# A review and implementation of some approaches to metric clustering

Wu Zhou

Dec 14, 2004

# Contents

# 1 Introduction

Given so many databases which are used in the real world and a huge amount of information which is generated and communicated through the Internet every day, it is easy to image that we are living in a digital world. This explosive growth in data and databases has dramatically increased the demand for data mining. Data mining focuses on using efficient techniques and tools to discover useful information and knowledge from data. Among various data mining techniques, cluster analysis or unsupervised learning is an interesting and fast growing topic.

Cluster analysis methods try to find groups or clusters of data objects in the data set. A cluster is a maximal collection of similar data objects so that the purpose of a clustering method is to assign data objects similar to one another to the same cluster and objects dissimilar to different clusters.

Existing clustering methods fall into five general categories: partitioning methods, hierarchical methods, density-based methods, grid-based methods and model-based methods. My research work focuses on the first three categories.

Generally, there are two types of attributes involved in the data to be clustered: metric and nonmetric. If all the data attributes are metric, a data object can be represented by a vector in the metric space. A metric space is a set S with a global distance function (the metric g) that, for every two points x, y in S, gives the distance between them as a nonnegative real number $g(x, y)$. A metric space must also satisfy:

1. $g(x, y) = 0 \quad iff \quad x = y$
2. $g(x, y) = g(y, x)$
3. The triangle inequality: $g(x, y) + g(y, z) \geq g(x, z)$

In this paper, I am interested in the clustering approaches to group data in metric spaces. In a metric space, the elements in a data vector can be formed by the value of each attribute. If the number of attributes in a data set is $m$, the dimension of the data vectors is also $m$. We can say the data come from the m-dimension metric space. Besides, the data set can be represented by an $n$ by $m$ matrix, if the number of objects in the data set is $n$.

In many clustering problems, we do not have metric data attributes. For example, we only have the dissimilarities between data objects. The dissimilarity between two data objects can be metric or nonmetric. To obtain data in the metric space from these dissimilarities, a possible solution is multidimensional scaling (MDS). Besides, MDS can be used to transfer data from a higher dimensional metric space, say $m$-dimension, to a lower dimensional metric space, say $p$-dimension, where $p < m$.

This paper is organized as follows: In Section 2, I discuss MDS, in Section 3, I discuss some approaches to partitioning, hierarchical, and density-based clustering. Section 4 gives the conclusion. In Appendix, I give the codes to implement some methods in MDS and density based clustering.

# 2 Multidimensional scaling

Multidimensional scaling (MDS) was proposed to solve the problem of finding a proper configuration of points given some measurements of dissimilarities between objects. Given a set of objects and a function $\delta_{rs}$ which gives the dissimilarity between objects r and s in the set, the goal of multidimensional scaling is to find coordinates $\mathbf{x_r}$ and $\mathbf{x_s}$ in the metric space such that, one of the following two conditions must be satisfied.

- If $\delta_{rs}$ satisfies the triangle inequality, the Euclidean distances $d_{rs}$ be-

tween these coordinates match or nearly match the original dissimilarities. This is the metric MDS.

- If $\delta_{rs}$ is an unknown monotonic increasing function $\delta_{rs} = f(d_{rs})$, Where $d_{rs}$ is the Euclidean distance between objects $r$ and $s$. The rank order of Euclidean distances $d^*_{rs}$ between objects $r$ and $s$ in the new configuration match the original rank order of dissimilarities $\delta_{rs}$, no matter $\delta_{rs}$ satisfies the triangle inequality or not. This is the nonmetric MDS.

Since MDS can find a new configuration of points in the lower dimensional space with only a little loss of information, it could be employed in the applications which require lower dimensional data sets.

## 2.1 Metric multimensional scaling

The purpose of the metric MDS is to find a new configuration (or coordinates) probably in a low dimensional space, such that the Euclidean distance of any pair of the new coordinates closely approximates the prescribed value. For example, how can we draw a map of Canada if we only know the distances between all pairs of Canadian cities?

To complete the metric MDS, a principal coordinates analysis is employed first to find a new configuration from the given dissimilarity matrix. Then, a least squares scaling is applied afterwards to minimize the disparities between the original data's dissimilarities and the new configuration's dissimilarities.

### 2.1.1 Principal coordinates analysis (classical MDS)

Let $f = (x_{rj})$ be the actual coordinates of $n$ points in the $m$ dimensional space, where $r = 1, 2, \cdots, n, j = 1, 2, \cdots, m$. Let $D = (d_{rs})$ to be the $n \times n$

matrix of Euclidean distances between each pair of points. If define $D_{rs} = d_{rs}^2$, then

$$D_{rs} = \sum_{j=1}^{m} (x_{rj} - x_{sj})^2. \tag{1}$$

In order to find $X$ from $D$, define another $n \times n$ matrix $B = (b_{rs})$, where $B = XX^T$. Therefore,

$$b_{rs} = \sum_{j=1}^{m} x_{rj} x_{sj}. \tag{2}$$

Since we can write $D$ in terms of $B$,

$$D_{rs} = b_{rr} + b_{ss} - 2b_{rs}, \tag{3}$$

$X$ can be solved by finding $B$ from $D$ first, then factoring it in the form $B = XX^T$.

There are many more unknowns than equations which relate them. To obtain the unique solution when finding $B$ from $D$, one possible way is to add $m$ location constraints, which are,

$$\sum_{r=1}^{n} x_{rj} = 0 \quad \forall j. \tag{4}$$

Since the rows and columns of $B$ add up to 0, we have,

$$\sum_{r=1}^{n} D_{rs} = tr(B) + nb_{ss},$$

$$\sum_{s=1}^{n} D_{rs} = nb_{rr} + tr(B),$$

$$\sum_{r=1}^{n}\sum_{s=1}^{n} D_{rs} = 2n\ tr(B).$$

Let,

$$\overline{D}_{r+} = \frac{\sum_{r=1}^{n} D_{rs}}{n},$$
$$\overline{D}_{+s} = \frac{\sum_{s=1}^{n} D_{rs}}{n},$$
$$\overline{D}_{++} = \frac{\sum_{r=1}^{n}\sum_{s=1}^{n} D_{rs}}{n^2}.$$

Now, we can derive $B$ from $D$ as follows,

$$\begin{aligned} b_{rs} &= -\frac{1}{2}D_{rs} + \frac{1}{2}b_{rr} + \frac{1}{2}b_{ss} \\ &= -\frac{1}{2}D_{rs} + \frac{1}{2}\Big(\overline{D}_{r+} - \frac{tr(B)}{n}\Big) + \frac{1}{2}\Big(\overline{D}_{+s} - \frac{tr(B)}{n}\Big) \\ &= -\frac{1}{2}\Big(D_{rs} - \overline{D}_{r+} - \overline{D}_{+s} + \overline{D}_{++}\Big). \end{aligned} \tag{5}$$

To simplify calculation, we define an $n \times n$ matrix $C = (c_{rs})$, where $c_{rs} = -\frac{1}{2}D_{rs}$. Now we have,

$$b_{rs} = c_{rs} - \overline{c}_{r+} - \overline{c}_{+s} + \overline{c}_{++} \tag{6}$$

Therefore, $B$ can be derived from $C$ by *double centring* as

$$B = (I - n^{-1}\mathbf{1}\mathbf{1}^T)C(I - n^{-1}\mathbf{1}\mathbf{1}^T), \tag{7}$$

where $\mathbf{1}$ is an $n \times 1$ matrix all of whose elements are 1.

Since $d_{rs}$ is the Euclidean distance between object $r$ and $s$, $B$ can be shown to be a positive semi-definite matrix [1, Theorem 5.7], it is possible to obtain the coordinate matrix $X$ from $B$ by applying an eigenvector analysis. If the rank of $B$ is $p$, then $B$ has $p$ positive eigenvalues, which are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0$. Let $\mathbf{e_i}$ be the corresponding eigenvector of $\lambda_i$, a possible coordinate matrix $X$ is: $X = (\sqrt{\lambda_1}e_1, \sqrt{\lambda_2}e_2, \cdots, \sqrt{\lambda_p}e_p)$.

The dimension of the resulting coordinate matrix $X$ could be further reduced to k ($k < p$) by selecting the first k biggest eigenvalues to form $X$. The accuracy of such reduction can be measured by an *agreement measure* $T$, where $T = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$. Finding a new configuration by principal coordinates analysis is called classical multidimensional scaling as well.

### 2.1.2 Least squares scaling

After we derive the new configuration by principal coordinates analysis, the disparities between the original data's dissimilarities and the new configuration's dissimilarities should be minimized. The problem can be described as follows.

Given the matrix $D = (d_{rs})$, which describes the *distances* between $n$ points in $p$ dimensions, we want to approximate $D$ as closely as possible by the *distances* between $n$ points in $k$ dimensions, where $k \leq p$. Define the matrix $D^* = (d^*_{rs})$ to describe the approximated *distances* between $n$ points in $k$ dimensions; the problem is to find a matrix $D^*$ such that the difference between the approximatad distances $d^*_{rs}$ and the given distances $d_{rs}$ is minimized. One possible way to measure such difference is to use a least squares stress S, which is

$$S = \sum_{r=1}^{n} \sum_{s=1}^{n} (d_{rs} - d^*_{rs})^2. \tag{8}$$

To minimize S, the solution can be found by

$$\frac{\partial S}{\partial x_{rj}} = 0 \quad \forall r, j. \tag{9}$$

Where,

$$(d^*_{rs})^2 = \sum_j (x_{rj} - x_{sj})^2. \tag{10}$$

By the chain rule,

$$\frac{\partial S}{\partial x_{rj}} = \sum_s \left(\frac{\partial S}{\partial d^*_{rs}} \frac{\partial d^*_{rs}}{\partial x_{rj}}\right).$$

Since,

$$-\frac{1}{2}\frac{\partial S}{\partial d^*_{rs}} = (d_{rs} - d^*_{rs}),$$

and

$$\frac{\partial d^*_{rs}}{\partial x_{rj}} = \frac{x_{rj} - x_{sj}}{d^*_{rs}},$$

the solution satisfies the following equation,

$$\sum_s (d_{rs} - d^*_{rs})\left(\frac{x_{rj} - x_{sj}}{d^*_{rs}}\right) = 0 \quad \forall r, j. \tag{11}$$

We can simplify the above equation by defining an $n \times n$ matrix $F$, such that

$$f_{rs} = \frac{d_{rs} - d^*_{rs}}{d^*_{rs}} \quad (r \neq s),$$

$$f_{rr} = -\sum_{r \neq s} \left(\frac{d_{rs} - d^*_{rs}}{d^*_{rs}}\right).$$

Now, we can rewrite (11) as,

$$FX = 0. \tag{12}$$

We now define another $n \times n$ matrix $F^* = (f^*_{rs})$, to derive an iterative equation for finding the solution. Let

$$f^*_{rs} = f_{rs} + 1 \quad (r \neq s),$$

$$f^*_{rr} = f_{rr} - (n - 1).$$

The relation between $F$ and $F^*$ is,

$$F = F^* + (n\mathbf{I} - \mathbf{1}\mathbf{1}^T).$$

$F^*$ is a symmetric matrix whose row and column sums are zero. If $X$ is also in a centred form such that its column sums are zero, then,

$$\mathbf{1}^T X = \mathbf{0}.$$

Therefore, the (12) can be written in terms of $F^*$ as follows,

$$-\frac{1}{n} F^* X = X. \tag{13}$$

Which suggests an iterative update equation,

$$-\frac{1}{n} F_i^* X_i = X_{i+1}. \tag{14}$$

Usually, the initial configuration $X_1$ is the column centred form of the coordinate matrix calculated by the classical MDS. It has been shown that the value of the *stress* $S$ never increases during the above iterations and that the sequence will converge to a solution under normal circumstances [2].

