Clustering Algorithm (DBSCAN)

VISHAL BHARTI
Computer Science Dept.
GC, CUNY

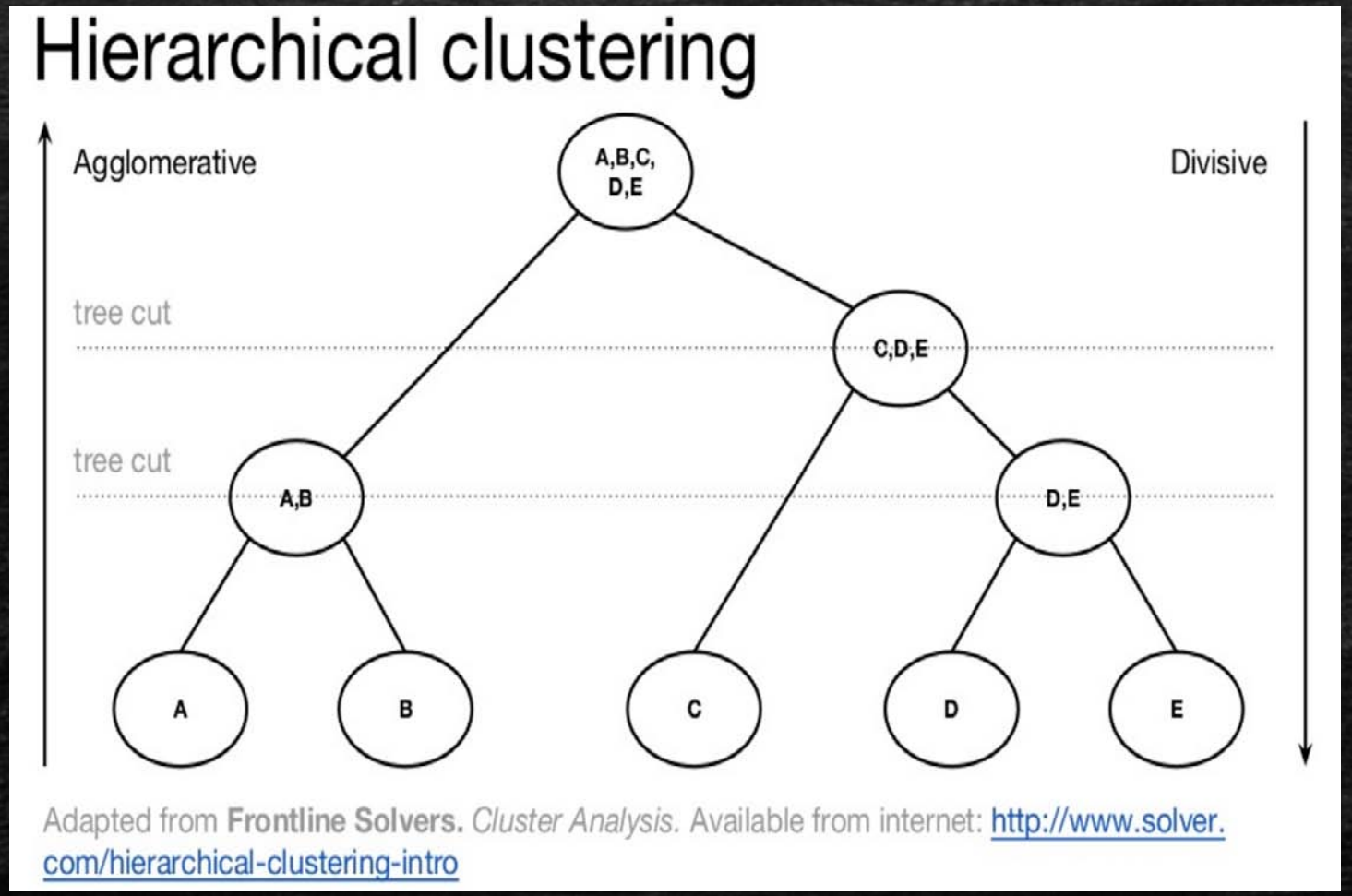
Clustering Algorithm

- Clustering is an unsupervised machine learning algorithm that divides a data into meaningful sub-groups, called clusters.
- The subgroups are chosen such that the intra-cluster differences are minimized and the inter-cluster differences are maximized.
- The very definition of a ‘cluster’ depends on the application. There are a myriad of clustering algorithms.
- These algorithms can be generally classified into four categories: partitioning based, hierarchy based, density based and grid based.

Hierarchical clustering algorithms

- Hierarchical clustering algorithms seek to build a hierarchy of cluster. They start with some initial clusters and gradually converge to the solution.
- The Hierarchical clustering algorithms can take two approaches :
  - Agglomerative (top-down) approach : Each point has its own cluster and clusters are gradually built by combining points.
  - Divisive (bottom-up) approach : All points belong to one cluster and this cluster is gradually broken into smaller clusters.

