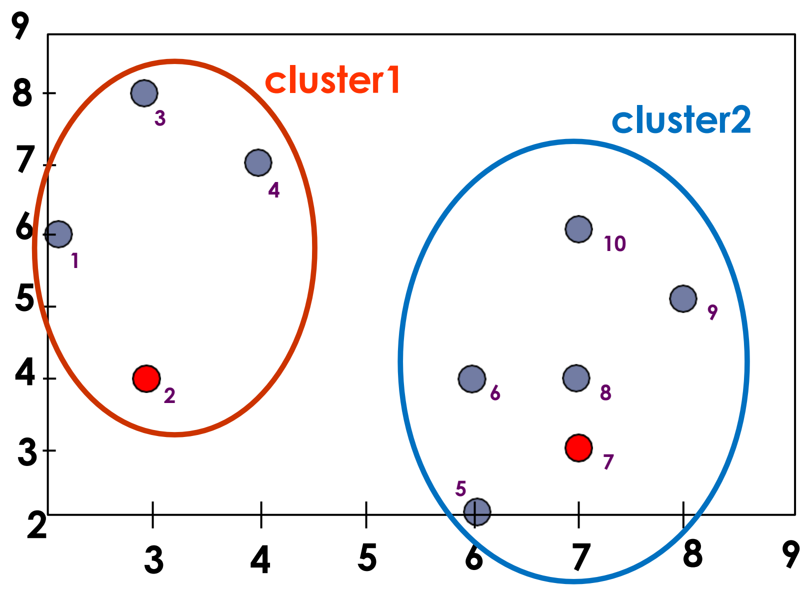
- Choose a random object O_7
- Swap **O8** and **O7**
- Compute the absolute error criterion **[for the set of Medoids (O2,O7)]**

$$E = (3+4+4)+(2+2+1+3+3) = 22$$

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



Compute the cost function

Absolute error [for O_2,O_7] – Absolute error [O_2,O_8]

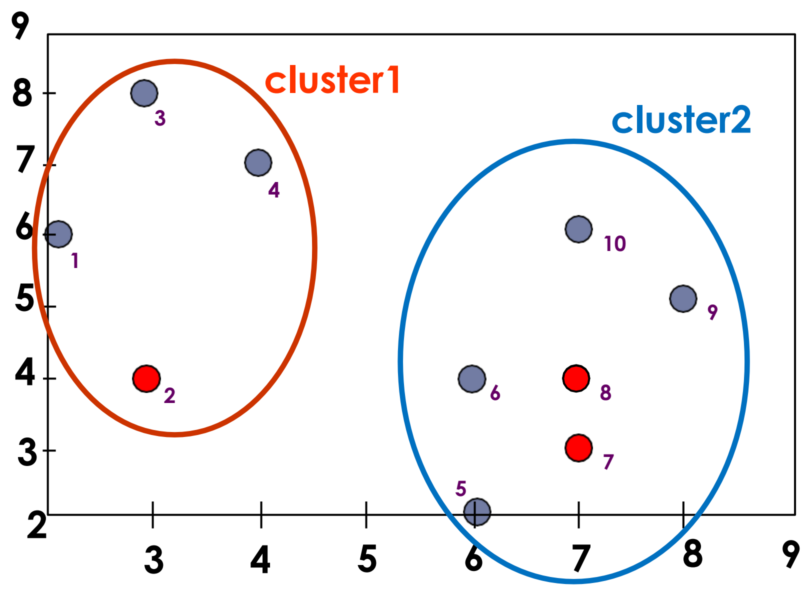
$$S = 22 - 20$$

S> 0 $\rightarrow$ it is a bad idea to replace O_8 by O_7

K-Medoids Method: Example

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



- In this example, changing the medoid of cluster 2 did not change the assignments of objects to clusters.
- What are the possible cases when we replace a medoid by another object?

K-Medoids Algorithm(PAM)

PAM : Partitioning Around Medoids

- **Input**
  - K: the number of clusters
  - D: a data set containing n objects
- **Output**: A set of k clusters
- **Method**:

(1) Arbitrary choose k objects from D as representative objects (seeds)

(2) Repeat

(3) Assign each remaining object to the cluster with the nearest representative object

(4) For each representative object O_j

(5) Randomly select a non representative object O_{random}

(6) Compute the total cost **S** of swapping representative object Oj with O_{random}

(7) if $S<0$ then replace O_j with O_{random}

(8) Until no change

K-Medoids Properties(k-medoids vs.K-means)

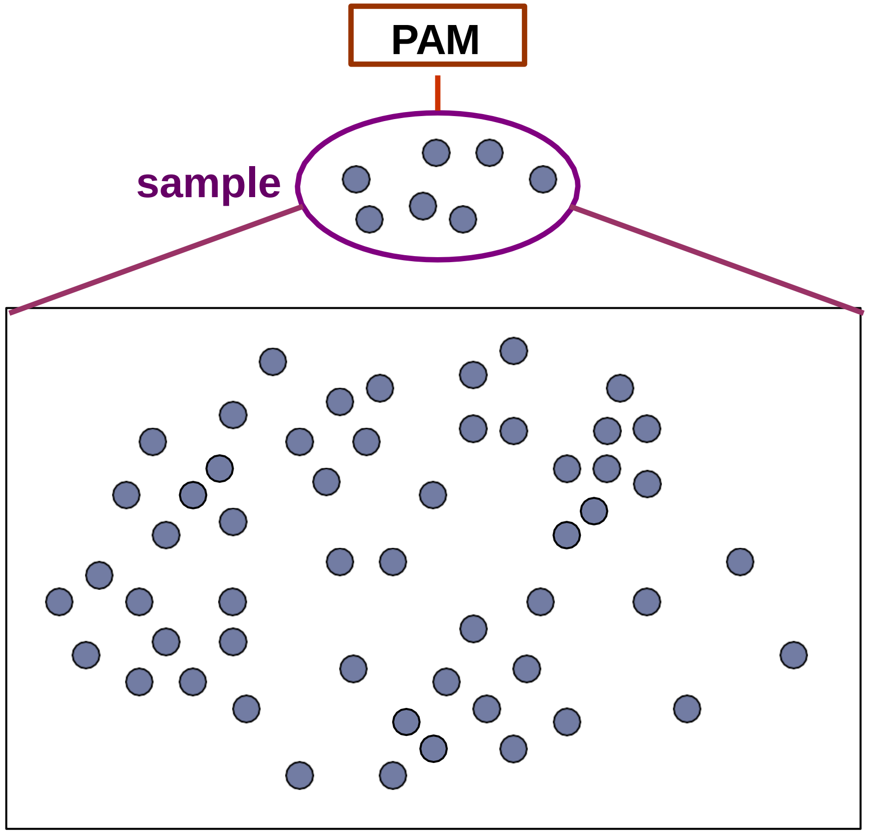
- The complexity of each iteration is **$O(k(n-k)^2)$**
- For large values of n and k, such computation becomes very costly
- **Advantages**
  - K-Medoids method is more robust than k-Means in the presence of noise and outliers
- **Disadvantages**
  - K-Medoids is more costly that the k-Means method
  - Like k-means, k-medoids requires the user to specify k
  - It does not scale well for large data sets

CLARA

- **CLARA (Clustering Large Applications)** uses a sampling-based method to deal with large data sets

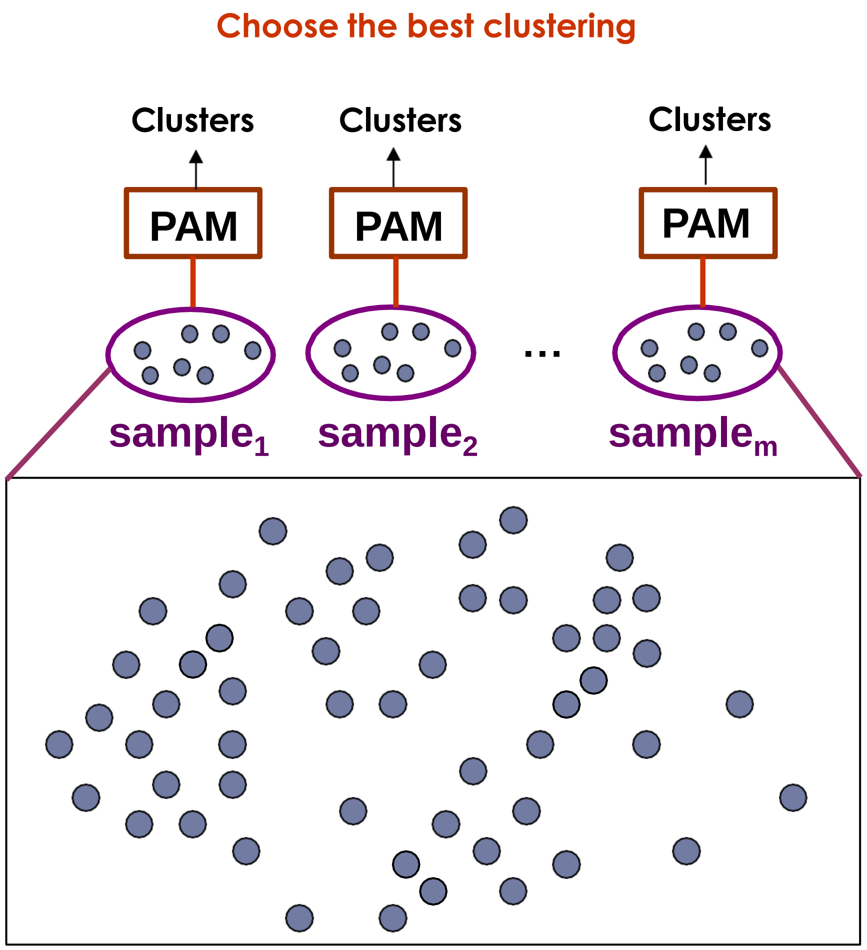
- A random sample should closely represent the original data

- The chosen medoids will likely be similar to what would have been chosen from the whole data set



CLARA

- Draw multiple samples of the data set
- Apply PAM to each sample
- Return the best clustering



CLARA - Algorithm

```
Set mincost to MAXIMUM;
Repeat q times // draws q samples

        Create S by drawing s objects randomly from D;
        Generate the set of medoids M from S by applying the PAM algorithm;
        Compute cost(M,D)
        If cost(M, D)<mincost
            Mincost = cost(M, D);
            Bestset = M;
        Endif;

Endrepeat;
Return Bestset;
```

CLARA Properties

- Complexity of each Iteration is: **$O(ks^2 + k(n-k))$**
  - **s**: the size of the sample
  - **k**: number of clusters
  - **n**: number of objects

- **PAM** finds the best k medoids among a given data, and **CLARA** finds the best k medoids among the selected samples

- **Problems**
  - The best k medoids may not be selected during the sampling process, in this case, CLARA will never find the best clustering
  - If the sampling is biased we cannot have a good clustering

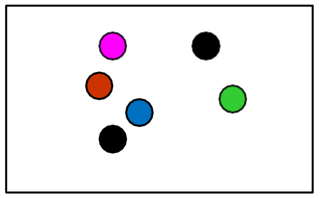
CLARANS

- **CLARANS (Clustering Large Applications based upon RANdomized Search)** was proposed to improve the quality and the scalability of CLARA
- It combines sampling techniques with PAM
- It does not confine itself to any sample at a given time
- It draws a sample with some randomness in each step of the search

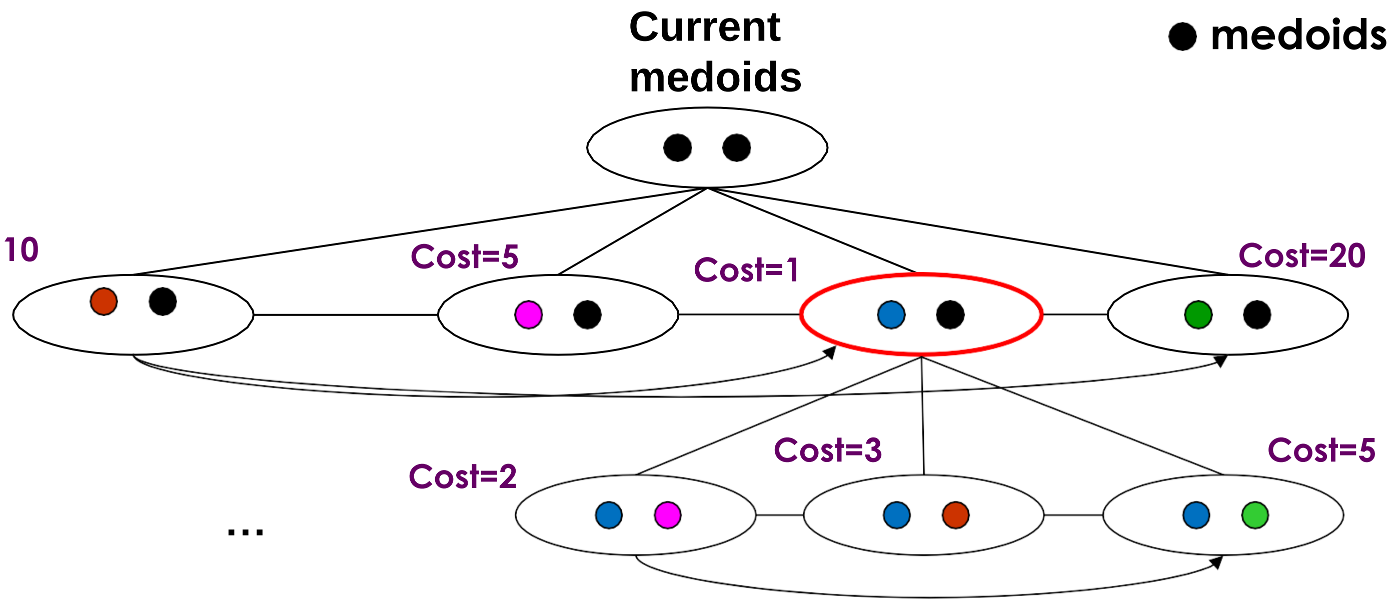
- CLARANS
- Does not confine the search to a localized area
- Stops the search when a local minimum is found
- Finds several local optimums and output the clustering with the best local optimum

CLARANS: The idea

Clustering view



● **medoids**



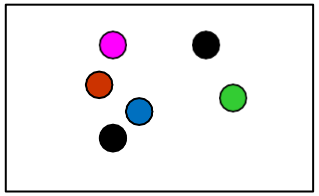
Keep the current medoids

CLARANS: The idea

CLARA

- Draws a sample of nodes at the beginning of the search
- Neighbors are from the chosen sample
- Restricts the search to a specific area of the original data

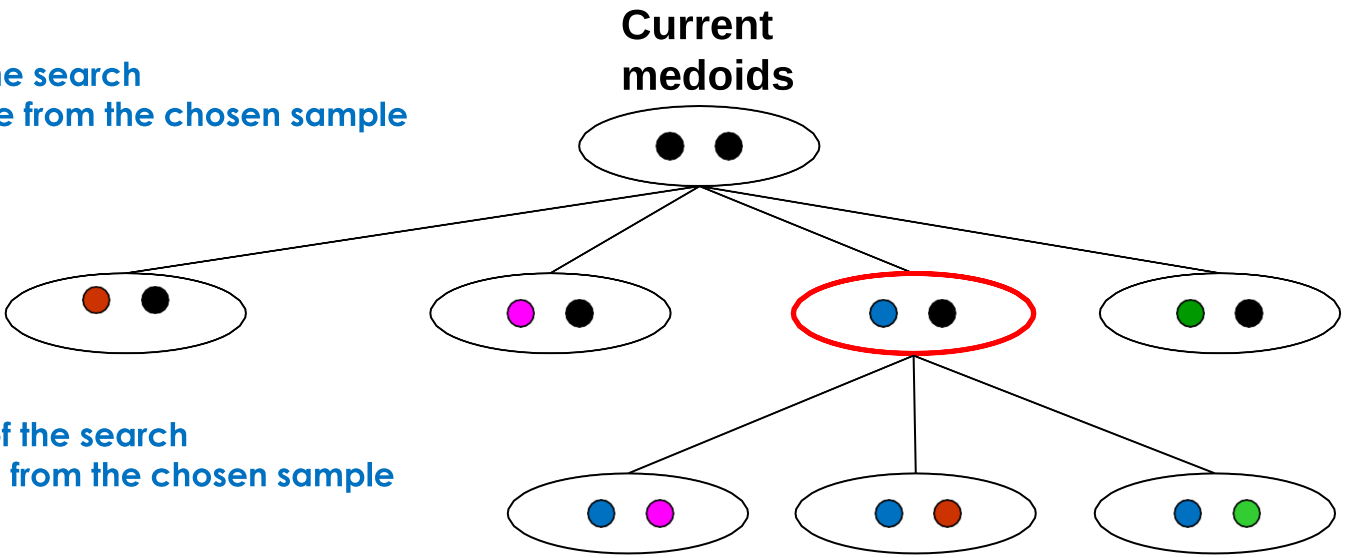
Sample



● **medoids**

First step of the search
Neighbors are from the chosen sample

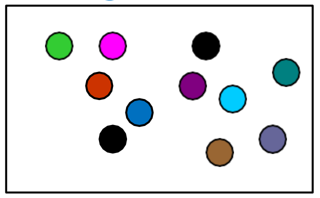
second step of the search
Neighbors are from the chosen sample



…

CLARANS: The idea

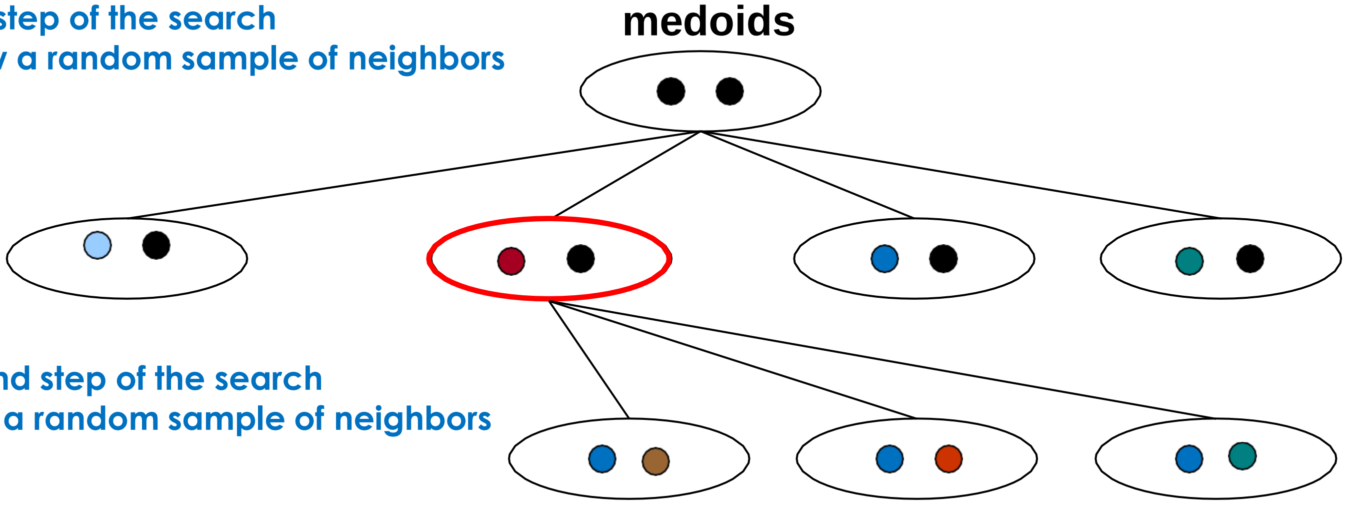
Original data



● medoids

First step of the search
Draw a random sample of neighbors

second step of the search
Draw a random sample of neighbors



…

The number of neighbors sampled from the original data is specified by the user

CLARANS - Algorithm

- Set mincost to MAXIMUM;
- For i=1 to h do // find h local optimum

```
Randomly select a node as the current node C in the graph;
J = 1;  // counter of neighbors
Repeat
        Randomly select a neighbor N of C;
        If Cost(N,D)<Cost(C,D)
                Assign N as the current node C;
                J = 1;
        Else  J++;
        Endif;
Until J > m
Update mincost with Cost(C,D) if applicableEnd for;
```

