Clustering algorithms

Konstantinos Koutroumbas

Unit 8

- Agglomerative hierarchical clustering algorithms based on:
  - Matrix theory (cont.)
  - Graph theory
- Hierarchical clust. Algorithms for large data sets (CURE algorithm)

koutroum@noa.gr

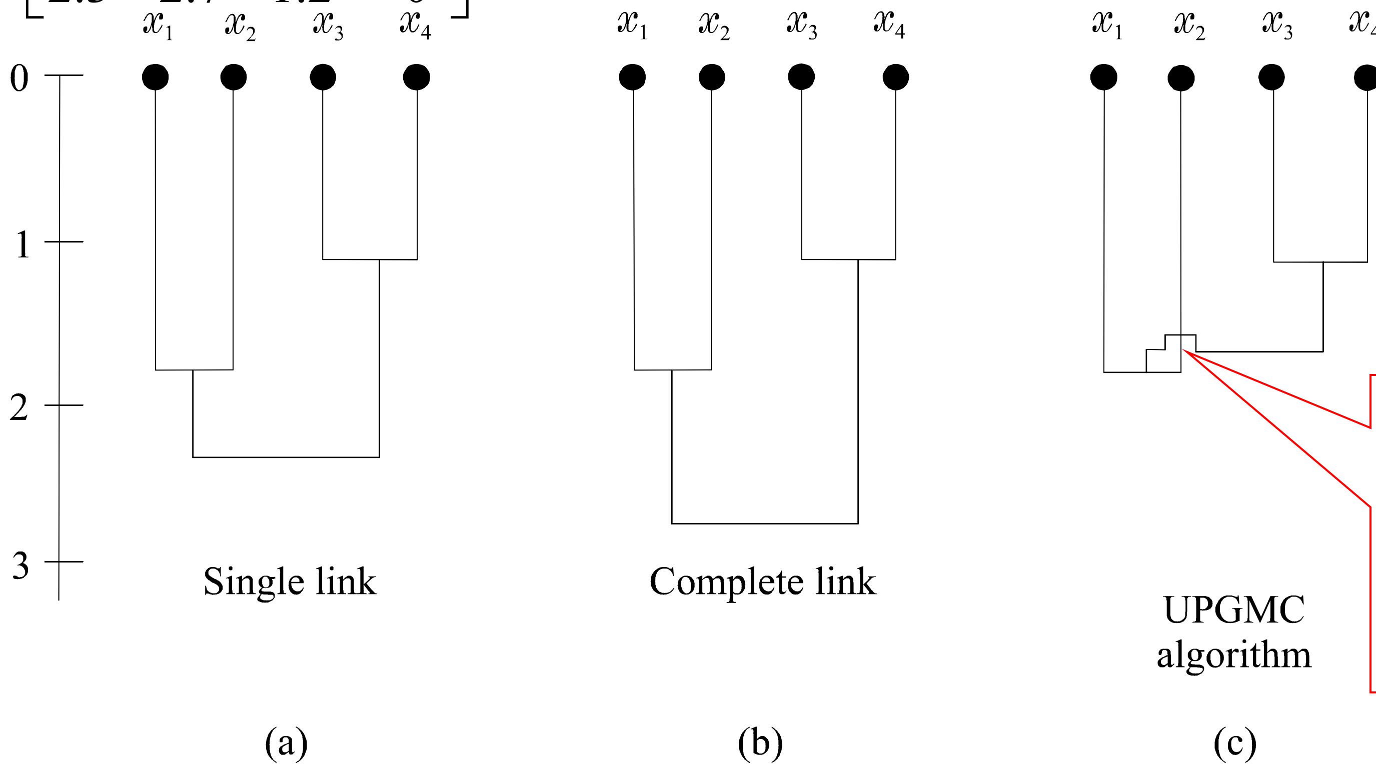
Agglomerative matrix theory based Clustering Algorithms

Monotonicity and crossover:

For the following dissimilarity matrix

$$P = \begin{bmatrix} 0 & 1.8 & 2.4 & 2.3 \\ 1.8 & 0 & 2.5 & 2.7 \\ 2.4 & 2.5 & 0 & 1.2 \\ 2.3 & 2.7 & 1.2 & 0 \end{bmatrix}$$

the dissimilarity dendrograms produced by single link, complete link and UPGMC (the same result is produced if WPGMC is employed) are:



$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}$ is formed at lower dissimilarity level than $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ (crossover)

Agglomerative matrix theory based Clustering Algorithms

Example (in detail): The WPGMC case

$(C_q = C_i \cup C_j,\ d_{qs} = \frac{1}{2}d_{is} + \frac{1}{2}d_{js} - \frac{1}{4}d_{ij})$

P_0:

	$\{x_1\}$	$\{x_2\}$	$\{x_3\}$	$\{x_4\}$
$\{x_1\}$	0	1.8	2.4	2.3
$\{x_2\}$	1.8	0	2.5	2.7
$\{x_3\}$	2.4	2.5	0	**1.2**
$\{x_4\}$	2.3	2.7	**1.2**	0

	$\{x_1\}$	$\{x_2\}$	$\{x_3\}$	$\{x_4\}$
$\{x_1\}$	0	1.8	2.4	2.3
$\{x_2\}$	1.8	0	2.5	2.7
$\{x_3\}$	2.4	2.5	0	**1.2**
$\{x_4\}$	2.3	2.7	**1.2**	0

$$d_{(3,4),1} = \frac{1}{2}d_{3,1} + \frac{1}{2}d_{4,1} - \frac{1}{4}d_{3,4}$$
$$= \frac{1}{2}2.4 + \frac{1}{2}2.3 - \frac{1}{4}1.2 = 2.05$$

$$d_{(3,4),2} = \frac{1}{2}d_{3,2} + \frac{1}{2}d_{4,2} - \frac{1}{4}d_{3,4}$$
$$= \frac{1}{2}2.5 + \frac{1}{2}2.7 - \frac{1}{4}1.2 = 2.3$$

P_1:

	$\{x_1\}$	$\{x_2\}$	$\{x_3, x_4\}$
$\{x_1\}$	0	**1.8**	2.05
$\{x_2\}$	**1.8**	0	2.3
$\{x_3, x_4\}$	2.05	2.3	0

	$\{x_1\}$	$\{x_2\}$	$\{x_3, x_4\}$
$\{x_1\}$	0	**1.8**	2.05
$\{x_2\}$	**1.8**	0	2.3
$\{x_3, x_4\}$	2.05	2.3	0

$\mathfrak{R}_0 = \{\{\underline{x}_1\},\{\underline{x}_2\},\{\underline{x}_3\},\{\underline{x}_4\}\}, (\mathbf{0})$

$\mathfrak{R}_1 = \{\{\underline{x}_1\},\{\underline{x}_2\},\{\underline{x}_3,\underline{x}_4\}\}, (\mathbf{1.2})$

$\mathfrak{R}_2 = \{\{\underline{x}_1,\underline{x}_2\},\{\underline{x}_3,\underline{x}_4\}\}, (\mathbf{1.8})$

$\mathfrak{R}_3 = \{\{\underline{x}_1,\underline{x}_2,\underline{x}_3,\underline{x}_4\}\}, (\mathbf{1.275}\ !!)$

$$d_{(1,2),(3,4)} = \frac{1}{2}d_{1,(3,4)} + \frac{1}{2}d_{2,(3,4)} - \frac{1}{4}d_{1,2}$$
$$= \frac{1}{2}2.05 + \frac{1}{2}2.3 - \frac{1}{4}1.8 = 1.275$$

P_2:

	$\{x_1, x_2\}$	$\{x_3, x_4\}$
$\{x_1, x_2\}$	0	**1.275**
$\{x_3, x_4\}$	**1.275**	0

P_3:

	$\{x_1, x_2, x_3, x_4\}$
$\{x_1, x_2, x_3, x_4\}$	0

Agglomerative matrix theory based Clustering Algorithms

- Monotonicity condition:

If clusters C_i and C_j are selected to be merged in cluster C_q, at the tth level of the hierarchy, the condition

$$d(C_q, C_k) \geq d(C_i, C_j)$$

must hold for all $C_k, k \neq i, j, q$.

In other words, the monotonicity condition implies that a clustering is formed at higher dissimilarity level than any of its components.

Remarks:

- Monotonicity is a property that is **exclusively related** to the clustering algorithm and not to the (initial) proximity matrix.
- An algorithm that does not **satisfy** the monotonicity condition, does not necessarily **produce** dendrograms with crossovers.
- Single link, complete link, UPGMA, WPGMA and the Ward's algorithm **satisfy** the monotonicity condition, while UPGMC and WPGMC **do not satisfy** it.

