Clustering Algorithms for High Dimensional Data Literature Review

S. Geetha[#], K. Thurkai Muthuraj[*]

[#]*Department of Computer Applications, Mepco Schlenk Engineering College, Sivakasi, TamilNadu, India*
[*]*Tata Consultancy Services*

Abstract **– In modern world, the complex data sets are growing. Clustering high dimensional data is challenging due to its dimensionality problem and it affects time complexity, space complexity, scalability and accuracy of clustering methods. This review will be more helpful to find clustering algorithms suitable for high dimensional data.**

I. INTRODUCTION

In modern scientific and business domains, the high-dimensional data are involved. Clustering in high dimensional spaces presents much difficulty [1]. When talking about clustering high dimensional data, the clustering steps, Dimensionality Reduction, Subspace Clusteringand Co-Clustering will be more helpful to address the problem of high dimension [20].

Clustering is helpful to understand the structure and abstract of the large data set [2]. Clustering methods are available for categorical data, spatial data, etc. Clustering methods are applied in object recognition, pattern recognition, image processing, text mining and information retrieval [21].

Clustering means partitioning data point into a set of groups.Simplification can be achieved by representing data in fewer clusters [3]. There are number of clustering algorithms introduced for clustering high dimensional data. It is broadly classified into partitioning and Hierarchical.Partitioning subdivided into K-means and K-medoids [8].CLARA and CLARANS are popular to deal with large datasets. Hierarchical further classified into Agglomerative and Divisive. BIRCH, Chameleon, ROCK and CURE are examples of hierarchical method which deal with large amount of data [5]. Other categories of clustering methods are Model based clustering, Density based clustering, Grid–based clustering, and Constrained based clustering.

Cluster analysis has been an area of research for many decades. Many new methods are still being developed. In section 2, popular and mostly used clustering algorithms are discussed.

II. HIGH DIMENSIONAL DATA

A dataset with large amount of attributes or features is called high dimensional data. There are many application domains where the data is of considerably higher dimensionality such as spatial data, medical data, gene data, ecological data, social media data, web log data, financial data, etc [4].The dimensionality of data makes learning problems hardly amenable. In particular, the high dimensionality of data is a highly challenging factor for the clustering task [13]. The following problems need to be faced for clustering high-dimensional data:

- Curse of dimensionality.
- Concentration effects.
- Local feature relevance problem.
- In arbitrarily oriented affine subspaces.
- High-dimensional data could likely include irrelevant features, which may obscure the effect of the relevant ones.

Forthcoming section of the paper is organized as follows. In Section 3, we review clustering algorithms, based on the naturesof generated clusters and techniques and theories behind them [10].Furthermore, we discuss approaches for clustering large data sets, and high-dimensional data.

III. HIGH-DIMENSIONAL DATA CLUSTERING ALGORITHMS

This paper presents various clustering algorithms with by considering the problems of high dimensional such as time complexity, space complexity, scalability, etc. The overview of clustering taxonomy for high dimensional data is shown the figure 3.1.

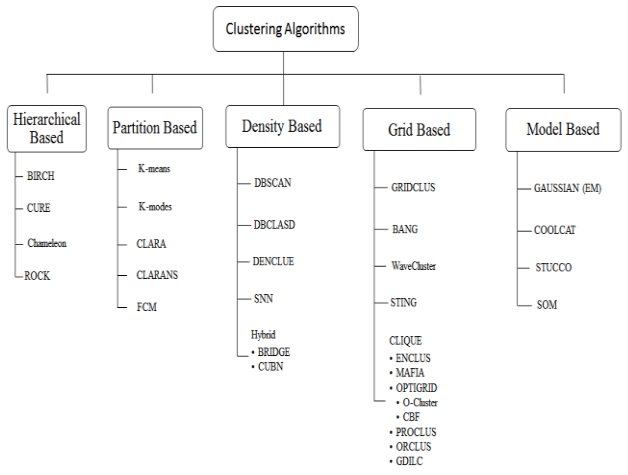


Fig 1. Overview of Clustering Algorithms

Hierarchical Based Clustering

Hierarchical clustering techniques is one of the clustering technique used for clustering of high dimensional data. Hierarchical clustering techniques works in two ways. One is Agglomerative (top-bottom) and another on is Divisive (bottom- top). In Agglomerative approach,initially starts with one object and successively merges the neighbour objects based on the distance (minimum, maximum and average) [9]. The process is continuous until a desired cluster is shaped. In Divisive approach, starts with set of objects as single cluster and divides them into further clusters until desired number of clusters are shaped. BIRCH [2], CURE [16], ROCK, Chameleon [6], Echidna, Wards, CACTUS are some of Hierarchical clustering.

Partition Based Clustering

All objects are considered initially as a single cluster. The objects are divided into number of partitions by iteratively locating the points between the partitions. The partitioning algorithms like K-means, K-modes, ROCK, CLARA, CLARANS, and FCM.

Density Based Clustering

Data objects are categorized into core points, border points and noise points. All the core points are connected together based on the densities to form cluster [7]. Arbitrary shaped clusters are formed by various clustering algorithms such as DBSCAN, DBCLASD, DENCLU, SUBCLU, SNN and hybrid clustering algorithms such as BRIDGE (which combines k-means and density based) and CUBN (which combines density based and distance based).

Grid Based Clustering

Grid based algorithm partitions the data set into number of cells to form a grid structure. Clusters are formed based on the grid structure [9]. To form clusters, Grid algorithm uses subspace and hierarchical clustering techniques. GRIDCLUST, BANG, STING, Wave cluster [11], CLIQUE, ENCLUS [14], MAFIA [12], OptiGrid (O-cluster and CBF), PROCLUS, ORCLUS, GRIDCLUS, GDILC [9], FC and STIRR. Compare to all Clustering algorithms, Grid algorithms are very fast processing algorithms.

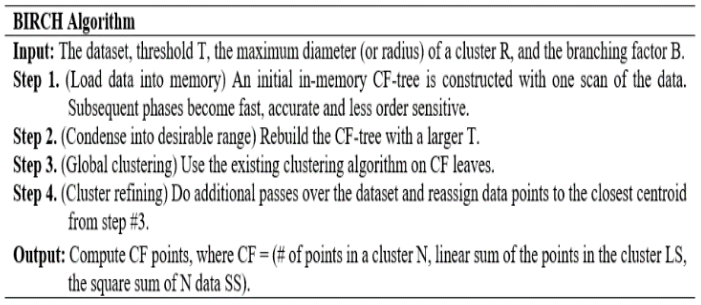
Model Based Clustering

Set of data points are connected together based on various strategies like statistical methods, conceptual methods, and robust clustering methods [19]. There are two approaches for model based algorithms one is neural network approach and another one is statistical approach [17].Algorithms such as EM [19], COOLCAT, STUCCO, SOM [18], and SLINK are well known Model based clustering algorithms.