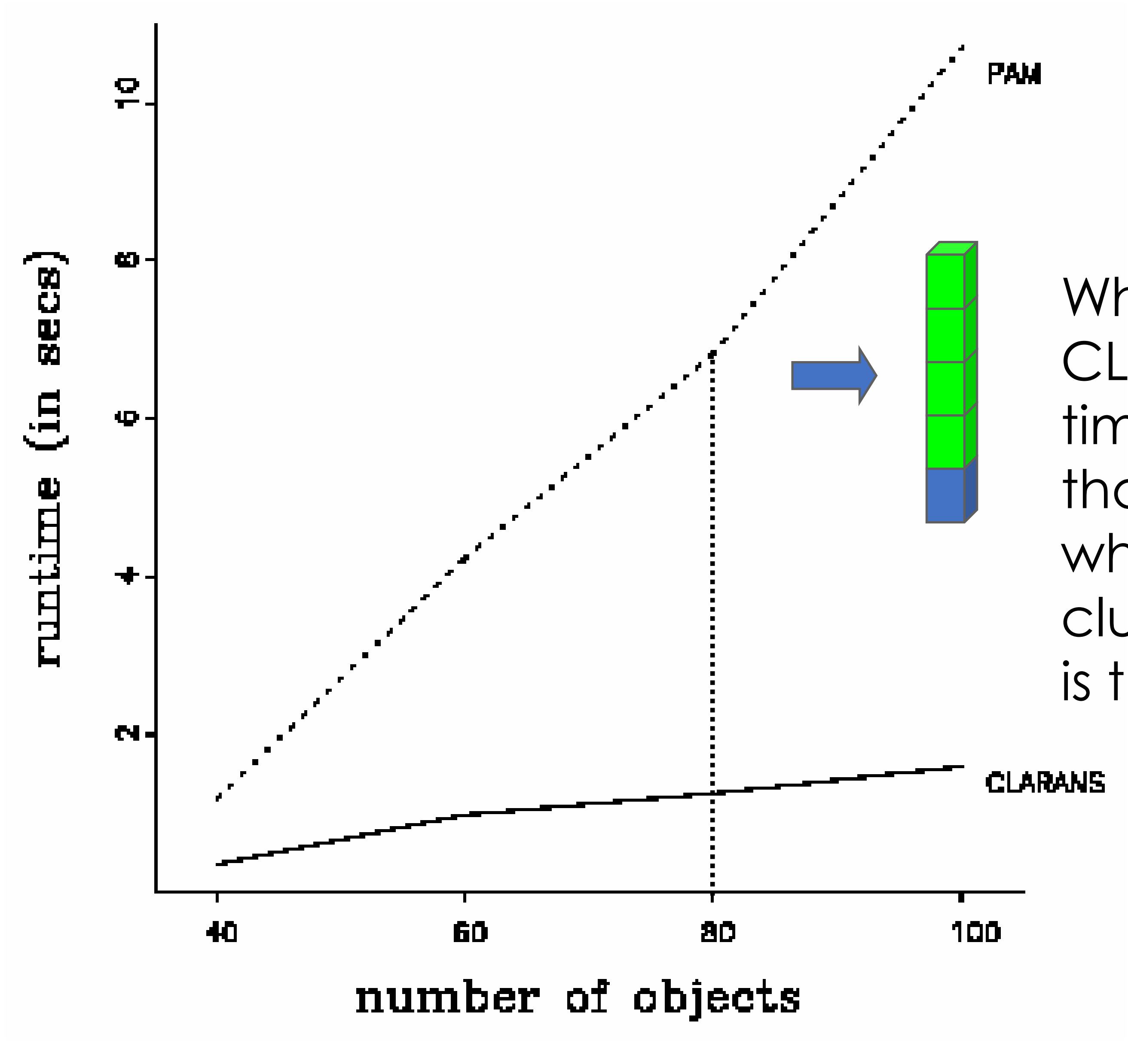
- End For
- Return bestnode;

CLARANS Properties

- **Advantages**
  - Experiments show that CLARANS is more effective than both PAM and CLARA
  - Handles outliers
- **Disadvantages**
  - The computational complexity of CLARANS is **$O(n^2)$**, where n is the number of objects
  - The clustering quality depends on the sampling method

Comparison with PAM

- For large and medium data sets, it is obvious that CLARANS is much more efficient than PAM
- For small data sets, CLARANS outperforms PAM significantly



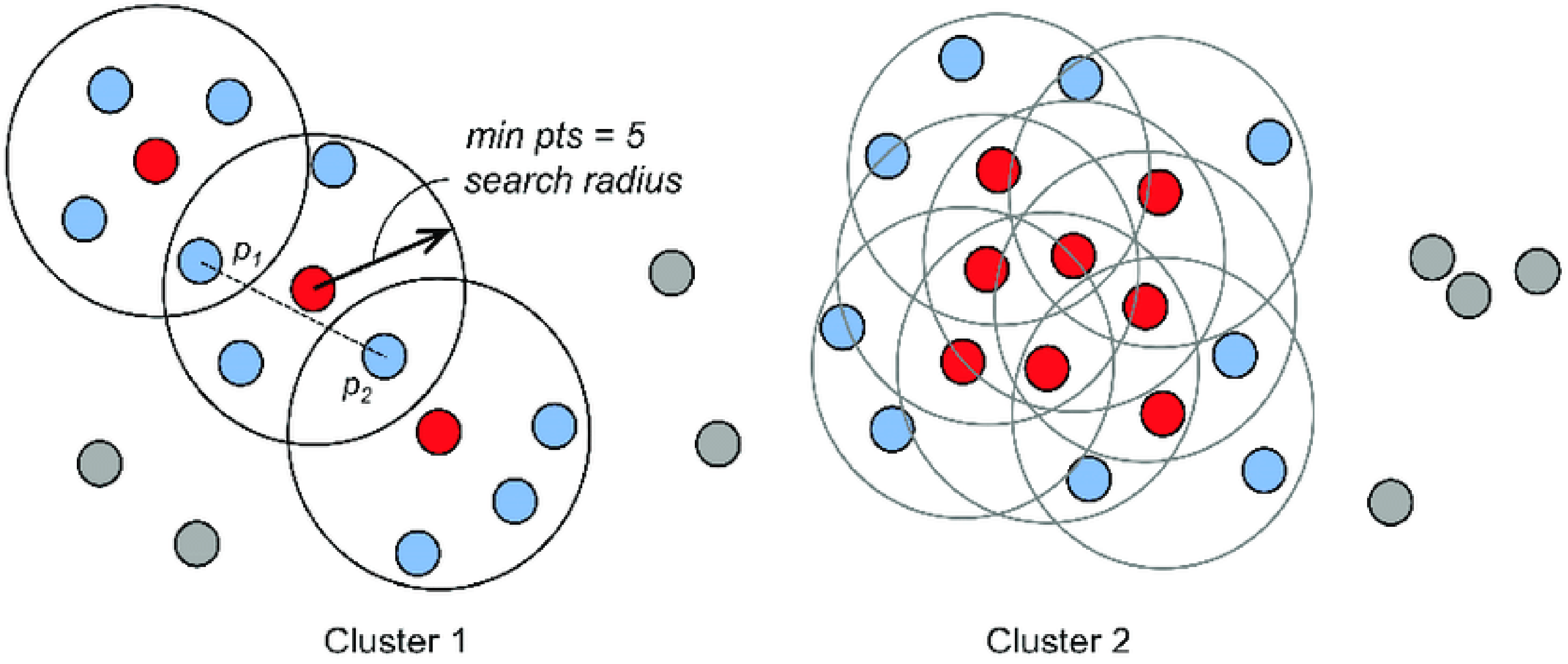
When n=80, CLARANS is 5 times faster than PAM, while the cluster quality is the same.

Comparison with CLARA

- CLARANS is always able to find clusterings of better quality than those found by CLARA; CLARANS may use much more time than CLARA

- When the time used is the same, CLARANS is still better than CLARA

Density Based Clustering - DBScan



Density-based Approaches

- Why Density-Based Clustering methods?
  - Discover clusters of arbitrary shape.
  - Clusters – Dense regions of objects separated by regions of low density
- DBSCAN – the first density-based clustering
- OPTICS – density-based cluster-ordering
- DENCLUE – a general density-based description of cluster and clustering

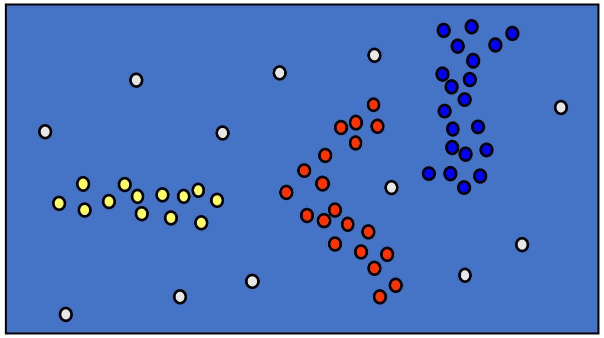
DBSCAN: Density Based Spatial Clustering of Applications with Noise

- Proposed by Ester, Kriegel, Sander, and Xu (KDD96)
- Relies on a density-based notion of cluster: A cluster is defined as a maximal set of density-connected points.
- Discovers clusters of arbitrary shape in spatial databases with noise

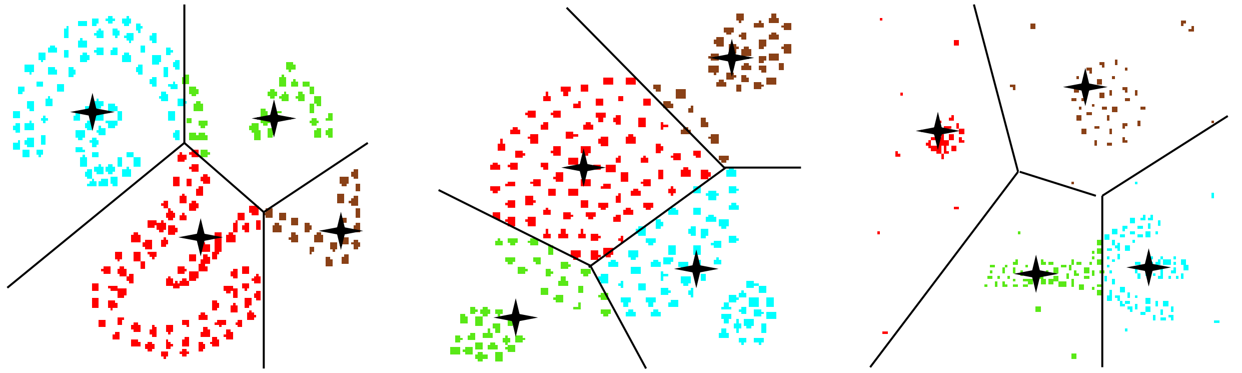
Density-Based Clustering

- *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

Density Based Clustering: Basic Concept

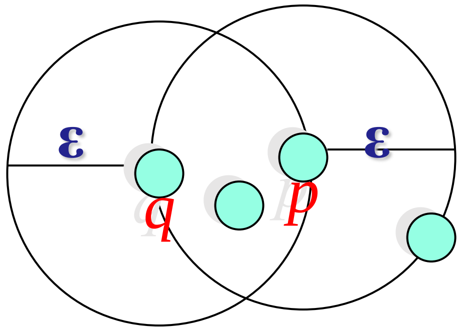
- Intuition for the formalization of the basic idea
  - For any point in a cluster, the local point density around that point has to exceed some threshold
  - The set of points from one cluster is spatially connected
- Local point density at a point *p* defined by two parameters
  - ε – radius for the neighborhood of point p:
    $N_{\varepsilon}(p) := \{q \text{ in data set } D \mid dist(p, q) \leq \varepsilon\}$
  - *MinPts* – minimum number of points in the given neighbourhood *N*(*p*)

ε-Neighborhood

- ε-Neighborhood – Objects within a radius of ε from an object.

$$N_{\varepsilon}(p):\{q \mid d(p,q) \le \varepsilon\}$$

- “High density” - ε-Neighborhood of an object contains at least *MinPts* of objects.



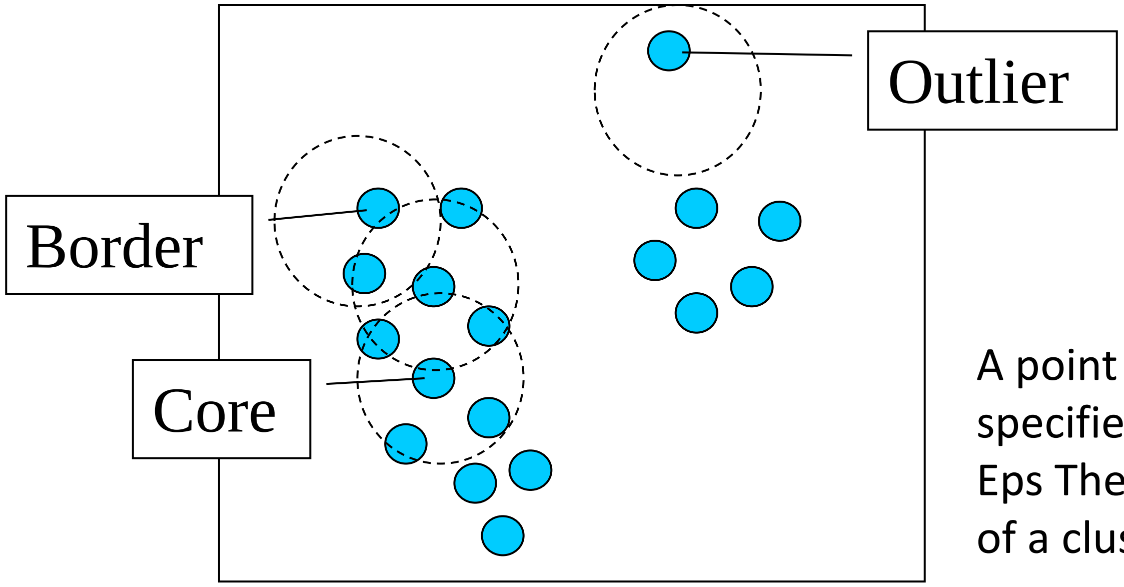
ε-Neighborhood of *p*

ε-Neighborhood of *q*

Density of p is “high” (MinPts = 4)

Density of q is “low” (MinPts = 4)

Core, Border & Outlier



ε = 1unit, MinPts = 5

Given ε and *MinPts*, categorize the objects into three exclusive groups.

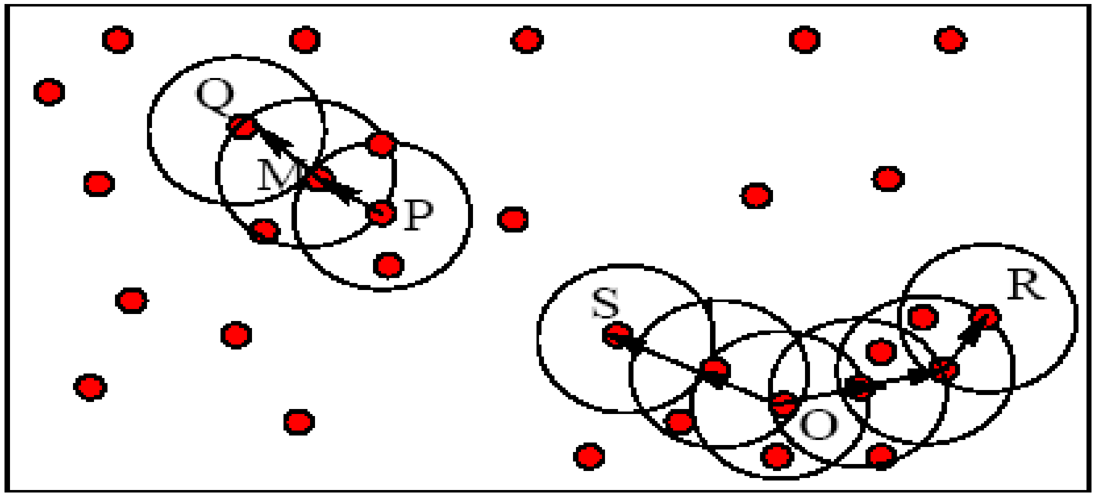
A point is a core point if it has more than a specified number of points (MinPts) within Eps These are points that are at the interior of a cluster.

A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point.

A noise point is any point that is not a core point nor a border point.

Example

- M, P, O, and R are core objects since each is in an Eps neighborhood containing at least 3 points

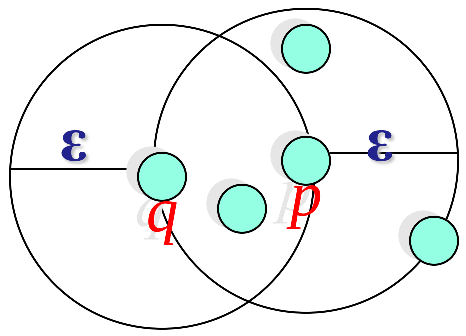


Minpts = 3

Eps=radius of the circles

Density-Reachability

- **Directly density-reachable**
  - **An object q is directly density-reachable from object p if p is a core object and q is in p’s ε-neighborhood.**

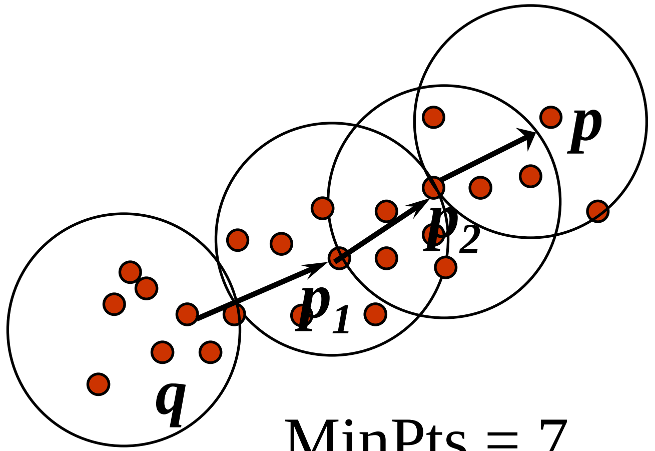


MinPts = 4

- **q is directly density-reachable from p**
- **p is not directly density- reachable from q?**
- **Density-reachability is asymmetric.**

Density-reachability

- Density-Reachable (directly and indirectly):
  - A point p is directly density-reachable from p2;
  - p2 is directly density-reachable from p1;
  - p1 is directly density-reachable from q;
  - p←p2←p1←q form a chain.