### 2.1.3 Algorithm

Normally, two phases are included in the metric MDS.

**Classical scaling phase:** As described in the sub section 2.1.1, find the symmetric matrix $B$ first, then obtain the new configuration $X$ by the eigenvector analysis of $B$.

**Least squares scaling phase:** As described in the sub section 2.1.2. After obtaining the new configuration by the classical scaling phase, we can employ (14) iteratively to obtain a new configuration which makes the *stress* less than a predefined value. The dimension of the resulting configuration is the same as that of the initial configuration obtained by the classical scaling phase.

The detailed algorithm is as follows:

1. Form the matrix $C$ from $D$,
2. Get $B$ by double centering $C$,
3. Take the eigen-decomposition of $B$,
4. Choose the first $k$ biggest non-zero eigenvalues of $B$, use these eigenvalues and the corresponding eigenvectors to form $X_0$,
5. Get $F_0^*$,
6. Iterate (14) until converge.

## 2.2 Non-metric Multidimensional Scaling

In metric MDS, the quantity of dissimilarity is derived out of Euclidean distance directly. It is not suitable when people are concerned more about preserving the rank order of the dissimilarities than the actual numerical values of these dissimilarities. In such cases, the restrictions from the metric inequality may be violated. Non-metric MDS was proposed to solve this problem.

The purpose of non-metric MDS is to obtain new coordinates for a set ef data objects in the lower dimensional space given the dissimilarities between these objects, such that the rank order of dissimilarities is preserved.

In non-metric MDS, one common approach is proposed by Shephard (1962) and Kruskal (1964) [3, 4]. The approach employs both the pooled-adjacent-violator algorithm (PAV) to achieve the monotonicity and the steepest descent procedure to obtain the new configuration of points.

### 2.2.1 Ensuring monotonicity

Given a set of two dimensional data $\{(x_i, y_i)\}_{i=1}^n$, sort them by the value of each $x_i$ to obtain the ordered data set $\{(x_{(i)}, y_{(i)})\}_{i=1}^n$. The problem is to find the estimated value $\widehat{y}_{(i)}$ for each $y_{(i)}$ such that $\sum_{i=1}^n (y_{(i)} - \widehat{y}_{(i)})^2$ is minimized subject to the monotone requirement that $\widehat{y}_{(1)} \leq \widehat{y}_{(2)} \leq \cdots \leq \widehat{y}_{(n)}$. This is called the monotone regression. The pooled-adjacent-violator (PAV) algorithm was proposed to solve such problem. The process of PAV is shown below.

Start from $y_{(1)}$, loop in series from $y_{(1)}$ to $y_{(n)}$. If any pair of adjacent values $(y_{(i)}, y_{(i+1)})$ violates the monotonicity restriction, do the following 3 steps.

- Pool $y_{(i)}$ and $y_{(i+1)}$ by replacing both of them by their average.
- Go backwards, check if $y_{(i-1)}$ and the pooled $y_{(i)}$ obey the monotone requirement, if not, pool $y_{(i-1)}$, $y_{(i)}$ and $y_{(i+1)}$ into one average.
- Continue to the left until the monotonicity requirement is satisfied. Proceed to the right.

The PAV algorithm could be used in the non-metric MDS. Given the dissimilarity of the $i^{th}$ pair of the data objects, which is $\delta_i$, we obtain the configu-

ration of these data objects by the metric MDS, then calculate the distance between the two objects in the $i^{th}$ pair of the data objects, which is $d_i$. Now we have a set of two dimensional data $\{(\delta_i, d_i)\}_{i=1}^{t}$, where $t = n(n-1)/2$, $n$ is the number of objects in the data set. We sort them by the value of each $\delta_i$ to obtain the ordered date set $\{(\delta_{(i)}, d_{(i)})\}_{i=1}^{t}$. To preserve the rank order of the given dissimilarities, we use the above PAV process. Then, we obtain a new ordered data set $\{(\delta_{(i)}, \hat{d}_{(i)})\}_{i=1}^{t}$, which satisfies the requirement of monotonicity.

### 2.2.2 Steepest Descent Procedure

The steepest descent procedure is used to find the nearest local minimum of a function $y = g(x)$ provided the gradient of this function can be calculated. This procedure starts with the initial point $y_0 = g(x_0)$, move downhill gradually along the curve of the function in the direction of $-\nabla g(x_0)$, which is the local downhill gradient, it is calculated by $-g'(x_0)$. Therefore, the new value can be found by

$$x_{i+1} = x_i - \epsilon g'(x_i). \tag{15}$$

In the non-metric MDS, after obtained the ordered data set $\{(\delta_{(i)}, \hat{d}_{(i)})\}_{i=1}^{t}$ by the PAV process, we want to obtain a new configuration of these data such that the disparity of distances $d_i^*$ from the new configuration and the estimated values $\hat{d}_i$ is minimized. We use the *stress* $S$ to measure such disparity. $S$ is computed by $S = \sum_{r=1}^{n} \sum_{s=1}^{n} (\hat{d}_{rs} - d_{rs}^*)^2$, where $r$ and $s$ are two objects in the $i^{th}$ pair of objects in the data set. The new coordinates could be calculated using the steepest descent procedure. In the above *stress* function, $\hat{d}_{rs}$ is given and fixed, the value of $d_{rs}^*$ will be updated by iterations to find the local minimum of the *stress* $S$. The steepest descent procedure in the nonmetric MDS is as follows.

We have,

$$S = \sum_{r=1}^{n} \sum_{s=1}^{n} (\hat{d}_{rs} - d^*_{rs})^2,$$

where,

$$d^*_{rs} = \sqrt{\sum_{j=1}^{p} (x_{rj} - x_{sj})^2},$$

and $p$ is the dimension of the coordinates. By the chain rule,

$$\frac{\partial S}{\partial x_{rj}} = \sum_{s=1}^{n} (\frac{\partial S}{\partial d^*_{rs}} \frac{\partial d^*_{rs}}{\partial x_{rj}}) = \alpha \sum_{s=1}^{n} \Big( (1 - \frac{\hat{d}_{rs}}{d^*_{rs}})(x_{rj} - x_{sj}) \Big),$$

where $\alpha$ is a constant. Applying the steepest descent procedure, we have the following step function,

$$x^{(i+1)}_{rj} = x^{(i)}_{rj} - \epsilon \sum_{s=1}^{n} \Big( (1 - \frac{\hat{d}_{rs}}{d^{*(i)}_{rs}})(x^{(i)}_{rj} - x^{(i)}_{sj}) \Big). \tag{16}$$

This step function can be used to obtain new coordinates in the non-metric MDS.

### 2.2.3 Algorithm

Kruskal and Shephard's algorithm for the non-metric MDS contains three phases.

**Initial phase:** Get the matrix $D$ from the given dissimilarities $\delta_i$ of any pair of objects in the data set. Use the classical MDS to derive the

| | Pistons | Rocket | Clippers | Raptors |
|---|---|---|---|---|
| Pistons | - | 93-72 | 99-96 | 89-101 |
| Rocket | 72-93 | - | 91-86 | 88-95 |
| Clippers | 96-99 | 86-91 | - | 101-89 |
| Raptors | 101-89 | 95-88 | 89-101 | - |

| | Pistons | Rocket | Clippers | Raptors |
|---|---|---|---|---|
| Pistons | 0 | 21 | 3 | 12 |
| Rocket | 21 | 0 | 5 | 7 |
| Clippers | 3 | 5 | 0 | 12 |
| Raptors | 12 | 7 | 12 | 0 |

Figure 1: Dissimilarity Matrix of the NBA team example

start coordinates $X_0$ in the required lower dimensional space, say $p$ dimensional space.

**Non-metric phase:** Use the PAV algorithm to derive any estimated distance $\hat{d}_i$ from the current coordinates such that the required monotonicity is satisfied.

**Metric phase:** Use the step function (16) to obtain the new coordinates such that the stress $S$ is very small.

The algorithm starts from the initial phase, then, iteratively do the non-metric phase and the metric phase until the required rank order of dissimilarities is satisfied.

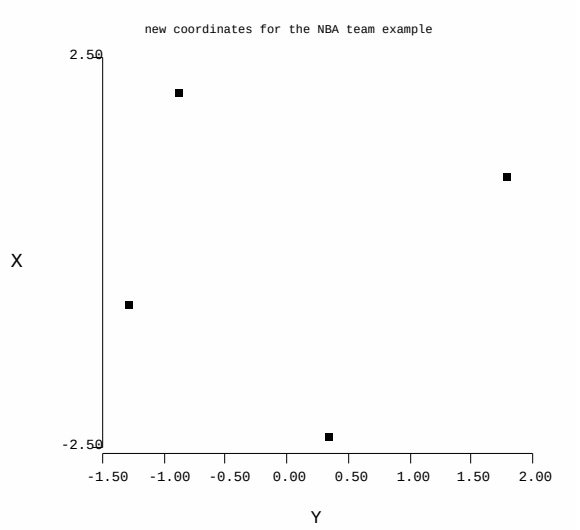


Figure 2: New configuration for the NBA team example

The following example is a simple application of non-metric MDS. The left table in Figure 1 shows the scores of each match among four NBA teams. The dissimilarities are calculated by the difference in the scores of two teams in a match. The dissimilarities among these four teams are shown in the right table in Figure 1. Some of the dissimilarities do not satisfy the triangle inequality. Now we want the configuration of these four teams in the 2-dimensional metric space. Non-metric MDS can do the work. Figure 2 shows the result obtained by the codes I wrote to implement the non-metric MDS. The rank order of these dissimilarities is preserved in the reproduced distances in the new configuration.

# 3 Clustering methods

Human beings have a good sense of grouping. We can distinguish blue from green, circle from triangle, flower from fruit. We group things naturally, by

comparing objects on the features which are the most outstanding to us, such as, colour, size, shape, weight and so on. In a natural grouping, we are using dissimilarities to recognize one group from the other. Most of the proposed clustering algorithms, no matter simple or complicated, employ the similar way to find clusters. In clustering methods, the dissimilarities can be measured by some metric functions like the Euclidean distance.

The remainder of this section discusses three major categories of clustering methods: partitioning methods, hierarchical methods, and the density-based clustering methods.

