Clustering algorithms

Konstantinos Koutroumbas

Unit 9

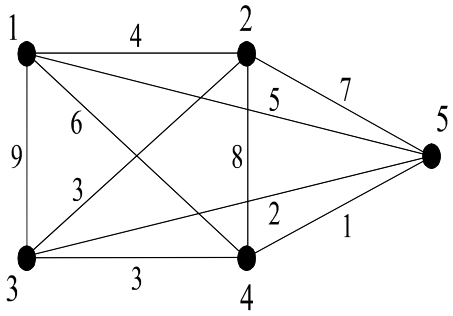
– Divisive clustering algorithms
– Hierarchical alg. For large data sets (CURE, ROCK, Chameleon)

koutroum@noa.gr

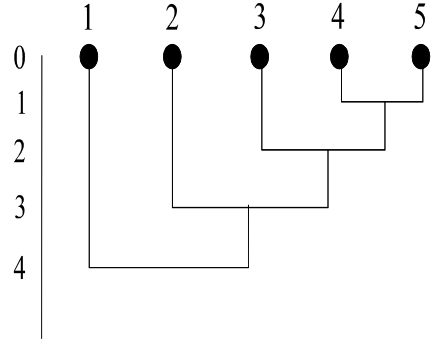
Agglomerative graph theory based Clustering Algorithms

- Ties in the proximity matrix
  - SL produces the same hierarchy of clusterings, independently of the order of consideration of edges with equal weights.
  - CL may produce different hierarchies, depending on the order of consideration of edges with equal weights.
  - The other graph theory-based algorithms behave as the CL.
  - The same trend appears in the matrix-based algorithms. In this case, *ties may appear at a later stage of the algorithm*.
- Example 6: Let

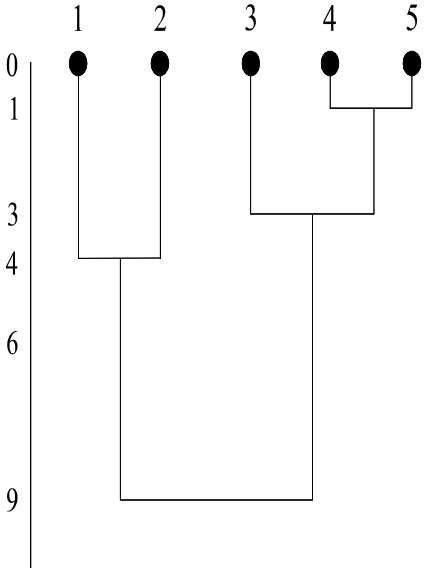
$$P = \begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 3 & 8 & 7 \\ 9 & 3 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix}$$



(a)

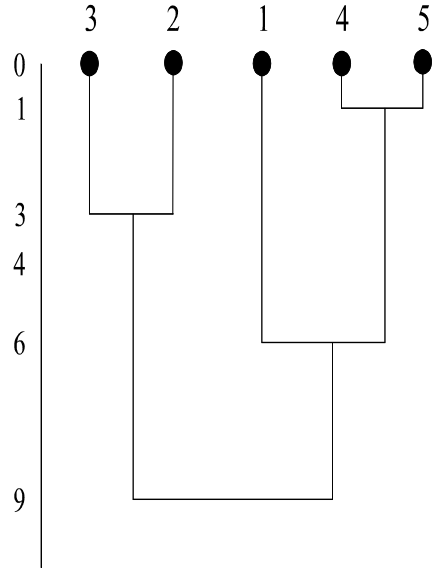


(b)



(c)

(CL(a))



(d)

(CL(b))

Note that $P(2,3) = P(3,4)$.

Agglomerative Clustering Algorithms: Cophenetic matrix

This is an alternative way to **represent** a hierarchical clustering.

Cophenetic distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, $d_C(\boldsymbol{x}_i, \boldsymbol{x}_j)$: The proximity level, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are found in the same cluster for the first time (distance metric).

Cophenetic matrix: An $N \times N$ matrix containing the **cophenetic distances** associated with all pairs of data vectors.