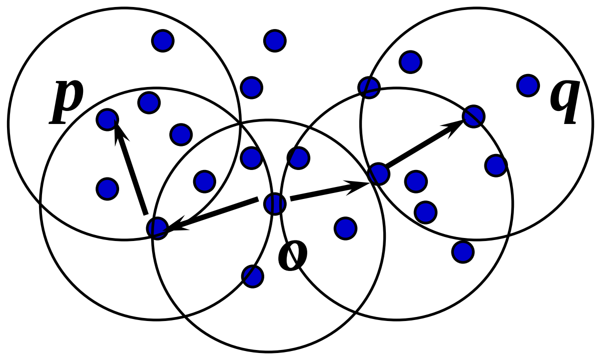


MinPts = 7

- **p is (indirectly) density-reachable from q**
- **q is not density- reachable from p?**

Density-Connectivity

- **Density-reachable is not symmetric**
  - **not good enough to describe clusters**
- **Density-Connected**
  - **A pair of points p and q are density-connected if they are commonly density-reachable from a point o.**



- **Density-connectivity is symmetric**

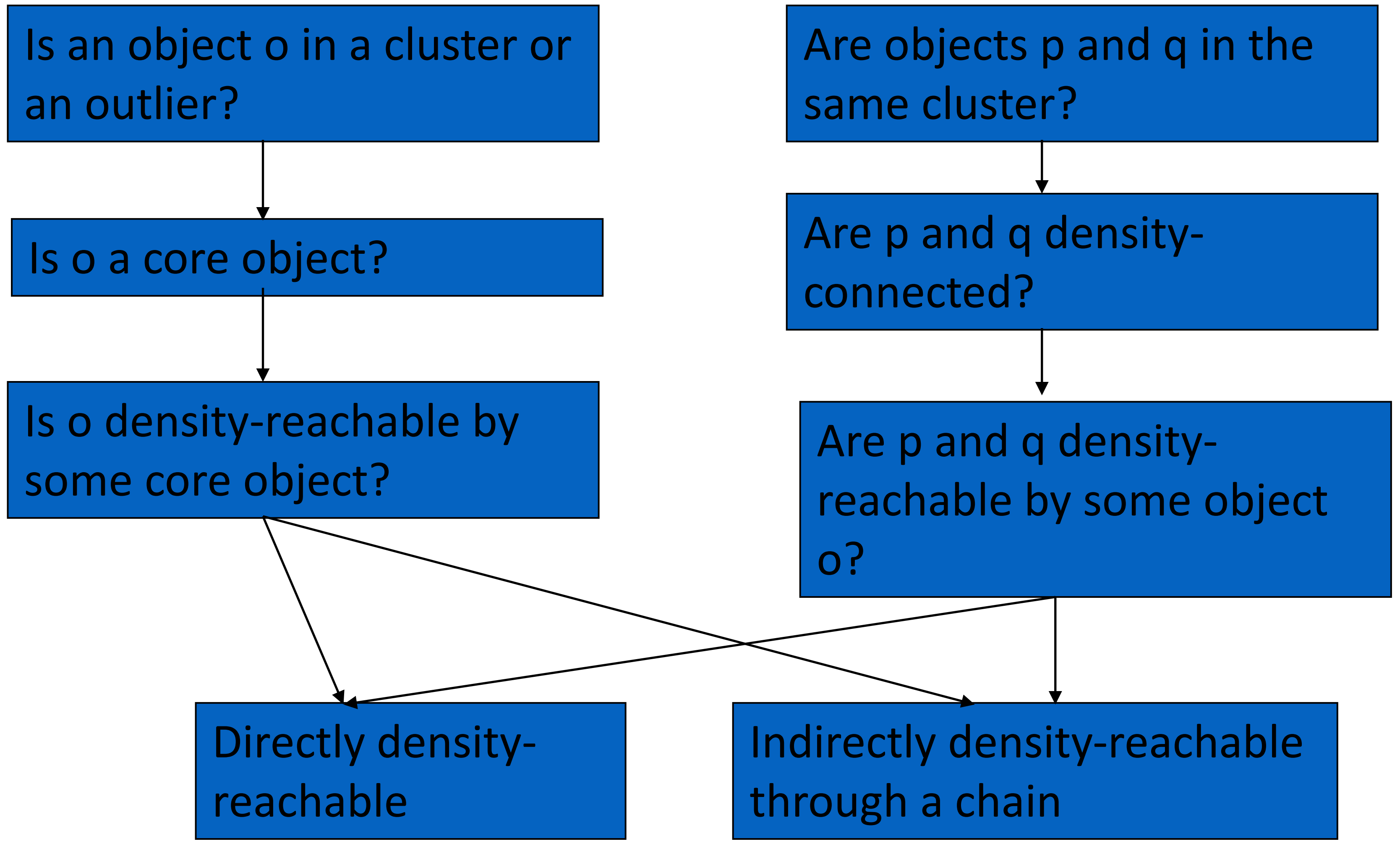
Formal Description of Cluster

- Given a data set D, parameter ε and threshold MinPts.
- A cluster C is a subset of objects satisfying two criteria:
  - *Connected:* V p,q ϵ C: p and q are density-connected.
  - *Maximal:* V p,q: if p ϵ C and q is <u>density-reachable from p</u>, then q ϵ C. (avoid redundancy)

$\downarrow$

P is a core object.

Review of Concepts


Is an object o in a cluster or an outlier?
Is o a core object?
Is o density-reachable by some core object?
Are objects p and q in the same cluster?
Are p and q density-connected?
Are p and q density-reachable by some object o?
Directly density-reachable
Indirectly density-reachable through a chain

DBSCAN Algorithm

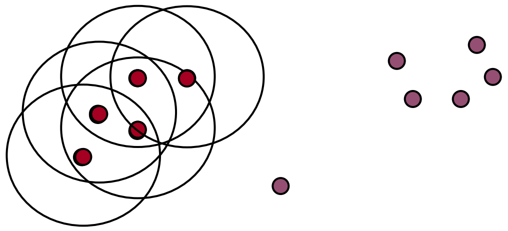
```
Input: The data set D
Parameter: $\varepsilon$, MinPts
For each object p in D
    if p is a core object and not processed then
        C = retrieve all objects density-reachable from p
        mark all objects in C as processed
        report C as a cluster
    else mark p as outlier
    end if
End For
```

DBSCAN: The Algorithm

- Arbitrary select a point *p*
- Retrieve all points density-reachable from *p* wrt *Eps* and *MinPts*.
- If *p* is a core point, a cluster is formed.
- If *p* is a border point, no points are density-reachable from *p* and DBSCAN visits the next point of the database.
- Continue the process until all of the points have been processed.

DBSCAN Algorithm: Example

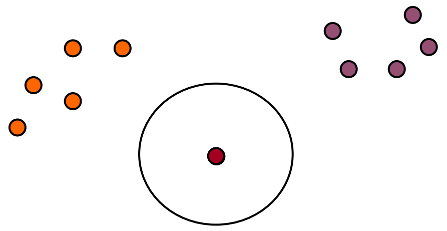
- Parameter
  - ε = 2 cm
  - *MinPts* = 3



```
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

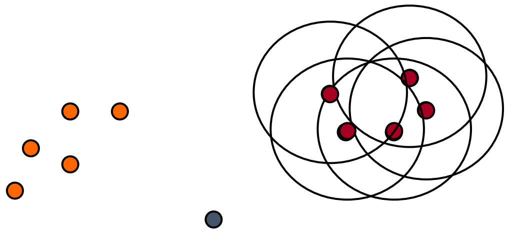
- Parameter
  - ε = 2 cm
  - *MinPts* = 3



```
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
```

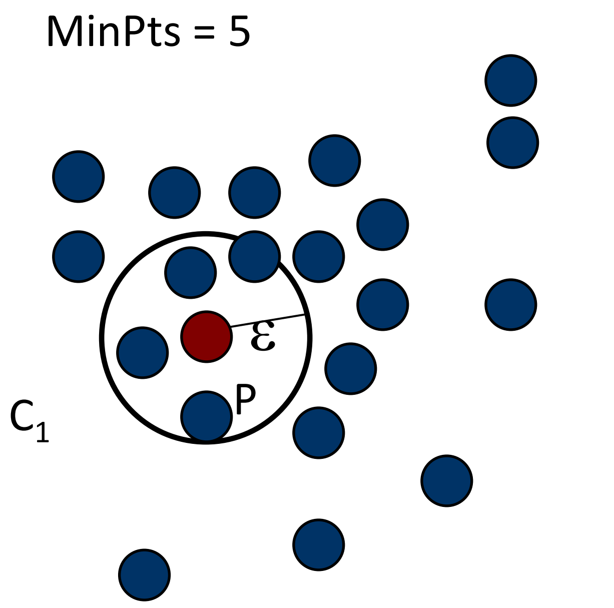
DBSCAN Algorithm: Example

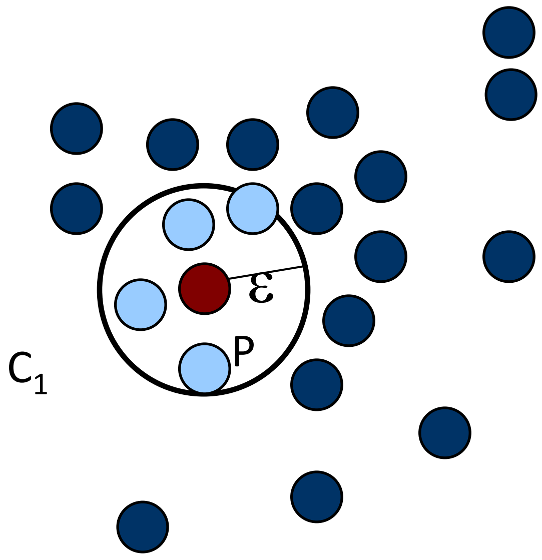
- Parameter
    - ε = 2 cm
    - *MinPts* = 3

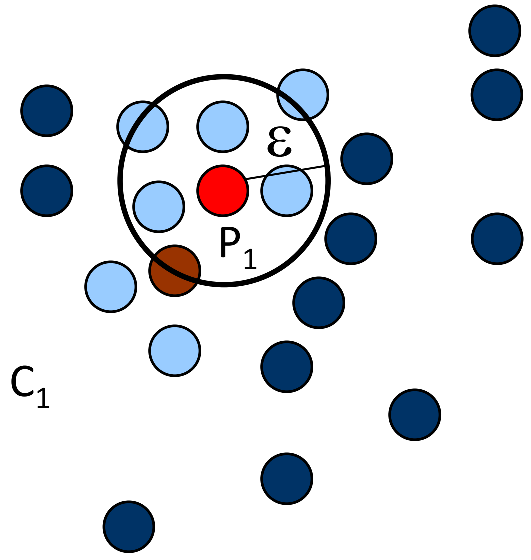