## 3.1 Partitioning methods

Partitioning methods appeared early in the history of clustering methods, long before the emergence of data mining. The idea is as follows: Separate objects into a fixed number of clusters such that the total deviation of each object from its cluster centre is minimized, therefore, the objects in the same cluster are close to each other, whereas, the objects in different clusters are far away from each other.

### 3.1.1 Clusters represented by their means

The k-means is a typical partitioning clustering method. In k-means each cluster is represented by its mean, which is the average of the data vectors in a cluster [5].

The k-means tries to minimize $E$, which is the average dissimilarities from any object in the data set to the centre of its cluster.

The algorithm of K-means:

1. The value of $k$ is given by the user.
2. Randomly choose $k$ objects from the data set to be the cluster centres at the initial state.
3. Assign each of the remaining objects to a cluster where centre is closest.
4. Recalculate the cluster mean to be the centre for each cluster.
5. Repeat steps 3 and 4 until no change happens.

Although k-means often finds the local optimum, it works well when the objects within each cluster are quite close to each other while the objects from different clusters are far away. The computational complexity of k-means is $O(nkt)$, where $n$ is the number of objects in the data set, $k$ is the number of clusters and $t$ is the number of iterations for the algorithm to converge. It is efficient if $k << n$ and $t << n$. Since k-means uses the mean of all the data vectors in a cluster to represent the cluster, the drawback is that, the clustering result is easy to be affected by the outliers in the data set, and it does not work well when the clusters are not sphere-shaped.

### 3.1.2 Clusters represented by their medoids

Similar with k-means, PAM (Partitioning Around Medoids) constructs clusters by using a single object to be the representative of a cluster [6]. In PAM, a medoid is used to represent a cluster. Given a cluster $C$, we define $d(O_p, O_c)$ to be the distance between an object $O_p$ in $C$ and the cluster centre $O_c$, which is the vector mean of all the objects in $C$. The object $O_p$ is a medoid of $C$ only if $d(O_p, O_c) = min_{\forall O_q \in C} d(O_q, O_c)$.

To find the medoids for $k$ clusters, where $k$ is given by the user, PAM begins with an arbitrary selection of $k$ objects, these objects are candidate medoids.

Then, at each step, PAM swaps a candidate medoid with an unselected object as long as such a swap would decrease $E$, which is the average dissimilarities from any object in the data set to the medoid of its cluster. In PAM, the difference of $E$ caused by a swap is called the cost of the swap, it is denoted by $S_{swap} = E_{beforeswap} - E_{afterswap}$.

Suppose $m$ is the current medoid to be replaced, $p$ is a possible new medoid to replace $m$, $r$ is any other object which is not a medoid, and $m'$ is a current medoid which is nearest to $r$ without $m$ and $p$. The cost $S_{swap}$ can be calculated from 4 different cases.

1. Object $r$ currently belongs to the cluster represented by $m$ and is closer to $m'$ than $p$, then, if $m$ is replaced by $p$, $r$ will be assigned to the cluster represented by $m'$, so the cost of the replacement as far as $r$ is concerned will be: $d(r, m') - d(r, m)$, which is positive.

2. Object $r$ currently belongs to the cluster represented by $m$ and is closer to $p$ than $m'$, then, if $m$ is replaced by $p$, $r$ will remain in the same cluster, now with medoid $p$ rather than $m$, and the cost will be: $d(r, p) - d(r, m)$, which is negative.

3. Object $r$ currently belongs to the cluster represented by $m'$ and is closer to $m'$ than $p$, then, if $m$ is replaced by $p$, $r$ will stay in the cluster represented by $m'$, so the concerned cost is: 0

4. Object $r$ currently belongs to the cluster represented by $m'$ and is closer to $p$ than $m'$, then, if $m$ is replaced by $p$, $r$ will be assigned to the cluster represented by $p$, so the cost is $d(r, p) - d(r, m')$, which is negative.

To calculate the total cost of a replacement, we sum up the cost from all the non-medoid objects. The above 4 cases also indicate the way to reconstruct the clusters when a medoid is replaced.

The idea of the PAM algorithm is: Given the number of clusters decided by the user to be $k$, and the data matrix or data dissimilarities matrix, PAM finds $k$ clusters by minimizing $E$.

The algorithm of PAM:

1. The value of $k$ is given by the user.
2. Randomly choose $k$ objects from the data set to be the cluster medoids at the initial state.
3. Assign each of the remaining objects to a cluster if the object is closest to the medoid of the cluster.
4. For any medoid $m$ do the following: For any object $p$ which is not a medoid, compute the cost $S_{swap}$ when replacing $m$ by $p$, if the cost $S_{swap}$ is negative, replace $m$ by $p$ and reconstruct the clusters.
5. Repeat steps 3 and 4 until no change happens

K-means and k-medoids are different in the following two points: PAM tries to use the most centrally located object within a cluster to be the representative. Such an object is a medoid, it is an existing object in the data set. Since a medoid is an existing object in the data set, the second major difference comes from the way to recalculate the cluster representatives and to reconstruct the clusters when the cluster representatives (candidate medoids) are changed.

PAM is not efficient for a large data set. By reviewing the above algorithm, there are $k \times (n-k)$ pairs of $(m,\ p)$, and for each pair it needs $n-k$ calculations to obtain the total cost for a replacement, so the complexity of only one iteration is $O(k(n-k)^2)$. PAM can be too costly for large data set. How to improve its efficiency has been taken up by many researchers.

### 3.1.3 PAM with sampling

The key point in PAM is to find the $k$ medoids to minimize $E$. Its way to search medoids has been carefully analyzed, and a searching graph can be derived as follows. A node in the graph is denoted by a set of $k$ currently selected medoids. Two nodes are the neighbours of each other if and only if their sets differ by only one medoid. To make a neighbour of a node $N$, we simply replace only one medoid in the node $N$. PAM actually starts from an arbitrarily chosen node in the graph and search all its neighbours to find the least cost neighbour node and moves to that node until all the neighbours of the current node $N$ cost more than the node $N$. PAM has a big chance to find the global optimum when the number of objects in the data set is much bigger than the number of clusters. However, the number of nodes in the searching graph is too huge, it is very inefficient when the data set is large.

To reduce the number of nodes in the graph is a possible way to improve the efficiency of PAM. At least two methods have been proposed based on PAM and sampling techniques to simplify the searching graph. One is called CLARA [6], the other is called CLARANS [7] which is motivated by improving the effectiveness of CLARA.

The algorithm of CLARA:

1. The value of $k$ is input by the user.
2. Preset a large value for the average dissimilarities of the clustering to be the current minimum.
3. Draw a random sample with a small size, say $40 + 2k$ objects, from the large data set.
4. Use PAM to find the $k$ medoids in the selected sample.

5. Assign each non-medoid object in the entire data set to one of the $k$ clusters if the object is closest to the medoid of that cluster.

6. Calculate the average dissimilarities of the clustering in step 5 and compare it with the current minimum, if it is less than the current minimum, set it to be the value of the current minimum, and keep the $k$ medoids found in step 4 to be the best set of medoids found so far.

7. Repeat step 3 to step 6, 5 times.

CLARA searches in the sub-graphs of the original searching graph described above. These sub-graphs are formed by the random samples to make the number of searching nodes much fewer than that in PAM. CLARA therefore increases the efficiency rapidly.

Because the global optimum node probably may not be contained in any of the sub-graphs, reducing the entire searching graph by drawing a small sample several times seems not reasonable to keep the effectiveness of PAM. CLARANS was proposed to solve the problem.

CLARANS searches in the entire graph without comparing the current node $N$ with all its neighbours. Instead, in order to find a better node to replace $N$, the algorithm compares $N$ with each node in a random sample drawn from all the neighbours of $N$, if $N$ is better than all the sample nodes, it is retained as a local optimum. CLARANS tries to find a relative large number of such local optima by iteration in the algorithm and obtain the best one among them as the result.

The algorithm of CLARANS:

1. The value of $k$ is given by the user.

2. Two other parameters **numlocal** and **maxneighbour** are chosen by the user as well, where **numlocal** is the maximum number of restarts

that the algorithm could have, **maxneighbour** is the maximum size of the sample which is used for comparing the current node with a random sample of its neighbours.

3. Set the current node to be an arbitrary node in the entire searching graph. This is done by arbitrarily selecting $k$ medoids from the whole data set as the candidate medoids.

4. Initialize $i$ to be 1 and **mincost** to be a large value that would be bigger than the cost $S_{swap}$ when the current node is replaced by any of its neighbours.

5. Set $j$ to be 1.

6. Randomly choose a neighbour node $R$ of the current node and compute the cost $S_{swap}$ if the current node is replaced by $R$. The randomly chosen neighbour node is obtained by arbitrarily switching one candidate medoid in the current node with any other node in the data set, which is not a candidate medoid in the current node.

7. If the cost is negative, set the current node to be $R$ and go to step 5, otherwise, increase $j$ by 1

8. If $j <$ **maxneighbour**, go to step 6, otherwise, compare the cost of the current node with **mincost**, if the former is less, set the **mincost** to be the cost of the current node and set the **bestnode** to be the current node

9. Increase $i$ by 1, if $i >$ **numlocal**, output **bestnode**, otherwise go to step 5

By comparing CLARA and CLARANS with PAM, CLARA first draws random samples of the data set and then do PAM on these samples. Unlike CLARA, CLARANS draws a random sample from all the neighbour nodes of the current node in the searching graph described above.

From k-means to PAM, then, from PAM to CLARANS, we can see the efforts made by the researchers to obtain better partioning clustering methods. However, these partitioning methods are more likely to find sphere-shaped clusters. There exist common problems in these methods:

- All these methods rely on the user to choose the number of clusters as a parameter.
- They use a single point as the representative of a cluster. A single representative could not capture the shape of the cluster easily if the clusters have arbitrary shapes, for example, cluster shapes are convex.
- Since the purpose of these algorithms is to find clusters to minimize $E$, they fail for data in which points in a given cluster are close to the centre of another cluster than the centre of their own cluster.

## 3.2 Hierarchical methods

The hierarchical methods construct a hierarchical decomposition of the data set. Two typical ways are applied in the hierarchical methods. One is called bottom-up or agglomerative and the other is called top-down or divisive. In the initial step of bottom-up methods, each object forms a cluster. If the number of objects in the data set is $n$, there are $n$ clusters at the initial state of the algorithm. The algorithm seeks two **closest** clusters by some metric measurement and merges these two clusters until only one cluster is left. In top-down methods, the algorithm goes in the opposite way. It starts with only one cluster in which all the objects are contained, and keeps splitting until each distinct object forms its own cluster. For either method, a stop criterion is often introduced so as to convince at some desirable set of clusters.

A tree structure (or a dendrogram) can be used to record the merging or splitting of the clusters and indicate the distance between two joined clusters.