A. Hierarchical Based Clustering

BIRCH -Balanced **I**terative **R**educing and **C**lustering using **H**ierarchies

It is an unsupervised algorithm used to perform clustering on large datasets. It is the first clustering algorithm proposed in the database area to handle noise effectively



BIRCH Algorithm

Input: The dataset, threshold T, the maximum diameter (or radius) of a cluster R, and the branching factor B.

Step 1. (Load data into memory) An initial in-memory CF-tree is constructed with one scan of the data. Subsequent phases become fast, accurate and less order sensitive.

Step 2. (Condense into desirable range) Rebuild the CF-tree with a larger T.

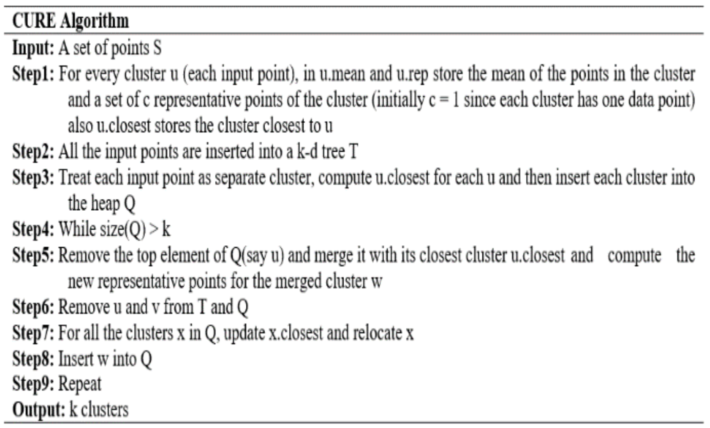
Step 3. (Global clustering) Use the existing clustering algorithm on CF leaves.

Step 4. (Cluster refining) Do additional passes over the dataset and reassign data points to the closest centroid from step #3.

Output: Compute CF points, where CF = (# of points in a cluster N, linear sum of the points in the cluster LS, the square sum of N data SS).

CURE – Clustering **U**sing **RE**presentatives

It is a novel hierarchical clustering algorithm that adopts a middle ground between the centroid-based and the all point extremes.



CURE Algorithm

Input: A set of points S

Step1: For every cluster u (each input point), in u.mean and u.rep store the mean of the points in the cluster and a set of c representative points of the cluster (initially c = 1 since each cluster has one data point) also u.closest stores the cluster closest to u

Step2: All the input points are inserted into a k-d tree T

Step3: Treat each input point as separate cluster, compute u.closest for each u and then insert each cluster into the heap Q

Step4: While size(Q) > k

Step5: Remove the top element of Q(say u) and merge it with its closest cluster u.closest and compute the new representative points for the merged cluster w

Step6: Remove u and v from T and Q

Step7: For all the clusters x in Q, update x.closest and relocate x

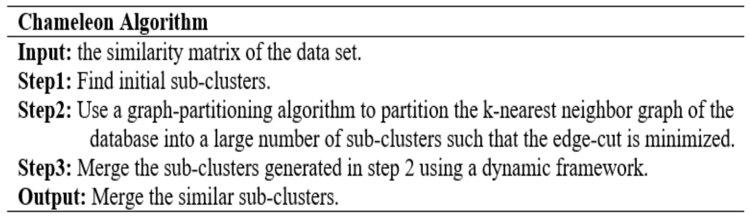
Step8: Insert w into Q

Step9: Repeat

Output: k clusters

Chameleon

The chameleon algorithm is a graph-based clustering algorithm. Given a similarity matrix of the database, construct a sparse graph representation of the data items based on the commonly used k-nearest neighbor graph approach. Finally, use the agglomerative hierarchical clustering algorithm to merge the most similar sub-clusters by taking into account the relative interconnectivity and closeness of the clusters.



Chameleon Algorithm

Input: the similarity matrix of the data set.

Step1: Find initial sub-clusters.

Step2: Use a graph-partitioning algorithm to partition the k-nearest neighbor graph of the database into a large number of sub-clusters such that the edge-cut is minimized.

Step3: Merge the sub-clusters generated in step 2 using a dynamic framework.

Output: Merge the similar sub-clusters.

ROCK –Robust **C**lustering using lin**K**s

ROCK is an agglomerative hierarchical clustering algorithm that employs links to merge clusters. ROCK uses the link-based similarity measure to measure the similarity between two data points and between two clusters.

```
ROCK Algorithm
Input: n - number of data points, D - data set
Step1: for i = 1 to n − 1 do
Step2:         for j = i + 1 to n do
Step3:                 Compute link($p_i$, $p_j$)
Step4:         end for
Step5: end for
Step6: for i = 1 to n do
Step7:         Build a local heap q[i] that contains every cluster $C_j$ such that link[$C_i$, $C_j$] is non-zero {at
               first, each data point forms a cluster}
Step8: end for
Step9: Build a global heap Q that is ordered in decreasing order of the best goodness measure and contains
       all the clusters
Step10: repeat
Step11: Merge the max cluster $C_j \in Q$ and the max cluster in q[j] into a new cluster W, delete the two
        clusters from Q, and update link
Step12: until the number of links between every pair of the remaining clusters becomes zero or size(Q) = k
Output: Merge the clusters
```

Table 1. Computational Complexity and Scalability Analysis on Hierarchical based Clustering Algorithms

Algorithms	Computational Complexity	Scalability
BIRCH	Time Complexity: $O(n)$	Y
CURE	Time Complexity: $O(n^2 \log n)$ Space Complexity: $O(n)$	Y
Chameleon	Time Complexity: $O(nm + n \log n + m^2 \log m)$	Y
ROCK	Time Complexity: $O(n^2 + nm_m m_a + n^2 \log n)$ Space Complexity: $O(\min\{n^2, nm_m m_a\})$	Y

B. Partition Based Clustering

K-Means

k-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem.

```
k-Means Algorithm
Input: Data set D, Number of Clusters k, Dimensions d:
        {$C_i$ is the i^th cluster}
        {1. Initialization Phase}
Step1: ($C_1$, $C_2$,…,$C_k$) = Initial partition of D.
        {2. Iteration Phase}
Step2: repeat
Step3:          $d_{ij}$ = distance between case i and cluster j
Step4:          $n_i = \arg\min_{1\leq j\leq k} d_{ij}$
Step5:          Assign case i to cluster $n_i$
Step6:          Re-compute the cluster means of any changed clusters above
Step7: until no further changes of cluster membership occur in a complete iteration
Output: Results
```

