Ecole Polytechnique de Louvain

Master Thesis

Spectral clustering algorithms for directed graphs

Author

Hadrien VAN LIERDE

Supervisor

Prof. Jean-Charles DELVENNE

Readers

Prof. Paul VAN DOOREN
Prof. Marco SAERENS

June 2015

I wish to thank Prof. Jean-Charles Delvenne for his time and his availability. His advices and his ideas were always very helpful.

Contents

Introduction

The past years have witnessed the emergence of large networks in various disciplines. Social networks, including networks based on social connections (Facebook, Twitter, etc.), multimedia sharing networks (e.g. YouTube) or professional networks (e.g. LinkedIn) are used to model the social interactions between individuals. Biological networks such as neural networks or food webs help understand the ties between natural entities. Technological networks such as the network formed by Internet Service Providers also form graphs with a high number of connections.

Several approaches showed that the structure of graphs based on real-world data is not governed by randomness. The degree distribution follows a power-law distribution ([1],[2]), the small-world phenomenon states that the average distance between nodes is small ([3],[4]), the graph may include connexions that are not reciprocal, edges are not uniformly distributed ([5]), etc. This last observation states that nodes in real-world graphs tend to cluster into groups with high internal edge density and low density of edges between groups.

This property has motivated the design of a plethora of methods identifying such groups in various types of networks (see e.g. [6]). These methods are known as *clustering algorithms* and the groups of nodes they extract are referred to as *clusters*. They all rely on the same principle: grouping nodes that are similar in some sense. The most common definition of similarity is based on the presence or absence of edge connecting pairs of nodes, which identifies clusters with a large amount of "intra-cluster" edges and few "inter-cluster" edges. This basic definition of similarity already proved its ability to extract insightful information about the structure of networks (see [6] for a review of algorithms and fields of applications).

Most of the methods available were essentially designed for undirected networks. However, many real-world networks have an inherent structure of directed graph with the presence of asymmetric connections between entities. Such examples of directed graphs include some social networks (such as Twitter), biological networks (e.g. trophic webs governed by predator-prey relationships), etc. When dealing with directed networks, an important question arises: do we take the directionality of edges into account in the clustering task and if yes, how do we do so?

The challenges in clustering directed graphs

When a graph contains asymmetric ties, the extension of methods that were designed for undirected networks is hard for two major reasons.

1. The nature of data changes: the adjacency matrix of a directed graph is no longer symmetric, the notion of connected component becomes ambiguous, etc. As pointed out by Santo Fortunato in [14], these differences in the structure of data cause many undirected graph clustering algorithms to fail on directed networks if not adapted.

2. It is unclear how directionality of edges should be taken into account in the clustering task. In other words, there is an ambiguity in the definition of the notion of similarity

between nodes in directed graphs. Several approaches were proposed resulting in various definitions of the clustering problem for directed networks[1].

As an illustration, we consider the graph depicted by figure 1 representing a food web. In such network, nodes are living beings and edges represent predator-prey relationships. We expect that a clustering algorithm would extract the classical trophic levels, namely producers, primary, secondary and tertiary consumers and top-level predators. It is clear that nodes within a trophic level are very unlikely to be connected by an edge. Hence classical clustering methods based on the density of edges would fail to detect the levels. An efficient algorithm should take into account the fact that all nodes within a trophic level are connected to other levels in a similar way, highlighting at the same time the global hierarchy between clusters of the graph[2].

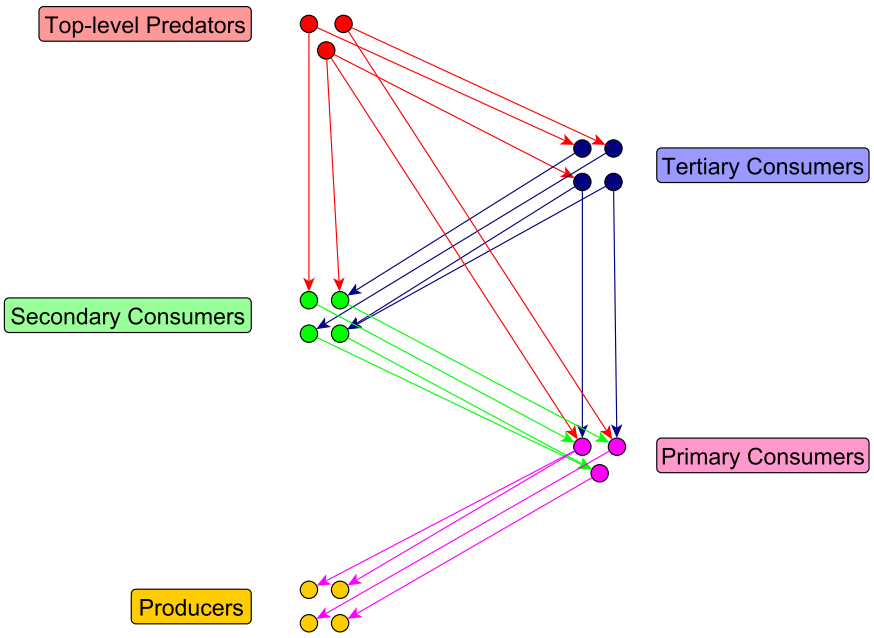


Figure 1: Trophic network. Nodes are living beings, a connection (a, b) means a is a predator of b. Nodes are coloured according to their trophic level.

The power of spectral methods

Among the clustering algorithms for undirected graphs, spectral clustering is very popular due to its high flexibility. Donath and Hoffman [7] were the first to suggest the use of eigenvectors of matrices[3] related to undirected graphs for the extraction of balanced clusters. This idea paved the way for the design of new clustering methods for undirected based on the spectrum of graph-related matrices, referred to as *spectral clustering algorithms*. The key property on which spectral-related algorithms are based is the strong relationship between the magnitude of the second eigenvalue of these matrices and the presence of two balanced clusters in the associated graph.

[1]These different definitions will be described in chapter 1, see [15] for a comprehensive survey about methods of directed graph clustering.

[2]Trophic networks and a more general class of networks with an underlying hierarchy between clusters will be further analysed in chapter 3 of this report.

[3]For instance, the adjacency matrix or the Laplacian matrix of an undirected graph that will be defined in chapter 1, the Laplacian can be viewed as a normalized an version of the adjacency matrix.

In her paper about the *Algorithmic Challenges in Web Search Engines* ([8], 2003), Monika Henzinger describes six challenges concerning web search engines that are not yet or only partially solved. One of these challenges is that of showing "whether or not the eigenvectors of the matrices associated to a directed graph are also related to balanced decompositions of the directed graph". In other words, Henzinger's concern was to know whether or not the spectrum of some matrices related to directed graphs could be used to extract clusters and, if this is the case, what kind of clusters it would detect.

The major difficulty when dealing with directed graphs is that the graph-related matrices (such as the adjacency matrix) are not symmetric, causing their spectrum to be complex in the general case. Several attempts of spectral clustering algorithms for directed graphs were proposed, some of which are based on a symmetrization of graph-related matrices (see for instance [21], [25] and [26]). However, as we will show in chapter 1 of this report, it is unclear how these methods take directionality of edges into account. In particular, these methods may fail to detect tropic levels in the graph depicted by figure 1.

The major part of this report is dedicated to the description of new spectral clustering algorithms for directed graphs. The specificity of these methods is that they are entirely based on the computation of complex eigenvalues and eigenvectors of a non-symmetric matrix associated to a directed graph, referred to as the *graph Laplacian* of a directed graph. We claim that our algorithms succeed in detecting clusters when the connection between the clusters follow one of the two following patterns.

1. The two first algorithms (described in chapter 2), referred to as *Multiple Cycle Eigenvalues* (MCE) and *Single Cycle Eigenvalue* (SCE) clustering algorithms, are specifically designed for networks where the connections between clusters of nodes form a cyclic pattern (such as the graph depicted in figure 1 (left)).

2. The third algorithm (described in chapter 3), referred to as *Block Acyclic Spectral* (BAS) clustering algorithm detects clusters in directed graphs where the connections between clusters follow an acyclic pattern such as the graph depicted in figure 1 (right)).