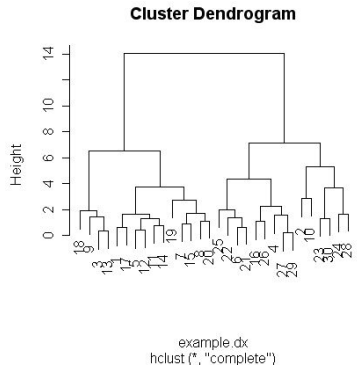


Figure 3: Example of a dendrogram

The dendrogram thus displays a nested sequence of clusters and can be used by the users to choose the number of clusters. Figure 3 shows an example of a dendrogram obtained by hierarchical clustering on the Gauss2 data set (the data set is available in Stat 441 course web page). In the dendrogram, points or groups of points were merged first when they are closer to each other, after two big groups were formed, they merged to be one group, the distance between these two groups are relatively long. The height in the dendrogram demonstrates the distance between two groups if there exists a merge of them.

Early hierarchical methods employ some simple metric functions to measure the dissimilarities between any two clusters, say $C_i$ and $C_j$. Three common measurements are:

- Single linkage:

$$d_{min}(C_i, C_j) = min_{\forall p \in C_i, \forall q \in C_j} \|p - q\|$$

- Complete linkage:

$$d_{max}(C_i, C_j) = max_{\forall p \in C_i, \forall q \in C_j} \|p - q\|$$

- Average linkage:

$$d_{avg}(C_i, C_j) = \frac{1}{|c_i||c_j|} \sum_{p \in C_i, q \in C_j} \|p - q\|$$

To obtain better hierarchical methods, two approaches are proposed recently and are necessary to be mentioned here. The first approach, represented by BIRCH [8], creates a hierarchical structure that can be used to refine the clustering result and save the computation resource requirement. The second approach, represented by CURE [9] and CHAMELEON [10], adopts more complicated but reasonable measurements to determine the dissimilarities between clusters.

### 3.2.1 Clustering using the Clustering Feature

A Clustering Feature (CF) is used to represent a sub-cluster by the general statistical information of it. The structure of a $CF$ is a triple storing the number of data objects, the linear and the square sum of the data vectors in the represented sub-cluster. The definition is: $CF = \{N, LS, SS\}$.

We can observe a very useful property of additivity from $CF$. Suppose that $CF_1 = \{N_1, LS_1, SS_1\}$ and $CF_2 = \{N_2, LS_2, SS_2\}$ are the $CF$s of two disjoint clusters $C_1$ and $C_2$. Then, the $CF$ of the cluster obtained by merging $C_1$ and $C_2$ is exactly equal to $CF_1 + CF_2 = \{N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2\}$.

Other necessary statistical information used for typical hierarchical or partitioning clustering methods can be calculated easily by using the $CF$s. For example, by giving the related $CF$s of two clusters $C_1$ and $C_2$, it is not difficult to calculate their centroids $X_{0_1}$ and $X_{0_2}$, the radius $R$ of a single cluster,

and the centroid Euclidean distance $D$ between two clusters. Suppose the two $CF$s are $\{N_1, LS_1, SS_1\}$ and $\{N_2, LS_2, SS_2\}$, the calculation is as follows:

$$X_{0_1} = \frac{\sum_{\forall X_i \in C_1} X_i}{N_1} = \frac{LS_1}{N_1}$$

$$X_{0_2} = \frac{\sum_{X_i \in C_2} X_i}{N_2} = \frac{LS_2}{N_2}$$

$$R_{C_1} = \left( \frac{\sum_{\forall X_i \in C_1} \|X_i - X_{0_1}\|^2}{N_1} \right)^{\frac{1}{2}} = \frac{1}{N_1} sum_{element}\{(SS_1 - \frac{LS_1^2}{N_1})^{\frac{1}{2}}\}$$

$$D = \|X_{0_1} - X_{0_2}\| = \|\frac{LS_1}{N_1} - \frac{LS_2}{N_2}\|$$

Where, $sum_{element}\{V\}$ is the summation of all the elements in the vector $V$.

In BIRCH, $CF$s are stored in a Clustering Feature tree (CF tree), which is used for hierarchical clustering. To build the CF tree, two parameters are given by the users: the branching factor $B$ and the threshold $T$. These two parameters constrain the growing of the tree. The branching factor $B$ specifies the maximum number of children per non-leaf node. Therefore, a non-leaf node in the tree can have at most $B$ children. Every non-leaf node stores the sum of $CF$s from all its children. For a leaf node, it stores the $CF$ calculated from all the objects in the sub-cluster which is represented by the leaf node. The threshold $T$ is used to limit the size of the sub-clusters, which are represented by leaf nodes. To be more precise, the radius of all the sub-clusters should be less than $T$.

The algorithm for inserting an entry *ent* into the $CF$ tree has three main steps. Here, an entry can be a single data object or a group of data objects. The algorithm goes as follows:

1. Identify the appropriate leaf to insert: Find the leaf node, which represents a sub-cluster that is closest to the inserted entry. The closeness can be measured by the Euclidean distance between the centroids of the sub-cluster and *ent*. This operation starts from the root and recursively descends the $CF$ tree by always choosing the closest child node.

2. Modify the leaf node: Suppose the sub-cluster in the leaf node $LN_i$ is closest to *ent*. Check if $LN_i$ can absorb *ent* without violating the threshold constraint $T$. If so, simply modify the $CF$ in $LN_i$. If not, add a new leaf node, say $LN_j$ to be a new child of the parent node of $LN_i$. Then, form a group of objects by all the objects in the sub-group represented by $LN_i$ and all the objects in *ent*. Divide this group to be two sub-groups by the methods: First, find two objects that are the most far away from each other in the group; Suppose these two objects are $obj_1$ and $obj_2$, assign all the objects in the group to sub-group 1 if the objects are closer to $obj_1$ than $obj_2$, the other objects form the sub-group 2. Now, assign sub-group 1 and 2 to be the sub-clusters represented by $LN_i$ and $LN_j$ respectively.

3. Modify the path to the root node: The $CF$ in each node on the path from the affected leaf nodes to the root must be modified after *ent* has been inserted. If a split did not happen in step 2, only the affected $CF$s are necessary to be modified. If a split happened, then check if the constraint of the branching factor $B$ is violated. If so, the parent node has to be split, and this split may affect the nodes in high levels the same way as well. If the root node is split, the height of the tree increases by 1. The way to split a non-leaf node is similar as the way to split a leaf node in step 2, the only difference is that, when split a non-leaf node, use all its children to form a group to be divided.

Figure 4 shows an example of building a $CF$ tree from a simple data set

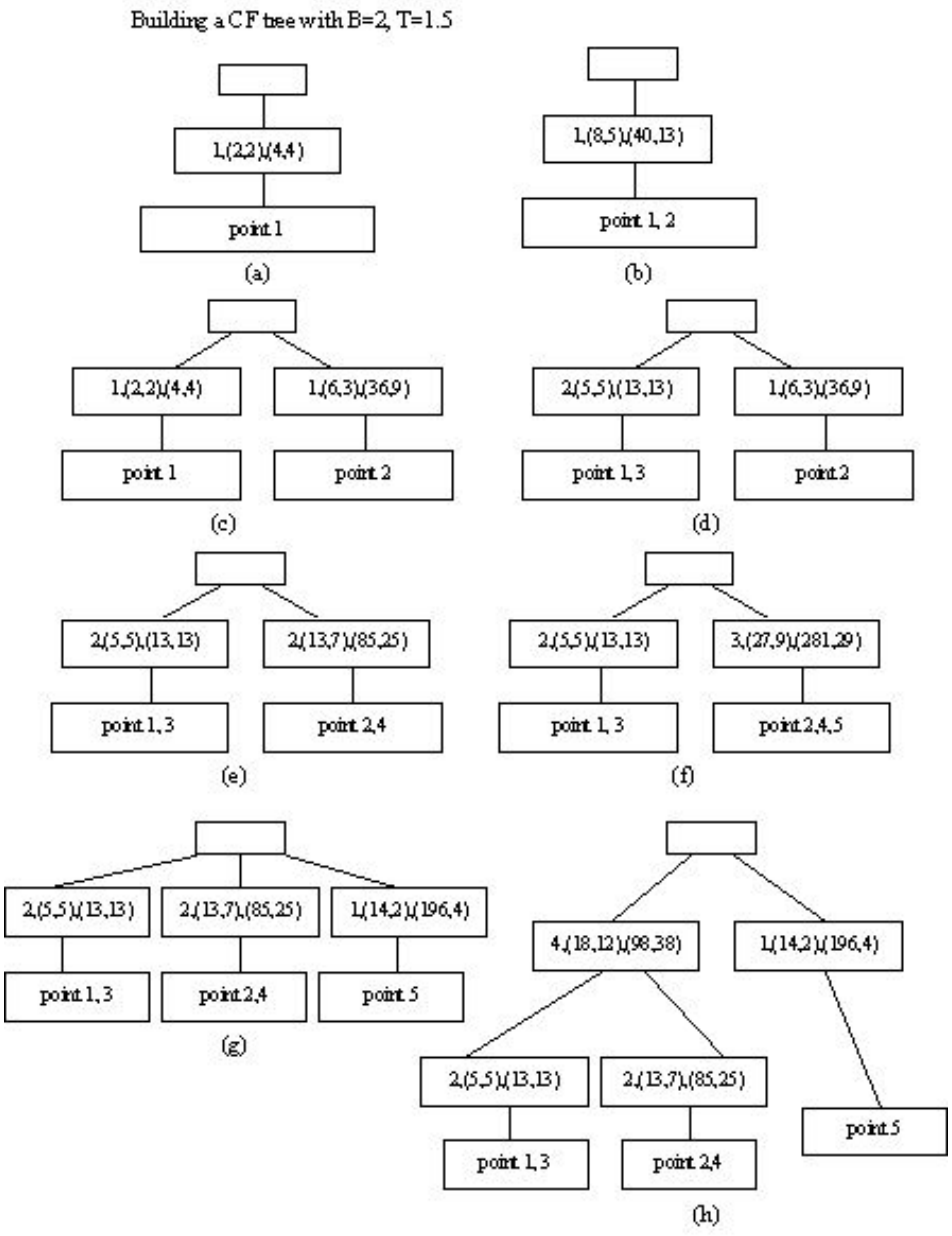


Figure 4: The example of building a $CF$ tree

including five objects. The coordinates of these five objects are: (2,2) (6,3) (3,3) (7,4) (14,2). We use $B = 2$ and $T = 1.5$ as the threshold. The $CF$s are displayed for each nodes in the plot. When point 2 is added after point 1 (part (b) of the plot), the radius of the leaf node, which contains point 1 and 2, violates the threshold $T$, it has to be split (part (c)). When point 5 is added at last (part (g)), the root node has three children, this violates the threshold $B$, it has to be split and the height of the tree is increased by 1 (part (h)). Each time when a point is added, the $CF$s in the affected nodes have to be updated.

BIRCH builds the $CF$ tree by scanning the total data set once, then obtains the clustering result based on the $CF$ tree, which could be done in the main memory of the computer. This algorithm is significantly appropriate for finding clusters in a very large data set. Since the $CF$ tree can be constructed incrementally, any data objects given later can be added into the tree without rebuilding it. So this incremental clustering is another notable advantage of BIRCH.