Agglomerative matrix theory based Clustering Algorithms

Complexity issues:

- GAS requires, in general, $O(N^3)$ operations.
- More efficient implementations require $O(N^2 \log N)$ computational time.
- For a class of widely used algorithms, implementations that require $O(N^2)$ computational time and $O(N^2)$ or $O(N)$ storage have also been proposed.
- Parallel implementations on SIMD machines have also been considered.

Agglomerative graph theory based Clustering Algorithms

Some basic definitions from graph theory:

- A graph, G, is defined as an ordered pair $G = (V, E)$, where $V = \{v_i, i = 1, \ldots, N\}$ is a set of vertices and E is a set of edges connecting some pairs of vertices. An edge connecting v_i and v_j is denoted by e_{ij} or (v_i, v_j).

- A graph is called undirected if there is no direction assigned to any of its edges. Otherwise, we deal with directed graphs.

- A graph is called unweighted if there is no cost associated with any of its edges. Otherwise, we deal with weighted graphs.

- A path in G between vertices v_{i_1} and v_{i_n} is a sequence of vertices and edges of the form $v_{i_1} e_{i_1 i_2} v_{i_2} \ldots v_{i_{n-1}} e_{i_{n-1} i_n} v_{i_n}$.

- A loop in G is a path where v_{i_1} and v_{i_n} coincide.

- A subgraph $G' = (V', E')$ of $G = (V, E)$ is a graph with $V' \subseteq V$ and $E' \subseteq E_1$, where E_1 is a subset of E containing edges that connect vertices of V'. Every graph is a subgraph to itself.

Agglomerative graph theory based Clustering Algorithms

Some basic definitions from graph theory (cont.):

- A connected subgraph $G' = (V', E')$ is a subgraph where there exists at least one path connecting any pair of vertices in V'.

- A complete subgraph $G' = (V', E')$ is a subgraph where for any pair of vertices in V' there exists an edge in E' connecting them.

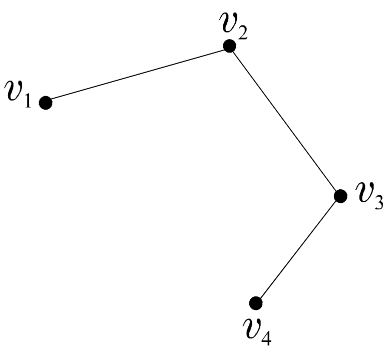
- A maximally connected subgraph of G is a connected subgraph G' of G that contains as many vertices of G as possible.

- A maximally complete subgraph of G is a complete subgraph G' of G that contains as many vertices of G as possible.

Examples for the above, are shown in the following figure.

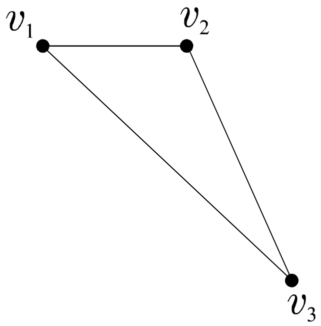
Agglomerative graph theory based Clustering Algorithms

Some basic definitions from graph theory (cont.):



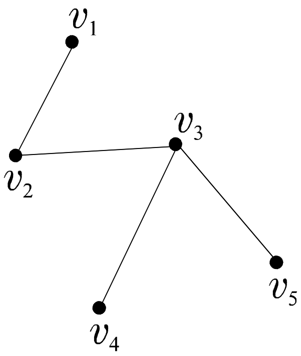
Path

(a)



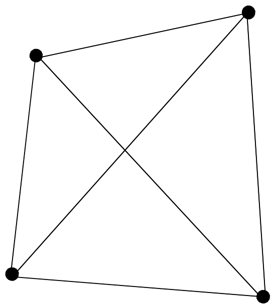
Loop

(b)



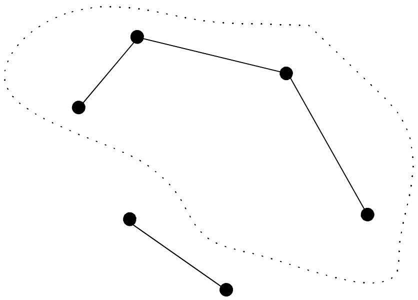
Connected graph

(c)



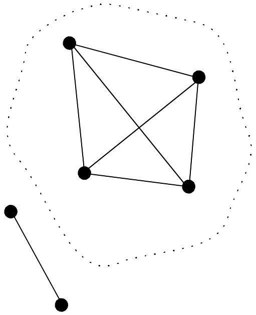
Complete graph

(d)



Maximally connected subgraph

(e)



Maximally complete subgraph

(f)

Agglomerative graph theory based Clustering Algorithms

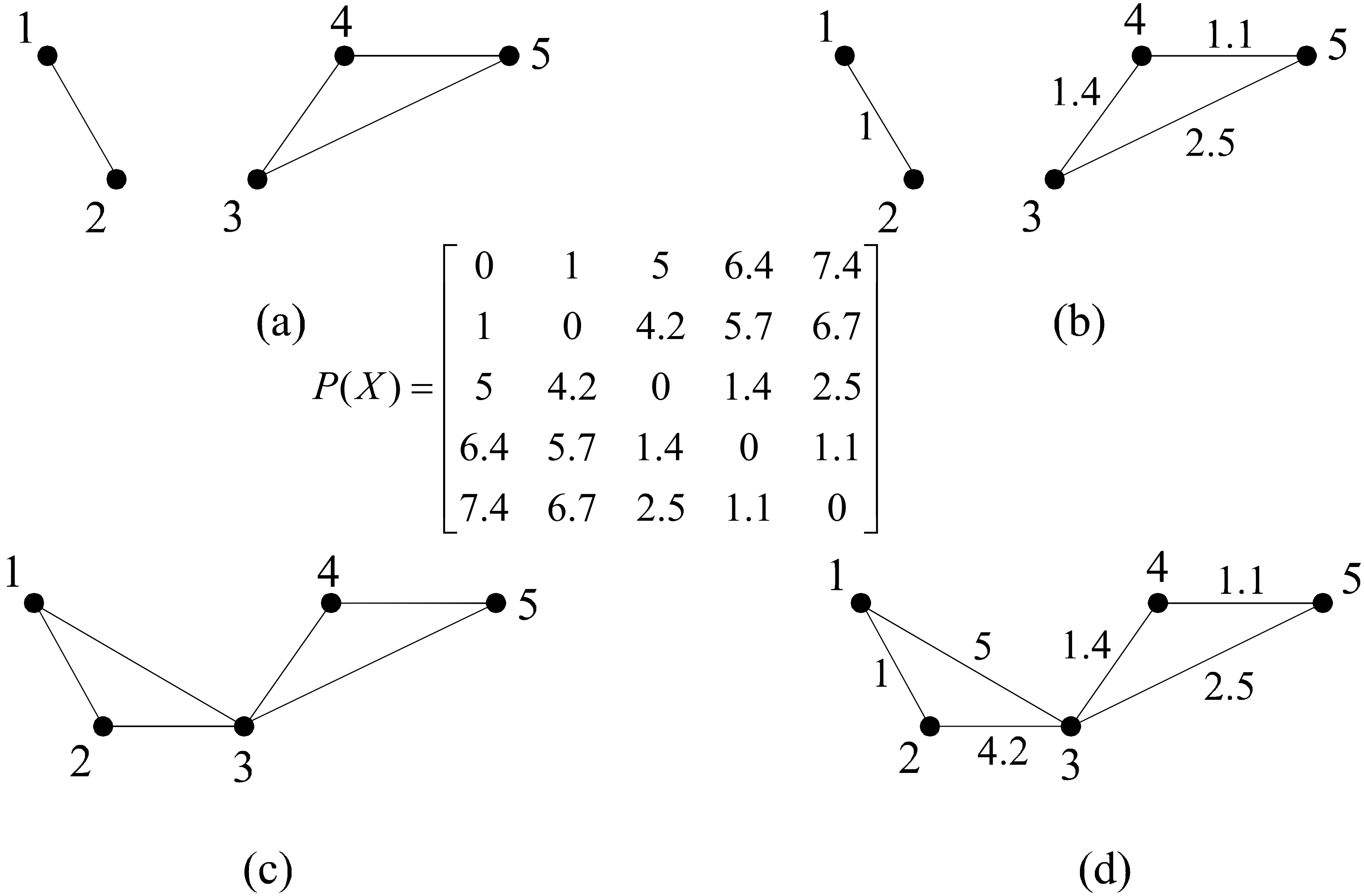
NOTE: In the framework of clustering, each vertex of a graph corresponds to a feature vector.

Useful **tools** for the algorithms based on graph theory are the threshold graph and the proximity graph.

- A threshold graph $G(a)$
  - is an undirected, unweighted graph with N nodes, each one corresponding to a vector of X.
  - No self-loops or multiple edges between any two vertices are encountered.
  - The set of edges of $G(a)$ contains those edges (v_i, v_j) for which the distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between the vectors corresponding to v_i and v_j is less than or equal to a.