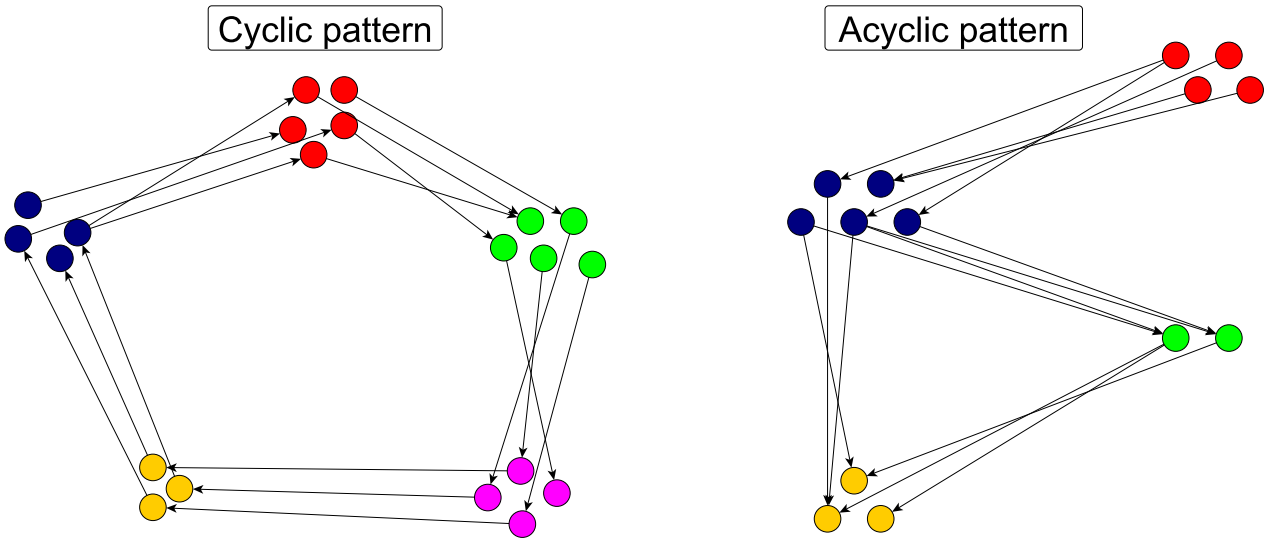


Figure 2: Directed graphs where nodes are clustered and the connexions between clusters follow a cyclic (left) or acyclic (right) pattern, colours denote clusters of nodes.

All three algorithms are tested on synthetic data and the third one is also tested on three real-world networks: a network based on scores of teams in a sport competition, a trophic network

and a large network of Internet Service Providers.

The goal of this report is twofold: firstly, to show that the spectrum of matrices related to a directed graph give insightful information about its structure and secondly, to use these properties for the design of algorithms detecting clusters with a particular pattern of connections.

Structure of the report

The rest of this report is organized as follows.

Chapter 1 is a short survey about existing clustering algorithms with a particular focus on spectral methods. The classical spectral clustering algorithm for undirected graphs is first described. Then we provide an overview of the existing clustering methods for directed graphs. Finally, we describe in more details three spectral clustering algorithms for directed graphs that were already proposed in the literature.

Chapter 2 contains the description of two spectral algorithms for the detection of clusters with a cyclic pattern of connections in directed graphs (such as in figure 2 (left)). A proof of consistency and the results of tests on synthetic data are also provided.

Chapter 3 is dedicated to a spectral algorithm for the detection of clusters with an acyclic pattern of connections in directed graphs (such as in figure 2 (right)). The algorithm is tested on three real-world networks (a network based on sport competitions, a trophic network and a network of Internet Service Providers).

Notations and terminology used in the report

In this report, we refer to graphs as triplets $G = (V, E, W)$ where V is a set of n nodes labelled from 1 to n, $E \subseteq V \times V$ is a set of edges and $W \in \mathbb{R}^{n \times n}$ is the adjacency matrix or weight matrix verifying

$$(u, v) \in E \Leftrightarrow W_{uv} \neq 0.$$

We restrict ourselves to the case of positive weights. Although in some situations networks may contain negative weighted edges (such as in neuronal networks where the synapses may be excitatory or inhibitory [29]), in the applications analysed in this report (networks of service providers or trophic networks for instance), weights of edges are positive. Moreover, requiring positive weights make the demonstration of some theorems possible and allows to use some algorithmic techniques that are further described in this report.

As the majority of graphs considered are directed, we use some terms that are intimately related to the directionality of edges: for any pair of node a, b connected by a directed edge (a, b) in a directed graph, we refer to b as an *out-neighbour* of a and to a as an *in-neighbour* of b and we refer to edge (a, b) as an *out-edge* for node a and an *in-edge* for node b.

Chapter 1

Overview of spectral clustering algorithms for directed and undirected graphs

In this chapter, we first present the classical spectral clustering algorithm for undirected graphs and how it is related to the clustering problem in the case of undirected networks. Then, we provide an overview of the existing formulations of the clustering problem for directed networks. Next, we provide a brief overview of the existing approaches for clustering directed graphs and we analyse the success or failure of these approaches in detecting different types of clusters. Finally, we describe the extensions of the classical spectral clustering algorithm to the case of directed graphs that were proposed in the literature.

1 Spectral clustering of undirected graphs

Spectral clustering can be described as a method using eigenvalues and eigenvectors of graph-related matrices in order to detect clusters of nodes in the network. This method was first introduced by Donath and Hoffman in [7] (1973) and by Fiedler in [9] (1973) who suggested to use eigenvectors and eigenvalues of a graph-related matrix called the *graph Laplacian*[1] to solve the problem of detecting clusters in an undirected graph. Several advances in other fields allowed a formalization of these first approaches, in particular the link between the detection of two clusters in a graph and the spectrum of its Laplacian that was suggested by the work of Cheeger in [10] (1970). Advances in the design of algorithms for the computation of eigenvalues (such as Lanczos algorithm) allowed the formulation of various spectral clustering approaches, especially in the 1990's (Spielman and Teng (1996) give an overview of the history of spectral clustering methods in [11]). However, it is not until 2008 that a complete proof of consistency of the algorithm was provided by Ulrike von Luxburg in [12]. Before that, the algorithm was essentially based on heuristic arguments.

The *Tutorial on Spectral Clustering* [13] by Ulrike Von Luxburg (2007) provides a self-contained introduction to spectral clustering of undirected graphs. In this section we give an overview of the theoretical background and we give the classical formulation of spectral clustering algorithm, these developments provide the basis for meaningful extensions to the directed case that are described in subsequent sections.

[1] The definition of graph Laplacian is given in section 1.2, the Laplacian can be viewed as a normalized and shifted version of the adjacency matrix.

1.1 Problem formulation

Given an undirected and possibly weighted graph $G = (V, E)$, the clustering task can be viewed as partitioning V into groups of nodes such that there is a large amount of edges inside groups and a low amount of edges between groups. In the following paragraphs we denote by k the number of such groups (or clusters) that we wish to find.

In the particular case where $k = 2$, one can formulate the problem as the partitioning of V into two sets S and $\bar{S}$ such that the quantity

$$\frac{|E(S, \bar{S})|}{\min(|E(S, V)|, |E(\bar{S}, V)|)}$$

is minimized, where $E(A, B)$ denotes the set of edges with one endpoint in A and the other endpoint in B. In the theoretical developments of next paragraph, we focus on the case $k = 2$.

Spectral clustering algorithms aim at finding an approximate solution to the optimization problem above. However, we must insist on the fact that these methods can solve far more general problems. Indeed, let us assume we are given a dataset consisting of n points $X_1, ..., X_n$. We define a symmetric similarity function $s : \{1, ..., n\}^2 \to \mathbb{R}^+$ where high value of $s(i, j)$ means X_i and X_j are highly similar. Based on this similarity function, we can build an undirected graph $G = (V, E)$ with adjacency matrix W such that $W_{ij} = s(i, j)$. Applying a spectral clustering algorithm to this graph will extract clusters such that points within the same cluster are rather similar and points belonging to different clusters are dissimilar.

Apart from the fact that it must be symmetric and positive valued, the similarity function can take any form (for example, when the dataset consists of points in an Euclidean space, the similarity function can be created by using a k-nearest-neighbour algorithm, see [13] for more details). This property makes spectral clustering algorithms more flexible than other clustering algorithms such as k-means where the similarity between points is defined as a distance.

1.2 Background in spectral graph theory

Spectral graph theory encompasses spectral properties of matrices associated to undirected graphs in order to extract insightful information about these networks. The reader is referred to the monograph by Fan Chung [23] for a comprehensive overview of spectral graph theory.

Spectral properties of graphs should in general be interpreted as the spectral properties of a particular matrix, called the graph Laplacian.

Definition 1 (*graph Laplacian*)

Given an undirected graph $G = (V, E, W)$ with weight matrix $W \in \mathbb{R}_+^{n \times n}$, we define the $n \times n$ Laplacian matrix L of G as

$$L_{uv} = \begin{cases} 1 - \frac{W_{uu}}{d_u} & \text{if } u = v \text{ and } d_u \neq 0 \\ -\frac{W_{uv}}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

where d_u is the degree of node u namely

$$d_u = \sum_{(u,v) \in E} W_{uv}.$$

A more popular way to define the Laplacian when all nodes have strictly positive degree is

$$L = I - D^{-1/2} W D^{-1/2}$$

where D is the diagonal matrix of degrees: $D = diag(d_u : u \in V)$.

Matrix L is obviously real and symmetric, hence it is diagonalizable and has real eigenvalues and eigenvectors. We can easily prove that these eigenvalues lie in the interval $[0, 2]$. Zero is always an eigenvalue of L and the vector of all ones is the corresponding eigenvector.

