PARIS 13 UNIVERSITY - SORBONNE PARIS CITÉ

ÉCOLE DOCTORALE GALILÉE

Thesis

Specialty : Informatics

Presented by

Nhat-Quang DOAN

Hierarchical and topological models for clustering and visualization

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisors Dr. Hanane Azzag and Dr. Mustapha Lebbah for the continuous support of my study and research, for their patience, motivation, enthusiasm, and immense knowledge. Without their guidance and persistent help this thesis would not have been possible.

I would like to show my appreciation to the rest of my committee members. I would like to thank Prof. Gilles Venturini and Dr. Cédric Wemmert who have reported my thesis and for theirs constructive questions. I thank also Prof. Yann Chevaleyre, Prof. Cyrille Bertelle and Dr. Cyril De Runz for accepted being in my thesis jury and for their interesting and relevant comments.

I am also grateful to the Computer Science Laboratory of Paris-Nord (LIPN), especially the Machine Learning and Applications Team (A3) for providing me with all the necessary facilities, for their advice and suggestions which have been very helpful.

I take this opportunity to record my sincere thanks to all fellow members, colleagues of LIPN and all my friends for their help and encouragement. My sincere thanks also goes to Prof. Younès Bennani for his helps at the beginning of my thesis and Guillaume Santini for his collaboration in research.

Let me reserve my final appreciation to my wife Thanh Hue and my parents Thi Tuyet and Van Ban. Without the care and love from them, I definitely could not have completed my doctoral degree.

Abstract

This thesis focuses on clustering approaches inspired from topological models and an autonomous hierarchical clustering method. The clustering problem becomes more complicated and difficult due to the growth in quality and quantify of structured data such as graphs, trees or sequences. In this thesis, we are particularly interested in self-organizing maps which have been generally used for learning topological preservation, clustering, vector quantization and graph visualization. Our study concerns also a hierarchical clustering method AntTree which models the ability of real ants to build structure by connect themselves. By combining the topological map with the self-assembly rules inspired from AntTree, the goal is to represent data in a hierarchical and topological structure providing more insight data information. The advantage is to visualize the clustering results as multiple hierarchical trees and a topological network.

In this report, we present three new models that are able to address clustering, visualization and feature selection problems. In the first model, our study shows the interest in the use of hierarchical and topological structure through several applications on numerical datasets, as well as structured datasets e.g. graphs and biological dataset.

The second model consists of a flexible and growing structure which does not impose the strict network-topology preservation rules. Using statistical characteristics provided by hierarchical trees, it accelerates significantly the learning process.

The third model addresses particularly the issue of unsupervised feature selection. The idea is to use hierarchical structure provided by AntTree to discover automatically local data structure and local neighbors. By using the tree topology, we propose a new score for feature selection by constraining the Laplacian score. Finally, this thesis offers several perspectives for future work.

Contents

List of Figures

List of Tables

Notation

α, β	pre-defined constants assuming convergence of GNG and GoT algorithm
b	number of edges on the boudary of a cluster
B	set of ouput label
$\mathcal{C}$	set of K clusters $\{C_i, .., C_K\}$
d	dimensionality of input data object or space
$d(\mathbf{x}_i, \mathbf{x}_j)$	distance between two objects $\mathbf{x}_i$ and $\mathbf{x}_j$
$\delta(\mathbf{x}_i, C_j)$	shortest distance from object $\mathbf{x}_i$ to cluster C_j
D	degree matrix
$deg(v)$	degree of vertex v
$\mathbf{e}$	eigenvector of Laplacian matrix
E	set of edges containing all possible (v_i, v_j)
$\mathbf{f}$	row vector, most often representing a feature vector
G or $G(V, E)$	graph composed of a set of edges V and a set of edges E
G_z	graph of pairwise protein domain structured by the alignment of $z - score$
K	number of trees or clusters
$\mathcal{K}^{\mathcal{T}}$	neighborhood function
L	set of labels
λ	set of eigenvectors
$List$	list contains all input objects
m	number of internal edges in a cluster
μ	mean vector
M_L	Laplacian matrix
n	often number of objects in a cluster
N	often total number of input objects or vertices in graph context
N_{iter}	number of iterations necessary to terminate the learning process
$\phi(\mathbf{x})$	assignment function of object $\mathbf{x}$
$\mathcal{R}$	often refering to the cost function
$\mathbb{R}^d$	set of real numbers
σ	standard deviation
$subtree_{\mathbf{x}}$	sub-tree composed of $\mathbf{x}$ as its root and all nodes recursively connected to $\mathbf{x}$
T, T_{min}, T_{max}	temperature constant used in neighborhood function $\mathcal{K}^{\mathcal{T}}$

$T_{Dissim}(\mathbf{x}_{pos})$	the lowest similarity value which can be observed among the child nodes of $\mathbf{x}_{pos}$
$tree$	set of trees
v	vertex in graph context
V	set of vertices containing all vertices v
(v_i, v_j)	edge in graph context
$\mathbf{w}$	weight vector or prototype associated with a cluster
W	adjacency matrix
$\mathcal{W}$	topological network where $\mathcal{C}$ is located
$\mathcal{X}$	feature space containing all possible objects $\mathbf{x}$
$\mathbf{x}$	column vector, most often representing an object
$\mathbf{x}^+$, $\mathbf{x}^-$	two objects connected to $\mathbf{x}_{pos}$ which are respectively the most similar and dissimilar to $\mathbf{x}$
$\mathbf{x}_{pos}$	position where $\mathbf{x}$ is located
$\|\mathbf{x}\|$	Euclidean length of vector x
Y	set of input label
y	input label of object $\mathbf{x}$
z	score measuring an alignment of two protein domains

Chapter 1

Context of the work

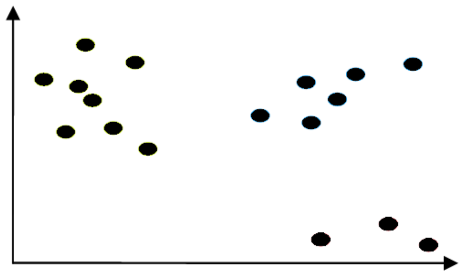
Contents

1.1 Introduction

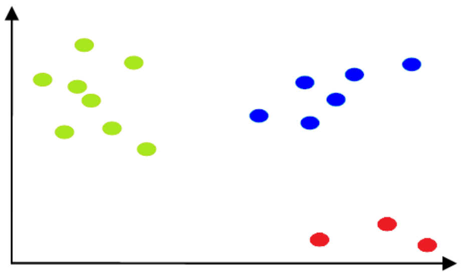
Clustering is considered as the most important unsupervised learning problem. It is a main task of data mining and the common technique that has been used in many fields, including machine learning, data mining, pattern recognition, web mining, textual document collection, image segmentation, biology, etc... [Everitt 2011].

A typical example of the clustering problem is depicted in Figure 1.1. Assuming that a distribution of input objects in a feature space is given as in Figure 1.1(a). The problem is now to distribute these objects into different groups such that the objects in each group carry a common property. It becomes very complicated while no extra information on object class or label is available. Then what should be the rule to put an object in a specific group? Likely, only similarity among objects becomes useful in this case. Similarity between two objects is often measured by their respective distance in the case of continuous data. Therefore, two objects are supposed *similar* if they are *close* to each other in the data space. A desired solution is shown in Figure 1.1(b) where three groups (or clusters): blue, green and red are well separated. The objects in each group are close together and far apart from the others.

We can re-define the clustering as following: the clustering task consists in grouping a set of unlabeled data objects based on similarity such that the objects in the same *cluster* (group) are similar to each other and dissimilar to those in the other clusters [Jain 1999]. In this kind of problem, there is little prior information available about data, in other words, the labels (pre-classified) are not provided [Everitt 2011]. Due to the absence of class label, clustering becomes more challenging and complicated. How can we analyze or validate the clustering results? For clustering quality, many



(a) Data sample in 2D



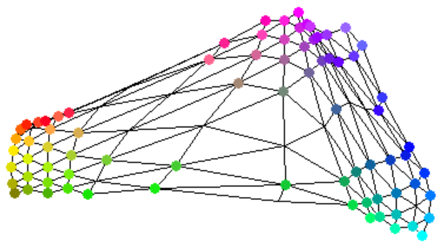
(b) A clustering solution with 3 clusters

Figure 1.1: Data clustering

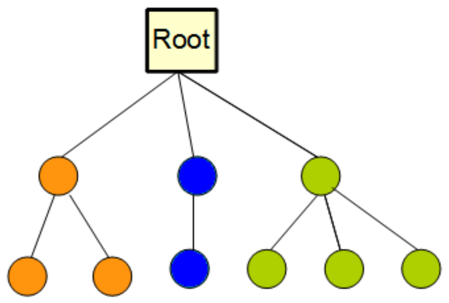
criteria such as Davies-Bouldin index [Davies 1979], Dunn index [Bezdek 1995], Rand index [Rand 1971], Jaccard index [Tan 2005], etc... have been proposed. They measure various cluster aspects, for example: distance between centroids, cluster density, etc... Clustering visualization becomes an indispensable step to help in analyzing and discovering quickly meaningful information of clustering. There are several explorable tasks of clustering visualization such as:

- how clusters are well defined?
- how the clusters are different from each other?
- what the clusters' size is?
- how we can detect the objects belong strongly to the cluster or vice-versa?

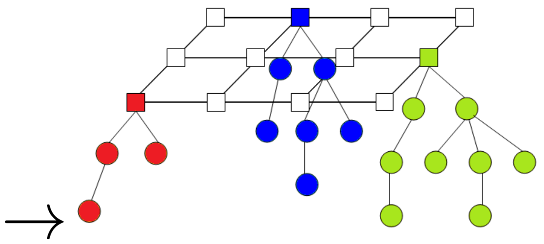
Due to the explosive growth in the quantity, quality of data, especially structured data which is available in form of graphs, trees, sequences, etc..., clustering and visualization become crucial steps to help data analysis. For example, many practical problems in biological, social and information systems can be represented by graphs [West 2000, Bang-Jensen 2008]. An iterative visualization can quickly provide insight information of clustering results that may suggest the adequacy of the solution and what further experiments to conduct. In this thesis, we will present a solution for data visualization. An attractive way to visualize data is to map high dimensional data in low dimensional space, which is also able to preserve topological properties of the input space. Furthermore, a hierarchical structure enables to define a relationship between a pair of objects. Both hierarchical and topological structures are well studied in the clustering problem. The question is how to combine hierarchical and topological structures to deal with the clustering problem?



(a) Topological network: Self-Organizing Map (SOM) [Kohonen 2001].



(b) Hierarchical structure: AntTree [Azzag 2007].



(c) Hierarchical and topological model: SoT implemented the principle of SOM and AntTree.

Figure 1.2: SoT principle

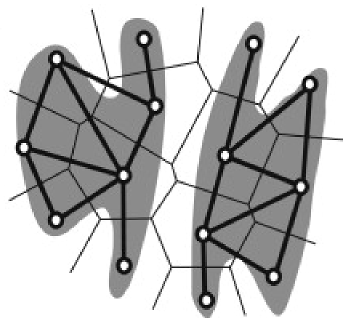
1.2 Objectives and contributions

This thesis is divided into three parts:

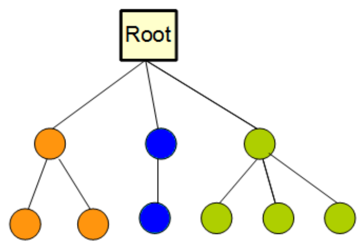
- Part 1: Self-Organizing Trees (SoT)
- Part 2: Growing Self-Organizing Trees (GSoT)
- Part 3: Hierarchical Laplacian Score (HLS)

In the first two parts, we don't only focus on the clustering problem, but also on the problem of clustering visualization. To deal with these two problems at the same time, we propose two new hierarchical and topological models: Self-Organizing Trees (SoT) and Growing Self-Organizing Trees (GSoT) which can perform unsupervised learning, determine cluster confidence and visualize clustering results as trees. We will show how these structures are employed when we obtain clusters of various granularities organized into several levels.

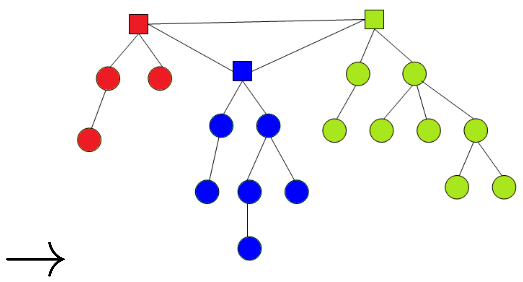
In fact, self-organizing or topological models are based on unsupervised learning. The principle is to map data from high dimensional space to low dimensional space which is non-linear and discrete. This map space is often represented in 1D



(a) Topological network: Growing Neural Gas (GNG) [Fritzke 1995b].



(b) Hierarchical structure: AntTree [Azzag 2007].



(c) Hierarchical and topological model: GSoT implemented the principle of GNG and AntTree.

Figure 1.3: GSoT principle

or 2D [Kohonen 2001]. This dimensional reduction makes topological models a good approach for data visualization. On the other hand, hierarchical structures (or *trees*) are often used to illustrate data arrangement. Typically, in human perspective, a hierarchical tree is an efficient tool and an optimal representation to represent the nature of data structure [Vicente 2004, Hastie 2009]. In this bias, we are interested in particularly in AntTree [Azzag 2007] modeling the ability of artificial ants to build automatically complex structures. Due to the self-assembly rules defined by AntTree, this approach can be adaptive to the self-organizing models.

Our goal is to propose two new models which represent data in a hierarchical and topological structure simultaneously. The advantage of using the hierarchical and topological structure is to visualize a clustering result as multiple hierarchical trees and a topological network as example in Figure 1.2 and 1.3. This way provides much more insights into data structure. Two proposed structures consist of:

- *Network* (or map) describes the topological space where data will be mapped discretely.

- *Network node* (square node) represents a cluster in the topological space. This node is called also *support* or the tree root to which the first tree nodes are going to connect.

- *Topological link* is created between a pair of network nodes if they are considered as neighbors because of a certain neighborhood function.
- *Tree node* (circle node) represents one input object in the map space.
- *Hierarchical link* is created between a pair of tree nodes if they satisfy a similarity test.

Nevertheless, both structures can be easily exploited by descending from a general structure, i.e. topological level (network) to a particular part, i.e. any level of hierarchical trees. This provides useful information about the clustering and data structure as well as data topology. The interesting uses of these models in clustering and visualization will be shown in the experiments studied on various types of data such as vector, image, graph and protein.

The third part of this thesis is devoted to the unsupervised feature selection problem. All along with clustering, feature selection has got a lot of attention in data mining. Data often come with large and high dimension and many clustering algorithms are sensitive to high dimensionality. It leads to the *curse of dimensionality* [Bellman 2003]. An efficient way of handling this problem is to select a subset of relevant features [Liu 1998, Guyon 2006]. As a step in data pre-processing, choosing appropriate and relevant features brings advantages and effects to clustering results differently. It helps in finding clusters efficiently, understanding the data better and reducing data size for efficient storage, collection and processing.

Feature selection consists in selecting the most discriminative and relevant features for data analysis. This is a challenge in feature selection research when dealing with unlabeled data. In literature, Laplacian score is well-known for its ability to reflect the underlying manifold structure. In fact, the hierarchical relations between a pair of objects provided by AntTree may be integrated into the Laplacian score [He 2005] as hierarchical constraints. By constraining the Laplacian score, we focus on ranking the relevance of a feature according to its locality preservation. The performance of this score will be evaluated by both supervised and unsupervised learning methods.

1.3 Organization

This thesis is organized as following:

1. Chapter 2 defines the clustering problem in which the research is placed. In this chapter, we briefly review some aspects of the clustering problem. Then we provide some possible approaches. The principles of these approaches are

highlighted.
In this thesis, we are particularly interested in self-organizing maps which has several beneficial properties such as vector quantization and projection; and hierarchical methods which are an optimal representation of data. Here we will differ the proposed models to the other existing methods which share the same principles in the hierarchical and topological structure.

2. Chapter 3 presents the first method called Self-Organizing Trees (SoT) to address both the clustering problem and data visualization. It offers the ability to represent the data in a 2D map where data are organized in tree-like structures. SoT offers a simple representation of the dataset allowing visualizing clusters as multi-hierarchical trees related to each other due to their topological relationship. It provides remarkably more degrees of freedom. This makes SoT a good tool for clustering and visualization.

3. In Chapter 4, we introduce our second model called Growing Self-Organizing Trees (GSoT) which is an extension of SoT. As a growing algorithm, GSoT can be used for learning topological preservation, clustering, vector quantization, quick data indexation as well as data visualization. Using statistical characteristics provided by hierarchical trees, GSoT is able to accelerate significantly the learning process.

4. Chapter 5 presents our work on unsupervised feature selection. Many different approaches proposed robust scores to evaluate the relevance of each feature. In this work, the hierarchical relations in the tree structure built by AntTree are employed as score constraints to improve the performance efficiency of Laplacian Score.

5. Finally, this thesis ends up with Chapter 6 where we will sum up all our works and propose some perspectives extending from these works.

Chapter 2

Clustering generalities

Contents

Clustering have been intensely studied and successfully applied to many applications in the past several decades. Different approaches have been proposed to deal with specific problems.

Before going into details, we define all the necessary notations in Section 2.1 and data pre-processing before learning in Section 2.2. We resume some similarity measures in Section 2.3.We highlight some clustering approaches which are the most adapted or closest to our algorithms in Section 2.4. To compare the running time of these methods, the complexity of each method will be presented in Section 2.5. Section 2.6 presents some benchmark criteria to measure clustering quality. Finally, this chapter comes up with a conclusion.

2.1 Notations

In this thesis we assume that a set of N objects $\mathcal{X}$ are given. An object i (where $i = 1, .., N$) is described by a vector containing a set of d real values (features), thus an object i is represented by the feature vector $\mathbf{x}_i = (f_{1_i}, ..., f_{d_i}) \in \mathbb{R}^d$. Each object i is thus represented as one data point $\mathbf{x}_i$ in a feature space $\mathcal{X}$. The clustering task is to group a set of data objects based on similarity (or a distance measure) such that the objects in the same group are similar to each other and dissimilar to those in the other groups. The number of clusters is denoted by K, thus we denote that $\mathcal{C} = \{C_1, .., C_K\}$ is a random variable for cluster assignments. Each cluster C_k is often associated with a weight vector or prototype $\mathbf{w}_k$.

2.2 Data pre-processing

Data pre-processing plays a very important role in many learning algorithms [Pyle 1999, Famili 1997]. It has a significant impact to learning results.

2.2.1 Data normalization

In practice, many methods work better after the data has been normalized [Shalabi 2006]. In our research, our datasets are all normalized to minimize redundancy and dependency between data. Normalization scales all numeric features in the range [0,1]. Two below formulas are usually well-known for data normalization

$$f_{r_{new}} = \frac{f_r - f_{min}}{f_{max} - f_{min}} \tag{2.1}$$

where f_{min} and f_{max} are respectively the minimal and maximal values of each feature $r = 1, .., d$

$$\mathbf{x}_{new} = \frac{\mathbf{x} - \mu}{\sigma} \tag{2.2}$$

where μ is the mean of all input vectors; and σ their standard deviation.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \tag{2.3}$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \mu)^2} \tag{2.4}$$

2.2.2 Spectral clustering

Spectral clustering [Belkin 2001, Kannan 2004, Luxburg 2007] is known as a pre-processing step to change the data representation. Spectral clustering techniques make use of the spectrum including eigenvectors and eigenvalues of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. Spectral clustering is usually applied on graphs where the similarity between a pair of objects is not explicitly available. But what is a graph? Which properties does graph possess?

2.2.2.1 Notations and definition

Graph [West 2000, Bang-Jensen 2008] is a combinatorial object described by degree, connectedness, path and edge weight. We denote $G(V, E)$ an undirected graph, a set of vertices V and a set of edges E. A direct link between two vertices v_i and $v_j \in V$ creates an edge $(v_i, v_j) \in E$. The adjacency matrix of G on N vertices is defined as

$$W(i,j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \text{ and } i \neq j, \forall i, j = 1, .., N \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

The adjacency matrix contains only binary information of the connectedness between vertices of G. The other information that can be obtained is the degree. The degree of vertex v_i denoted by $deg(v_i)$ is the number of edges incident to the vertex. The degree matrix is a diagonal matrix defined as:

$$D(i,j) = \begin{cases} deg(v_i) & \text{if } i = j, \, \forall i = 1, .., N \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

We associate to each cluster C_k a prototype denoted $L(C_k)$ whose expression is defined as follows:

$$L(C_k) = \arg \max_{i=1,..,n_{C_k}} (deg(v_i)) \tag{2.7}$$

where the local degree $deg(v_i)$ is the number of internal edges incident to v_i; and n_{C_k} is the cardinality of C_k.

In the graph context, we associate to each $\mathbf{w}_k$ a prototype denoted L_k whose expression is defined as follows:

$$L_k = \arg \max_{i=1,..,n_{C_k}} (deg(v_i)) \tag{2.8}$$

where the local degree $deg(v_i)$ is the number of internal edges incident to v_i; and n_{C_k} is the cardinality of C_k. In C_k, the set of internal edges in the graph is E_k $(\forall i, j = 1..n_{C_k}, E_k = \{(v_i, v_j) \in E\})$. The expression for the local degree of node v_i is as follows:

$$deg(v_i) = \sum_{j=1}^{n_{C_k}} a(i,j) \tag{2.9}$$

where

$$a(i,j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E_k \text{ and } v_i, v_j \in C_k \\ 0 & \text{otherwise.} \end{cases}$$

2.2.2.2 Laplacian matrix

The degree and adjacency matrices are not considered as a metric that explicitly calculate either an appropriate similarity or distance measure. Thus, the Laplacian matrix [Chung 1997, Luxburg 2007] is used. The Laplacian matrix denoted by M_L is a representation of graph through the adjacency and degree matrix from 2.5 and 2.6.

- The unnormalized Laplacian matrix

$$M_L = D - W \tag{2.10}$$

- The normalized Laplacian matrix

$$M_{L,normalize} = D^{-\frac{1}{2}} M_L D^{-\frac{1}{2}} \tag{2.11}$$

  since the degree matrix D is diagonal, its reciprocal square root $D^{-1/2}$ is simply defined as a diagonal matrix, having diagonal entries which are the reciprocals of the positive square roots of the corresponding positive diagonal entries of D.

The study of the Laplacian called spectral graph theory has pointed out that some of the underlying structure can be seen in the properties of the Laplacian. These properties are available in the eigenvalues and eigenvectors of the Laplacian.

$$M_L \mathbf{e} = \lambda \mathbf{e} \tag{2.12}$$

where $\mathbf{e}$ is an eigenvector and λ is an eigenvalue.

The Laplacian and its eigenvectors can be used to describe many properties of graphs [Chung 1997, Luxburg 2007]. A combination of eigenvectors is enough to form a sub-space representing all graph properties. In this sub-space, it is possible to compute vertex similarity or geometric distance.

Following spectral graph theory, we take the first N_e ($N_e \leq N$) smallest eigenvectors of the Laplacian matrix.

$$\mathcal{X}_{new} = \begin{pmatrix} \mathbf{e}_1^T, & \mathbf{e}_2^T, & \cdots & ,\mathbf{e}_{N_e}^T \end{pmatrix} \tag{2.13}$$

where $\forall i = 1, .., N_e$ and $\mathbf{e}_i \in \mathbb{R}^N$. Instead of a $N \times N$ dimensional space, a $N \times N_e$ dimensional one is used. A row $\mathbf{x}_i \in \mathcal{X}_{new}$ illustrates a vertex $v_i \in V$. Thus we have reduced and mapped a graph G into $\mathcal{X}_{new}$ where the similarity between two vertices v_i and v_j is defined as the distance between their respective vectors $\mathbf{x}_i$, $\mathbf{x}_j$.

From this moment in graph context, we consider three notations: an object i, a vector $\mathbf{x}_i$ and a vertex v_i which are identical.

2.2.3 Feature selection

With the goal of data visualization, feature selection is addressed to reduce the large number of dimension. In the unsupervised framework, it is an important challenge due to the absence of class labels that would guide the search for relevant information. Feature selection algorithms can be divided into three categories: the *filter*, the *wrapper* and the hybrid models [Kohavi 1997, Yu 2003, Guyon 2006].

The wrapper principle is shown in Figure 2.1. Wrapper methods always require a

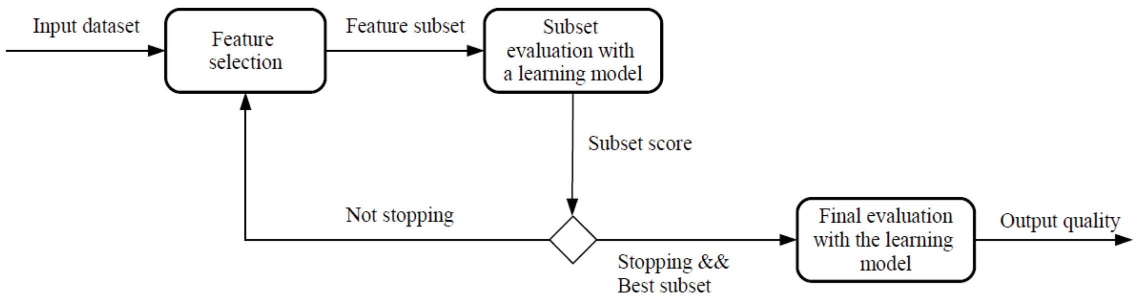


Figure 2.1: Main wrapper steps for unsupervised feature selection

learning model to train each new subset which is generated in certain way. The error rate of the model will be the score for that subset. These processes will be repeated until stopping criteria are fulfilled or a subset that is considered as the *best* one is found. As wrappers train a new model for each subset, the wrappers are usually computationally expensive [Kohavi 1997] and costly to be applied on large datasets; however they usually provide the good performing feature subset for that particular model.

On the other hands, the filter principle is seen in Figure 2.2. Filter methods use a measure instead of the error rate to score each feature. Revelant features often return a good score. A number of features will be selected to be trained by a model that will perform an evaluation of the selection. Without a learning model to train each subset, filters are less computationally costly than wrappers.

In the next sub-sections, we review briefly several well-known filter methods, but

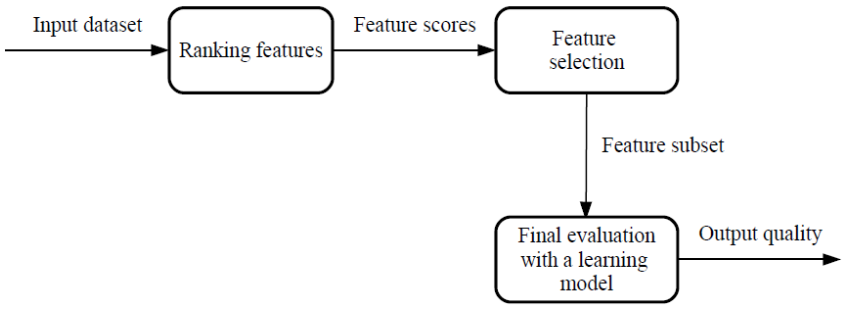


Figure 2.2: Main filter steps for unsupervised feature selection

we firstly give out the necessary notations for the feature selection problem. We use $\mathbf{f}_1, \mathbf{f}_2, ...\mathbf{f}_d$ to denote the d features. Let f_{r_i} denote the i-th sample of the r-th feature, $i = 1, .., N$ and $r = 1, .., d$, thus the row vector is defined as:

$$\mathbf{f}_r = (f_{r_1}, ..., f_{r_N})^T$$

2.2.3.1 Max Variance

Data variance is perhaps the simplest way to select relevant features. This criterion essentially projects the data points along the dimensions of maximum variances [He 2011]. A high variance indicates a large spread in the data distribution. We remark that data objects in the same group are often close in term of distance. The large spread indicates that there are data objects of different groups. As we don't have any information about data class, features that give high variances can be taken into account to best describe the input data. The variance of a feature can be written as

$$var(\mathbf{f}_r) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{f}_{r_i} - \mu_r)^2$$

where $\mu_r = \frac{1}{N}\sum_{i=1}^{N}\mathbf{f}_{r_i}$

2.2.3.2 Laplacian Score

Laplacian score is a recently proposed unsupervised feature selection algorithm [He 2005]. Laplacian Score tends to select relevant features with stronger locality preserving power or its consistency with the manifold structure and with large variances, which are more representative of the high-dimensional data. In order to approximate the manifold structure, one can construct a nearest neighbor graph k-NN. The importance of a feature can be thought of as the degree to which it respects this graph structure. A good feature should be the one on which two data objects are close to each other and they are neighbors. Selecting features with LS is to minimize the following score:

$$L_r = \frac{\sum_{i,j}(f_{r_i} - f_{r_j})^2 S_{ij}}{\sum_i (f_{r_i} - \mu_r)^2 D_{ii}} \tag{2.14}$$

There are many choices of the weight matrix S. Let $\mathcal{N}(\mathbf{x}_i)$ denote the set of k nearest neighbors of $\mathbf{x}_i$. The simplest definition of S is as follows:

$$S_{ij} = \begin{cases} 1 & \mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i) \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Let D be a diagonal matrix, $D_{ii} = \sum_{j=1}^{N} S_{ij}$. When $S_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbor, a direct link is created between these two objects. Then a nearest neighbor graph is defined as: a graph composes of the input objects as its set of vertices and their neighbor link as its set of edges [Eppstein 1997].

We develop LS by following some algebraic steps as follows:

$$\sum_{i,j}(f_{r_i} - f_{r_j})^2 S_{ij} = \sum_{ij}(f_{r_i}^2 + f_{r_j}^2 - 2f_{r_i}f_{r_j})S_{ij}$$

$$= 2\sum_{ij} f_{r_i}^2 S_{ij} - 2\sum_{ij} f_{r_i} S_{ij} f_{r_j} = 2\mathbf{f}_r^T D\mathbf{f}_r - 2\mathbf{f}_r^T S\mathbf{f}_r = 2\mathbf{f}_r^T L\mathbf{f}_r \tag{2.16}$$

where $L = D - S$, L is the Laplacian matrix.

$$\mu_r = \frac{1}{N}\sum_{i=1}^{N} \mathbf{f}_{r_i} = \sum_{i=1}^{N}(f_{r_i}\frac{D_{ii}}{\sum_i D_{ii}})$$

$$= \frac{1}{\sum_i D_{ii}}(\sum_{i=1}^{N} f_{r_i} D_{ii}) = \frac{\mathbf{f}_r^T D\mathbf{1}}{\mathbf{1}^T D\mathbf{1}} \tag{2.17}$$

where $\mathbf{1}$ is the all-ones vector of length N. A new feature vector is defined as:

$$\widetilde{\mathbf{f}}_r = \mathbf{f}_r - \frac{\mathbf{f}_r^T D\mathbf{1}}{\mathbf{1}^T D\mathbf{1}} \tag{2.18}$$

Thus, from 2.17 and 2.18, we have

$$\sum_i (f_{r_i} - \mu_r)^2 D_{ii} = \widetilde{\mathbf{f}}_r^T D\widetilde{\mathbf{f}}_r \tag{2.19}$$

As consequence from 2.16 and 2.19, we have:

$$L_r = \frac{\mathbf{f}_r^T L\mathbf{f}_r}{\widetilde{\mathbf{f}}_r^T D\widetilde{\mathbf{f}}_r} = \frac{\widetilde{\mathbf{f}}_r^T L\widetilde{\mathbf{f}}_r}{\widetilde{\mathbf{f}}_r^T D\widetilde{\mathbf{f}}_r} \tag{2.20}$$

2.3 Similarity measures

It is most common to calculate the similarity between two objects using a distance measure defined on the input space. Since similarity is fundamental to determine a cluster, the distance measure must be chosen carefully. The choice of the distance is very important for the clustering method depending on the data type [Jain 1999, Everitt 2011].

- Euclidean distance, in $\mathbb{R}^d$, describes the geometric distance between two objects.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r=1}^{d} \sqrt{(f_{r_i} - f_{r_j})^2} \tag{2.21}$$

- Mahalanobis distance is a descriptive statistic that provides a relative measure of a data object's distance from a common point.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r=1}^{d} \sqrt{\frac{(f_{r_i} - f_{r_j})^2}{\sigma_r^2}} \tag{2.22}$$

  where σ_r is the standard deviation of the f_{r_i} and f_{r_j} over the sample set.

- Manhattan distance is a measure between two objects, which is the sum of the absolute differences of their vector in the data space.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r=1}^{d} |f_{r_i} - f_{r_j}| \tag{2.23}$$

- Chebyshev distance (or maximum distance) between two vectors is defined as the greatest of their differences along any feature dimension.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \max_{r=1,..,d} (|f_{r_i} - f_{r_j}|) \tag{2.24}$$

- Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle θ between them.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \cos(\theta) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{||\mathbf{x}_i|| ||\mathbf{x}_j||} \tag{2.25}$$

In this work we use the Euclidean distance in Equation 2.26 which is the most adapted for our algorithms and for the nature of the selected datasets.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r=1}^{d} \sqrt{(f_{r_i} - f_{r_j})^2} = \|\mathbf{x}_i - \mathbf{x}_j\| \tag{2.26}$$

2.4 Clustering algorithm categories

The taxonomy of clustering techniques can be described as following [Jain 1988, Jain 1999]:

- Agglomerative vs divisive: An agglomerative approach begins with each object in a distinct cluster and successively merges clusters together until a stopping criterion is fulfilled. A divisive method begins with all learning objects in a single cluster and performs splitting it into smaller clusters until a stopping criterion is met.
- Deterministic vs stochastic: A deterministic model is used in that situation wherein the result is established straighforwardly from a series of conditions. In a situation wherein the cause and effect relationsip is stochastically or randomly determined the stochastic model is used.
- Incremental vs non-incremental: This issue arises when the data set to be clustered is in high dimension. It affects directly to execution time or memory space. The increment is able to reduce the number of objects examined during execution or reduce the size of data structures used in the algorithm operations.