- A proximity graph $G_p(a)$ is a threshold graph $G(a)$, all of whose edges (v_i, v_j) are weighted with the proximity measure $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Agglomerative graph theory based Clustering Algorithms



(a)

(b)

$$P(X)=\begin{bmatrix} 0 & 1 & 5 & 6.4 & 7.4 \\ 1 & 0 & 4.2 & 5.7 & 6.7 \\ 5 & 4.2 & 0 & 1.4 & 2.5 \\ 6.4 & 5.7 & 1.4 & 0 & 1.1 \\ 7.4 & 6.7 & 2.5 & 1.1 & 0 \end{bmatrix}$$

(c)

(d)

(a) The threshold graph $G(3)$, (b) the proximity (dissimilarity) graph $G_p(3)$, (c) the threshold graph $G(5)$, (d) the dissimilarity graph $G_p(5)$, for the dissimilarity matrix $P(X)$ shown above.

Agglomerative graph theory based Clustering Algorithms

More definitions:

- In this framework, we consider graphs G, of N nodes, where each node corresponds to a vector of X.
- Valid clusters are connected components of G that satisfy an **additional** graph property $h(k)$.

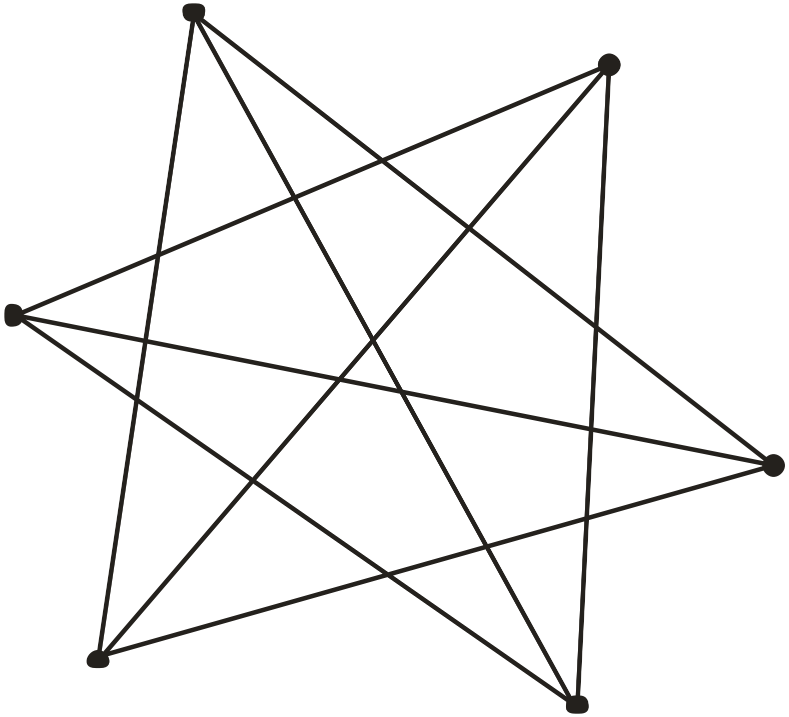
Typical graph properties for a connected component (subgraph) $G´$ of G are:

- Node connectivity: The largest integer k such that all pairs of nodes of $G´$ are joined by at least k paths having no nodes in common.

- Edge connectivity: The largest integer k such that all pairs of nodes are joined by at least k paths having no edges in common.

- Node degree: The largest integer k such that each node has at least k incident edges.

Agglomerative graph theory based Clustering Algorithms



Node connectivity : 3

Edge connectivity : 3

Node degree : 3

Agglomerative graph theory based Clustering Algorithms

➢ Proximity function in the graph theory framework

- The **proximity function** $g_{h(k)}(C_r, C_s)$ between two clusters is **defined** in terms of
  - a proximity measure between vectors (nodes)
  - certain constraints **imposed** by property $h(k)$ on the subgraphs that are formed.

In mathematical language:

(4)

$$g_{h(k)}(C_r, C_s) = \min_{\boldsymbol{x}_u \in C_r, \boldsymbol{x}_v \in C_s} \left\{ \begin{array}{c} d(\boldsymbol{x}_u, \boldsymbol{x}_v) \equiv a: \textit{the } G(a) \textit{ subgraph defined by } C_r \cup C_s \textit{ is} \\ (a) \ \boldsymbol{connected} \textit{ and either } (b1) \textit{ has the } \boldsymbol{property} \ h(k) \textit{ or } (b2) \textit{ is } \boldsymbol{complete} \end{array} \right\}$$

or

$g_{h(k)}(C_r, C_s)$ equals to the **smallest** possible value a such that in *the* $G(a)$ *subgraph defined by* $C_r \cup C_s$ *is* (*a*) ***connected*** *and either* (*b*1) *has the* ***property*** $h(k)$ *or* (*b*2) *is* ***complete***.

Agglomerative graph theory based Clustering Algorithms

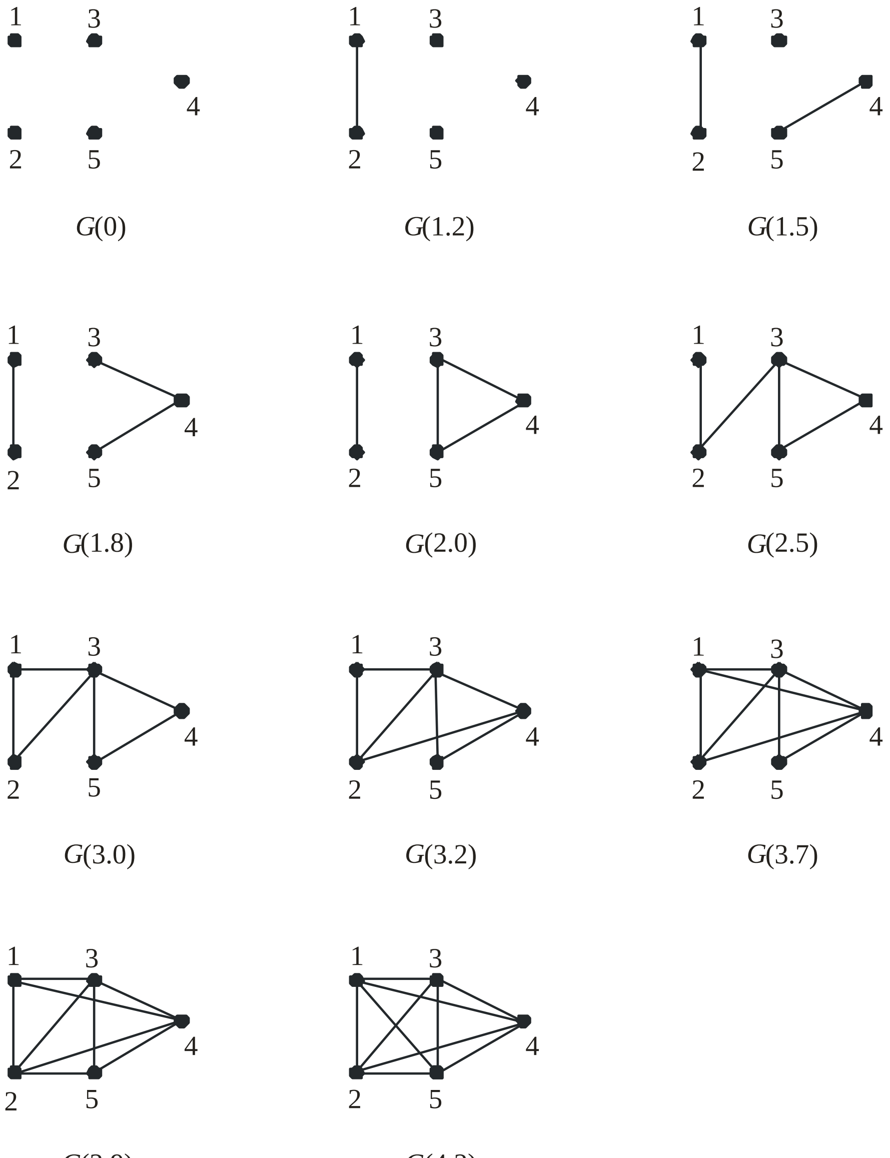
- Example: For the dissimilarity matrix,

$$P = \begin{bmatrix} 0 & 1.2 & 3 & 3.7 & 4.2 \\ 1.2 & 0 & 2.5 & 3.2 & 3.9 \\ 3 & 2.5 & 0 & 1.8 & 2.0 \\ 3.7 & 3.2 & 1.8 & 0 & 1.5 \\ 4.2 & 3.9 & 2.0 & 1.5 & 0 \end{bmatrix}$$

all possible $G(a)$ graphs are shown next.