K-modes

The k-modes algorithm comes from the k-means algorithm, and it was designed to cluster categorical data sets. The main idea of the k-modes algorithm is to specify the number of clusters (say, k) and then to select k initial modes, followed by allocating every object to the nearest mode.

```
k-modes Algorithm
Input: Data set D, Number of Clusters k, Dimensions d:
Step1: Select k initial modes Q = {$q_1, q_2, . . . , q_k$}, and $q_l$ for cluster l
Step2:   for i = 1 to n do
Step3:           Find an l such that $d_{sim}(x_i, q_l) = \min_{1\leq t\leq k} d_{sim}(x_i, q_t)$
Step4:           Allocate $x_i$ to cluster l
Step5:           Update the mode $q_l$ for cluster l
Step6: end for
Step7: repeat
Step8:   for i = 1 to n do
Step9:                   Let l0 be the index of the cluster to which $x_i$ belongs
Step10:          Find an $l_1$ such that $d_{sim}(x_i, q_{l_2}) = \min_{1\leq t\leq k, t\neq l_0} = d_{sim}(x_i, q_t)$

Step11:          if $d_{sim}(x_i, q_{l_1})) < d_{sim}(x_i, q_{l_0})$ then
Step12:                  Reallocate $x_i$ to cluster $l_1$
Step13:                  Update $q_{l_0}$ and $q_{l_1}$
Step14:          end if
Step15: end for
Step16: until No changes in cluster membership
Output: Results.
```

CLARA –Clustering **LAR**ge **A**pplication

CLARA algorithm randomly chooses a small portion of the actual data as a representative of the large data. A sample dataset D' is drawn from the original dataset D, and the PAM algorithm is applied to D' to find the k medoids. Calculate the current dissimilarity using these k medoids and the dataset D. If it is smaller than the previous iteration, then these k medoids are kept as the best k medoids. The whole process is performed iteratively till getting the best clusters.

```
CLARA Algorithm
Input: X, d, k
Step1: bestDissim ← ∞
Step2: for t ← 1 to S
Step3: do X' ← RANDOM-SUBSET(X, s)
Step4:          D ← BUILD-DISSIM-MATRIX(X',d)
Step5:          (C',M) ← PAM(X',D,k)
Step6:          C ← ASSIGN-MEDOIDS(X,M,S)
Step7:          dissim ← TOTAL-DISSIM(C,M,D)
Step8:          if dissim < bestDissim
Step9:                  then bestDissim ← dissim
Step10:                 $C_{best}$ ← C
Step11:                 $M_{best}$ ← M
Step12: return ($C_{best}$, $M_{best}$)
Output: Best clusters
```

CLARANS –Clustering **L**arge **A**pplications based on **RAN**domized **S**earch

CLARANS has two parameters: the maximum number of neighbors examined (Max neighbor) and the number of local minima obtained (numlocal). The higher the value of max neighbor, the closer is CLARANS to PAM, and the longer is each search of a local minima. But, the quality of such local minima is higher and fewer local minima need to be obtained.

```
CLARANS Algorithm
Input: Parameters numlocal and maxneighbor. Initialize i to 1, and mincost to a large number
Step1: Set current to an arbitrary node in $G_{n,k}$
Step2: Set j to 1
Step3: Consider a random neighbor S of current, and based on 5, calculate the cost differential of the two nodes
Step4: If S has a lower cost, set current to S, and go to Step 3
Step5: Otherwise, increment j by 1. If j maxneighbor, go to Step 4
Step6: Otherwise, when j > maxneighbor, compare the cost of current with mincost. If the former is less than
       mincost, set mincost to the cost of current and set bestnode to current
Step7: Increment i by 1. If i > numlocal, output bestnode and halt. Otherwise, go to Step 2
Output: Best node
```

FCM –Fuzzy **C-M**eans

The FCM algorithm partitions X into c fuzzy clusters and find out each clusters center so that the cost function (objective function) of dissimilarity measure is minimization or below a certain threshold. FCM analyze membership value of each data in each cluster.

FCM Algorithm

```
Input: Given the dataset, set the desire number of clusters c, the fuzzy parameter m (a constant > 1),
       stopping condition, initialize the fuzzy partition matrix, and set stop = false
Step 1: Do:
Step 2:   Calculate the cluster centroids and the objective value J
Step 3:   Compute the membership values stored in the matrix
Step 4:   If the value of J between consecutive iterations is less than the
              stopping condition, then stop = true
Step 5: While (!stop)
Output: A list of c cluster centres and a partition matrix are produced
```

Table 2. Computational Complexity and Scalability Analysis on Partition based Clustering Algorithms

Algorithms	Computational Complexity	Scalability
k-means	Time Complexity: O(I*K*m*n) Space Complexity: O((m+K)n)	Y
k-modes	Time Complexity: O(|U|)	N
CLARA	Time Complexity: O(ks^2+k(n-k))	Y
CLARANS	Time Complexity: O(n^2)	Y
FCM	Time Complexity: O(tkNn^2)	Y

C. Density Based Clustering

DBSCAN - Density-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise

Density based clustering algorithm has played a vital role in finding non-linear shapes structure based on the density. DBSCAN is most widely used density based algorithm. Only one input parameter is required, and the algorithm also supports the user in determining an appropriate value for this input parameter.

DBSCAN Algorithm

```
Input: Let X = {$x_1, x_2, x_3, ..., x_n$} be the set of data points. DBSCAN requires two parameters: ε (eps) and the
       minimum number of points required to form a cluster (minPts).
Step1: Start with an arbitrary starting point that has not been visited.
Step2: Extract the neighborhood of this point using ε (All points which are within the ε distance are
       neighborhood).
Step3: If there are sufficient neighborhood around this point then clustering process starts and point is marked
       as visited else this point is labelled as noise (Later this point can become the part of the cluster).
Step4: If a point is found to be a part of the cluster then its ε neighborhood is also the part of the cluster and
       the above procedure from step 2 is repeated for all ε neighborhood points. This is repeated until all
       points in the cluster is determined.
Step5: A new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.
Step6: This process continues until all points are marked as visited.
```

DBCLASD –Distribution **B**ased **C**lustering of **La**rge **S**patial **D**atabases

DBCLASD is an incremental algorithm, i.e. the assignment of a point to a cluster is based only on the points processed so far without considering the whole cluster or even the whole database. DBCLASD incrementally augments an initial cluster by its neighboring points as long as the nearest neighbor distance set of the resulting cluster still fits the expected distance distribution.