```
for each $o \in D$ do
    if $o$ is not yet classified then
        if $o$ is a core-object then
            collect all objects density-reachable
from $o$
            and assign them to a new cluster.
        else
            assign $o$ to NOISE
```

MinPts = 5

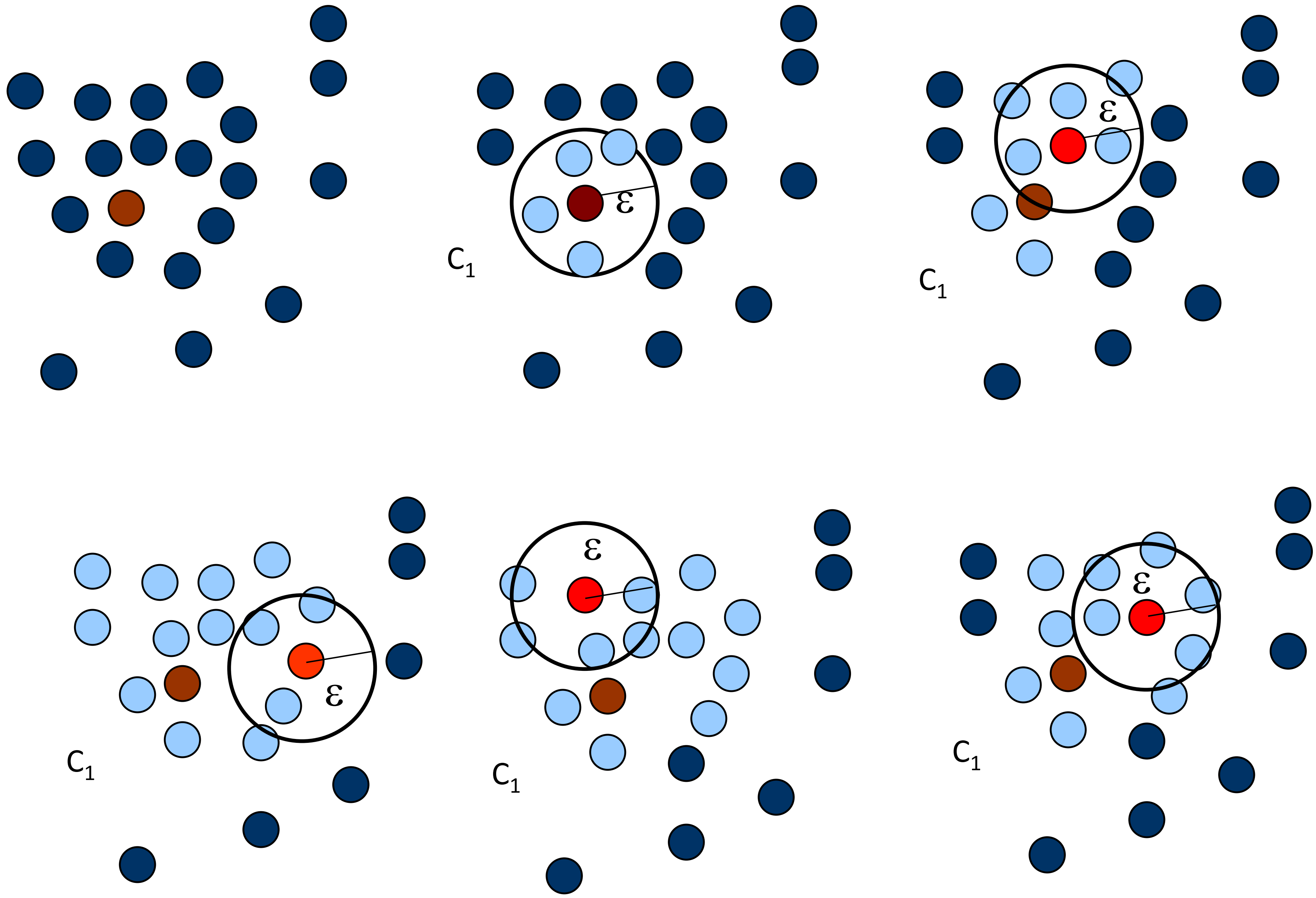




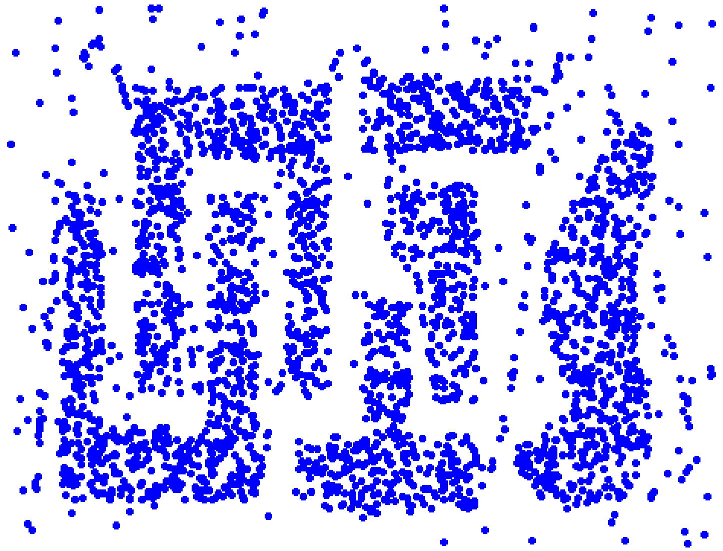


1. Check the ε-neighborhood of p;
2. If p has less than MinPts neighbors then mark p as outlier and continue with the next object
3. Otherwise mark p as processed and put all the neighbors in cluster C

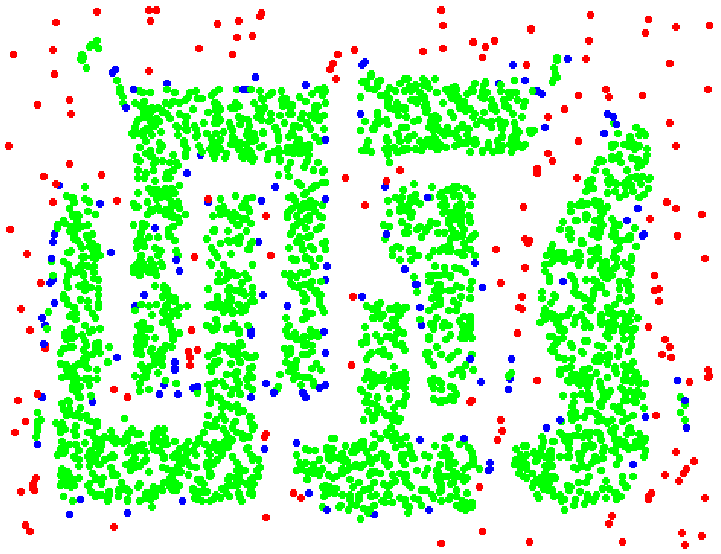
1. Check the unprocessed objects in C
2. If no core object, return C
3. Otherwise, randomly pick up one core object p_1, mark p_1 as processed, and put all unprocessed neighbors of p_1 in cluster C


ε
C$_1$
ε
C$_1$
C$_1$
ε
ε
C$_1$
ε
C$_1$
ε
C$_1$

Example



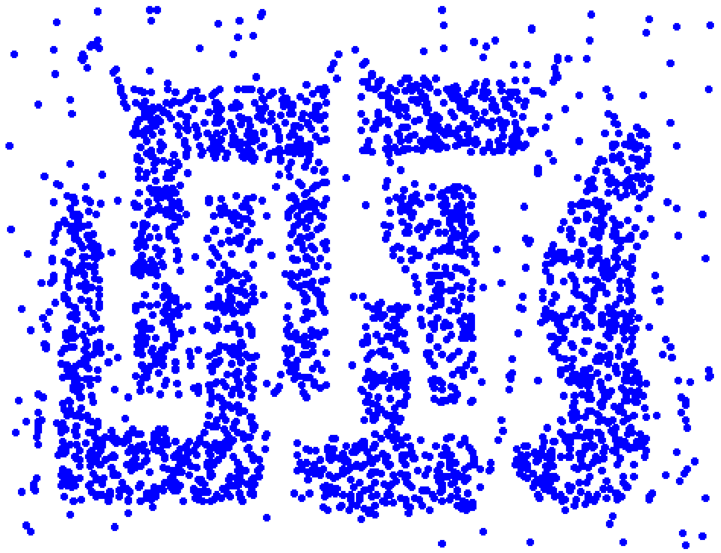
Original Points



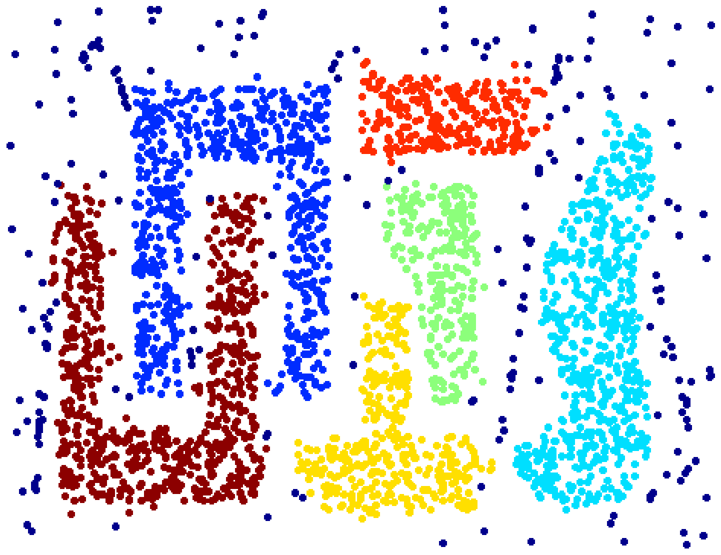
Point types: core, border and outliers

ε = 10, MinPts = 4

When DBSCAN Works Well



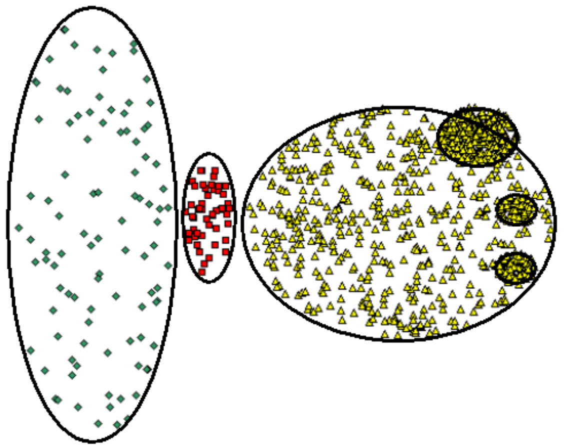
Original Points



Clusters

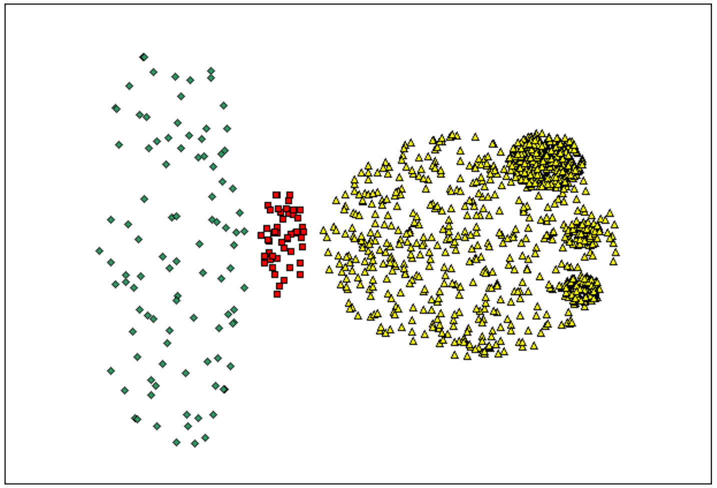
- **Resistant to Noise**
- **Can handle clusters of different shapes and sizes**

When DBSCAN Does NOT Work Well

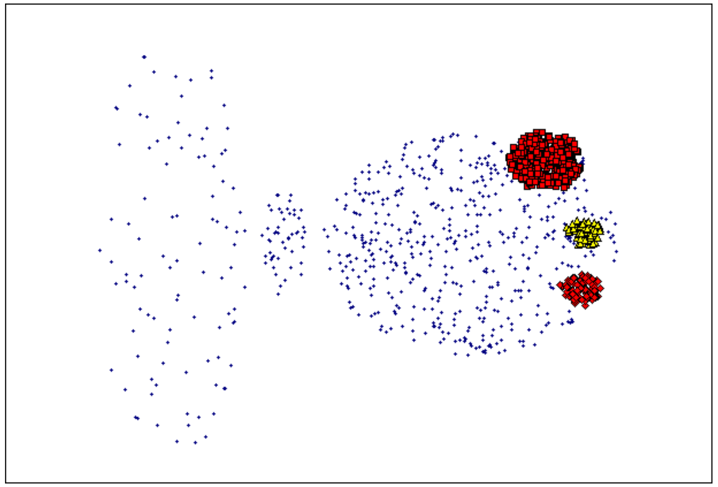


Original Points

- **Cannot handle Varying densities**
- **sensitive to parameters**



(MinPts=4, Eps=9.92).



(MinPts=4, Eps=9.75)

DBSCAN: Sensitive to Parameters

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

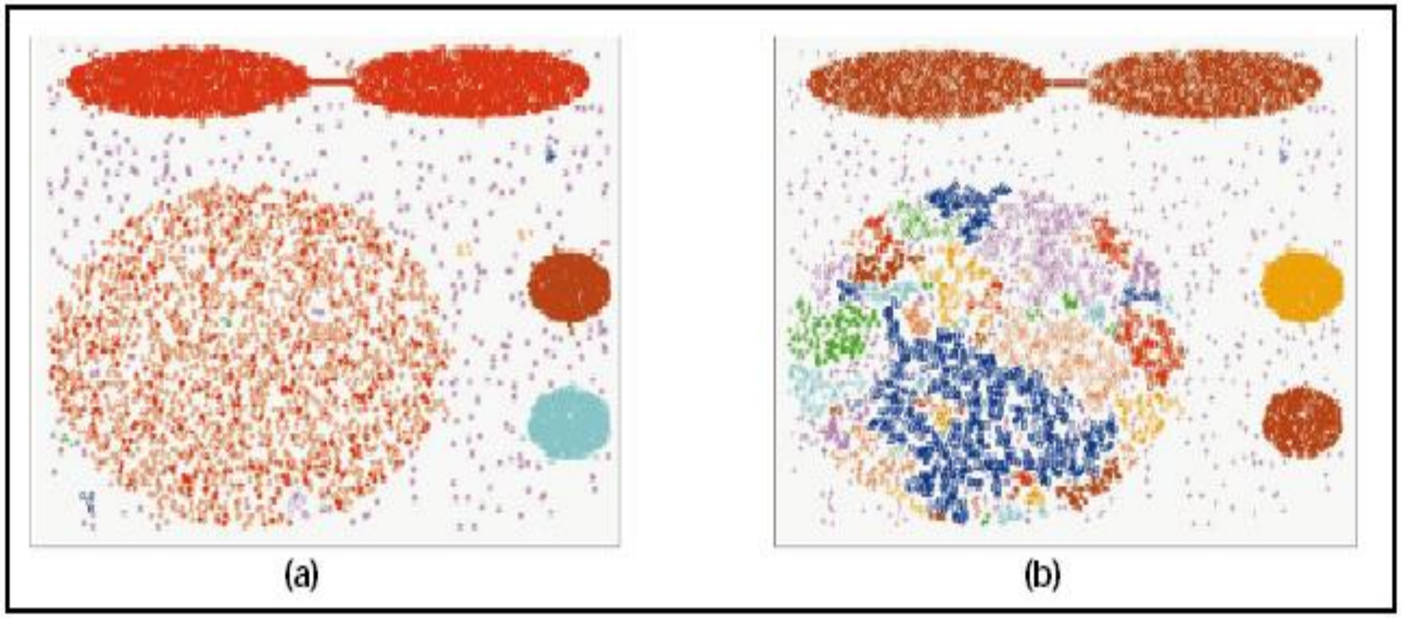
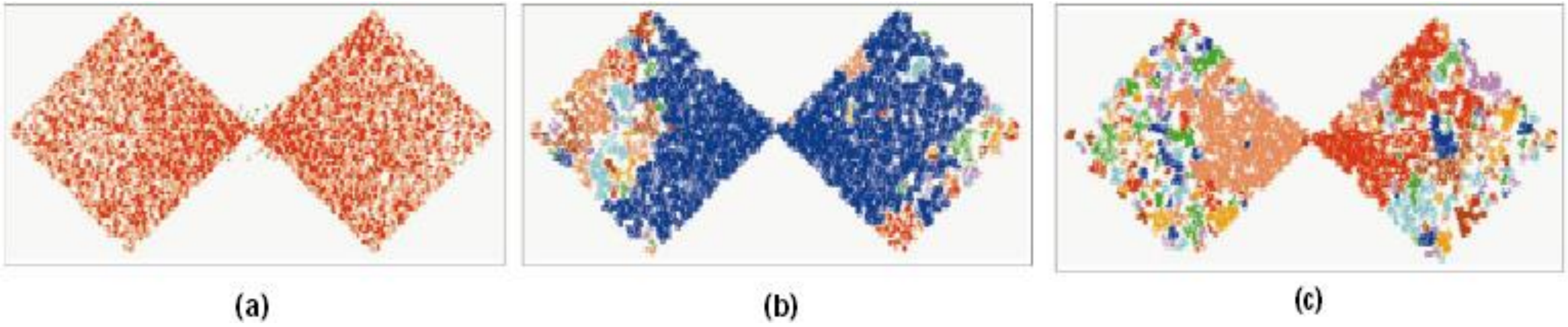
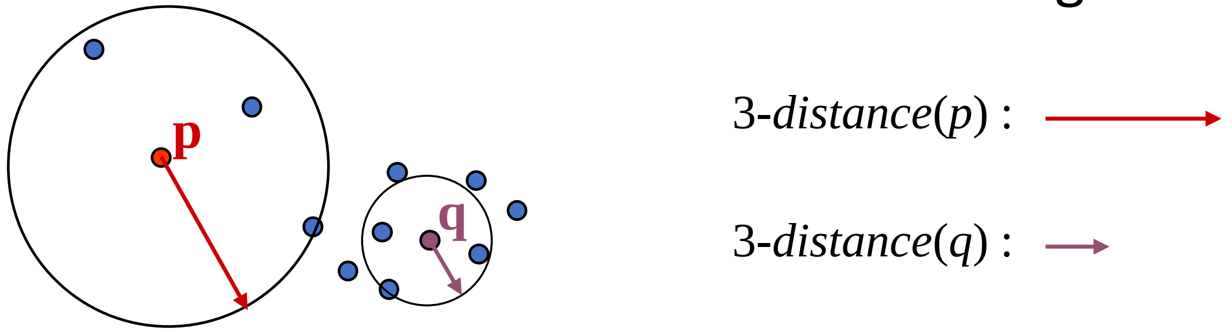


Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.



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

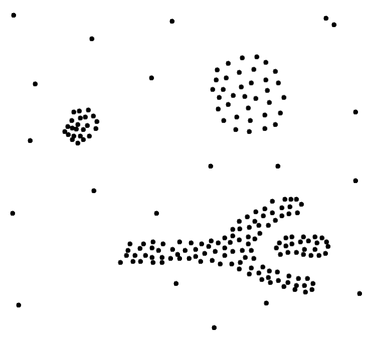


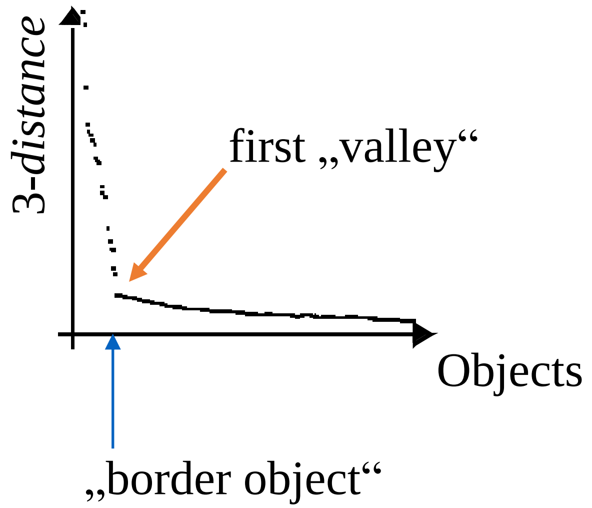
- Function *k-distance*(*p*): distance from *p* to the its *k*-nearest neighbor
- *k-distance plot*: *k*-distances of all objects, sorted in decreasing order

https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Determining the Parameters ε and *MinPts*

- Example *k*-distance plot

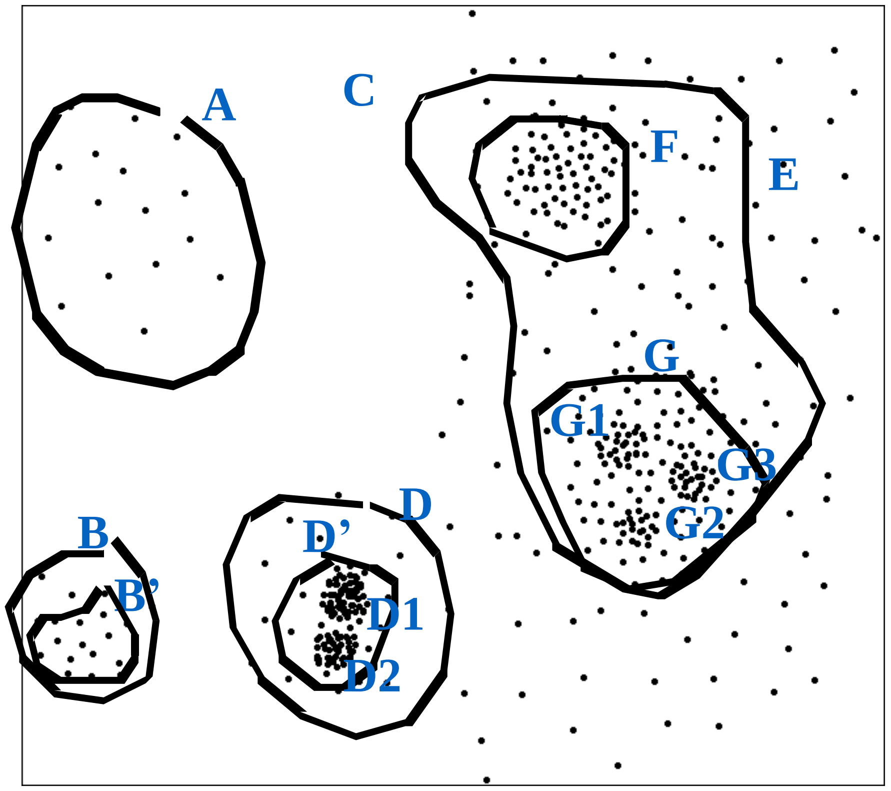


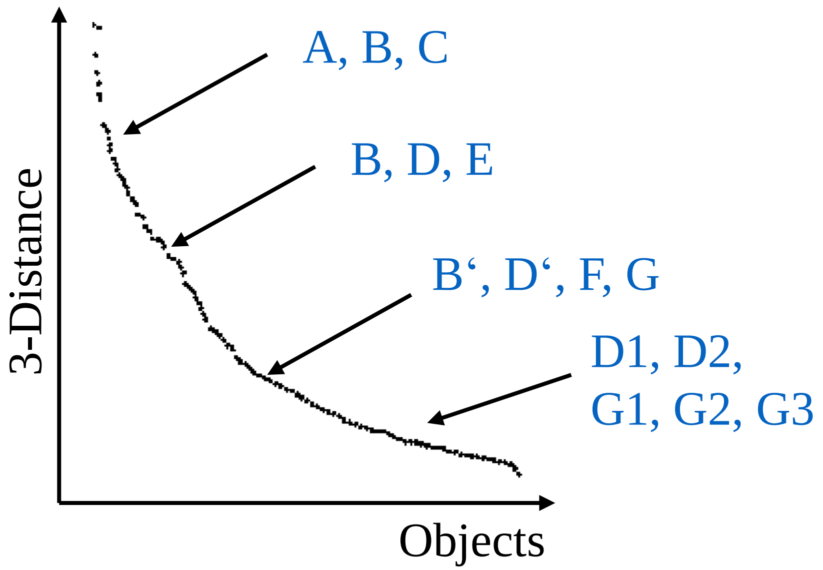


- Heuristic method:
  - Fix a value for *MinPts* (default: $2 \times d - 1$)
  - User selects “border object” *o* from the *MinPts-distance* plot; ε is set to *MinPts-distance*(o)

Determining the Parameters ε and *MinPts*

- Problematic example





Density Based Clustering: Discussion

- Advantages
  - Clusters can have arbitrary shape and size
  - Number of clusters is determined automatically
  - Can separate clusters from surrounding noise
  - Can be supported by spatial index structures
- Disadvantages
  - Input parameters may be difficult to determine
  - In some situations very sensitive to input parameter setting

OPTICS: Ordering Points To Identify the Clustering Structure

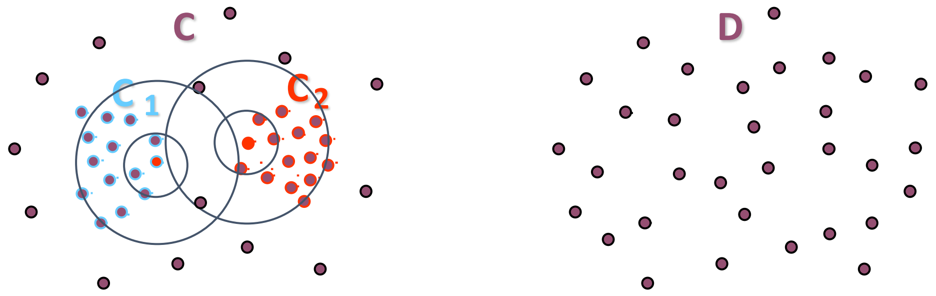
- DBSCAN
  - Input parameter – hard to determine.
  - Algorithm very sensitive to input parameters.
- OPTICS – Ankerst, Breunig, Kriegel, and Sander (SIGMOD'99)
  - Based on DBSCAN.
  - Does not produce clusters explicitly.
  - Rather generate an ordering of data objects representing density-based clustering structure.

OPTICS con’t

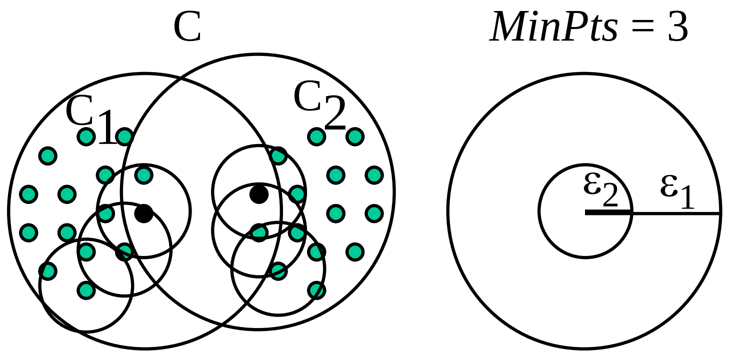
- Produces a special order of the database wrt its density-based clustering structure
- This cluster-ordering contains info equiv to the density-based clusterings corresponding to a broad range of parameter settings
- Good for both automatic and interactive cluster analysis, including finding intrinsic clustering structure
- Can be represented graphically or using visualization techniques

Density-Based Hierarchical Clustering

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



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

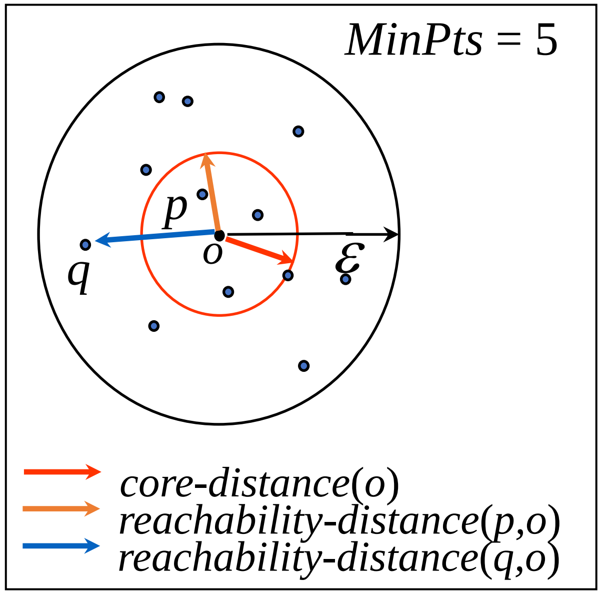


Core- and Reachability Distance

- Parameters: "generating" distance ε, fixed value *MinPts*

- *core-distance*$_{\varepsilon,MinPts}$(*o*)
  "smallest distance such that *o* is a core object"
  (if that distance is $\leq \varepsilon$; "?" otherwise)

- *reachability-distance*$_{\varepsilon,MinPts}$(*p, o*)
  "smallest distance such that *p* is
  directly density-reachable from *o*"
  (if that distance is $\leq \varepsilon$; "?" otherwise)



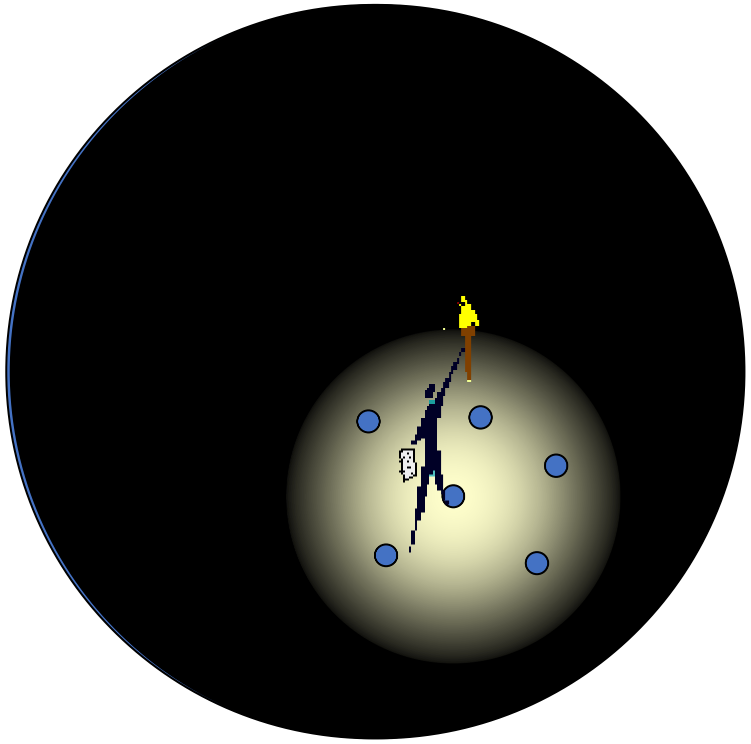
OPTICS: Extension of DBSCAN

- Order points by shortest *reachability distance* to guarantee that clusters w.r.t. higher density are finished first. (for a constant MinPts, higher

  density requires lower ε)

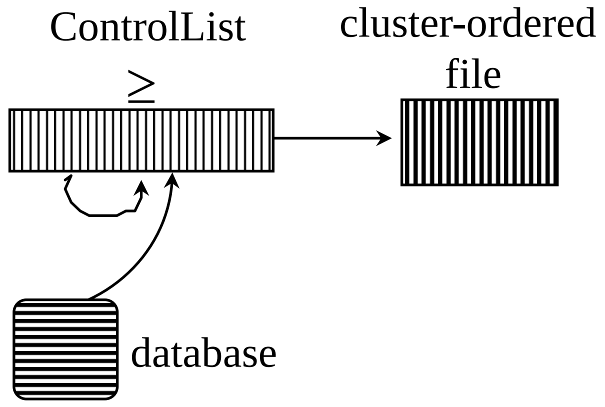
The Algorithm OPTICS

- 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



The Algorithm OPTICS

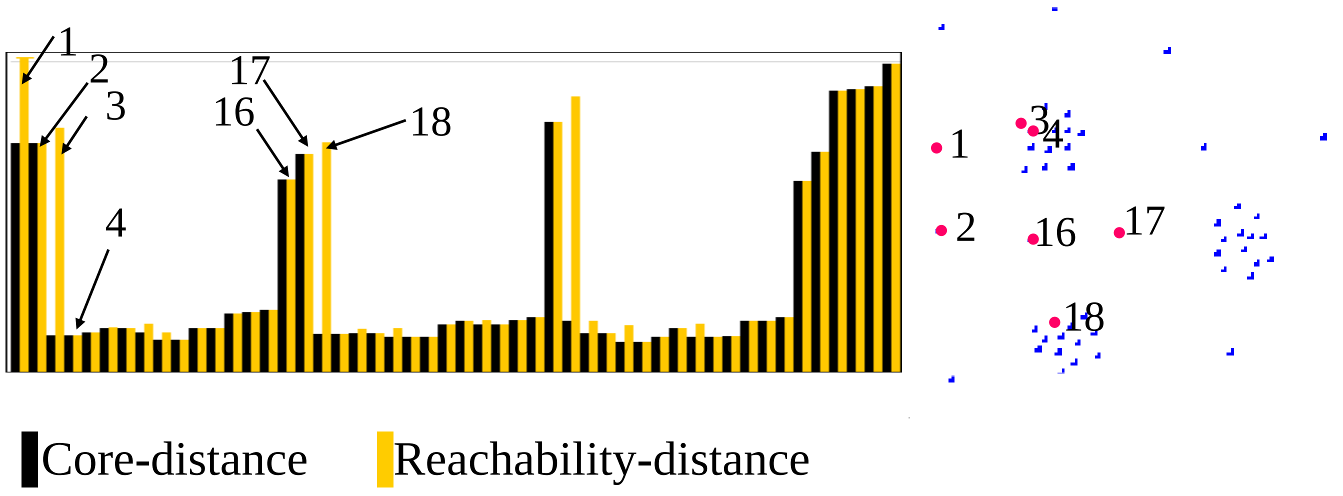
- *ControlList* ordered by reachability-distance.