Assuming that $h(2)$ is the node connectivity property, it is

$$g_h(\{x_1\},\{x_2\}) = 1.2 \text{ (complete)}$$
$$g_h(\{x_1\},\{x_5\}) = 4.2 \text{ (complete)}$$
$$g_h(\{x_1,x_2\},\{x_3\}) = 3 \text{ (compl.-}h(2)\text{)}$$
$$g_h(\{x_1,x_2\},\{x_3,x_5\}) = 3.9 \text{ (}h(2)\text{)}$$



G(0) G(1.2) G(1.5)

G(1.8) G(2.0) G(2.5)

G(3.0) G(3.2) G(3.7)

G(3.9) G(4.2)

Agglomerative graph theory based Clustering Algorithms

- Graph theory-based algorithmic scheme (GTAS): It is the GAS in the context of graph theory. In the context of GTAS, the definition of the proximity between the clusters is based on graph theory concepts. Thus

Generalized Agglomerative Scheme (GAS)

- 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

$$g_{h(k)}(C_i, C_j) = \begin{cases} min_{r,s}\, g_{h(k)}(C_r, C_s), & for\ disim.functions \\ max_{r,s}\, g_{h(k)}(C_r, C_s), & for\ sim.functions \end{cases}$$

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

Agglomerative graph theory based Clustering Algorithms

- Single link (SL) algorithm. Here

$g_{h(k)}(C_r, C_s)$
$= min_{\boldsymbol{x}_u \in C_r, \boldsymbol{x}_v \in C_s}\{d(\boldsymbol{x}_u, \boldsymbol{x}_v) \equiv a: the\ G(a)\ subgraph\ defined\ by\ C_r \cup C_s\ is\ \boldsymbol{connected}\}$
$\equiv min_{\boldsymbol{x} \in C_r, \boldsymbol{y} \in C_s} d(\boldsymbol{x}, \boldsymbol{y})$ (why?)

- Remarks:

– No property $h(k)$ or completeness is required.

– The SL stemming from the graph theory is exactly the same with the SL stemming from the matrix theory.

- Complete link (CL) algorithm. Here

$g_{h(k)}(C_r, C_s)$
$= min_{\boldsymbol{x}_u \in C_r, \boldsymbol{x}_v \in C_s}\{d(\boldsymbol{x}_u, \boldsymbol{x}_v) \equiv a: the\ G(a)\ subgraph\ defined\ by\ C_r \cup C_s\ is\ \boldsymbol{complete}\}$
$\equiv max_{\boldsymbol{x} \in C_r, \boldsymbol{y} \in C_s} d(\boldsymbol{x}, \boldsymbol{y})$ (why?)

- Remarks:

– No property $h(k)$ is required.

– The CL stemming from graph theory is exactly the same with the CL stemming from matrix theory.

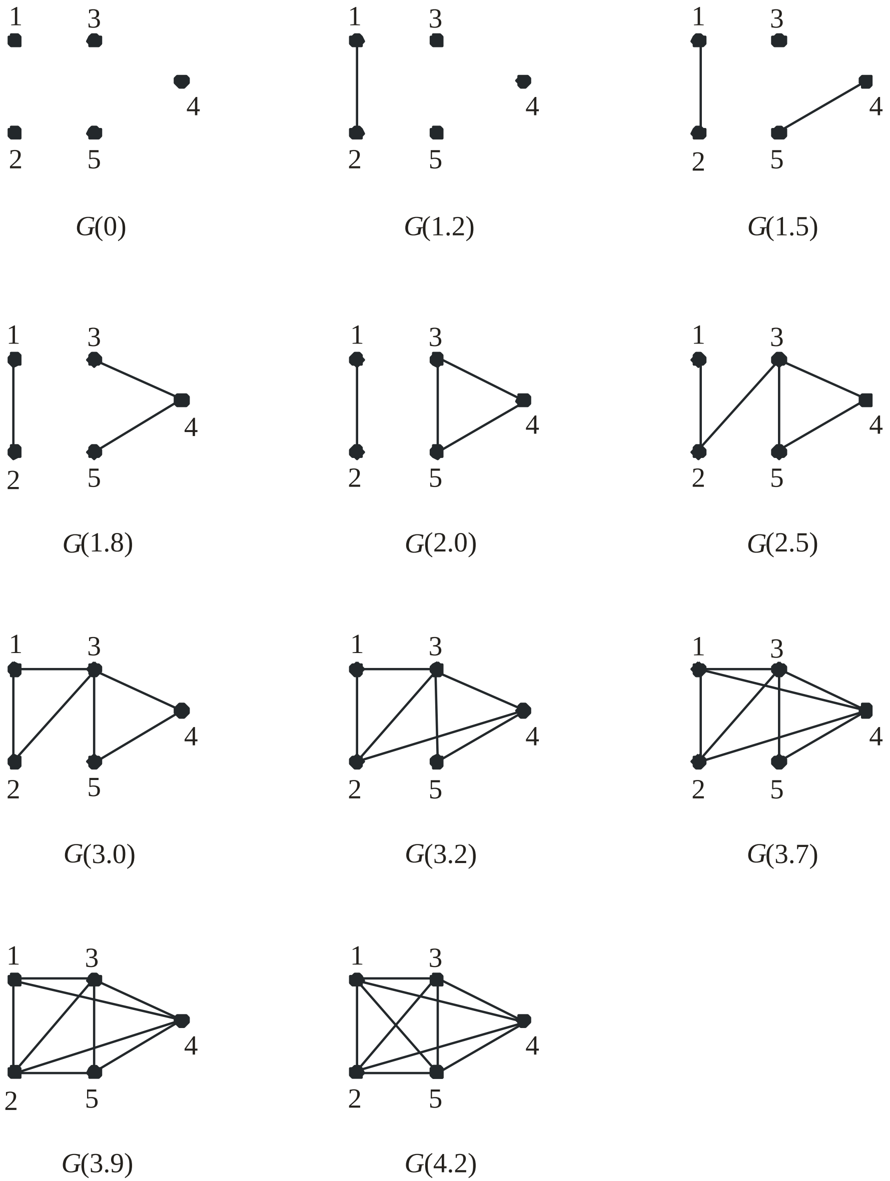
Agglomerative graph theory based Clustering Algorithms

- **Example:** For the dissimilarity matrix,

$$P = \begin{bmatrix} 0 & 1.2 & 3 & 3.7 & 4.2 \\ 1.2 & 0 & 2.5 & 3.2 & 3.9 \\ 3 & 2.5 & 0 & 1.8 & 2.0 \\ 3.7 & 3.2 & 1.8 & 0 & 1.5 \\ 4.2 & 3.9 & 2.0 & 1.5 & 0 \end{bmatrix}$$

SL and CL produce the same hierarchy of clusterings at the levels given in the table.

Clustering	SL	CL
$\Re_0 = \{\{x_1\},\{x_2\},\{x_3\},\{x_4\},\{x_5\}\}$	0	0
$\Re_1 = \{\{x_1,x_2\},\{x_3\},\{x_4\},\{x_5\}\}$	1.2	1.2
$\Re_2 = \{\{x_1,x_2\},\{x_3\},\{x_4,x_5\}\}$	1.5	1.5
$\Re_3 = \{\{x_1,x_2\},\{x_3,x_4,x_5\}\}$	1.8	2.0
$\Re_4 = \{\{x_1,x_2,x_3,x_4,x_5\}\}$	2.5	4.2



G(0) G(1.2) G(1.5)

G(1.8) G(2.0) G(2.5)

G(3.0) G(3.2) G(3.7)

G(3.9) G(4.2)

Agglomerative graph theory based Clustering Algorithms

Remarks:

- SL poses the weakest possible graph condition (connectivity) for the formation of a cluster, while CL poses the strongest possible graph condition (completeness) for the formation of a cluster.
- A variety of graph theory-based algorithms, that lie between these two extremes result for various choices of $h(k)$.
  - –For $k = 1$ all these algorithms **collapse** to the single link algorithm.
  - –As k increases, the resulting subgraphs approach completeness.

Clustering algorithms based on the Minimum Spanning Tree (MST)

Definitions:

Spanning Tree: It is a connected graph (containing all the vertices of the graph), with no loops (only one path connects any two vertices).

Weight of a Spanning Tree: The sum of the weights of its edges (provided a weight has been assigned to each one of them).

Minimum Spanning Tree (MST): A spanning tree with the smallest weight among the spanning trees connecting all the vertices of the graph.

Agglomerative graph theory based Clustering Algorithms

Remarks:

- The MST has $N-1$ edges.
- When all the weights are different from each other, the MST is unique. Otherwise, it may not be unique.