We can see from the BIRCH algorithm how these two parameters $B$ and $T$ control the building of the $CF$ tree. In BIRCH, the choice of them is mainly determined by memory constraint from the computers. Since the cluster size is limited by these parameters, a cluster found from the $CF$ tree does not always correspond to a nature cluster. This is a problem of BIRCH.

### 3.2.2 Clusters represented by multiple representatives

When assigning objects to clusters, merging two clusters into one or splitting a cluster, some clustering methods use a single object to represent the whole cluster, such as the cluster mean or a representative point in that cluster. Other methods use all the objects in the cluster to represent their cluster, such as the single linkage, complete linkage and the average linkage clustering.

Even BIRCH can be classified into such a one or all category. None of these methods work well for finding non-sphere or arbitrary shaped clusters.

CURE was proposed to deal with the above problem by using a constant number of well scattered points in a cluster as the representatives. The algorithm of CURE is based on the following idea: The scattered points could capture the shape and extend of the cluster. During the clustering process, the chosen points are made to be the representatives by being shrunk towards the centroid of the cluster by a fraction $\alpha$. CURE is a hierarchical clustering method, the merging policy is similar with the typical hierarchical methods. At each step of the algorithm, two clusters with the closest pair of representative points are merged.

Four key points are necessary to be mentioned to make the algorithm clear.

- How to deal with the effect from outliers
- How to choose the scattered points
- How to shrink these chosen points to make them the representatives of the cluster
- How to calculate the distance between a pair of representative points

The solutions from CURE are straightforward. The reason why CURE shrinks the selected points is to get rid of the surface abnormalities and mitigate the effects from the outliers. If an outlier, which is far away from the centroid of the cluster, has been selected, it can be moved a relatively large distance towards the centroid during the shrinking, and this could degrade the effect due to the outlier. The scattered points are chosen by selecting the point farthest from the centroid first, then recursively picking up the point farthest from the scattered point, which has been chosen the last, until the number of selected points reaches the preset constant. After selecting these

scattered points, CURE shrinks them by the following method: Suppose the data vector of a scattered point and the centroid of the cluster are $\mathbf{P}$ and $\mathbf{C}$ respectively, the shrinking fraction is $\alpha$, then replace $\mathbf{P}$ by $\mathbf{P}+\alpha(\mathbf{C}-\mathbf{P})$. The shrinking fraction is a number between 0 and 1. At each step of the clustering process, CURE selects two clusters to merge if the smallest distance between two representative points coming from these two clusters respectively is the shortest. After the merging, the representative points are recalculated for the new cluster. The computational complexity of CURE is shown in [9] to be $O(N^2)$. To improve the efficiency, the randomly sampling and partitioning techniques are considered in CURE.

### 3.2.3 Clustering by a dynamical model

CHAMELEON is a hierarchical clustering method that uses a dynamical and flexible merging policy to discover nature and homogeneous clusters. Unlike some other clustering methods, in CHAMELEON, two clusters are merged only if the inter-connectivity and closeness between two clusters are high relative to the internal inter-connectivity of the clusters and closeness of items within the clusters. Such a complicated policy is proposed to overcome the drawbacks from many existing clustering methods.

Quite a few hierarchical and partitioning clustering methods measure the similarity between two clusters using the absolute closeness between two clusters. For example, PAM uses the distance between medoids from two clusters, CURE uses the closest pair of the representatives belonging to different clusters, and single linkage clustering uses the closest pair of objects belonging to different clusters.

Different from the above methods, CHAMELEON finds clusters by a two phase algorithm: First, the whole data set is grouped into a large number of relatively small sub-clusters by some graph partitioning technique, these

small sub-clusters are regarded to be atoms in the algorithm, then, during the second phase, it employs a hierarchical clustering algorithm to obtain the clusters by recursively merging the sub-clusters until the user-specified stop criterion is satisfied.

In order to measure the similarity between two clusters and perform the graph partitioning in phase one, CHAMELEON uses the k-nearest neighbour graph approach. In the k-nearest neighbour graph, each vertex represents a data object and if two objects are the k-most similar neighbours of each other, there exists an edge to connect these two vertices. The weight of an edge is determined by measuring its length in the way that longer edge weights less. Such an approach has the following advantages:

- Data objects that are far apart are completely disconnected in the graph
- The graph captures the concept of neighbourhood dynamically in the way that the neighbourhood radius of an object is determined by the density of the region in which this object resides.
- The graph can be partitioned into a large number of small initial sub-clusters efficiently by current graph partitioning techniques, such as min-cut. A min-cut refers to partitioning the graph into roughly two parts of equal size such that the total weight of the edges being cut is minimized.

CHAMELEON determines the similarity between each pair of clusters $C_i$ and $C_j$ according to both their relative inter-connectivity $RI(C_i, C_j, G)$ and their relative closeness $RC(C_i, C_j, G)$, where $G$ is the current clustering graph initially derived from the k-nearest neighbour graph with all the edges, which will be cut by min-cut recursively, removed to obtain the initial sub-clusters. The graph $G$ is changed each time after a merging of two clusters. To explain the above concepts well, some definitions are necessary.

A clustering graph is defined to be $G = \{V, E\}$, where each vertex $v \in V$, represents a data object, and in the initial state, a weighted edge $(v_i, v_j) \in E$ exists between two vertexes $v_i$ and $v_j$, if and only if $v_i$ is one of the k-nearest neighbours of $v_j$, and this edge has not been cut by min-cut during the process to obtain the initial sub-clusters. The weight of an edge denotes the closeness between the two objects it connects. An edge weighs more if its two connected objects are closer to each other. After each merging of two clusters, the edge set $E$ is grown by adding back all the edges that are used to connect the two clusters in the original k-nearest neighbour graph but have been cut in the initial state of $G$.

**Definition 3.1 (split of a cluster)** *A split of a cluster $C$ (or a separated sub-graph in $G$) is defined to be $split(C) = \{C_1, C_2, G\}$, where $C_1$ and $C_2$ are the two sub clusters obtained by applying a min-cut on $C$.*

**Definition 3.2 (set of edges connecting two clusters)** *The set of edges that connect two clusters $C_i$ and $C_j$ in $G$ is defined to be $EC(C_i, C_j, G)$.*

**Definition 3.3 (set of edges that will be cut when splitting a cluster)** *The set of edges that will be cut in $G$ when splitting a cluster $C$ into $C_1$ and $C_2$ by applying min-cut on $C$ is defined to be $ES(C) = EC(split(C))$.*

**Definition 3.4 (total weight of a set of edges)** *The total weight of a set of edges is defined to be $weight(E_c) = \sum_{e \in E_c} weight(e)$.*

**Definition 3.5 (average weight of a set of edges)** *The average weight of a set of edges is defined to be $avg(E_c) = \frac{weight(E_c)}{\#E_c}$, where $\#E_c$ is the number of edges in the set $E_c$.*

**Definition 3.6 (relative inter-connectivity between a pair of clusters)** *The relative inter-connectivity between a pair of clusters is defined to be* $RI(C_i, C_j, G) = \frac{weight(EC(C_i,C_j))}{\frac{weight(ES(C_i))+weight(ES(C_j))}{2}}$.

The relative inter-connectivity takes into account the differences in both cluster shape and the degree of connectivity of different clusters.

**Definition 3.7 (relative closeness between a pair of clusters)** *The relative closeness between a pair of clusters is defined to be* $RC(C_i, C_j, G) = \frac{avg(EC(C_i,C_j))}{\frac{\#V(C_i)}{\#V(C_i \cup C_j)} avg(ES(C_i)) + \frac{\#V(C_j)}{\#V(C_i \cup C_j)} avg(ES(C_j))}$, *where* $\#V(C)$ *is defined to be the number of vertices in the separated sub-graph representing the cluster* $C$ *in* $G$.

The relative closeness is able to conduct the algorithm to merge clusters whose resulting cluster exhibits a uniformity in the degree of closeness between the items in the cluster.

Based on the above method to model the similarity between clusters, CHAMELEON finds clusters by using a two phase clustering algorithm. In phase one, it obtains initial sub-clusters by applying an efficient graph partitioning method called min-cut on the k-nearest neighbour graph to partition the graph or the data set into a large number of small clusters, which are treated as the initial sub-clusters. In phase two, it uses a dynamic merging policy to merge these initial sub-clusters iteratively until the stop criterion is satisfied. The merging policy is: The algorithm compares the values of a similarity function for any pair of current clusters, and merge the pair with the largest value. The similarity function $f$ is defined to be: $f(C_i, C_j, G) = RI(C_i, C_j, G)^{\alpha} RC(C_i, C_j, G)$, where $\alpha$ is a user specified parameter, if $\alpha > 1$, then, CHAMELEON regards the relative inter-connectivity as more important, while $\alpha < 1$, the algorithm gives more importance to the relative closeness.

The overall computational complexity of CHAMELEON is $O(N^2)$, similar with CURE in computing time, but the clustering performance is snown in [10] to be better than CURE and quite a few other hierarchical clustering methods due to its dynamical model for the merging policy.

BIRCH, CURE and CHAMELEON are all hierarchical clustering methods. They differ in the following main points:

- BIRCH builds a $CF$ tree to obtain clusters, the clustering is done when the tree is built by scanning the data set only once. When an object or a group of objects is absorbed into a leaf node, it does a merge of objects; when a node is split, it does a split of a sub-cluster. The tree is built after such merges and splits. This is not the same as many hierarchical methods in which only merges or only splits are allowed.
- CURE uses several representatives to capture the shape and extend of a cluster, then does a single linkage hierarchical clustering on only these representatives for each clusters.
- CHAMELEON does hierarchical clustering on the partitions of the k-nearest neighbour graph. It uses a dynamic merging policy, which consider both the dissimilarity between two clusters and the similarity within a possible cluster.

## 3.3 Density based clustering methods

With density-based methods, density can be defined to be the number of objects in a predefined unit area in the data space. The purpose of this kind of clustering is to group points from each high-density region into a cluster respectively and ignore the objects in low-density regions. The density-based approaches find arbitrarily shaped clusters. Some major differences in such

approaches are the way they measure the density and the strategy they use to achieve clusters from high-density areas.

### 3.3.1 Finding clusters by a global density level

DBSCAN is a method based on the assumption: The density within each cluster is higher than outside of the cluster; The density in the area of noises is lower than the density in any of the clusters [11].

Unlike most of the partitioning and hierarchical clustering methods, DBSCAN neither requires the user to know how many clusters there are in advance, nor depends on some stop policy. In DBSCAN, only a global density level is required to determine the high-density areas in the data space.

The following definitions are helpful.

**Definition 3.8 ($\epsilon$-neighbourhood of a point)** *The $\epsilon$-neighbourhood of a point $p$, denoted by $N_\epsilon(p)$, is defined by, $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$, where $D$ is the given data set, $p \in D$, and $dist(p,q)$ is a distance function which calculates the dissimilarity between $p$ and $q$.*

**Definition 3.9 (core-point)** *For any point $p$ in $D$, $p$ is a core-point wrt $\epsilon$ and $Minpts$ if and only if $N_\epsilon(p) \geq Minpts$, where $Minpts$ is a given constant.*

Figure 5 shows an example of a core-point, the region of $\epsilon$-neighbourhood of the point $p$ is marked by a circle. Suppose we set $Minpts$ to be 4, there are 6 points in that region, so $p$ is a core-point.