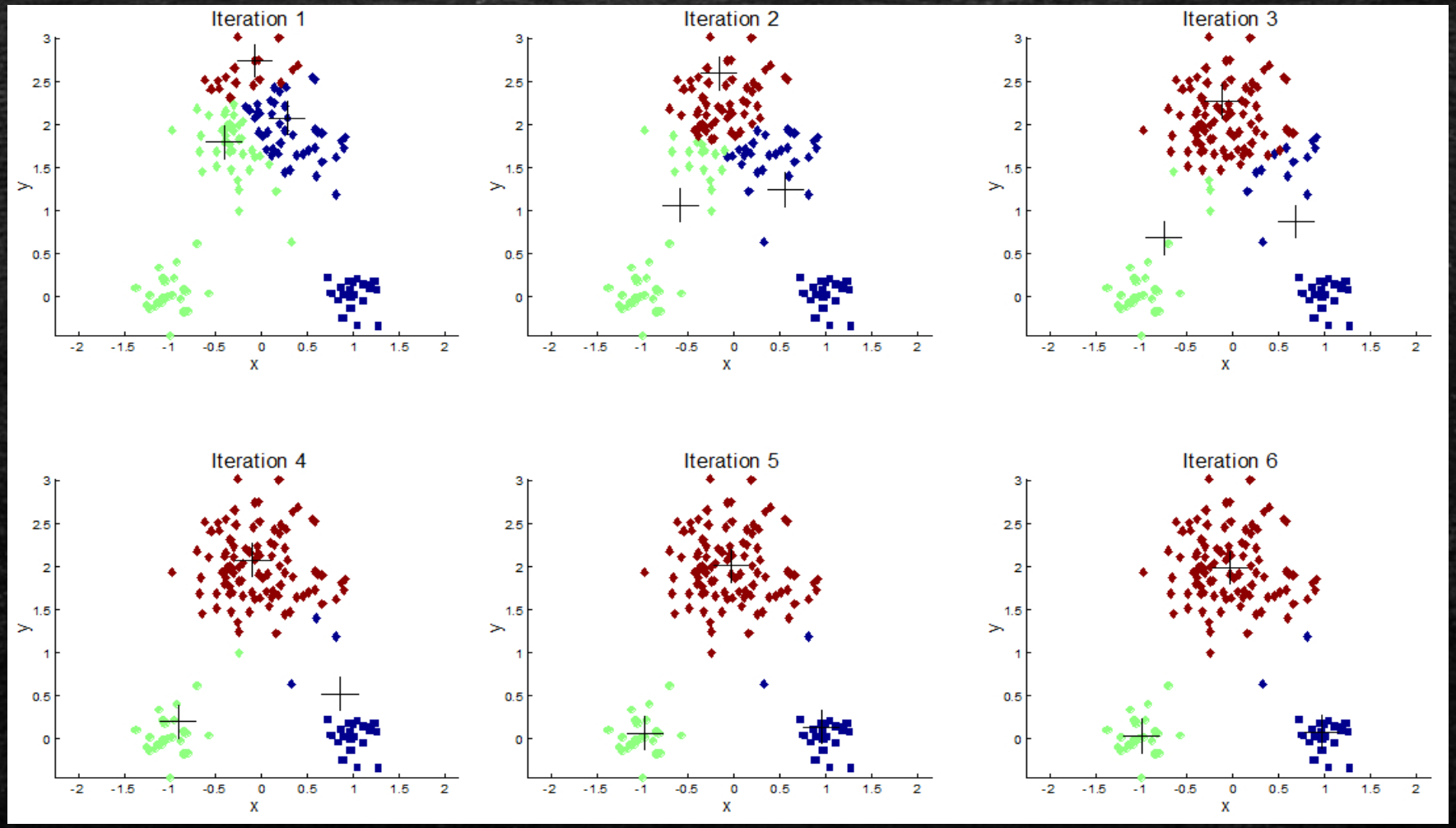
Hierarchical clustering algorithms



Partitioning based clustering algorithms

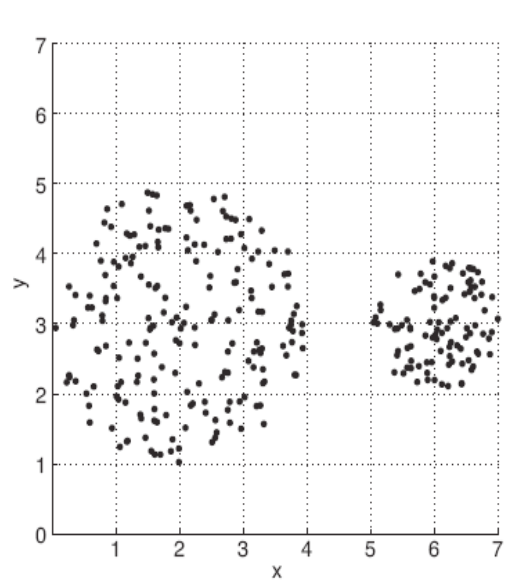
- Partitioning based clustering algorithms divide the dataset into initial 'K' clusters and iteratively improve the clustering quality based on a objective function.
- K-means is an example of a partitioning based clustering algorithm.
- The objective function in K-means is the SSE.
- Partitioning based algorithm are sensitive to initialization.