Spectral properties of graphs are formulated as a relation, often an inequality, between eigenvalues of L and constants that are related to some aspects of the graph (dynamic, density of edges, etc.). One of these constants is the Cheeger constant.

Definition 2 (*Cheeger constant*)

Given an undirected graph $G = (V, E, W)$ with weight matrix $W \in \mathbb{R}_+^{n \times n}$ and a subset $S \subset V$, the volume of the set S is given by

$$vol(S) = \sum_{u \in S} d_u.$$

We also define the quantity

$$\phi_G(S) = \frac{|E(S, \bar{S})|}{\min(vol(S), vol(\bar{S}))}$$

where $E(S, \bar{S})$ is the set of edges having one endpoint in S and the other in $\bar{S}$.
Then, the Cheeger constant of G is given by

$$\phi_G = \min_{S \subset V} \phi_G(S).$$

Hence the Cheeger constant is an indication about the quality of a partition of G into two clusters. The relation between ϕ_G and the spectrum of L is given by the following theorem, which actually combines *Cheeger inequality* (proved by Cheeger in 1970 in [10]) and another inequality given by Fan Chung in her monograph (see [23]).

Theorem 1

Given an undirected graph G with Laplacian L with eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$, the Cheeger constant of G is related to λ_1 by the following inequalities:

$$\begin{array}{rcll} \frac{\phi_G^2}{2} & \leq & \lambda_1 & \textit{Cheeger inequality} \\ \lambda_1 & \leq & 2\phi_G & \textit{Other inequality by Chung.} \end{array}$$

This theorem highlights the information carried by the first non-trivial eigenvalue λ_1 of L: if λ_1 is close to zero, then the Cheeger constant of G is low and the graph may be partitioned into two clusters with high density of edges within the clusters and few inter-cluster edges.

1.3 Spectral clustering algorithm

The key point of spectral clustering is to observe that the problem of clustering the nodes of a graph into two clusters can be related to the spectrum of the Laplacian. Indeed, as shown in [13], a relaxation of this problem can be formulated as

$$\begin{array}{l} \min\limits_{f\in\mathbb{R}^n} f^T L f \\ \text{subject to } \left\{ \begin{array}{l} f \perp \mathbf{1} \\ \|f\|_2 = 1 \end{array} \right. \end{array} \tag{1.1}$$

where $\mathbf{1}$ is the vector of all ones. To get some insight about the relation between this optimization problem and the detection of two clusters, we remind that $\mathbf{1}$ is the eigenvector of L associated to eigenvalue 0. Hence the solution to 1.1 is the eigenvector associated to the second smallest eigenvalue λ_1 which is related to the Cheeger constant as stated in theorem 1. Moreover, in [13], Ulrike Von Luxburg shows that the sign of the corresponding eigenvector may be interpreted as an indicator of the two corresponding clusters. This statement leads to the following bi-partitioning algorithm for an undirected graph $G = (V, E)$:

1. compute the Laplacian L of G,
2. find the eigenvector associated to the second smallest eigenvalue by solving 1.1,
3. partition V into V_1 and V_2 where

$$\begin{array}{rcl} V_1 & = & \{u \in V : f_u \geq 0\} \\ V_2 & = & \{u \in V : f_u < 0\}. \end{array}$$

A non trivial generalization of the algorithm above to the case $k \geq 2$ involves the computation of the k smallest eigenvalues and the use of all k corresponding eigenvectors. This is the well-known spectral clustering algorithm.

Algorithm 1.1: Spectral clustering algorithm

Input: Adjacency matrix $W \in \mathbb{R}_+^{n\times n}$;
Parameters: $k \in \{2, 3, ..., n\}$;
Step 1: Compute the graph Laplacian L;
Step 2: Find the k eigenvectors associated to the k smallest eigenvalues and store them as the columns of a matrix $\Gamma \in \mathbb{R}^{n\times k}$;
Step 3: Consider each row of Γ as a point in $\mathbb{R}^k$ and cluster these points using a k-means algorithm. Let $\phi : \{1, ..., n\} \to \{1, ..., k\}$ be the function assigning each row of Γ to a cluster;
Step 4: Compute the estimation of cluster membership function $f : V \to \{1, ..., k\}$: $f(u) = \phi(u)$ for all $u \in \{1, ..., n\}$;
Output: estimation of cluster membership f;

There exist some variants of the algorithm based on slightly different definitions of the Laplacian but the global structure of the method stays the same. At step 3, we make use of k-means algorithm which is a popular algorithm for clustering a set of points in $\mathbb{R}^n$ into k clusters (for any positive integers n, k). k-means algorithm will be described in more details in section 2.4.3 of chapter 2. In next paragraph, we give an intuitive idea about why clustering the rows of Γ should give back the clusters of the graph

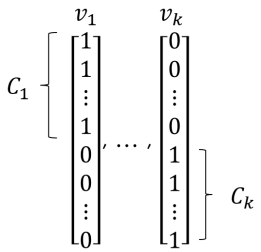
1.4 Alternative approach

In this section, we show an alternative and more intuitive way of understanding why spectral clustering algorithm works. This approach is not a theoretical justification of spectral methods but provides some intuition on how the method could be generalized to the case of directed networks.

Let us consider an undirected graph $G = (V, E)$ (possibly with positive weights) consisting of exactly k connected components. We denote by L the Laplacian of G and by L_1,...,L_k the Laplacians associated to each of the k connected components of G. Then, up to a permutation of the labels of vertices, L has a block diagonal form

$$L = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{bmatrix}.$$

Hence, L has at least k eigenvectors associated to eigenvalue 0 and each of these k eigenvectors is an indicator of the corresponding connected component. If the k connected components are C_1,...,C_k then the k eigenvectors v_1,...,v_k associated to eigenvalue 0 have the form



Hence, storing these eigenvectors as the columns of matrix $\Gamma \in \mathbb{R}^{n \times k}$ and clustering the rows of Γ into k clusters gives back the connected components.

Now if we append a few edges connecting nodes that did not belong to the same connected components, we change the number of connected components, however there are still k clusters of nodes with very few links between them. We wish to detect these clusters. By continuity, we may expect that there will still be k eigenvalues close to zero and the rows of the corresponding matrix Γ of eigenvectors will cluster in the same way: if two nodes belong to the same cluster, the corresponding rows are similar to each other (in terms of Euclidean distance for instance).

2 Clustering directed graphs

In a paper published recently, Santo Fortunato insists on the difficulty of designing clustering methods for directed networks: "Developing methods of community detection for directed graphs is a hard task. For instance, a directed graph is characterized by asymmetrical matrices (adjacency matrix, Laplacian, etc.), so spectral analysis is much more complex. Only a few methods can be easily extended from the undirected to the directed case. Otherwise, the problem must be formulated from scratch" [14]. This statement captures the fundamental difficulty that arises when dealing with directed networks: the nature of data changes. The asymmetry of matrices makes some methods much harder to define and it is unclear how the additional information carried by the directionality of edges should be taken into account. In this section, we first present an overview of the different definitions of the directed graph clustering problem encountered in the literature. Then, we briefly describe the existing methods that carry out these clustering tasks. For a review of most of the methods invented in 2013 and before, the reader is referred to the survey published by Malliaros and Vazirgiannis in 2013 [15].

2.1 Defining the problem

As in the undirected case (and as for the general clustering task), clustering methods for directed graphs should group nodes that are similar in the same cluster. Similarity should be a symmetric function. In the undirected case, similarity can be defined by the graph itself, where the similarity between two nodes is zero if they are not connected and, if they are connected by an edge, similarity is given by the weight of the corresponding edge. However, in the directed case, connections are not necessarily symmetric, making the definition of similarity more difficult.

According to the classification suggested in [15], clustering methods for directed networks can be divided into two categories: density-based and pattern-based clustering.

Firstly, density-based clustering can be viewed as a generalization of the methods designed for the undirected case. The goal is to group nodes of a graph into clusters by maximizing the number of intra-cluster edges and/or minimizing the number of inter-cluster edges. Figure 1.1 shows an example where nodes clearly form three clusters based on the density of edges. In cases where we only wish to detect groups of nodes that are closely connected, this type of cluster is appropriate. However, it does not clearly take the directionality of edges into account. Moreover, the extensions of methods designed for undirected networks is not always straightforward due to the difficulty encountered in extending some graph-theoretical concepts to the directed case (connectivity for instance can be interpreted as weak connectivity, connectivity or strong connectivity for directed graphs).