➢ Employing the GTAS and substituting $g_{h(k)}\ (C_r, C_s)$ with

$$g(C_r, C_s) = min_{ij}\{w_{ij}\colon\ \boldsymbol{x}_i \in C_r, \boldsymbol{x}_j \in C_s\}$$

where $w_{ij} = d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we can determine the MST.

➢ **On the other hand,** a hierarchy of clusterings may be obtained by the MST as follows:

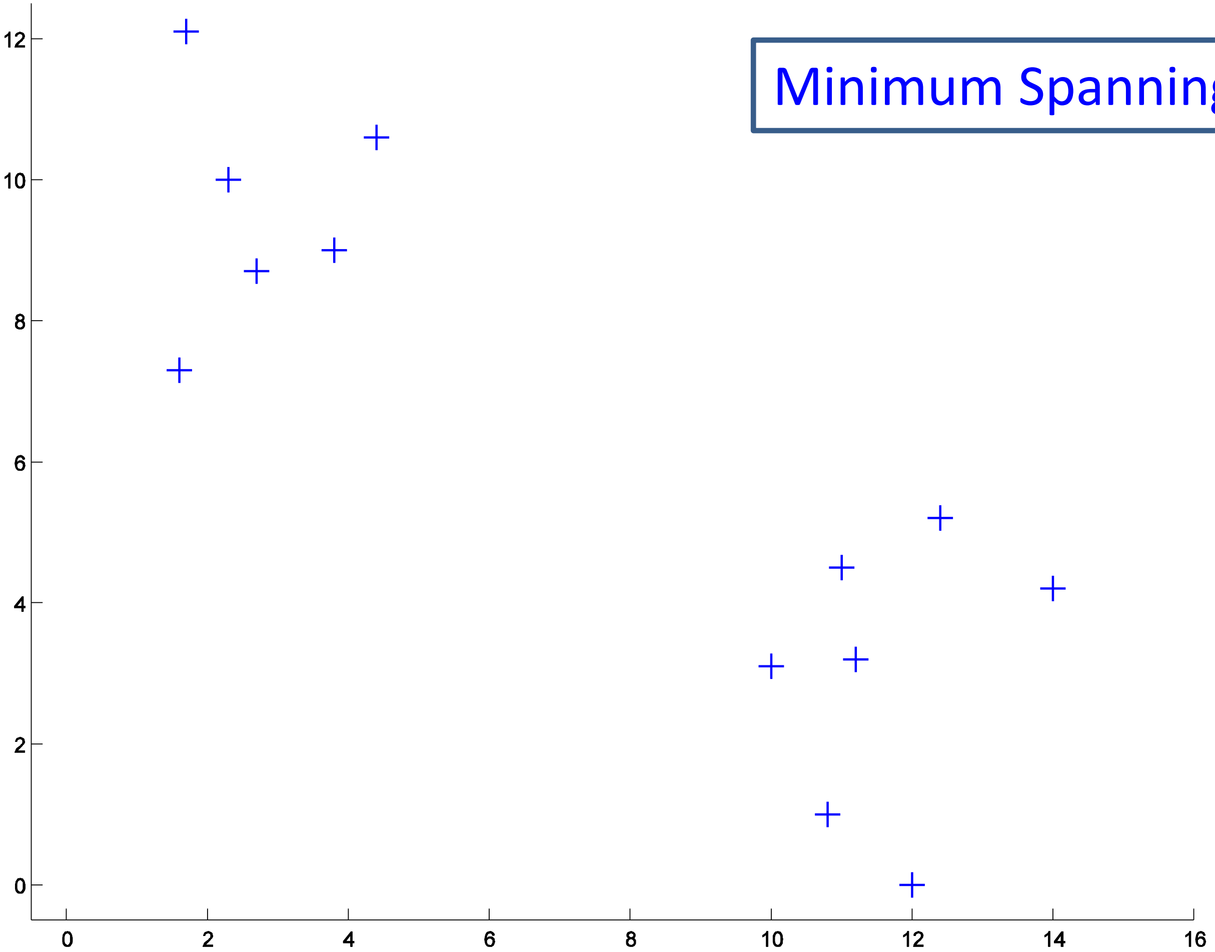
The clustering $\Re_t$ at the t −th level is the set of connected components of the MST, when only its t smallest weights are considered.

Remark:

The hierarchy produced by MST is the same with that produced by the single link algorithm, at least when all w_{ij}'s are different from each other.

Agglomerative graph theory based Clustering Algorithms

Example:

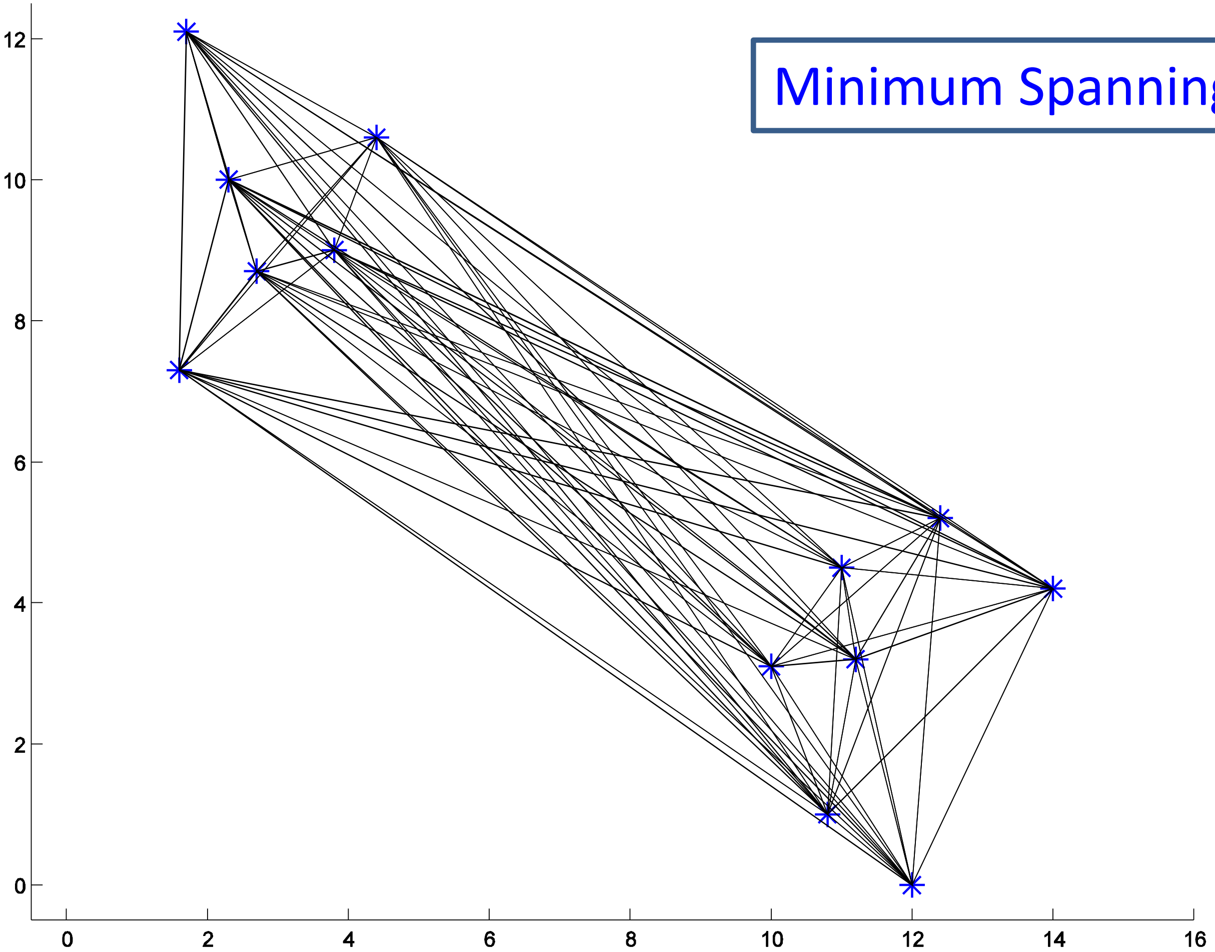


Minimum Spanning Tree (MST)

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.