Partitioning based clustering algorithms



Grid based clustering algorithms

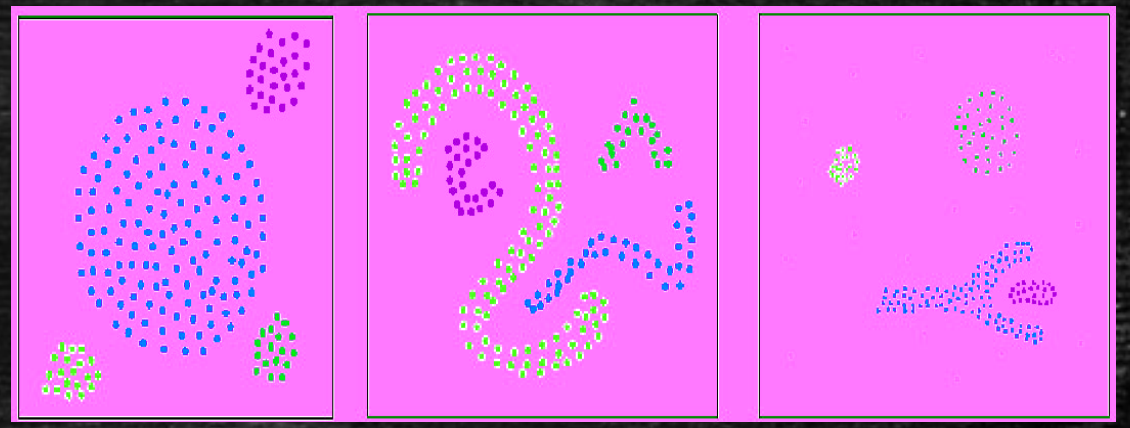
- In grid based clustering algorithm, the entire dataset is overlaid by a regular hypergrid.
- The clusters are then formed by combining dense cells.
- Some consider it as a variant of density based clustering algorithms.
- CLIQUE is a grid based clustering algorithm.



0	0	0	0	0	0	0
0	0	0	0	0	0	0
4	17	18	6	0	0	0
14	14	13	13	0	18	27
11	18	10	21	0	24	31
3	20	14	4	0	0	0
0	0	0	0	0	0	0

Density based clustering algorithms

- Density based clustering algorithms make an assumption that clusters are dense regions in space separated by regions of lower density.
- A dense cluster is a region which is "density connected", i.e. the density of points in that region is greater than a minimum.
- Since these algorithms expand clusters based on dense connectivity, they can find clusters of arbitrary shapes.
- DBSCAN is an example of density based clustering algorithm.



DBSCAN (Density Based Spatial Clustering of Applications with Noise)

- Published by Ester et. al. in 1996
- The algorithm finds dense areas and expands these recursively to find dense arbitrarily shaped clusters.
- Two main parameters to DBSCAN are 'ε' and 'minPoints'. 'ε' defines radius of the 'neighborhood region' and 'minPoints' defines the minimum number of points that should be contained within that neighborhood.
- Since it has a concept of noise, it works well even with noisy datasets.

DBSCAN Algorithm

- Epsilon neighborhood *(N_ε)* : set of all points within a distance 'ε'.
- Core point : A point that has at least 'minPoint' (including itself) points within it's N_ε .
- Direct Density Reachable (DDR) : A point *q* is directly density reachable from a point *p* if *p* is core point and $q \in N_\varepsilon$.
- Density Reachable (DR) : Two points are DR if there is a chain of DDR points that link these two points.
- Border Point: Point that are DDR but not a core point.
- Noise : Points that do not belong to any point's N_ε .