```
foreach $o \in$ Database
  // initially, $o$.processed = false for all objects $o$
  if $o$.processed = false;
     insert ($o$, "?") into ControlList;
  while ControlList is not empty
        select first element ($o$, $r$-$dist$) from ControlList;
        retrieve $N_\varepsilon(o)$ 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 $\leq \varepsilon$
          foreach $p \in N_\varepsilon(o)$ not yet processed;
                determine $r\_dist_p$ = reachability-distance($p$, $o$);
                if ($p$, _) $\notin$ ControlList
                    insert ($p$, $r\_dist_p$) in ControlList;
                else if ($p$, $old\_r\_dist$) $\in$ ControlList and $r\_dist_p < old\_r\_dist$
                    update ($p$, $r\_dist_p$) in ControlList;
```

OPTICS: Properties

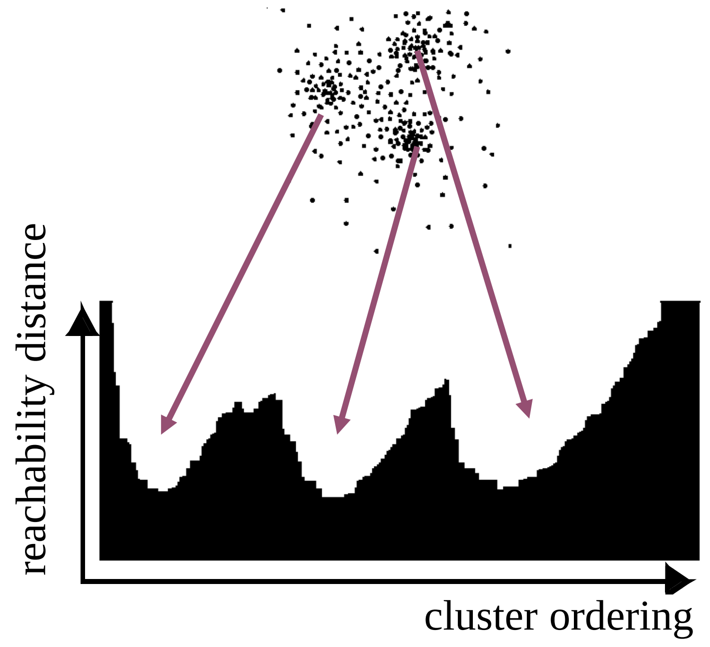
- “Flat” density-based clusters wrt. $\varepsilon^* \leq \varepsilon$ and *MinPts* afterwards:
  - Starts with an object *o* where *c-dist*(*o*) $\leq \varepsilon^*$ and *r-dist*(*o*) > ε^*
  - Continues while *r-dist* $\leq \varepsilon^*$

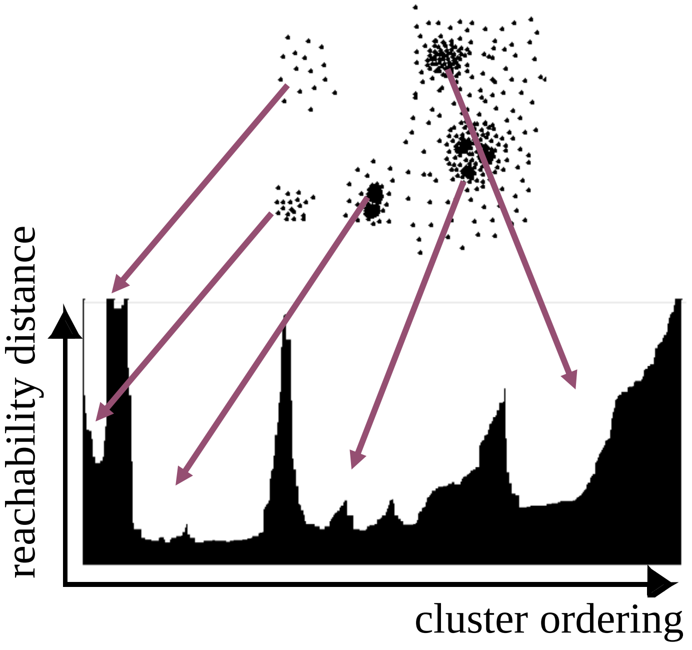


- Performance: approx. runtime(DBSCAN(ε, *MinPts*))
  - O(*n* * runtime(ε-neighborhood-query))
    - without spatial index support (worst case): O(n^2)
    - e.g. tree-based spatial index support: O($n * \log(n)$)

OPTICS: The Reachability Plot

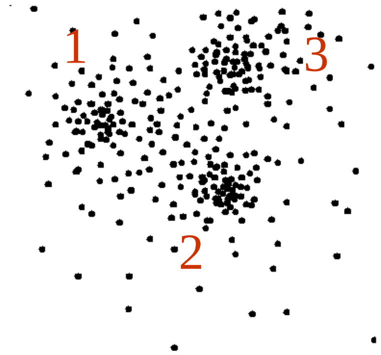
- represents the density-based clustering structure
- easy to analyze
- independent of the dimension of the data



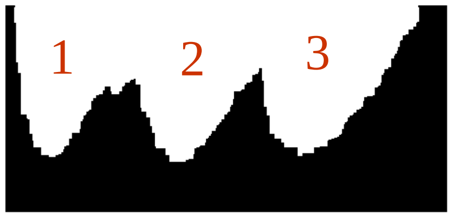


OPTICS: Parameter Sensitivity

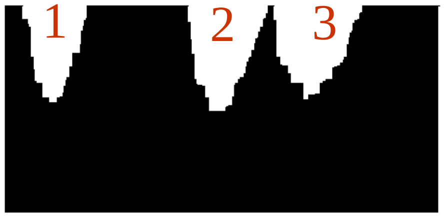
- Relatively insensitive to parameter settings
- Good result if parameters are just "large enough"



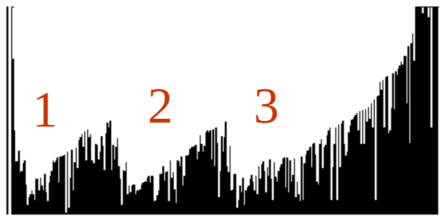
MinPts = 10, ε = 10



MinPts = 10, **ε = 5**

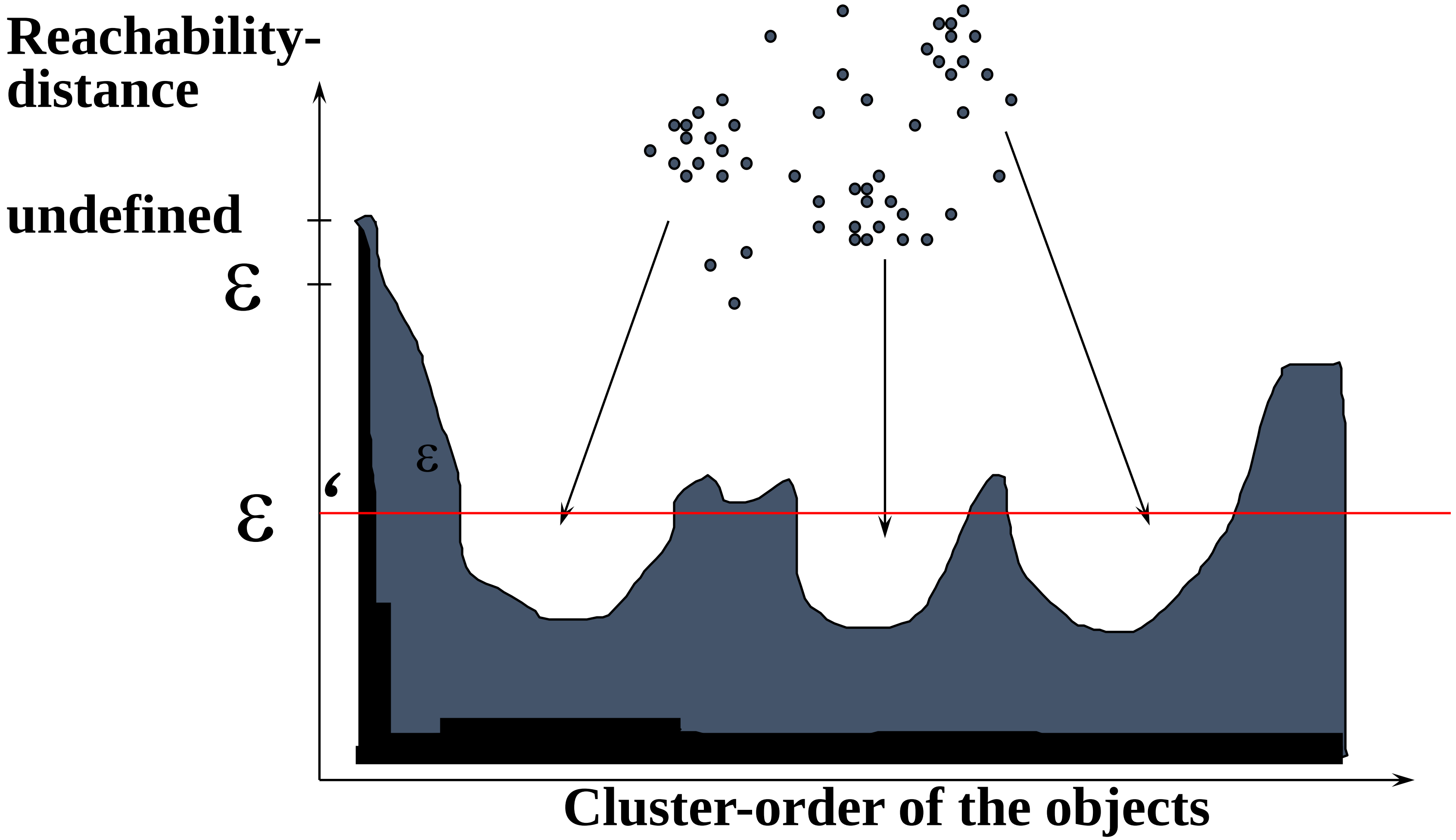


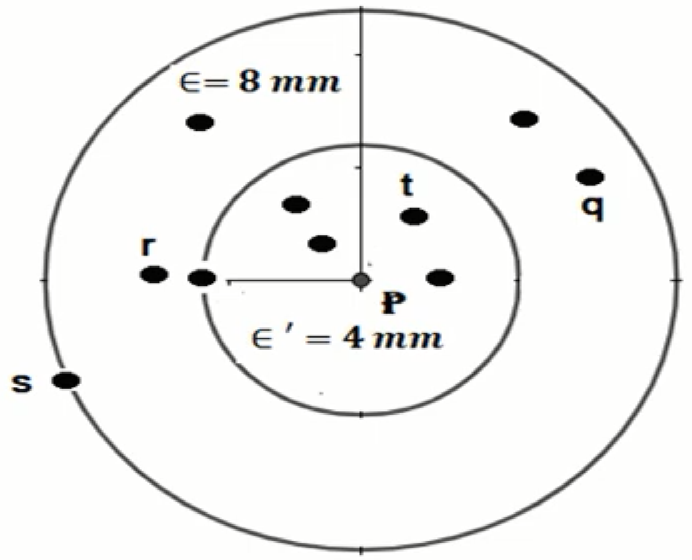
***MinPts* = 2**, ε = 10



An Example of OPTICS

neighboring objects stay close to each other in a linear sequence.





MinPts=6   Eps(ε) =8mm

Dist(p, q)=7mm

Dist(p, r)=5mm

Dist(p, s)=8mm

Dist(p, t)=3mm

P is core object. As no. of objects are more than equal to 6 within a radius of 8mm.

Core distance of p (ε') =4mm. As p have 6 objects (including p) at distance of 4mm.

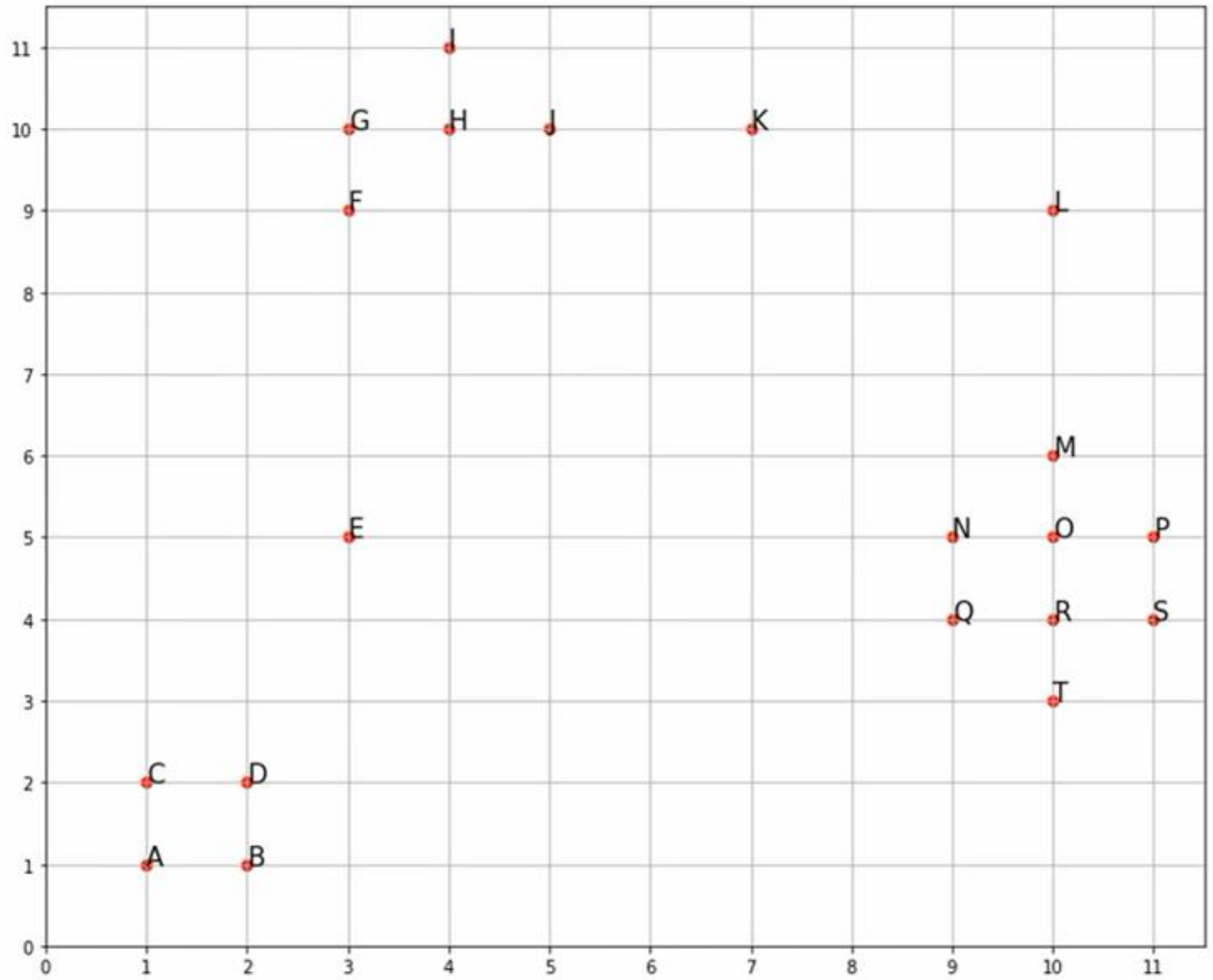
Reachability-distance of an object (q, r, s and t)

Max(core distance of p, dist(p, q))= max(4,7)=7

Max(core distance of p, dist(p, r))= max(4,5)=5

Max(core distance of p, dist(p, s))= max(4,8)=8

Max(core distance of p, dist(p, t))= max(4,3)=4



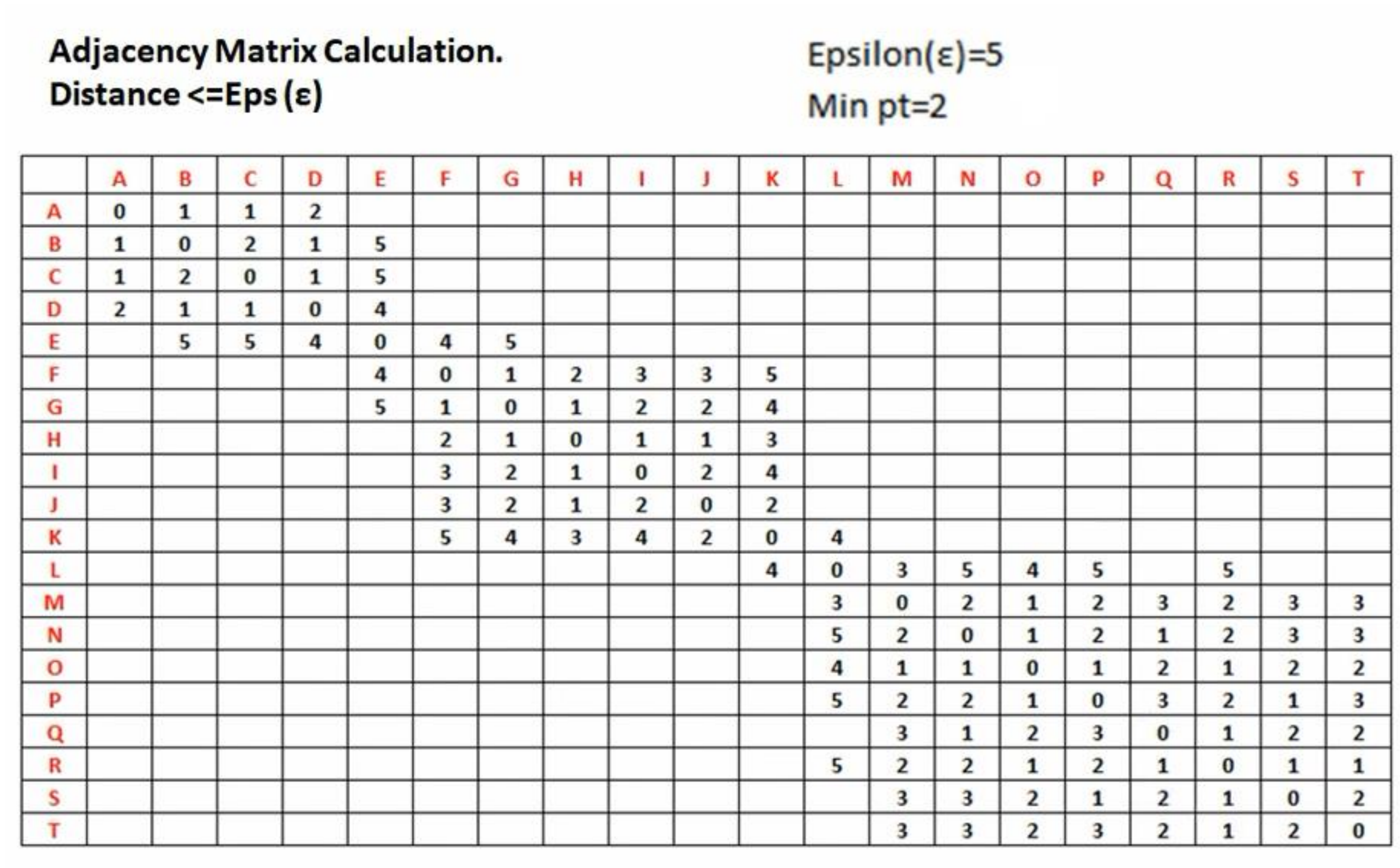
Epsilon(ε)=5

Min pt=2

Points	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
X	1	2	1	2	3	3	3	4	4	5	7	10	10	9	10	11	9	10	11	10
Y	1	1	2	2	5	9	10	10	11	10	10	9	6	5	5	5	4	4	4	3

Adjacency Matrix Calculation

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A	0	1	1	2	6	10	11	12	13	13	15	17	14	12	13	14	11	12	13	11
B	1	0	2	1	5	9	10	11	12	12	14	16	13	11	12	13	10	11	12	10
C	1	2	0	1	5	9	10	11	12	12	14	16	13	11	12	13	10	11	12	10
D	2	1	1	0	4	8	9	10	11	11	13	15	12	10	11	12	9	10	11	9
E	6	5	5	4	0	4	5	6	7	7	9	11	8	6	7	8	7	8	9	9
F	10	9	9	8	4	0	1	2	3	3	5	7	10	10	11	12	11	12	13	13
G	11	10	10	9	5	1	0	1	2	2	4	8	11	11	12	13	12	13	14	14
H	12	11	11	10	6	2	1	0	1	1	3	7	10	10	11	12	11	12	13	13
I	13	12	12	11	7	3	2	1	0	2	4	8	11	11	12	13	12	13	14	14
J	13	12	12	11	7	3	2	1	2	0	2	6	9	9	10	11	10	11	12	12
K	15	14	14	13	9	5	4	3	4	2	0	4	7	7	8	9	8	9	10	10
L	17	16	16	15	11	7	8	7	8	6	4	0	3	5	4	5	6	5	6	6