DBCLASD Algorithm

```
Input: initialize the points of db as being assigned to no cluster
Step1: initialize an empty list of candidates
Step2: initialize an empty list of unsuccessful candidates
Step3: initialize an empty set of processed points
Step4: for each point p of the database db do
Step5:          if p has not yet been assigned to some cluster then
Step6:                  create a new cluster C and insert p into C
Step7:                  reinitialize all data structures for cluster C
Step8:                  expand cluster C by 29 neighboring points
Step9:          for each point p1 of the cluster C do
Step10:                 answers := retrieve_neighborhood(C,p1)
Step11:                 update_candidates(C,answers)
Step12:         end for each point p1 of the cluster C
Step13:         expand_cluster (C)
Step14: end if p has not yet been assigned to some cluster
Step15: end for each point of the database
Step16: procedure expand_cluster (cluster C)
Step16: procedure expand_cluster (cluster C)
Step17: change := TRUE
Step18: while change do
Step19:         change := FALSE
Step20:         while the candidate list is not empty do
Step21:                 remove the first point p from the candidate list
Step22:                 assign it to cluster C
Step23:         if distance set of C still has the expected distribution then
Step24:                 answers:= retrieve_neighborhood(C,p)
Step25:                 update_candidates(C,answers)
Step26:                 change := TRUE
Step27:         else
Step28:                 remove p from the cluster C
Step29:                 insert p into the list of unsuccessful candidates
Step30:         end if distance set still has the expected distribution
Step31: end while the candidate list is not empty
Step32: list of candidates := list of unsuccessful candidates
Step33:end while change
Step34: listOfPoints procedure retrieve_neighborhood(cluster C; point p)
Step35: calculate the radius m according to (ii) in section 4.1
Step36: return result of circle query with center p and radius m
Step37: procedure update_candidates(cluster C; listOfPoints points)
Step38: for each point in points do
Step39:         if point is not in the set of processed points then
Step40:                 insert point at the tail of the list of candidates
Step41:                 insert point into the set of processed points
Step42:         end if point is not in the set of processed points
Step43: end for each point in points
```

DENCLUE -DENsity-based **CLU**st **E**ring

The DENCLUE algorithm works in two steps. Step one is a pre-clustering step, in which a map of the relevant portion of the data space is constructed. The map is used to speed up the calculation of the density function which requires to efficiently access neighboring portions of the data space. The second step is the actual clustering step, in which the algorithm identifies the density-attractors and the corresponding density-attracted points.

DENCLUE Algorithm

```
Input: The dataset,σ, and ξ.
Step1: Take dataset in a map which has each side is of 2σ, consider only populated cubes.
Step2: Calculate the mean of each populated cubes.
Step3: Find highly populated cubes.
Step4: Determine the connection between each highly populated cube, and other cubes (Highly or just
       populated cubes) by the distance between their means. If d(mean(c1), mean(c2)) < 4σ, then the two
       cubes are connected.
Step5: Only the highly populated cubes and cubes which are connected to a highly populated cube are
       considered in determining clusters.
Step6: Find the representative of the hyper-cube.
Step7: Connecting the representatives of hyper-cubes having the same path to form a cluster.
Output: Assignment of data values to clusters.
```

SNN –Shared **N**ear **N**eighbour Graph)

Initially, the SNN algorithm finds the K nearest neighbours of each point of the dataset. The similarity between pairs of points is calculated in terms of how many nearest neighbours the two points share. The points are classified as core points, if the density of the point is equal or greater than MinPts (core point threshold). If any points that are not classified into any cluster will be classified as noise points.

```
SNN Algorithm
Input: K, the neighbours' list size, Eps, the threshold density, MinPts, the threshold that define the core
       points.
Step1: Identify the k nearest neighbours for each point (the k points most similar to a given point, using a
       distance function to calculate the similarity).
Step2: Calculate the SNN similarity between pairs of points as the number of nearest neighbours that the two
       points share. The SNN similarity is zero if the second point in not in its list of k nearest neighbours,
       and vice-versa.
Step3: Calculate the SNN density of each point: number of nearest neighbours that share Eps or greater
       neighbours.
Step4: Detect the core points. If the SNN density of the point is equal or greater than MinPts then classify the
       point as core.
Step5: Form the cluster from the core points. Classify core points into the same cluster if they share Eps or
       greater neighbours.
Step6: Identify the noise points. All non-core points that are not within a radius of Eps of a core point is
       classified as noise.
Step7: Assign the remainder points to the cluster that contains the most similar core point.
Output: Assignment of data values to clusters.
```

Hybrid Clustering Algorithm

BRIDGE (Combines k-means and density based)

BRIDGE, which integrates the popular k-means algorithm and the density-based algorithm DBSCAN. BRIDGE enables DBSCAN to handle very large databases and at the same time improves the quality of k-means clusters by removing noise.

```
BRIDGE Algorithm
Input: k: number of clusters.
Step1: Run the k-means algorithm and label each data point with the k-means cluster ID and core/$\epsilon$-
       core/noncore.
Step2: Determine $\epsilon$ and $N_{min}$ for DBSCAN.
Step3: Run DBSCAN for core and $\epsilon$ -core points of each k-means cluster.
Step4: Run DBSCAN for all core and noncore points.
Step5: Resolve multiple cluster IDs.
Step6: Run the k-means algorithm without the noise found in DBSCAN, taking earlier centers as initia
       points.
```

CUBN (Combines density based and distance based)

The CUBN algorithm consists of three phases. At the first phase, the erosion operation is used to find border points. Then the nearest neighbor method is used to cluster the border points. Finally, the nearest neighbor method is employed to cluster the inner points.

```
CUBN Algorithm
Algorithm for Finding border points
Input: the data set S (n points in d-dimensional space), p – the value in the matrix B
Step1: $r = {}^{p}/_{\sqrt{2}}$
Step2: for each column b in matrix B
Step3:         for each point in x in S
Step4:         $\{ k = \#\{ y \mid y \in O(x+b, r) \cap S\}$
Step5:         if k = 0
Step6:         $C \leftarrow C \cup \{x\} \}$
Output: C – a set of border points

Algorithm for clustering border points
Input: C – a set of border points
Step1: regard the C as the initial PC, i.e., PC ← C;
Step2: repeat
Step3:         for each $PC_i$ and $PC_j$
Step4:                 if $\exists u \in PC_i, \exists u \in PC_j$, and distance (u,v) ≤ e
Step5:                 merge $PC_i$ and $PC_j$, i.e., $PC_i \leftarrow PC_i \cup PC_j$
Step6: until no change
Output: PC – a set of border clusters

Algorithm for clustering inner points
Input: PC – a set of border cluster
Step1: Z ← S –C, i.e., Z is the set of all inner points
Step2: RC ← ∅
Step3: while Z≠ ∅
Step4:         I = ∅
Step5:         for each inner point x
Step6:                 for each $PC_i$
Step7:                         if $\exists y \in PC_i$ and distance (x, y) ≤ e
Step8:                                 $I_i \leftarrow I_i \cup \{x\}$
Step9:                                 $Z \leftarrow Z - \{x\}$
Step10: PC ← I
Step11: RC ← RC ∪ I
Output: RC – a set of clusters
```