In the next sections of this chapter we will introduce the main families of clustering algorithms:

- Centroid-based clustering algorithms,
- Hierarchical clustering algorithms,
- Density-based clustering algorithms,
- Distribution-based clustering,
- Hybrid clustering algorithms.

2.4.1 Centroid-based clustering algorithms

The centroid-based clustering algorithms aim to group N observations into a number of clusters in which each observation belongs to the cluster with the nearest mean. Clusters are often represented by a weighted vector (or prototype). The goal is now to determine K clusters ($\mathcal{C} = \{C_1, .., C_K\}$ and $K \leq N$) into which N objects are divided in the way that all objects found in a cluster C_i ($i = 1, .., K$) are similar. A similarity measure (or a distance measure) is a metric on the feature space used to quantify data similarity. The clustering problem is known as an optimization problem of NP-hard, and thus the common approach is to search only for approximate solutions. One of the most well-known algorithms of this family is K-means.

2.4.1.1 K-means algorithm

The most common example of clustering algorithms is K-means [Jain 1988]. Clusters are represented by a mean vector called weighted vector or prototype $\mathbf{w}_k$, where $k = 1, .., K$, which may not necessarily have a physical point in data space. Thus we can re-define the clustering problem as an optimization problem: find the cluster centers such that the intra-class variance is minimized, i.e the sum of squared distances from each object within a cluster to its corresponding prototype. To minimize intra-class variances means to minimize the quantization error expressed as following:

$$error = \arg\min \sum_{k=1}^{K} \sum_{i=1}^{n_{C_k}} \|\mathbf{x}_i - \mathbf{w}_k\|^2 \tag{2.27}$$

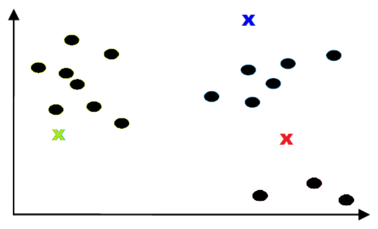
where n_{C_k} is the cardinality of cluster C_k.

The minimization of Equation 2.27 is the known problem: minimize the quantization error of data gathering within each cluster [Everitt 2011]. To minimize this error, each object should be assigned to the nearest cluster center. For each assignment, the prototype vector will move toward the assigned object. The quantization error will be minimized step by step, iteration by iteration. In the mean time, K-means also tries to maximize the distance between a couple of two centroids. The algorithm is resumed in Figure 2.3 and written as following:

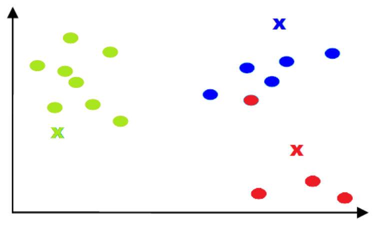
Algorithm 1 K-means algorithm

```
1:  initialize randomly $K$ prototypes
2:  repeat
3:     for $i = 1$ to $N$ do
4:        $k = \arg\min_{k=1,..,K} \|\mathbf{x}_i - \mathbf{w}_k\|^2$
5:        $C_k = C_k \cup \mathbf{x}_i$      // assign $\mathbf{x}_i$ to cluster $C_k$
6:     end for
7:     for $k = 1$ to $K$ do
8:        $\mathbf{w}_k = \frac{1}{n_{C_k}} \sum_{j=1}^{n_{C_k}} \mathbf{x}_j$      // update prototype $k$, where $n_{C_k}$ is the cardinality of
          cluster $C_k$
9:     end for
10: until stopping criterion has been fulfilled
```

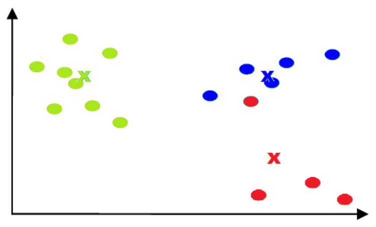
K-means is the heuristic algorithm which has few defaults: it converges to a local optimum, and the result depends on K and the initial prototypes. Therefore, some variants have been developped including topological models for example: SOM and Neural Gas that give more advantages, because of the topological preservation.



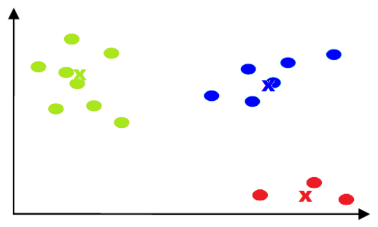
(a) Initial step: Data distribution is given and $K = 3$ initial prototypes are randomly generated within the data space.



(b) Assignment step: The clusters are formed by assigning every single object to the cluster whose prototype is the nearest to this object.



(c) Update step: The prototypes are updated and move to the local minimum and to cover data distribution.



(d) The algorithm repeats until convergence or a stopping criterion has been fulfilled.

Figure 2.3: Clustering with K-means

2.4.1.2 Self-Organizing Map

Based on K-means, SOM proposed by Kohonen [Kohonen 2001] is a type of artificial neural network for unsupervised learning. SOM has the ability of creating spatially organized internal representations of input objects and their abstractions. As in Figure2.4, SOM produces a low-dimensional (1D or 2D) discretized representation (called a map or network) from the high-dimensional space of the input objects. SOM uses a neighborhood function to preserve the topological properties of the input space. SOM forms a discretely topological map where similar objects are grouped close together and dissimilar ones apart. Like most artificial neural networks [Haykin 1998], SOM has two-fold objectives:

- Training map (projection): build the topological map using the input objects. A map consists of a number of network nodes arranged according to a structure defined a priori. The usual arrangement of network nodes is a 1D or 2D, hexagonal or rectangular grid. Associated with each network node is a prototype (or a weight vector) of the same dimension as the input data objects.

- Mapping (quantization): put the input objects into a non-linear, discrete map. Vector quantization techniques encode a data object in a prototype such that the distance from this object to the best match prototype is closest. This process will respect the neighborhood function to preserver data topology. Data objects which are similar into the input space will be put onto neighbor network nodes.

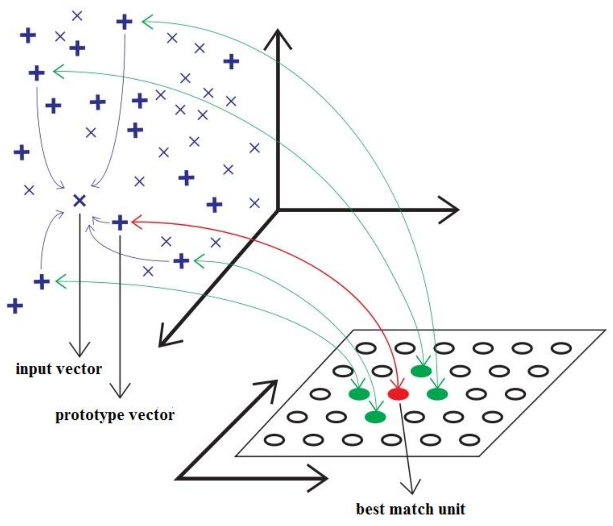


Figure 2.4: SOM principles: mapping and quantization

At the start of the learning, a discrete topological map of size $p \times q = K$ is initialized. We denote $\mathcal{C} = \{C_1, .., C_K\}$ where C_i $(i = 1, .., K)$ is a network node. $\mathcal{C}$ is associated with $\mathcal{W} = \{\mathbf{w}_1, .., \mathbf{w}_K\}$ where $\mathbf{w}_i$ is the prototype associated with the network node C_i. For each pair of network nodes C_c and C_r on $\mathcal{C}$, their mutual influence is defined by the function $\mathcal{K}^{\mathcal{T}}(\delta(C_c, C_r))$ [Elghazel 2009]. A Gaussian function is a common choice for $\mathcal{K}$ that will shrink with time. Due to the use of this function $\mathcal{K}$, in the training the whole neighborhood network nodes move along in the same direction towards the learning data, similar data tend to be put in the adjacent network nodes [Kohonen 2001]. $\delta(C_c, C_r)$ is defined as the shortest distance (Euclidean distance, see Equation 2.26) between two network nodes ($\delta(C_c, C_r) = \min_{i=1,..,n_c; j=1,..,n_r} \|\mathbf{x}_i - \mathbf{x}_j\|^2$ where n_c, n_r are respective the number of data grouped in the network node C_c, C_r). T represents the temperature function that controls the size of the neighborhood and the algorithm convergence. Assume that N_{iter} is the number of iterations and ith is the current iteration; T_{max} and T_{min} are respectively denoted for the initial and final temperature, the expression to calculate T is as following:

$$T = T_{max}(\frac{T_{min}}{T_{max}})^{\frac{ith}{N_{iter}-1}}$$

We present two versions of SOM algorithm: stochastic (Algorithm 2) and batch (Algorithm 3), both aim to minimize the cost function presented in Equation 2.28.

$$\mathcal{R}(\phi, \mathcal{W}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T(\delta(\phi(\mathbf{x}_i), C_k)) \|\mathbf{x}_i - \mathbf{w}_k\|^2 \tag{2.28}$$

where $\phi(\mathbf{x}_i)$ is the assignment function, or more presicely, $\phi(\mathbf{x}_i)$ returns the network node to which $\mathbf{x}_i$ get assigned.

The learning steps are the same as K-means:

- Initialization: initalize the map structure, i.e: the number of network nodes (or K clusters), the arrangement shape: hexagonal or rectangular and the initial prototypes.
- Assignment: assign data objects to the nearest prototype (best match unit). This assures that the cost function $\mathcal{R}(\phi, \mathcal{W})$ is minimized in regard to the assignment function ϕ assuming that the prototype vectors are constant. Additionally, this step maps data into network nodes.
- Update: re-compute the prototype vectors. The prototypes and their neighbors move along together towards the assigned data such that the map tends to approximate the data distribution. It includes minimizing the cost function $\mathcal{R}(\phi, \mathcal{W})$ in regard to the prototypes vectors assuming that data are all assigned to the best match unit.

Unlike the stochastic version, the batch one has taken into account the neighborhood function in the assignment step. This is to minimize rigorously the cost function in Equation 2.28.

Stochastic SOM

Algorithm 2 Stochastic SOM version

```
1: initialize $K$ prototypes and $\mathcal{W}$
2: while stopping criteria have not been fulfilled do
3:   for $i = 1 \to N$ do
4:     $\phi(\mathbf{x}_i) = \arg\min_{k=1,..,K} \|\mathbf{x}_i - \mathbf{w}_k\|^2$   // Find the best match unit to the current selected vector.
5:     for all $C_r$ is a neighbor of $\phi(\mathbf{x}_i)$ (including $\phi(\mathbf{x}_i)$ itself) do
6:       $\mathbf{w}_r = \mathbf{w}_r + \mathcal{K}^T(\delta(C_{\phi(\mathbf{x}_i)}, C_r))(\mathbf{x}_i - \mathbf{w}_r)$   // Update the nodes in the neighborhood of $\phi(\mathbf{x}_i)$ (including the node $\phi(\mathbf{x}_i)$ itself) by pulling them closer to the input vector.
7:     end for
8:   end for
9: end while
```

Batch SOM

Algorithm 3 Batch SOM version

```
1: initialize $K$ prototypes and $\mathcal{W}$
2: while stopping criteria have not been fulfilled do
3:   for $i = 1 \to N$ do
4:     $\phi(\mathbf{x}_i) = \arg\min_{k=1,..,K} \mathcal{K}^T(d(\mathbf{x}_i, \mathbf{w}_k))\|\mathbf{x}_i - \mathbf{w}_k\|^2$   // Find the best match unit to the current selected vector.
5:     $C_{\phi(\mathbf{x}_i)} = C_{\phi(\mathbf{x}_i)} \cup \{\mathbf{x}_i\}$   // Put $\mathbf{x}_i$ into cluster $\phi(\mathbf{x}_i)$
6:   end for
7:   for $k = 1 \to K$ do
8:     $\mathbf{w}_k = \frac{\sum_{r=1}^{K} \mathcal{K}^T(\delta(C_c,C_r)) \sum_{j=1}^{n_{C_r}} \mathbf{x}_j}{\sum_{r=1}^{K} \mathcal{K}^T(\delta(C_c,C_r)) n_r}$   // Update prototype vectors where $n_r$ is the number of data found in cluster $r$
9:   end for
10: end while
```

2.4.1.3 Neural Gas

As a robustly converging alternative to the K-means, Neural Gas [Martinetz 1991, Fritzke 1991] is inspired by the SOM. While SOM map dimensionality must be chosen a prior; depending on the data distribution, the topological network of neural gas may have different arrangement. Neural Gas is more flexible network capable of quantizing tological data and learning the similarity among the input objects without defining a network topology. Unlike SOM, the adaptation strength in Neural Gas is constant over time and only the best match prototype and its direct topological neighbors are adapted.

Given a network of K clusters $\mathcal{C} = \{C_1, .., C_K\}$ associated with K prototypes $\mathcal{W} = \{\mathbf{w}_1, .., \mathbf{w}_K\}$. They are adapted independently of any topological arrangement of the network nodes within the network. Instead, the adaptation step is affected by the topological arrangement within the input space. For each object $\mathbf{x}_i$ is selected, prototypes will be ajusted by distortions $\mathcal{D}(\mathbf{x}_i, C_k) = \|\mathbf{x}_i - \mathbf{w}_k\|, \forall k = 1, .., K$. The resulting adaptation rule can be described as a "winner take most" instead of a "winner take all" rule [Fritzke 1991]. The winner network node denoted by k_0 is determined by the assignment function $k_0 = \phi(\mathbf{x}_i) = \arg\min_{k=1,..,K} \|\mathbf{x}_i - \mathbf{w}_k\|$. The network node adjacent denoted by k_1 to the winner node develops connection between each other which is manipuled by a matrix $\mathcal{S}$ representing the neighborhood relationships among the input data.

$$\mathcal{S}_{ij} = \begin{cases} 1 & \text{if a connection exists between } C_i \text{ and } C_j \ (\forall i, j = 1, .., K, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

When an object is selected, the prototypes moves toward this object by adjusting the distortion $\mathcal{D}(\mathbf{x}_i, C_{k_0})$. Furthemore, it is in the way controlled by a neighborhood function $\mathcal{K}^T$. In [Fritzke 1991], this function is fixed, e.g. $\mathcal{K}^T = \exp^{knn_k/T}$ where knn_k is the number of neighborhood network nodes of C_k. This affects directly to the adaptation step for $\mathbf{w}_k$ which is determined by:

$$\mathbf{w}_k = \mathbf{w}_k + \varepsilon\mathcal{K}^T(\mathbf{x}_i - \mathbf{w}_k)$$

To capture the topological relations between the prototypes, each time an object is presented, the connection between k_0 and k_1 is established by setting $\mathcal{C}_{k_0,k_1}$ from zero to one. Each connection is associated with an "age" variable. Only the connection between k_0 and k_1 is reset, the other connections of k_0 age, i.e. their age increment. When the age of connection exceeds a specific lifetime Max_{age}, it is removed, i.e. $\mathcal{C}_{ij}$ is re-set to zero. The way to update the age of the connections is to increase with each incoming input object is learnt. Finally, Neural Gas can be summarized by the following algorithm:

Algorithm 4 Neural Gas algorithm

```
1: Initialize $K$ prototypes and set all $\mathcal{S}_{ij}$ to zero
2: for all $\mathbf{x}_i \in \mathcal{X}$ do
3:     determine the sequence $(C_{k_0}, C_{k_1}, ..., C_{k_{N-1}})$ such that
           $\|\mathbf{x}_i - \mathbf{w}_{k_0}\| < \|\mathbf{x}_i - \mathbf{w}_{k_1}\| < .. < \|\mathbf{x}_i - \mathbf{w}_{k_{K-1}}\|$
       // $\mathbf{w}_{k_0}$ is the best match prototype, i.e the nearest prototype; $\mathbf{w}_{k_1}$ is the second nearest prototype to $\mathbf{x}_i$
4:     for all $C_j$ with $\mathcal{S}_{k_0,j} == 1$ do
5:         $\mathbf{w}_j = \mathbf{w}_j + \varepsilon\mathcal{K}^T(\mathbf{x}_i - \mathbf{w}_j)$    // perform an adaptation step for the prototypes
6:     end for
7:     if $\mathcal{S}_{k_0,k_1} == 0$ then
8:         $\mathcal{S}_{k_0,k_1} = 1$    // create a topological connection between $C_{k_0}$ and $C_{k_1}$
9:         $age_{k_0,k_1} = 0$    // set age for this connection
10:    end if
11:    for all $C_j$ with $\mathcal{S}_{k_0,j} == 1$ do
12:        $age_{k_0,j} = age_{k_0,j} + 1$    // increase the age of all connections of $k_0$ by one
13:        if $age_{k_0,j} > Max_{age}$ then
14:            $\mathcal{S}_{k_0,j} = 0$    // remove all connections of $k_0$ which exceeded their age
15:        end if
16:    end for
17: end for
```

In these two algorithms, stopping criteria can be either:

- a number of iterations
- a threshold for the quantization error.

2.4.1.4 Growing Neural Gas

The incremental variant of Neural Gas [Fritzke 1995a, Fritzke 1995b], Growing Neural Gas (GNG), is introduced, which is able to learn the important topology in the data. This method has no input parameters which change over time and is able to continue learning, adding network units and connections. As an incremental variant of Neural Gas, GNG inherits its principle; however it does not impose the strict network-topology preservation rule. The network incrementally learns the topological relationships inherent in the dataset, and continues until a stopping criterion is fulfilled. Before learning, only $K = 2$ prototypes are initialized. Step by step, after a certain number of iterations (called epoch), a new network node is successively added into the topological network. But how to add a new network node? Now, this relates to quantization error. In the clustering problem, the goal is always to minimize the quantization error of datasets or data within the clusters. Therefore, the cluster that provides a high value of quantization error is not a good one. We should divide this cluster into smaller clusters. GNG finds the two clusters C_p and C_q which have the highest quantization error. Then a new node is inserted halfway between these two nodes by the following expression:

$$\mathbf{w}_{new} = \frac{1}{2}(\mathbf{w}_p - \mathbf{w}_q) \tag{2.29}$$

The node insertion will be repeated until a stoping criterion is fulfilled.

2.4.2 Hierarchical clustering algorithms

Hierarchical clustering methods [Vicente 2004, Jain 1999] share the same principle based on the similarity between a couple of objects. The intuition is that objects are more related to nearby objects than to objects farther away. A hierarchical structure (tree) will be formed according to different similarities. These algorithms do not provide directly a clustering of the data set; however, a data hierarchy is provided.

2.4.2.1 Agglomerative hierarchical clustering

Agglomerative hierarchical clustering (AHC) generates a hierarchical binary cluster tree or dendrogram [Hastie 2009]. A dendrogram consists of many links that connect data objects. This algorithm is a bottom-up approach:

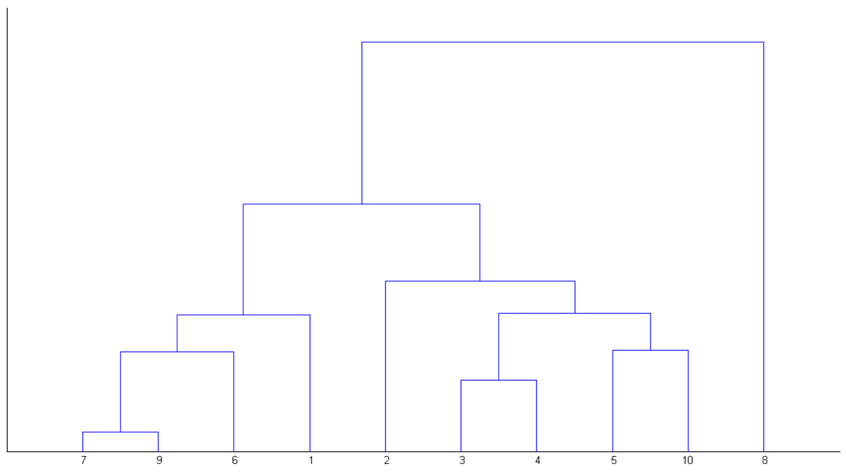


Figure 2.5: Example of the AHC structure

- Initially all objects represent a separate cluster

- Successively, a pair of clusters is merged into a new cluster according to their similarity.
- The algorithm proceeds until a stopping criterion is fulfilled. Stopping criterion can be distance criterion where the merge clusters are too far apart; or number criterion where a number of clusters has been reached.

The very simple algorithm is presented in Algorithm 5. The nearest clusters in Line 2 can be variously defined as:

- Single-linkage: the link between two clusters is made by a single pair of objects that are closest to each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose objects are involved [Hastie 2009].
$$d(C_i, C_j) = \min_{\mathbf{x}\in C_i, \mathbf{x}'\in C_j} d(\mathbf{x}, \mathbf{x}')$$
- Complete-linkage: the link between two clusters contains all object pairs, and the distance between clusters is equal to the distance between those two objects that are farthest away from each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose objects are involved [Everitt 2011].
$$d(C_i, C_j) = \max_{\mathbf{x}\in C_i, \mathbf{x}'\in C_j} d(\mathbf{x}, \mathbf{x}')$$
- And the other distances that have been presented in Section 2.3 can be employed to build the AHC structure.

Algorithm 5 AHC algorithm

```
Input: Each objects $\mathbf{x}_1, .., \mathbf{x}_N \in \mathcal{X}$ is in its own cluster $C_1, .., C_N$
1: repeat
2:     merge the nearest clusters involving $C_i$ and $C_j$
3: until only one cluster is left
```

2.4.2.2 Minimum Spanning Tree

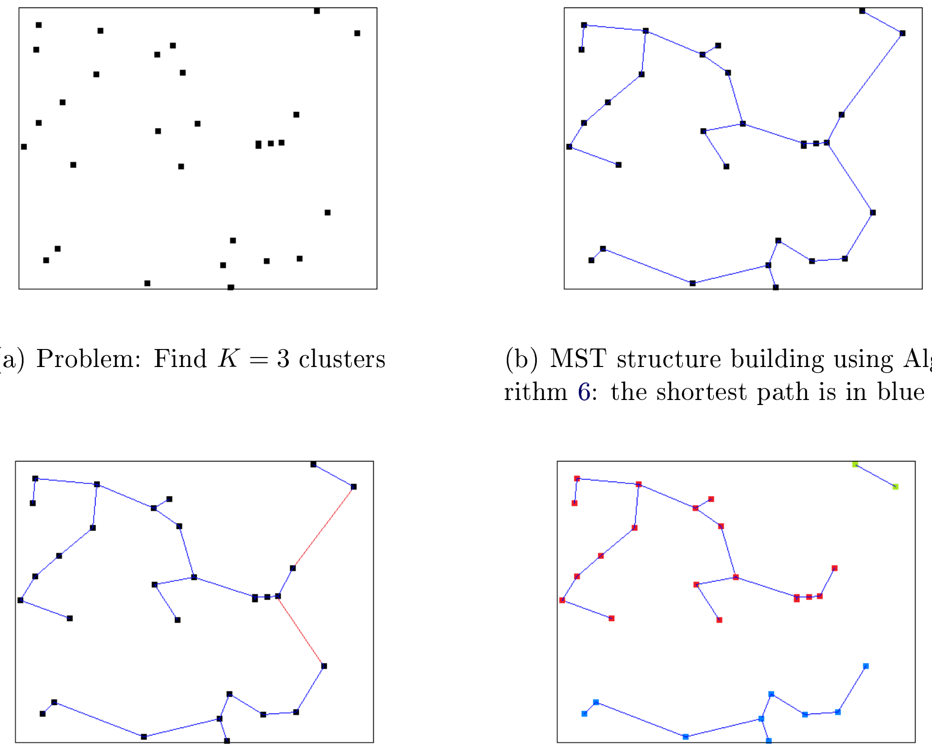
In the graph context, we assume that a undirected and weighted graph with a set of vertices and edges is given. A spanning tree of this graph is a sub-graph such that it finds the way to connect all the vertices using the given set of edges. Thus, a graph can have many different spanning trees. A minimum spanning tree (MST) is the spanning tree having the lowest sum of the weights of the edges in spanning trees.

Algorithm 6 MST: Prim's algorithm

```
Input: A set of objects $\mathcal{X}$ and $d(\mathbf{x}_i, \mathbf{x}_j)$ is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j \in \mathcal{X}$
1: $\mathcal{X}_{new} = \{\mathbf{x}_i\}$    // where $x_i$ is randomly initialized and $i = 1, .., N$
2: $tree = \{\}$
3: repeat
4:     for all $\mathbf{x}_i \in \mathcal{X}_{new}$ do
5:         $\mathbf{x}_j = \arg\min_{\forall \mathbf{x}_j \notin \mathcal{X}_{new}} d(\mathbf{x}_i, \mathbf{x}_j)$    // find $\mathbf{x}_j$ which has the shortest distance
           to $\mathbf{x}_i$ and $\mathbf{x}_j$ has not yet found in $\mathcal{X}_{new}$
6:         $\mathcal{X}_{new} = \{\mathbf{x}_j\} \cup \mathcal{X}_{new}$    // add $\mathbf{x}_j$ to $\mathcal{X}_{new}$
7:         $tree = \{(\mathbf{x}_i, \mathbf{x}_j)\} \cup tree$    // add $(\mathbf{x}_i, \mathbf{x}_j))$ to tree
8:     end for
9: until $\mathcal{X}_{new} == \mathcal{X}$    // $\mathcal{X}_{new}$ contains every object of $\mathcal{X}$
```

In the clustering context, if we consider data objects as graph vertices, links between a pair of objects as graph edges, and distances between a pair of objects as graph weights. MST structure can be considered as a hierarchical tree based on data proximity. More precisely, it finds a subset of the links that forms a tree. A MST includes every input objects such that the total distance in the tree is minimized.

In MST structure, there is only distance metric between data, there is no notion of cluster prototype or centroid though. However, the main principle is that the clusters should be far away from each other, therefore, larger distances should be penalized. Ideally, to find K clusters [Grygorash 2006], the $K-1$ longest links in the tree have to be removed as example in Figure 2.6. There are two common techniques to build MST: Prim [Prim 1957] and Kruskal [Kruskal 1956]. We present Prim's algorithm in Algorithm 6.



(a) Problem: Find $K = 3$ clusters

(b) MST structure building using Algorithm 6: the shortest path is in blue

(c) Finding the $K-1$ longest links (red) in the obtained MST structure.

(d) $K = 3$ clusters are formed

Figure 2.6: MST for clustering

2.4.3 Hybrid clustering algorithms

Hybrid algorithms usually refer to the ones that combine different approaches to solve a specific problem. In this thesis, the study concerns hybrid algorithms providing hierarchical and topological structure for clustering and visualization. Of course, the idea of creating hierarchical and topological networks is not new. Finding hierarchical relations among the input data can be addressed conveniently within the data topology. Recenlty, many tree-structured variants of SOM or GNG have been developped. A tree-structured representation provides remarkably more degrees of freedom to analyze input data. It allows to compare directly the similarity between data objects in the hierarchical structure.

The methods are different from each other in many features. In this section, we just discuss about the main principles and the way of building the hierarchical and topological structure of these methods. In Table 2.1 we make a comparison of the hierarchical and topological approaches. The aspects to compare are network size, hierarchy level, number of output trees, type of tree node and number of data gathering in each tree node. As it can be noted in this table, only SoT and GSoT output multiple trees while the others build one single tree.

Method	Network	Hierarchy	Trees	Node type	# Data/node
GH-SOM	variant	layers/maps	single	vector	single
S-TREE	fixed	prototypes	single	vector	variant
SOM-AT	fixed	data	single	attribute	single
TreeSOM	fixed	prototypes / data	single	vector	variant
TS-SOM	variant	layers/prototypes	single	vector	single
HOGNG	variant	layers	single	vector	variant
TreeGNG	variant	prototypes	single	vector	variant
SoT	fixed	data	multiple	vector	single
GSoT	variant	data	multiple	vector	single

Table 2.1: Numerous features in structure of different hierarchical and topological methods

2.4.3.1 Growing Hierarchical Self-Organizing Maps

Growing Hierarchical Self-Organizing Maps (GH-SOM) [Dittenbach 2000] proposed a hierarchical structure of multiple layers where each layer consists of a number of independent self-organizing maps (Figure 2.7). Each layer is exactly a SOM map. The model starts with a virtual layer 0 which consists of only one single network node initialized as the average of all input data. The training process basically continues with the layer 1 which must be initalized a prior. After a certain number of training iterations, the network node with the highest intra-class variance is selected. In between this network node and its most dissimilar neighbor in terms of similarity (or distance measure), either a new row or a new column of network nodes is inserted. This new map will be added to the hierarchy as the next layer and the input data mapped on the respective layer are self-organized in this new map. This principle is repeated with any further layers until the quantization error has reached a threshold. In this method, the hierarchy represents map evolution during the unsupervised training process.

2.4.3.2 S-Tree

S-TREE [Campos 2001] is used for unsupervised learning to construct a tree-structured model. S-TREE divides the input space into a nested structure which

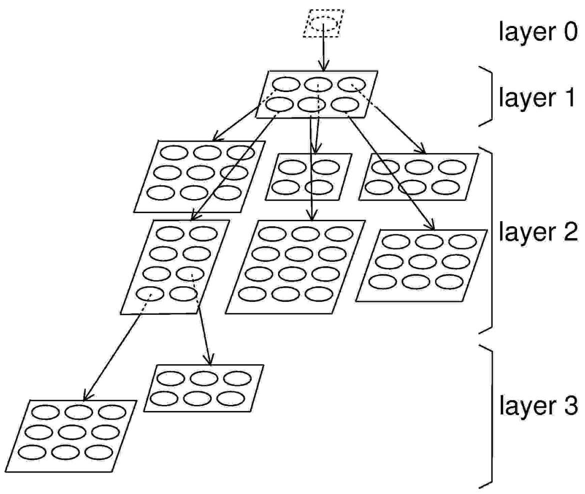


Figure 2.7: GH-SOM structure extracted from [Dittenbach 2000]

corresponds to a binary tree (Figure 2.8). Each tree node is associated with a prototype vector. The learning begins with the root node. Each input object is traversed with a search from the root to the leaf node. At each internal node, a test is performed to compare the input vector to the prototypes of the two child nodes and then select the child node whose the prototype is the closest to this object. After a leaf node has been found, S-TREE performs a test to decide whether the tree node will be modified. If the distortion at the winning leaf node is too great, this node will be splitted. If the tree has reached its maximum size, the tree will be pruned.

2.4.3.3 Self-Organizing Map of Attribute Trees

Self-Organizing Map of Attribute Trees (SOM-AT) [Peura 1999] is based on adjusting schemes for attribute trees (Figure 2.9). In the input, the standard vectors are replaced by attribute trees. The key idea is based on subtree indexing: subtrees are recursively represented and matched according to their topological descriptors. The goal is to maximize the number of matched branches and not to violate the structure of the input trees by looping or stretching.

2.4.3.4 TreeSOM

TreeSOM [Samsonova 2006] represents SOM map as a consensus tree (Figure 2.10) which represents an average of a set of trees with frequencies of occurrence of its branches compared to the set of all trees representing reliable clusters as subtrees. In this method, only the leaf nodes may get many data elements, and other nodes none at all.

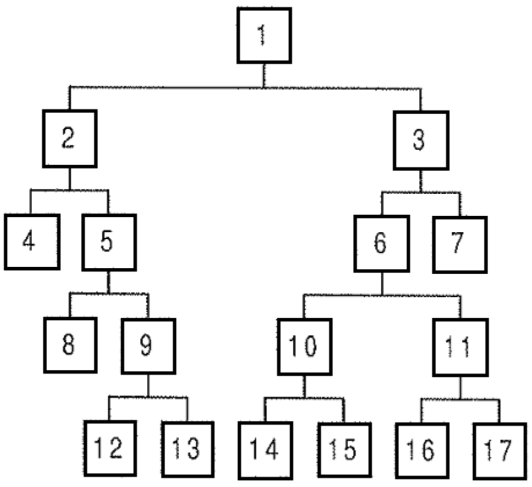


Figure 2.8: S-TREE structure extracted from [Campos 2001]

2.4.3.5 Tree-Structured Self-Organizing Maps

Tree-Structured Self-Organizing Maps (TS-SOM) [Koikkalainen 2007] has apparently the same idea with GH-SOM for the use of layers to build a hierarchy (Figure 2.11). Instead self-organzing maps are built at each layer, each tree node of TS-SOM corresponds to the prototype which is associated with a subgroup of data. Similar to GH-SOM, the centroid on the layer 0 defines the mean of all data. Then the next layer l will be generated with $n_{l-1} \times 2^D$ network nodes where D is the map dimension and usually, $D = 1$ or 2; and n_{l-1} is the number of network nodes in the previous layer. The training is repeated layer by layer using knowledge about the prototypes of the frozen layer in the assignment on the next layer. The assignment is restricted to the small set of objects that are linked to the prototypes and its neighbors on the previous level. As a significant advantage of the TS-SOM, this reduces the computational complexity.