**Definition 3.10 (directly density-reachable)** *A point $q$ is directly density-reachable from a point $p$ wrt $\epsilon$ and $Minpts$ if and only if $q \in N_\epsilon(p)$ and $p$ is*

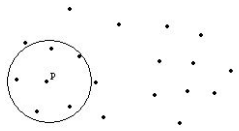


Figure 5: Example of a core-point

*a core-point wrt $\epsilon$ and $Minpts$*

In figure 6, $q$ sites inside the region of $\epsilon$-neighbourhood of $p$, and $p$ is a core-point, so $q$ is directly density-reachable from $p$. Because $q$ is not a core-point, $p$ is not directly density-reachable from $q$.

**Definition 3.11 (density-reachable)** *A point $q$ is density-reachable from a point $p$ wrt $\epsilon$ and $Minpts$ if and only if there exists a chain of points $p_1, p_2, ..., p_n$, where $p_1 = p, p_n = q$, such that $p_{i+1}$ is directly density-reachable from $p_i$.*

**Definition 3.12 (density-connected)** *A point $q$ is density-connected with a point $p$ wrt $\epsilon$ and $Minpts$ if and only if there exists a point $r$, such that both $p$ and $q$ are density-reachable from $r$ wrt $\epsilon$ and $Minpts$.*

In figure 7, since both $p$ and $r$ are core-points, $q$ is density-reachable from $r$, $q$ and $r$ are also density-connected.

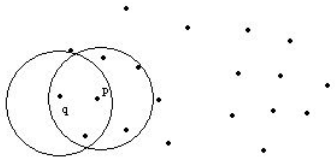


Figure 6: Example of directly density-reachable

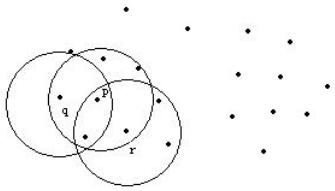


Figure 7: Example of density-reachable and density-connected

Now we can define the notation of a cluster in the density-based clustering by the above concepts.

**Definition 3.13 (cluster)** *Let $D$ be a data set of points. A cluster $C$ wrt $\epsilon$ and $Minpts$ is a non-empty subset of $D$ satisfying both of the following two conditions.*

- *$\forall p, q \in D$, if a core-point $p \in C$ and $q$ is density-reachable from $p$ wrt $\epsilon$ and $Minpts$, then $q \in C$.*
- *$\forall p, q \in C$, $p$ is density-connected to $q$ wrt $\epsilon$ and $Minpts$.*

Definition 3.13 implies that each cluster must contain at least one core-point wrt $\epsilon$ and $Minpts$.

**Definition 3.14 (border point)** *Any point in a cluster that is not a core-point is a border point in the cluster.*

**Definition 3.15 (noise)** *A point in the data set that does not belong to any cluster is a noise point.*

Since all the core-points in $C$ are density-reachable from each other, each point in a cluster $C$ is density-reachable from any of the core-points in $C$, thus, $C$ contains exactly the points which are density-reachable from an arbitrary core-point in $C$. This property gives us a feasible way to obtain a cluster.

The algorithm of DBSCAN:

1. $\epsilon$ and $Minpts$ are given by the user.

2. Mark all the points in the data set to be unclassified.

3. Find an unclassified core-point $p$ wrt $\epsilon$ and $Minpts$. Mark $p$ to be classified. Start a new cluster to be the current cluster and assign $p$ to the current cluster.

4. Find all the unclassified points in the $\epsilon$-neighbour of $p$. Create a set of seeds and put all these points into the set.

5. Get a point $q$ in the seeds, mark $q$ to be classified, assign $q$ to the current cluster, and remove $q$ from the seeds.

6. Check if $q$ is a core-point wrt $\epsilon$ and $Minpts$, if it is, add all the unclassified points in the $\epsilon$-neighbour of $q$ to the set of seeds.

7. Repeat step 5 to step 6 until the set of seeds is empty.

8. Start a new cluster and repeat step 3 to step 7 until no more core-points can be found.

9. Output all the clusters found so far, and mark all the points, which do not belong to any cluster, to be noise.

The algorithm starts with an arbitrary core-point $p$ wrt $\epsilon$ and $Minpts$, and retrieves all points, which are density-reachable from $p$ to be a cluster. When no more points can be added into the current cluster, DBSCAN begins to grow the other clusters by the same method until no more clusters can be found.

In the DBSCAN algorithm, a core-point acts exactly as a high-density attractor by whom the points site in its $\epsilon$-neighbourhood are absorbed. The average run time complexity of this algorithm is $O(NlogN)$.

Although DBSCAN can find arbitrary shaped clusters, it only takes into account the global density level. In many data sets, to discover clusters in different regions of the data space, different local density levels are required.

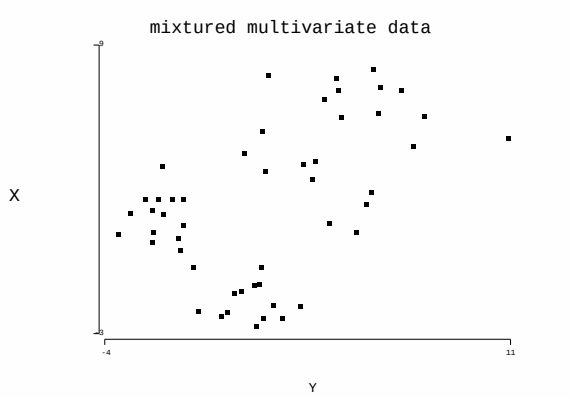


Figure 8: Example of a date set from a mixture of three distributions

Figure 8 shows a data set derived from a mixture of three 2-d multivariate normal distributions. The points at the right part of the plot are simulated from a multivariate normal with relatively large variances and covariances, the points at the left top part and the left bottom part of the plot are simulated from two multivariate normal distributions with relatively small variances and covariances. DBSCAN is hard to find all these three groups using only a global density level. Although, it can eventually find all the interesting patterns in the data set by running several times with different global density levels, the cost of computing is expensive.

Figure 9 shows the clustering on the mixture of three multivariate normal data set. The result is obtained by running the codes I wrote for implementing the DBSCAN algorithm. I set $Minpts$ to be 4 and $\epsilon$ to be 2. DBSCAN finds three groups. I marked the points in different groups by different shapes, and marked the noise by cross .The clustering is reasonable by just looking at the data at a glance, but it does not reveal the true pattern of the data.

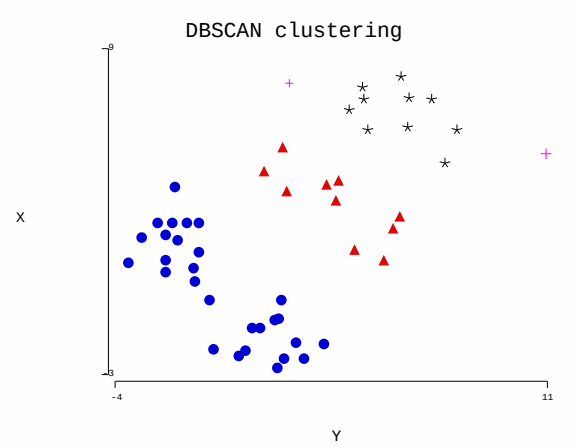


Figure 9: Example of DBSCAN clustering ($Minpts = 4$, $\epsilon = 2$)

### 3.3.2 Finding clusters from an ordered density-based clustering structure

Many clustering methods suffer from the similar problems:

- They require input parameters, which are hard to determine exactly. (e.g. k-means, PAM, and CLARANS require the number of resulting clusters as the input parameter; DBSCAN needs $Minpts$ and $\epsilon$ to determine the density level).
- Such clustering methods are very sensible to their parameter values. Slightly changing the value of the parameter can yield significantly different clustering.
- The data objects in many high-dimensional real data sets often have a very skewed distribution that cannot be revealed by a clustering algorithm using the global parameter setting.

DBSCAN's global density parameter setting cannot reveal clusters in data sets where different local densities are necessary to find clusters in different

regions of the data space.

A possible solution may be applying the DBSCAN multiple times with different parameter settings. However, there is infinite number of possible parameter settings, the run time cost would be much expensive.

OPTICS is then proposed to solve the above problems by constructing an augmented ordering of the data set to represent its density based clustering structure [12]. With this structure, any DBSCAN results can be obtained efficiently with a fixed value of $Minpts$ and any $\epsilon$ values that are less than a generating distance $\epsilon_0$.

The idea of OPTICS comes from the following observation: When $Minpts$ is fixed, the clusters found by DBSCAN with a smaller $\epsilon$ value are completely included by the clusters obtained by DBSCAN with a bigger value of $\epsilon$. This property makes it possible to find density based clusters with any density levels based on an ordered clustering structure constructed by using a relatively large value of $\epsilon$. To decribe such a structure, additional notation is introduced.

**Definition 3.16 (core distance)** *Let $dist_{Minpts}(p)$ be the distance from a point $p$ to its $(Minpts-1)'s$ neighbour. If $p$ is a core point wrt $\epsilon$ and $Minpts$, $dist_{Minpts}(p)$ will be called the core distance of $p$ wrt $\epsilon$ and $Minpts$.*

The core-distance of a point $p$ is used to provide the information about the minimum value of $\epsilon$ to make the point $p$ to be a core-point.

**Definition 3.17 (reachability-distance)** *The reachability-distance of a point $p$ to another point $q$ wrt $\epsilon$ and $Minpts$, is denoted as reachability-distance$_{\epsilon,Minpts}(p,q)$. It is defined by reachability-distance$_{\epsilon,Minpts}(p,q) = max(dist_{Minpts}(q), distance(p,q))$, if $q$ is a core-point and distance(p,q) $\leq \epsilon$.*

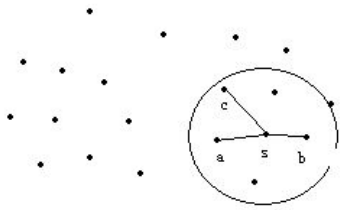


Figure 10: Example of core-distance and reachability-distance

The reachability-distance of a point $p$ to $q$ determines the minimum value of $\epsilon$ to make the point $p$ directly density-reachable from $q$, if $q$ is a core-point.

In figure 10, because $a$ is the third nearest neighbor of $s$, if we set $Minpts$ to be 4, the core distance of $s$ is the distance from $s$ to $a$. The reachability-distance of $b$ to $s$ is the core-distance of $s$, the reachability-distance of $c$ to $s$ is the distance from $c$ to $s$.

The algorithm of OPTICS includes two parts. The first part deals with generating the ordered clustering structure. The second part deals with deriving density-based clusters given any $\epsilon$ as long as $\epsilon \leq \epsilon_0$, with $Minpts$ fixed.