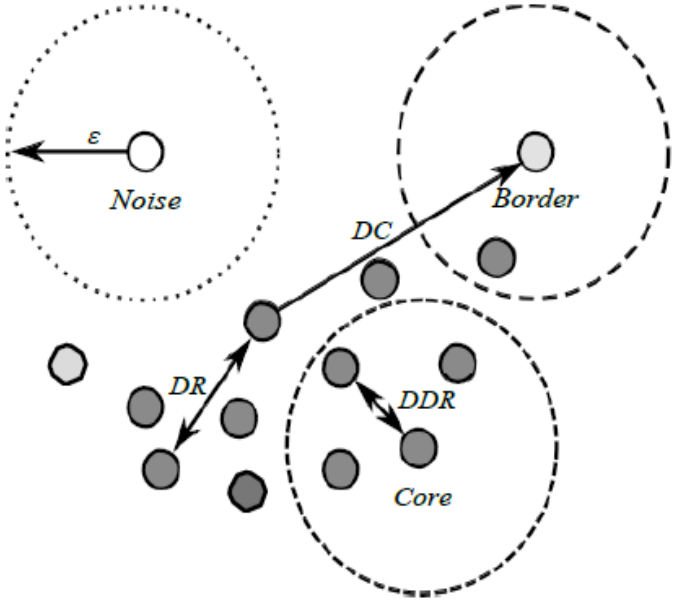
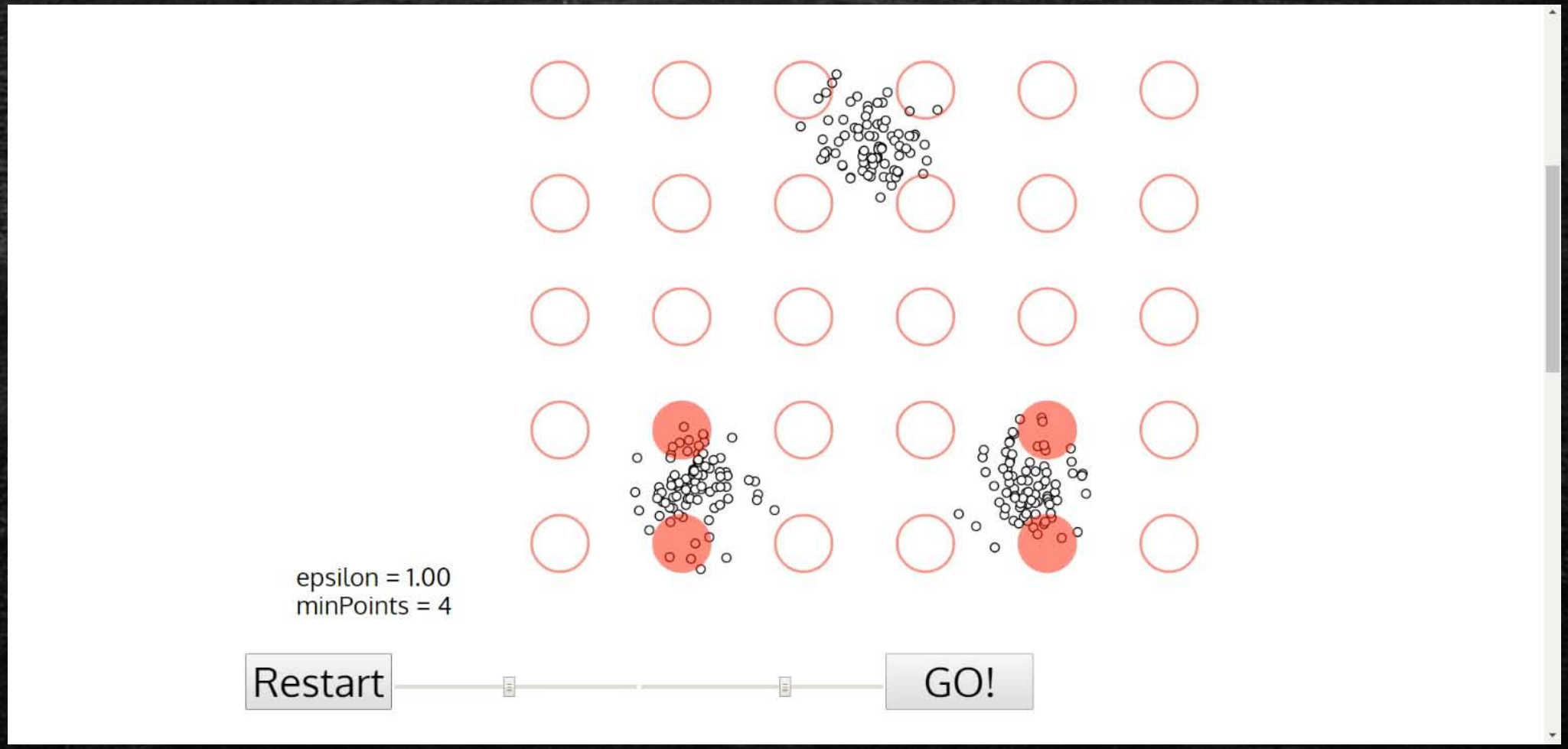