Table 3. Computational Complexity and Scalability Analysis on Density based Clustering Algorithms

Algorithms	Computational Complexity	Scalability
DBSCAN	Time Complexity: O(m log m) Space Complexity: O(m)	N
DBSCLAD	Time Complexity: $O(3n^2)$	Y
DENCLUE	Time Complexity: O(log|D|)	Y
SNN	Time Complexity: $O(n^2)$	Y
BRIDGE	Time Complexity: O(n)	Y
CUBN	Time Complexity: O(n)	N

D. Grid Based Clustering

GRIDCLUS – GRIDCLUStering

```
GRIDCLUS Algorithm
Algorithm GRIDCLUS
Step1: Initialization
Step2: Create the Grid Structure
Step3: Calculate the block densities $D_{Bi}$
Step4: Generate a sorted block sequence S = <$B_{1'}$. $B_{2'}$, ... $B_{b'}$>
Step5: Mark all blocks 'not active' and 'not clustered'
Step6: while a 'not active' block exist do
Step7:          u := u + 1
Step8:          Find active blocks $B_{i'}$ .. $B_{j'}$
Step9:          for each 'not clustered' block $B_{k'}$ := $B_{i'}$ .. $B_{j'}$ do
Step10:         create a new cluster set C[u]
Step11:         W[u] := W[u] + 1
Step12:         C[u, W[u]] <- $B_{k'}$
Step13:         mark block Bk' clustered
Step14:         NEIGHBOR-SEARCH($B_{k'}$, C[u, W[u]})
Step15: endfor
Step16: for each 'not active' block $B_l$ do
Step17:         W[u] := W[u] + 1
Step18:         C[u, W[u]] <- $B_l$
Step19: endfor
Step20: Mark all blocks 'not clustered'
Step21: endwhile
end GRIDCLUS
Procedure NEIGHBOR-SEARCH(B, C)
Step22: for each 'active' and 'not clustered' neighbor Bn of B do
Step23: C <- $B_n$
Step24: mark block $B_n$ clustered
Step25: NEIGHBOR-SEARCH($B_n$, C)
Step26: endfor
end NEIGHBOR-SEARCH
```

BANG

In the BANG-clustering system, the feature space into is partitioned into a hierarchical set of grid regions and each grid region is assigned a unique identity (r, l), where r is the region number and l is the level number. The blocks are sorted according to their density indices. Blocks with the highest density index become clustering centers and the remaining blocks are clustered iteratively in order of their density indices and the remaining blocks either build new cluster centers or merge with existing clusters.

```
BANG Algorithm
Step1: Partition the feature space into rectangular grids such that each grid contains up to a maximum of $p_{max}$
       data points.
Step2: Construct a binary tree to maintain the populated grids, in which the partition level according to the
       node depth in the tree.
Step3: Calculate the dendrogram in which the density indices of all regions are calculated and sorted in
       decreasing order.
Step4: Starting with the highest-density region (i.e., the first region), determine all the neighbors and classify
       them in decreasing order. BANG Clustering places the found  regions to the right of the original
       regions in the dendrogram.
Step5: Repeat step 4 for the remaining regions of the dendrogram.
```

WaveCluster

WaveCluster is an algorithm for clustering spatial data based on wavelet transforms. WaveCluster is insensitive to noise, capable of detecting clusters of arbitrary shape at different degrees of detail, and efficient for large databases.

```
WaveCluster Algorithm
Step1: Partitions the original data space into non-overlapping hyper-rectangles, i.e., cells.
Step2: The j^th dimension is segmented into $m_j$ of intervals.
Step3: Each cell $c_i$ is the intersection of one interval from each dimension and has the form $(c_{i1}, c_{i2}, \ldots, c_{id})$,
       where $c_{ij} = [l_{ij}, h_{ij})$ is the right open interval in the partitioning of the j^th dimension and d is the
       number of dimensions.
Step4: A point $x = (x_1, x_2, \ldots, x_d)$ is said to be contained in a cell $x_i$ if $l_{ij} \le x_j < h_{ij}$ for $j = 1, 2, \ldots, d$.
Step5: Let $c_i \cdot$ count denote the number of points contained in the cell $c_i$.
Step6: a wavelet transforms to $c_i \cdot$ count values.
Step7: transformed space is defined as the set of cells after the wavelet transformation on the count values of
       the cells in the quantized space.
```

STING –Statistical **In**formation **G**rid-based clustering method

STING is a query-independent approach since the statistical information exists independent of queries. The computational complexity of STING for cluster is O(K), and this is quite efficient in clustering large data sets especially when K << N.

```
STING Algorithm
Step1: Construct the grid hierarchical structure according to the database and generate the parameters of each
       cell.
Step2: Determine a layer to begin with
Step3: For each cell in this layer, compute the confidence interval of the probability that  this cell is relevant
       to the query
Step4:         if this layer is not the bottom layer then
Step5:                 Go to the next level in the hierarchy structure and go to step 3 for the relevant cells
                       of the higher-level layer
Step6:         else if the specification of the query is met then
Step7:                 Find the regions of relevant cells and return those regions that meet the
                       requirements of the query
Step8:         else
Step9:                 Reprocess the data in the relevant cells and return the results that meet the
                       requirements of the query
Step10: end if
```

CLIQUE

CLIQUE is a clustering algorithm that is able to identify dense clusters in subspaces of maximum dimensionality. The CLIQUE algorithm consists of three steps. In the first step, the subspaces that contain clusters are identified. In the second step, the clusters embedded in the subspaces identified in step 1 are found. Finally, a minimal description of each cluster is generated.