Example: Consider the following dissimilarity matrix (Euclidean distance)

$$P_0 = \begin{bmatrix} 0 & 1 & 2 & 26 & 37 \\ 1 & 0 & 3 & 25 & 36 \\ 2 & 3 & 0 & 16 & 25 \\ 26 & 25 & 16 & 0 & 1.5 \\ 37 & 36 & 25 & 1.5 & 0 \end{bmatrix}$$

The associated **cophenetic matrix** is

$$D_C = \begin{bmatrix} 0 & 1 & 2 & 16 & 16 \\ 1 & 0 & 2 & 16 & 16 \\ 2 & 2 & 0 & 16 & 16 \\ 16 & 16 & 16 & 0 & 1.5 \\ 16 & 16 & 16 & 1.5 & 0 \end{bmatrix}$$

The results of the **single link** algorithm are (in parenthesis the proximity level where the associated clustering has been formed):

$\mathcal{R}_0 = \{\{\boldsymbol{x}_1\}, \{\boldsymbol{x}_2\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_4\}, \{\boldsymbol{x}_5\}\}$, $(\mathbf{0})$

$\mathcal{R}_1 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_4\}, \{\boldsymbol{x}_5\}\}$, $(\mathbf{1})$

$\mathcal{R}_2 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5\}\}$, $(\mathbf{1.5})$

$\mathcal{R}_3 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5\}\}$, $(\mathbf{2})$

$\mathcal{R}_4 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}\}$, $(\mathbf{16})$

Divisive Clustering Algorithms

- Let $g(C_i, C_j)$ be a **dissimilarity** function between two clusters.
- Let C_{tj} denote the j-th cluster of the t-th clustering $\Re_t$, $t = 0, \ldots, N-1$, $j = 1, \ldots, t+1$.

Generalized Divisive Scheme (GDS)

- Initialization
  - Choose $\Re_0 = \{X\}$ as the initial clustering.
  - $t = 0$
- **Repeat**
  - $t = t + 1$
  - For $i = 1$ to t
    - **Among** all possible pairs of clusters (C_r, C_s) that form a partition of $C_{t-1,i}$, **find** the pair $(C^1{}_{t-1,i}, C^2{}_{t-1,i})$ that gives the max. value for g.
  - End for
  - From the t pairs defined in the previous step, choose the one that **maximizes** g. Suppose that this is $(C^1{}_{t-1,j}, C^2{}_{t-1,j})$.
  - The new clustering is:

$$\Re_t = (\Re_{t-1} - \{C_{t-1,j}\}) \cup \{C^1{}_{t-1,j}, C^2{}_{t-1,j}\}$$

  - **Relabel** the clusters of $\Re_t$.
- **Until** each vector lies in a single cluster.

Divisive Clustering Algorithms

Remarks:

- Different choices of g give rise to different algorithms.
- The GDS is computationally very demanding even for small N.
- Algorithms that rule out many partitions as not "reasonable", under a pre-specified criterion, have also been proposed.
- Algorithms where the splitting of the clusters is based on all features of the feature vectors are called polythetic algorithms. Otherwise, if the splitting is based on a single feature at each step, the algorithms are called monothetic algorithms.