2.4.3.6 Hierarchical Overlapped Growing Neural Gas

In Hierarchical Overlapped Growing Neural Gas (HOGNG) [Cao 2003], the network (Figure 2.12) is initialized with just one layer called the base layer. The base layer is trained using the unsupervised GNG algorithm. Having completed the learning in the base layer, a new layer called SGNG is created for each node in the base layer. The learning process continues for each layer consisting in a small group of data given by the base layer. The SGNG network consists of a hidden layer and an output layer. The hidden layer of each SGNG network in the second level is initialized using the values of its base layer unit (i.e., root unit) and the direct topological neighbors of

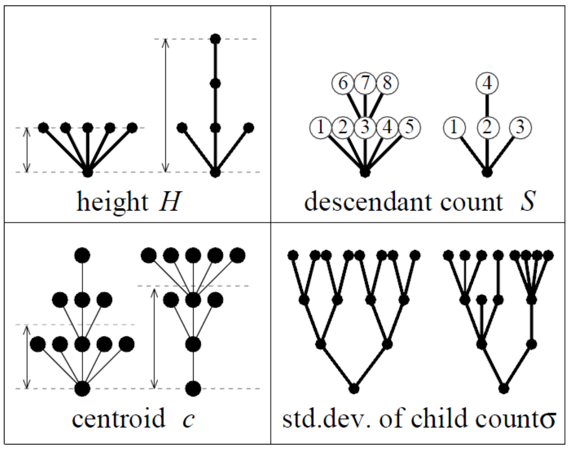


Figure 2.9: SOM-AT structure extracted from [Peura 1999]

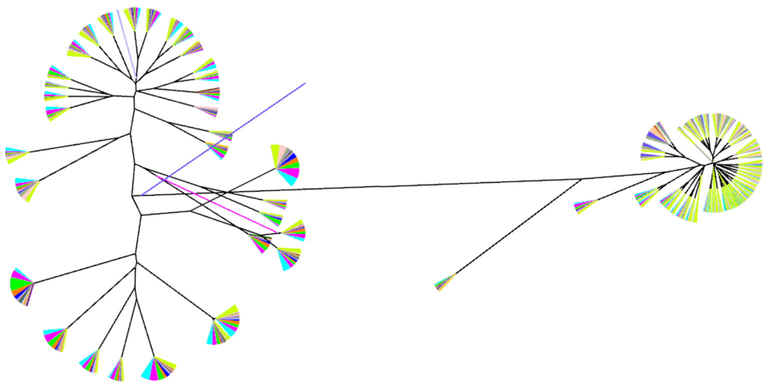


Figure 2.10: TreeSOM structure extracted from [Samsonova 2006]

the base layer unit, with their connections duplicated.

2.4.3.7 TreeGNG

TreeGNG [Doherty 2005] (Figure 2.13) extends the GNG algorithm by maintaining a time history of the learned topological mapping. In (i), a network is initialized with two nodes linked by a single topological connection. The tree consists of a single root node R. In (ii), following a period of standard GNG dynamics, the dashed connection is deleted from the GNG network. As this connection is deleted in (iii), the tree splits and node R grows two children A and B to represent the increase in the number of tree nodes. A growth window is opened for node R. The step repeats until a stopping criterion has been fulfilled. From (iv) to (viii), the tree structure expands more nodes and the leaf nodes represent obtained clusters.

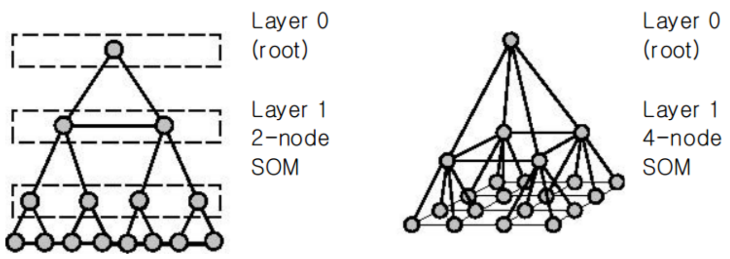


Figure 2.11: TS-SOM structure extracted from [Koikkalainen 2007]

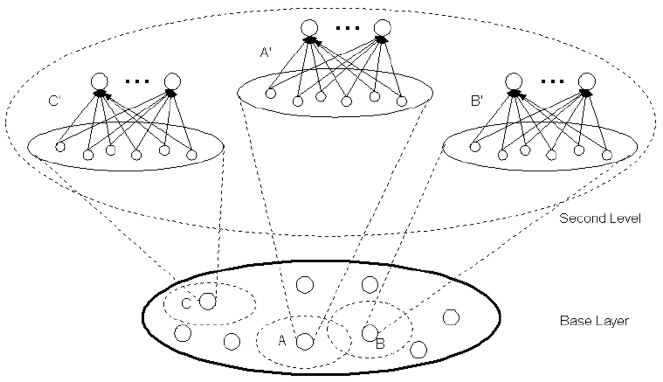


Figure 2.12: HOGNG structure extracted from [Cao 2003]

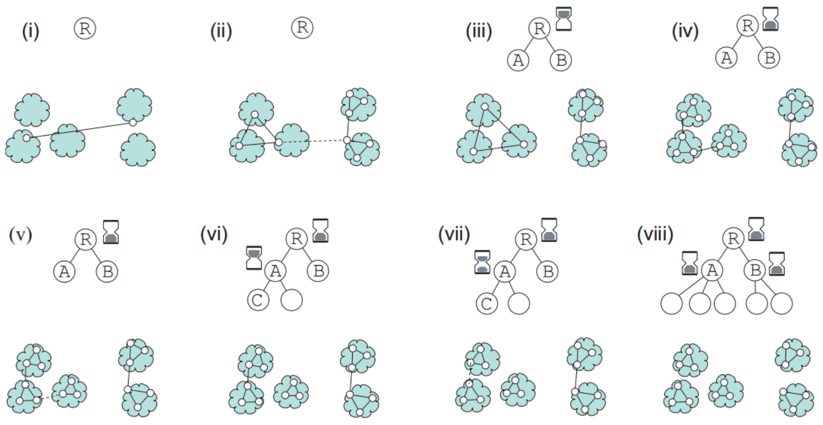


Figure 2.13: TreeGNG structure extracted from [Doherty 2005]

2.4.4 Bio-inspired clustering algorithms

Bio-inspired algorithms belong to a category of algorithms that imitate the way nature performs. Recently this category has been quite popular, since numerous problems can be solved without rigorous mathematical approaches.

In DNA computing [Adleman 1994, Amos 2005], self-assembling models concern the sequential construction of a given graph, referring to the Accretive Graph Assembly Problem (AGAP) [Reif 2005, Angelov 2006]. An AGAP instance is a triplet (i.e. a graph G_A with weighted edges, a seed vertex in G_A and a system temperature) for which one determines if a vertex sequence exists from the seed to build a graph. The edge weights influence the attraction and repulsion forces for a given temperature. After stabilizing the system, an assembled sub-graph is obtained. In some studies, the graph structure is determined in advance, and the problem to be solved is finding a sequence of actions to build the target graph [Garzon 1999, Danos 2005, Danos 2007].

In robotics, interesting studies consider "modular robots", as in [Murata 1994], where complex shapes can be achieved, or the Swarm-Bot project [Mondada 2004], where collective robot assemblages increase their abilities to drag objects or cross empty spaces. A recent work surveyed this topic [Groß 2007].

The self-assembly behavior of individuals can be observed in several insect species, including ants and bees [Anderson 2002]. This biological phenomenon is a particular case of self-organization that represents the minimum assembly of two similar entities with the same connecting mechanisms without human intervention [Camazine 2001, Krasnogor 2005].

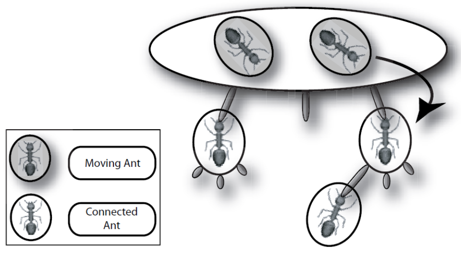
The principles of bio-inspired clustering algorithms are often based on:

- Genetic algorithms
- Ant-based algorithms
- Swarm Intelligence
- etc...

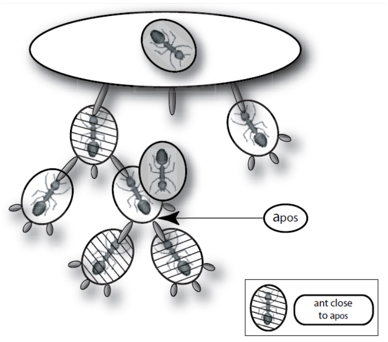
In the next sections, we give a review on the bio-inspired clustering algorithm AntTree from which our researches extend.

2.4.4.1 AntTree

AntTree [Azzag 2007] provides the hierarchical structure where each tree node represents one data object. The main principles are the following (Figure 2.14(a)): Initially, all objects are placed on the *support* which corresponds to the tree roof. An object will connect to the support or a connected object in order to connect itself to a convenient location in the tree structure. The way to connect an object to another depends on a similarity test (Figure 2.14(b)). Once all the objects are connected in the tree, the tree structure can be interpreted.



(a) General principle of AntTree for tree building with self-assembly rules (a data object is represented by an ant). [Azzag 2007]



(b) Connecting rules to find the nearest ant

Figure 2.14: AntTree principles

Considering the clustering problem defined in the previous chapter, during the assembly of the structure, each object $\mathbf{x}_i$ will be either:

- moving on the tree: $\mathbf{x}_i$ moving over the support or over an other object denoted by $\mathbf{x}_{pos}$, but $\mathbf{x}_i$ is not connected to the structure. It is thus completely free to move on the support or toward another object within its neighborhood. If $\mathbf{x}_{pos}$ denotes the object where $\mathbf{x}_i$ is located on, then $\mathbf{x}_i$ will move randomly to any immediate neighbors of $\mathbf{x}_{pos}$ in the tree.

- connected to the tree: $\mathbf{x}_i$ can no longer move anymore from the structure. Each object has only one connection with other ants.

Let us denote: $\mathbf{x}^+$, $\mathbf{x}^-$ are respectively the most similar and dissimilar child node of $\mathbf{x}_{pos}$ to $\mathbf{x}_i$; $T_{Dissim}(\mathbf{x}_{pos})$ the lowest similarity value, which can be observed among the children of $\mathbf{x}_{pos}$; $subtree_{\mathbf{x}_i}$ the subtree contains $\mathbf{x}_i$ and its child nodes. $\mathbf{x}_i$ is connected to $\mathbf{x}_{pos}$ if and only if the connection of $\mathbf{x}_i$ decreases further this value. Since this minimum value can only be computed with at least two nodes, then the first two objects are automatically connected to the tree structure as two first tree nodes without any test (Rule 1 in Algorithm 7). This may result in "abusive" connections for the second node. Therefore the second node is removed and disconnected as soon as the third node is connected (Rule 2). For this latter node, we are certain that the dissimilarity test has been successful. We note that for each $\mathbf{x}_{pos}$, we allow to disconnect only once to assume the convergence of the algorithm. If we have already disconnected data from $\mathbf{x}_{pos}$, the Rule 3 is employed.

Algorithm 7 AntTree algorithm

```
1: **Rule 1**: less than 2 data connected to $\mathbf{x}_{pos}$
2:     connect $\mathbf{x}_i$ to $\mathbf{x}_{pos}$
3: **End rule**
4: **Rule 2**: more than 2 data connected to $\mathbf{x}_{pos}$ and for the first time
5:     $T_{Dissim}(\mathbf{x}_{pos}) = max(d(\mathbf{x}_i, \mathbf{x}_j))$    // where $\mathbf{x}_i$ and $\mathbf{x}_j$ are any pair of data connected to $\mathbf{x}_{pos}$; $d(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||^2$, $\mathbf{x}_i, \mathbf{x}_j$ are normalized
6:     **if** $sim(\mathbf{x}_i, \mathbf{x}^+) < T_{Dissim}(\mathbf{x}_{pos})$ **then**
7:         disconnect $\mathbf{x}^+$ from $\mathbf{x}_{pos}$    // disconnect recursively all childs of $\mathbf{x}^+$
8:         connect $\mathbf{x}_i$ to $\mathbf{x}_{pos}$
9:     **else**
10:        move $\mathbf{x}_i$ to $\mathbf{x}^-$
11:    **end if**
12: **End rule**
13: **Rule 3**: more than 2 data connected to $\mathbf{x}_{pos}$ and for the sencond time
14:    **if** $sim(\mathbf{x}_i, \mathbf{x}^+) < T_{Dissim}(\mathbf{x}_{pos})$ **then**
15:        connect $\mathbf{x}_i$ to $\mathbf{x}_{pos}$
16:    **else**
17:        move $\mathbf{x}_i$ to $\mathbf{x}^+$
18:    **end if**
19: **End rule**
```

In classical clustering methods such as dendrogram or HCA, tree nodes are often fixed after being get assigned; however the AntTree rules allow to disconnect objects recursively at a moment so it's able to correct mis-classification of previous steps.

2.5 Complexity

The clustering algorithms are proposed to tackle the same goal, but their computational time [Goldreich 2008] may be various depending on their principles. Usually the efficiency or running time of an algorithm is related to the length of learning process or number of steps (time complexity) and storage locations (space complexity). Complexity is often denoted by O or Θ. In Table 2.2 we study the time complexity of the main algorithms presented previously [Martinetz 1993, Xu 2005, Hastie 2009].

Clustering algorithm	Complexity
K-meams	$\Theta(KN)$
SOM	$\Theta(KNlogN)$
Neural Gas	$\Theta(KNlogN)$
AHC	$\Theta(N^3)$
AntTree	$\Theta(NlogN)$
MST	$\Theta(N^2logN)$

Table 2.2: Computional complexity of clustering algorithms

2.6 Clustering quality

Now the next question is how to define a good clustering? or how to measure clustering quality? There are many suggestions for a quality measure [Rand 1971, Everitt 2011]. Such a measure can be used to compute the quality of a clustering. Several methods take into account high similarity within a cluster and low similarity among clusters to compare the clustering quality given by different clustering algorithms. We will introduce some well-known criteria to measure the clustering quality.

2.6.1 Internal validation

The internal validation measures [Estivill-Castro 2002] base on the data that was clustered. These methods usually find a high score to the algorithm that produces clusters with high similarity within a cluster and low similarity between clusters. One drawback of using internal criteria in cluster validation is that high scores on an internal measure do not necessarily result in effective information retrieval applications. And these index works well on distance-based methods for example K-means and the results may be local.

2.6.1.1 Davies-Bouldin index

Davies-Bouldin (DB) index [Davies 1979] is an internal measure between two clusters. A good algorithm output clusters with high intra-cluster similarity and low inter-cluster similarity will have a low DB index. This index is defined as:

$$DB = \frac{1}{N} \sum_{i=1}^{K} \max_{j, i \neq j} \frac{\mu_i + \mu_j}{d(\mathbf{w}_i, \mathbf{w}_j)} \tag{2.30}$$

where μ_i is the average distance of all elements in cluster C_i to its respective centroid, i.e prototype $\mathbf{w}_i$.

2.6.1.2 Dunn index

The Dunn index [Bezdek 1995] measures dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. The Dunn index can be calculated by the following formula:

$$Dunn = \min_{i=1,..,K}\left(\min_{j=1,..,K;i\neq j}\left(\frac{d(\mathbf{x}_i,\mathbf{x}_j)}{\max_{l=1,..,K}\Delta(l)}\right)\right) \tag{2.31}$$

where $\Delta(l)$ is the intra-cluster distance which may be measured in a variety ways, such as the maximal distance between any pair of objects in cluster. Algorithms that ouput clusters with high Dunn index are more desirable.

2.6.2 External validation

It's hard to evaluate a clustering with little data information given a priori. In external validation [Färber 2010], clustering results are evaluated based on data whose class labels are available as external benchmarks. Such benchmarks consist of a set of pre-classified items, N objects are associated with a set of L classes. Clustering methods group these objects into K clusters, thus two partitions to compare are defined: $\mathcal{X} = \{\mathbf{x}_1, .., \mathbf{x}_N\}$, where $\phi(\mathbf{x}_i)$, the cluster contains $\mathbf{x}_i$, $\mathcal{C} = \{C_1, .., C_K\}$ is a random variable for cluster assignments, and $Y = \{y_1, .., y_N\}$, where $y_l \in \mathcal{B} = \{B_1, .., B_L\}$ is a variable for labels. The contingency table can be expressed as in Table 2.3.

$\mathcal{B}\backslash\mathcal{C}$	C_1	C_2	$\cdots$	C_k	$\cdots$	C_K	Sum
B_1	n_{11}	n_{12}	$\cdots$	n_{1k}	$\cdots$	n_{1K}	n_{B_1}
B_2	n_{21}	n_{22}	$\cdots$	n_{2k}	$\cdots$	n_{2K}	n_{B_2}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
B_l	n_{l1}	n_{l2}	$\cdots$	n_{lk}	$\cdots$	n_{lK}	n_{B_l}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
B_L	n_{L1}	n_{L2}	$\cdots$	n_{Lk}	$\cdots$	n_{LK}	n_{B_L}
Sum	n_{C_1}	n_{C_2}		n_{C_k}		n_{C_K}	N

Table 2.3: Contingency table

2.6.2.1 Accuracy

A simple approximation of accuracy for unsupervised learning often uses external class information. Between the class labels and the clusters in a given dataset, accuracy criterion reflects the proportion of objects that were correctly assigned. A high value for this measure generally indicates a good clustering. The formula to compute accuracy is below:

$$Acc = \frac{1}{N}\sum_{k=1}^{K} \max_{l=1,..,L}(n_{lk}) \tag{2.32}$$

2.6.2.2 Normalized Mutual Information

Normalized Mutual Information (NMI) [Strehl 2003] measures how much information is shared between a clustering and a ground-truth classification that can detect a non-linear similarity between two clusterings. A good clustering method will produce high quality clusters or the intra-cluster variance is high and the inter-cluster variance is low. So the problem is now to maximize NMI which is defined by the following formula:

$$NMI = \frac{\sum_{l=1}^{L}\sum_{k=1}^{K} n_{lk} log_2(\frac{N n_{lk}}{n_{B_l} n_{C_k}})}{(\sum_{l=1}^{L} n_{B_l} log_2(\frac{n_{B_l}}{N}))(\sum_{k=1}^{K} n_{C_k} log_2(\frac{n_{C_k}}{N}))} \tag{2.33}$$

2.6.2.3 Rand index

The Rand index [Rand 1971] computes how similar the obtained clusters are to the benchmark classifications. Rand value varies between 0 and 1, where 1 means that the two partitions are identical, and 0 indicates that the partitions have no common objects.

$$Rand = \frac{N_{00} + N_{11}}{\frac{N}{2}} \tag{2.34}$$

where N_{11} is the number of data pairs in the same cluster in both $\mathcal{B}$ and $\mathcal{C}$ and N_{00} is the number of data pairs in different clusters in both $\mathcal{B}$ and $\mathcal{C}$.

2.6.2.4 Jaccard index

The Jaccard index [Tan 2005] is similar to the Rand index. The Jaccard index is used to quantify the similarity between two partitions.

$$Jaccard = \frac{N_{11}}{N_{00} + N_{10} + N_{01}} \tag{2.35}$$

where N_{11} is the number of pairs in the same cluster in both $\mathcal{B}$ and $\mathcal{C}$; N_{10} is the number of data pairs in the same cluster in $\mathcal{B}$ but not $\mathcal{C}$; N_{01} is the number of data pairs in the same cluster in $\mathcal{C}$ but not $\mathcal{B}$; and N_{00} is the number of data pairs in different clusters in both $\mathcal{B}$ and $\mathcal{C}$.

2.6.2.5 Quality measures for graph clustering

For graph datasets we present several quality criteria [Leskovec 2010] as below. We consider a graph $G(V, E)$ divided into K clusters $\mathcal{C} = \{C_1, .., C_K\}$. The number of nodes in C_k is denoted n_{C_k}; m_{C_k} the number of internal edges in C_k, $m_{C_k} = |\{\{u, v\}; u, v \in C_k\}|$; and b_{C_k} the number of edges on the boundary of C_k, $b_{C_k} = |\{\{u, v\}; u \in C_k, v \notin C_k\}|$. Criteria used are defined as

- **Conductance** [Kannan 2004] measures the fraction of total edge volume that points outside the cluster C_k. A desired graph clustering is in the sense that there are more internal edges and fewer edges connected with inter-clusters. It will return a low value of conductance.

$$Conductance = \frac{b_{C_k}}{2m_{C_k} + b_{C_k}} \tag{2.36}$$

- **Expansion** [Radicchi 2004] measures the number of edges per node that point outside the cluster C_k.

$$Expansion = \frac{b_{C_k}}{n_{C_k}} \tag{2.37}$$

- **Cut Ratio** [Fortunato 2010] measures the fraction of all possible edges leaving the cluster. Algorithms what produce a few number of edges on the boundary of cluster are desired, thus they minimize this cut ratio.

$$Ratio = \frac{b_{C_k}}{n_{C_k}(N - n_{C_k})} \tag{2.38}$$

- **Internal density** [Radicchi 2004] is the internal edge density within the cluster C_k. High internal density indicates a good partition.

$$Density = \frac{m_{C_k}}{n_{C_k}(n_{C_k} - 1)/2} \tag{2.39}$$

2.7 Conclusion

As an important technique for data mining, clustering often consists in forming a set of groups according to a similarity measure. This process includes a different number of steps, ranging from data pre-processing, selecting similarity measure and algorithm development, to solution validity and evaluation. Each of them is tightly related to each other and becomes great challenges to the scientific disciplines.

This chapter presented an introduction to the clustering algorithms as an important technique for data mining. Clustering consists in either constructing a hierarchical structure, or forming a set of groups. We have reviewed a wide variety of approaches appearing in the literature. These algorithms evolve from different research in numerous fields. Clustering algorithms can offer approximate solutions or favor some type of biases. Although some comparisons are possible, there is no clustering algorithm that solves all problems for all the data types. Though many real-world applications are successfully resolved by clustering, but there remain many open problems due to the growth of data in many ways, e.g.: structured data, fuzzy data, sequence data, etc.

In the next chapters we will present our work on the extensions of the SOM model. We have selected SOM because it allows data visualization that our studies concern. Another aspect of the clustering problem is the visualization which is not possible for the most of clustering techniques. This is why we focus on the SOM algorithm which allows to tackle both the data clustering and the data visualization problems.

Chapter 3

Self-Organizing Trees

Contents

3.1 Introduction

As mentioned in the previous chapters, our study is always placed in the clustering problem. We use Kohonen's Self-Organizing Maps and AntTree as the basic models for our model SoT (Self-Organizing Trees), which allows to view the clustering from different perspectives leading to reliable clusters. SoT makes one further step in proposing a network revealing the topology of clusters and multiple trees revealing the hierarchy of input data.

In literature, SOM is a powerful technique for data visualization due to the ability of projecting high-dimensional data onto a 1D or 2D network [Peura 1999, Vesanto 1999, Kohonen 2001, Doherty 2005]. This method was employed in many numerous practical problems in order to reduce the dimensionality then visualize data. Another useful property of these models is topology preservation, i.e. the preservation of neighborhood relations among data. A mapping also preserves neighborhood relations if nearby objects in the input space remain close in the map space.

Data gathering in clusters are often similar in term of distance. In the meantime, AntTree introduces the rules of connecting objects to hierarchical structure and/or disconnecting objects from it. It can be naturally integrated into the self-organizing models like SOM so that we are able to correct the misclassification of previous steps. By combining both SOM and AntTree, our purpose is to decompose a given dataset into multi-hierarchical trees in order to preserve topological structure within network nodes and find a clustering. Practically, SoT creates a set of hierarchical links that cover all similar objects. A hierarchical link represents an one-to-one relation between a pair of objects. Ideally, these links could be considered as shortest paths.

This chapter will provide the details of SoT including the principles, the algorithms, the computional complexity, etc. In order to evaluate SoT, a numerous series of applications are conducted on real-world datasets. Furthermore, some intensive applications on structured data such as graph or protein are carefully studied. Graph as structured data is popularly used to model structural relationships among objects in many application domains such as web, social networks and interaction networks, etc. In this study, we seek to show the use of the hierarchical and topological structure in data analysis, especially in retrieving information from structured data. Moreover, the shortest paths given by SoT can be studied thoroughly. During the learning process, SoT is able to create a link between two graph nodes if they pass a similarity test. This kind of links can be either *original* (existing in the original graph) or *synthetized* (not existing in the original graph). Several interesting issues are as following:

- Could the synthetized links be helpful to analyze data?

- Are the original links all good? or are these links the most important in the graph?

To answer these questions, few experiments are conducted and divided into three parts:

1. The first application on vectorial datasets compares the SoT effeciency with the other clustering methods in term of clustering quality measures. Not only the numerical results are provided but also the visual ones are available.

2. The second application is conducted on graph datasets which are used as benchmarks to test algorithms. The purpose of this application is to summarize a graph into a smaller one by proposing new graph decomposition. The summarization provided by the proposed hierarchical and topological structure will simplify the original graph. It becomes particularly attractive for graph clustering, especially for graph visualization. Concerning graph datasets, the problem of graph clustering and summarization is well studied and the literature on the

subject is very rich [Kannan 2000, Everitt 2009, LeFevre 2010, Zhang 2010]. Indeed, most of the research in graph summarization suggests reducing the number of nodes by grouping similar node in the same cluster.

3. The third application is conducted on the protein dataset ASTRAL. The motivation is to study intensively the hierarchical and topological structure provided by SoT. Since an expert classification of this dataset is available in form of hierarchy, so we seek to compare it with the hierarchical clustering classification.

3.2 Principles

3.2.1 Clustering discovery with hierarchical relations

The idea is to add a new hierarchical dimension to a SOM map (or network) (see Figure 3.1). It means to generate a set of trees arranged according to certain topology. The network preserves the data topology while data similarity lies on the hierarchical structure. The network nodes represent the prototypes in the grid. The tree nodes in SoT structure corresponds to the input data. The proposed algorithm is able to detect clusters and to represent these clusters as topological and hierarchical structure. Here each tree is not only a cluster but a sub-tree from any branch is also a sub-cluster. Confidence of each cluster may be easily observed because of hierarchical relations among the input data.

In this hierarchical and topological structure, similarity between a pair of data objects is available through hierarchical links, while distances between two clusters depend on the topological links. Additionally, the hierarchical links can be seen as the shortest paths connecting objects together according their similarity.

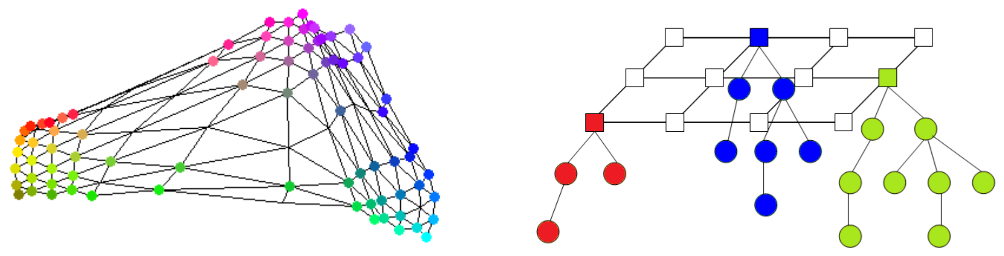

(a) SOM (b) SoT

Figure 3.1: SOM vs SoT

3.2.2 SoT - Topological and hierarchical clustering

For us in this thesis, the notations: an input data object or vector $\mathbf{x}_i$ and a tree node i are all identical. A cluster k, a network node C_k, the prototype $\mathbf{w}_k$ are also identical.

In the SoT model, the size of the grid denoted by $K = p \times q$ must be provided a priori. Each network node $C_k \in \mathcal{C}$ associated with a prototype $\mathbf{w}_k \in \mathcal{W}$ as well as a $tree_k$. So by following this initialization, we should have K *trees* in the network. For each pair of $tree_c$ and $tree_r$, their mutual influence is defined by the function $\mathcal{K}^T(\delta(C_c, C_r)) = \exp(\frac{-\delta(C_c,C_r)}{T})$. Based on Equation 2.28, the new cost function is re-written to adapt to the hierarchical structure as in Equation 3.1.

$$\mathcal{R}(\phi, \mathcal{W}) = \sum_{c=1}^{K} \sum_{i=1}^{n_{C_k}} \sum_{r=1}^{K} \mathcal{K}^T(\delta(\phi(subtree_{\mathbf{x}_i}), C_r)) \|\mathbf{x}_i - \mathbf{w}_r\|^2 \tag{3.1}$$

where ϕ is the assignment function which is expressed as following:

$$\phi(subtree_{\mathbf{x}_i}) = \phi(\mathbf{x}_i) = \underset{r=1,..,K}{\arg\min} \sum_{c=1}^{K} \mathcal{K}^T\left(d(C_c, C_r)\right) \|\mathbf{x}_i - \mathbf{w}_c\|^2 \tag{3.2}$$

where $subtree_{\mathbf{x}_i}$ contains $\mathbf{x}_i$ and all tree nodes recursively connected to it. It should be noted that data in the sub-tree are similar due to the self-assembly rules. We take this as an advantage to reduce algorithm complexity. Lemma 3.2.1 shows how to assign simultaneously a data group.

Lemma 3.2.1

$$\forall \mathbf{x}_j \in subtree_{\mathbf{x}_i}, \phi(\mathbf{x}_j) = \phi(\mathbf{x}_i) \tag{3.3}$$

Proof Let $d(\mathbf{x}_i, C_c)$ be the distance from $\mathbf{x}_i$ to $\mathbf{w}_c$ and suppose that $C_c = \phi(\mathbf{x}_i)$ then we have:

$$d(\mathbf{x}_i, C_c) < d(\mathbf{x}_i, C_k), \forall k = 1, .., K$$

When $\mathbf{x}_j \in subtree_{\mathbf{x}_i} \implies \mathbf{x}_i$ and $\mathbf{x}_i$ are similar, their distance should tend to 0.

$$\lim_{\mathbf{x}_j \to \mathbf{x}_i} d(\mathbf{x}_i, \mathbf{x}_j) = 0$$

We deduce that:

$$d(\mathbf{x}_j, C_c) \approx d(\mathbf{x}_i, C_c) + d(\mathbf{x}_j, \mathbf{x}_i) < d(\mathbf{x}_i, C_k) + d(\mathbf{x}_j, \mathbf{x}_i) \approx d(\mathbf{x}_j, C_k)$$

$$\implies \phi(\mathbf{x}_j) = C_c \text{ or } \phi(\mathbf{x}_j) = \phi(\mathbf{x}_i) = C_c$$

Minimizing cost function $\mathcal{R}(\phi, \mathcal{W})$ is a combinatorial optimization problem. In this work we propose to minimize the cost function in the same way as batch version using statistical characteristics provided by trees to accelerate the convergence of the algorithm. These characteristics are used in the assignment function 3.2.

3.2.3 Batch algorithm

Here we want to show how to adapt the self-assembly rules provided by AntTree in Algorithm 7 to topological models including SOM. During the learning process, the status of a tree node can be various due to the connecting or disconnecting rules. Therefore, we define three possibilities for the tree node status:

1. *Initial* : the default status before training;
2. *Connected* : the tree node is currently connected to another node;
3. *Disconnected* : the tree node were connected at least once but now gets disconnected.

Let us denote $\mathbf{x}_i^{pos}$ is the support (the tree root) or the tree node position where $\mathbf{x}_i$ is located. At the beginning, $\mathbf{x}_i$ is located on the support and will move in the tree (toward other tree nodes) in order to find its best position; $\mathbf{x}^+$ and $\mathbf{x}^-$ two tree nodes connected to $\mathbf{x}_i^{pos}$ which are respectively the most similar and dissimilar tree node to $\mathbf{x}_i$. Let $\mathcal{L}$ denote a list of tree nodes. Before training, $\mathcal{L}$ contains all the *initial* tree nodes, whenever a tree node becomes *connected*, it is immediately removed from $\mathcal{L}$; alternatively whenever a tree node and its children get disconnected from a tree, we put them back onto $\mathcal{L}$. After several iterations, $\mathcal{L}$ might contain both the *initial* and the *disconnected* tree nodes.

The batch algorithm is shown in Algorithm 8. The algorithm includes three steps as in the SOM algorithm (Section 2.4.1.2). Besides, SoT has an additional step for tree construction. While the initialization stays the same as the one in SOM, the others must be modified to adapt to the new structure. Then SoT can proceed by alternating between three steps: assignment, tree construction and prototype adaptation.

Assignment

This step (Line 6 in Algorithm 8) is to assign data objects to the nearest prototype or the best match tree, such in Equation 3.2. This assures that the cost function $\mathcal{R}(\phi, \mathcal{W})$ in Equation 3.1 is minimized in regard to the assignment function ϕ assuming that the prototype vectors are constant. An *initial* object $\mathbf{x}_i$ is assigned, it will be connected to the best match tree in the tree construction step. However, if $\mathbf{x}_i$ has currently one of the other status: *connected* or *disconnect*, we have to check whether it does exist other child nodes in $subtree_{\mathbf{x}_i}$? If in the case, following Lemma 3.2.1, the child nodes will follow the assigment of $\mathbf{x}_i$.