ENCLUS –ENtropy-based **CLUS**tering

ENCLUS is an entropy-based subspace clustering algorithm for clustering numerical data. It can find arbitrarily shaped clusters embedded in the subspaces of the original data space. It follows similar approaches suggested by CLIQUE, but does not make any assumptions about the cluster shapes and hence is capable of finding arbitrarily shaped clusters embedded in subspaces.

```
ENCLUS Algorithm
Input: D-Data set, ω-Entropy threshold, -Interest threshold
Step1: Let $l \Leftarrow 1$ and $C_1$ be one-dimensional subspaces
Step2: while $C_l = \emptyset$ do
Step3:          for all $c \in C_l$ do
Step4:                  Calculate the density $f_c(\cdot)$
Step5:                  Calculate the entropy $H(c)$ from $f_c(\cdot)$
Step6:                  if $H(c) < \omega$ then
Step7:                          if interest(c) > then
Step8:                                  $S_l \Leftarrow S_l \cup c$;
Step9:                          else
Step10:                         $NS_l \Leftarrow NS_l \cup c$
Step11:                 end if
Step12:         end if
Step13: end for
Step14: Generate $C_{l+1}$ from $NS_l$
Step15: $l \Leftarrow l + 1$
Step16: end while
Output: $\cup_l S_l$ as significant subspaces
```

MAFIA – **M**erging of **A**daptive **F**inite **I**ntervals

MAFIA is a parallel subspace clustering algorithm using adaptive computation of the finite intervals in each dimension that are merged to explore clusters embedded in subspaces of a high dimensional data set. It is also a density- and grid-based clustering algorithm.

```
MAFIA Algorithm
Input: Di – Domain of i^th attribute, N – Total number of data points in the data set, a – Size of the generic
       interval
Step1: for i = 1 to d do
Step2:          Divide $D_i$ into intervals of some small size x
Step3:          Compute the histogram for each interval in the i^th dimension, and set the value of the
                window to the maximum in the window
Step4:          From left to right, merge two adjacent intervals if they are within a threshold β
Step5:          if number of bins == 1 then
Step6:                  Divide the i^th dimension into a fixed number of equal intervals and set a threshold
                        β' for it
Step7:          end if
Step8: end for
```

OPTIGRID

The OptiGrid clustering algorithm is a very efficient algorithm for clustering high-dimensional databases with noise.

```
OptiGrid Algorithm
Input: data set D, q, min_cut_score
Step1: Determine a set of contracting projections $P = \{P_0, P_1, ..., P_k\}$ and calculate all the projections of the
       data set D: $P_i(D)$, i = 1, 2,...,k
Step2: Initialize a list of cutting planes BEST _CUT ⇐ ∅, CUT ⇐ ∅
Step3: for i = 0 to k do
Step4:          CUT ⇐ best local cuts $P_i(D)$
Step5:          CUT   SCORE ⇐ Score best local cuts $P_i(D)$
Step6:          Insert all the cutting planes with a score ≥ min_cut_score into BEST _CUT
Step7:          if BEST _CUT = ∅ then
Step8:                  Return D as a cluster
Step9:          else
Step10:         Select the q cutting planes of the highest score from BEST _CUT and construct a
                multidimensional grid G using the q cutting planes
Step11:         Insert all data points in D into G and determine the highly populated grid cells in G and add
                these cells to the set of clusters C
Step12:         Refine C
Step13:         for all clusters $C_i$ in C do
Step14:                 Do the same process with data set $C_i$
Step15:         end for
Step16: end if
Step17: end for
```

Variants of OptiGrid that were introduced to address the issues of the scalability of the grid structure, especially with respect to available memory,and a clear criterion for the selection of cutting planes. They are O-Cluster and CBF which is discussed in the below section.

O-cluster –Orthogonal partitioning **CLUSTER**ing

A O-cluster introduced to address three limitations of OptiGrid. One is scalability in terms of data relative to memory size, second, lack of clear criterion to determine if a cutting plane is optimal or not, and third one is sensitivity to threshold parameters for noise and cut plane density.

```
O-cluster Algorithm
Input: Load data buffer.
Step1: Compute histograms for active partitions.
Step2: Find “best” splits for active partitions.
Step3: Flag ambiguous and “frozen” partitions.
Step4: Split active partitions.
Step5: Reload buffer.
```

CBF –Cell-**B**ased **F**iltering

CBF focuses on the scalability of the grid structure, handling large data sets in memory, and the efficiency of insertion and retrieval of clusters from the grid structure. It also offers a clear criterion for a cutting plane.

```
CBF Algorithm
Input: Load Data.
Step1: Split each dimension into a set of partitions.
Step2: Optimal split section in each dimension.
Step3: Create cells from overlapping regions of the partitions in each dimensi
Step4: Insert the cells with higher density into the clusters in the index structu
Step5: Provide the cluster information file by grouping clusters into sections.
Step6: For each sections of cluster
Step7:          if density_of_section > threshold then
Step8:                  secondary_index = true
Step9:          else
Step10:                 secondary_index = false
```

PROCLUS – (**PRO**jected **CLUS**tering)

PROCLUS is a variation of the k-medoid algorithm in subspace clustering. The algorithm consists of three phases: the initialization phase, the iteration phase, and the refinement phase.

```
PROCLUS Algorithm
Input: D-Data set, k-Number of clusters, l-Average dimensions of cluster
Step1: Let A, B be constant integers
Step2: Draw a sample S of size A · k randomly
Step3: Let medoids set M ⇐ Greedy(S, B · k)
Step4: Let BestObjective ⇐ ∞ and $M_{curr}$ ⇐ random set of medoids $\{m_1, m_2, . . . , m_k\} \subset M$
Step5: repeat
Step6:          Let δi be the distance to the nearest medoid from $m_i$ for i = 1, 2 . . . , k
Step7:          Let $L_i$ be the set of points in a sphere centered at $m_i$ with radius $\delta_i$ for i = 1, 2, . . , k
Step8:          $(P_1, P_2, . . . , P_k)$ ⇐ FindDimensions(k, l, $L_1, L_2, . . . , L_k$)
Step9:          $(C_1, C_2, . . . , C_k)$ ⇐ AssignPoints($P_1, P_2, . . . , P_k$)
Step10: Objective ⇐ EvaluateClusters($C_1, . . . , C_k, P_1, . . . , P_k$)
Step11: if Objective < BestObjective then
Step12:         Let BestObjective ⇐ Objective, $M_{best}$ ⇐ $M_{curr}$
Step13:         Compute the bad medoids in $M_{best}$
Step14: end if
Step15: Compute $M_{curr}$ by replacing the bad medoids in $M_{best}$ with random points from M
Step16: until Stop criterion
Step17: $(P_1, P_2, . . . , P_k)$ ⇐ FindDimensions(k, l, $L_1, L_2, . . . , L_k$)
Step18: $(C_1, C_2, . . . , C_k)$ ⇐ AssignPoints($P_1, P_2, . . . , P_k$)
Step19: Return $M_{best}, P_1, P_2, . . . , P_k$
```