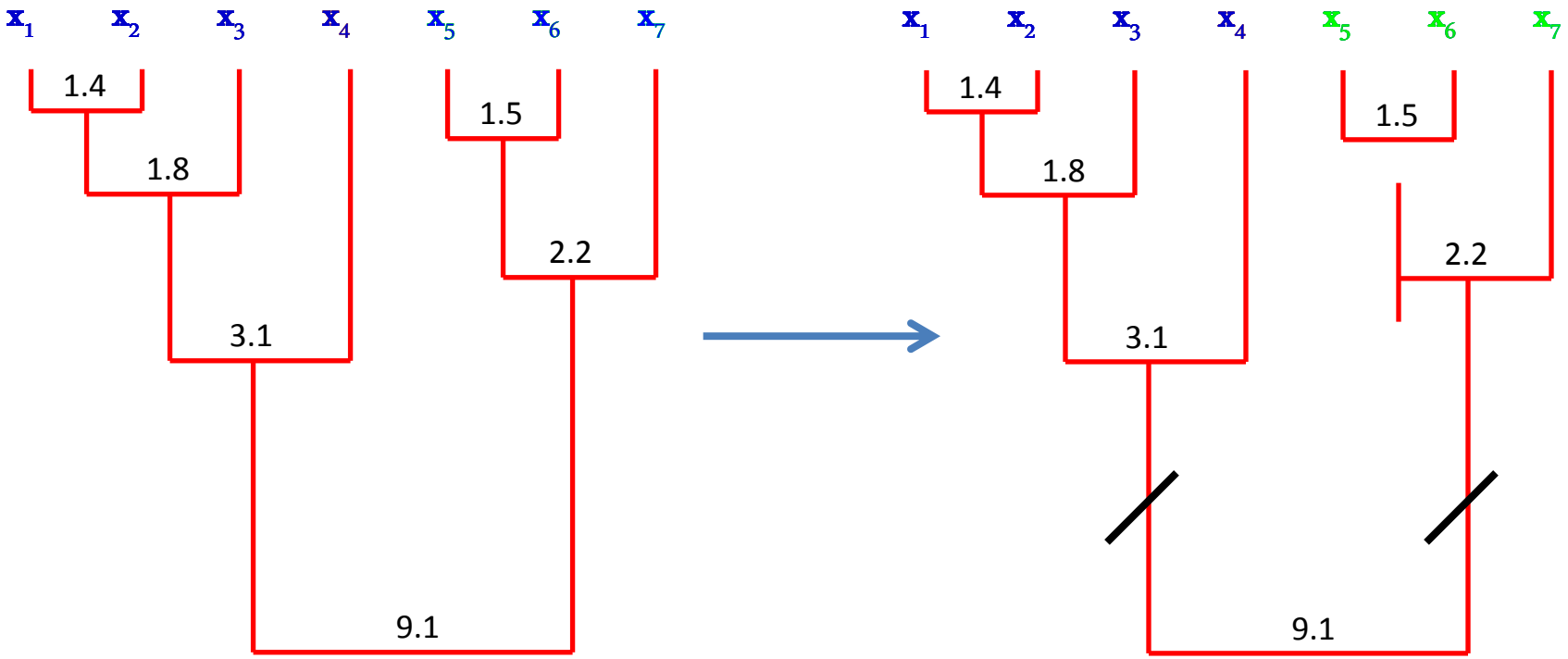
Choice of the best number of clusters

A major issue associated with hierarchical algorithms is:

"How the clustering that best fits the data is extracted from a hierarchy of clusterings?"

Some **approaches**:

- Search in the proximity dendrogram for clusters that have a large lifetime (the difference between the proximity level at which a cluster is created and the proximity level at which it is absorbed into a larger cluster (however, this method involves human subjectivity)).



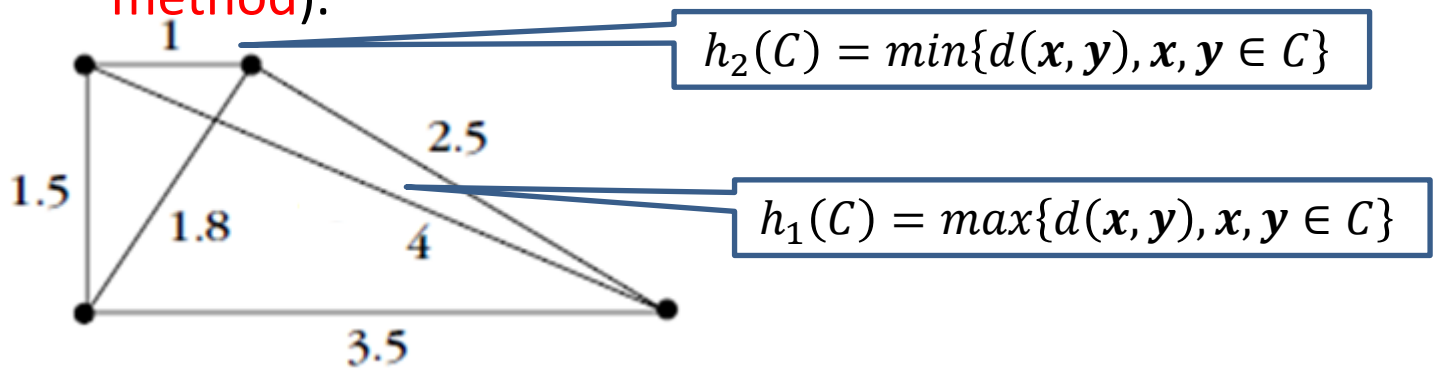
Choice of the best number of clusters

A major issue associated with hierarchical algorithms is:

"How the clustering that best fits the data is extracted from a hierarchy of clusterings?"

Some **approaches**:

- Define a function $h(C)$ that measures the dissimilarity between the vectors of the same cluster C ("self-dissimilarity"). Then, we have two alternatives:
  - Let θ be an appropriate threshold for $h(C)$. Then $\Re_t$ *is the final clustering if there exists a cluster* C *in* $\Re_{t+1}$ *with* $h(C) > \theta$ (extrinsic method).



  - If $\theta = \mu + \lambda\sigma$, where μ is the average distance of any two vectors of X and σ is the associated standard deviation, then the need for specifying an appropriate value of θ is **transferred** to the choice of an appropriate value for λ.