An example where a sub-tree assignment happens is shown in Figure 3.2. Due to the last update, $subtree_{\mathbf{x}}$ consists of three violet nodes that are no longer connected to $tree_{c_{old}}$. Now we have to determine the new best match $tree_c$ for the node $\mathbf{x}$ using $\phi(subtree_{\mathbf{x}})$. The assignments of child nodes of $\mathbf{x}$ follow automatically the one of $\mathbf{x}$

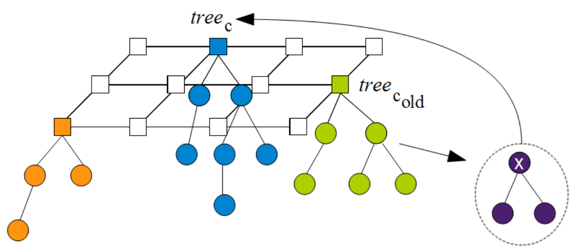


Figure 3.2: Sub-tree assignment from $tree_{c_{old}}$ to $tree_c$

using the statistical characteristics of tree, each object is connected to its nearest neighbor (1-NN). Even though these three nodes are now found in the new cell c, the hierarchy in the new cell remains that of the old cell c_{old}.

Tree construction

This step is realized by the procedures *constructTree* and *constructTreeSub* (respectively, Algorithm 10 and 9) in which we use the rules from Algorithm 7. Here we must differ these two procedures:

1. *constructTree* is called for one single object (see Line 9 in Algorithm 8). This procedure is used to connect to a tree or probably disconnect an object from the existing tree.

2. *constructTreeSub* is called by a sub-tree (see Line 21 in Algorithm 8). This procedure will call directly *constructTree* for only the sub-tree root (see Line 1 Algorithm 9). It should be noted that data in the sub-tree are similar due to the self-assembly rules. The child nodes will automatically follow the sub-tree root if it is succesfully *connected*. In this case, we try keeping the sub-tree structure as in Line 3 Algorithm 9. By using statistical characteristics of hierarchy, we take this as an advantage to reduce algorithm complexity.

During the learing process, there is a chance that objects can be *disconnected*. Concerning to the disconnection, there are two distinct cases:

1. disconnect tree node(s) due to the assignment (Line 18 in Algorithm 8),

2. disconnect tree node when a tree node comes into play (Line 7 in Algorithm 10).

Whenever a tree node gets disconnected from a tree, we have to check whether it does exist other child nodes in $subtree_{\mathbf{x}_i}$ or not? If in the case, we disconnect all of

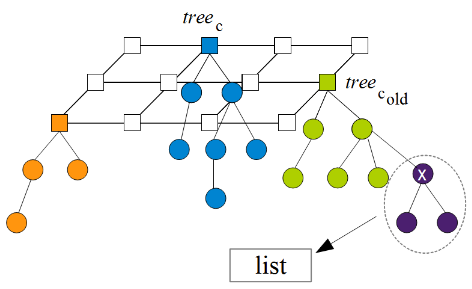


Figure 3.3: Disconnect $subtree_{\mathbf{x}}$ from $tree_{c_{old}}$ and put it in *list*

them or for more specific $subtree_{\mathbf{x}_i}$. A simple example of disconnection for a group of nodes (or sub-tree) is depicted in Figure 3.3. Given $tree_{c_{old}}$ as in this example, the tree node $\mathbf{x}$ consisting of three violet nodes is to disconnect from this tree. All the nodes connected to $\mathbf{x}$ must be recursively disconnected too; it applies to two child nodes of $\mathbf{x}$. Therefore $subtree_{\mathbf{x}}$ has *disconnected* status and is immediately put back onto the list L.

Suppose that $\mathbf{x}_i$ becomes *connected* at a moment, we will keep this sub-tree structure by re-connecting these child nodes together; hence this way can accelerate the learning process. For example, let's re-take the example in Figure 3.3. After getting new assignment, $\mathbf{x}$ is going to connect to $tree_c$. It leads to that the child nodes of $\mathbf{x}$ have $tree_c$ as their best match tree too. We systematically connect this sub-tree to $tree_c$ and the result is shown in Figure 3.4. We remind that this subtree is not kept till the end of learning, there is a possiblity that the nodes in the subtree will be disconnected in next iterations.

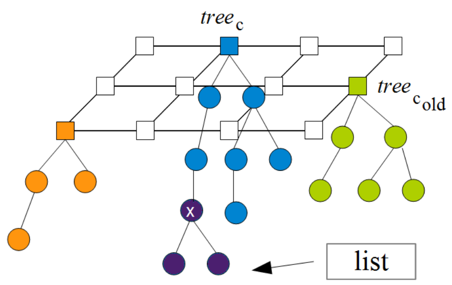


Figure 3.4: Re-connect $subtree_{\mathbf{x}}$ to $tree_c$

Prototype adaptation

This step is necessary to compute the new prototypes. We can easily adapt our algorithm to choose a representative prototype. The prototypes and their neighbors move along together towards the assigned data such that the map tends to approximate the data distribution. It includes minimizing the cost function $\mathcal{R}(\phi, \mathcal{W})$ in Equation 3.1 in regard to the prototypes vectors assuming that data are all assigned to the best match unit. The choice of prototype can depend on data structure, it can be seen as centroid in the case of traditional continuous datasets or as leaders [Stanoev 2011] (see Equation 2.8) in the case of graph datasets.

Algorithm 8 SoT batch algorithm

```
1: initialize $k$ prototypes
2: while stopping criteria have not been fulfilled do
3:    initialize $\mathcal{L}$
4:    while $\mathcal{L}$ is not empty do
5:       $\mathbf{x}_i$ is the first data from $\mathcal{L}$
6:       $c = \phi(\mathbf{x}_i)$    // find the best match tree
7:       if $\mathbf{x}_i$ is initial then
8:          $\mathbf{x}_i^{pos} = 0$    // set the tree root as the position of $\mathbf{x}_i$
9:          $constructTree(tree_c, \mathbf{x}_i, \mathbf{x}_i^{pos})$    // connect $\mathbf{x}_i$ to $tree_c$
10:      else
11:         if $\mathbf{x}_i^{pos}$ is in $tree_c$ then
12:            $\mathbf{x}_i^{pos}$ gets the current position of $\mathbf{x}_i$
13:         else
14:            $\mathbf{x}_i^{pos} = 0$
15:         end if
16:         $subtree_{\mathbf{x}_i} = \{\mathbf{x}_i$ and all tree nodes recursively and temporarily connected
            to $\mathbf{x}_i\}$
17:         if $\mathbf{x}_i$ is connected and $c \neq c_{old}$ then
18:            $subtree_{\mathbf{x}_i}$ = disconnect $subtree_{\mathbf{x}_i}$ from $tree_{c_{old}}$
19:         end if
20:         if $\mathbf{x}_i$ is disconnected then
21:            $constructTreeSub(tree_c, subtree_{\mathbf{x}_i}, \mathbf{x}_i^{pos})$    // connect either $\mathbf{x}_i$ or
               $subtree_{\mathbf{x}_i}$ to $tree_c$
22:         end if
23:      end if
24:      if $\mathbf{x}_i$ is not connected then
25:         put $\mathbf{x}_i$ at the end of $\mathcal{L}$
26:      else
27:         remove $subtree_{\mathbf{x}_i}$ from $\mathcal{L}$
28:      end if
29:   end while
30:   update the prototypes
31: end while
```

Algorithm 9 constructTreeSub

```
Input: $tree_c, subtree_{\mathbf{x}_i}, \mathbf{x}_i^{pos}$
Output: $tree_c, \mathbf{x}_i\mathbf{x}_i^{pos}$
1: $tree_c = constructTree(tree_c, \mathbf{x}_i, \mathbf{x}_i^{pos})$   // connect the sub-tree root to the tree
2: if $\mathbf{x}_i$ is connected then
3:    connect recursively all nodes in $subtree_{\mathbf{x}_i}$ to $tree_c$
4: end if
```

Algorithm 10 constructTree

```
Input: $tree_c, \mathbf{x}_i, \mathbf{x}_i^{pos}$
1: if less than 2 data connected to $\mathbf{x}_i^{pos}$ (Rule 1) then
2:    connect $\mathbf{x_i}$ to $\mathbf{x}_i^{pos}$
3: else
4:    $T_{Dissim}(\mathbf{x}_i^{pos}) = max(d(\mathbf{x_i}, \mathbf{x}_j))$   // where $\mathbf{x_i}$ and $\mathbf{x}_j$ are any pair of data
      connected to $\mathbf{x}_i^{pos}$; $d(\mathbf{x}, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||^2$, $\mathbf{x}, \mathbf{x}_j$ are normalized
5:    if more than 2 data connected to $\mathbf{x}_i^{pos}$ and for the first time (Rule 2) then
6:       if $sim(\mathbf{x_i}, \mathbf{x}^+) < T_{Dissim}(\mathbf{x}_i^{pos})$ then
7:          disconnect $\mathbf{x}^+$ from $\mathbf{x}_i^{pos}$   // disconnect recursively all childs of $\mathbf{x}^+$
8:          connect $\mathbf{x_i}$ to $\mathbf{x}_i^{pos}$
9:       else
10:         $\mathbf{x}_i^{pos} = \mathbf{x}^-$   // set $\mathbf{x}^-$ as the new position of $\mathbf{x}$
11:      end if
12:   else
13:      // Rule 3
14:      if $sim(\mathbf{x}_i, \mathbf{x}^+) < T_{Dissim}(\mathbf{x}_i^{pos})$ then
15:         connect $\mathbf{x}$ to $\mathbf{x}_i^{pos}$
16:      else
17:         $\mathbf{x}_i^{pos} = \mathbf{x}^+$   // set $\mathbf{x}^+$ as the new position of $\mathbf{x}_i$
18:      end if
19:   end if
20: end if
```

3.2.4 Computational complexity

Lemma 3.2.2 *SoT has a complexity of $\Theta(N_{iter} N \log N)$, where N_{iter} is the number of total iterations when the learning has been terminated.*

Proof Algorithm 8 requires N_{iter} iterations to reach all the stopping criteria. Covergence or quantization error can be used as a stopping criterion. During one iteration, an operation includes three iterative steps: assignment, tree construction and updating support. The number of assignments starts from 100% and decreases taking the advantage from the tree structure (the subtree nodes follow the node root assignment), which allows the run time to be reduced. To terminate training for N objects and to exit from the second loop (Line 4 in Algorithm 8), it requires a complexity of $N \log N$ operations. This is exactly the AntTree complexity [Azzag 2007]. Finally, SoT has a complexity of $\Theta(N_{iter} N \log N)$ which is competitive with traditional topological map SOM.

3.3 Experiments on real-world datasets

The experimental section is divided into three parts:

1. Application on several classical datasets from the UCI Repository (http://archive.ics.uci.edu/ml/) as well as few image datasets. This helps to validate SoT along with other clustering methods such as SOM and MST. We would like to show that SoT can achieve the performance that is comparable to the one given by the other methods.

2. Application on graph datasets. The hierarchical and topological relations in the SoT structure contain insight information that is interesting to analyze. Especially for graph datasets, this structure can be implicitly used for graph summarization.

3. Application on the biological dataset ASTRAL. We applied SoT on bioinformatics domain to show how SoT structure provides much more information than the expert classification.

The experimental part includes two phases: numerical and visual validations.

3.3.1 First application on vectorial datasets

3.3.1.1 Dataset description

We use traditional vectorial datasets that come from the UCI machine learning repository and additionally, two image datasets: COIL-20 [Nene 1996] consists of the

Dataset	Size	# Features	# Classes
Arrhythmia	420	278	2
Cancer	699	9	2
Ecoli	336	7	8
Glass	214	9	6
Ionosphere	351	17	2
Iris	150	4	3
Pima	768	8	2
Sonar	208	60	2
Thyroid	215	5	3
Wine	178	13	3
COIL-20	1440	1024	20
Yale	165	1024	15

Table 3.1: Vectorial dataset description

set which contains images of 20 different objects with 72 images per object; Yale [Cai 2007] contains 165 grayscale images of 15 individuals. Table 3.1 describes the characteristics of the selected datasets. These datasets are often used as benchmarks for clustering algorithm validations.

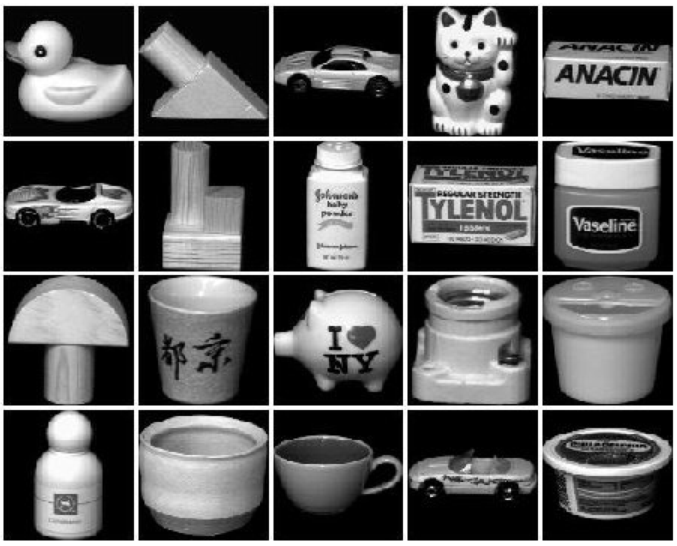


Figure 3.5: Sample images of 20 objects in COIL-20 dataset

3.3.1.2 Experimental protocol

We define the input parameters and experimental protocol as following:

1. Each dataset must be normalized before training as in Equation 2.1. For each dataset, 10 runs are conducted using random initializations.

2. The shape of map's topology is rectangular. A network node must have at least one data object at the end of the learning process.

3. The size of map is set 3×3 for real datasets from UCI in Table 3.1, except for COIL-20 5×5 and Yale 4×4. The motivation is to choose the map size superior to the number of classes for each dataset so that a map is large enough to cover data space.

4. We select SOM and not the hierarchical SOM variants presented in Section 2.4.3 due to the fact that these variants don't have the same structure and initialization parameters. Moreover, they don't improve the assignment rules so their performance should be the same as SOM's. We have also studied the experimental results of MST because both SoT and MST are based on a similar tree-like structure, preseve the data topology using the paths connecting among data objects.

3.3.1.3 Numerical validation

In order to measure the clustering quality, the three criteria: Accuracy (Equation 2.32), NMI (Equation 2.33) and Rand (Equation 2.34) are employed. Table 3.2 shows the values of these quality measures averaged over 10 runs for the selected datasets. We observe that SoT provides similar results and quite comparable to SOM in most of cases. Looking to the columns associated to SoT comparing to MST, we observe that the SoT performance is high for the majority of datasets. Our purpose through this comparison, is not to assert that our method is the best, but to show that SoT can obtain the same good results as other clustering algorithms. We present below a particular study of COIL-20 dataset to highlight the hierarchical and topological properties of SoT.

Dataset	Map size	Method	Acc	NMI	Rand
Arrhythmia	2×2	SOM	0.581	0.013	0.504
		MST	0.571	0.0321	0.509
		SoT	0.601	0.033	0.513
	3×3	SOM	0.619	0.037	0.528
		MST	0.580	0.058	0.511
		SoT	0.623	0.051	0.530

Cancer	2×2	SOM	0.972	0.774	0.938
		MST	0.656	0.018	0.546
		SoT	0.940	0.548	0.764
	3×3	SOM	0.965	0.782	0.933
		MST	0.656	0.018	0.548
		SoT	0.952	0.710	0.908
Ecoli	2×2	SOM	0.723	0.556	0.824
		MST	0.476	0.201	0.372
		SoT	0.673	0.453	0.740
	3×3	SOM	0.750	0.630	0.856
		MST	0.580	0.457	0.585
		SoT	0.740	0.563	0.824
Glass	2×2	SOM	0.478	0.366	0.624
		MST	0.467	0.315	0.54
		SoT	0.507	0.262	0.638
	3×3	SOM	0.574	0.354	0.648
		MST	0.443	0.156	0.389
		SoT	0.590	0.320	0.675
Ionosphere	2×2	SOM	0.891	0.038	0.460
		MST	0.894	0.033	0.605
		SoT	0.894	0.035	0.461
	3×3	SOM	0.891	0.005	0.806
		MST	0.900	0.148	0.819
		SoT	0.892	0.027	0.807
Iris	2×2	SOM	0.796	0.668	0.803
		MST	0.753	0.653	0.777
		SoT	0.812	0.597	0.788
	3×3	SOM	0.900	0.756	0.888
		MST	0.906	0.614	0.820
		SoT	0.930	0.813	0.918
Pima	2×2	SOM	0.682	0.080	0.539
		MST	0.652	0.017	0.544
		SoT	0.682	0.059	0.518
	3×3	SOM	0.695	0.0706	0.577
		MST	0.654	0.032	0.547
		SoT	0.703	0.079	0.582
Sonar	2×2	SOM	0.584	0.022	0.505
		MST	0.533	0.091	0.497
		SoT	0.567	0.013	0.502

	3 × 3	SOM	0.600	0.028	0.518
		MST	0.557	0.066	0.504
		SoT	0.630	0.051	0.532
Thyroid	2 × 2	SOM	0.765	0.357	0.558
		MST	0.711	0.095	0.549
		SoT	0.754	0.241	0.563
	3 × 3	SOM	0.825	0.378	0.744
		MST	0.730	0.172	0.576
		SoT	0.839	0.448	0.755
Wine	2 × 2	SOM	0.854	0.662	0.814
		MST	0.634	0.533	0.697
		SoT	0.771	0.544	0.761
	3 × 3	SOM	0.733	0.416	0.744
		MST	0.634	0.465	0.667
		SoT	**0.733**	**0.416**	**0.739**
COIL-20	5 × 5	SOM	0.554	0.727	0.939
		MST	0.116	0.241	0.179
		SoT	**0.556**	**0.649**	**0.931**
	7 × 7	SOM	0.653	0.699	0.952
		MST	0.309	0.454	0.539
		SoT	0.722	0.710	0.955
Yale	4 × 4	SOM	0.378	0.464	0.865
		MST	0.157	0.238	0.217
		SoT	**0.400**	**0.414**	**0.853**
	5 × 5	SOM	0.451	0.504	0.910
		MST	0.230	0.428	0.455
		SoT	0.513	0.528	0.911

Table 3.2: Competitive performance on vectorial datasets. The quality criteria are averaged over 10 runs.

Case of COIL-20 datasets

Columbia Object Image Library dataset (COIL-20) [Nene 1996] contains 72 gray level images for each of a set of 20 various objects, taken at intervals of five degrees 3D-rotation. The COIL-20 visualizations are presented in Figure 3.6, 3.7 and 3.8. In our work, we use Tulip as framework [Auber 2003] to draw all the visualizations. Figure 3.6 presents the multi-level view where the network nodes in black color represent the root of trees; and the colored nodes correspond to the tree node or the input data. Data from the same class are depicted by the same color. Many regions

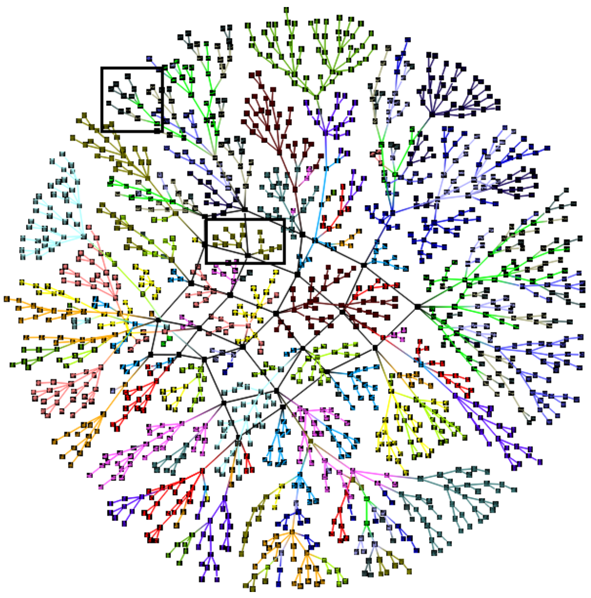


Figure 3.6: Hierarchical and topological structure given by SoT on COIL-20 dataset

with pure colors can be noticed from Figure 3.6, this results from a good clustering.

Due to the nature that a COIL-20 vector is an image, we generate two zoom samples extracted from Figure 3.6. In the first zoom (Figure 3.7) extracted from the top left of Figure 3.6, even though we have grouped images of different objects in the same sub-tree, the geometric shapes of objects, i.e.: cars, are quite similar. It would be interesting to note that in sub-trees, the car objects turn respectively in the same direction of their parent node. Figure 3.8 displays the second zoom extracted from the center. This is the result of a good classification, since the images of a "cup" object are found together. The benefit of SoT is to decompose and represent COIL-20 dataset in multi-level organization. Thus, hierarchical schemes provide a multi-level decomposition of the original data in a tree structure producing nested clusters at different levels either from top to bottom or vice versa.

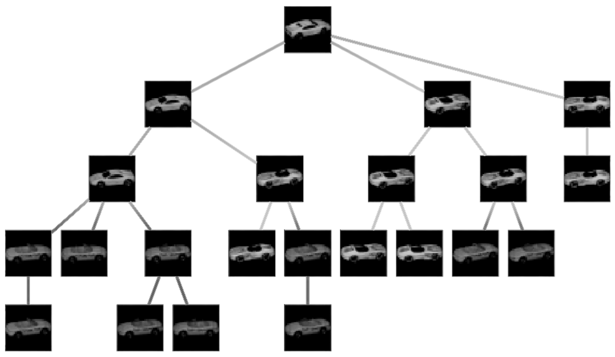


Figure 3.7: First zoom extracted from the top left of Figure 3.6

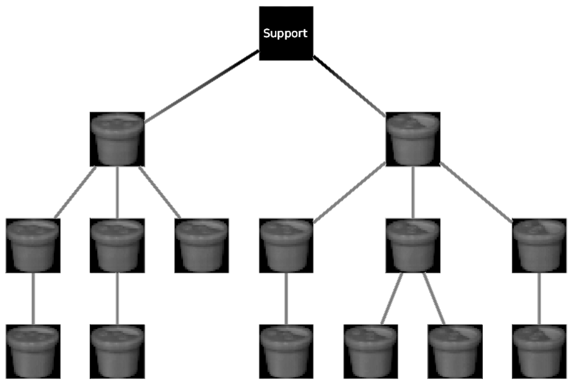


Figure 3.8: Second zoom extracted from the center of Figure 3.6

3.3.2 Second application on graph datasets for graph summerization

3.3.2.1 Graph decomposition and summarization

Graph clustering aims to partition the graph into several dense connected components. The majority of existing methods focus on the topological structure of a graph so that each partition achieves a cohesive internal structure. Such methods include clustering based on normalized cut [Shi 1997], modularity [Girvan 2002] or structural density [Xu 2007]. Thereby self-organizing map is commonly employed to solve problems of graph clustering in the sense that they work with topology preserving mapping ability [Macdonald 2000, Boulet 2008].

By applying SoT on graph datasets, our purpose is to summarize a given graph into a smaller one by proposing a new decomposition of original graph. Simultaneously, we provide a topological map and a topological trees using self-organizing maps. Thus, it is possible to summarize large graph into topological and tree-like organizations. Especially the hierarchical nature of the summarization data structure is particularly attractive.

Intuitively, the vertices in a community should have a high density of edges within the community than between other communities. As the reason, minimizing the number of edges running between clusters or finding many edges within each cluster as many as possible are the most often adopted approach for graph summarization.

3.3.2.2 Dataset description

Dataset	Size	# Features	# Classes
Adjective and Noun	112	425	2
Football Teams	115	616	10
Les Miserables	77	254	N/A
Political blogs	1490	19090	2

Table 3.3: Graph dataset description

The selected datasets for this experiment is presented in Table 3.3. They are available at `http://www-personal.umich.edu/~mejn/netdata/`. Graphs in these datasets are undirected and unweighted, which is enough to compute the Laplacian (Equation 2.10). Their description is as following:

- **Adjective and Noun**: An adjacency network of common adjectives and nouns in the novel David Copperfield by Charles Dickens introduced by [Newman 2006] where each node corresponds to a word whose type is either *adjective* or *noun*.

- **Football Teams**: The network of American football games between Division IA colleges during regular season Fall 2000 [Girvan 2002]. Vertices in the graph represent teams, while edges represent regular-season games between the two teams they connect.

- **Les Miserables**: An unweighted network of characters in the novel "Les Miserables" [Knuth 1993]. The characters become the vertices of a graph. Two vertices are adjacent if the corresponding characters encounter each other, in selected chapters of the book.

- **Political Blogs**: A directed network of hyperlinks between webblogs on US politics, recorded in 2005 by Adamic and Glance [Adamic 2005]. The vertices are the blogs retrieved from Internet. If a blog cited by other, an edge is created. There are only 2 classes, the number of citations (≥ 17 or < 17) in a blog decides the label of blog.

3.3.2.3 Experimental protocol

We define the input parameters and experimental protocol which is completely the same as the previous experiment:

1. For the graph datasets, by following the spectral clustering in Section 2.2.2 we select a number of first smallest eigenvectors $\lambda = \{5, \sqrt{N}\}$ of the regularized Laplacian. For each dataset, 10 tests are conducted using random initializations.

2. The shape of map's topology is rectangular. A network node must have at least one data object.

3. The size of map is respectively fixed $K = 3 \times 3$ for "Les Miserables" and "Adjective and noun"; $K = 5 \times 3$ for "Football Teams" 5×5 ($k = 25$) for "Political blogs".

4. For graph datasets, leaders [Stanoev 2011] are considered as prototypes. Instead of following Line 23 in Algorithm 8, the prototype expression is modified as following:

$$\mathbf{w}_c = \arg \max_{\forall i \in tree_c} (deg(v_i)) \tag{3.4}$$

   where the local degree $deg(v_i)$ is the number of internal edges incident to v_i. Choosing a representative prototype allows easily adapting our algorithm.

3.3.2.4 Numerical validation

In this section, we performed extensive experiments to evaluate the SoT performance on graph datasets. We compare the efficiency of SoT with different similar clustering algorithms. To measure the quality of SoT, Accuracy Equation 2.32, NMI Equation 2.33, presented in Section 2.6, are used. For graph datasets, two other extra measures: Conductance Equation 2.36 and Density Equation 2.39 [Leskovec 2010] are computed.

In Table 3.4, the performance values show that the quality measures for SOM and

Method	λ	Accuracy $\nearrow$	Conductance $\searrow$	Density $\nearrow$	NMI $\nearrow$
		Adjective and noun (3×3)			
SOM	5	0.574	0.743	0.174	0.073
	11	0.576	0.691	0.213	0.072
MST	5	0.553	0.903	0.007	0.290
	11	0.562	0.910	0.006	0.280
SoT	5	**0.565**	**0.783**	**0.207**	**0.115**
	11	**0.560**	**0.736**	**0.299**	**0.134**
		Football Teams (5×3)			
SOM	5	0.568	0.529	0.692	0.505
	11	0.406	0.645	0.735	0.544
MST	5	0.330	0.956	0.0643	0.299
	11	0.400	0.968	0.045	0.313
SoT	5	**0.880**	**0.564**	**0.714**	**0.687**
	11	**0.878**	**0.532**	**0.791**	**0.685**
		Les Miserables (3×3)			
SOM	5	N/A	0.482	0.455	N/A
	9	N/A	0.426	0.430	N/A
MST	5	N/A	0.794	0.350	N/A
	9	N/A	0.899	0.010	N/A
SoT	5	N/A	**0.546**	**0.589**	N/A
	9	N/A	**0.486**	**0.572**	N/A
		Political Blogs (5×5)			
SOM	5	0.861	0.854	0.068	0.064
	39	0.827	0.844	0.061	0.056
MST	5	0.530	0.960	0.006	0.148
	39	0.512	0.560	0.000	0.141
SoT	5	**0.854**	**0.884**	**0.093**	**0.168**
	39	**0.767**	**0.785**	**0.226**	**0.178**

Table 3.4: Competitive performance on graph datasets. The quality criteria are averaged over 10 runs.

SoT are mostly equal. However, the poor results for MST are quite disappointing. On the "Football" dataset, our method is superior where density is equal to 0.791. On the contrary, the difference in conductance values is not significant. Unlike the "Adjective and Noun" dataset, the clustering quality of MST appears much more better but still far behind. For "Les Miserables", a small set like "Adjective and Noun", on all quality criteria there is no method that obtains the best performance. However, SoT produces satisfactory results. The best values of SoT is on the "Political blogs" dataset where the proposed method dominates all the quality criteria except for Accuracy.

Comparing the two tree-structure methods: MST and SoT. In general SoT manages to output better values in term of quality measures while results obtained from MST are unexpected. MST has a weakness: dependence on the distribution of data. It works moderately well in the case of "Football Teams" where the nodes are almost uniformly distributed, i.e. their difference in degree is not too considerable. The clusters are well separated one from each other. The only competitive method with SoT is SOM.

3.3.2.5 Graph visualization: decomposition and summarization

We are interested here to present the advantage of SoT for graph visualization. Our objective is to propose summarized graphs by building tree representation that deletes inconsistent edges in the original graph, i.e. the edge distance values are significantly larger for nearby edges in the trees. SoT is applied on graph datasets to detect and remove the inconsistent edges, which results in a set of disjoint trees. Each tree represents a cluster and possesses topological links connected to the neighbor trees. These topological links provided by SoT structure permit to not disconnect the graph.

Our advantage is to simultaneously propose multi-level structure: hierarchical and topological. These structures simplify the exploitation of the graph by proposing a summarized graph. We use also Tulip [Auber 2003] as the framework to visualize the graph. Using GEM layout, we provide here two types of architecture for graph visualization:

1. **Original view**: the default graph is drawn from the original collection of edges and vertices. We discriminate the leaders from the other nodes using the symbol L and the node size that varies with their degree.

2. **Summarized view**: here we propose a new organization of graph which is more visible and easier to visualize and analyze than the original organization. The new graph is drawn from the graph nodes as well as map nodes, where their structure has a form of hierarchy and topology given by SoT. The map cells (square) are located in the center surrounded by trees. Links between map cells represent the topological links defined by SoT. We have the same number of trees as the map size defined a prior. A tree including its root is attached to the respective map node.

In the aim to analyze the connection between a pair of node, we have introduced a new visualization illustrated in Fig. 3.9 and 3.10. For this purpose we explore the structure of the graph by analyzing the added and/or the deleted direct links between a pair of nodes in the original graph. In the proposed visualization *Red links* represent the original links (links are founded in both the original graph and the summarized graph). *Green links* represent the synthetized links (new direct links created by SoT in the summarized graph). *Gray links* represent the non-used links (the original links of the default graph that are not used in the summarized graph).

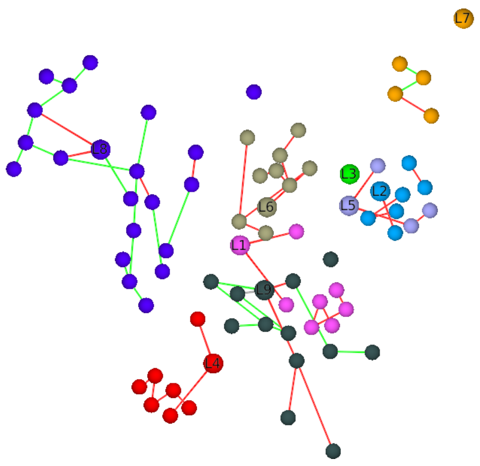
After studying the visualizations we notice that visual results given by SoT lead to important insight on the graph content. Our approach tries to improve the standard visualization by building summarized topological and hierarchical ordered clusters. SoT preserves and maintains links when graph is dense and contains a large number of edges. Whereas, SoT deletes links when the graph is sparse (with a less number of edges) or when clusters are far away from each other. In this last case SoT creates a short path to maintain a link between similar nodes. Atypical nodes are also clearly pinpointed with this approach and can be further studied by the analyst. The SoT decomposition provides a clear visualization in which the analyst can easily navigate. A hierarchical visual exploration is provided by descending from topological level to the last level of trees, which provides useful information about the clustering and nodes.

In the multi-level visualizations, each tree can be considered as a small cluster. Due to the topological relationship, we eliminate isolated vertices or isolated groups of vertices. A node is isolated if its degree is equal to 0, a group of vertices are isolated if these vertices have only internal connections between them and not external connections to the others. We also note that nodes with the highest degree are not always selected as leaders because some nodes have external edges towards other clusters. In multi-level visualizations, leaders are not central of clusters. This is due to two reasons: leader is not the centroid and the position of leaders depends on the order of nodes which were firstly selected to connect to the root.

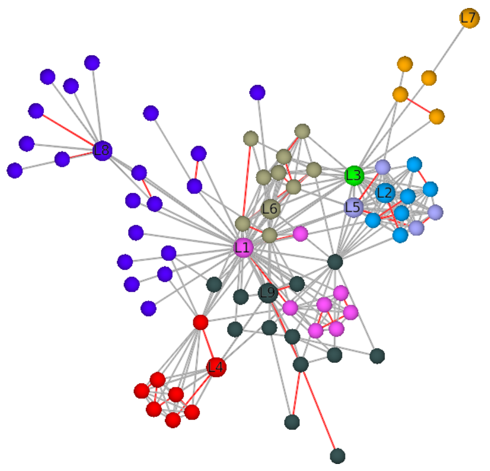
Case of "Les Miserables"

The visualization of "Les Miserables" is shown in Figure 3.9. Each cluster is represented by a unique color. We note that 57.89% of direct edges in the summarized view exist in the original graph. We also compute the percentage of edges (52.36%) formed between a pair of nodes in the original graph but these edges are removed in the summarized view and these two nodes are separated into two different cells.