M	14	13	13	12	8	10	11	10	11	9	7	3	0	2	1	2	3	2	3	3
N	12	11	11	10	6	10	11	10	11	9	7	5	2	0	1	2	1	2	3	3
O	13	12	12	11	7	11	12	11	12	10	8	4	1	1	0	1	2	1	2	2
P	14	13	13	12	8	12	13	12	13	11	9	5	2	2	1	0	3	2	1	3
Q	11	10	10	9	7	11	12	11	12	10	8	6	3	1	2	3	0	1	2	2
R	12	11	11	10	8	12	13	12	13	11	9	5	2	2	1	2	1	0	1	1
S	13	12	12	11	9	13	14	13	14	12	10	6	3	3	2	1	2	1	0	2
T	11	10	10	9	9	13	14	13	14	12	10	6	3	3	2	3	2	1	2	0



Adjacency Matrix Calculation.
Distance <=Eps (ε)

Epsilon(ε)=5
Min pt=2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A	0	1	1	2																
B	1	0	2	1	5															
C	1	2	0	1	5															
D	2	1	1	0	4															
E		5	5	4	0	4	5													
F					4	0	1	2	3	3	5									
G					5	1	0	1	2	2	4									
H						2	1	0	1	1	3									
I						3	2	1	0	2	4									
J						3	2	1	2	0	2									
K						5	4	3	4	2	0	4								
L											4	0	3	5	4	5		5		
M												3	0	2	1	2	3	2	3	3
N												5	2	0	1	2	1	2	3	3
O												4	1	1	0	1	2	1	2	2
P												5	2	2	1	0	3	2	1	3
Q													3	1	2	3	0	1	2	2
R												5	2	2	1	2	1	0	1	1
S													3	3	2	1	2	1	0	2
T													3	3	2	3	2	1	2	0

Epsilon(ε)=5

Min pt=2

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B			1
C			1
D			2

Core distance of A (ε') =1

Reachability-distance of an object (B, C and D)

Max(core distance of A, dist(A, B))= max(1,1)=1

Max(core distance of A, dist(A, C))= max(1,1)=1

Max(core distance of A, dist(A, D))= max(1,2)=2

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C			1
D			2.1
E			5

Core distance of B (ε') =1

Reachability-distance of an object (C, D and E)

Max(core distance of B, dist(B, C))= max(1,2)=2. Retain previous value

Max(core distance of B, dist(B, D))= max(1,1)=1 Modify existing value

Max(core distance of B, dist(B, E))= max(1,5)=5

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C	(C,0), (A,D-1),(B,2),(E,5)	1	1
D			2,1
E			5

Core distance of C (ε') =1

Reachability-distance of an object (D, E)

Max(core distance of C, dist(C, D))= max(1,1)=1 Retain existing value

Max(core distance of C, dist(C, E))= max(1,5)=5 Retain existing value

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C	(C,0), (A,D-1),(B,2),(E,5)	1	1
D	(D,0), (B,C-1),(A,2),(E,4)	1	2,1
E			5,4

Core distance of D (ε') =1

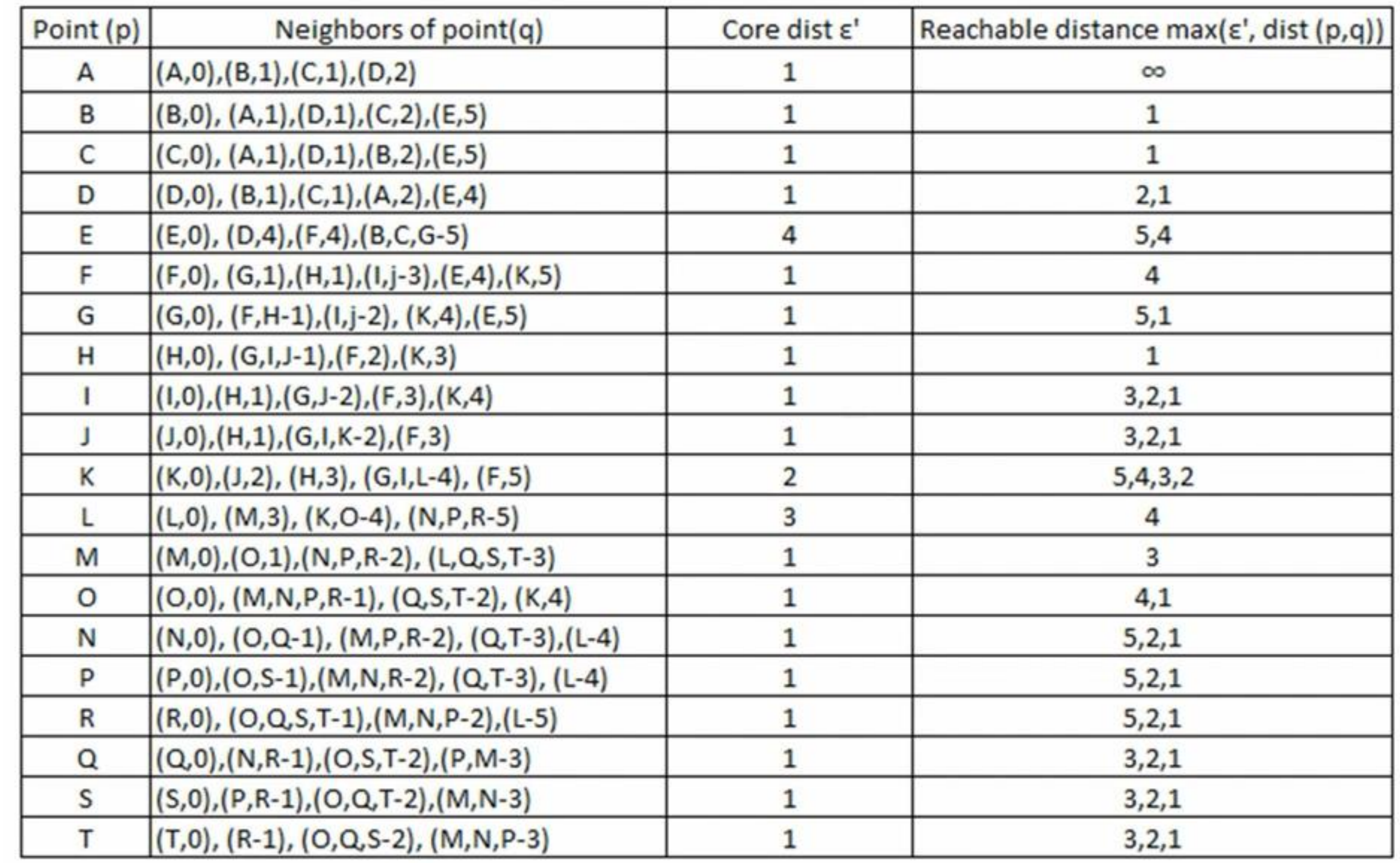
Reachability-distance of an object (E)

Max(core distance of D, dist(D, E))= max(1,4)=4 Modify existing value

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C	(C,0), (A,D-1),(B,2),(E,5)	1	1
D	(D,0), (B,C-1),(A,2),(E,4)	1	2,1
E			5,4

Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C	(C,0), (A,D-1),(B,2),(E,5)	1	1
D	(D,0), (B,C-1),(A,2),(E,4)	1	2,1
E	(E,0), (D,F-4) ,(B,C,G-5)	4	5,4
F			4
G			5

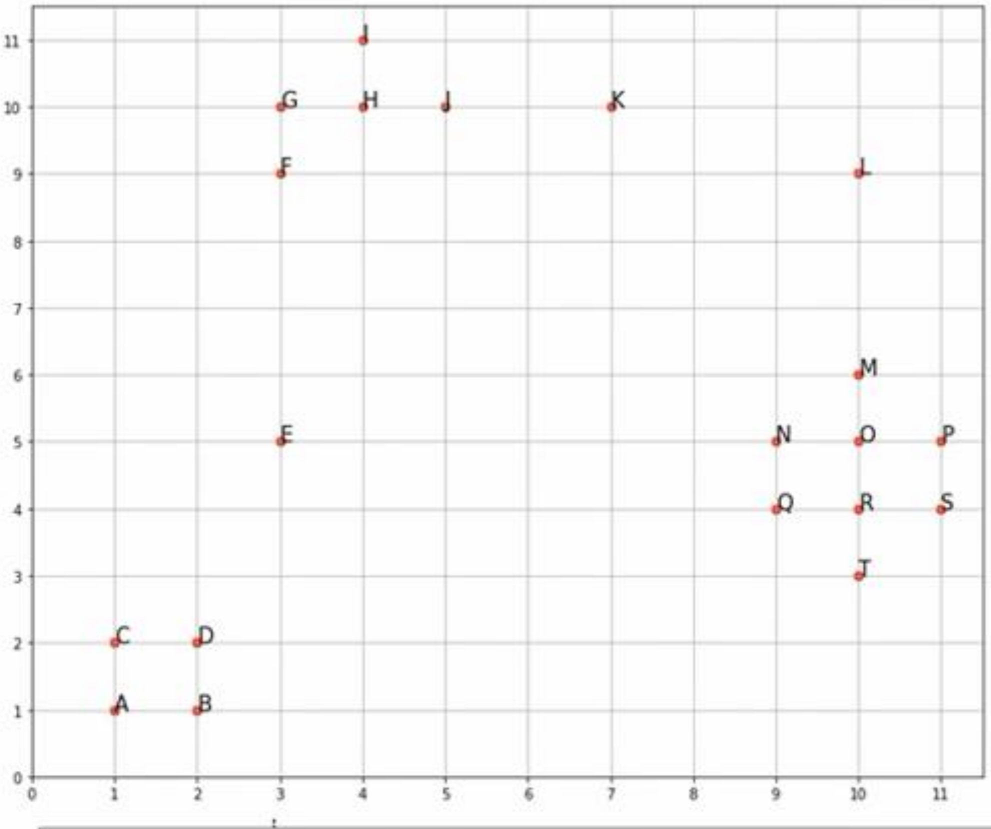
Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,C-1),(D,2)	1	∞
B	(B,0), (A,D-1),(C,2),(E,5)	1	1
C	(C,0), (A,D-1),(B,2),(E,5)	1	1
D	(D,0), (B,C-1),(A,2),(E,4)	1	2,1
E	(E,0), (D,F-4) ,(B,C,G-5)	4	5,4
F	(F,0), (G,1),(H,2),(I,J-3),(E,4),(K,5)	1	4
G			5,1
H			2
I			3
J			3
K			5

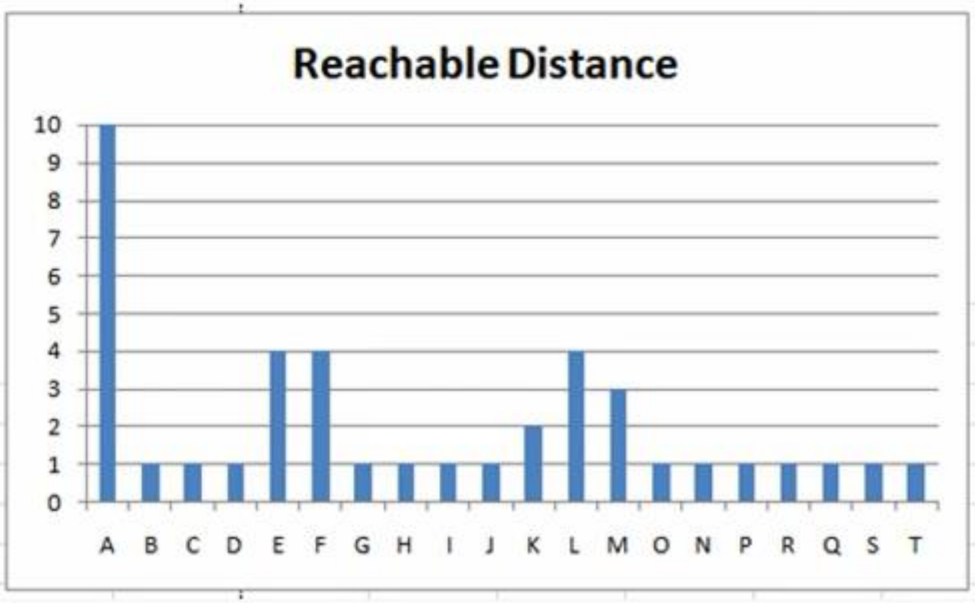


Point (p)	Neighbors of point(q)	Core dist ε'	Reachable distance max(ε', dist (p,q))
A	(A,0),(B,1),(C,1),(D,2)	1	∞
B	(B,0), (A,1),(D,1),(C,2),(E,5)	1	1
C	(C,0), (A,1),(D,1),(B,2),(E,5)	1	1
D	(D,0), (B,1),(C,1),(A,2),(E,4)	1	2,1
E	(E,0), (D,4),(F,4),(B,C,G-5)	4	5,4
F	(F,0), (G,1),(H,1),(I,j-3),(E,4),(K,5)	1	4
G	(G,0), (F,H-1),(I,j-2), (K,4),(E,5)	1	5,1
H	(H,0), (G,I,J-1),(F,2),(K,3)	1	1
I	(I,0),(H,1),(G,J-2),(F,3),(K,4)	1	3,2,1
J	(J,0),(H,1),(G,I,K-2),(F,3)	1	3,2,1