Choice of the best number of clusters

A major issue associated with hierarchical algorithms is:

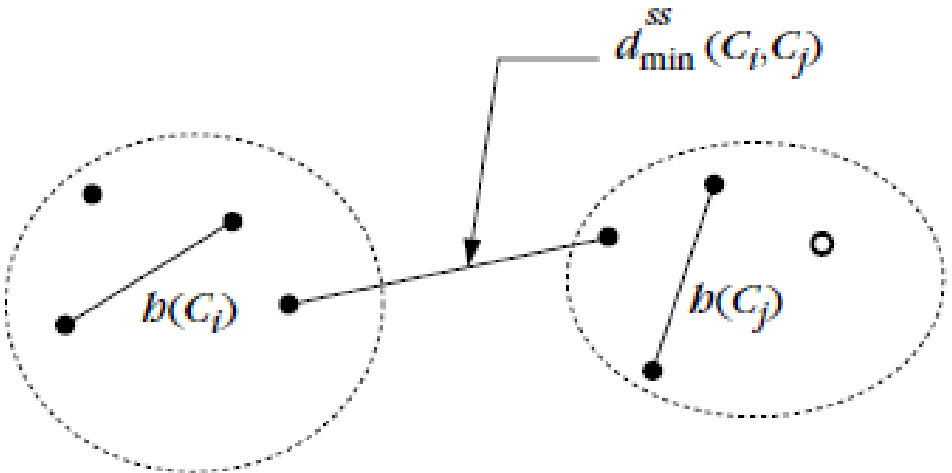
"How the clustering that best fits the data is extracted from a hierarchy of clusterings?"

Some **approaches**:

- Define a function $h(C)$ that measures the dissimilarity between the vectors of the same cluster C ("self-dissimilarity"). Then, we have two alternatives:
  - The final clustering $\mathcal{R}_t$ must satisfy the following condition:

$$d^{ss}{}_{min}(C_i, C_j) > max\{h(C_i), h(C_j)\}, \qquad \forall C_i, C_j \in \mathcal{R}_t$$

In words, in the final clustering, the dissimilarity between every pair of clusters is larger than the "self-dissimilarity" of each one of them (intrinsic method).



Hierarchical Algorithms for large data sets

Remark:

Since the number of operations required by GAS is greater than $O(N^2)$, algorithms resulting by GAS are prohibitive for very large data sets encountered, for example, in web mining and bioinformatics. To overcome this drawback, various hierarchical algorithms of special type have been developed that are tailored to handle large data sets.

Typical examples are:

- The CURE algorithm.
- The ROCK algorithm.
- The Chameleon algorithm.

The CURE (Clustering Using Representatives) algorithm

In CURE:

- Each cluster C is represented by a set, R_C, of $k > 1$ representatives.
- These representatives try to "**capture**" the "shape " of the cluster.
- They are **chosen** at the "border" of the cluster and then, they are **pushed** toward its mean, in order to **discard** the **irregularities** of the **border**.

- **Determination** of R_C:
  - Select $\boldsymbol{x} \in C$, with the maximum distance from the mean $\boldsymbol{m}_C$ of C and set $R_C = \{\boldsymbol{x}\}$
  - For $i = 2$ to $\min\{k, n_C\}$ (n_C is the number of points in C)
    - Determine $\boldsymbol{y} \in C - R_C$ that lies farthest from the points of R_C and set $R_C = R_C \cup \{\boldsymbol{y}\}$.
  - Shrink the points $\boldsymbol{x} \in R_C$ toward the mean $\boldsymbol{m}_C$ in C by a factor $a \in (0,1)$. That is $\boldsymbol{x} = (1-a)\,\boldsymbol{x} + a\,\boldsymbol{m}_C$, $\forall \boldsymbol{x} \in R_C$.

CURE is a **special case** of GAS (single link) where the distance between two clusters is defined as: $d(C_i, C_j) = min_{\boldsymbol{x} \in R_{C_i}, \boldsymbol{y} \in R_{C_j}} d(\boldsymbol{x}, \boldsymbol{y})$

The CURE (Clustering Using Representatives) algorithm

Clustering Using REpresentatives ($CURE(X)$)

- Initialization
  - **Choose** $\Re_0 = \{\{\boldsymbol{x}_1\}, \ldots, \{\boldsymbol{x}_N\}\}$
  - $t = 0$
- **Repeat**
  - $t = t + 1$
  - **Choose** (C_i, C_j) in $\Re_{t-1}$ such that

$$d(C_i, C_j) = min_{r,s} d(C_r, C_s)$$

  - **Define** $C_q = C_i \cup C_j$ and produce $\Re_t = \left(\Re_{t-1} - \{C_i, C_j\}\right) \cup \{C_q\}$
  - **Determine** R_{C_q} (*)
- **Until** all vectors lie in a single cluster.

$d(C_r, C_s) = min_{\boldsymbol{x} \in R_{C_r}, \boldsymbol{y} \in R_{C_s}} d(\boldsymbol{x}, \boldsymbol{y})$

(*) The **determination** of R_{C_q} may be conducted:

(i) Either by performing the procedure of the previous slide taking into account **all the data points** of C_q (more accurate but slower approach).