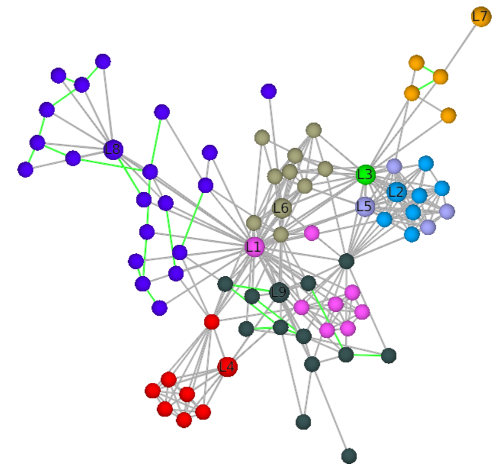
Looking to Figure 3.9(a), 3.9(b) and 3.9(c), we observe that when clusters in the original graph are dense, SoT preserves the majority of original links. These links correspond to red links in Figure 3.9(a) and 3.9(b). Whereas SoT does not maintain



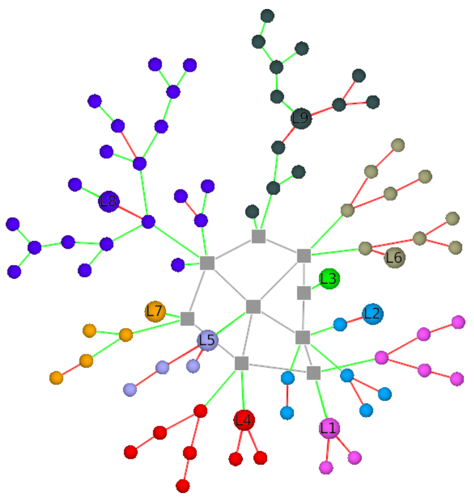
(a) Original graph. The red links represent the original links that exist in both the original and the summarized graph; the green links represent the synthetized links or the new shortest paths given by SoT.



(b) Original graph. The red links represent the original links that exist in both the original and the summarized graph; the gray links represent non-used links.



(c) Original graph. The green links represent the synthetized links or the new shortest paths given by SoT; Gray links represent non-used links.



(d) Summarized graph. Links between square nodes represent topological links.

Figure 3.9: Graph visualization of "Les Miserables". Each cluster provided by SoT is set with a unique color.

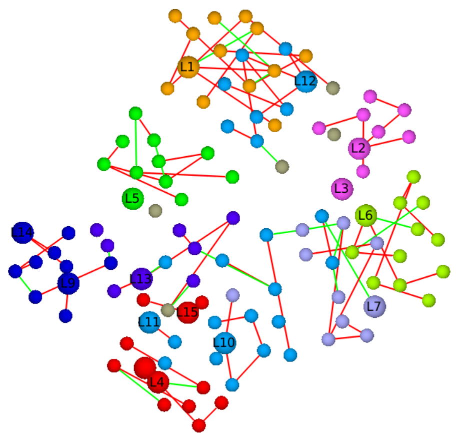
links in the clusters that have a low density, as an example we can observe the yellow and purple cluster which represent the isolated cluster located in the top right in Figure 3.9(a). However even SoT removes links from the original graph, it creates the new shortest paths that best represent clusters with a low density. The synthetized links are represented by the green links in Figure 3.9(a) and 3.9(c).

Case of "Football"

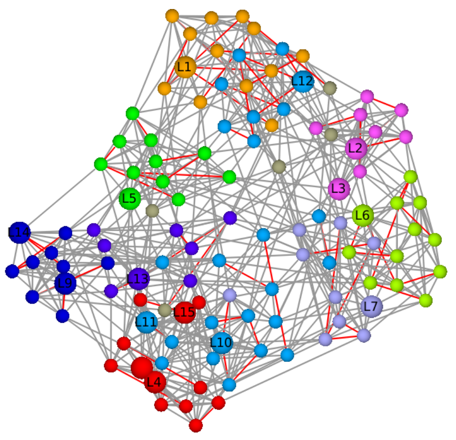
Different from the other, this dataset has more balanced distribution of data. "Football" dataset has 115 nodes classified into 10 different classes, the obtained visualization is shown in Figure 3.10. Each class label is represented by a single color. We have chosen a number of cells superior than the number of classes (i.e. 15 cells vs 10 classes). The number of nodes are quite balanced in each cell. Figure 3.10(d) shows several pure clusters represented by leaders $L1$, $L2$, $L6$. In this case, the differences between the proposed clustering and the ground truth are not important. The SoT structure and the original graph have several common links (see Figure 3.10(a), 3.10(b)). Indeed, Football dataset being very dense, as explained previously, SoT preserves much more original links. Thus we observe that 79.54% of direct links in the summarized view exist in the original graph.

Conclusion

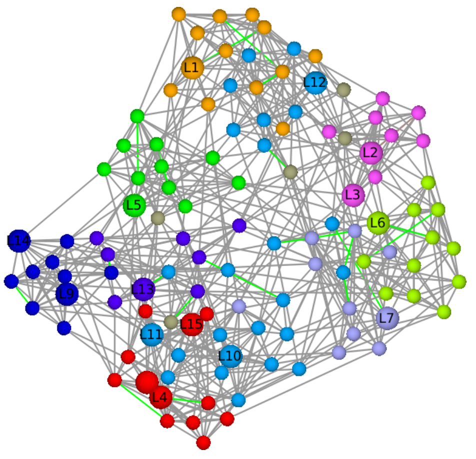
In the graph summarization problem, the other methods try to compress the original graphs by reducing the number of nodes as well as the number of edges. In this second application, we have approached this problem using SoT. It results in reducing only a large number of graph edges but not nodes. It will be hard to perform a comparison with existing methods. The links in the SoT structure should be studied profoundly by experts ou by user-validations. As consequence, we carry out another experiment on the protein dataset with an expert.



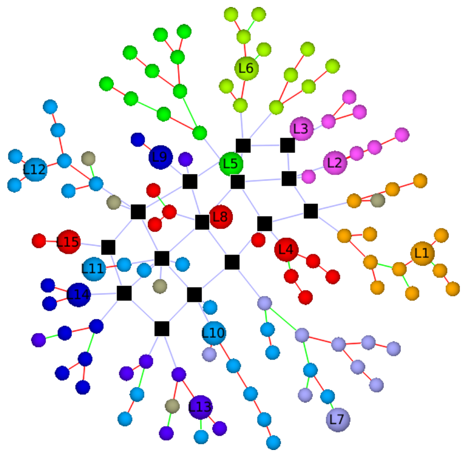
(a) Original graph. The red links represent the original links that exist in both the original and the summarized graph; the green links represent the synthetized links or the new shortest paths given by SoT.



(b) Original graph. The red links represent the original links that exist in both the original and the summarized graph; the gray links represent non-used links.



(c) Original graph. The green links represent the synthetized links or the new shortest paths given by SoT; Gray links represent non-used links.



(d) Summarized graph. The links between square nodes represent the topological links.

Figure 3.10: Graph visualization of "Football". Each class label is set with a unique color.

3.3.3 Third application on biological datasets

3.3.3.1 Motivation

In bioinformatics, visualizing high-dimensional and large-scale biological datasets such as protein or gene expression data is a challenge. Proteins receive attention from experts because their structures harbor information that is not immediately obvious without integration and analysis. Graphical representations help experts to analyze and interpret easily protein datasets.

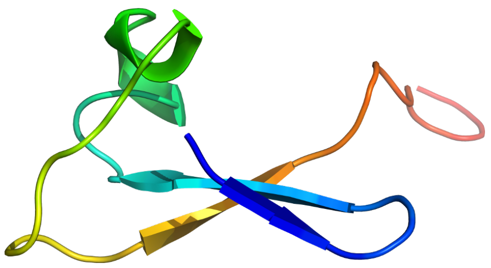
Proteins are large polymeric molecules made of one or more amino acids chains. They support a wide range of biological functions such as structure, transport, binding, catalysis, and so on. The function of a protein is closely related to the three dimensional (3D) arrangement (fold) of its chain into a compact structure. This structure can be described at many levels: locally at the atomic level to understand biochemical processes and more globally at the domain level to understand evolutionary processes. A protein domain can be define as a compact module of the protein structure shared by multiple proteins. Usually a protein can be made by one or more domains where each domain may support a different role of the protein function. Protein domains are considered to have a specific sequence and structure evolution pattern.

The Protein Data Bank [Bernstein 1977] (PDB) offers a comprehensive view of the available three-dimensional (3D) protein structures. In order to investigate their evolutionary relationships by detecting shared similarities at the sequence and at the structure level, a first step is to establish a classification of these objects. As for all complex objects, the clustering process is a difficult task. It can be done automatically by an algorithm, *manually* by a human expert or by combining automated and manual approaches. So far, reference protein structure classifications used in structural biology like SCOP [Murzin 1995] or CATH [Pearl 2003] are completely or partially constructed by human experts. If they present the advantage of being of high quality, the counterpart is the difficulty to maintain their exhaustivity in a context of quadratic growth of solved 3D-structures. For example, the last release of SCOP classification [Andreeva 2008] (version 1.75) done in june 2009 do not take into account PDB entries added from that date which represent more than 25000 structures and a growth of about 40% between 2009 and 2012. To face with this issue a first solution consists in the development of more accurate algorithmic tools to cluster or classify automatically 3D biological molecules. Another way is to propose to human experts new solutions to help in the investigation of protein structure similarities.

In this work we propose an original visualization tool to investigate protein domain structure similarity proximities.

3.3.3.2 Expert classification of protein domains

The protein domain dataset used in this study derives from the SCOP_v_1.75 classification [Andreeva 2008] which is available at http://scop.berkeley.edu/sunid=0 and its statistics are resumed in Table 3.5. It contains 110800 domains described in term of sequences relying to 38221 solved 3D structures of the Protein Data Bank. They are organized in a 4-level hierarchical classification, namely the "Class", "Fold", "Super Family" and "Family" levels, organizing the domains by increasing similarities. For example, domains in the same "Class" only share very coarse structural similarities (secondary structure composition) and domains of the same "Family" are selected to share a high similarity level in term of length and fine spatial organization of the amino-acids.


(a) >d1co4a_ 7.47.1.1 (A:) Zinc domain conserved in yeast copper-regulated transcription factors

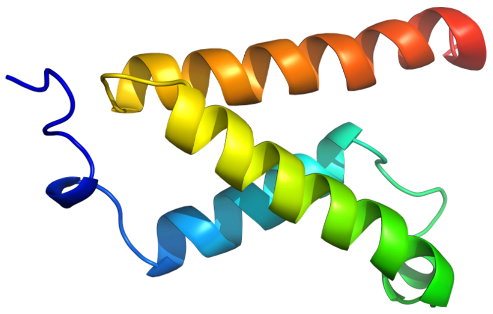

(b) >d2aghb1 1.12.1.1 (B:586-672) Kix domain of CBP (creb binding protein)

Figure 3.11: Sample protein domains

Class	# Folds	# Super Family	# Family
1: All alpha proteins	284	507	937
2: All beta proteins	174	354	820
3: Alpha and beta proteins (1/2)	147	244	911
4: Alpha and beta proteins (1+2)	376	552	1190
5: Multi-domain proteins (1 and 2)	66	66	101
6: Membrane, cell surface proteins, peptides	57	109	129
7: Small proteins	90	129	231

Table 3.5: SCOP_v_1.75 statistics

From these 110800 protein domains we kept the ASTRAL_40 subset [Brenner 2000]: the 3D structures of SCOP domains presenting less than 40%

sequence identity. For each pair of protein domains, a structure alignment is done with Yakusa [Carpentier 2005]. This algorithm seeks to find the set of longest common sub-structures between a query protein domain and any protein of a data base. Each alignment is evaluated in term of a z-score measuring the significance of spatially compatible aligned blocks relatively to found alignments: higher is the z-score, more significant is the alignment and more similar the protein domains are considered.

3.3.3.3 Graph of pairwise similarities

Let $G_z = (V, E)$ be the graph of pairwise protein domain structures similarities defines as in [Santini 2012]. Each vertex $v_i \in V$ represents a protein domain, and an edge (v_i, v_j) occurs between two domains if their alignment provided by Yakusa presents a $z\_score > z$ where $z\_score = \frac{s_i - \overline{S}}{\sigma_S}$. Figure 3.12 displays the original graph ASTRAL G_7 and G_6 and Table 3.6, 3.7 and 3.8 show the statistics of G_6 and G_7. The protein domains from the same SCOP "Class" in the 4-level hierarchical classification have the same tone of color, i.e. all the "alpha" proteins are associated with red tone; all the "beta" proteins are associated with green tone; all the "alpha and beta" proteins are associated with blue tone, etc. Within each tone, each SCOP "Family" has a unique color. Given two vertices v_i and v_j with four labels according to the 4-level hierarchical SCOP classification $class(v_i) = \{l, m, n, o\}$ and $class(v_j) = \{l', m', n', o'\}$ where l, m, n and o (ie l', m', n' and o') stand respectively for "Class", "Fold", "Super Family" and "Family" SCOP identificators. More precisely, let's take an example in Figure 3.11. In Figure 3.11(a), *d1co4a_* indicates the protein id; 7.47.1.1 indicates that "Class": $l = 7$, "Fold": $m = 47$, "Super Family": $n = 1$, Family $o = 1$. In Figure 3.11(b), *d2aghb1* indicates the protein id; 1.12.1.1 indicates that "Class": $l = 1$, "Fold": $m = 12$, "Super Family": $n = 1$, Family $o = 1$.

Class	# Folds	# Super Family	# Family
1: All alpha proteins	192	320	505
2: All beta proteins	123	265	562
3: Alpha and beta proteins (1/2)	127	202	674
4: Alpha and beta proteins (1+2)	243	329	656
5: Multi-domain proteins (1 and 2)	52	52	72
6: Membrane, cell surface proteins, peptides	55	79	89
7: Small proteins	66	98	167

Table 3.6: ASTRAL G_6 statistics

Class	# Folds	# Super Family	# Family
1: All alpha proteins	126	206	317
2: All beta proteins	99	204	416
3: Alpha and beta proteins (1/2)	108	164	549
4: Alpha and beta proteins (1+2)	181	236	462
5: Multi-domain proteins (1 and 2)	38	38	57
6: Membrane, cell surface proteins, peptides	55	79	89
7: Small proteins	66	98	167

Table 3.7: ASTRAL G_7 statistics

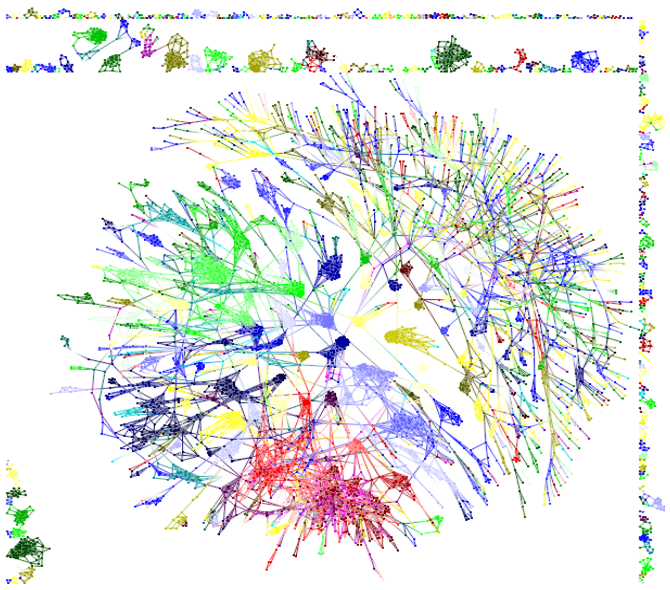
Class	# Data/G_6	# Data/G_7
1: All alpha proteins	1188	786
2: All beta proteins	1822	1482
3: Alpha and beta proteins (1/2)	2397	2103
4: Alpha and beta proteins (1+2)	2004	1617
5: Multi-domain proteins (1 and 2)	168	153
6: Membrane, cell surface proteins, peptides	149	118
7: Small proteins	499	347
Total	8227	6606

Table 3.8: Data statistics per class. The last row shows the total number of data in the G_6 and G_7

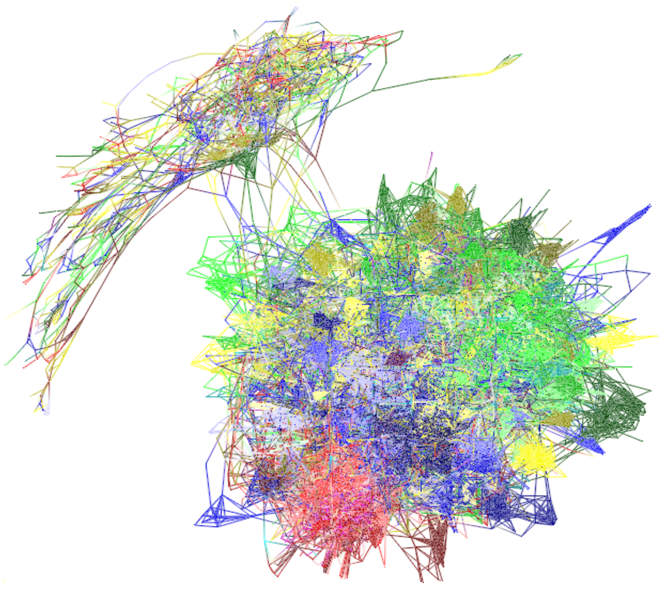
We define the class distance Δ_{class} between two vertices as a function of their SCOP classification as following:

- $\Delta_{class}(v_i, v_j) = 0$ if $class(\mathbf{x}_i) = class(\mathbf{x}_i)$ or $l = l'; m = m'; n = n'; o = o'$
- $\Delta_{class}(v_i, v_j) = 1$ if $l = l'; m = m'; n = n'$
- $\Delta_{class}(v_i, v_j) = 2$ if $l = l'; m = m'$
- $\Delta_{class}(v_i, v_j) = 3$ if $l = l'$
- $\Delta_{class}(v_i, v_j) = 4$ otherwise

It is clear that both graphs G_7 and G_6 show many difficulties to explore the graph. Thus, this is one of the motivations to use SoT algorithm, which allows decomposing the original graph into another clear structure. This structure has the distinction of being topological and hierarchical.



(a) G_7



(b) G_6

Figure 3.12: Original graph of the ASTRAL G_7 and G_6 dataset. Each SCOP "Family" has a color in the SCOP "Class" color tone.

3.3.3.4 Experimental protocol

We set input parameters as following:

1. Each dataset must be normalized before training. For each dataset, 10 tests are conducted using random initializations.

2. The shape of map's topology is rectangular. A network node must have at least one data object.

3. As a graph dataset, we will use the Laplacian (Equation 2.10) to generate data space $\mathcal{X}$. The number of the selected smallest eigenvectors of the Laplacian is equal to the number of clusters, $\lambda = K$.

4. For graph datasets, leaders [Stanoev 2011] are considered as prototypes. Instead of following Line 23 in Algorithm 8, the prototype expression is modified as following:
$$\mathbf{w}_c = \arg \max_{\forall i \in tree_c} (deg(v_i)) \tag{3.5}$$
where the local degree $deg(v_i)$ is the number of internal edges incident to v_i.

3.3.3.5 Study on the removed and synthetized edges

$m_z^{K,dist}$	Δ_{class}	4	3	2	1	0	M_z^K
G_7	$K = 900$	653	490	287	3274	11257	15961
	$K = 2000$	629	477	292	3281	11489	16165
	n_7^{dist}	699	518	309	3526	13147	18199
G_6	$K = 900$	2789	1999	738	6558	17402	29486
	$K = 2000$	2793	1997	746	6620	17752	29908
	m_6^{dist}	2793	2000	747	6690	18258	30488

Table 3.9: The number of removed edges relative $m_z^{K,dist}$ to the class distance. M_z^K is the total number of removed edges in the respective graph given by SoT for each value of K. m_z^{dist} is the number of edges in G_z relative to the class distance.

The ratio $R^{dist} = \frac{m_z^{K,dist}/M_z^K}{m_z^{dist}/M_z}$ is a measure of how SoT removes edges in function of distance $dist$ during learning. M_z is the total number of edges in graph G_z; M_z^K the total number of edges removed from G_z by SoT algorithm for each value of K; $m^z_{K,dist}$ the number of G_z edges between vertices at class distance Δ_{class}; and m_z^{dist} the number of edges in G_z between vertices at class distance Δ_{class}. As shown in Table 3.10 for different values of K and z, we observe that the smallest value of R^{dist}

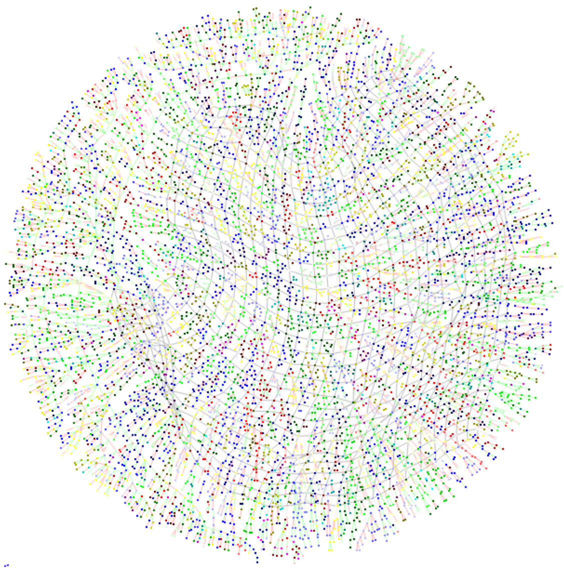
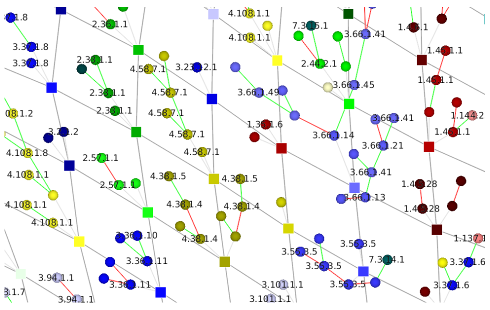
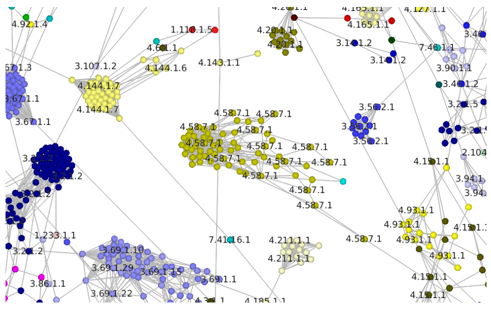


Figure 3.13: ASTRAL G_7 - SoT: hierarchical and topological structure. Each node has respectively its color in the original graph corresponding to SCOP class.



(a) SoT structure extracted from Figure 3.13



(b) Original graph extracted from Figure 3.12(a)

Figure 3.14: Sample visualizations for $z = 7$ centered on cluster 4.58.7.1

R^{dist}	Δ_{class}	4	3	2	1	0
G_7	$K = 900$	1.065	1.0786	1.0590	1.0587	0.9763
	$K = 2000$	0.9880	1.0184	1.0396	1.0483	0.9838
G_6	$K = 900$	1.0325	1.0335	1.0215	1.0136	0.9855
	$K = 2000$	1.0194	1.0179	1.0180	1.0087	0.9911

Table 3.10: The ratio of removed edges R^{dist}

Table 3.11: Number of synthesized edges with respect with the distance for each value of K in the G_7 and G_6 graph.

	Δ_{class}	4	3	2	1	0	Total
G_7	$K = 900$	950	477	92	509	611	2639
	$K = 2000$	633	189	22	80	183	1107
G_6	$K = 900$	3307	1453	130	365	394	5649
	$K = 2000$	3049	1314	69	235	265	4932

is reached when $\Delta_{class} = 0$. So for each value of K, SoT removes relatively less edges between data within the SCOP "Family".

SoT creates a tree topology structure where each tree node represents a data. Deleting links is not enough to reach this structure, thus SoT also creates synthesized edges to build tree structure. In the original graph, these synthesized edges allow to find a path between the tree nodes belonging to the same class. Table 3.11 lists the number of synthesized edges for different values of K. It is difficult to see the result on the original graph, but it is clear that the tree structure provided by SoT in Figure 3.14(a) and 3.16(a) decreases complexity of visualization and exploration of these graphs. Here we provide various views for data visualization which proofs that our method can offer more useful information. Tulip [Auber 2003] is used as the framework for the visualization. In this section, we only use $K = 900$ to visualize the ASTRAL G_7 and G_6 datasets.

Figure 3.13 and 3.15 present tree topology view of ASTRAL G_7 and G_6 respectively. The map cells (squares) are located in the center surrounded by trees. Figure 3.14 and 3.16 show two zooms of a region in the SoT structure and a related zone of the original graph involving same data points for comparison. Here Figure 3.14(a) and 3.16(a) complete the visualization along with the original graphs presented in Figure 3.14(b) and 3.16(b).

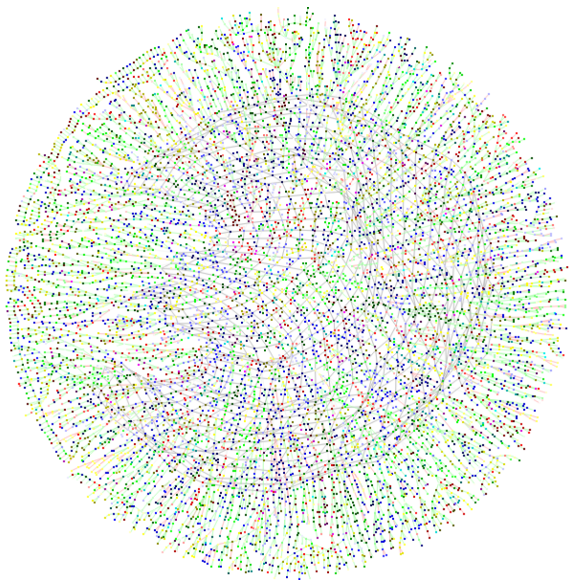
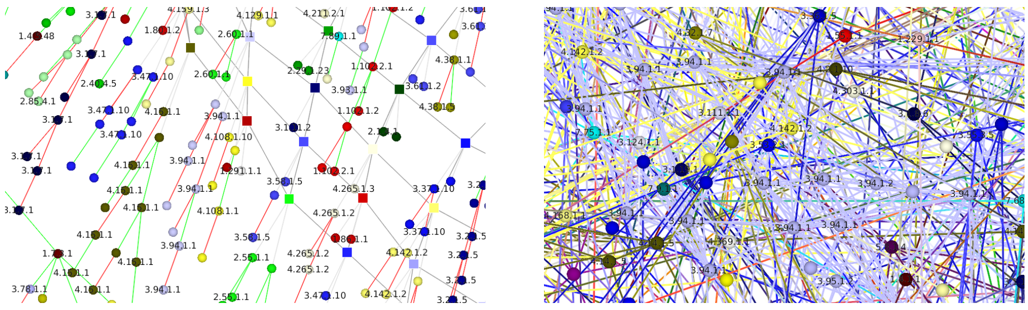


Figure 3.15: ASTRAL G_6 - SoT: hierarchical and topological structure. Each node has respectively its color in the original graph.



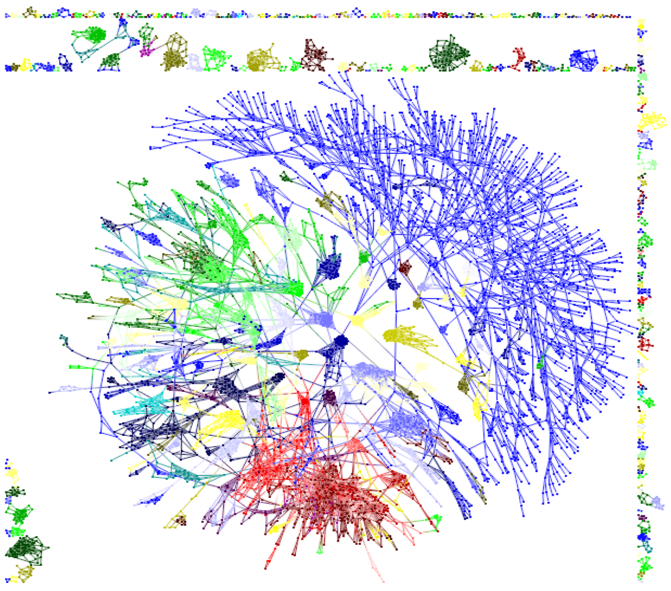
(a) SoT extracted from Figure 3.15

(b) Original graph extracted from Figure 3.12(b)

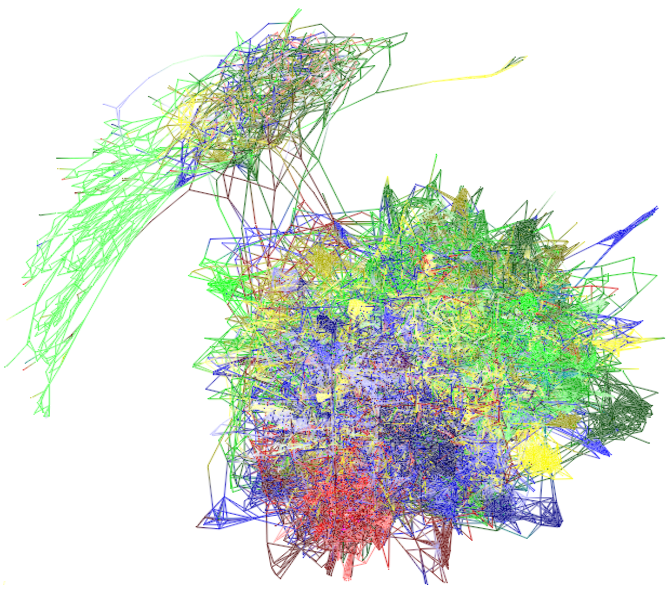
Figure 3.16: Sample visualizations for $z = 6$ centered on cluster 3.91.1.1.

3.3.3.6 Clustering quality

In this dataset, the protein domains are connected by pair to form an aligmnent. For this reason, the clusters which contain only one protein domain are eliminated. We don't take into account these clusters but only the ones with at least two protein domains to compute the clustering quality. SoT provides a good clustering: the Rand values are close to 1 in Table 3.12. Moreover we can compare the clustering quality provided by SoT in visual aspect. Figure 3.17 shows the original graph of the ASTRAL G_7 and G_6 dataset after applying the majority vote rule. These figures enable the difference in color with Figure 3.12.



(a) G_7



(b) G_6

Figure 3.17: Original graph of the ASTRAL G_7 and G_6 datasets with the majority vote.

	K	NMI	Rand
G_7	900	0.943	0.781
	2000	0.941	0.805
G_6	900	0.994	0.731
	2000	0.994	0.739

Table 3.12: Quality measure (NMI and Rand) for G_7 and G_6

3.4 Conclusion

We have presented in this chapter the first approach named Self-Organizing Trees. This novel method provides a new look at self-organizing models allowing better hierarchical clustering. Thus, we improve their applicability to structured (graph) and no structured datasets for the clustering problem. The SoT network is able to determine both the hierarchical distribution of the nodes and its structured topology. In addition, SoT reveals the ability to represent data distribution in multiple levels of granularity, and this is achieved without the necessity of computing the entire tree again. In practice, the SoT model provides tree-like for every network node. This leads us to find the proximity between two trees which are located in neighbor network nodes. We may conclude that those trees belong to one cluster.

Furthermore, our model proposes a friendly visualization space by offering a summarized visualization in the case of graph dataset. The benefit of our approach is to fast analyze with different visualizations from the general dataset to a particular part of data. We notice that this method works well with several real world datasets through the experiments.

In the bioinformatics context, the SoT approach revealed to be very efficient. It allows the exploration of protein domains. It simplifies the representation of structural neighborhoods, not only by reducing and reorganizing the links between protein domains, but also by providing a relevant clustering of 3D structures coherent to the stand of the art expert classifications. In addition, tree organization that is provided by SoT algorithm is of great interest and should be investigated in a further analysis. In particular the hierarchy of structure and the derived distances that could be computed between proteins from these trees could give evolutionary informations helping to investigate processes that drive the conservation or the divergence of protein structure and function.

The purpose of SoT for protein dataset certainly needs more investigations and testing. We want to perform a user-study to highlight the possible application of the proposed method to interactive clustering. The main advantage of Interactive Protein Data Clustering is that the expert has a high confidence in these results because he/she has visually validated them.

Chapter 4

Growing Self-Organizing Trees

Contents

4.1 Introduction

Discovering the inherent structure and its uses in datasets has become a major challenge for data mining applications. An attractive way to help analysts to explore data is to use unsupervised approaches that allow for the clustering and mapping of high-dimensional data in a low-dimensional space. In the previous chapter, we have shown how SoT can address both clustering and visualization at the same time. However, as one of SOM variants, SoT is sensible to the initial parameters a prior. In this study, we introduce the second model GSoT that can overcome this issue.