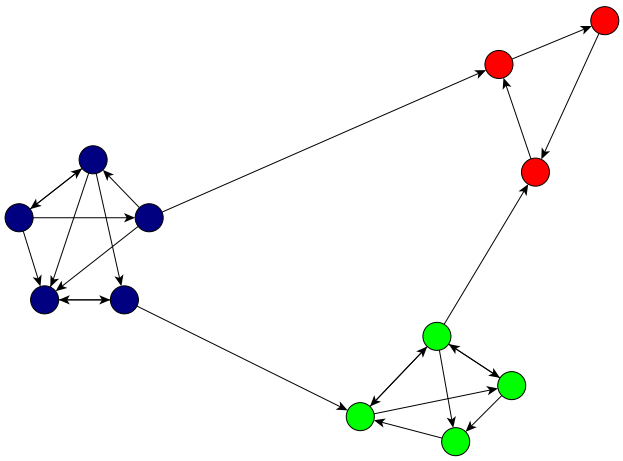


Figure 1.1: Directed graph with density-based clusters. Colouring of nodes denotes cluster membership.

Secondly, pattern-based clustering captures information about the global dynamic of the graph by detecting patterns of movements expressed by the directionality of edges. This means that nodes within a cluster are not necessarily connected by an edge. We may identify two types of pattern-based clustering methods: co-citation and flow-based clustering.

In the case of co-citation networks, nodes are clustered according to common patterns of in- or out-edges: if two nodes a and b both have out-edges towards a cluster α and both have in-edges coming from a cluster β then, a and b are likely to be themselves in the same cluster. This pattern is described in figure 1.2. The reason why it is called co-citation is the strong link it has with citation networks where we may group documents that cite the same category of articles and are cited by the same type of articles.

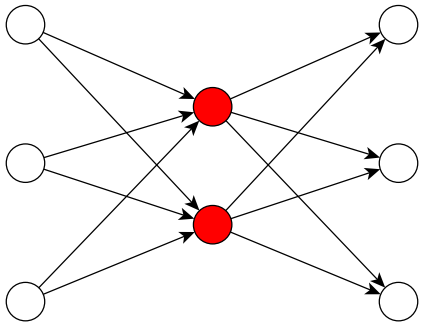


Figure 1.2: Directed graph with a co-citation-based cluster. The two nodes coloured in red are expected to belong to the same cluster.

In flow-based networks, edges form a pattern of flow between clusters. This situation is described by figure 1.3. In some situations, the flow between clusters can take a particular form (for instance a cycle as in the case depicted by figure 1.3).

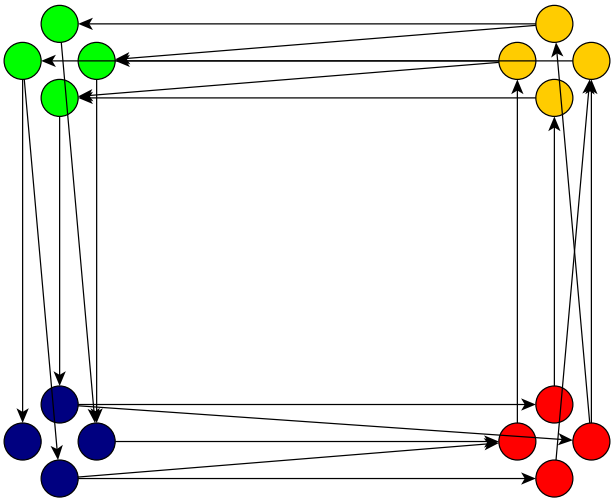


Figure 1.3: Directed graph with a cyclic pattern of flow between clusters. Colouring of nodes represents the cluster membership.

We insist on the fact that the definition of pattern-based cluster is not very restrictive as long as the clusters that we identify are based on the directionality of edges and not on the density of edges regardless of their direction. We could thus define other types of pattern-based clusters different from co-citation-based and flow-based networks. Moreover, both co-citation-based and flow-based clusters may be encountered in the same network.

2.2 Overview of clustering algorithms for directed networks

There is a wide variety of clustering methods for directed graphs, each of which detects a particular type of cluster. Generally speaking, algorithms can be partitioned into two categories: given a directed graph, either we build a related undirected graph and apply known clustering algorithms to it or we work directly on the directed graph and formulate new methods from scratch.

The list below briefly describes five categories of clustering methods for a directed (possibly weighted) graph $G = (V, E, W)$ with adjacency matrix W (this classification is inspired by the one in [15] but we do not include all the methods described in the survey).

Method 1: Transformation to an undirected graph

In that case, a new undirected graph G_u with symmetric adjacency matrix W_u is created from G. Then clustering methods for undirected graphs are applied to G_u. Information about directionality of edges in G should be included in the definition of G_u. The symmetric adjacency matrix W_u of the undirected graph can take the following forms.

1. $W_u = W + W^T$: this transformation simply ignores the directionality of edges. Hence, it should be avoided if the goal is to take the directionality of edges into account.

2. $W_u = W^T W + W W^T$: this transformation (called bibliometric symmetrization) was proposed in [16]. This method is particularly efficient to detect co-citation networks as depicted in figure 1.2. Clusters in G_u tend to group nodes that share the same neighbours in G. The reason is that $W^T W$ measures common in-neighbours and $W W^T$ measures common out-neighbours for every pair of nodes in G.

There exist some variations of the bibliometric symmetrization such as the degree-discounted symmetrization, which appends a normalizing factor to W_u taking degree of nodes into account (see [15] for more details).

Method 2: Modularity

Modularity-based methods follow the idea that graphs where nodes tend to form clusters differ from random graphs. One may define an objective function that quantifies the deviation of a graph with respect to a random graph for a specific partition of nodes into clusters. This objective function is called *modularity*. Then, maximizing the modularity, we find the optimal clustering of nodes. There are different ways of defining the criterion to maximize.

1. The first method is a straightforward generalization from the undirected case. According to the definition by Leicht and Newman ([17]) modularity is defined as the difference between the fraction of edges lying within clusters and the expected fraction of edges lying within clusters if edges where placed randomly in the graph. This method detects density-based clusters but it does not take explicitly the directionality of edges into account and it can fail to detect pattern-based clusters such as the ones appearing in figures 1.2 and 1.3.

2. Another definition of modularity was proposed in [18]: in this case, it measures the difference between the fraction of time spent by a random walker inside clusters and the expected time he would spent inside clusters if edges were randomly placed in the graph. If applied to the web graph, this algorithm would extract groups of webpages where a random surfer is likely to get trapped with little probability of travelling from one group of webpages to another. Hence, this algorithm still better detects density-based clusters and perhaps not pattern-based clusters such as co-citation-based clusters of figure 1.2.

Method 3: Clustering methods based on Chung's directed Laplacian

The adjacency matrix of a directed graph is not symmetric. Hence, a straightforward generalization of the Laplacian matrix to the directed case leads to a non-symmetric matrix which makes spectral analysis more difficult (complex eigenvalues, defective matrices, etc.). Fan Chung ([24]) proposed a symmetrized version of the Laplacian that applies to directed graphs. Based on this definition of Laplacian, D. Gleich ([25]) proposed a spectral clustering algorithm for directed graphs. From the results presented in [25], the algorithm seems to detect well density-based clusters. However, D. Gleich does not clearly analyse how well it detects pattern-based clusters. This algorithm will be further analysed in next section.

Method 4: Blockmodelling approach

Blockmodelling was first defined for undirected networks and then extended to directed graphs in [20]. We now give the intuition lying behind this approach but a more formal definition will be given in section 3.2. These methods attempt to partition the nodes of a graph $G = (V, E, W)$ into k blocks $V_1, ..., V_k$ respecting the following principle: given any two blocks V_i, V_j and any two nodes $a, b \in V_i$, then there is approximately the same number of in-edges (respectively out-edges) connecting a to nodes in V_j and of in-edges (respectively out-edges) connecting b to nodes in V_j. Hence, nodes within a block can be viewed as equivalent in terms of connections to other blocks. Clustering methods based on a blockmodelling tends to partition a graph into

such equivalent nodes.

These methods theoretically succeed in detecting pattern-based clusters such as co-citation-based clusters such as the one depicted in figure 1.2. Several such algorithms based on a blockmodelling approach are closely related to the spectral clustering algorithm for undirected graphs, one of which is based on the singular values and singular vectors of the adjacency matrix of directed graphs ([21]). This particular method is further described in section 3.2.

Method 5: Approaches based on random walk theory

These methods are based on the transition matrix P associated to a directed graph. Clusters are detected by computing a few extremal eigenvalues of P, then applying clustering techniques related to spectral clustering. These methods were first proposed in [26] and then further developed in [27]. In these documents, the methods are validated on networks with density-based clusters. This approach is further developed in next section.

This classification is not exhaustive, there exist other methods (for instance, based on information-theoretic concepts). However, the categories described above cover the majority of the existing approaches and the clustering problems they aim to solve.

3 Existing spectral clustering algorithms for directed networks

In this section, we describe three spectral clustering algorithms for directed graphs. These algorithms are the *spectral clustering algorithm based on Chung's directed Laplacian*, the *SVD clustering algorithm based on a blockmodelling approach*[2] and the *hierarchical clustering based on random walk theory*. All three were already presented in last section. The only characteristic they have in common is the fact that they are based on the spectrum of some graph-related matrix. The reason why we describe them in more details is that each of them is related to the new spectral clustering algorithms presented in subsequent chapters.