(ii) Or by performing the procedure of the previous slide taking into account the **data points** in $R_{C_i} \cup R_{C_j}$ (the union of the representatives of the clusters that constitute C_q) (less accurate but faster approach).

The CURE (Clustering Using Representatives) algorithm

- Worst case time complexity for CURE: $O(N^2 \, log_2 \, N)$.
- This is prohibitive for very large data sets.
- Solution: Adoption of the random sampling technique.
  The size N' of a sample data set X', created from X, via random sampling, should be sufficiently large in order to ensure that the probability of missing a cluster due to sampling is low.

The CURE (Clustering Using Representatives) algorithm

Clustering Using Representatives- Random Sampling (*CURE-RS*(X))

Identification of clusters

- **Partition** randomly X into $p = N/N'$ sample data sets, $X_1, X_2, \ldots, X_p$.
- For $i = 1$ to p
  - **Run** *CURE-RS*(X_i) and **return** the ${\Re_k}^i$ clustering with N'/q clusters (at the most) (q is user-defined).
- End – For
- **Set** $X' = {\Re_k}^1 \cup {\Re_k}^2 \cup \cdots \cup {\Re_k}^p$
- **Run** *CURE*(X') and determine the most appropriate clustering ${\Re_m}'$.

Only the k representatives from each cluster are **considered**.

The algorithm starts from the ${\Re'}_{p*\left(\frac{N'}{q}\right)} \left(\equiv {\Re'}_{\frac{N}{q}}\right)$

and ends with the ${\Re_m}'$ clustering

Assignment of points to clusters

- For each of the m clusters of ${\Re_m}'$ **select** a random sample of k representative points.
- **Assign** each point $\boldsymbol{x}$ that is not cluster representative to the cluster that contains the representative closest to it.

The CURE (Clustering Using Representatives) algorithm

Remarks:

- CURE is sensitive to the parameters k, N', a. Specifically:
  - k must be large enough to capture the geometry of each cluster.
  - N' must be higher than a certain percentage of N (typically $N' \geq 2.5\%\ N$)
  - For small a CURE **behaves** like the single-link algorithm, while for large a it resembles the algorithms that use a single point representative for each cluster.
- Worst case time complexity for CURE using random sampling: $O(N'^2 \log_2 N')$
- The algorithm exhibits low sensitivity to outliers within the clusters.
- A few stages before its termination, CURE **checks** for clusters containing very few data points and removes them (since they are likely to be outliers).
- If N'/q is sufficiently large, compared to m, it is expected that the partition of X will not affect significantly the final clustering obtained by CURE.
- CURE can, in principle, **reveal** clusters of non-spherical or elongated shapes, as well as clusters of wide variance in size.
- CURE can be implemented efficiently using the *heap* and the *k-d tree* data structures.

The ROCK (RObust Clustering using linKs) algorithm

It is best suited for **nominal** (**categorical**) features.

- Some preliminaries
  - Two points $\boldsymbol{x}, \boldsymbol{y} \in X$ are considered neighbors if $s(\boldsymbol{x}, \boldsymbol{y}) \geq \theta$, where $s(.)$ is a similarity function and θ a user-defined similarity threshold between two vectors ($0 \leq s(\boldsymbol{x}, \boldsymbol{y}) \leq 1$ and, consequently, $0 \leq \theta \leq 1$).
  - $link(\boldsymbol{x}, \boldsymbol{y})$ is the number of common neighbors between $\boldsymbol{x}$ and $\boldsymbol{y}$.

In the graph whose vertices correspond to data points and edges **connect** neighboring points, $link(\boldsymbol{x}, \boldsymbol{y})$ is the **number** of **distinct paths** of **length 2** that connect $\boldsymbol{x}, \boldsymbol{y}$.

- Assumption: There **exists** a function $f(\theta)$ (< 1) such that:
  "Each point assigned to a cluster C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i (n_i is the number of points in C_i) "

It can be proved that the expected total number of links **among** all pairs in C_i is $n_i^{1+2f(\theta)}$.

$$link(C_i) = \sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{y} \in C_i} link(\boldsymbol{x}, \boldsymbol{y})$$

The ROCK (RObust Clustering using linKs) algorithm

- ROCK is a **special case** of GAS where
  - The closeness between two clusters is defined as

$$link(C_i, C_j) = \sum_{x \in C_i} \sum_{y \in C_j} link(x, y)$$

$$g(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

The denominator is the expected total number of links *between* the two clusters.

The larger the $g(\cdot)$, the more similar the clusters C_i and C_j *are* .