Other SOM variants allow sensitivity to topology to be overcome by dynamically growing the grid or network, including the Growing Neural Gas (GNG) [Fritzke 1995b], and a growing hierarchical self-organizing map [Michael Dittenbach 2002]. A growing algorithm is generally used for learning topological preservation, clustering, vector quantization and quick data indexation [Costa 2007]. Identifying these hierarchical relations cannot be addressed conveniently within the GNG or SOM models. We do not aim to find an optimal data clustering but to obtain good insight into the data cluster structure for data mining. AntTree has been adapted into the self-organizing model in the previous chapter. Here by combining the GNG and AntTree, the incrementally hierarchical and topological structure is offered. The clustering method should thus be robust and visually efficient.

The GSoT method can assure the clustering task and provide hierarchical and topological structure. A tree is constructed from data assigned to its respective cluster (or network node). Tree building is based on rules from the AntTree method. The growing algorithm is the adaptive process where a network adapts supports to cover the data distribution. In other words, network nodes follow the probability density of input data. Statistical information is used to determine an appropriate position to insert a new cell in the network [Fritzke 1995b]. Due to its nature, the network size is incremental over time, and the evolutive network topology depends on the input data. The reference vector position varies with the random selection of data used in the training step. A connection has an *age* variable that is used to decide when to remove old edges and keep the topology updated.

In Section 4.2, we will present the GSoT principles implemented from GNG. In Section 4.3 the experiments are performed on several real-world datasets. In this chapter, we also discuss about the acceleration process in the GSoT algorithm. This chapter ends up with the conclusion in Section 4.4.

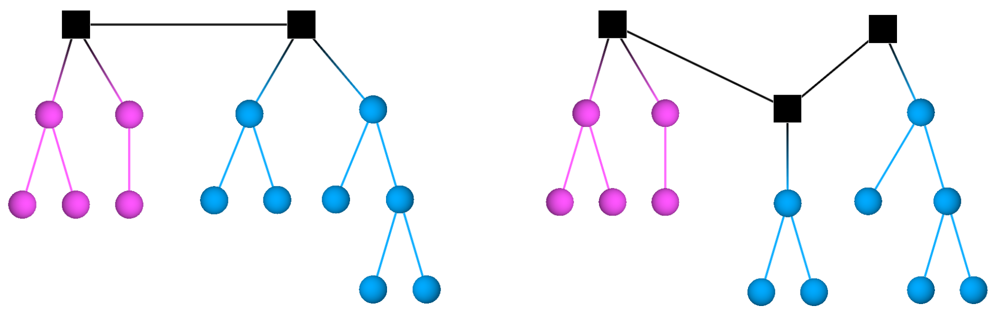
4.2 Principles

4.2.1 Incrementally hierarchical and topological clustering

The GSoT algorithm seeks to successively add new trees to an initially small network by evaluating local statistical measures. During one epoch, GSoT generates a number of trees corresponding to the number of units in the network. Each network unit represents a tree root and is associated with a prototype.

The GSoT output space contains a set of trees arranged in a network, usually in 1D or 2D (see Figure 4.1). The number of trees is incremental, and the topology is evolutive. Similar to SoT, each tree node represents one single data object. This allows an immediate comparison among data using visualization. A hierarchical structure can be exploited by descending from the topological level to the last level of trees, which provides useful information about the clustering, data structure and data topology.

The GSoT principles are defined as the example in Figure 4.1. Black square nodes refer to supports (network units); circle nodes refer to tree nodes, which correspond to input data (observation). Two trees are topological neighbors if an edge is created between their supports. The network is first initialized with only two supports, as



(a) GSoT structure: two trees linked by a topological connection (b) New GSoT structure adapting to an inserted new tree

Figure 4.1: The GSoT architecture in two successive growing process epochs. Black square nodes refer to supports. Circle nodes refer to observations.

in Figure 4.1(a). These supports are considered hierarchical tree roots. When all data are connected to their best match tree, we add a new support to extend the network. The next step is computing new assignments for input data. When the data and their associated sub-trees have found their new best match support, they are disconnected from the old tree and reconnected to the new one. The network grows incrementally, and several hierarchical trees are obtained while the training process still continues (Figure 4.1(b)).

4.2.2 Algorithm

GSoT is autonomous because we have limited parameters. We do not fix the number of observations used in the assignment step, as does the traditional GNG. Our method constructs tree-like organizations over time. The only required parameters are stopping criteria or quantization errors. The algorithm and new support insertions stop when a maximum tree has been reached. Topological neighbors are considered to update the tree network. The GSoT network consists of:

- A set of K trees (or K clusters $\mathcal{C} = \{C_1, .., C_K\}$ associated with $\mathcal{W} = \{\mathbf{w}_1, .., \mathbf{w}_K\}$). A tree root is represented by a network node that does not contain any information of input objects. Only the internal or leaf nodes represent the input objects.

- A set of connections between pair of supports. These connections preserve the topological relations between the trees in the network. Connections are characterized by an age. At begining, all connection ages are set to zero.

We use in this section the same notations presented in Chapter 3. The GSoT algorithm (see Algorithm 11) proceeds by alternating four steps: assignment, tree construction, prototype adaption, tree insertion in order to minimize the quantization error:

$$error = \arg\min \sum_{k=1}^{K} \sum_{i=1}^{n_{C_k}} \|\mathbf{x}_i - \mathbf{w}_k\|^2$$

In order to stop autonomously the algorithm, we can employ a threshold of quality measure such as the quantization error. Or simply, we just consider a number of trees as stopping criterion [Martinetz 1991, Fritzke 1991].

Assignment

This step (see Line 6 in Algorithm 11) is necessary to minimize the quantization error while supposed that the prototypes are all constant. The goal is to find the best match tree for each object. This assignment step is extactly the same as that of SoT. The assignment function is as in Equation 3.2. Then we will verify whether a sub-tree exists, it follows the principe of the sub-tree assignment.

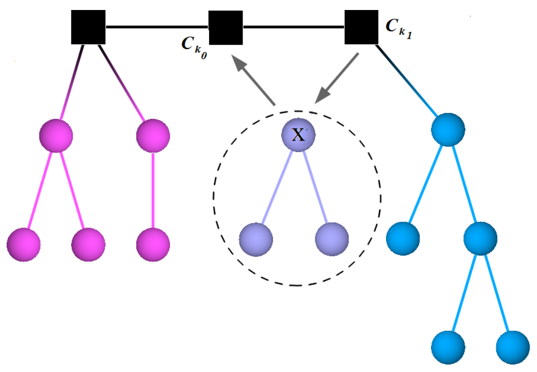


Figure 4.2: Sub-tree assignment from C_{k_1} to C_{k_0}

We recall here how to do the sub-tree assignment in Figure 4.2. In this example, $subtree_{\mathbf{x}}$ consists of three tree nodes that get assigned from $tree_{k_1}$ to $tree_{k_0}$. These three nodes are now found in the new network node C_{k_0} with the same hierarchy as in the old one C_{k_1}.

Tree construction

The connecting and disconnecting rules were used in the SOM model. This time they will be adapted in the GNG model. The tree construction step is entirely realized by the functions *constructTree* and *constructTreeSub* presented respectively by Algorithm 9 and 10) in Chapter 3. Here each object will pass a series of similarity tests to be connected to a tree or disconnected from an existing tree.

Algorithm 11 GSoT algorithm

```
1: intialize two prototypes $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2\}$
2: **while** stopping criteria have not been fulfilled **do**
3:    initialize $\mathcal{L}$
4:    **while** $\mathcal{L}$ is not empty **do**
5:       $\mathbf{x}_i$ is the first data from $\mathcal{L}$
6:       $k_0 = \phi(\mathbf{x}_i)$    // *find the best match tree*
7:       **if** $\mathbf{x}_i$ is *initial* **then**
8:          $\mathbf{x}_i^{pos} = 0$    // *set the tree root as the position of* $\mathbf{x}_i$
9:          $constructTree(tree_{k_0}, \mathbf{x}_i, \mathbf{x}_i^{pos})$    // *connect* $\mathbf{x}_i$ *to* $tree_{k_0}$
10:      **else**
11:         **if** $\mathbf{x}_i^{pos}$ is in $tree_c$ **then**
12:            $\mathbf{x}_i^{pos}$ gets the current position of $\mathbf{x}_i$
13:         **else**
14:            $\mathbf{x}_i^{pos} = 0$
15:         **end if**
16:         $subtree_{\mathbf{x}_i} = \{\mathbf{x}_i$ and all tree nodes recursively and temporarily connected
            to $\mathbf{x}_i\}$
17:         **if** $\mathbf{x}_i$ is *connected* and $k_0 \neq k_{old}$ **then**
18:            $subtree_{\mathbf{x}_i}$ = disconnect $subtree_{\mathbf{x}_i}$ from $tree_{k_{old}}$
19:         **end if**
20:         **if** $\mathbf{x}_i$ is *disconnected* **then**
21:            $constructTree(tree_{k_0}, subtree_{\mathbf{x}_i}, \mathbf{x}_i^{pos})$    // *connect either* $\mathbf{x}_i$ *or*
               $subtree_{\mathbf{x}_i}$ *to* $tree_{k_0}$
22:         **end if**
23:      **end if**
24:      **if** $\mathbf{x}_i$ is *connected* **then**
25:         remove $subtree_{\mathbf{x}_i}$ from $\mathcal{L}$
26:      **end if**
27:      $updatePrototype(\mathcal{W}, k_0, \mathbf{x}_i)$
28:   **end while**
29:   $\mathcal{W}$ = **addNewNode**$(\mathcal{W})$    // *add new network node; also add a new empty*
      *tree associated with this new node*
30:
31: **end while**
```

Prototype adaptation

While the clusters are all fixed with a number of objects, this step requires to minimize the quantization error. GSoT follows the same process as the Neural Gas and

GNG model to update prototypes. Once an object has been assigned to a prototype, this prototype and its neighbors are ajusted and moved toward the assigned object according to the "winner take most" rule [Fritzke 1991]. For each time it happens, the age variables are also updated as in Algorithm 12.

Algorithm 12 updatePrototype: updating the prototypes

Input: $\mathcal{W}$, $k_0, \mathbf{x}_i$

Output: $\mathcal{W}$

```
1: for all $\mathbf{x}_j \in subtree_{\mathbf{x}_i}$ do
2:     increment the age of all edges from $k_0$
3:     add the squared distance between $\mathbf{w}_{k_0}$ and $\mathbf{x}_j$ to the local $error_{k_0}$:
           $error_{k_0} = error_{k_0} + \parallel \mathbf{w}_{k_0} - \mathbf{x}_j \parallel^2$
4:     move $k_0$ and its topological neighbors (i.e., all supports connected to $k_0$ by an
       edge) towards $\mathbf{x}_i$ by fractions $\varepsilon_b$ and $\varepsilon_r$ of the distance:
           $\Delta_{k_0} = \varepsilon_b(\mathbf{x}_j - \mathbf{w}_{k_0})$      (4.1)
           $\Delta_r = \varepsilon_r(\mathbf{x}_j - \mathbf{w}_r)$      (4.2)
           $\forall r$ is neighbor to $k_0$      (4.3)
5:     find the second nearest support $k_1$ of $\mathbf{x}_i$.
6:     if $tree_{k_0}$ and $tree_{k_1}$ are connected by an edge then
7:         set the age of that edge to 0.
8:     else
9:         create a new edge between them.
10:    end if
11:    remove the edges with an age larger than $Max_{age}$. If there are supports with
       no edges after this step, then remove these supports
12:    decrease the error of all network node by factor $\beta$
           $error_k = error_k - \beta \times error_k$
13: end for
```

Tree insertion

The network is made flexible by the possibility of adding new trees. New trees can only be added if an epoch has passed. In our case, an epoch is a number of iterations necessary to successfully connect all objects from $\mathcal{L}$. When a new tree is inserted into the network, objects are re-assigned again to define new tree structures. GSoT can thus avoid misclassification in the previous step. Algorithm 13 gives details of this step.

Algorithm 13 addTree: adding new trees into the network

```
Input: $\mathcal{W}$
Output: $\mathcal{W}$
1: find the network node $q$ with the maximum accumulated error.
       $q = \arg\max(error_k),\ \forall k = 1, .., K$
2: insert a new network node $p$ halfway between $q$ and its neighbor $f$ with the largest
   error:
       $\mathbf{w}_p = \frac{1}{2}(\mathbf{w}_q - \mathbf{w}_f)$
3: initialize a new tree for the node $p$
4: insert edges connecting $p$ with $q$ and $f$ and remove the edge between $q$ and $f$.
5: decrease the local errors of $q$ and $f$ by multiplying them by a constant $\alpha$
6: initialize the error value of $p$ with the new error value of $q$.
```

4.2.3 Computational complexity

Lemma 4.2.1 *GSoT has a complexity of* $\Theta(KN \log N)$.

Proof Algorithm 4 requires K epochs to build K trees. During one epoch, an operation includes three iterative steps: assignment, tree construction and updating support. GSoT algorithm uses a small data subset in each epoch. The number of assignments starts from 100% and decreases; few data must be re-assigned, which allows the run time to be reduced. To terminate training for N observations, these steps require $N \log N$ operations. A new support is added, and this step is executed once per epoch. Finally, GSoT has a complexity of $\Theta(KN \log N)$.

4.3 Experiments

In this section, several experiments are performed to show the effectiveness of GSoT for a clustering knowledge discovery task. The experimental part includes two phases: numerical and visual validation. Our experiments employ real-world datasets, which are used in the previous chapter. Table 4.3 re-summarizes the data features.

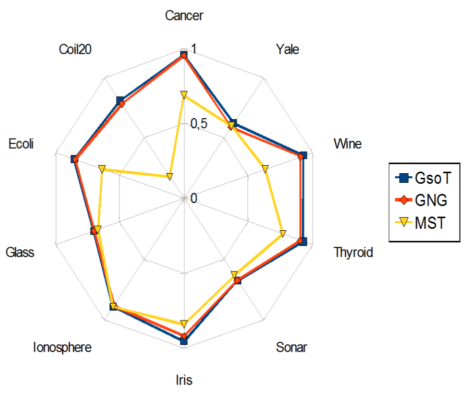
Dataset	**# data**	**# features**	**# class**
Arrhythmia	420	278	2
Cancer	699	9	2
COIL-20	1440	1024	20
Ecoli	336	7	8
Glass	214	9	6
Ionosphere	351	17	2
Iris	150	4	3
Sonar	208	60	2
Thyroid	215	5	3
Wine	178	13	3
Yale	165	1024	15

Table 4.1: Data features

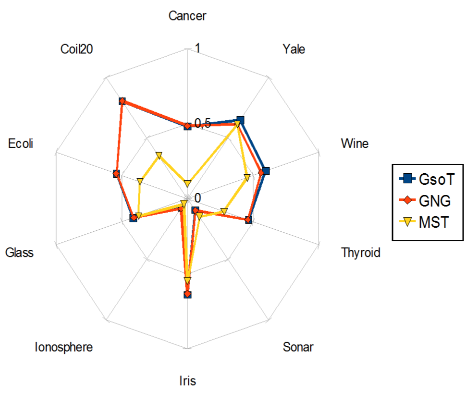
4.3.1 Numerical validation: competitive performance

In this experiment, we compare the GSoT model to two algorithms that possess similar structure or use the same initialization e.g. GNG and MST. For each dataset, we choose a value for K that is greater than the number of classes: $K = 30$ clusters for COIL-20 and Yale, $K = 20$ trees for Ecoli and Glass and $K = 10$ for the others. For each experiment, 10 tests are conducted using random selections with several constant parameters: maximum age (max = 50) and scaling factors to reduce errors in nodes ($\alpha = 5.10^{-2}$, $\beta = 5.10^{-3}$, $\varepsilon_b = 0.1$, $\varepsilon_r = 5.10^{-4}$).

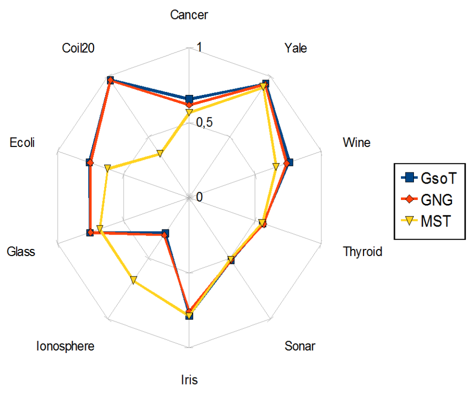
Table 4.2 and the radar chart (Figure 4.3) show the clustering quality of three algorithms. In practice, GSoT and GNG provide equivalent values in most selected datasets. MST is always less efficient than the others. Because of its poor performance, MST should be avoided. The most significant cases for which GSoT dominates all three measures are the Cancer, Iris, Wine and Yale datasets. For COIL-20, our method provides better Accuracy than the others (i.e., Acc = 0.809 vs. 0.782 and 0.181). For Ecoli, GSoT obtains better values in Accuracy (Acc = 0.856) and Rand index (Rand = 0.761). For Yale, although GSoT and GNG have almost equal Rand index values, our algorithm obtains higher values for Accuracy (Acc = 0.624) and NMI (NMI = 0.648).



(a) Accuracy



(b) Normalized Mutual Information



(c) Rand

Figure 4.3: Quality criteria in a radar chart

Dataset	**Method**	**Accuracy**	**NMI**	**Rand**
Cancer	GSoT	0.962	0.479	0.656
	GNG	0.957	0.484	0.618
	MST	0.691	0.097	0.567
COIL-20	GSoT	0.809	0.803	0.969
	GNG	0.782	0.805	0.968
	MST	0.181	0.353	0.360
Ecoli	GSoT	0.856	0.540	0.761
	GNG	0.848	0.542	0.753
	MST	0.639	0.363	0.622
Glass	GSoT	0.703	0.414	0.754
	GNG	0.700	0.410	0.750
	MST	0.672	0.374	0.681
Ionosphere	GSoT	0.892	0.075	0.291
	GNG	0.883	0.081	0.309
	MST	0.894	0.046	0.684
Iris	GSoT	0.954	0.641	0.789
	GNG	0.918	0.634	0.759
	MST	0.840	0.559	0.780
Sonar	GSoT	0.676	0.094	0.514
	GNG	0.673	0.096	0.513
	MST	0.634	0.148	0.508
Thyroid	GSoT	0.930	0.462	0.558
	GNG	0.907	0.458	0.566
	MST	0.772	0.241	0.553
Wine	GSoT	0.934	0.594	0.761
	GNG	0.910	0.559	0.740
	MST	0.634	0.453	0.666
Yale	GSoT	0.624	0.648	0.939
	GNG	0.593	0.613	0.933
	MST	0.600	0.612	0.911

Table 4.2: Competitive performance

4.3.2 Discussions: GSoT components

4.3.2.1 Growing performance

In this experiment, we study how GSoT trees evolve over time in their Accuracy and NMI values. We use the same parameters as in the previous experiment, except that $K = 30$ is fixed for all selected datasets. It is interesting to observe the clustering performance when the network is growing and has {5,10,20,30} trees. Table 4.3 summarizes the average GSoT measures versus the number of trees for 10 runs. For several datasets, including Cancer, Ionosphere, Iris, Thyroid and Wine, the Accuracy values increase from $K = 2$ to $K = 30$, while the NMI values decrease significantly. These datasets share one common feature: only two or three classes are known beforehand. For COIL-20 and Yale, a good performance is obtained when the network has at least 20 trees. Accuracy thus increases and NMI decreases over time. The possibility of overfitting exists. To avoid this disagreement, we run 10 tests for each experiment.

4.3.2.2 Number of assignments

The trees in GSoT are considered intermediary steps to memorize all assignments in the previous epoch. The ratio expression of the number of assignments is defined as follows:

$$Ratio = \frac{\text{number of assignments}}{\text{number of data}} \tag{4.4}$$

Figure 4.4 and Table 4.4 display the ratios of the number of assignments during training. The x-axis refers to the number of epochs, and the y-axis refers to the ratio of the number of assignments over the number of data. The entire dataset must be trained using the GNG algorithm (i.e., $Ratio = 100\%$), but the GSoT algorithm uses a small data subset for training in each epoch. Initially, the number of assignments starts from 100%; the number of new assignments then decreases because only few datasets must be re-assigned to new trees. Most selected datasets are in this case, where the ratio decreases to 40% after 29 first epochs. However, data can be disconnected and assigned to another best match tree several times because of autonomous connection/disconnection rules and updated supports at each epoch, which explains why several curves increase, including Cancer, COIL-20 and Thyroid.

Dataset	Criterion	5 trees	10 trees	20 trees	30 trees
Cancer	Accuracy	0.954	0.962	0.965	0.967
	NMI	0.555	0.479	0.407	0.392
COIL-20	Accuracy	0.206	0.357	0.609	0.809
	NMI	0.380	0.536	0.707	0.803
Ecoli	Accuracy	0.714	0.791	0.856	0.868
	NMI	0.530	0.552	0.540	0.527
Glass	Accuracy	0.506	0.614	0.703	0.733
	NMI	0.284	0.344	0.414	0.432
Ionosphere	Accuracy	0.891	0.892	0.897	0.903
	NMI	0.049	0.075	0.120	0.132
Iris	Accuracy	0.930	0.954	0.961	0.956
	NMI	0.717	0.641	0.567	0.535
Sonar	Accuracy	0.620	0.676	0.770	0.836
	NMI	0.038	0.094	0.185	0.252
Thyroid	Accuracy	0.900	0.930	0.931	0.951
	NMI	0.479	0.462	0.438	0.434
Wine	Accuracy	0.912	0.934	0.953	0.958
	NMI	0.655	0.594	0.550	0.528
Yale	Accuracy	0.203	0.340	0.573	0.624
	NMI	0.221	0.382	0.602	0.648

Table 4.3: Evolutive GSoT performance on the real-world datasets

Data-set	5 trees	10 trees	20 trees	30 trees
Cancer	0.605	1.047	1.034	0.767
COIL-20	1.037	1.121	0.713	0.392
Ecoli	0.458	0.776	0.529	0.229
Glass	0.869	0.500	0.163	0.144
Ionosphere	0.626	1.103	0.467	0.364
Iris	0.626	1.103	0.153	0.060
Sonar	0.626	1.103	0.153	0.060
Thyroid	1.288	0.644	0.120	0.028
Wine	0.584	0.735	0.196	0.050
Yale	0.848	0.575	0.121	0.072

Table 4.4: Assignment ratio

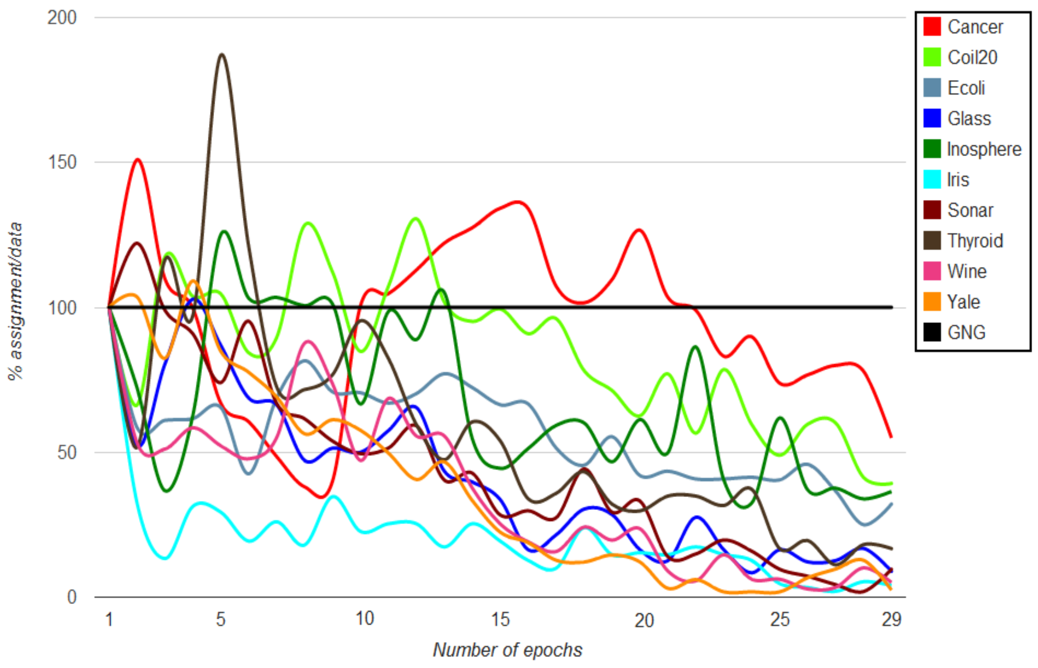
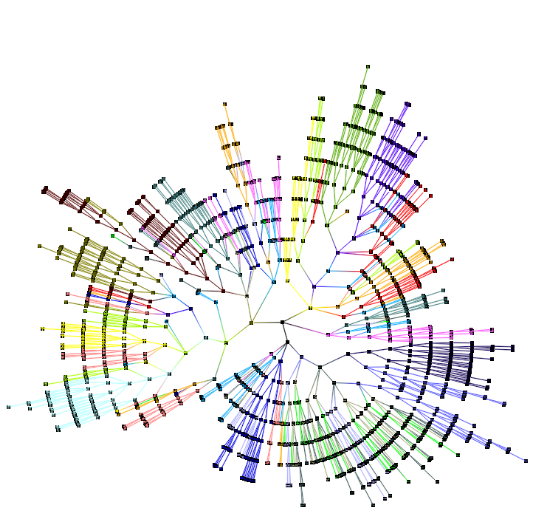
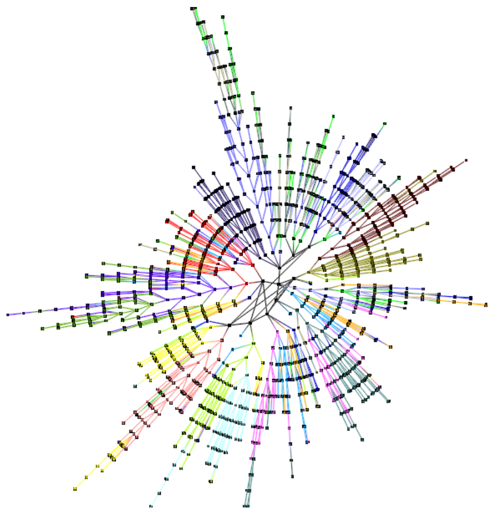


Figure 4.4: Percentage of new assignments during the training process for all chosen datasets


(a) Tree structure for a 2-tree network after majority vote (NMI = 0.356; Acc = 0.146; Rand = 0.685)


(b) Tree structure for a 10-tree network after majority vote (NMI = 0.559; Acc = 0.368; Rand = 0.895)

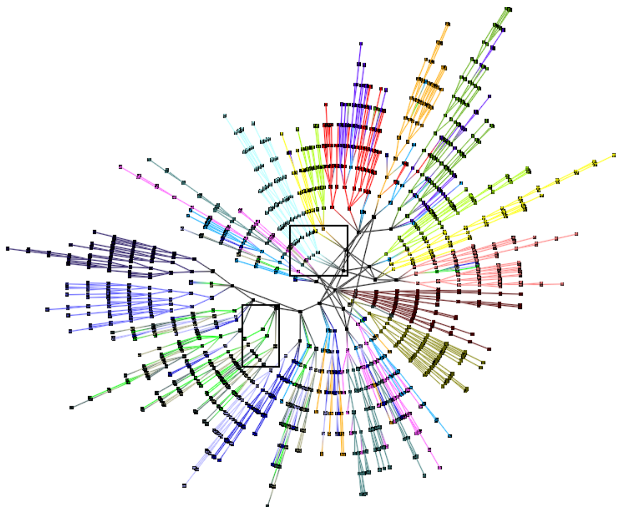
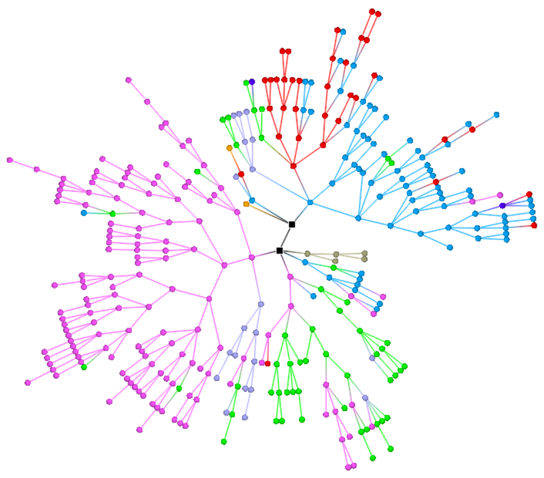
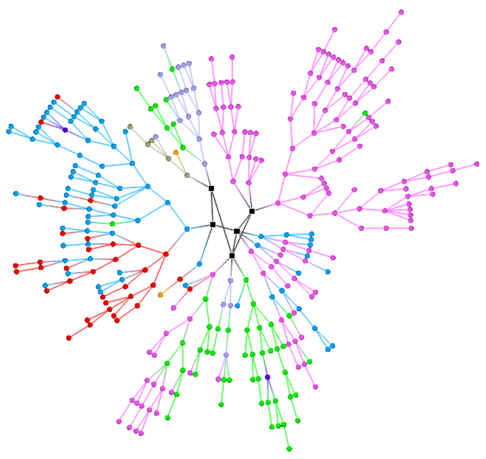

(c) Tree structure for a 20-tree network after majority vote (NMI = 0.735; Acc = 0.612; Rand = 0.945)

Figure 4.5: COIL-20: Data structure presented using the GSoT algorithm. Each color represents one real input data class.


(a) Tree structure for a 2-tree network (NMI = 0.421; Acc = 0.616; Rand = 0.674)


(b) Tree structure for a 5-tree network (NMI = 0.550; Acc = 0.726; Rand = 0.821)

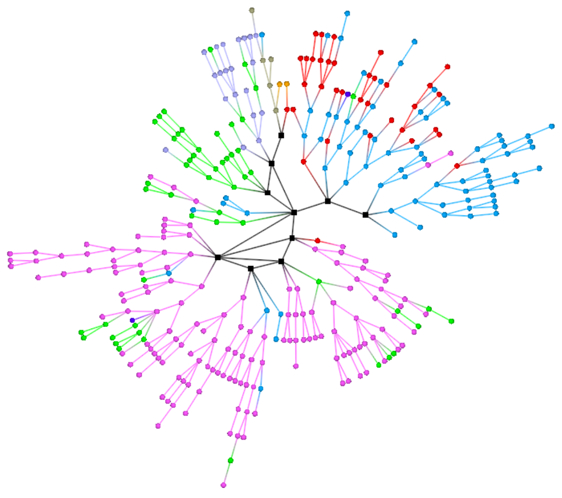
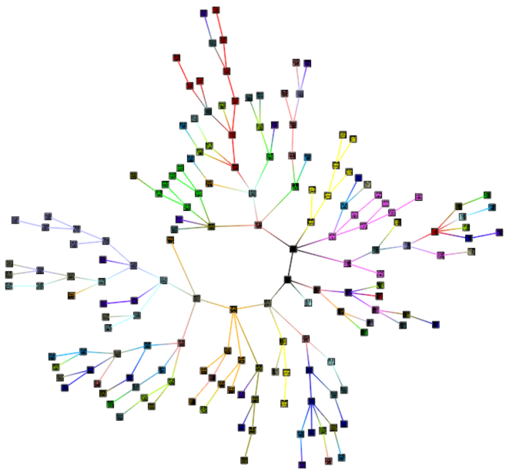
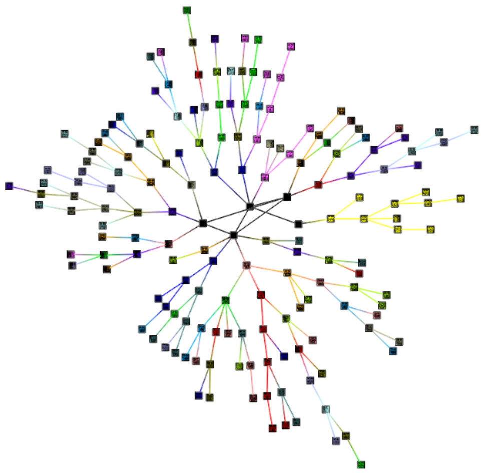

(c) Tree structure for a 10-tree network (NMI = 0.637; Acc = 0.794; Rand = 0.869)

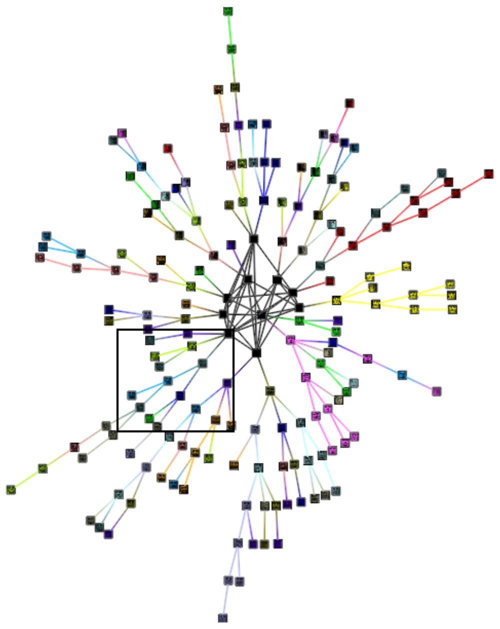
Figure 4.6: Ecoli: Data structure presented using the GSoT algorithm. Each color represents one real input data class.