Figure 1: *DBSCAN* **clustering with** $minPoints = 4$

DBSCAN Visualization



DBSCAN serial algorithm

Algorithm 1 The DBSCAN algorithm. Input: A set of points X, distance threshold eps, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1:  procedure DBSCAN(X, eps, minpts)
2:      for each unvisited point x ∈ X do
3:          mark x as visited
4:          N ← GETNEIGHBORS(x, eps)
5:          if |N| < minpts then
6:              mark x as noise
7:          else
8:              C ← {x}
9:              for each point x' ∈ N do
10:                 N ← N \ x'
11:                 if x' is not visited then
12:                     mark x' as visited
13:                     N' ← GETNEIGHBORS(x', eps)
14:                     if |N'| ≥ minpts then
15:                         N ← N ∪ N'
16:                 if x' is not yet member of any cluster then
17:                     C ← C ∪ {x'}
```

- The algorithm proceeds by arbitrarily picking up point in the dataset (until all points have been visited).
- If there are at least 'minPoint' points within a radius of 'ε' to the point then we consider all these points to be part of the same cluster.
- The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point.
- The complexity of this algorithm is $O(n^2)$, where n is the number of points.
- With spatial indexing (kd-tree or r-tree), the complexity is $O(n \log n)$.

Parallelizing DBSCAN

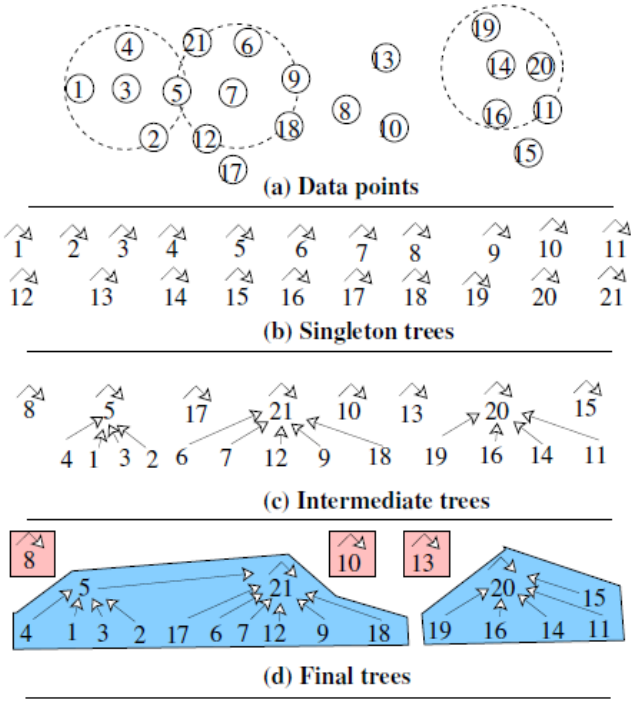


Figure 1. An example showing the proposed DSDBSCAN algorithm at different stages. (a) Sample data points with the circles denoting *eps* and $minpts = 4$. (b) The singleton trees when the algorithm starts. (c) Intermediate trees after exploring the *eps*-neighborhood of three randomly selected points 3, 7, and 14. (d) The resulting trees when the algorithm terminates where the singleton trees (8, 10, and 13) are noise points and the two blue colored large trees (rooted at 20 and 21) are clusters.

- Patwari et. al. (2011) presented a disjoint set based DBSCAN algorithm (DSDBSCAN).
- The algorithm uses a disjoint set data structure.
- Initially all points belong to a singleton tree.
- If two points belong to the same cluster, their trees are merged.
- The process is repeated until all clusters have been found.

Parallelizing DBSCAN

Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X, distance eps, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1:  procedure DSDBSCAN(X, eps, minpts)
2:      for each point x ∈ X do
3:          p(x) ← x
4:      for each point x ∈ X do
5:          N ← GETNEIGHBORS(x, eps)
6:          if |N| ≥ minpts then
7:              mark x as core point
8:              for each point x' ∈ N do
9:                  if x' is a core point then
10:                     UNION(x, x')
11:                 else if x' is not yet member of any cluster then
12:                     mark x' as member of a cluster
13:                     UNION(x, x')
```

- The DSDBSCAN algorithm uses the Rem's algorithm(1976) to construct the disjoint set tree structure.
- The complexity of DSDBSCAN is *O(n log n).*
- The cluster trees are constructed without any specific order.
- The DSDBSCAN algorithm is hence suitable for a parallel implementation.

Parallel DBSCAN on Distributed Memory computers(PDSDBSCAN-D)

- Since the DSDBSCAN algorithm constructs the tree sets arbitrarily, this algorithm can be highly parallelized.
- The data set is split into ‘p’ portions and each processor runs a sequential DSDBSCAN algorithm to get the local clusters(trees).
- All the local clusters are then merged to get the final clusters.
- In the merging phase the Master just switched the pointer of root of one tree to the other root to form the root of the merged tree, when relabeling is done.

PDSDBSCAN-D Algorithm

Algorithm 5 The parallel DBSCAN algorithm on a distributed memory computer (PDSDBSCAN-D) using p processors. Input: A set of points X, distance eps, and the minimum number of points required to form a cluster, $minpts$. Let X be divided into p equal disjoint partitions $\{X_1, X_2, \ldots, X_p\}$ for the p running processors. Each processor t also has a set of remote points, X'_t stored locally to avoid communication. Each processor t runs PDSDBSCAN-D to compute the clusters. Output: A set of clusters.

```
1:  procedure PDSDBSCAN-D(X, eps, minpts)
2:      for each point x ∈ X_t do                    ▷ Stage: Local comp. (Line 2-16)
3:          p(x) ← x
4:      for each point x ∈ X_t do
5:          N ← GETLOCALNEIGHBORS(x, eps, X_t)
6:          N' ← GETREMOTENEIGHBORS(x, eps, X'_t)
7:          if |N| + |N'| ≥ minpts then
8:              mark x as a core point
9:              for each point y ∈ N do
10:                 if y is a core point then
11:                     UNION(x, y)
12:                 else if y ∉ any cluster then
13:                     mark y as member of a cluster
14:                     UNION(x, y)
15:             for each point y' ∈ N' do
16:                 Y_t ← Y_t ∪ UNIONQUERY(x, P_x, y')   ▷ P_x = P_t
17:     Send UNIONQUERY(x, P_x, y') ∈ Y_t to P_y'   ▷ Stage: Merging (L. 17-28)
18:     Receive UNIONQUERY from other processors
19:     for each received UNIONQUERY (x, P_x, y') do
20:         if y' is a core point then
21:             PARALLELUNION(x, P_x, y')
22:         else if y' ∉ any cluster then
23:             mark y' as member of a cluster
24:             PARALLELUNION(x, P_x, y')
25:     while (any processor has any UNIONQUERY) do
26:         Receive UNIONQUERY from other processors
27:         for each received UNIONQUERY (x, P_x, y') do
28:             PARALLELUNION(x, P_x, y')
```