Agglomerative graph theory based Clustering Algorithms

Example:



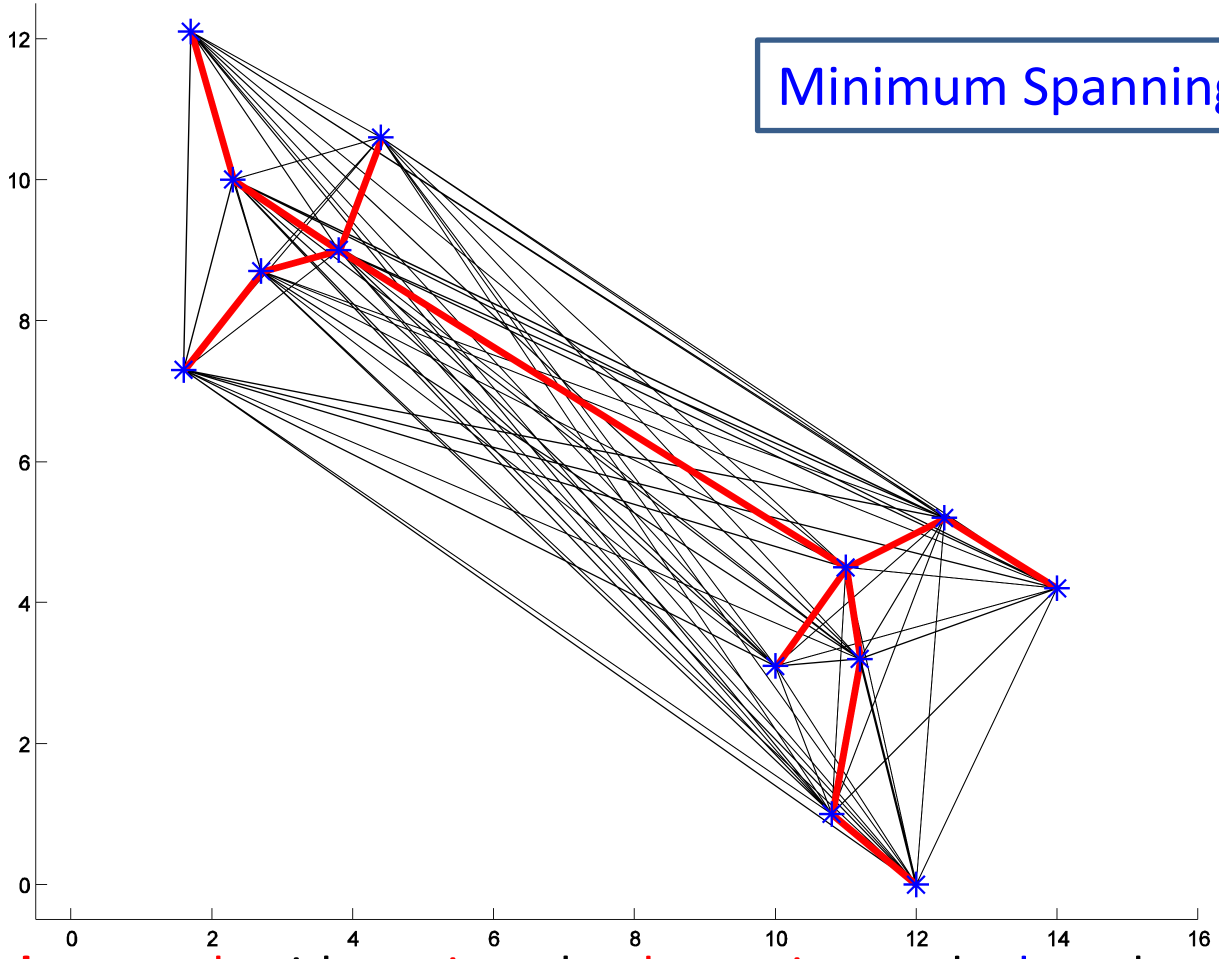
Minimum Spanning Tree (MST)

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.

Agglomerative graph theory based Clustering Algorithms

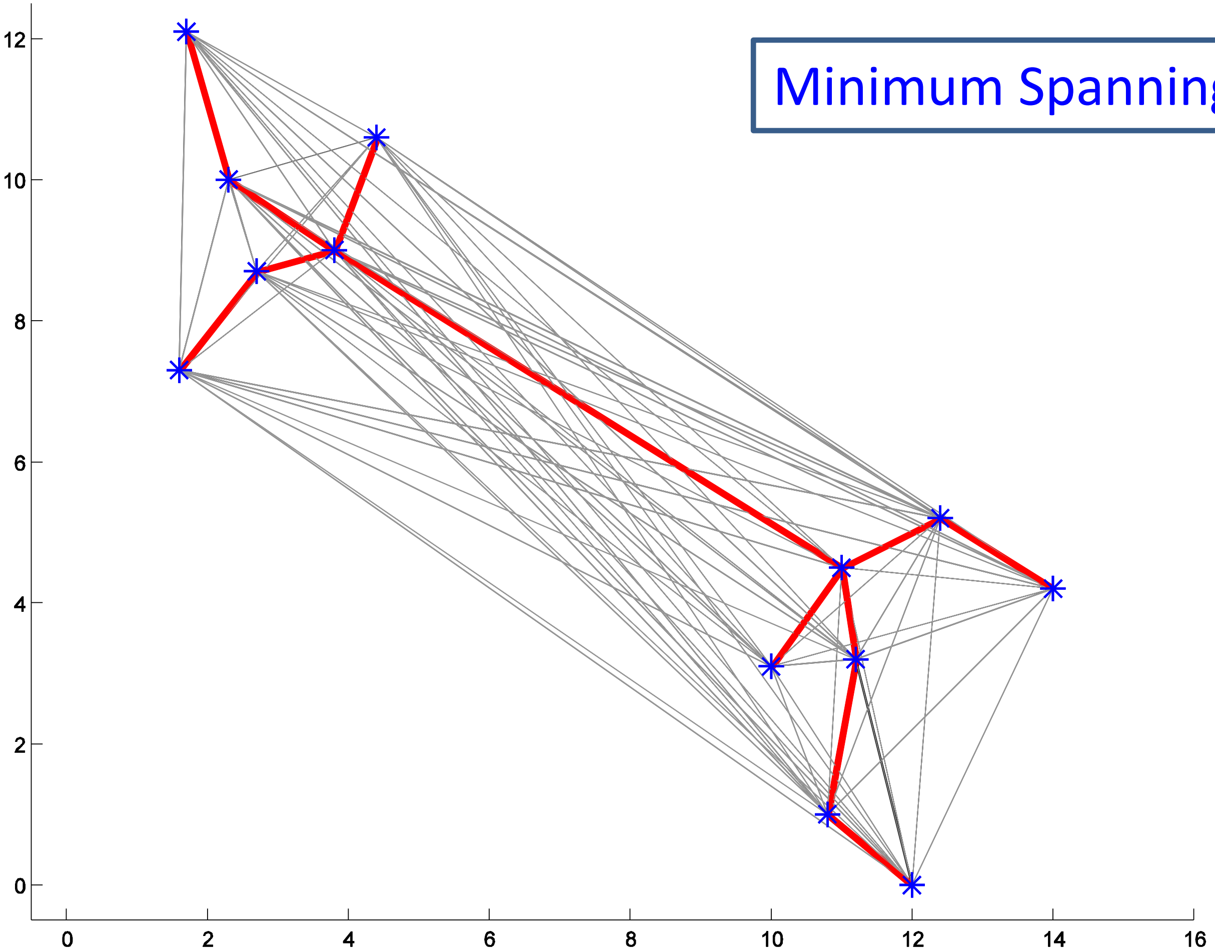
Example:

Minimum Spanning Tree (MST)



- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.