(a) Tree structure for a 2-tree network (NMI = 0.114; Acc = 0.121; Rand = 0.513)



(b) Tree structure for a 5-tree network (NMI = 0.270; Acc = 0.236; Rand = 0.736)



(c) Tree structure for a 10-tree network (NMI = 0.387; Acc = 0.315; Rand = 0.850)

Figure 4.7: Yale: Data structure presented using the GSoT algorithm. Each color represents one real input data class.

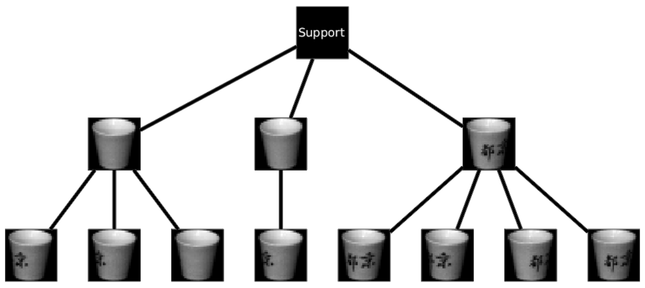
4.3.3 Visual validation

This experimental phase shows how the proposed method provides more information than other clustering approaches. Its main advantage is that it provides simultaneous hierarchical and topological structure. This feature simplifies data exploration by offering friendly and interactive visualization. We use Tulip [Auber 2003] as the framework to visualize and analyze a network. Figures 4.5, 4.6 and 4.7 show the evolution of the Ecoli, COIL-20 and Yale data structures, respectively, provided by GSoT in successive epochs. In the new organization, data are more visible and easier to visualize and analyze. The supports (black squares) located in the center are entoured by trees. Each support is associated with its respective tree. We provide the multi-level structure given by GSoT when the network size is {2, 10, 20} for COIL-20 and {2, 5, 10} for Ecoli and Yale. Each time, a view displays unique colors for different input classes.

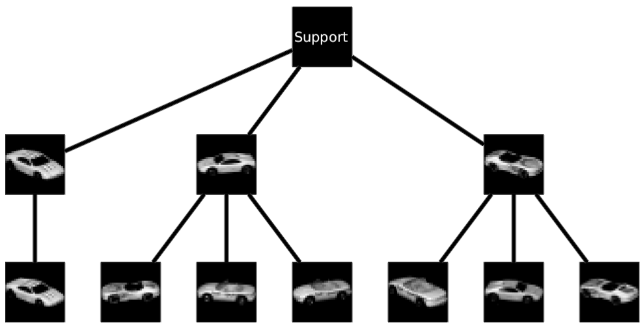
Figure 4.5 shows several pure clusters, including the celeste and brown classes. We zoom into two samples extracted from Figure 4.5(c) to study similarity between images (data) in hierarchical sub-trees. We select two sub-tree levels, as in Figure 4.8: one from the celeste cluster (the top square) and another from a mixed cluster (the bottom square). In the first zoom (Figure 4.8(a)), GSoT has good classification. The same object type (a cup) with visible symbols is grouped together. The second zoom (Figure 4.8(b)) shows images of different objects in the same cluster. The geometric shapes are similar e.g. in the form of cars. In these sub-trees, cars have the same direction as their parent node.

For the Ecoli dataset, we have 336 data points grouped into eight clusters. There are two clusters with only two data types (purple and orange), which are often misclassified using a majority vote. When a network has two or five trees (Figure 4.6(a) and 4.6(b)), GSoT cannot detect all input classes. Otherwise, the network size increases and is superior to the number of input classes. Figure 4.6(c) contains large clusters.

Figure 4.7 shows a visual sample from the Yale dataset, in which some data are misclassified due to the size of this dataset, which has 15 classes. However, the results improve after each epoch. One pure yellow tree is generated and found in Figure 4.7(c). Figure 4.9 shows a zoom to analyze information lying in sub-tree images. This zoom displays the three first levels of a sub-tree. The images in the sub-tree have one similar feature to their parent node. In the left sub-tree, the two images are from two different classes, but both show a face with glasses. In the middle sub-tree, the face shows either the eyes or mouth opened. The same case is found for the right sub-tree, where men have one or two eyes closed. The same analysis could be performed using the rest of the dataset.



(a) First zoom sample extracted from the top center of Figure 4.5(c): images belong to one class



(b) Second zoom sample extracted from the left of Figure 4.5(c): images belong to three different classes

Figure 4.8: Zoom views of COIL-20

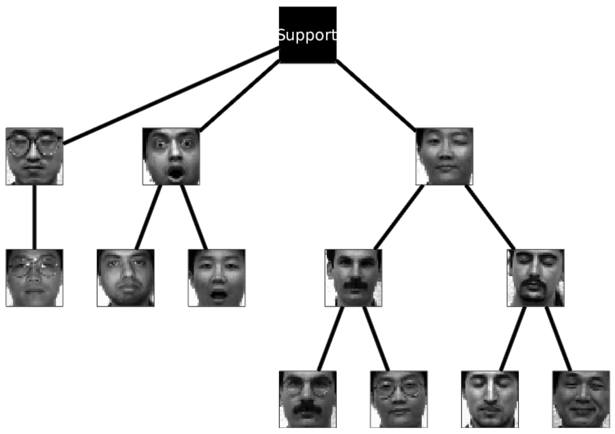


Figure 4.9: Zoom sample view of Yale extracted from the top center of Figure 4.7(c)

4.4 Conclusion

This chapter presented the second model Growing Self-Organizing Trees to address both clustering and visualization. The proposed method provides a topological and hierarchical model based on growing process and autonomous tree-building technique. Furthermore, it is able to continue learning, evolving the topological structure as well as hierarchical one until a criterion has been met. The GSoT network is more dynamic and flexible due to its evolutive network.

The tree structure allows the user to understand and analyze large amounts of data in an explorative manner. GSoT attempts to improve the standard visualization by offering topological and hierarchical structure. Data are pinpointed using this approach and can be further studied. Furthermore, we can directly visualize data in hierarchical and topological trees. In practice, analysts can apply our multi-structure to data indexation and exploration (e.g. a search engine).

The GSoT algorithm requires no *a priori* information about the input data, including an estimate of the number of clusters or cluster shapes. Moreover, the advantage of using GSoT is to accelerate the learning processing and to reduce the number of assignments epoch by epoch. The GSoT efficiency is demonstrated by several experiments on real-world datasets. The experimental results are promising, and the proposed clustering methodology produces high classification success.

Chapter 5

Hierarchical Laplacian Score for feature selection

Contents

5.1 Introduction

This chapter is devoted to the problem of unsupervised feature selection. Feature selection is an important step in data pre-processing. The feature selection problem has been addressed by the statistics and machine learning communities for many years. Feature selection is particularly a problem to deal with the objects e.g., images and texts which are represented as points in high dimensional space [Saeys 2007]. High dimensionality significantly increases the time and space requirements for training data, moreover learning tasks such as clustering or classification are analytically or computationally manageable in low dimensional spaces. Feature selection [Guyon 2006] is a task which focuses mainly on selecting the small subset of relevant features. This reduction in dimensions seeks to represent better the input dataset. There are many potential benefits of feature selection such as:

- defying the curse of dimensionality to improve the performance of machine learning algorithms,

- reducing in dimensions to facilitate data visualization,
- reducing the computational costs and speeding up the learning process.

The main contributions are as following:

1. The main interest of using the tree structure is to automatically discover local data structure and local data neighbors. AntTree is able to determine nearest neighbors for each data object, thus the parameter k is no longer necessary.

2. We propose to constrain the Laplacian Score (LS) using unsupervised hierarchical constraints. The idea is to discover a natural local structure and constraints within data without knowledge of any class labels. Our purpose is to assess the ability of features in preserving the local geometric structure offered by unlabeled data, while respecting the unsupervised constraints.

Due to the low cost and easy implementation, filters are more interesting for researchers. In litterature, LS as a filter method, is well-known because of its topology preservation. LS tries to discover feature properties given by the nearest neighbor graph, k-NN. In pratice, the hierarchical tree provided by AntTree can be considered as a neighbor graph. Therefore, it may be integrated into the Laplacian score [He 2005] as hierarchical constraints. We propose, in this chapter, a new score named Hierarchical Laplacian Score (HLS) that constrains the Laplacian Score.

5.2 Hierarchical Laplacian Score

5.2.1 Tree topology structure as nearest neighbor graph

Many feature selection approaches which have been proposed [Roweis 2000, Tenenbaum 2000, Belkin 2001, Zhao 2007, Zhao 2013] require a nearest neighbor graph to model the local geometric structure and perform the feature selection task. The general method to build a nearest neighbor graph is generally k-NN.

Based on this idea, we are interested in the tree structure of AntTree. As we mentioned in the two previous chapters, AntTree build a shortest path within the input data. This is enough to be considered as a nearest neighbor graph. Comparing to k-NN, this algorithm takes an advantage in the complexity ($\theta(N \log N)$) and runs automatically without any initial parameter.

In this structure, the objects belonging to a sub-tree are close to each other. Instead an object must have equally k neighbors, each object $\mathbf{x}_i$ (a tree node) will automatically obtain its k_i neighbors in the proposed graph. These k_i neighbors represent the objects that are directly connected to $\mathbf{x}_i$. For example in Figure 5.1, $\mathbf{x}_1$

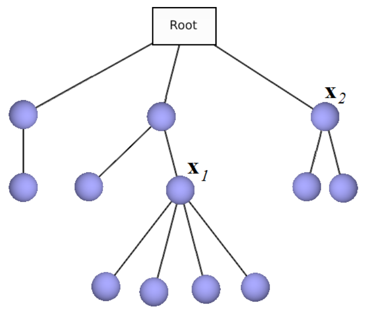


Figure 5.1: An example of tree topology structure.

has $k_1 = 5$ (one with the higher level and four with the lower level) and $\mathbf{x}_2$ has only $k_2 = 2$ (two with the lower level). Here the direct links connected to the root are not counted because the root is not an object and does not contain any data information. The way to build the tree topology structure is shown in Algorithm 10 in Chapter 3.

5.2.2 Hierarchical constraints for unsupervised feature selection

Studying the proposed hierarchical structure, we propose to constraint LS while preserving local structure ability. This hierarchical information is certainly necessary for feature selection, especially for unlabeled data. Here we want to define SL (Strong Link) and WL (Weak Link) as following:

- **Strong Link constraint (SL)**: involving $\mathbf{x}_i$ and $\mathbf{x}_j$, specifies that they are neighbors.

$$SL_{ij} = 1 \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are neighbors.}$$

- **Weak Link constraint (WL)**: involving $\mathbf{x}_i$ and $\mathbf{x}_j$, specifies that they are NOT neighbors.

$$WL_{ij} = 1 \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are not neighbors.}$$

With the unsupervised constraints defined above, a new score named HLS is to be minimized as in Equation 5.1.

$$HLS_r = \frac{\sum_{i,j}(f_{r_i} - f_{r_j})^2 S_{ij}}{\sum_i (f_{r_i} - \alpha_r^i)^2 D_{ii}} \tag{5.1}$$

where

$$S_{ij} = \begin{cases} e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\lambda}} & \text{if } \forall \mathbf{x}_i \text{ and } \mathbf{x}_j, SL_{ij} = 1 \\ 0 & \text{if } \forall \mathbf{x}_i \text{ and } \mathbf{x}_j, WL_{ij} = 1 \end{cases} \tag{5.2}$$

and

$$\alpha_r^i = \frac{1}{N - k_i} \sum_{j, WL_{ij}=1} f_{rj} \tag{5.3}$$

LS selects features with large variance which have more representative power and respect the pre-defined graph. Indeed, LS tries to find relevant features while maximizing the variance of data features for both similar and dissimilar objects. It would be important to note that since data objects are not close, their features should be significantly distinct. Our purpose is to maximize the local feature variance at each graph node. By not considering its neighbors (similar data), we maximize the variance of only dissimilar objects. While computing this variance, the matrix SL and WL model the local density of the data objects

HLS is similar to LS by developping the algebraic steps:

$$\sum_{i,j} (f_{r_i} - f_{r_j})^2 S_{ij} = 2\mathbf{f}_r{}^T L \mathbf{f}_r \tag{5.4}$$

We develop the expression in 5.3 as following:

$$\alpha_r^i = \frac{1}{N - k_i} \sum_{j, WL_{ij}=1} f_{rj}$$

$$\alpha_r^i = \frac{1}{N - k_i} \mathbf{f}_r^T WL\mathbf{1}$$

We define $\widetilde{\mathbf{f}'_r}$ as following

$$\widetilde{\mathbf{f}'_r} = \mathbf{f}_r - \frac{1}{N - k_i} \mathbf{f}_r^T WL\mathbf{1}$$

Thus,

$$\sum_i (f_{r_i} - \alpha_r^i)^2 D_{ii} = \widetilde{\mathbf{f}'_r}^T D \widetilde{\mathbf{f}'_r} \tag{5.5}$$

We re-define HLS as the below expression

$$HLS_r = \frac{\mathbf{f}_r{}^T L \mathbf{f}_r}{\widetilde{\mathbf{f}'_r}^T D \widetilde{\mathbf{f}'_r}} \tag{5.6}$$

Algorithm 14 presents the three steps for feature selection and ranking using HLS.

Lemma. Algorithm 14 is computed in time $\theta(max(N \log N, d \times max(\log d, N^2)))$

Proof. The first step of the algorithm has a complexity $\theta(N \log N)$. The second step computes the scores for d features requiring dN^2 operations. The last step ranks features according to their scores requiring $d \log(d)$ operations.

Algorithm 14 HLS

```
Input: Data set $\mathcal{X}$
1: Build a tree topology structure as in Algorithm 7 and generate $SL$ and $WL$ matrices
2: for $r = 1$ to d
       compute $HLS_r$ score for each feature $r$ according to Equation 5.6
   end
3: Rank the features $r$ according to their score in ascending order
```

5.3 Experiments

In this section, several experiments were performed on several real-world data set. These experiments are divided into two categories: clustering application and nearest neighbor classification. In order to compare the performance, we use the following two feature selection algorithms:

- Laplacian Score, which selects the variables that can preserve the local structure.
- MaxVariance, which selects the variables that maximize variances in order to reflect representative power.

5.3.1 Datasets and methods

Data set	Size	# Features	# Classes
ARP10P	130	2400	10
COIL-20	1440	1024	20
Isolet	1559	617	26
Sonar	208	60	2
Soybean	47	35	4

Table 5.1: Data features

We run the experiments on several datasets which are collected from different repositories. The features of these databases are summarized in Table 5.3.1 and their description is as following:

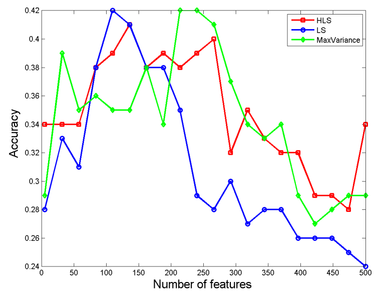
- AR10P can be found in `http://featureselection.asu.edu/datasets.php`. This database contains face images of 10 different persons with 13 images per person.
- COIL-20 [Cai 2011] consists of a set which contains images of 20 different objects with 72 images per object.

- Isolet [Cai 2011] contains 150 subjects who spoke the name of each letter of the alphabet twice. They are grouped into sets of 30 subjects each, and are referred to as Isolet1 through Isolet5. Isolet5 is selected in our experiment.
- Sonar and Soybean are the two famous real-world data sets that can be downloaded from UCI Machine Learning Repository.

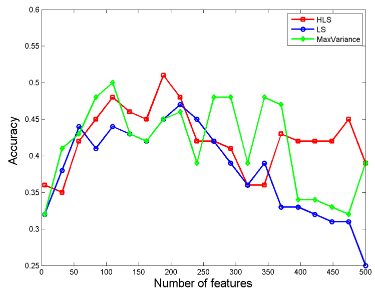
5.3.2 Clustering results

	Accuracy			NMI		
K	10	15	20	10	15	20
HLS	**0.376**	**0.461**	**0.584**	**0.416**	**0.474**	**0.511**
LS	0.292	0.438	0.576	0.274	0.374	0.450
MaxVariance	0.407	0.492	0.538	0.384	0.410	0.426
All Features	0.184	0.307	0.400	0.133	0.268	0.371

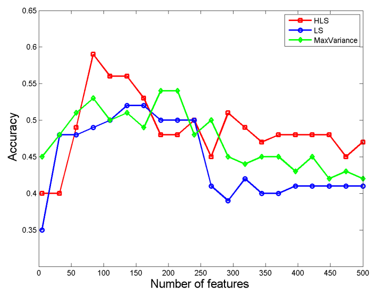
Table 5.2: Clustering quality by using 250 selected features on the AR10P data set.The last row shows the performance by using all the 2400 features.



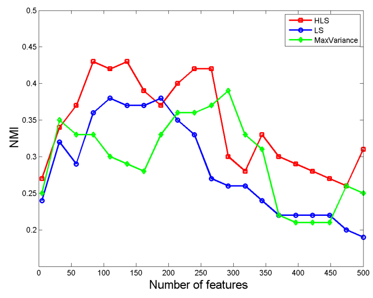
(a) 10 Clusters



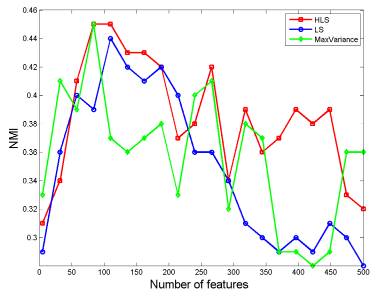
(b) 15 Clusters



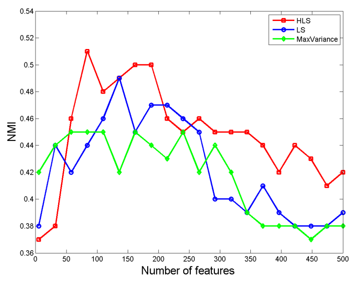
(c) 20 Clusters



(d) 10 Clusters



(e) 15 Clusters

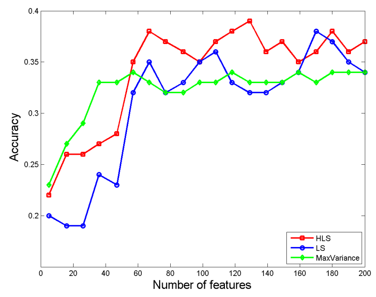


(f) 20 Clusters

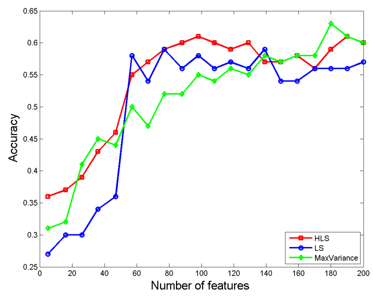
Figure 5.2: Clustering performance (Accuracy and NMI) vs. the number of selected features on AR10P database

	Accuracy			NMI		
K	10	20	30	10	20	30
HLS	**0.346**	**0.608**	**0.722**	**0.425**	**0.680**	**0.695**
LS	0.348	0.576	0.683	0.390	0.628	0.640
MaxVariance	0.333	0.548	0.640	0.382	0.587	0.645
All Features	0.361	0.646	0.761	0.573	0.660	0.781

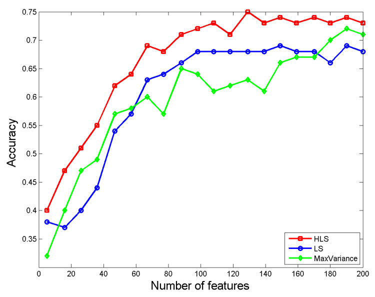
Table 5.3: Clustering quality by using 100 selected features on the COIL-20 data set.The last row shows the performance by using all the 1024 features.



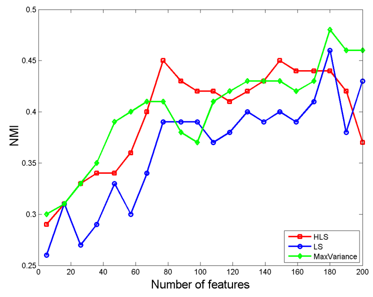
(a) 10 Clusters



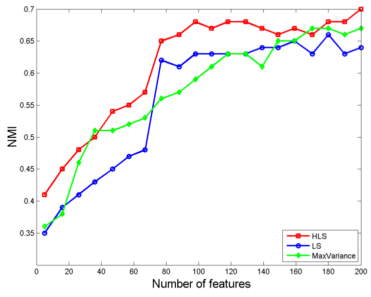
(b) 20 Clusters



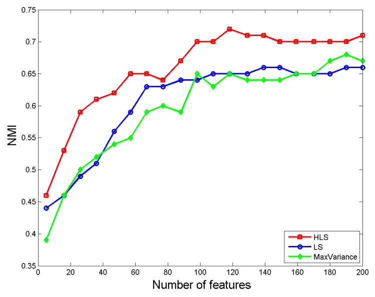
(c) 30 Clusters



(d) 10 Clusters



(e) 20 Clusters



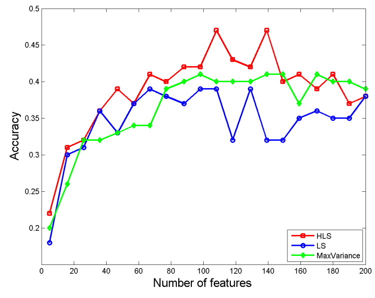
(f) 30 Clusters

Figure 5.3: Clustering performance (Accuracy and NMI) vs. the number of selected features on COIL-20 database

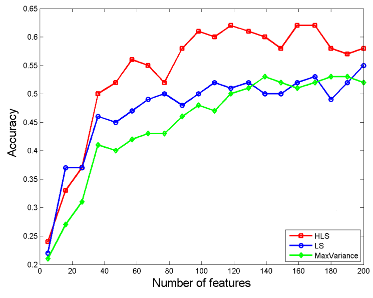
To facilitate the comparison between different methods, we perform K-means clustering by using the selected features. In these experiments, we fix $k = 5$ to build k-NN graph for Laplacian Score. For the proposed score, we construct tree structure using Algorithm 7. To evaluate the clustering quality with different number of clusters, we adopt this parameter s as following $K = 10$, 15, 20 on AR10P, $K = 10$, 20, 30 on COIL-20, $K = 13$, 26, 39 on Isolet and $K = 4$, 6, 8 on Sonar and Soybean. For each test of K, we conduct different algorithms to select a number of features m ($m = 1, ..., 500$ on AR10P, $m = 1, ..., 200$ on COIL-20 and Isolet, $m = 1, ..., 60$ on Sonar and $m = 1, ..., 35$ on Soybean) and apply K-means for clustering. The K-means

	Accuracy			NMI		
K	13	26	39	13	26	39
HLS	**0.423**	**0.606**	**0.626**	**0.726**	**0.702**	**0.675**
LS	0.385	0.499	0.559	0.606	0.646	0.610
MaxVariance	0.405	0.479	0.510	0.704	0.629	0.598
All Features	0.406	0.532	0.585	0.596	0.701	0.672

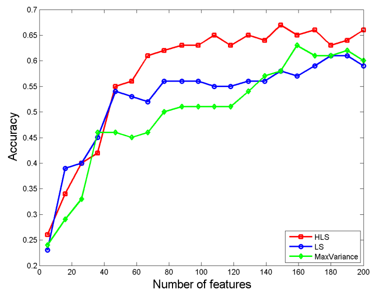
Table 5.4: Clustering quality by using 100 selected features on the Isolet data set. The last row shows the performance by using all the 617 features.



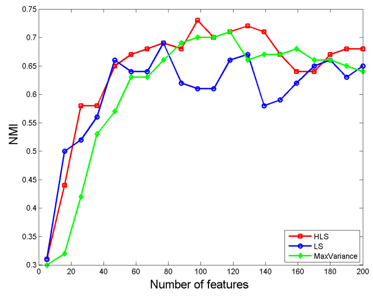
(a) 13 Clusters



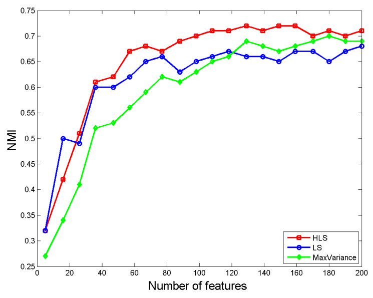
(b) 26 Clusters



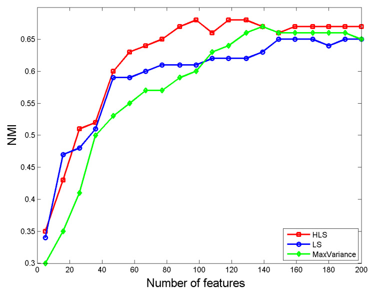
(c) 39 Clusters



(d) 13 Clusters



(e) 26 Clusters



(f) 39 Clusters

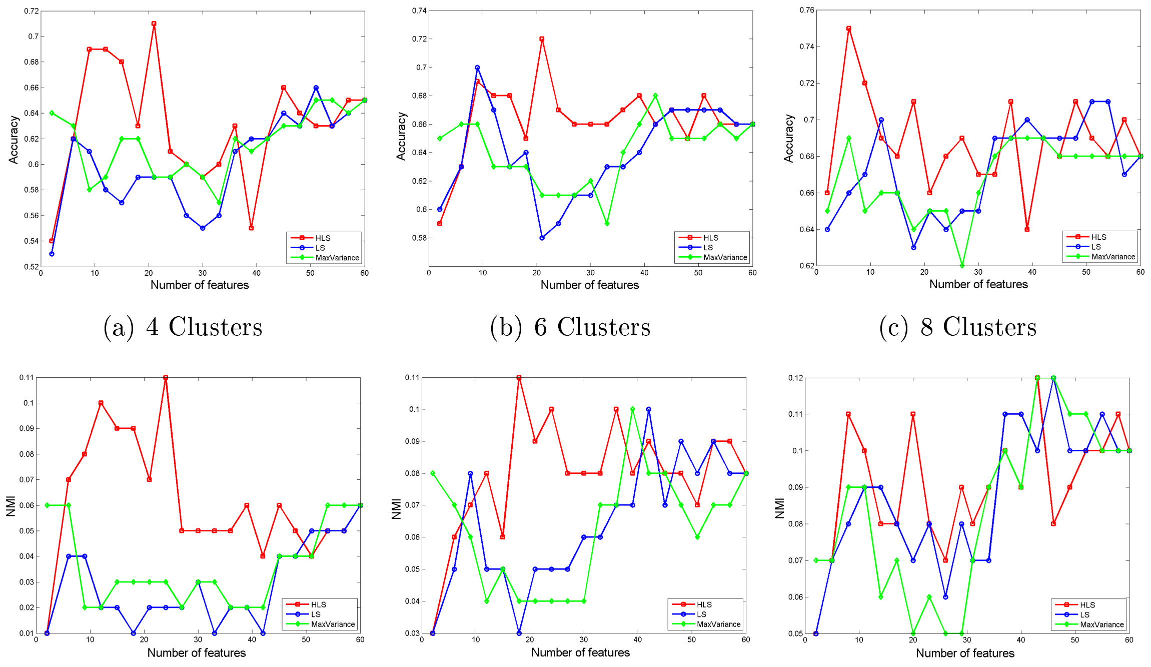
Figure 5.4: Clustering performance (Accuracy and NMI) vs. the number of selected features on Isolet database

algorithm are applied 10 times with different random initialization.

Figure 5.2 - 5.6 display the plots of clustering performance (Accuracy and NMI) versus the number of selected features on the datasets: AR10P, COIL-20, Isolet, Sonar and Soybean, respectively. Generally HLS consistently outperforms all the other methods. In Figure 5.2, HLS algorithm provides reasonably good results in Accuracy and NMI criteria. On the COIL-20 dataset (Figure 5.3), HLS gives the best results in Accuracy and NMI with typically around 50-100 features for all three tests. We note that, for $K = 30$, our proposed algorithm are clearly better than the others. On the other hand, a study on Figure 5.4 reveals that the accuracy of HLS increases

	Accuracy			NMI		
K	4	6	8	4	6	8
HLS	**0.663**	**0.716**	**0.759**	**0.067**	**0.078**	**0.109**
LS	0.596	0.634	0.677	0.015	0.040	0.083
MaxVariance	0.620	0.615	0.620	0.031	0.046	0.049
All Features	0.649	0.658	0.682	0.055	0.080	0.097

Table 5.5: Clustering quality by using 15 selected features on the Sonar data set. The last row shows the performance by using all the 60 features.



(a) 4 Clusters (b) 6 Clusters (c) 8 Clusters

(d) 4 Clusters (e) 6 Clusters (f) 8 Clusters

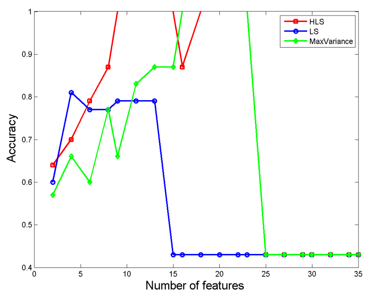
Figure 5.5: Clustering performance (Accuracy and NMI) vs. the number of selected features on Sonar database

steadily. As we have been seen in COIL-20 data set, HLS dominates over LS and MaxVariance with $K = 30$. In Figure 5.5, we can note that our values are better than the others, even better than the quality values by using all the features. Figure 5.6 presents the clustering results for Soybean dataset. Remarkably, HLS achieves Acc $= 100\%$ in most of cases $K = 4, 6$ and 8 by using only 9 features.

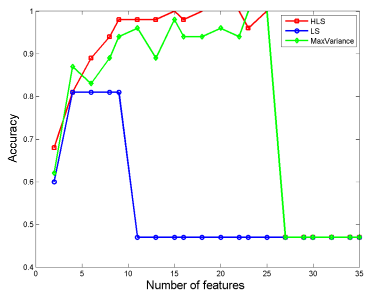
In Table 5.2 - 5.6, we summarize the clustering results of AR10P, COIL-20, Isolet, Sonar and Soybean respectively. The numerical results of AR10P presented in Table 5.2 show us an improvement in NMI performance provided by HLS. For Accuracy, HLS is comparable with the two others LS and MaxVariance. In Table 5.3 for COIL-

	Accuracy			NMI		
K	4	6	8	4	6	8
HLS	**1**	**0.978**	**0.978**	**0.979**	**0.696**	**0.632**
LS	0.787	0.766	0.851	0.720	0.516	0.500
MaxVariance	0.829	0.957	0.936	0.752	0.663	0.577
All Features	0.425	0.468	0.510	0.079	0.196	0.303

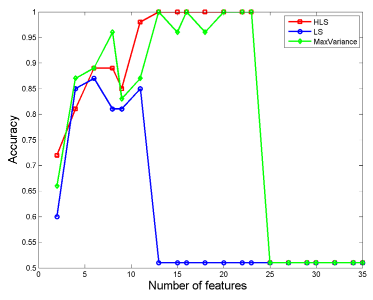
Table 5.6: Clustering quality by using 9 selected features on the Soybean data set. The last row shows the performance by using all the 35 features.



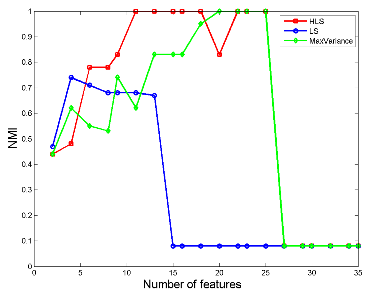
(a) 4 Clusters



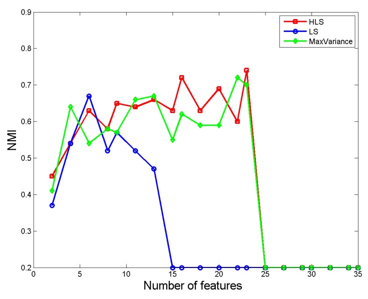
(b) 6 Clusters



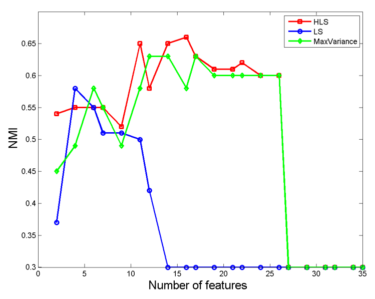
(c) 8 Clusters



(d) 4 Clusters



(e) 6 Clusters



(f) 8 Clusters

Figure 5.6: Clustering performance (Accuracy and NMI) vs. the number of selected features on Soybean database

20 dataset, we remark that in 20 clusters case and only 100 features are selected, the NMI value given by HLS is 68.0%, which is even better than the clustering result by using all the 1024 features (66.0%). For Isolet (Table 5.4), even though the Accuracy and NMI results with 100 features are not the best but they are very comparable with the ones with 617 features. In Table 5.5, it can be noted that our methods perform slightly well on Sonar dataset in Accuracy. We can achieve Accuracy value Acc $=$ 68.2% by using 9 features with $K = 6$ or 8. Finally, on Soybean dataset, HLS is the only method which can achieve good results in both Accuracy and NMI.