As said before, the major difficulty one encounters for the generalization of spectral clustering algorithms to the directed case is the asymmetry of the adjacency matrix. If we simply follow the steps suggested by the spectral clustering algorithm for undirected graphs, we must first define from which matrix we extract eigenvalues and eigenvectors. The first method described below is based on the spectrum of a symmetric matrix while the two others are based on non-symmetric matrices.

3.1 Spectral clustering algorithm based on Chung's directed Laplacian

This method is essentially based on the definition of a symmetric Laplacian for directed graphs given by Fan Chung in [24] and further developed by David Gleich in [25].

3.1.1 Definition of Laplacian and Cheeger inequality

We denote the matrix defined below as *Chung's directed Laplacian*.

Definition 3 (*Chung's directed Laplacian*)

Given a strongly connected directed, aperiodic[3]*graph $G = (V, E)$ and a weight matrix $W \in \mathbb{R}_+^{n\times n}$, we define the following quantities,*

- *D the out degree matrix:*
$$D = W\mathbf{1}$$
where $\mathbf{1}$ is the column vector of 1's, D is invertible since the graph is connected,
- *P the transition matrix of the graph,*
$$P = D^{-1}W$$
- *$\pi \in \mathbb{R}^{n+}$ the stationary distribution of the associated random walk and $\Pi = diag(\pi(u))_{u\in V}$.*

Then Chung's directed Laplacian of G is given by

$$L(G) = I - \frac{1}{2}(\Pi^{1/2}P\Pi^{-1/2} + \Pi^{-1/2}P^T\Pi^{1/2}).$$

[2]SVD is an abbreviation for *Singular Value Decomposition*.

The major interest of this definition is that it allows insightful extensions of the spectral theory of undirected graphs (in particular, the concepts of Cheeger constant and Cheeger inequality). The following proposition (see [25] for more details) provides the basis of these theoretical extensions.

Proposition 1

Given a strongly connected graph $G = (V, E)$ with weight matrix W and matrices P and Π as defined previously, Chung's directed Laplacian of G corresponds to the Laplacian of an undirected graph whose adjacency matrix is

$$\tilde{W} = \frac{\Pi P + P^T \Pi}{2}.$$

In her paper [24], Fan Chung also extends the notion of Cheeger constant to directed graphs.

Definition 4 (*Cheeger Constant*)

Given a directed graph $G = (V, E)$, its transition matrix P and the stationary distribution vector π

- *Given a subset $S \subset V$, the volume of the set S and the cut ∂S are defined by*

$$\begin{array}{rcl} vol(S) & = & \sum_{u \in V, v \in S} \pi(u) P(u, v) \\ vol(\partial S) & = & \sum_{u \in S, v \in \bar{S}} \pi(u) P(u, v) \end{array}.$$

- *The conductance of a cut $S \subset V$ is given by*

$$\phi_G(S) = \frac{vol(\partial S)}{\min(vol(S), vol(\bar{S}))}.$$

- *The Cheeger constant of the graph is defined by*

$$\phi_G = \min_{S \subset V} \phi_G(S).$$

Combining this new definition of Cheeger constant and the relation between directed and undirected network stated in proposition 1, one may prove a Cheeger inequality for directed graphs.

Theorem 2

Given a directed graph G with eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$, the Cheeger constant of G is related to λ_1 by the following inequalities:

$$\frac{\phi_G^2}{2} \leq \lambda_1 \leq 2\phi_G.$$

[3] If G is strongly connected Perron-Frobenius theorem guarantees the existence of at least one positive eigenvector π such that $\pi P = \pi$ which corresponds to the stationnary distribution of the random walk, moreover if G is aperiodic, all other eigenvalues have norm strictly smaller than 1. If G is not connected (or not aperiodic), one may replace P by the PageRank transition matrix of the form $P_{PR} = \alpha P + \frac{1-\alpha}{n} \mathbf{1}\mathbf{1}^T$

3.1.2 Spectral clustering algorithm

Cheeger inequality of theorem 2 provides the ingredient for a spectral clustering algorithm. Indeed if λ_1 is small ϕ_G also tends to be small and the graph may be partitioned into two clusters. Moreover, Gleich also shows in [25] that the associated eigenvector is an indicator of the clusters, which leads to the following algorithm for the partition of a directed graph $G = (V, E)$ into two clusters.

1. compute the Laplacian L of G,
2. find the eigenvector associated to the second smallest eigenvalue of L,
3. partition V into V_1 and V_2 where

$$\begin{aligned} V_1 &= \{u \in V : f_u \geq 0\} \\ V_2 &= \{u \in V : f_u < 0\}. \end{aligned}$$

Due to the relation with the Cheeger constant, we expect that this partition of V into V_1 in V_2 approximately minimizes the quantity

$$\frac{vol(\partial V_1)}{\min(vol(V_1), vol(V_2))}.$$

How do we interpret the result of this clustering algorithm? For a random walk in steady state on directed graph $G = (V, E)$ and for any subset A of V, $vol(A)$ and $vol(\partial A)$ are interpreted in the following way:

- $vol(A)$ is the probability of being inside A at a time t, while being in any node of the graph at time $t - 1$,
- $vol(\partial A)$ is the probability of being inside $V \setminus A$ at a time t, while being inside A at time $t - 1$.

Hence, if we give an interpretation based on web graph, this spectral clustering algorithm returns two clusters where a random surfer is likely to be trapped, with little chance of going from one cluster to the other. For this reason, we expect this algorithm to detect well density-based clusters (such as the one depicted in figure 1.1). However, this algorithm fails to detect pattern-based clusters such as co-citation-based clusters (figure 1.2).

As for the undirected case, this algorithm can be generalized for the detection of $k \geq 2$ clusters, by computing all k smallest eigenvalues of the Laplacian.

Algorithm 3.1: Spectral clustering algorithm based on Chung's directed Laplacian (CDL Spectral clustering)

Input: Adjacency matrix $W \in \mathbb{R}_+^{n \times n}$ of a directed graph;
Parameter: $k \in \{2, 3, ..., n\}$;
Step 1: Compute the graph Laplacian L;
Step 2: Find the k first eigenvectors and store them as the columns of a matrix $\Gamma \in \mathbb{R}^{n \times k}$;
Step 3: Consider each row of Γ as a point in $\mathbb{R}^k$ and cluster these points using a k-means algorithm. Let $\phi : \{1, ..., n\} \to \{1, ..., k\}$ be the function assigning each row of Γ to a cluster;
Step 4: Compute the estimation of cluster membership function $f : V \to \{1, ..., k\}$: $f(u) = \phi(u)$ for all $u \in \{1, ..., n\}$;
Output: estimation of cluster membership f;

3.2 SVD clustering algorithm based on blockmodelling approach

Blockmodels were first defined for undirected graphs (see for instance [19]) and then extended to the directed case by Wang et al in [20]. In this section, we first give a formal definition of stochastic blockmodel, then we briefly describe a spectral algorithm given in [21] in order to find blocks in a graph generated by a stochastic blockmodel.

As we briefly explained in last section, the blockmodelling approach aims at partitioning the nodes of a graph $G = (V, E, W)$ into k blocks $V_1, ..., V_k$ respecting the following principle: given any two blocks V_i, V_j and any two nodes $a, b \in V_i$, then there is approximately the same number of in-edges (respectively out-edges) connecting a to nodes in V_j and of in-edges (respectively out-edges) connecting b to nodes in V_j. Intuitively, methods based on a blockmodelling approach are expected to detect block of nodes that are equivalent in terms of connexions to other blocks. Equivalently, these methods attempt to find a block-structure in the adjacency matrix of a graph G, which yields an adjacency matrix of the form depicted by figure 1.4.

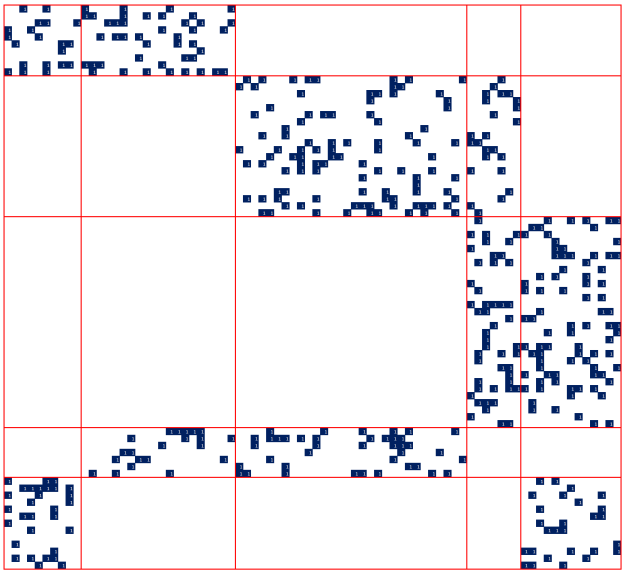


Figure 1.4: Typical shape of the adjacency matrix of a graph following a stochastic blockmodel with five blocks. Blue squares represent nonzero entries. Nodes are reordered according to their block membership.