K	(K,0),(J,2), (H,3), (G,I,L-4), (F,5)	2	5,4,3,2
L	(L,0), (M,3), (K,O-4), (N,P,R-5)	3	4
M	(M,0),(O,1),(N,P,R-2), (L,Q,S,T-3)	1	3
O	(O,0), (M,N,P,R-1), (Q,S,T-2), (K,4)	1	4,1
N	(N,0), (O,Q-1), (M,P,R-2), (Q,T-3),(L-4)	1	5,2,1
P	(P,0),(O,S-1),(M,N,R-2), (Q,T-3), (L-4)	1	5,2,1
R	(R,0), (O,Q,S,T-1),(M,N,P-2),(L-5)	1	5,2,1
Q	(Q,0),(N,R-1),(O,S,T-2),(P,M-3)	1	3,2,1
S	(S,0),(P,R-1),(O,Q,T-2),(M,N-3)	1	3,2,1
T	(T,0), (R-1), (O,Q,S-2), (M,N,P-3)	1	3,2,1

Epsilon(ε)=5

Min pt=2

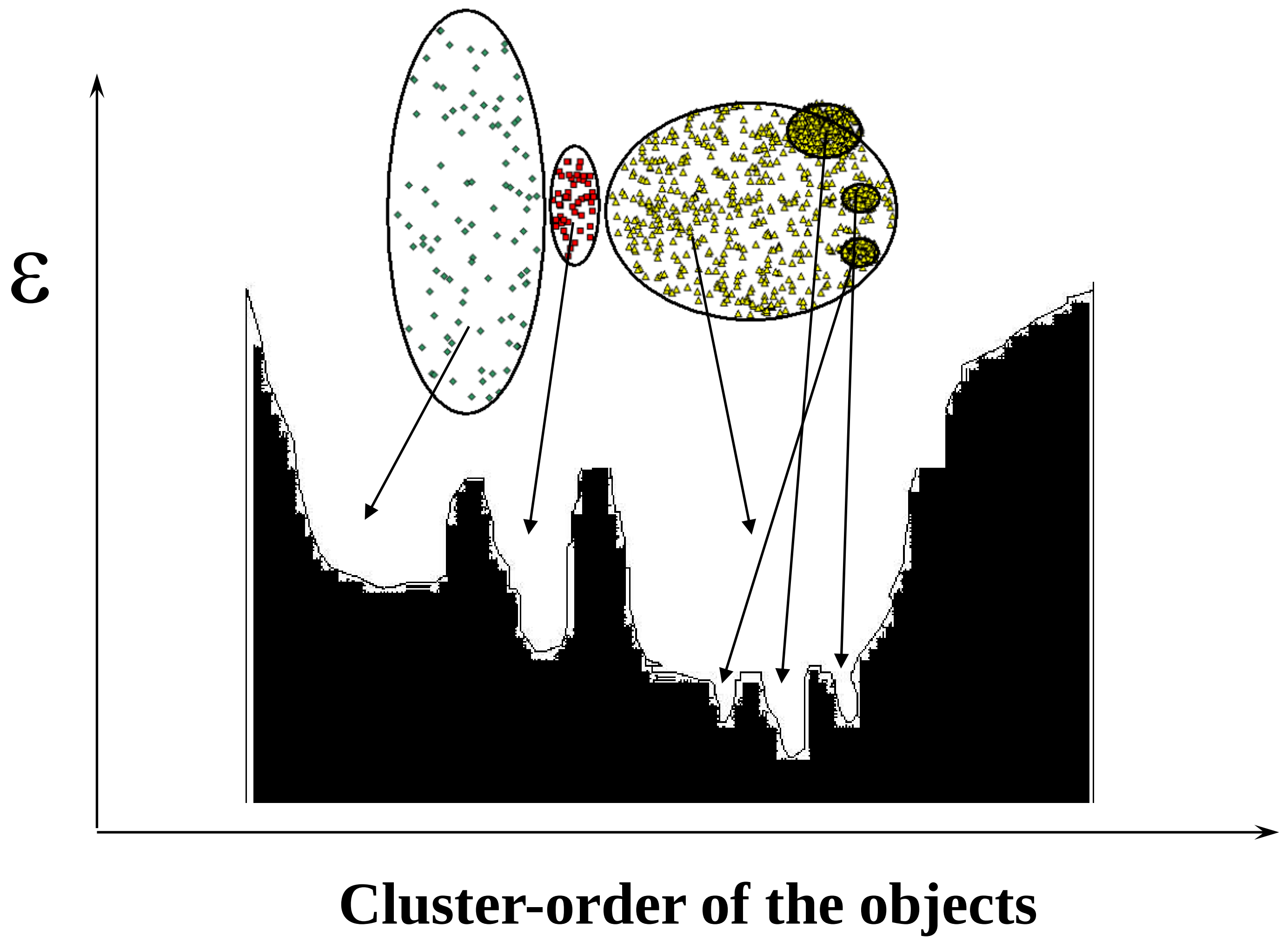




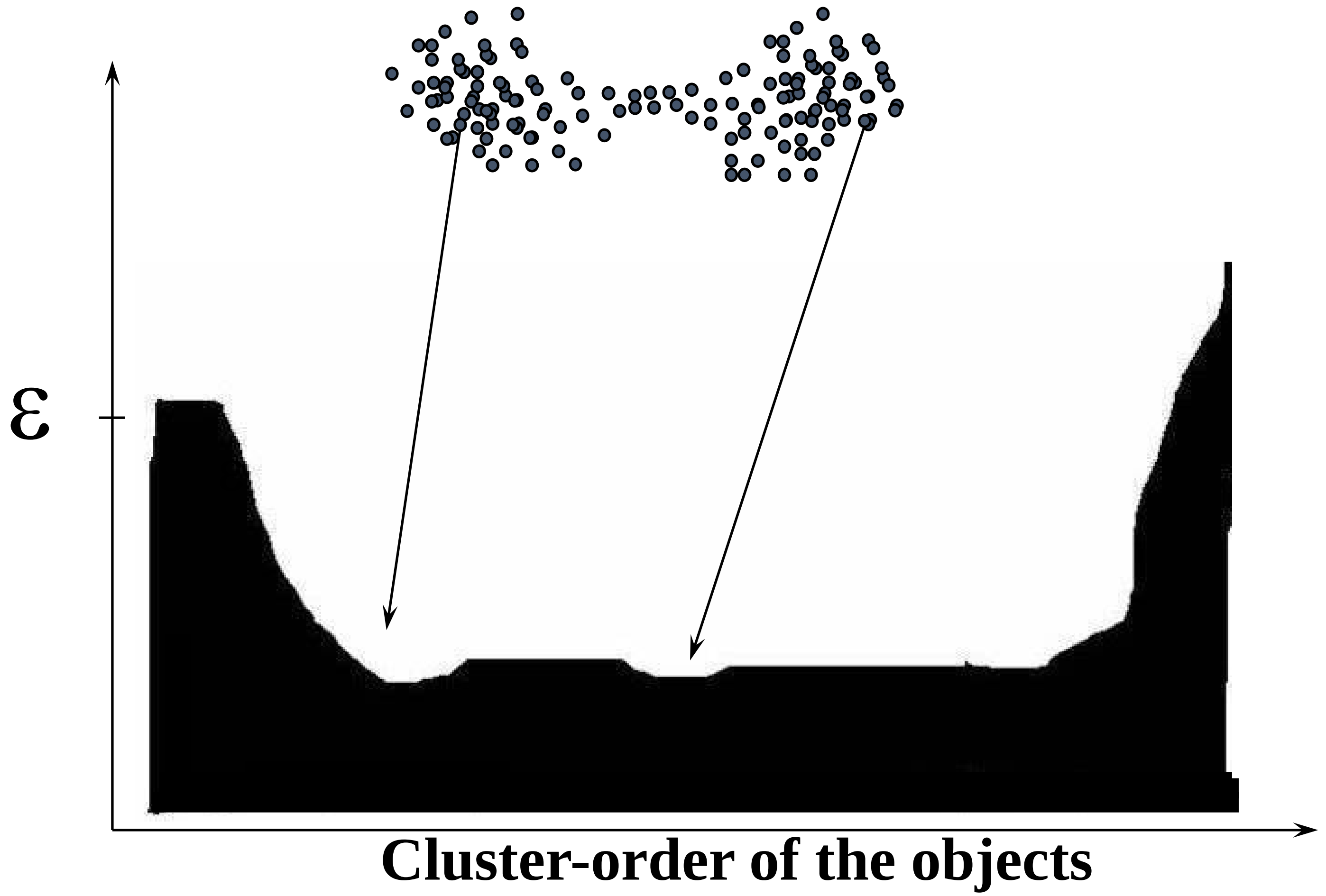
DBSCAN VS OPTICS

	DBSCAN	OPTICS
Density	Boolean value (high/low)	Numerical value (core distance)
Density-connected	Boolean value (yes/no)	Numerical value (reachability distance)
Searching strategy	random	greedy

When OPTICS Works Well



When OPTICS Does NOT Work Well



https://www.youtube.com/watch?v=8kJjgowewOs

DENCLUE: using density functions

- DENsity-based CLUstEring by Hinneburg & Keim (KDD’98)
- Major features
  - Solid mathematical foundation
  - Good for data sets with large amounts of noise
  - Allows a compact mathematical description of arbitrarily shaped clusters in high-dimensional data sets
  - Significantly faster than existing algorithm (faster than DBSCAN by a factor of up to 45)
  - But needs a large number of parameters

Denclue: Technical Essence

- Model density by the notion of influence
- Each data object exert influence on its neighborhood.
- The influence decreases with distance
- Example:
  - Consider each object is a radio, the closer you are to the object, the louder the noise
- Key: Influence is represented by mathematical function

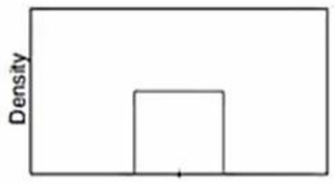
Denclue: Technical Essence

- Influence functions: (influence of y on x, σ is a user given constant)
  - Square : $f^{y}_{square}(x) = 0$, if dist(x,y) > σ,

    1, otherwise
  - Guassian: $$f^{y}_{Gaussian}(x) = e^{-\frac{d(x,y)^2}{2\sigma^2}}$$

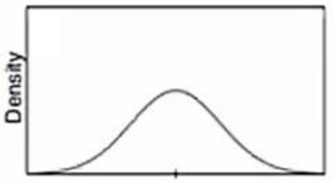
- Kernel density function: mathematical description of the influence a data object has within its neighborhood (not to be confused with "Kernel Trick")
- Kernel density function
  - K(x) >=0
  - K(-x)=K(x)

$$\int_{\mathbb{R}^d} K(x)dx = 1$$



(a) Square wave function

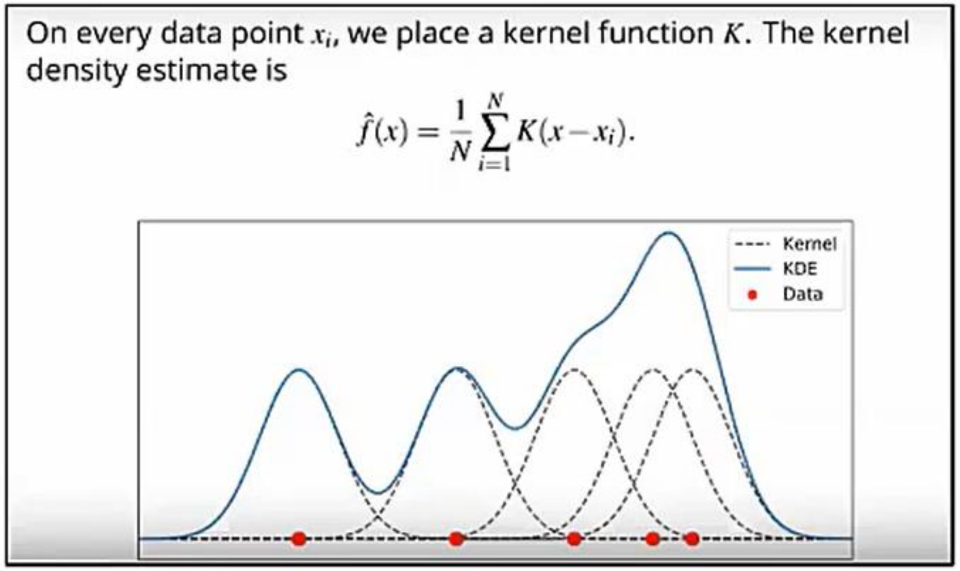
- Example kernel functions
  - Square : $f^{y}_{square}(x)$ = 0, if dist(x,y) > σ,
    1, otherwise



(b) Gaussian function

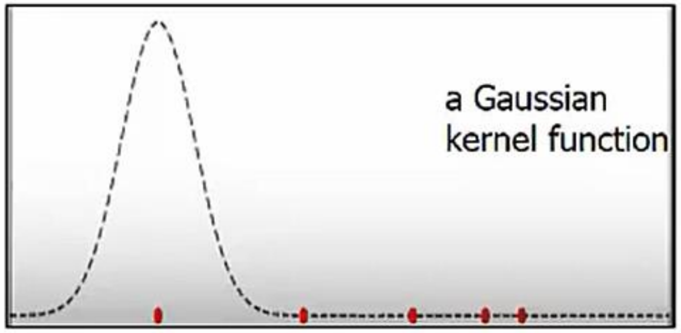
  - Gaussian kernel function: $K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-xi)^2}{2h^2}}$

- DENsity-based CLUstEring
- Inspired by the **kernel density estimation** method.
  - Estimate the unknown probability density function of a random variable, based on a sample of points, using a kernel function.



On every data point x_i, we place a kernel function K. The kernel density estimate is

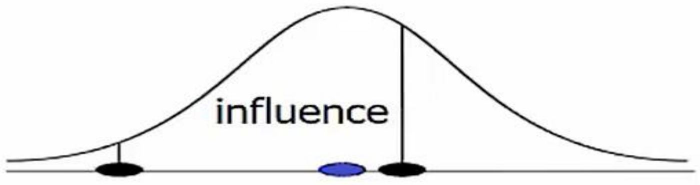
$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} K(x - x_i).$$

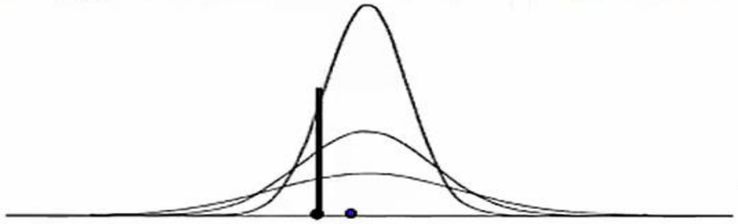


The kernel density estimate (blue curve) is the sum of all N kernels/N, so the final result is also a Density function (with integral or area under the curve ==1).

Density Attractor

- Intuition
  - **Influence** of a point on its neighbors can be modeled with a density function: stronger influence on the points that are closer.

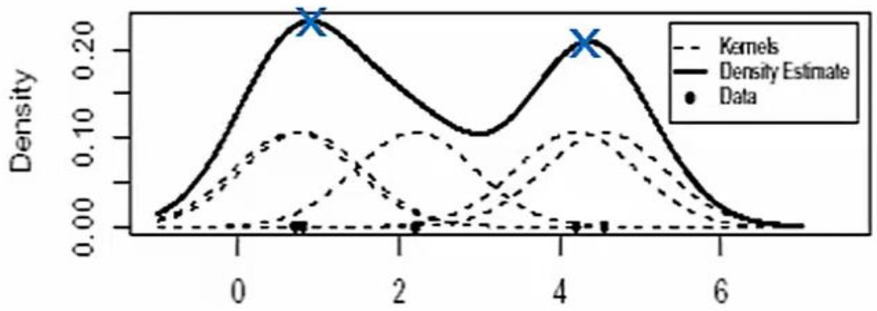




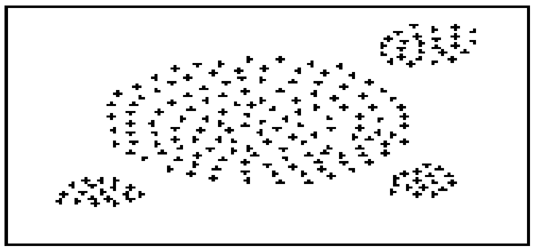
Smoothing parameter h

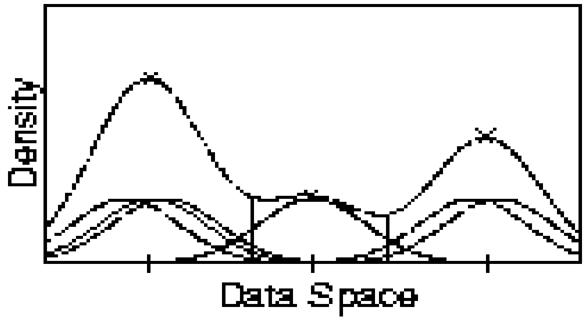
$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-xi)^2}{2h^2}}$$