- The stopping criterion is:
  - the number of clusters becomes equal to a predefined number m *or*
  - $link(C_i, C_j) = 0$ for every pair in a clustering $\mathcal{R}_t$.

- Time complexity for ROCK: Similar to CURE for large N.
- Prohibitive for very large data sets.
- *Solution:* **Adoption** of random sampling techniques.

The ROCK (RObust Clustering using linKs) algorithm

➢ROCK utilizing Random Sampling

- Identification of clusters
  - –Select a subset X' of X via random sampling
  - –Run the original ROCK algorithm on X'
- Assignment of points to clusters
  - –For each cluster C_i select a set L_i of n_{L_i} points
  - –For each $\boldsymbol{z} \in X - X'$
    - o**Compute** $t_i = N_i/(n_{L_i}+1)^{f(\theta)}$, where N_i is the no of neighbors of $\boldsymbol{z}$ in L_i.
    - o**Assign** $\boldsymbol{z}$ to the cluster with the maximum t_i.

Remarks:

- A choice for $f(\theta)$ is $f(\theta) = (1-\theta)/(1+\theta)$, with $(\theta < 1)$.
- $f(\theta)$ **depends** on the data set and the type of clusters we are interested in.
- The hypothesis about the existence of $f(\theta)$ is very strong. It may lead to poor results if the data do not satisfy it.
- It can be used for discrete-valued data sets.

The ROCK (RObust Clustering using linKs) algorithm

An application:

- Grouping the customers of supermarket according to their purchases.
- Each customer (entity) is represented by the set of goods he/she buys (categorical data representation).
- The similarity between two customers may be quantified via the **Jaccard coefficient**

For two finite sets T_i and T_j, the **Jaccard coefficient** is defined as

$$J(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}$$

- For example, assuming that $T_1 = \{A, B, C\}, T_2 = \{A, B, D\}, T_3 = \{A, B, D, E\}$ are the sets corresponding to three customers, it is

$$J(T_1, T_1) = \frac{3}{3} = 1, \quad J(T_1, T_2) = \frac{2}{4} = 0.5, \quad J(T_1, T_3) = \frac{2}{5} = 0.4,$$
$$J(T_2, T_3) = \frac{3}{4} = 0.75$$

Choosing $\theta = 0.45$, T_1 and T_2 are neighbors, T_2 and T_3 are neighbors but T_1 and T_3 are not neighbors. However, T_1 and T_3 share a common neighbor.

- For this application, a good choice for $f(\theta)$ is $f(\theta) = (1 - \theta)/(1 + \theta)$, with $(\theta < 1)$.

The ROCK (RObust Clustering using linKs) algorithm

Example: Consider a three-cluster clustering $\{C_1, C_2, C_3\}$, where the number of points in each one of them is $n_1 = 500$, $n_2 = 500$ and $n_3 = 100$, respectively.

$$g(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Define $f(\theta)$ as $f(\theta) = \frac{1-\theta}{1+\theta}$, with $\theta = \frac{1}{3}$.

Let $link(C_1, C_2) = 100$ and $link(C_1, C_3) = 100$.

Compute $g(C_1, C_2)$ and $g(C_1, C_3)$ and draw your conclusions

Answer: It is $1 + 2f(\theta) = 1 + 2\frac{1-\theta}{1+\theta} = 1 + 2\frac{1-\frac{1}{3}}{1+\frac{1}{3}} = 2$,

$$(n_1 + n_2)^{1+2f(\theta)} - n_1^{1+2f(\theta)} - n_2^{1+2f(\theta)} = (500 + 500)^2 - 500^2 - 500^2 = 500000$$

$$(n_1 + n_3)^{1+2f(\theta)} - n_1^{1+2f(\theta)} - n_3^{1+2f(\theta)} = (500 + 100)^2 - 500^2 - 100^2 = 100000$$

Then $g(C_1, C_2) = \frac{100}{500000} = 0.0002$ and $g(C_1, C_3) = \frac{100}{100000} = 0.001$

Thus, among the clusters that have the same degree of similarity with C_1 wrt the $link(.)$ criterion, according to the normalized link criterion ($g(\cdot)$) C_1 is more similar with the smallest cluster (C_3), and not with the equally sized C_3.

The Chameleon algorithm

- This algorithm is not based on a "static" modeling of clusters like CURE (where each cluster is represented by the same number of representatives) and ROCK (where constraints are posed through the function $f(\theta)$).
- It enjoys both divisive and agglomerative features.

- *Some preliminaries:*

  Let $G = (V, E)$ be a graph where:

  - each vertex of V **corresponds** to a data point in X.
  - E is a set of edges connecting pairs of vertices in V. Each edge is weighted by the **similarity** of the corresponding points.