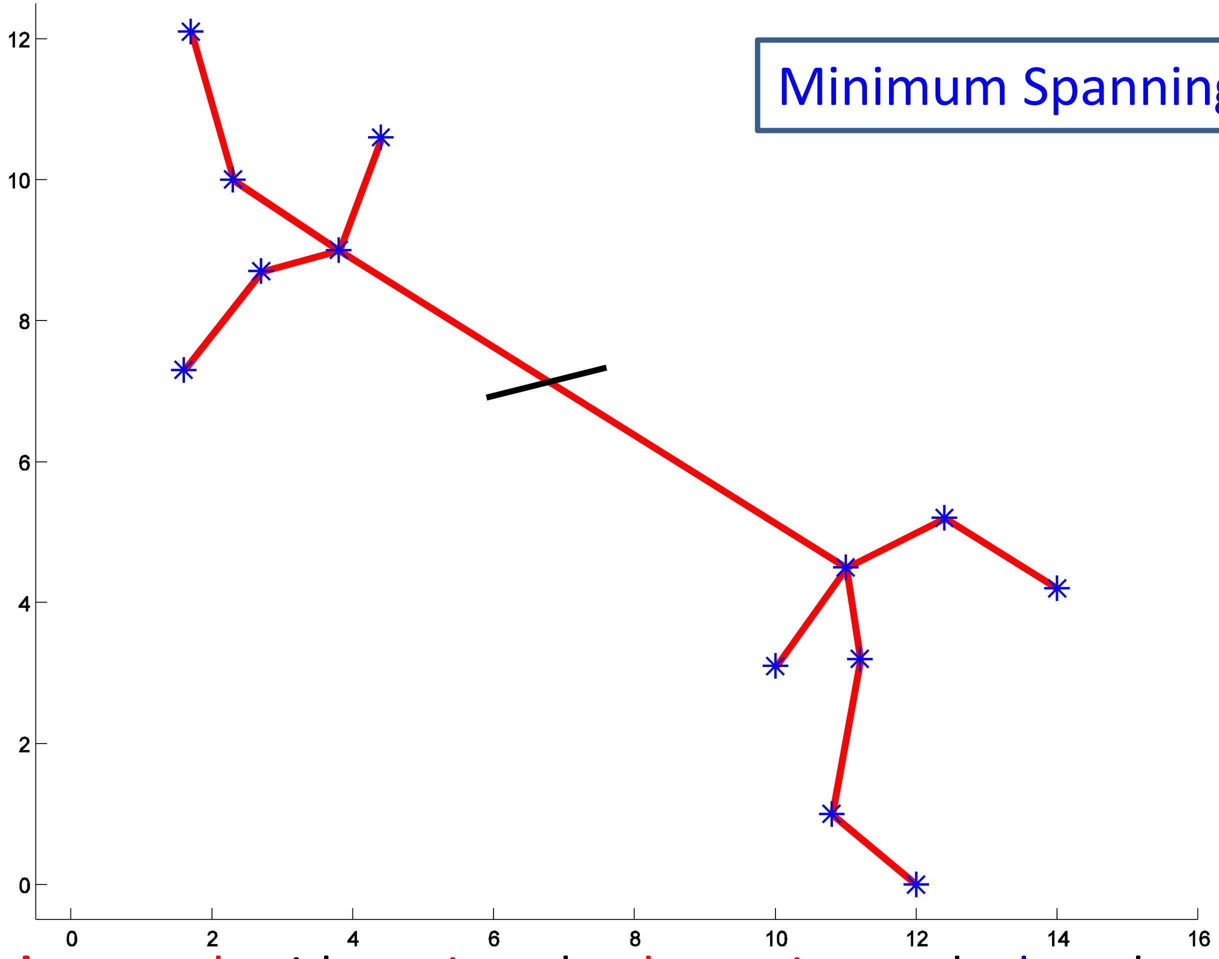
Agglomerative graph theory based Clustering Algorithms



Minimum Spanning Tree (MST)

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.

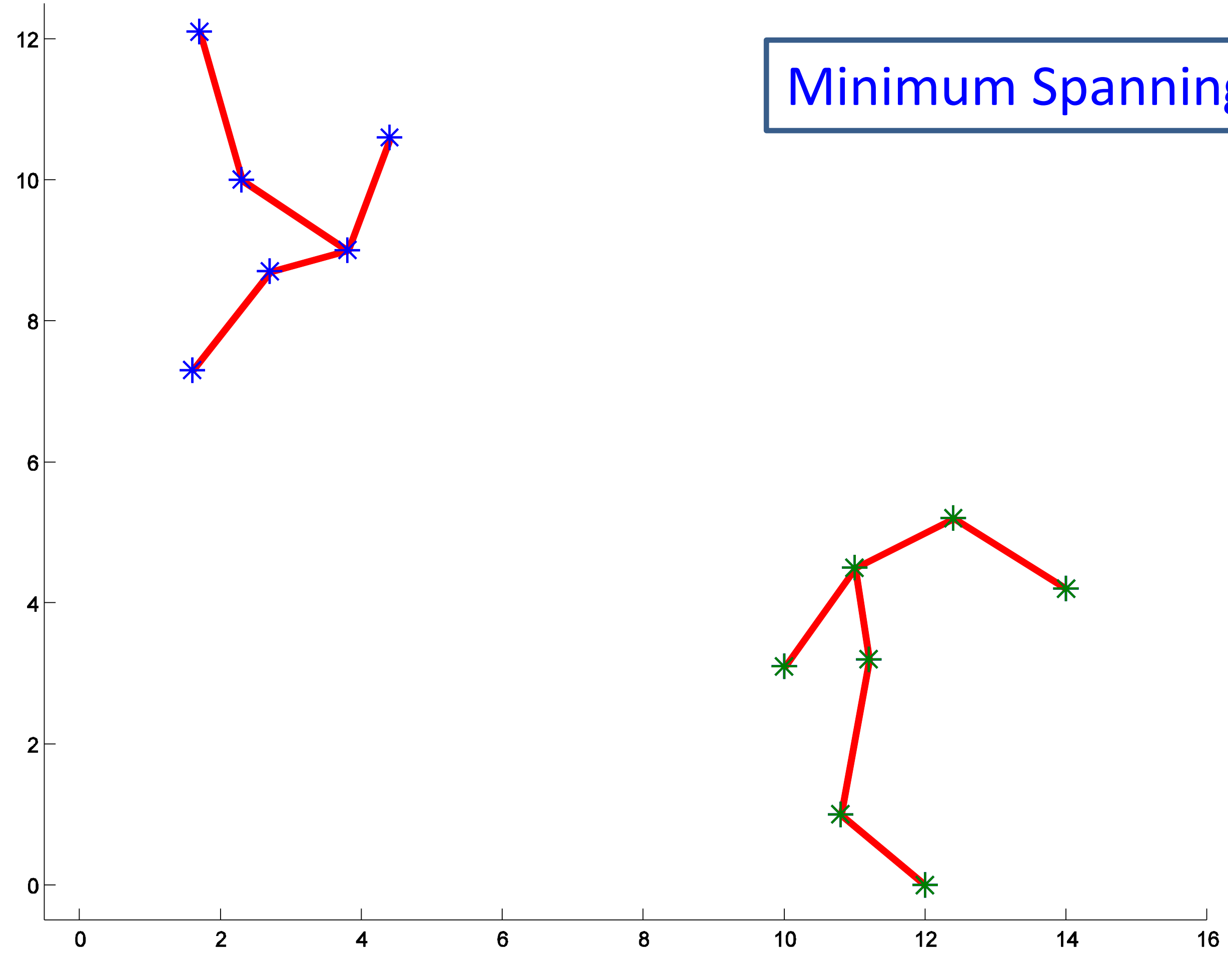
Agglomerative graph theory based Clustering Algorithms



Minimum Spanning Tree (MST)

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.

Agglomerative graph theory based Clustering Algorithms



Minimum Spanning Tree (MST)

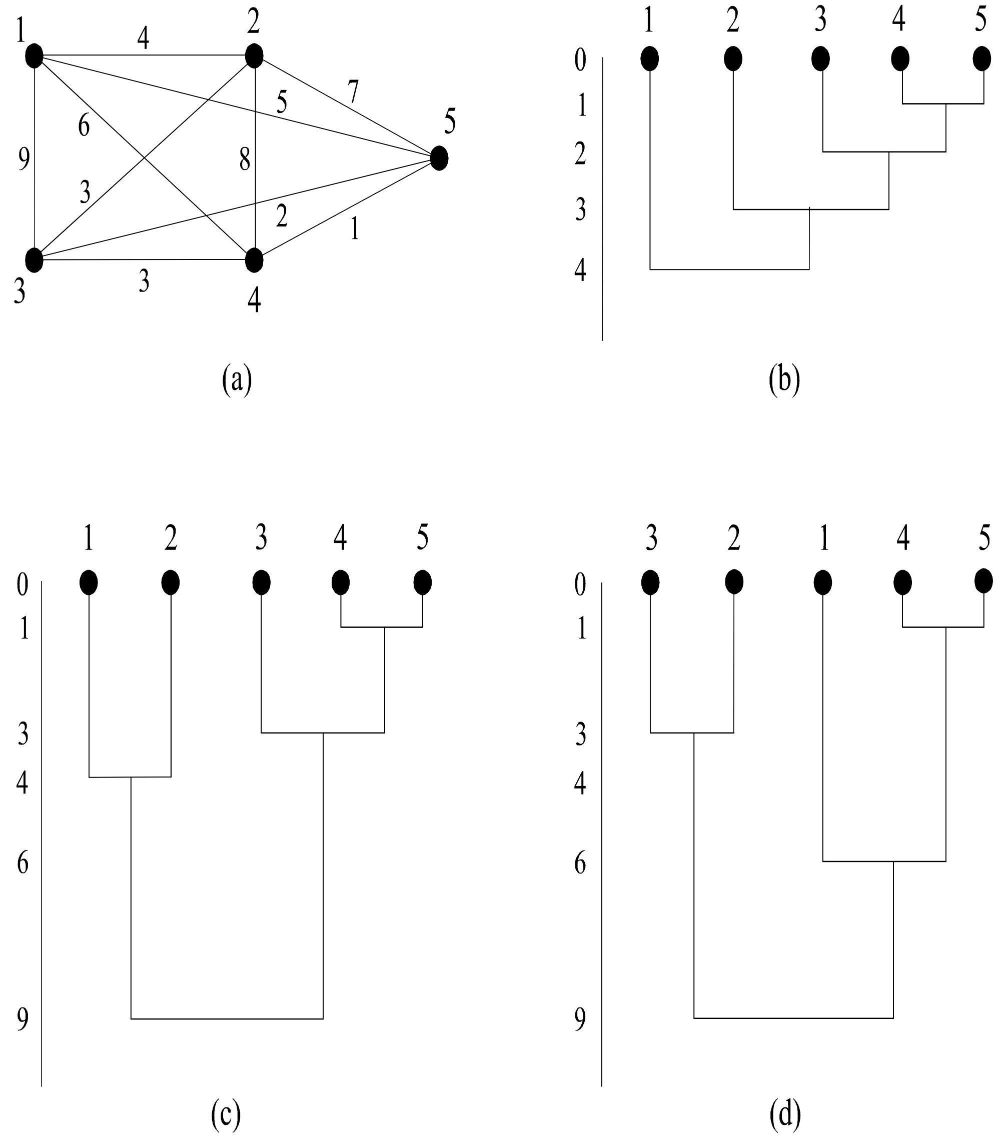
- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph.
- **Retaining** the edges with the *t* **smallest weights**, the resulting connected components define the clusters of the $\Re_t$ clustering.