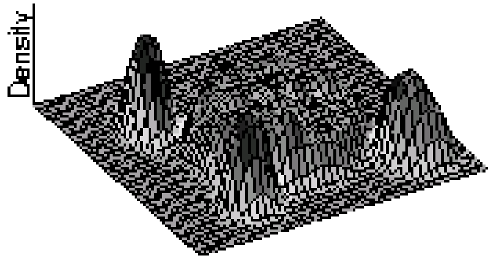
  - **Density attractors** can be identified in the kernel density estimate.



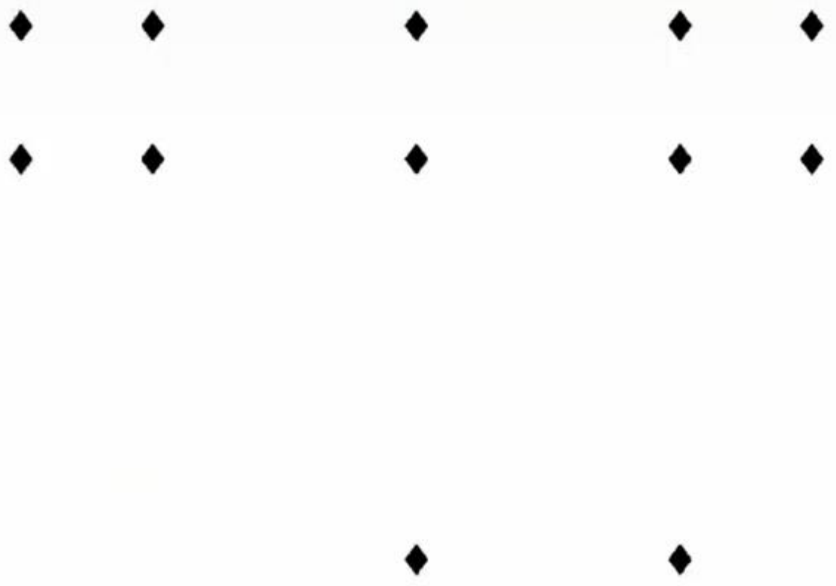
$$\hat{f}(x) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

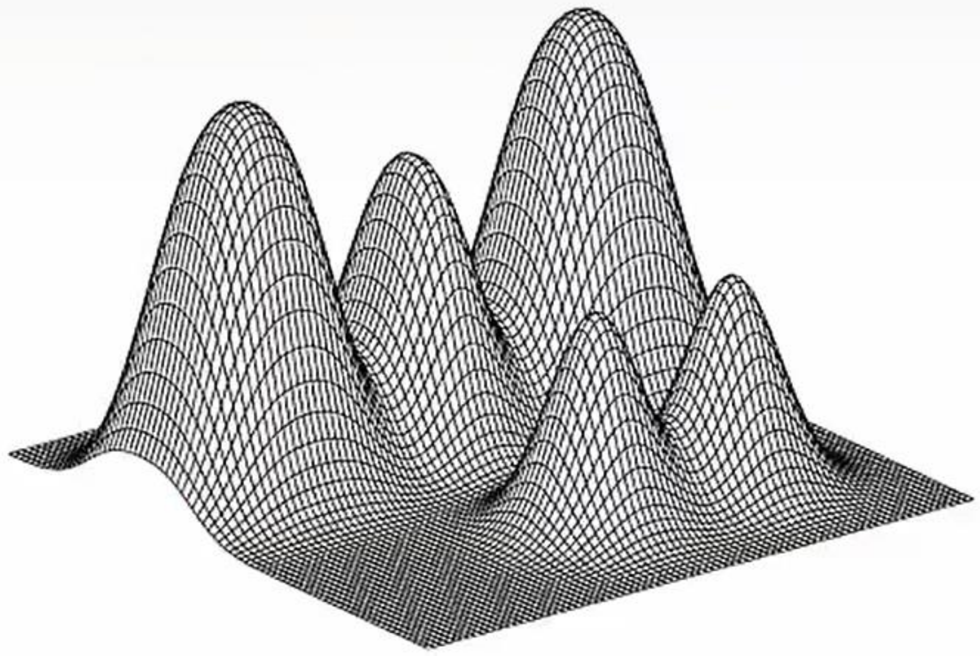

(a) Data Set

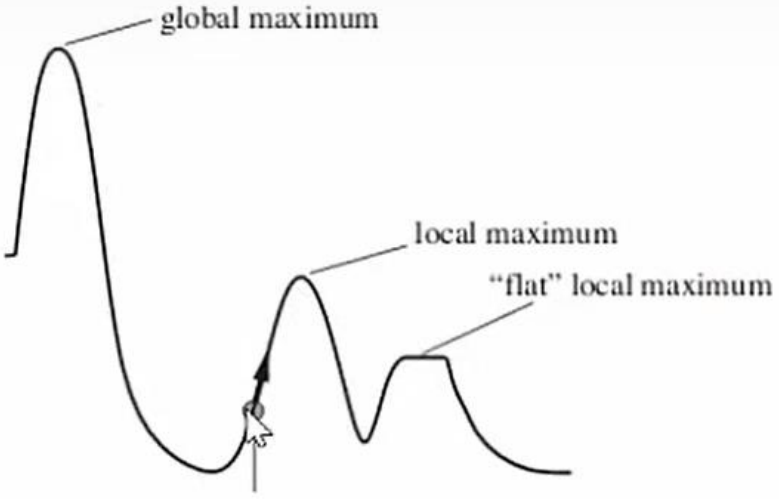




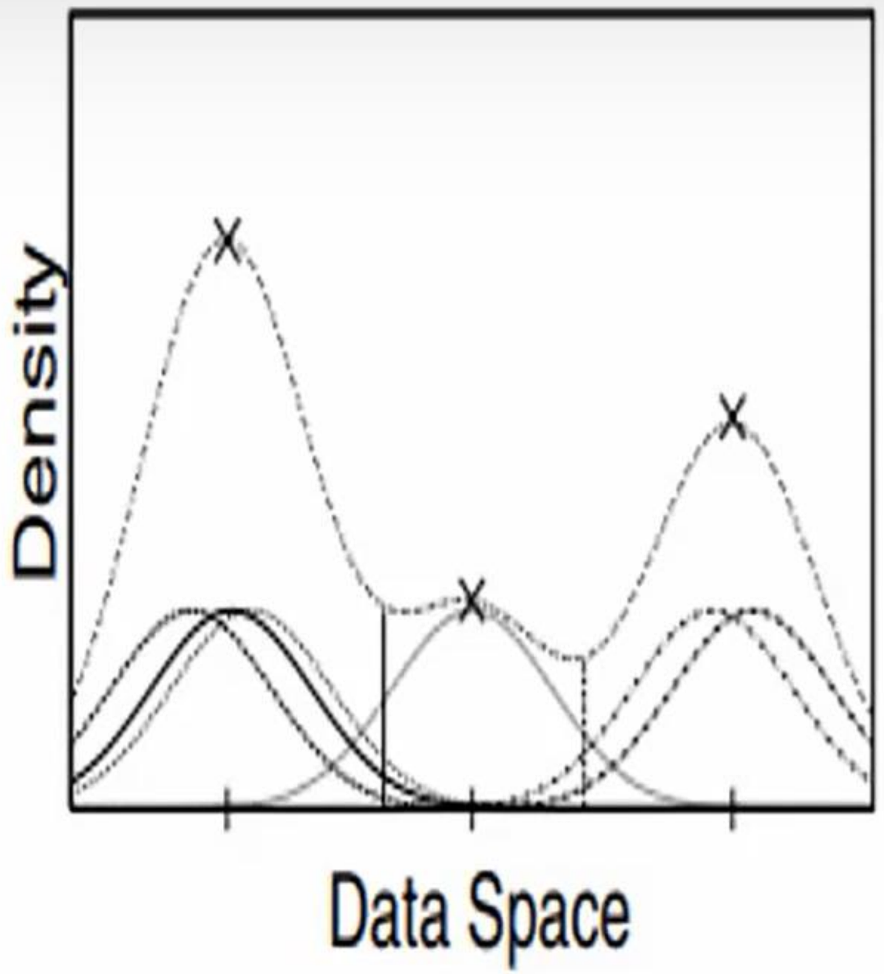
(c) Gaussian


Set of 12 points.


Overall density—surface plot.



Hill climbing



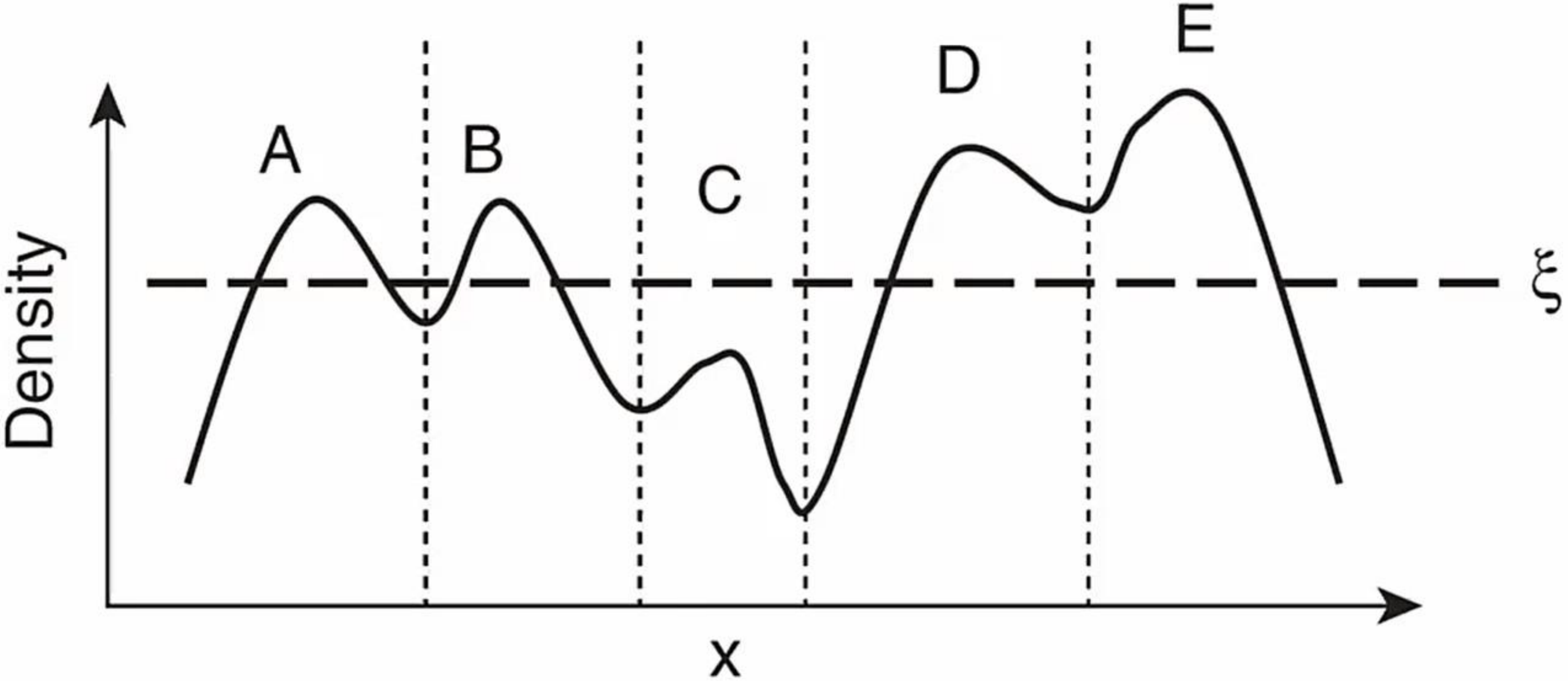
Gradient: The steepness of a slope

- Example

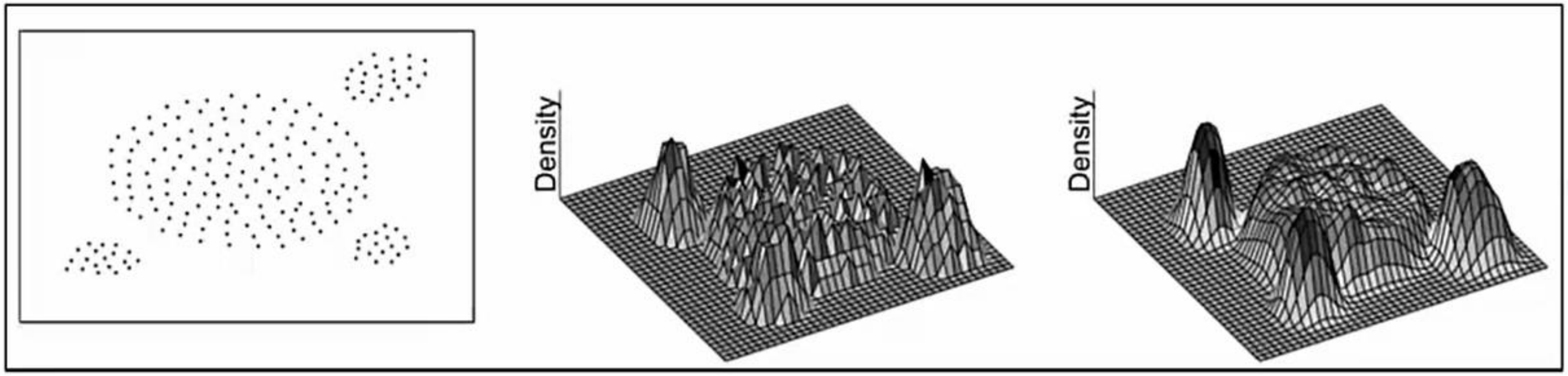
$$f_{Gaussian}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}$$

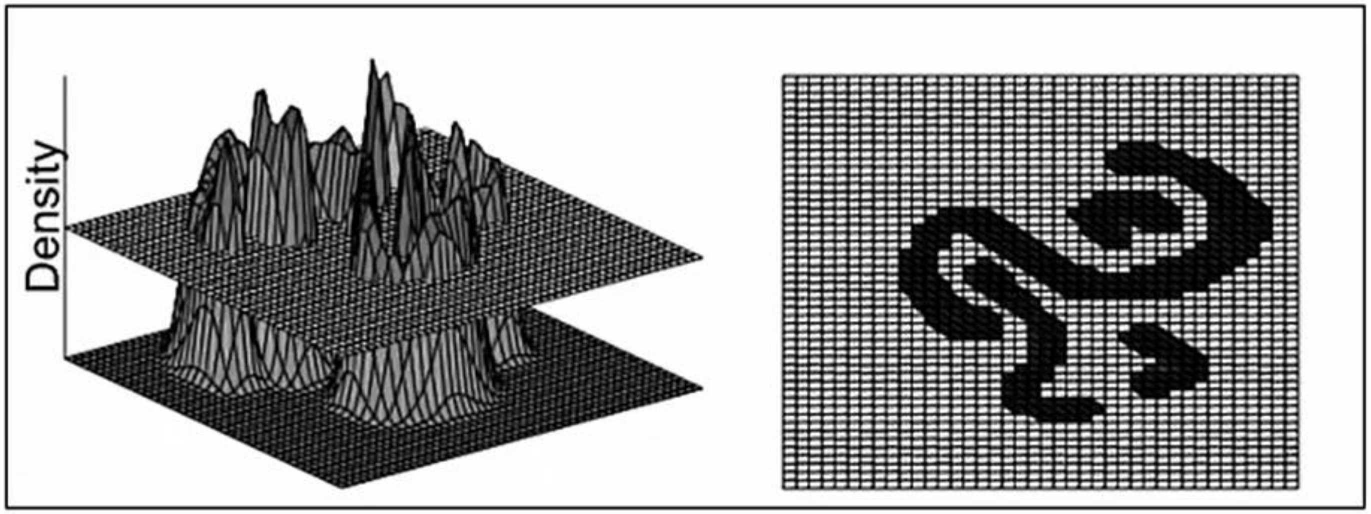
$$f_{Gaussian}^{D}(x) = \sum_{i=1}^{N} e^{-\frac{d(x,x_i)^2}{2\sigma^2}}$$

$$\nabla f_{Gaussian}^{D}(x, x_i) = \sum_{i=1}^{N} (x_i - x) \cdot e^{-\frac{d(x,x_i)^2}{2\sigma^2}}$$



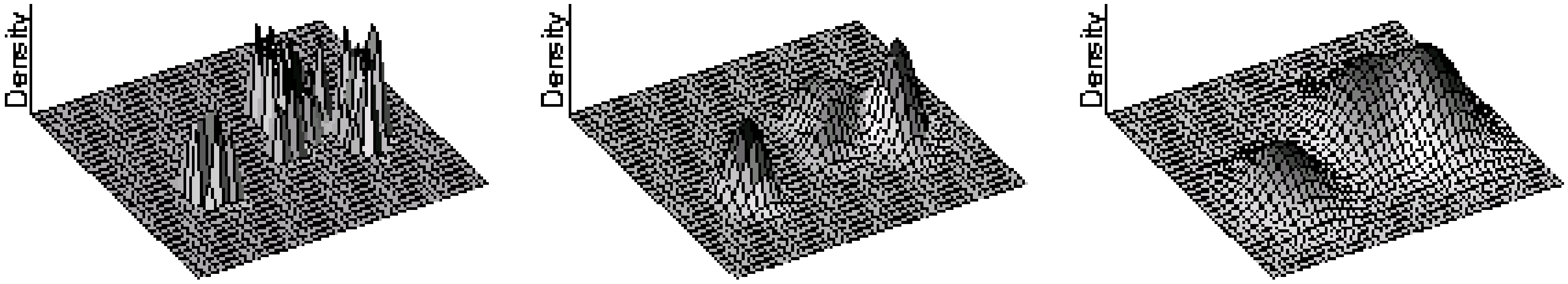
- ξ : *noise threshold*
- Density attractors with density < ξ are considered noise.





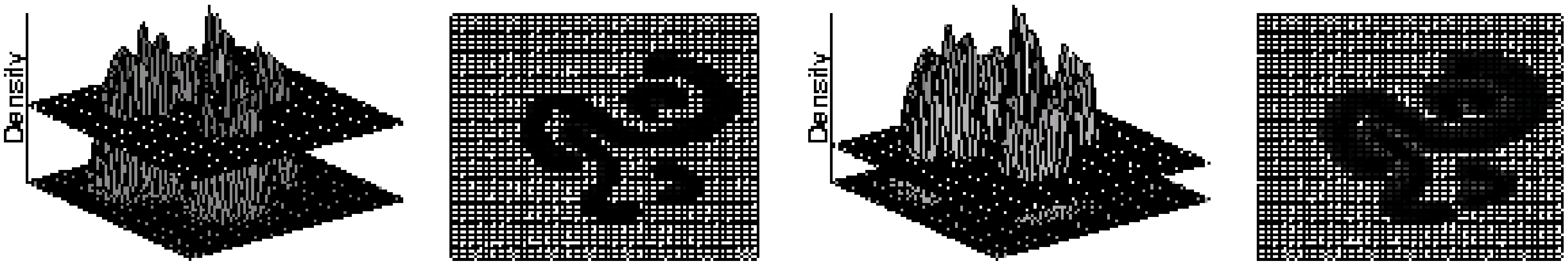
- Points close to a density attractor forms a cluster.
- Clusters can be of different shapes, sizes, and densities.
- Can be partitional or hierarchical

Center-Defined and Arbitrary



(a) $\sigma = 0.2$ (b) $\sigma = 0.6$ (d) $\sigma = 1.5$

Figure 3: Example of Center-Defined Clusters for different σ



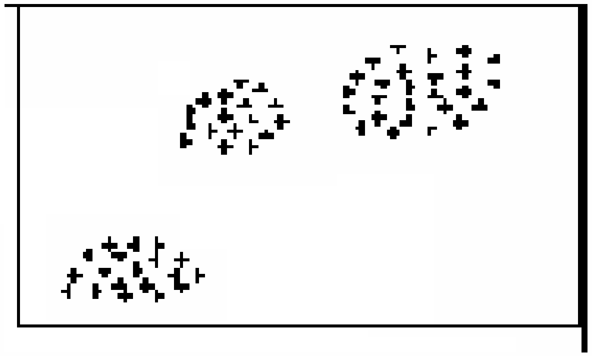
(a) $\xi = 2$ (b) $\xi = 2$ (c) $\xi = 1$ (d) $\xi = 1$

Figure 4: Example of Arbitray-Shape Clusters for different ξ

Density Function – Effect of Smoothing Factor (h, σ)

- Density Definition is defined as the sum of the influence functions of all data points.

$$f^D_{Gaussian}(x) = \sum_{i=1}^{N} e^{-\frac{d(x,x_i)^2}{2\sigma^2}}$$



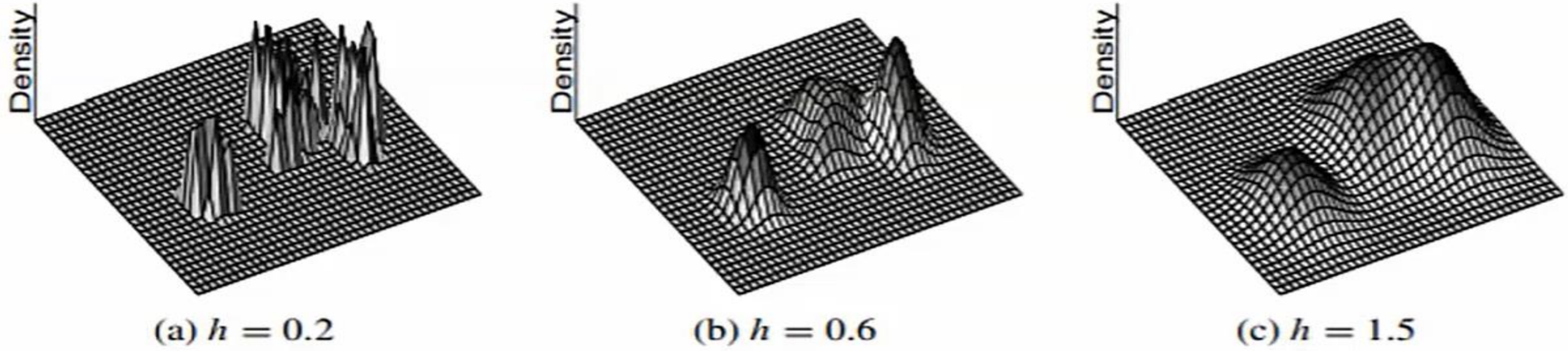


Fig. 4. Example of center-defined clusters for different h.

Denclue: Technical Essence

- Clusters can be determined mathematically by identifying density attractors.
- Density attractors are local maximum of the overall density function.

Cluster Definition

- Center-defined cluster
  - A subset of objects attracted by an attractor x
  - density(x) ≥ ξ
- Arbitrary-shape cluster
  - A group of center-defined clusters which are connected by a path P
  - For each object x on P, density(x) ≥ ξ.

DENCLUE: How to find the clusters

- Divide the space into grids, with size 2σ
- Consider only grids that are highly populated
- For each object, calculate its density attractor using hill climbing technique
  - Tricks can be applied to avoid calculating density attractor of all points
- Density attractors form basis of all clusters

Features of DENCLUE

- Major features
  - Solid mathematical foundation
    - Compact definition for density and cluster
    - Flexible for both center-defined clusters and arbitrary-shape clusters
  - But needs a large number of parameters
    - σ: parameter to calculate density
    - ξ: density threshold
    - δ: parameter to calculate attractor

Other algorithms

- PAM, CLARANS: Solutions for the k-medoids problem
- BIRCH: Constructs a hierarchical tree that acts a summary of the data, and then clusters the leaves.
- MST: Clustering using the Minimum Spanning Tree.
- ROCK: clustering categorical data by neighbor and link analysis
- LIMBO, COOLCAT: Clustering categorical data using information theoretic tools.
- CURE: Hierarchical algorithm uses different representation of the cluster
- CHAMELEON: Hierarchical algorithm uses closeness and interconnectivity for merging