ORCLUS –ORiented projected **CLUS**ter generation

ORCLUS is an extension of PROCLUS. It diagonalizes the covariance matrix of each cluster and finds information about projection subspaces from the diagonalization of the covariance matrix.

```
ORCLUS Algorithm
Input: D-Data set, k-Number of clusters, and l-Number of dimensions
Step1: Pick $k_0 > k$ initial data points from D and denote them by $S = \{s_1, s_2, . . . , s_{k0}\}$
Step2: Set $k_c \Leftarrow k_0$ and $l_c \Leftarrow d$
Step3: For each i, set $P_i$ to be the original axis system
Step4: Set α ⇐ 0.5 and compute β
Step5: while $k_c > k$ do
Step6:          $(s1, . . . , s_{kc}, C_1, . . . , C_{kc})$ = Assign($s_1, . . . , s_{kc}, P_1, . . . , P_{kc}$) {find the partitioning induced
                by the seeds}
Step7:          Set $k_{new} \Leftarrow \max\{k, k_c \cdot \alpha\}$ and $l_{new} \Leftarrow \max\{l, l_c \cdot \beta\}$
Step8:          $(S_1, . . . , S_{knew}, C_1, . . . , C_{knew}, P_1, . . . , P_{knew})$ = Merge($C_1, . . . , C_{kc}, k_{new}, l_{new}$)
Step9:          Set $k_c \Leftarrow k_{new}$ and $l_c \Leftarrow l_{new}$
Step10: end while
Step11: for i = 1 to k do
Step12: $P_i$ = FindVectors($C_i$, l)
Step13: end for
Step14: $(s_1,...,s_k,C_1,...,C_k)$ = Assign($s_1,...,s_k,P_1,...,P_k$)
Step15: Output $(C_1,...,C_k)$
```

GDILC –Grid-based **D**ensity-**I**so**L**ine **C**lustering

GDILC is capable of eliminating outliers and finding clusters of various shapes. The distribution of data samples can be depicted very well by the so-called density-isoline figure. A grid-based method is employed to calculate the density of each data sample and find relatively dense regions.

```
GDILC Algorithm
Step1: (Initializing cells) Divide each dimension into m intervals
Step2: (Computing distance threshold RT) For each point x, compute the distances between x and every
       point in the neighbor cells of Cx. Then compute the distance threshold RT by calculating the average
       distance from those distances.
Step3: (Computing density vector and density threshold DT) For each point x, compute the density of x by
       counting the number of points within RT of x. Then compute the density threshold DT by calculating
       the average density from the density vector.
Step4: (Clustering)At first, take each point whose density is more than DT as a cluster. Then, for each point x,
       check whether the distance from x of each point whose density is more than DT in the neighbor cells
       of Cx is less than RT. If so, then merge the two clusters containing those two points. Continue the
       above merging process until all point pairs have been checked.
Step5: (Removing outliers) Remove those clusters whose sizes are less than a certain number.
```

Table 4. Computational Complexity and Scalability Analysis on Grid based Clustering Algorithms

Algorithms	Computational Complexity	Scalability
GRIDCLUS	Time Complexity: $O(n^2)$	Y
BANG	Time Complexity: O(n)	Y
WaveCluster	Time Complexity: O(n)	Y
STING	Time Complexity: O(n)	N
CLIQUE	Time Complexity: O(n+kˆ2)	Y
ENCLUS	Time Complexity: O(n)	Y
MAFIA	Time Complexity: $O(m^p+pN)$ Space Complexity: O (mp+pN)	Y
OPTIGRID	Time Complexity: $O(d \cdot N \cdot \log N)$	Y
O-cluster	Time Complexity: O(Nd)	Y
CBF	Time Complexity: O(N)	Y
PROCLUS	Time Complexity: O(nkl)	Y
ORCLUS	Time Complexity: $O(k_0^3 + k_0nd + k_0^2d)$	Y
GDILC	Time Complexity: O(n)	N

E. Model Based Clustering

Gaussian (EM)

This algorithm assumes apriori that there are 'n' Gaussian and then algorithm try to fits the data into the 'n' Gaussian by expecting the classes of all data point and then maximizing the maximum likelihood of Gaussian centers.

```
Gaussian(EM) Algorithm
Input: Let X = {x1, x2, x3, ..., xn} be the set of data points
       V = {μ1, μ2, μ3, ..., μc} be the set of means of Gaussian
       P = {p1, p2, p3, ..., pc} be the set of probability of occurrence of each Gaussian
Step1: On the i^th iteration initialize:
       $\lambda_t = \{\mu_1(t), \mu_2(t) \dots \mu_c(t), \Sigma_1(t), \Sigma_2(t) \dots \Sigma_c(t), p_1(t), p_2(t) \dots p_c(t)\}$
Step2: E-Step - Compute the "expected" classes of all data points for each class using:
       $P(w_i|x_k,\lambda_t) = \frac{p(x_k|w_i,\lambda_t)P(w_i|\lambda_t)}{p(x_k|\lambda_t)} = \frac{p(x_k|w_i,\mu_i(t),\Sigma_i(t))p_i(t)}{\sum_{j=1}^{c} p(x_k|w_j,\mu_j(t),\Sigma_j(t))p_j(t)}$
Step3: M-Step - Compute "maximum likelihood μ" given our data class membership distribution usin
       $\mu_i(t+1) = \frac{\sum_k P(w_i|x_k,\lambda_t)x_k}{\sum_k P(w_i|x_k,\lambda_t)}$    $p_i(t+1) = \frac{\sum_k P(w_i|x_k,\lambda_t)}{R}$
```

COOLCAT

COOLCAT algorithm is proposed to cluster categorical attributes using entropy. Given a data set D of N data points pˆ1, pˆ2, . . . ,pˆN, where each point is a multidimensional vector of d categorical attributes, i.e., pˆj = $\{p_j^1, p_j^2, \dots, p_j^d\}$, the goal of this algorithm is to minimize the entropy of the whole arrangement.

```
COOLCAT Algorithm
Step1: Draw a sample data set S (|S| << N) from the entire data set, where N is the size of
       the entire data set
       {1. Initialization phase}
Step2: Find the k most "dissimilar" records from the sample data set by maximizing the minimum pairwise
       entropy of the chosen data points {To do this, it first finds the two data points ps1, ps2 such that the
       entropy E(ps1, ps2) is maximized, i.e., E(ps1, ps2) ≥ E(p1, p2) ∀p1, p2 ∈ S, and then puts them in two
       separate clusters C1, C2 and marks the two data points; after it selects j − 1 points, it will find the j^th
       point such that
           $\min_{i=1,2,\dots,j-1}(E(p_{s_i}, p_{s_j}))$
       is maximized and then put this point in the j^th cluster Cj and mark this point.}
       {2. Incremental phase}
Step3: Process the unmarked |S| − k data points in the sample data set S and the remaining data points (i.e.,
       data points outside the sample). Given the k initial sets of clusters found in the first step, bring a
       batch of data $\check{C} = C_1, C_2, \dots, C_k$, points to the memory from disk and, for each data point p in
       this batch of data points, place p in cluster Ci and compute $\bar{E}(\check{C}^i)$, where C ˇ i denotes the cluster
       obtained by placing p in Ci, and then find the index j such that
           $\bar{E}(\check{C}^j) \le \bar{E}(\check{C}^i) \quad \forall i = 1, 2, \dots, k$
       and place p in cluster Cj. The above procedure is kept executing until all points have been assigned
       to some cluster.
```