Local Computation

Merging

Algorithm 6 Merging trees containing x and y' on $P_{y'}$ [30]

```
1:  procedure PARALLELUNION(x, P_x, y')
2:      r = FIND(y')
3:      if p(r) = r then          ▷ r is a global root at P_t = P_r = P_y'
4:          if r < x then
5:              p(r) = x
6:          else
7:              Send UNIONQUERY(r, P_r, x) to P_x
8:      else                      ▷ r is the last point on P_t on the find path towards the global root
9:          if p(r) < x then
10:             Send UNIONQUERY(x, P_x, p(r)) to P_p(r)
11:         else
12:             Send UNIONQUERY(p(r), P_p(r), x) to P_x
```

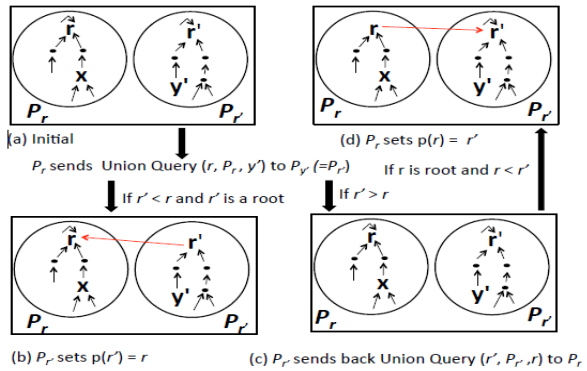


Figure 2. An example of the processing of a PARALLELUNION(x, P_x, y') operation. (a) Initial condition of the trees on P_r and $P_{r'}$ containing x and y', respectively. (b) After the merging when $p(r')$ is set to r. (c) $P_{r'}$ sends a UNIONQUERY(r', $P_{r'}$, P_r) to P_r. (d) After the merging when $p(r)$ is set to r'. The tree spans multiple processors after the PARALLELUNION operation.

HPDBSCAN (Highly Parallel DBSCAN)

- HPDBSCAN algorithm is an efficient parallel version of DBSCAN algorithm that adopts core idea of the grid based clustering algorithm.
- Proposed by Götz et. al. in 2015.
- The input data is overlaid with a hypergrid, which is then used to perform DBSCAN clustering.
- The grid is used as a spatial structure, which reduces the search space for neighborhood queries and facilitates efficient partitioning of the dataset along the cell boundaries.