5.3.3 Classification results with Nearest Neighbor

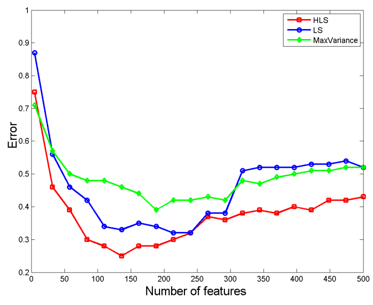
For the supervised learning, we use the nearest neighbor classifier to compare the performance of the selected algorithms. We perform leave-one-out cross validation as follows: For each data point $\mathbf{x}_i$, we find the nearest neighbor $NN(\mathbf{x}_i)$.

$$NN(\mathbf{x}_i) = argmin_{\forall j=1,..,N}(\mathbf{x}_i - \mathbf{x}_j)^2$$

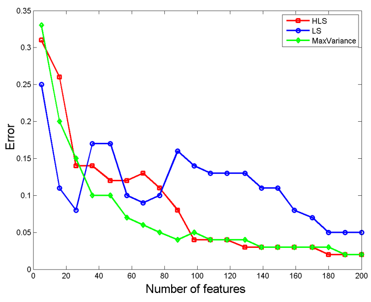
Let y'_i the class label of $\mathbf{x}_{NN(\mathbf{x}_i)}$ ($y'_i = NN(y_i)$). The classification error is thus defined as following:

$$Error = 1 - \frac{1}{N}\sum_{i=1}^{N}\delta(y'_i, y_i)$$

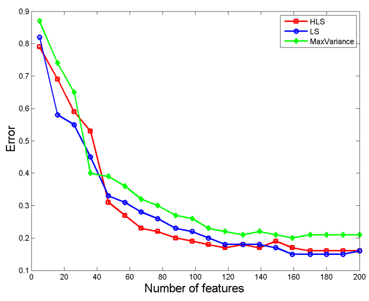
where $\delta(y'_i, y_i) = 1$ if $y'_i = y_i$ and 0 otherwise.



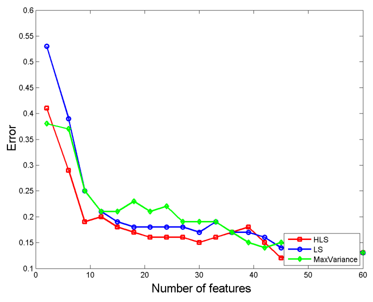
(a) AR10P



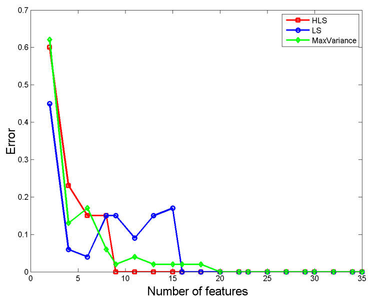
(b) COIL-20



(c) Isolet



(d) Sonar



(e) Soybean

Figure 5.7: Classification error vs. the number of selected features

Figure 5.7 shows the classification error of the five selected databases. In general, the classification errors of HLS are better than the errors of LS and MaxVariance. On AR10P set, we obtain a good classification by using only around 125 features (Error = 0.186). It is interesting to note on COIL-20 dataset, HLS can achieve the classification error = 0.043 by using only 100 features. On Isolet dataset, HLS starts converging around 50 features. The HLS result is slightly better than LS and MaxVariance on Sonar dataset. Particularly on Soybean dataset, HLS performs well

	AR10P	COIL-20	Isolet	Sonar	Soybean
# Features	250	100	100	15	9
HLS	**0.215**	**0.043**	**0.192**	**0.182**	**0**
LS	0.330	0.143	0.221	0.192	0.148
MaxVariance	0.415	0.045	0.258	0.256	0.02
All Features	0.500	0	0.124	0.125	0

Table 5.7: Classification error by using a number of features. The last row shows the error rate by using all the features.

and the classification error reaches 0 very fast with no more than 9 features. Table 5.7 presents the nearest neighbor classification error with different features for each dataset. HLS provides comparable results with that using all the features.

5.4 Conclusion

In general, it is significantly more challenging than the conventional setting of feature selection because of the lack of labeled training dataset. To address this challenge, we proposed a filter approach for unsupervised feature selection. The new score was proposed based on Laplacian Score to evaluate the relevance of features based on tree topology structure and unsupervised constraints. Our algorithm is essentially autonomous without using pre-defined parameters. Finally, experimental results on several datasets prove that HLS algorithm achieves significantly higher performance for clustering and classification.

As future work, it would be interesting to investigate hierarchical constraints for more efficient semi-supervised feature selection. Another direction for semi-supervised feature selection is to iteratively propagate labels from labeled data to unlabeled data while carrying out hierarchical constraints for feature selection. One may also extend the current algorithm to handle different features in the case of biological datasets in the aim to clustering gene expression datasets and selecting relevant genes.

Chapter 6

Conclusions

In this thesis we have mainly investigated the clustering problem. We tried to answer the following questions:

- How can data be grouped in clusters based on their similarity?
- What should be the basis for a decision to assign an object to a cluster?
- What should be the objective function to minimize the quantization error?
- How can a clustering result be evaluated when class labels are not available?

The clustering problem appears when in a classification problem labels in the data are not available. In this case only the similarity of the data might be known, clusters can be found with high confidence in some senses (see Chapter 2, 3 and 4). Several approaches proposed to find clusters according to data hierarchy or topology or even both.

A complication emerges when a clustering method is evaluated. When there is no known class labels, only the intra-variance or intra-density can be used. Unfortunately, these types of measure are under the influence of the number of clusters which must be defined a priori by a user. To overcome this small issue, the class labels become indispensable. In practice, the user has another option to evaluate clustering methods through data visualization. Clustering and visualization are both big challenges.

6.1 What has been done in this thesis?

In Chapter 3 we presented SoT, the derivation of the Self-Organizing Maps, which added a hierarchical dimension into the topological map. Instead of a hierarchical map, it builds automatically a hierarchical tree for the data group in each network node. In SoT, the map is *fixed* and has to be initialized with certain input parameters like in SOM. GSoT in Chapter 4 is extended Growing Neural Gas using the same principle. It appears that combining a topological model and a hierarchical model brings more advantages. These methods are, therefore, suited for data clustering and visualization mentioned above. The self-assembly rules that are used to build the hierarchical structure allow to correct the mis-classification of the previous iterations.

Furthermore, we profit the hierarchical properties to accelerate the learning process, and to reduce the number of assignment in each learning step.

Due to the hierarchical and topological structure, SoT and GSoT offer flexible and friendly way to visualize data. The topological map enables to visualize data from high-dimensional space to low-dimensional space; while the trees allow to visualize the data relations in any hierarchical level. Moreover, these relations can be considered as the shortest paths that connect all the data in a cluster. These properties have not seen in any clustering methods presented in Chapter 2. The hierarchical and topological structure seems to be interesting in the case of graph datasets where it exists links between a pair of graph vertices. The links that are found in the proposed structure can lead to insight information.

We also discussed the performance of SoT and GSoT on several real-world datasets e.g. vectors, images, proteins or graphs. The applicability of these methods for all types of data was measured through the experiments in Chapter 3 and 4. We have performed the evaluations in term of the clustering quality as well as the data visualization.

With the interest of data visualization in a low-dimensional space, the feature selection was studied in Chapter 5. Here we defined the hierarchical constraints implemented into a filter score for feature selection. This is the task of data pre-processing (see Chapter 2) that helps improving the algorithm performance. In this approach, we preserved locally the variance for each feature and the data topology.

6.2 What can be concluded from this thesis?

Some conclusions can be drawn from this thesis:

- First, the hybrid models combining the hierarchical and topological models are a good tool for visualizing data, especically high-dimenstional data. They enables both hierarchical and topological relations of input data.

- The properties of AntTree can be widely studied and applied in many problems. In this thesis, we have shown how to use these properties to deal with the clustering problem: combining with the topological models for data visualization and accelerating the learning process.

- Both SoT and GSoT work well on real-world datasets. Their competitive performances are acceptable comparing to other methods.

- Finally, the hierarchical constraints provided by the tree topology contain useful information for feature selection. These constraints are used to preserve the local structure properties.

6.3 What can be done in the future?

This thesis offers several perspectives for future work:

- In the short term, the problems encountered in data visualization can be intensively studied in the hierarchical and topological structure. There are also a number of potential avenues for future research, such questions still can be posed:

  - Is the hierarchical and topological structure useful?
  - Do all the synthetized and original links in the hierarchical and topological structure contain any helpful information?

  Indeed, these questions are very attractive in the problem of graph summarization, thus it requires throughly user-validation studies.

- GSoT has its own particulars as a growing model. This model can be extended to take into account the data that increment or evolve over time. It seems especially interesting to process data streaming. The purpose is to study how GSoT interferes and builds increasingly growing trees for this type of data.

- In the long term, another perspective work is that how to apply the hierarchical and topological models on big data? Big data often refer to a collection of datasets with a complex and large volume. It is hard to process using traditional methods, it should require new ways of processing to enhance decisions, discover insight information, etc. Heuristic learning algorithms should be signficantly improved to help in process optimization by reducing the implementation time and cost. To handle large data sets, one of newest paradigm is MapReduce which performs map and reduce operations. In map step, the input problem is divided into smaller sub-problems which are solved separately and independently. The reduce step collects the answers to all the sub-problems then forms the final answer to the input problem.

Appendix A

List of publications

1. Nhat-Quang Doan, Hanane Azzag, Mustapha Lebbah, Guillaume Santini, *Self-organizing Trees for visualizing protein dataset.* In Proc. of International Joint Conference on Neural Networks (IJCNN 2013), 2013.

2. Nhat-Quang Doan, Hanane Azzag, Mustapha Lebbah, *Sélection de variables non supervisée sous contraintes hiérarchiques*, In Proc. of Conférence Extraction et Gestion des Connaissances (EGC 2013), pages 67-78, 2013.

3. Nhat-Quang Doan, Hanane Azzag and Mustapha Lebbah, *Growing Self-organizing Trees for autonomous hierarchical clustering.* Neural Network Special Issue: Autonomous Learning, Volume 41, Pages 85-95, 2013.

4. Nhat-Quang Doan, Hanane Azzag and Mustapha Lebbah, *Self-organizing map and tree topology for graph summarization.* International Conference on Artificial Neural Networks (ICANN 2012), pages 363-370, 2012.

5. Nhat-Quang Doan, Hanane Azzag and Mustapha Lebbah, *Graph Decomposition using Self-Organizing Trees.* 16th International Conference Information VIsualisation (iV 2012), pages 246-251, 2012.

6. Nhat-Quang Doan, Hanane Azzag and Mustapha Lebbah, *Growing Self-organizing Trees for Knowledge Discover from Data.* In Proc. International Joint Conference on Neural Networks (IJCNN 2012), page 251-258, 2012.

7. Nhat-Quang Doan, Hanane Azzag and Mustapha Lebbah, *Clustering multi-niveaux de graphes : hiérarchique et topologique.* In Proc. of Conférence Extraction et Gestion des Connaissances EGC'12 (RNTI), pages 567-569, 2011.

Bibliography

[Adamic 2005] Lada Adamic et Natalie Glance. *The Political Blogosphere and the 2004 U.S. Election: Divided They Blog.* In In LinkKDD: Proceedings of the 3rd international workshop on Link discovery, pages 36–43, 2005. (Cited on page 69.)

[Adleman 1994] L. M. Adleman. *Molecular computation of solutions to combinatorial problems.* Science, vol. 266, no. 5187, pages 1021–1024, November 1994. (Cited on page 42.)

[Amos 2005] Martyn Amos. Theoretical and experimental dna computation, volume XIII, 173 pages of *Natural Computing Series.* Springer Verlag Berlin, Heidelberg, 2005. (Cited on page 42.)

[Anderson 2002] C. Anderson, G. Theraulaz et J. L. Deneubourg. *Self-assemblages in insect societies.* Insectes Sociaux, vol. 49, pages 99–110, 2002. (Cited on page 42.)

[Andreeva 2008] A. Andreeva, D. Howorth, J. Chandonia, S.E. Brenner, T.J.P. Hubbard, C. Chothia et A.G. Murzin. *Data growth and its impact on the SCOP database: new developments.* Nucleic Acids Res, vol. 36, no. Database issue, pages D419–25, 2008. (Cited on pages 76 and 77.)

[Angelov 2006] Stanislav Angelov, Sanjeev Khanna et Mirkó Visontai. *On the Complexity of Graph Self-assembly in Accretive Systems.* In Chengde Mao et Takashi Yokomori, editeurs, DNA, volume 4287 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2006. (Cited on page 42.)

[Auber 2003] D. Auber. *Tulip : A huge graph visualisation framework.* In P. Mutzel et M. Jünger, editeurs, Graph Drawing Softwares, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003. (Cited on pages 65, 71, 83 and 103.)

[Azzag 2007] Hanene Azzag, Gilles Venturini, Antoine Oliver et Christiane Guinot. *A hierarchical ant based clustering algorithm and its use in three real-world applications.* European Journal of Operational Research, vol. 179, no. 3, pages 906–922, June 2007. (Cited on pages 13, 14, 42, 43 and 61.)

[Bang-Jensen 2008] Jrgen Bang-Jensen et Gregory Z. Gutin. Digraphs: Theory, algorithms and applications. Springer Publishing Company, Incorporated, 2nd édition, 2008. (Cited on pages 12 and 19.)

[Belkin 2001] Mikhail Belkin et Partha Niyogi. *Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering.* In Advances in Neural Information Processing Systems 14, pages 585–591. MIT Press, 2001. (Cited on pages 19 and 108.)

[Bellman 2003] Richard Ernest Bellman. Dynamic programming. Dover Publications, Incorporated, 2003. (Cited on page 15.)

[Bernstein 1977] F.C. Bernstein, T.F. Koetzle, G.J. Williams, E.F.Jr Meyer, M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi et M. Tasumi. *The Protein Data Bank. A computer-based archival file for macromolecular structures.* Eur. J. Biochem., vol. 80, no. 2, pages 319–24, 1977. (Cited on page 76.)

[Bezdek 1995] J.C. Bezdek et N.R. Pal. *Cluster validation with generalized Dunn's indices.* In Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference on, pages 190–193, 1995. (Cited on pages 12 and 46.)

[Boulet 2008] Romain Boulet, BertGSoT Jouve, Fabrice Rossi et Nathalie Villa. *Batch kernel SOM and related Laplacian methods for social network analysis.* Neurocomputing, vol. 71, no. 7–9, pages 1257–1273, March 2008. (Cited on page 68.)

[Brenner 2000] Steven E. Brenner, Patrice Koehl et Michael Levitt. *The ASTRAL compendium for protein structure and sequence analysis.* Nucleic Acids Research, vol. 28, no. 1, pages 254–256, 2000. (Cited on page 77.)

[Cai 2007] Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han et Thomas Huang. *Learning a Spatially Smooth Subspace for Face Recognition.* In Proc. IEEE Conf. Computer Vision and Pattern Recognition Machine Learning (CVPR'07), 2007. (Cited on page 62.)

[Cai 2011] Deng Cai, Xiaofei He, Jiawei Han et Thomas S. Huang. *Graph Regularized Nonnegative Matrix Factorization for Data Representation.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 8, pages 1548–1560, 2011. (Cited on pages 111 and 112.)

[Camazine 2001] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz et E. Bonabeau. Self-organization in biological systems. Princeton University Press, 2001. (Cited on page 42.)

[Campos 2001] Marcos M. Campos et Gail A. Carpenter. *S-TREE: self-organizing trees for data clustering and online vector quantization.* Neural Netw., vol. 14, pages 505–525, May 2001. (Cited on pages 5, 37 and 39.)

[Cao 2003] Xiang Cao et Ponnuthurai N. Suganthan. *Video shot motion characterization based on hierarchical overlapped growing neural gas networks.* Multimedia Syst., vol. 9, no. 4, pages 378–385, 2003. (Cited on pages 5, 39 and 41.)

[Carpentier 2005] M. Carpentier, S. Brouillet et J. Pothier. *YAKUSA: a fast structural database scanning method.* Proteins, vol. 61, no. 1, pages 137–51, 2005. (Cited on page 78.)

[Chung 1997] Fan R. K. Chung. Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92). American Mathematical Society, Février 1997. (Cited on page 20.)

[Costa 2007] JosÃ© Alfredo F. Costa et Ricardo S. Oliveira. *Cluster Analysis using Growing Neural Gas and Graph Partitioning.* In IJCNN, pages 3051–3056. IEEE, 2007. (Cited on page 87.)

[Danos 2005] Vincent Danos et Fabien Tarissan. *Self Assembling Graphs.* In José Mira et José R. Álvarez, editeurs, IWINAC (1), volume 3561 of *Lecture Notes in Computer Science*, pages 498–507. Springer, 2005. (Cited on page 42.)

[Danos 2007] Vincent Danos et Fabien Tarissan. *Self-assembling graphs.* Natural Computing: an international journal, vol. 6, no. 3, pages 339–358, 2007. (Cited on page 42.)

[Davies 1979] David L. Davies et Donald W. Bouldin. *A Cluster Separation Measure.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. PAMI-1, no. 2, pages 224–227, 1979. (Cited on pages 12 and 45.)

[Dittenbach 2000] Michael Dittenbach, Dieter Merkl et Andreas Rauber. *The Growing Hierarchical Self-Organizing Map.* pages 15–19. IEEE Computer Society, 2000. (Cited on pages 5, 37 and 38.)

[Doherty 2005] Kevin Doherty, Rod Adams et Neil Davey. *TreeGNG - hierarchical topological clustering.* In ESANN, pages 19–24, 2005. (Cited on pages 5, 40, 41 and 51.)

[Elghazel 2009] Haytham Elghazel et Khalid Benabdeslem. *Towards B-Coloring of SOM.* In Proceedings of the 6th International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM ’09, pages 322–336, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 28.)

[Eppstein 1997] David Eppstein, Michael S. Paterson et Frances F. Yao. *On nearest neighbor graphs.* Discrete & Computational Geometry, vol. 17, no. 3, pages 263–282, April 1997. (Cited on page 23.)

[Estivill-Castro 2002] Vladimir Estivill-Castro. *Why so many clustering algorithms: a position paper.* SIGKDD Explor. Newsl., vol. 4, no. 1, pages 65–75, Juin 2002. (Cited on page 45.)

[Everitt 2009] Brian S. Everitt, Sabine Landau et Morven Leese. Cluster analysis. Wiley, 4th édition, Janvier 2009. (Cited on page 53.)

[Everitt 2011] Brian Everitt, Sabine Landau, Morven Leese et Daniel Stahl. Cluster analysis. Wiley, 5th édition, 2011. (Cited on pages 11, 24, 26, 34 and 45.)

[Famili 1997] A. Famili, W.-M. Shen, R. Weber et E. Simoudis. *Data Preprocessing and Intelligent Data Analysis.* Intelligent Data Analysis, vol. 1, pages 3–23(21), 1997. (Cited on page 18.)

[Färber 2010] I. Färber, S. Günnemann, H.P. Kriegel, P. Kröger, E. Müller, E. Schubert, T. Seidl et A. Zimek. *On using class-labels in evaluation of clusterings.* In MultiClust: 1st International Workshop on Discovering, Summarizing and Using Multiple Clusterings Held in Conjunction with KDD 2010, Washington, DC, 2010. (Cited on page 46.)

[Fortunato 2010] Santo Fortunato. *Community detection in graphs.* Physics Reports, vol. 486, 2010. (Cited on page 48.)

[Fritzke 1991] Bernd Fritzke. *Unsupervised Clustering With Growing Cell Structures.* In In Proceedings of the International Joint Conference on Neural Networks, pages 531–536. IEEE, 1991. (Cited on pages 30, 31, 90 and 92.)

[Fritzke 1995a] Bernd Fritzke. *Growing Grid - a self-organizing network with constant neighborhood range and adaptation strength.* Neural Processing Letters, vol. 2, pages 9–13, 1995. (Cited on page 32.)

[Fritzke 1995b] Bernd Fritzke. *A Growing Neural Gas Network Learns Topologies.* In Advances in Neural Information Processing Systems 7, pages 625–632. MIT Press, 1995. (Cited on pages 14, 32, 87 and 88.)

[Garzon 1999] Max H. Garzon, Russell J. Deaton et Ken Barnes. *On Self-Assembling Graphs in vitro.* In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela et Robert E. Smith, editeurs, Proceedings of the Genetic and Evolutionary Computation Conference, volume 2, pages 1805–1809, Orlando, Florida, USA, 13-17 Juillet 1999. Morgan Kaufmann. (Cited on page 42.)

[Girvan 2002] M. Girvan et M. E. J. Newman. *Community structure in social and biological networks.* Proceedings of the National Academy of Sciences, vol. 99, no. 12, pages 7821–7826, Juin 2002. (Cited on pages 68 and 69.)

[Goldreich 2008] Oded Goldreich. Computational complexity: A conceptual perspective. Cambridge University Press, New York, NY, USA, 1 édition, 2008. (Cited on page 44.)

[Groß 2007] Roderich Groß et Marco Dorigo. *Fifty years of self-assembly experimentation.* In IROS 2007, Modular Robotics Workshop, Marina del Rey, CA, 2007. USC Information Science Institute. (Cited on page 42.)

[Grygorash 2006] Oleksandr Grygorash, Yan Zhou et Zach Jorgensen. *Minimum Spanning Tree Based Clustering Algorithms.* In Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, ICTAI '06, pages 73–81, Washington, DC, USA, 2006. IEEE Computer Society. (Cited on page 35.)

[Guyon 2006] Isabelle Guyon, Steve Gunn, Masoud Nikravesh et Lotfi A. Zadeh. Feature extraction: Foundations and applications (studies in fuzziness and soft computing). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited on pages 15, 21 and 107.)

[Hastie 2009] Robert; Friedman Jerome Hastie Trevor; Tibshirani. The elements of statistical learning (2nd ed.). 2009. (Cited on pages 14, 33, 34 and 44.)

[Haykin 1998] Simon Haykin. Neural networks: A comprehensive foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd édition, 1998. (Cited on page 27.)

[He 2005] Xiaofei He, Deng Cai et Partha Niyogi. *Laplacian Score for Feature Selection.* In Advances in Neural Information Processing Systems 18, 2005. (Cited on pages 15, 22 and 108.)

[He 2011] Xiaofei He, Ming Ji, Chiyuan Zhang et Hujun Bao. *A Variance Minimization Criterion to Feature Selection Using Laplacian Regularization.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 10, pages 2013–2025, Octobre 2011. (Cited on page 22.)

[Jain 1988] Anil K. Jain et Richard C. Dubes. Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. (Cited on pages 25 and 26.)

[Jain 1999] A. K. Jain, M. N. Murty et P. J. Flynn. *Data clustering: a review.* ACM Comput. Surv., vol. 31, no. 3, pages 264–323, Septembre 1999. (Cited on pages 11, 24, 25 and 33.)

[Kannan 2000] R. Kannan, S. Vempala et A. Veta. *On clusterings-good, bad and spectral.* In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, FOCS '00, pages 367–, Washington, DC, USA, 2000. IEEE Computer Society. (Cited on page 53.)

[Kannan 2004] Ravi Kannan, Santosh Vempala et Adrian Vetta. *On clusterings: Good, bad and spectral.* J. ACM, vol. 51, no. 3, pages 497–515, Mai 2004. (Cited on pages 19 and 48.)

[Knuth 1993] Donald E. Knuth. The stanford graphbase - a platform for combinatorial computing. ACM, 1993. (Cited on page 69.)

[Kohavi 1997] Ron Kohavi et George H. John. *Wrappers for Feature Subset Selection.* Artificial Intelligence, vol. 97, no. 1-2, pages 273–324, 1997. (Cited on page 21.)

[Kohonen 2001] T. Kohonen, M. R. Schroeder et T. S. Huang, editeurs. Self-organizing maps. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd édition, 2001. (Cited on pages 13, 14, 27, 28 and 51.)

[Koikkalainen 2007] Pasi Koikkalainen et Ismo Horppu. *Handling Missing Data with the Tree-Structured Self-Organizing Map.* In IJCNN, pages 2289–2294, 2007. (Cited on pages 5, 39 and 41.)

[Krasnogor 2005] Natalio Krasnogor et Marian Gheorghe. *Systems Self-Assembly.* International Workshop, The Grand Challenge in Non-Classical Computation, 18-19th April 2005, the University of York, United-Kingdom, April 2005. (Cited on page 42.)

[Kruskal 1956] Joseph B. Kruskal. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.* Proceedings of the American Mathematical Society, vol. 7, no. 1, pages 48–50, Février 1956. (Cited on page 35.)

[LeFevre 2010] Kristen LeFevre et Evimaria Terzi. *GraSS: Graph Structure Summarization.* In SDM, pages 454–465, 2010. (Cited on page 53.)

[Leskovec 2010] Jure Leskovec, Kevin J. Lang et Michael Mahoney. *Empirical comparison of algorithms for network community detection.* In WWW '10: Proceedings of the 19th international conference on World wide web, pages 631–640, New York, NY, USA, 2010. ACM. (Cited on pages 48 and 70.)

[Liu 1998] Huan Liu et Hiroshi Motoda. Feature selection for knowledge discovery and data mining. Kluwer Academic Publishers, Norwell, MA, USA, 1998. (Cited on page 15.)

[Luxburg 2007] Ulrike Luxburg. *A tutorial on spectral clustering.* Statistics and Computing, vol. 17, pages 395–416, December 2007. (Cited on pages 19 and 20.)

[Macdonald 2000] Donald Macdonald et Colin Fyfe. *The kernel self-organising map.* In Knowledge-Based Intelligent Information and Engineering Systems, pages 317–320, 2000. (Cited on page 68.)

[Martinetz 1991] T. Martinetz et K. Schulten. *A "Neural-Gas" Network Learns Topologies.* Artificial Neural Networks, vol. I, pages 397–402, 1991. (Cited on pages 30 and 90.)

[Martinetz 1993] T. M. Martinetz, S. G. Berkovich et K. J. Schulten. *'Neural-gas' network for vector quantization and its application to time-series prediction.* Neural Networks, IEEE Transactions on, vol. 4, no. 4, pages 558–569, Juillet 1993. (Cited on page 44.)

[Michael Dittenbach 2002] Dieter Merkl Michael Dittenbach Andreas Rauber. *Uncovering Hierarchical Structure in Data Using the Growing Hierarchical Self-Organizing Map.* Neurocomputing, vol. 48, no. 1-4, pages 199–216, October 2002. (Cited on page 87.)

[Mondada 2004] Francesco Mondada, Giovanni C. Pettinaro, Andre Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella et Marco Dorigo. *Swarm-Bot: A New Distributed Robotic Concept.* Auton. Robots, vol. 17, no. 2-3, pages 193–221, 2004. (Cited on page 42.)

[Murata 1994] Satoshi Murata, Haruhisa Kurokawa et Shigeru Kokaji. *Self-assembling machine.* In IEEE International Conference on Robotics and Automation, volume 1, pages 441–448, Sans Diego, CA, USA, May 1994. (Cited on page 42.)

[Murzin 1995] A.G. Murzin, S.E. Brenner, T. Hubbard et C. Chothia. *SCOP: a structural classification of proteins database for the investigation of sequences and structures.* J Mol Biol, vol. 247, no. 4, pages 536–40, 1995. (Cited on page 76.)

[Nene 1996] S. A. Nene, S. K. Nayar et H. Murase. *Columbia Object Image Library (COIL-20).* Rapport technique, Feb 1996. (Cited on pages 61 and 65.)

[Newman 2006] M. E. J. Newman. *Finding community structure in networks using the eigenvectors of matrices.* Phys. Rev. E, vol. 74, page 036104, Sep 2006. (Cited on page 68.)

[Pearl 2003] F. M. G. Pearl, C. F. Bennett, J. E. Bray, A. P. Harrison, N. Martin, A. Shepherd, I. Sillitoe, J. Thornton et C. A. Orengo. *The CATH database: an extended protein family resource for structural and functional genomics.* Nucl. Acids Res., vol. 31, no. 1, pages 452–455, 2003. (Cited on page 76.)

[Peura 1999] Markus Peura. *The Self-Organizing Map of Attribute Trees*, 1999. (Cited on pages 5, 38, 40 and 51.)

[Prim 1957] R. C. Prim. *Shortest connection networks and some generalizations.* Bell System Technology Journal, vol. 36, pages 1389–1401, 1957. (Cited on page 35.)

[Pyle 1999] Dorian Pyle. Data preparation for data mining. Morgan Kaufmann, 1999. (Cited on page 18.)

[Radicchi 2004] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto et D. Parisi. *Defining and identifying communities in networks.* Proceedings of the National Academy of Sciences of the United States of America, vol. 101, no. 9, pages 2658–2663, Mars 2004. (Cited on page 48.)

[Rand 1971] William M. Rand. *Objective Criteria for the Evaluation of Clustering Methods.* Journal of the American Statistical Association, vol. 66, no. 336, pages pp. 846–850, Dec., 1971. (Cited on pages 12, 45 and 47.)

[Reif 2005] John H. Reif, Sudheer Sahu et Peng Yin. *Complexity of Graph Self-assembly in Accretive Systems and Self-destructible Systems.* In Alessandra Carbone et Niles A. Pierce, editeurs, DNA, volume 3892 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2005. (Cited on page 42.)

[Roweis 2000] Sam T. Roweis et Lawrence K. Saul. *Nonlinear dimensionality reduction by locally linear embedding.* SCIENCE, vol. 290, pages 2323–2326, 2000. (Cited on page 108.)

[Saeys 2007] Yvan Saeys, Iñaki Inza et Pedro Larrañaga. *A review of feature selection techniques in bioinformatics.* Bioinformatics, vol. 23, no. 19, pages 2507–2517, Octobre 2007. (Cited on page 107.)

[Samsonova 2006] Elena V. Samsonova, Joost N. Kok et Ad P. Ijzerman. *Treesom: Cluster analysis in the self-organizing map. Neural Networks.* American Economic Review, vol. 82, pages 1162–1176, 2006. (Cited on pages 5, 38 and 40.)

[Santini 2012] Guillaume Santini, Henry Soldano et Joël Pothier. *Automatic classification of protein structures relying on similarities between alignments.* BMC Bioinformatics, vol. 13, page 233, 2012. (Cited on page 78.)

[Shalabi 2006] Luai Al Shalabi et Zyad Shaaban. *Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix.* In Proceedings of the International Conference on Dependability of Computer Systems, DEPCOS-RELCOMEX '06, pages 207–214, Washington, DC, USA, 2006. IEEE Computer Society. (Cited on page 18.)

[Shi 1997] Jianbo Shi et Jitendra Malik. *Normalized Cuts and Image Segmentation.* In Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), CVPR '97, pages 731–, Washington, DC, USA, 1997. IEEE Computer Society. (Cited on page 68.)

[Stanoev 2011] Angel Stanoev, Daniel Smilkov et Ljupco Kocarev. *Identifying communities by influence dynamics in social networks.* CoRR, vol. abs/1104.5247, 2011. (Cited on pages 58, 69 and 81.)

[Strehl 2003] Alexander Strehl et Joydeep Ghosh. *Cluster ensembles — a knowledge reuse framework for combining multiple partitions.* J. Mach. Learn. Res., vol. 3, pages 583–617, Mars 2003. (Cited on page 47.)

[Tan 2005] Pang-Ning Tan, Michael Steinbach et Vipin Kumar. Introduction to data mining. Addison-Wesley, 2005. (Cited on pages 12 and 47.)

[Tenenbaum 2000] J. B. Tenenbaum, V. de Silva et J. C. Langford. *A global geometric framework for nonlinear dimensionality reduction.* Science (New York, N.Y.), vol. 290, no. 5500, pages 2319–2323, Décembre 2000. (Cited on page 108.)

[Vesanto 1999] Juha Vesanto. *SOM-Based Data Visualization Methods.* Intelligent Data Analysis, vol. 3, pages 111–126, 1999. (Cited on page 51.)

[Vicente 2004] Vellido A. Vicente D. *Review of Hierarchical Models for Data Clustering and Visualization.* Girldez, R., et al. (eds.), 2004. (Cited on pages 14 and 33.)

[West 2000] Douglas B. West. Introduction to Graph Theory (2nd Edition). Prentice Hall, Août 2000. (Cited on pages 12 and 19.)

[Xu 2005] Rui Xu et D. Wunsch. *Survey of clustering algorithms.* Neural Networks, IEEE Transactions on, vol. 16, no. 3, pages 645–678, Mai 2005. (Cited on page 44.)

[Xu 2007] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng et Thomas A. J. Schweiger. *SCAN: a structural clustering algorithm for networks.* In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '07, pages 824–833, New York, NY, USA, 2007. ACM. (Cited on page 68.)

[Yu 2003] Lei Yu et Huan Liu. *Feature selection for high-dimensional data: A fast correlation-based filter solution*. In in ICML, pages 856–863, 2003. (Cited on page 21.)

[Zhang 2010] Ning Zhang, Yuanyuan Tian et Jignesh M. Patel. *Discovery-driven graph summarization*. In ICDE, pages 880–891, 2010. (Cited on page 53.)

[Zhao 2007] Zheng Zhao et Huan Liu. *Spectral feature selection for supervised and unsupervised learning*. In Proceedings of the 24th international conference on Machine learning, ICML '07, pages 1151–1157, New York, NY, USA, 2007. ACM. (Cited on page 108.)

[Zhao 2013] Zheng Zhao, Lei Wang, Huan Liu et Jieping Ye. *On Similarity Preserving Feature Selection*. IEEE Trans. Knowl. Data Eng., vol. 25, no. 3, pages 619–632, 2013. (Cited on page 108.)