Part one: construct the ordered clustering structure

1. The generating distance $\epsilon_0$ and $Minpts$ are given by the user.

2. Build a vector with length to be the number of points in the data set, such that each point in the data set will have a corresponding element in the vector. Each element is a triple storing the point ID, say $p_1$, core-diatance of $p_1$ and the reachability-distance from $p_1$ to the closest core-point which sites before $p_1$ in the vector. The vector will be an ordered clustering structure. At the beginning, it is empty.

3. Arbitrarily select a point, whose information has not been stored in the ordered structure, and check if it is a core-point wrt $\epsilon_0$ and Minpts. If it is not a core-point, set both its core-diatance and the reachability-distance to be UNDEFINED. If it is a core point, calculate its core-distance, and assign its reachability-distance to be UNDEFINED since the current selected point is not directly density-reachable from any point that has been stored in the ordered structure before it. Now store the information of the current selected point in the ordered structure.

4. Repeat step 3 until a core-point, say $p$, is found.

5. Find all the points in the $\epsilon$-neighbours of $p$ wrt $\epsilon_0$ and Minpts, create an ordered set of seeds and put all the points in that $\epsilon$-neighbours into the set. All the points in the set are ranked by their reachability-distances to $p$, the shortest the first.

6. Get the first seed in the set, say point $q$, add the information of $q$, including the core-distance of $q$ and the reachability-distance to $p$, to the ordered structure. Then, remove $q$ from the set of seeds. If $q$ is a core-point, adds all the $\epsilon$-neighbours of $q$, which have not appeared in the ordered structure, to the set of seeds. If any point, say $r$ is already in the set, compare the reachability-distance from $r$ to $p$ with

that distance from $r$ to $q$, and keep the smaller value. Reorder the set of seeds by the current reachability-distances.

7. Repeat step 6 until the set of seeds is empty.

8. Repeat step 3 to step 7 until the information of all the points in the data set has been stored in the ordered structure.

OPTICS constructs the ordered clustering structure during completing a DBSCAN clustering with density level $Minpts$ and $\epsilon_0$. Each point in the structure holds the reachability-distance to its closest core-point appearing before it in the ordered structure, if that distance exists.

This particular order described above can be used to find DBSCAN clustering with $Minpts$ fixed and $\epsilon$ less than the generating distance $\epsilon_0$.

Part two: extract clusters from the ordered clustering structure

1. $\epsilon$ is given by the user.

2. Set $clusterID$ to be 1.

3. Get the first core-point wrt $\epsilon$ and Minpts in the ordered structure, say point $p$, assign $p$ to the current cluster identified by $clusterID$.

4. Assign all the successive points, which follow $p$ in the structure, to the current cluster if their reachability-distances are not larger than $\epsilon$. This step stops when the first point, say $q$ is found such that its reachability-distance in the structure is larger than $\epsilon$.

5. Increase $clusterID$ by 1.

6. Continue to scan the structure from point $q$, do step 3 to 5 to form the second cluster.

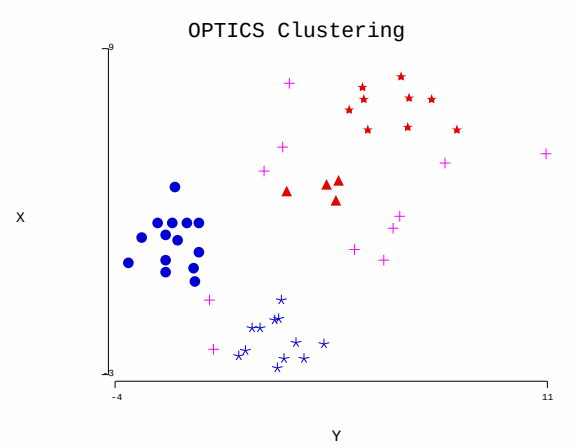


Figure 11: Example of OPTICS clustering ($\epsilon = 1.5$)

7. The algorithm keeps finding new clusters until all the points in the structure have been processed.

8. Output all the clusters. If a point is not in any clrster, it is a noise.

OPTICS is augmented DBSCAN. By using the ordered structure, the computational complexity has been reduced rapidly from $O(NlogN)$ to $O(N)$. Since the clusters found by DBSCAN using a small value of $\epsilon$ are contained in the clusters found by using a relative larger $\epsilon$, if we start from a large $\epsilon$ and reduce its value gradually, a hierarchical clustering structure could be generated based on the ordered clustering structure. However, in OPTICS, this has not been mentioned.

To discover more interesting patterns in the mixture of three multivariate normal data set (shown in figure 8), I implemented the OPTICS algorithm, and use different density levels to extract clusters based on the ordered clustering structure. I constructed the ordered clustering structure using $Mints = 4$ and $\epsilon_0 = 6$. Then I fixed $Minpts$ to be 4 and tried three different values of $\epsilon$: 1.5, 2, 2.5. When $\epsilon = 2$, the result is the same as I obtained in figure 9.

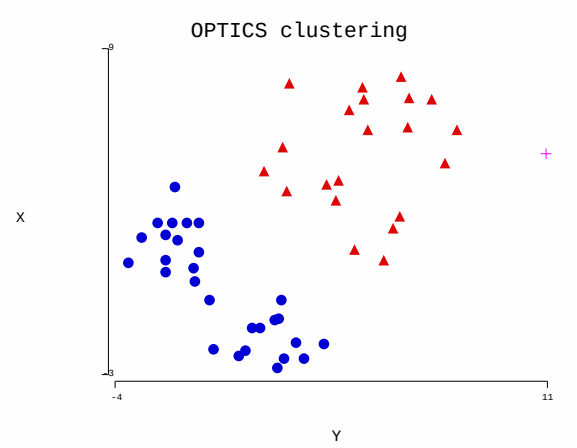


Figure 12: Example of OPTICS clustering ($\epsilon = 2.5$)

Figure 11 shows the result using $\epsilon = 1.5$, Figure 12 shows the result using $\epsilon = 2.5$. These results give us more ideas about the true patterns in the data set. If we use these three clustering results to form a hierarchical clustering tree, it will help us analyze the data more easily.

### 3.3.3 Runt pruning density based clustering

Some earlier proposed density-based clustering methods apply only the global density level to estimate clusters, such as DBSCAN. As described in the subsection 3.3.1, this approach cannot reveal natural clusters in many real data sets in which multiple modes exist with different density levels.

To solve the above problem, runt pruning density based clustering (runt pruning clustering for short) generates a clustering tree by employing the nearest neighbour density estimate on the minimal spanning tree (MST) of the data set [13]. The MST of a data set is a graph used to connect all the vertexes (objects in the data set) with minimal sum of edge length. Besides this, a runt pruning is applied on the clustering tree in order to find the

significant real clusters.

Runt pruning clustering is based on the following ideas or assumptions:

- Regard the data as an iid sample from some unknown probability density $p(x)$
- Regard the groups or clusters corresponding to modes of the density $p(x)$
- The clustering is to find the modes and assign each objects to the domain of attraction of a mode

To make the above ideas more precise, the notion of high density clusters is helpful [15, Section 11].

**Definition 3.18 (level set and high density clusters)** *Define the level set $L(\lambda; p)$ of a density $p$ at level $\lambda$ to be the subset of the feature space as long as the density for each object in this subset exceeds $\lambda$. $L(\lambda; p) = \{x | p(x) > \lambda\}$. Then, the high density clusters at level $\lambda$ are defined to be the maximally connected subsets of $L(\lambda; p)$.*

Since there are only three possibilities for any two high density clusters A and B found by the same or different density levels: $A \supset B$, or $B \supset A$, or $A \cap B = \phi$, it is easy to derive a hierarchical structure when collecting the high density clusters. A clustering tree can be constructed by the following method:

- The root node represents $L(0; p)$, which includes every objects in the data set

- Each node $N$ in the tree represents a maximally connected set $D(N)$, which is a subset of the level set $L(\lambda; p)$, where $\lambda$ is the density level corresponding to the node $N$.
- Each node has either two children or no children, if a node does not have any children, it is a leaf node.
- The two children of a node $N$ can be generated by finding the lowest level $\lambda_d$ such that $L(\lambda_d; p) \cap D(N)$ has at least two connected components (modes). If more than two components have been found, we assign the first one to the first child, and assign the others to the second child.
- If no such $\lambda_d$ could be found, this implies that $D(N)$ has only one mode, therefore $N$ is a leaf node.

To implement the above clustering tree, two problems must be solved.

- How to estimate the density $p$ for the level set $L(\lambda; p)$
- How to derive the maximally connected subsets of $L(\lambda; p)$

Runt pruning clustering has found the connection between the nearest neighbour density estimate and the minimal spanning tree of the data set. This connection suggests a solution to the above problems and has been employed in the runt pruning clustering algorithm to derive the clustering tree.

**Definition 3.19 (nearest neighbour density estimate)** *The nearest neighbour density estimate of an object $y$ denoted as $\widehat{p}(y)$, is defined to be $\widehat{p}(y) = \frac{1}{nVd(y,X)^m}$, where, $n$ is the total number of objects in the data set, $m$ is the dimension of each object in the data set, $V$ is the volume of the unit sphere in $R^m$, $d(y, X)$ is the distance from $y$ to its nearest neighbour. $Vd(y, X)^m$ is*

*the volume of the sphere centered at $y$ with radius $d(y, x_j)$, where $x_j$ is the nearest neighbour of $y$.*

The clustering tree can be built by the connection of the nearest neighbour estimate and MST:

- First, we use the nearest neighbour estimate on each object to determine the estimated level set $L(\lambda; \widehat{p})$. Therefore, given a density level $\lambda$, we obtain the estimated level set by $L(\lambda; \widehat{p}) = \cap_i \overset{o}{S}(x_i, r(\lambda))$. So, $L(\lambda; \widehat{p})$ is the union of open spheres centered at each objects, with radius $r(\lambda)$, where $r(\lambda) = \left(\frac{1}{nV\lambda}\right)^{\frac{1}{m}}$. By definition 3.19, $\widehat{p}(y) > \lambda$, if and only if $d(y, X) < r(\lambda)$. Then, the modes or the maximally connected components can be found from such unions.
- Define $T$ to be the MST of the given data set $X$. A partition of $T$ is obtained by breaking all the edges with length greater than $2r(\lambda)$. Suppose $T$ has been separated into $k$ sub trees, which are $T_1, T_2, ..., T_k$, and the corresponding partition of the data set also separates $X$ into $k$ sub sets, which are $X_1, X_2, ..., X_k$. Based on the above partition, we can derive all the connected subsets of $L(\lambda; \widehat{p})$ by getting the sets $L_j = \cap_{x_i \in X_j} \overset{o}{S}(x_i, r(\lambda))$, where $j = 1, 2, ..., k$. Moreover, $L_i$ and $L_j$ are disconnected if $i \neq j$.

A way to construct the clustering tree is breaking the longest edge of the MST recursively. This approach has been shown to be isomorphic to the single linkage dendrogram [14].