3.2.1 Definition of stochastic blockmodel

The model presented here is restricted to the unweighted case. For an extension to weighted graphs, we refer to [22].

The parameters of a stochastic blockmodel are the following for an unweighted graph G defined on a set of nodes V:

- the number k of blocks,
- the vector ρ of probabilities of memberships: if $f : V \to \{1, ..., k\}$ is a block membership function defined for each node in V,

$$\rho_i = Pr\{f(u) = i\} \ \forall u \in V$$

namely, ρ_i is the probability for any node to belong to block i,

- the matrix P of probabilities of connexions: given a block membership function $f : V \to \{1, ..., k\}$;
$$P_{ij} = Pr\{u \to v | f(u) = i \text{ and } f(v) = j\} \ \forall u, v \in V$$
where $u \to v$ denotes the existence of an edge (u, v) in G. Namely, P_{ij} is the probability of having an edge (u, v) if u belongs to block i and v belongs to block j.

Vector ρ somehow defines the size of the blocks while matrix P determines the existence of directed edges between blocks. Hence, for a graph generated by a stochastic blockmodel, nodes are partitioned into k blocks and the probability of an edge between two nodes depends only on their respective block membership. In that sense, nodes within a block can be viewed as equivalent with respect to the connexions to other blocks. This equivalence between nodes within a block of a stochastic blockmodel is called *stochastic equivalence* (refer to [21] for a more formal definition of stochastic equivalence). To design a clustering algorithm based on stochastic blockmodelling, we must take a reverse approach: given a directed graph G, we want to fit a stochastic blockmodel that partitions the nodes of G into k blocks such that nodes within a block are stochastically equivalent. Based on this idea, several methods were designed for the extraction of blocks of a graph generated by a stochastic blockmodel. We present one of these that is similar to spectral clustering in its structure.

3.2.2 Spectral algorithm for the detection of a stochastic blockmodel

In [21], Sussman et al propose the following algorithm for the detection of blocks in a graph generated by a stochastic blockmodel. This algorithm is based on the singular value decomposition of the adjacency matrix.

Algorithm 3.2: Clustering algorithm based on singular value decomposition (SVD spectral clustering)

Input: $W \in \{0, 1\}^{n \times n}$;
Parameters: $d \in \{1, 2, ..., n\}$, $k \in \{2, 3, ..., n\}$;
Step 1: Compute the singular value decomposition $W = U\Sigma V^T$ where Σ has decreasing main diagonal;
Step 2: Let $\tilde{U}$ and $\tilde{V}$ be the first d columns of U and V, let $\tilde{\Sigma}$ contain the first d columns and first d rows of Σ;
Step 3: Define $\tilde{Z} = [\tilde{U}\tilde{\Sigma}^{1/2} | \tilde{V}\tilde{\Sigma}^{1/2}]$ by concatenation;
Step 4: Cluster the rows of $\tilde{Z}$ using a k-means algorithm with Euclidean distance. Let $\phi : \{1, ..., n\} \to \{1, ..., k\}$ be the function assigning each row of $\tilde{Z}$ to a cluster;
Step 5: Compute the estimation of block membership: $f(u) = \phi(u)$ for all $u \in \{1, ..., n\}$;
Output: estimation of block membership f;

The algorithm includes a parameter d called the *latent feature dimension*. When the graph is generated by a stochastic blockmodel, d should be number of singular values of W significantly greater than 0, hence we expect that the adjacency matrix W is well approximated by a matrix of rank d.

Explaining why this algorithm works is difficult but Sussman et al give a proof of its consistency in [21]. The main difference with classical spectral clustering algorithms is that the non-symmetric adjacency matrix is used instead of a Laplacian matrix.

The advantage of this algorithm is that it is very general in the type of clusters it is able to detect. It can theoretically detect density-based clusters, co-citation networks and flow-based networks. However, the major disadvantage is that it requires from nodes inside a block to be stochastically equivalent which makes it fail in the presence of perturbations or when some nodes inside a block have much more connexions than other nodes. This situation is depicted in figure 1.5 showing a particular co-citation network: when partitioning the graph into two clusters, nodes 1 and 2 are expected to form one cluster and the other nodes are expected to form another cluster. However with respect to this partitioning in two blocks, nodes 1 and 2 do not seem to be stochastically equivalent as node 1 is connected to two nodes of the other block while node 2 is connected to all six nodes. In contrast, in the network of figure 1.6 nodes of both clusters seem to be stochastically equivalent.

Another major difficulty that arises is the estimation of the latent feature dimension d. [21] provides a consistent estimator of the latent dimension. However, the authors mention that some work need to be done to see if this estimator yields good results in practice.

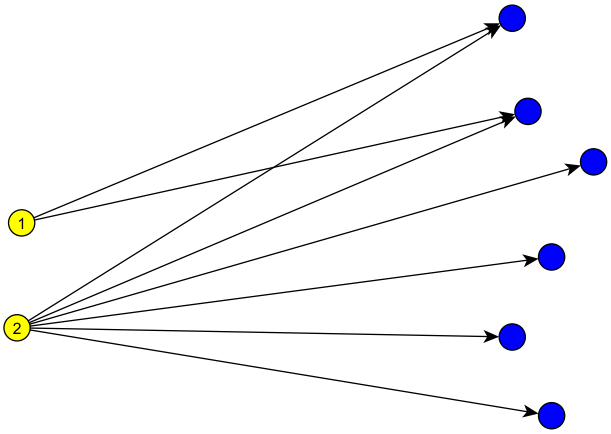


Figure 1.5: Co-citation network. Nodes within cluster 1 (yellow) do not seem to be stochastically equivalent.

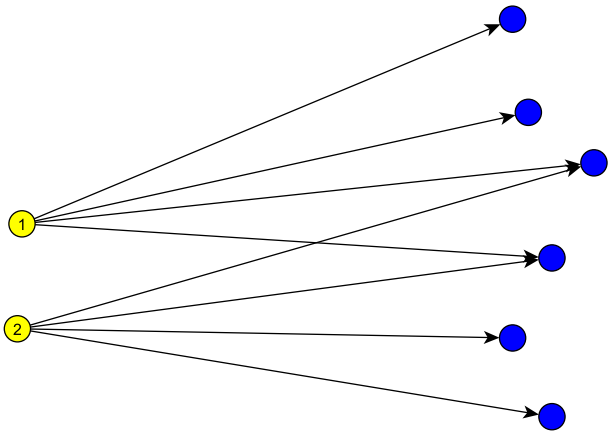


Figure 1.6: Co-citation network. Nodes within cluster 1 (yellow) seem to be stochastically equivalent.

3.3 Spectral clustering based on random walk theory

Spectral clustering algorithms based on random walks typically extract information from eigenvectors and eigenvalues of the transition matrix of a directed graph. Here we focus on the algorithm proposed by W. Pentney and M. Meila in [26]. This algorithm was first designed for the clustering of Biological Sequence Data where affinities between biological sequences are not necessarily symmetric, the network is thus directed.

3.3.1 Analogy with the undirected case

As we have seen in section 1, clustering an undirected graph $G = (V, E)$ into two clusters can be done by computing the eigenvector associated tot the second **smallest** eigenvalue of the Laplacian and using the sign of the components of the corresponding eigenvector to partition the nodes into two clusters. The method proposed in [26] is based on two observations regarding this algorithm.

Firstly, we observe that an equivalent algorithm could use the spectrum of the transition matrix P instead of the Laplacian L. Indeed, for an undirected[4] graph $G = (V, E)$ with weight matrix $W \in \mathbb{R}_+^{n\times n}$ and degree matrix $D = diag(W\mathbf{1})$,

$$\begin{array}{rcl} L & = & I - D^{-1/2}WD^{-1/2} \\ P & = & D^{-1}W \end{array} .$$

Hence,

$$L = I - D^{1/2}PD^{-1/2}.$$

Thus, if $u \in \mathbb{R}^n \setminus \{0\}$ is an eigenvector L with eigenvalue λ, namely

$$Lu = \lambda u$$

then $D^{-1/2}u$ is an eigenvector of P with eigenvalue $1 - \lambda$.

Hence, an equivalent spectral clustering algorithm would be to compute the second **largest** eigenvalue of P and to use the sign of the components of the corresponding eigenvector to partition the nodes into two clusters.

Secondly the authors of [26] observe that if the clusters are well defined (many intra-cluster edges and few inter-cluster edges) then the second eigenvector should be piecewise constant, with positive components corresponding to the nodes in one cluster and negative components corresponding to the nodes in the other cluster. Hence a curve representing the components of the second eigenvector should look like a step function, such as the one shown in figure 1.7.

[4]We also assume all nodes have at least one out-edge, hence matrix D defined further is invertible.