  - Edge cut set: Let C be a set of points corresponding to a subset of V. Assume that C is partitioned into two nonempty sets C_i and C_j. The subset E'_{ij} of the edges of E that connect points of C_i with points of C_j is called edge cut set.

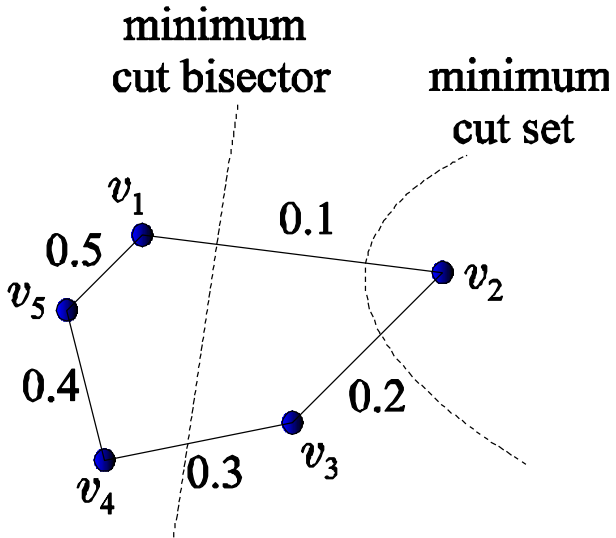
The Chameleon algorithm

- Minimum cut set: Let C be a set of points corresponding to a subset of V. If $|E'_{ij}| = min_{(C_u, C_v):\, C_u \cup C_v = C} |E'_{uv}|$, then (C_i, C_j) is the minimum cut set of C,

  where $|E'_{uv}|$ be the sum of weights of the edges in E'_{uv}.

  - Minimum cut bisector: If C_i, C_j are constrained to be of approximate equal size, the minimum cut set (over all possible partitions of approximately equal size) is known as the minimum cut bisector.

Example: The graph in the following figure consists of the 5 vertices and the edges shown, each one weighted by the similarity of the points that correspond to the vertices it connects. The minimum cut set and the minimum cut bisector are shown.



The Chameleon algorithm

Measuring the similarity between clusters

Relative interconnectivity:

– Let E_{ij} be the set of edges **connecting** points in C_i with points in C_j.

– Let E_i be the set of edges **corresponding** to the minimum cut bisector of C_i.

– Let $|E_i|$, $|E_{ij}|$ be the **sum** of the weights of the edges of E_i, E_{ij}, respectively.

– Absolute interconnectivity between C_i, $C_j = |E_{ij}|$

– Internal interconnectivity of $C_i = |E_i|$

– Relative interconnectivity between C_i, C_j:

$$RI_{ij} = \frac{|E_{ij}|}{\frac{|E_i| + |E_j|}{2}}$$

Relative closeness:

– Let S_{ij} be the **average** weight of the edges in E_{ij} .

– Let S_i be the **average** weight of the edges in E_i .

– Relative closeness between C_i and C_j:

$$RC_{ij} = \frac{S_{ij}}{\frac{n_i}{n_i + n_j} S_i + \frac{n_j}{n_i + n_j} S_j}$$

n_i, n_j: Number of points in C_i, C_j, resp.

The Chameleon algorithm

The Chameleon algorithm

Preliminary phase

Create a k-nearest neighbor graph $G = (V, E)$ such that:

- Each vertex of V corresponds to a data point.
- The edge between two vertices v_i and v_j is added to E if v_i is one of the k-nearest neighbors of v_j or vise versa.
- Each connected component of the resulting graph is **associated** with a cluster. Let $\Re$ be the clustering consisting of these clusters.

Divisive phase

Set $\mathcal{R}_0 = \mathcal{R}$

$t = 0$

Repeat

- $t = t + 1$
- **Select** the largest cluster C in $\mathcal{R}_{t-1}$.
- Referring to E, **partition** C into two sets so that:
  - the sum of the weights of the edge cut set between the resulting clusters is minimized.
  - each cluster contains at least 25% of the vertices of C.

Until each cluster in $\mathcal{R}_t$ **contains** fewer than q points.

The Chameleon algorithm

The Chameleon algorithm (cont)

Agglomerative phase

Set $\mathcal{R}'_0 = \mathcal{R}_t$

$t = 0$

Repeat

- $t = t + 1$
- **Merge** C_i, C_j in $\mathcal{R}'_{t-1}$ to a single cluster **if**

$$RI_{ij} \geq T_{RI} \text{ and } RC_{ij} \geq T_{RC} \quad \textbf{(A)}$$

(if more than one C_j satisfy the conditions for a given C_i, the C_j with the highest $|E_{ij}|$ is selected).