However, as mentioned in [13], the single linkage approach suffers from the following problems:

- The clustering tree tends to generate many small clusters because the longest edges of the MST are always located in lower density regions.

- The tree will have as many leaf nodes as the objects in the data set, therefore this is not a good estimate for the clustering tree of the underlying density.

To solve the above problems , a runt pruning method is then employed on the clustering tree. By the method, some nodes in the clustering tree is cut off to make each remaining node satisfy the minimal runt size.

**Definition 3.20 (runt size of a node in a clustering tree)** *The runt size of a node $N$ in a clustering tree is the number of objects in the cluster corresponding to the smaller child node of $N$.*

Since each node of the single linkage dendrogram corresponds to an edge of the MST, we can also define the runt size for an MST edge $e$.

**Definition 3.21 (runt size of an edge $e$ of the MST)** *Breaking all the MST edges that are as long or longer than $e$, we can find the two sub-trees of the MST rooted at the two MST nodes originally joined by $e$. The runt size of the edge $e$ is defined to be the smaller number of nodes of these two sub-trees.*

The runt size for an MST edge has a useful property: An MST edge with a large runt size implies the presence of multiple modes. It can be verified by the following observation: When constructing an MST, we can initialize each object to be a group, then merge the most two closest groups recursively until only one group is left. During the merging, an edge is added in the MST to connect the two closest objects that are from these two groups respectively. The number of objects in the smaller group is the runt size of this edge. If the underlying density has multiple modes, the initial merges in the above process tend to happen in high-density areas in which the distances between

objects are relatively small. So the MST tree fragments grow first in these regions respectively. When those fragments that probably are representing multiple modes have to be joined by edges, these edges will have relatively large runt sizes.

The algorithm of runt pruning clustering:

1. Choose a threshold for the minimal runt size, which is $r_0$.
2. Compute the MST graph $T$ for the given data set.
3. Construct the clustering tree by the following process:

- Let the root node corresponding to the whole MST. This is the same as use $L(0; p)$ to represent the root node.
- Let $T(N)$ be the sub-tree, which corresponds to the node $N$, in $T$. Obtain the lowest density level $\lambda_d$ for each node $N$ in the tree by finding the longest edge $e$ in $T(N)$ with runt size no less than the threshold $r_0$, if it could be found, the two sub trees rooted at the node N are obtained by breaking the edge $e$ of $T(N)$.
- Repeat the above process until no children can be found for all the leaf nodes in the clustering tree.

Before doing the runt pruning clustering, we should figure out how to choose the threshold of minimum runt size. Since the runt size for each edge in the MST graph can be calculated beforehand, we rank these runt sizes in the descending order and then find the first big gap. It gives us the suggestion about the minimum runt size.

The runt pruning clustering is effective for the data sets with multiple density modes. However, this approach relies on the nearest neighbour density

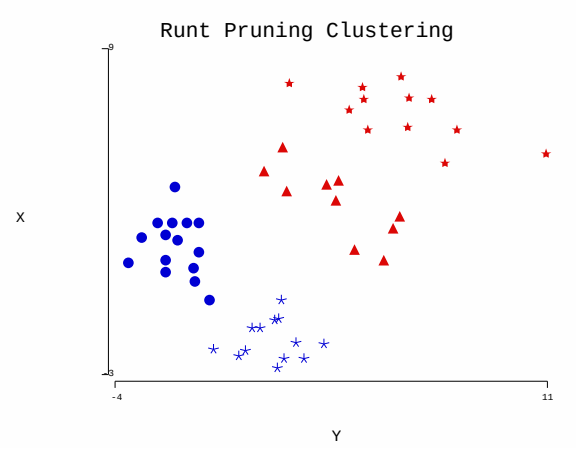


Figure 13: Example of the runt pruning clustering

estimate. Some better density estimates, such as $k^{th}$ neatest neighbour estimate, could not be used in this algorithm directly. Besides, the computational complexity of funding the MST dominants the total cost of the runt pruning clustering. Although some efficient algorithms could be employed to reduce the computing cost of MST to be $O(NlogN)$, it is still not practical for very large high dimensional data sets.

I implemented the runt pruning clustering algorithm and applied this method on the mixture data set (see figure 8). First, I built the MST of the data. Then, I ranked the runt size of edges in the MST. The first 6 runt sizes are: 21 13 10 5 4 4. Since there is a big gap from 10 to 5, I used 5 to be the minimum runt size to build the clustering tree (Although the first big gap is from 21 to 13, since I want see more patterns and compare the result with OPTICS, I chose 10). Figure 13 shows all the four groups found by the clustering tree. The clustering tree is shown in figure 14. Runt pruning clustering found the same result as OPTICS did, and the clustering tree gives us more clear ideas of the patterns in the data.

There is some connection in the density estimation between runt pruning

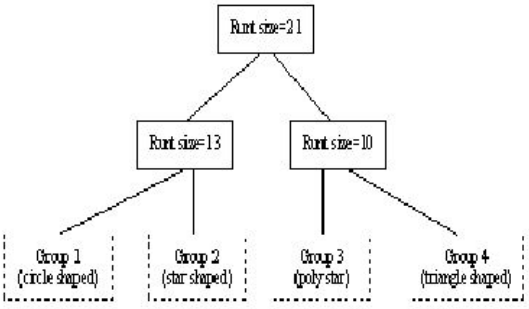


Figure 14: The clustering tree built by the runt pruning clustering

clustering and DBSCAN or OPTICS. In DBSCAN or OPTICS, the density is estimated by two parameters: $\epsilon$ and Minpts. If we set $Minpts = 2$, and $\epsilon = r(\lambda) = \left(\frac{1}{nV\lambda}\right)^{\frac{1}{m}}$, DBSCAN or OPTICS will also find all the maximally connected sub-sets of $L(\lambda; p)$. So, if we combine OPTICS with runt pruning and construct a hierarchical clustering tree, we will have the same result as the runt pruning clustering algorithm will do.

# 4 Conclusion

Among different ways to measure dissimilarities, the paper focuses on using metric functions. Getting the data sets from the metric spaces or reducing the dimensions of data sets in the metric spaces are necessary for many clustering methods. Multidimensional scaling (MDS) is a possible solution.

In this paper, three major categories of clustering methods are discussed. They are partitioning, hierarchical and density-based clustering methods.

Four typical partitioning methods are discussed: k-means, PAM, CLARA and CLARANS. All these methods use a centroid to represent a cluster. K-means uses the cluster mean and the others use the medoid. CLARA is PAM on samples of the dataset, and CLARANS reduces the search graph of PAM by sampling on the search graph.

BIRCH, CURE and CHAMELEON are three hierarchical clustering methods proposed resently. BIRCH builds a $CF$ tree to find clustes. It uses $CF$s to represent clusters and support incremental clustering. CURE uses several representatives to represent a cluster, and does a single linkage clustering on only these representatives of each cluster. CHAMELEON does hierarchical clustering on the partitions of the k-nearest neighbour graph. It uses a dynamic merging policy, which consider both the dissimilarity between two

clusters and the similarity within a possible cluster.

The relations among DBSCAN, OPTICS and runt pruning clustering are high. OPTICS is augmented DBSCAN. It builds an ordered clustering structure for running DBSCAN efficiently. It could also build a hierarchical clustering structure like runt pruning clustering does. If we set $Minpts = 2$ and $\epsilon = r(\lambda) = \left(\frac{1}{nV\lambda}\right)^{\frac{1}{m}}$, DBSCAN or OPTICS will also find all the maximally connected sub-sets of $L(\lambda; p)$. Runt pruning clustering was proposed to find such sub-sets. The difference is that, runt pruning clustering uses a minimum runt size to be a threshold on cutting the minimum spanning tree to obtain more reasonable clusters.

Not only some clustering methods but also the different categories of clustering methods are related. For example, hierarchical methods find a nested sequence of clustering, each clustering is a partition of the data, and so, in this sense, a hierarchical clustering is a nested sequence of partitions. The density based clustering algorithms use density estimation to partition the data.

I implemented the algorithms for non-metric MDS, DBSCAN, OPTICS and runt pruning clustering. The purpose of this work is to find out exactly how they work so as to lay the groundwork for future research.

More research work could be done in the future based on this paper. For MDS methods, only one stress is considered in the paper. There exist different ways to measure stress, which might also be investigated in the MDS approach. As mentioned in [13], some better density estimates, such as the $k^{th}$ nearest neighbour estimate might be used, however Stuetzle [13] did not find them efficient for direct using in the runt pruning clustering algorithm. However the relation between DBSCAN, OPTICS and runt pruning clustering has been found, it might be possible to combine the $k^{th}$ nearest neighbour estimate with runt pruning by using the DBSCAN/OPTICS approach. Fi-

nally, some other density estimates , which are more flexible, could also be considered to obtain better clustering.

# References


[1] Seber, 1984. *Multivariate Observations*, John Wiley & Sons.

[2] W. Ledermann(ed.), 1984. *Handbook of Applicable Mathematics: Vol. 6 Part B Statistics (ed. E Lloyd)*, Wiley-Interscience, Section 1.7.7, page 747.

[3] R.N. Shephard, 1962. *The Analysis of Proximities:Multidimensional Scaling with an Unknown Distance Function*, Psychometrika, 27, 125-140.

[4] J.B. Kruskal, 1964. *Nonmetric multidimensional Scaling: a numerical method*, Psychometrika, 29, 115-129.

[5] J. MacQueen, 1967. *Some Methods for Classification and Analysis of Multivariate Observation. Proc. 5th Berkeley Symp. Math. Statist, Prob.,1*, 281-297.

[6] L. Kaufman and P.J. Rousseeuw, 1990. *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons.

[7] R.T. Ng and J. Han, 2002. *CLARANS: A Method for Clustering Objects for Spatial Data Mining*, IEEE transactions on knowledge and data engineering, Vol. 14, No. 5.

[8] T. Zhang, R. Ramakrishnan, M. Livny, 1996. *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, SIGMOD'96 Montreal, Canada.

[9] S. Guha, R. Rastogi, K. Shim, 1998. *CURE: An Efficient Clustering Algorithm for Large Databases*, SIGMOD'98 Seattle, USA.

[10] G. Karypis, E. Han, V. Kumar, 1999. *CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling*, IEEE Computer, 32(8) 68-75.

[11] M.Ester, H. Kriegel, J. Sander, X. Xu, 1996. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Proc. Of KDD-96.

[12] M. Ankerst, M. Breunig, H. Kriegel, J. Sander, 1999. *OPTICS: Ordering Points To Identify the Clustering Structure*, Proc. ACM SIGMOD'99 Int. Conf. On Management of Data, Philadelphia USA.

[13] W. Stuetzle, 2003. *Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample*, Journal of classification, Vol 20, No. 5, 2003, 25-47..

[14] J.C. Gower and G. Ross, 1969. *Minimal Spanning Tree and Single Linkage Cluster Analysis*, Applied Statistics, 18, 54-64.

[15] J. Hartigan, 1975. *Clustering Algorithms*, Wiley.