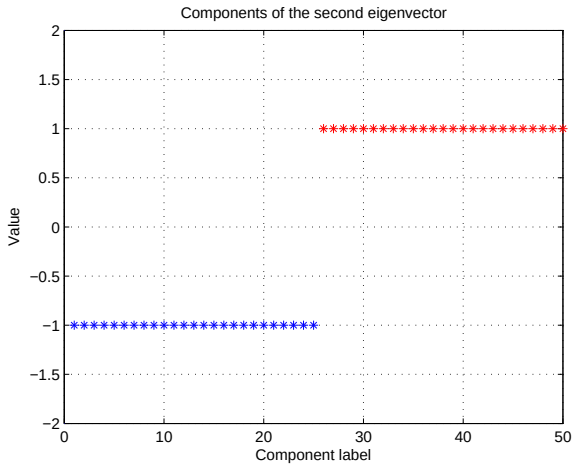


Figure 1.7: Typical behaviour of the components of the second eigenvector when nodes 1 to 25 belong to one cluster and nodes 26 to 50 belong to another cluster. This graph only shows the expected behaviour of components, components are not perfectly piecewise constant in practice.

3.3.2 Generalization to the directed case

Now let us consider a directed graph $G = (V, E, W)$. The algorithm presented in [26] relies on the fact that the graph of biological sequence data can be partitioned into density-based clusters (with very few inter-cluster edges and much more intra-cluster edges). The authors then assume that the spectrum of the transition matrix should follow the same behaviour as in the undirected case.

In order to partition the network into two clusters, they suggest to find the second largest eigenvalue of P in terms of magnitude[5] and then look at the components of the corresponding eigenvector. Components corresponding to nodes belonging to the same cluster should have approximately the same value. However, to extract the clusters properly, the authors suggest to first transform the components in the following way.

Firstly, as the eigenvector may be complex, they claim we should use the sum of the real parts and imaginary parts of these components: if the second eigenvector is $u \in \mathbb{C}^n$, we define another vector $v \in \mathbb{R}^n$ where $v_i = Re(u_i) + Im(u_i)$.

Secondly, extracting the clusters from v may be hard as slight perturbations may disrupt the piecewise constant behaviour of the components of v. Hence the authors suggest to smooth the components of v with a Gaussian kernel.

$$\Phi_v(x) = \frac{1}{nh} \sum_{j=1}^{n} \exp\left[-\frac{1}{2}\frac{(x - v_j)^2}{h^2}\right]$$

where h is a parameter representing the *kernel width*. Hence two nodes i and j should belong to the same cluster if $\Phi_v(v_i) \simeq \Phi_v(v_j)$.

Finally, they generalize their result by claiming that more than two clusters may be detected in the same way and still using only the second eigenvector of P. The main steps of their algorithm are summarized below.

[5] matrix P is not necessarily symmetric in the directed case, hence eigenvalues are complex.

Algorithm 3.3: Spectral clustering algorithm based on random walk (RW spectral clustering)

Input: $W \in \{0,1\}^{n \times n}$;
Parameters: $d \in \{1,2,...,n\}$, $k \in \{2,3,...,n\}$;
Step 1: Compute the transition matrix $P = D^{-1}W$;
Step 2: Calculate the second eigenvector u of P and calculate vector v such that $v_i = Re(u_i) + Im(u_i)$, for $1 \leq i \leq n$;
Step 3: Apply the Gaussian kernel function to the components of v and store the result in a vector w;

$$w_i = \Phi_v(v_i)$$

Step 4: Cluster the components of w into k clusters (for instance by using k-means algorithm). Let $\phi : \{1,...,n\} \to \{1,...,k\}$ be the function assigning each component of w to a cluster;
Step 5: Compute the estimation of cluster membership function $f : V \to \{1,...,k\}$: $f(u) = \phi(u)$ for all $u \in \{1,...,n\}$;
Output: estimation of cluster membership f;

The algorithm described in [26] includes a few additional steps (treatment of outliers etc.).

According to results described in the paper, the algorithm seems to perform well on density-based clusters (such as biological sequence data). However, the authors do not provide any theoretical proof of effectiveness of their method and they do not test it on other type clusters (such as co-citation-based networks similar to the one of figure 1.2).

4 Clustering directed graphs: a conclusion

In the two last sections, we distinguished two different types of clustering methods for directed graphs: density-based and pattern-based clustering algorithms.

Many methods are limited to *density-based* clustering for two major reasons. Firstly, finding density-based clusters is the natural clustering procedure for undirected graphs, hence it is often possible to extend objective functions and algorithmic tricks to the directed case (eg. modularity). Secondly, it is still interesting to find density-based clusters in some particular datasets even when edges are directed (eg. the biological sequence data).

All clustering techniques taking the directionality of edges explicitly into account belong to the category of *pattern-based* clustering methods. Most of the pattern-based clustering algorithms tend to find clusters of nodes that have similar adjacency lists: nodes sharing neighbours are grouped together. This is the case for instance for the *Bibliometric symmetrization* or for the *Blockmodelling* approaches. But do this methods cover all types of pattern-based clusters?

In the conclusion of their survey [15], Malliaros and Vazirgiannis mention the importance of designing application-driven approaches: "a different possible direction is to follow application-driven approaches, i.e., design domain-specific and application specific clustering algorithms for directed networks". Hence it may be interesting to design algorithms able to detect very specific pattern-based clusters encountered in particular fields of application. For instance, graphs with clusters that are connected in a hierarchical way (such as the one depicted at figure 1.8) are encountered in a wide range of fields including trophic networks and networks of Internet Service Providers.

How do the algorithms described previously perform on the clusters depicted in figure 1.8 ? The method based on Chung's directed Laplacian definitely fails to detect the clusters. The method based on singular value decomposition would succeed if nodes within a cluster are close to being stochastically equivalent otherwise it would fail. The algorithm based on random walks was initially designed for the detection of density-based clusters. However, it is still unclear how it would perform on pattern-based clusters such as the one depicted in figure 1.8.

Is it possible to design new spectral algorithms able to detect the clusters in the graph of figure 1.8 or other kinds of clusters with a specific structure? This question is treated in the two following chapters.

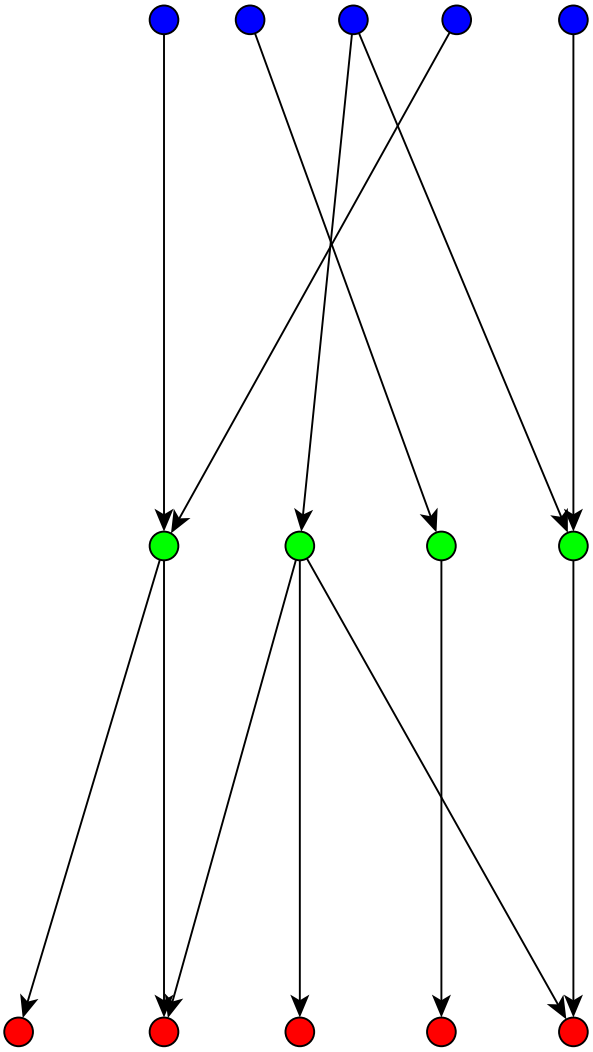


Figure 1.8: Graph with a hierarchical structure in three clusters. All edges are directed from block 1 (blue) to block 2 (green) or from block 2 to block 3 (red), hence block 1 is at the top of the hierarchy, then come block 2 and block 3.