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

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

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



(a) (b)

(c) (d)

(CL(a)) (CL(b))

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

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

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

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

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

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

The associated **cophenetic matrix** is

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

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 $\Re_t$, $t = 0, \dots, N-1$, $j = 1, \dots, t+1$.

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.

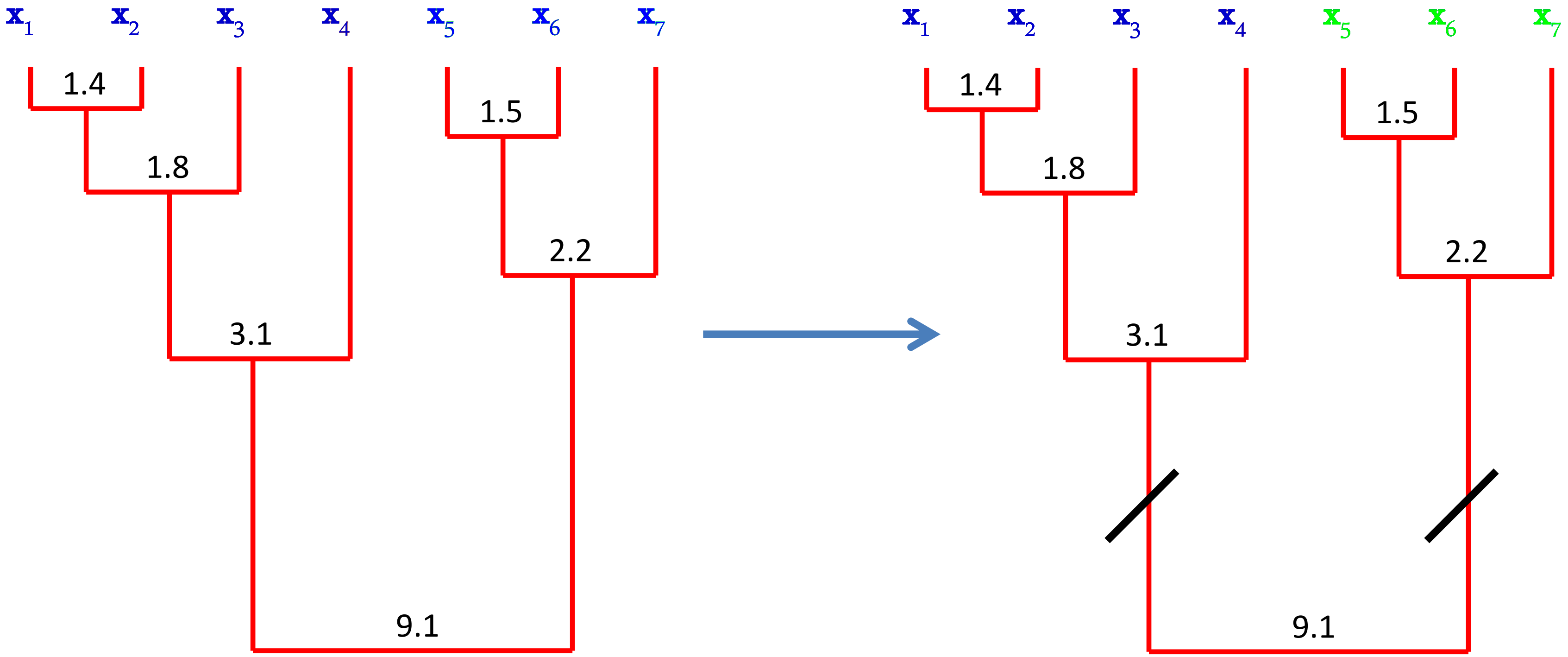
Choice of the best number of clusters

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)).


x_1 x_2 x_3 x_4 x_5 x_6 x_7
1.4
1.8
3.1
1.5
2.2
9.1
x_1 x_2 x_3 x_4 x_5 x_6 x_7
1.4
1.8
3.1
1.5
2.2
9.1

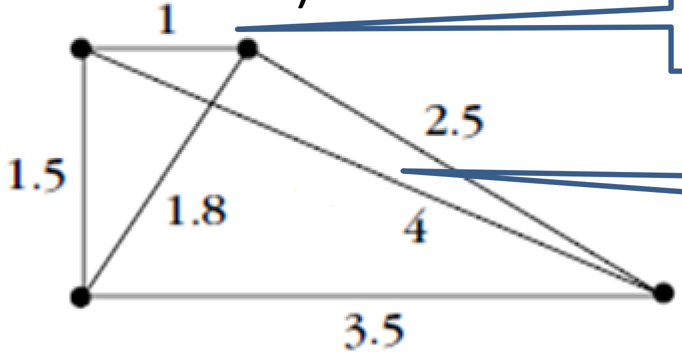
Choice of the best number of clusters

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).



$h_2(C) = min\{d(\boldsymbol{x},\boldsymbol{y}), \boldsymbol{x},\boldsymbol{y} \in C\}$

$h_1(C) = max\{d(\boldsymbol{x},\boldsymbol{y}), \boldsymbol{x},\boldsymbol{y} \in C\}$

  - 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:

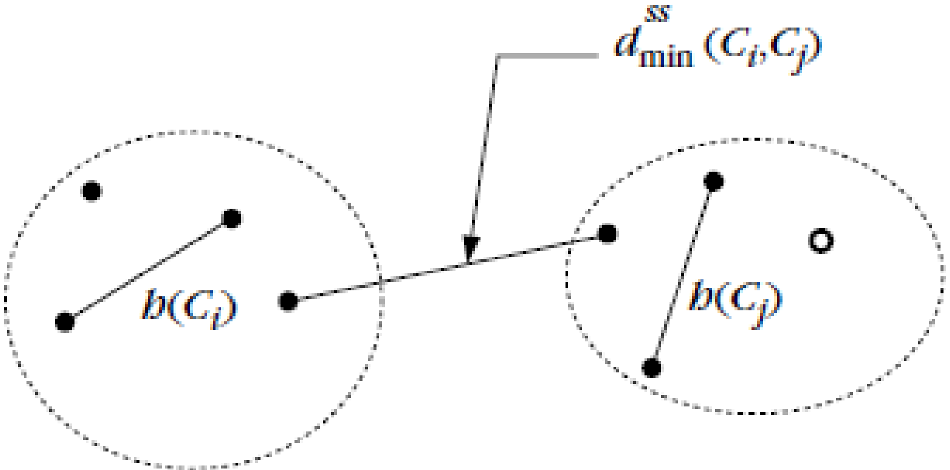
"How the clustering that best fits the data is extracted from a hierarchy of clusterings?"

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 $\Re_t$ must satisfy the following condition:

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

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

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

    $$d(C_r, C_s) = min_{\mathbf{x}\in R_{C_r}, \mathbf{y}\in R_{C_s}} d(\mathbf{x}, \mathbf{y})$$
  - **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.

(*) 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.

CURE utilizing random sampling

- Identification of clusters
  - Partition randomly X into $p = N/N'$ sample data sets.
  - For each one of the p sample data sets.
    - ❑ Apply the original version of CURE, until N'/q clusters (at the most) are formed (q is user-defined).
  - Consider all the above $p \cdot (N'/q)$ clusters (at the most) and apply the original CURE until the required number of clusters, m, is formed.
- Assignment of points to clusters
  - For each of the m clusters 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

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** $X' = \Re_k{}^1 \cup \Re_k{}^2 \cup \cdots \cup \Re_k{}^p$

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

➢**Run** *CURE*(X') and determine the most appropriate clustering $\Re_m{}'$.

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