STUCCO

In the algorithm STUCCO, a new concept of contrast-sets is defined in order to find the contrast-sets whose supports differ meaningfully among different groups. To do that, a method of tree searching is used to calculate all possible combinations of attribute values. One then retains the significant contrast-sets, post processes those contrast sets, and then selects a subset.

```
STUCCO Algorithm
Step1: Using canonical ordering of attributes to construct a search tree, scan the database, count the support
       for each group, and retain the significant contrast-sets.
Step2: Test the null hypothesis that contrast-set support is equal across all groups or contrast set support is
       independent of group membership to check whether a contrast-set is significant.
Step3: Prune the nodes that can never be significant contrast-sets.
Step4: Find surprising contrast-sets.
```

SOM –Self-**O**rganizing **M**ap

SOM is used for clustering data without knowing the class memberships of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM, the Self-Organizing Feature Map

```
SOM Algorithm
Step1: Select output layer network topology – Initialize current neighborhood distance, D(0), to a positive
       value
Step2: Initialize weights from inputs to outputs to small random values
Step3: Let t = 1
Step4: While computational bounds are not exceeded do
Step5:         Select an input sample li
Step6:         Compute the square of the Euclidean distance of li from weight vectors (wj) associated with
               each output node
                   $\sum_{k=1}^{n}(i_{l,k} - w_{j,k}(t))^2$
Step7:         Select output node j* that has weight vector with minimum value from step 6)
Step8:         Update weights to all nodes within a topological distance given by D(t) from j*, using the
               weight update rule:
                   $w_j(t+1) = w_j(t) + \eta(t)(i_l - w_j(t))$
Step9:         Increment t
Step10: Endwhile Learning rate generally decreases with time:
               $0 < \eta(t) \le \eta(t-1) \le 1$5
```

Table 5. Computational Complexity and Scalability Analysis on Model based Clustering Algorithms

Algorithms	Computational Complexity	Scalability
Gaussian(EM)	Time Complexity: $O(n^2)$	N
COOLCAT	Time Complexity: O(n)	Y
STUCCO	Time Complexity: O(n)	N
SOM	Time Complexity: O(l)	Y

IV. CONCLUSION

The purpose of survey paper is to present a comprehensive view of different clustering algorithms available for high dimensional data. Clustering high dimensional data sets is apervasive task. The enormous growth in data in every domain, there is great growth in high dimensional data spaces. This study focuses on various algorithms available for high dimensional data. Computational complexity and scalability are examined thoroughly for all clustering algorithms. There are many potential applications like bioinformatics, text mining with high dimensional data where subspace clustering, projected clustering approaches could help to uncover patterns missed by state-of-art clustering approaches. The major challenge for clustering high dimensional data is to overcome the “curse of dimensionality” [15]. This survey will be helpful to choose the right clustering algorithm for different applications.

REFERENCES

[1]. Jain A, Dubes R, Algorithms for clustering data, Prentice-Hall, Inc, Upper Saddle River, 2011.

[2]. Zhang T, Ramakrishnan R, Linvy M, Birch: an efficient data clustering method for large databases. In Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, 1996.

[3]. Qinbao Song, Jingjie Ni, Guangtao Wang, A Fast Clustering-Based Feature Subset Selection Algorithm for High-Dimensional Data, IEEE Transactions on Knowledge and Data Engineering, Vol. 25, No. 1, 2013.

[4]. Dongkuan Xu, Yingjie Tian, A Comprehensive Survey of Clustering Algorithms, © Springer-Verlag Berlin Heidelberg, 2015.

[5]. Ng R, Han J, Clarans: a method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering 14:1003–1016, 2002.

[6]. Karypis G, Han E, Kumar V, Chameleon: hierarchical clustering using dynamic modelling. Computer 32:68–75, 1999.

[7]. Cao F, Ester M, Qian W, Zhou A, Density-based clustering over an evolving data stream with noise. SDM 6:328–339, 2006.

[8]. Aggarwal C, Yu P, Redefining clustering for high-dimensional applications.IEEE Transactions on Knowledge and Data Engineering, 14(2):210–225, 2002.

[9]. Zhao Y, Song J, GDILC: A grid-based density-isoline clustering algorithm. In Proceedings of the international conferences on info-tech and info-net, volume 3, pages 140–145. Beijing: IEEE, 2001.

[10]. EverittB, Landau S,Leese M, Cluster Analysis, 4th edition. New York:Oxford University Press, 2001.

[11]. GholamhoseinSheikholeslami, Surojit Chatterjee, Aidong Zhang, Wavecluster:A multi-resolution clustering approach for very large spatial databases. In VLDB’98,pages 428–439, 1998.

[12]. Nagesh H, Goil S and Choudhary A, MAFIA: Efficient and scalable subspace clustering for very large data sets, 1999.

[13]. Chaoqun Ma,GuojunGan,Jianhong Wu, Data Clustering: Theory, Algorithms,and Applications, SIAM, 2007.

[14]. Chun-Hung Cheng, ENCLUS: Entropy-based Subspace Clustering for Mining Numerical Data, 1999.

[15]. Jain A,Murty M N, Flynn P J, Data Clustering: A Review, ACM Computing Surveys, Volume 31(3), pp. 264-323, 2011.

[16]. Guha S, Rastogi R, Shim K, CURE: An efficient clustering algorithm for large databases, Proc. Of ACM SIGMOD Conference, 2012.

[17]. Rui Xu, W. Donald, Survey of Clustering Algorithms, IEEE Transaction on Neural Network, vol. 16, 2009.

[18]. Fraley C, Raftery A, Model-based clustering, discriminant analysis and density estimation. Journal of American Statistical Association, 97, 611–631, 2002.

[19]. Fraley C, Algorithms for model-based Gaussian hierarchical clustering. SIAM Journal on Scientific Computing 20, 270–281, 1998.

[20]. Murty, M N, Krishna, G, A computationally efficient technique for data clustering. Pattern Recogn. 12, 153–158, 1980.

[21]. Song Q, Jingjie Ni, Wang G, A Fast Clustering-Based Feature Subset Selection Algorithm for High Dimensional Data, IEEE Transactions On Knowledge and Data Engineering Vol 25 No:1, 2013.