HPDBSCAN

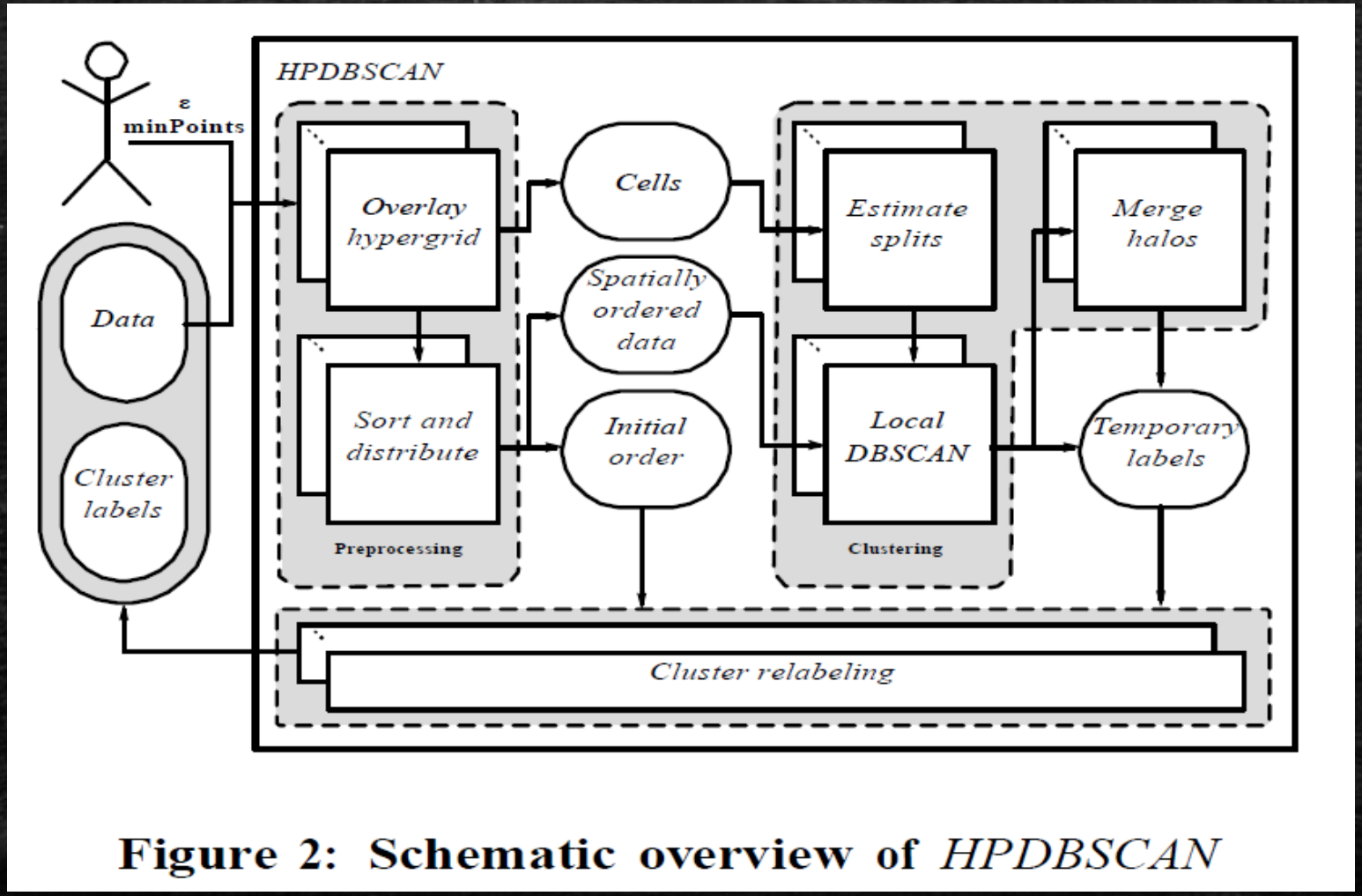


Figure 2: Schematic overview of *HPDBSCAN*

HPDBSCAN

- The HPDBSCAN algorithm has four main phases:
  - Data loading and pre-processing
  - Local clustering
  - Cluster merging
  - Cluster re-labeling

Data Preprocessing

- In this phase, the bounding region of the entire dataset is overlaid by a regular, non-overlapping hypergrid.
- This hypergrid is split into ‘p’ subspaces, each of which is assigned to one processor.
- Each processor than reads equal sized chunk of data in arbitrarily fashion.
- Then using the hashmap with offset and pointers the indexing is done with respect to the spatial alignment of the data point.
- Each processor then checks if the data points it has are a part of it’s assigned subspace or not. If not the point is sent to the appropriate destination processor. This is the redistribution phase.

Data Preprocessing(Indexing)

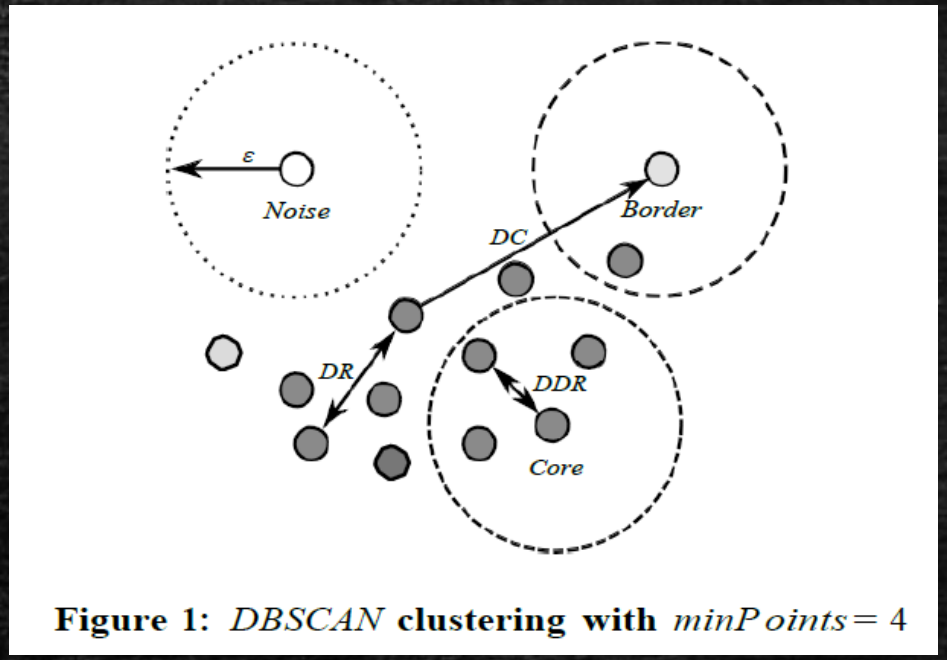
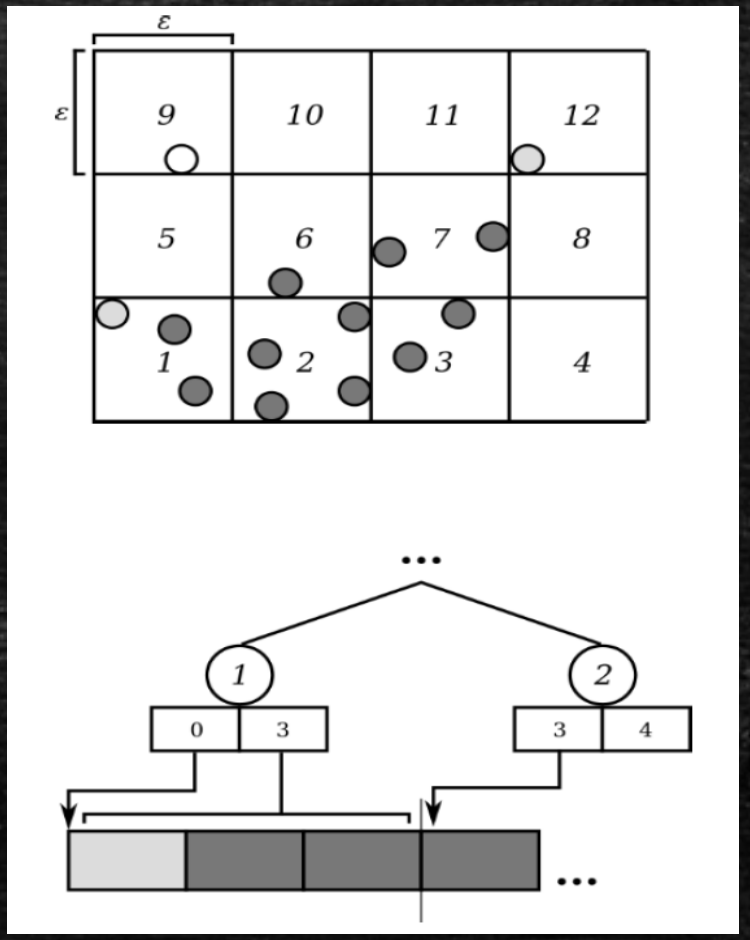