Until (A) does not hold for any pair of clusters in $\mathcal{R}'_{t-1}$.

Return $\mathcal{R}'_{t-1}$

NOTE: The internal structure of two clusters to be merged is of significant importance. The more similar the elements within each cluster the higher "their resistance" in merging with another cluster.

The Chameleon algorithm

Remarks:

- Condition **(A)** can be replaced by $(C_i, C_j) = max_{(C_u, C_v)} RI_{uv} \cdot RC_{uv}{}^{a}$

- Chameleon is not very sensitive to the choice of the user-defined parameters k (typically it is selected between 5 and 20), q (typically chosen in the range 1% to 5% of the total number of data points), T_{RI}, T_{RC} and/or a.

- Chameleon is well suited for **large data sets** (more accurate estimation of $|E_{ij}|, |E_i|, S_{ij}, S_i$)

- For **large** N, the worst-case time complexity of the algorithm is $O(N(\log_2 N + m))$, where m is the number of clusters formed by the divisive phase.

The Chameleon algorithm

Example: For the clusters shown in the figure we have:

$|E_1| = 0.48,\ |E_2| = 0.48,$

$|E_3| = 1.45,\ |E_4| = 1.45,$

$|S_1| = 0.48,\ |S_2| = 0.48,$

$|S_3| = 0.725,\ |S_4| = 0.725,$

$|E_{12}| = 0.4,\ |E_{34}| = 0.6,$

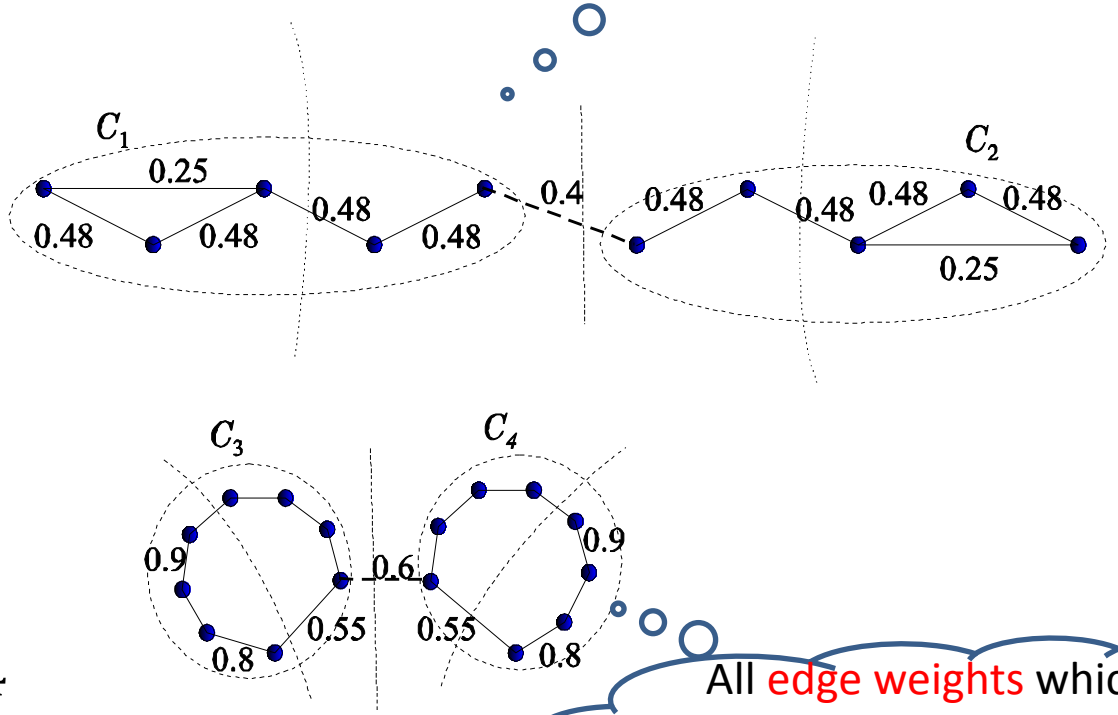
$|S_{12}| = 0.4,\ |S_{34}| = 0.6.$

Thus,

$RI_{12} = 0.833,\ RI_{34} = 0.414$

$RC_{12} = 0.833,\ RC_{34} = 0.828$

The values in the figure stand for **similarities**.



All edge weights which are not denoted explicitly are equal to 0.9.

In conclusion: Both RI and RC **favor** the merging C_1 and C_2 **against** the merging of C_3 and C_4.

Note that the single-link algorithm would **merge** C_3 and C_4 instead of C_1 and C_2.