Figure 1: *DBSCAN* **clustering with** $minPoints = 4$



Data Preprocessing

- The benefit of using the indexed structure is that the neighborhood queries have *O(1)* complexity.
- Cell Neighborhood : The cell neighborhood $N_{Cell}(c)$ of a given cell c denotes all cells d from the space of all available grid cells C that have a Chebychev distance $dist_{Chebychev}$ of zero or one to c, i.e., $N_{Cell}(c) = \{ d \mid d \in C \wedge dist_{Chebychev}(c, d) \leq 1 \}$.
- To get all neighborhood points within an assigned subspace, the processor need an additional one cell-thick layer of redundant data items. This is known as *halos or ghost* cells. These are transferred during the redistribution phase.
- After the redistribution phase, a local DBSCAN algorithm is run locally at each of the processors.
- To ensure a balanced data space division, they use a cost heuristic.

Cost Heuristic

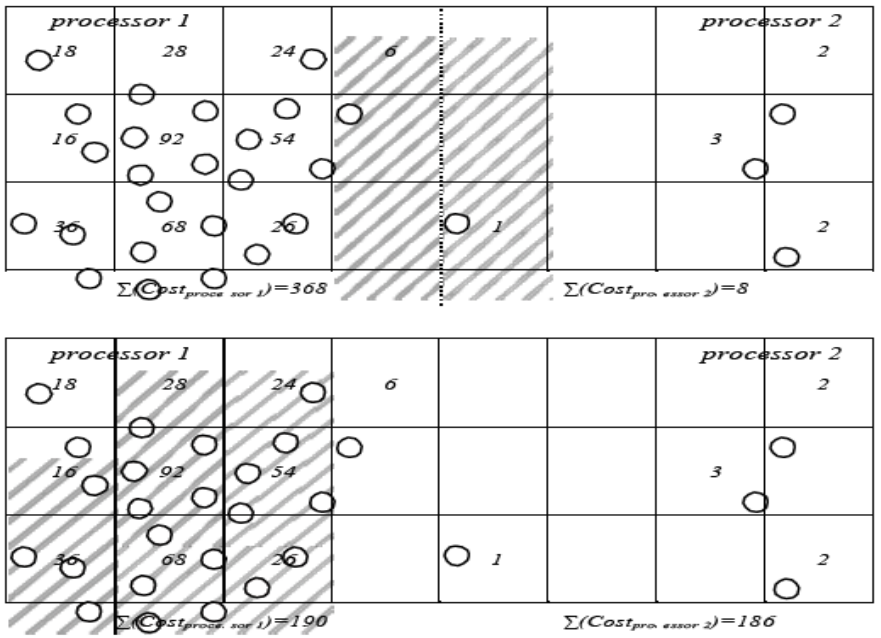


Figure 4: Impact of naive and heuristic-based hypergrid decompositions on compute load balancing. Halo cells are marked with a hatched pattern.

Local DBSCAN

```
1  def localDBSCAN(X, eps, minPts):
2    rules = Rules()
3    @parallel
4    for p in X:
5      Cp, Np = query(p, X, eps)
6      if length(Np) >= minPts:
7        markAsCore(p)
8        add(Cp, p)
9        for q in Np:
10         Cq = getCluster(q)
11         if isCore(q):
12           markAsSame(rules, Cp, Cq)
13         add(Cp, q)
14     elif notVisited(p):
15       markAsNoise(p)
16   return rules
```

Listing 2: Local *DBSCAN* pseudocode

Local DBSCAN

- For each point *p* the epsilon neighborhood is computed independently.
- If a point has at least minPoints neighbors and none of the neighbors is already labeled, *p* is marked as cluster core and cluster label created using *p*'s index.
- In the case when any one of the point is already labeled, we mark cluster equivalences in rules and move on.
- The cluster equivalence information is later used in the merger stage.
- The result of local DBSCAN is a list of sub-clusters along with the points and cores they contain, a list of noise points, and a set of rules describing which cluster labels are equivalent.

Cluster Merging and Re-labeling

- The label-mapping rules across different nodes are created based on the labeling rules generated by local DBSCAN.
- After the local DBSCAN run, each halo zone is passed to the node that owns the actual neighboring data chunk.
- Based on the comparison of local and halo points a set of re-labeling rules are generated. These rules are then broadcasted.
- To ensure uniqueness of cluster labeling the label of a cluster is taken as the lowest index of a core point in the cluster.
- Using these rules, final relabeling is done.

THANK YOU