Chapter 2

Spectral algorithm for the detection of cyclic patterns in directed graphs

The goal of this chapter and the next is to introduce new algorithms for the detection of predefined block structures in directed graphs, more specifically cyclic patterns (such as the graph depicted in figure 1 (left) in the introduction) in this chapter and acyclic patterns (such as the graph depicted in figure 1 (right) in the introduction) in next chapter. This approach is different from other approaches presented in last chapter: we expect that nodes of a directed graph can be grouped in clusters such that the connexions between clusters (namely edges joining vertices belonging to different clusters) follow a specific pattern, acyclic or cyclic. This approach can be compared to the self organizing map technique in data clustering which also requires the choice of a predefined structure for the Voronoi zones and attempts to fit this structure into the data space. Obviously, we must first make sure that the pattern is suitable for the network we analyse. For instance, it seems appropriate to detect an acyclic pattern in a trophic network of predator-prey relations. In this chapter, we give an algorithm based for the detection of cyclic patterns and in chapter 3, we extend this approach for the detection of acyclic patterns.

More specifically our algorithms are based on an appealing theory: a spectral theory of directed graphs based on a non-symmetric definition of the graph Laplacian. The techniques introduced in this chapter also offer the advantage of being efficient in terms of computational time. Moreover, the only parameter that should be carefully chosen in practice is the number of clusters.

In the first section of this chapter, a definition of the directed graph Laplacian is provided along with useful properties of this matrix. Then, we introduce two spectral clustering algorithms based on this directed version of the graph Laplacian in order to detect graphs with a cyclic pattern of connexions between clusters. We define these graphs as "block-cycles" (namely blocks of nodes that form directed cyclic structures). Next we prove a consistency theorem for our algorithms. Finally, we describe the performances of the algorithms on synthetic data.

1 Definition and properties of the directed Laplacian

In this section, we define a non-symmetric Laplacian for directed graphs based on a straightforward generalization of the Laplacian of undirected graphs. This Laplacian was introduced by Bauer (see [28] for more details). If we consider this definition of Laplacian, the generalization of all properties of spectral theory of undirected graphs is difficult. However, as we will see, a few interesting theorems give the relation between the spectrum of this non-symmetric Laplacian and some characteristics of the graph.

The structure of the section is as follows. We first give a formal definition of the Laplacian. Then, we give an expression for the region occupied by its spectrum in the complex plane. We further give the relationship between the spectrum of the Laplacian and the presence of isolated components in the graph and provide some intuitions about how this relationship might be used for the design of a spectral clustering algorithm detecting density-based clusters in a directed graphs (which is not further developed in this report). Finally, we analyse the relationship between the spectrum of the Laplacian and the presence of clusters with a cyclic pattern of connections (as was depicted in figure 2 in the introduction. This last property will provide the basis for the clustering algorithm presented in next section.

1.1 Useful concepts for the definition of the Laplacian

We consider a positive weighted graph $G = (V, E, W)$.

By convention,

$$e = (i, j) \in E \Leftrightarrow W_{ij} > 0.$$

We define the in-degree d_i^{in} and out-degree d_i^{out} of vertex $i \in V$ as

$$\begin{array}{rcl} d_i^{in} & = & \sum_{j \in V} W_{ji} \\ d_i^{out} & = & \sum_{j \in V} W_{ij} \end{array}.$$

We define the notion of isolated vertex and isolated component.

Definition 5 (*Isolated vertex*)

A vertex $i \in V$ is out-isolated (or simply isolated) if $W_{ij} = 0$ for all $j \in V$.

If weights in the graph are positive, a vertex is isolated if and only if its out-degree is zero.

Definition 6 (*Isolated component*)

A subset $S \subseteq V$ is an isolated component of G if $W_{ij} = 0$ for all $i \in S$, $j \in V \setminus S$.

In the following sections, for any subset of vertices $S \subseteq V$ of a graph $G = (V, E, W)$, we refer to $|S|$ as the number of vertices in S.

1.2 Laplacian for directed graphs

The following definition of graph Laplacian has been proposed by Frank Bauer in [28] by analogy with the undirected case. Laplacian matrix $\Delta \in \mathbb{R}^{n \times n}$ is defined in the following way.

$$\Delta_{ij} = \begin{cases} 1 - \frac{W_{ii}}{d_i^{out}} & \text{if } i = j \text{ and } d_i^{out} \neq 0 \\ -\frac{W_{ij}}{d_i^{out}} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

for all $i, j \in V$. We refer to this matrix as the *graph Laplacian* or simply as the *Laplacian* of a directed graph.

When there are no isolated vertices in the graph, the Laplacian can be rewritten as

$$\Delta = I - D^{-1}W \tag{2.1}$$

where $D = diag(d_i^{out}, i \in V)$.

An important fact regarding this definition of Laplacian operator is that it does not require the weights to be positive. In contrast, the definition of Chung's directed Laplacian given in section 3.1 of chapter 1 is based on the stationnary distribution of a random walk defined on the graph and implicitly requires positive weights. This is restrictive as some applications require the definition of positive and negative weights (such as neuronal network where the synapses may be excitatory or inhibitory [29]). In this report, we restrict ourselves to applications where weights are positive but further studies may include testing our methods on graphs with negative edge weights.

For the sake of simplicity, we sometimes use the transition matrix $P = D^{-1}W$ instead of Δ to prove some spectral properties of directed graphs. Indeed the relationship between $spec(P)$ and $spec(\Delta)$ is straightforward:

Proposition 2

The spectra of the Laplacian matrix and the transition matrix are related in the following way:

$$\lambda \in spec(P) \Leftrightarrow (1 - \lambda) \in spec(\Delta).$$

In the next sections, we give some useful properties of the directed graph Laplacian.

1.3 Region occupied by the spectrum of the Laplacian

The spectrum of the Laplacian of an undirected graph is enclosed in the interval $[0, 2]$. This property can be generalized for the Laplacian of directed graphs.

Theorem 3

Given a graph directed graph $G = (V, E, W)$ with Laplacian Δ,

$$spec(\Delta) \subset \{x \in \mathbb{C} : \|x - 1\|_2 \leq 1\}.$$

Proof: As suggested by Bauer in [28], this property follows from Gerschgorin's circle theorem applied to the directed Laplacian. ■

This property will be used in the design of our spectral clustering algorithms.

1.4 Spectrum of the Laplacian and isolated components

In the case of undirected graphs, the multiplicity of 0 as an eigenvalue of the Laplacian is equal to the number of connected components of the graph. What does the multiplicity of the

eigenvalue 0 represent in the directed case? The properties described next will not be used explicitly in the following sections but they highlight the strong relation between the spectrum of the Laplacian and the directionality of its edges. The propositions and proofs below are slight variations of the ones found in [28].

We first show that 0 is actually an eigenvalue of Δ.

Proposition 3

Given a directed graph G, 0 is an eigenvalue of the Laplacian of G.

Proof: For any directed graph $G = (V, E, W)$ with associated Laplacian Δ, we always have

$$\sum_j \Delta_{ij} \begin{cases} = & 0 \text{ if } d_i^{out} = 0 \\ = & 1 - \frac{1}{d_i^{out}} \sum_j W_{ij} = 0 \text{ otherwise} \end{cases}.$$

Hence $\Delta \mathbf{1} = \mathbf{0}$ and $\mathbf{1}$ is an eigenvector with associated eigenvalue 0. ■

Next we show that the proposition above can be generalized in the following way when weights are positive.

Proposition 4

Let $G = (V, E, W)$ be a positive weighted directed graph with associated graph Laplacian Δ. If G contains k strongly connected components $G_1 = (V_1, E_1), ..., G_k = (V_k, E_k)$ and Δ^i is the graph Laplacian of the graph defined by the $i-th$ connected component G_i, then, 0 is an eigenvalue of Δ^i if and only if G_i is an isolated strongly connected component of G, moreover, the geometric multiplicity of this eigenvalue is 1.

Proof: If G_i consists of one vertex, the proof follows directly from the definition of the Laplacian, hence we assume G_i consists of at least two vertices.

Let us first assume G_i is an isolated strongly connected component of G. Then, for any vertex $l \in V_i$, we have $d_l^{out} \neq 0$ and

$$\begin{aligned} 0 &= \textstyle\sum_j \Delta_{lj} \\ &= \textstyle\sum_{j \notin V_i} \Delta_{lj} + \sum_{j \in V_i} \Delta_{lj} \\ &= -\frac{1}{d_l^{out}} \textstyle\sum_{j \notin V_i} W_{lj} + \sum_{j \in V_i} \Delta_{lj} \\ &= \textstyle\sum_{j \in V_i} \Delta_{lj} \\ &= \textstyle\sum_{j \in V_i} \Delta_{lj}^i \end{aligned} \tag{2.2}$$

where the two last equalities follow from the fact that the component is isolated. Hence the vector of all ones is an eigenvector of Δ_i with associated eigenvalue 0. To show that the multiplicity of this eigenvalue is 1 we observe that the matrix $P^i = I - \Delta^i$ is non-negative and irreducible (as G_i is strongly connected). Hence, by Perron-Frobenius theorem, $\rho(P_i)$ (the spectral radius of P_i) is a simple eigenvalue of P_i. Moreover, the row sum P_i is equal to 1 (which can be shown with a reasoning similar to equation 2.2) which implies $\rho(P_i) \leq 1$. We conclude that 1 is a simple eigenvalue of P_i and 0 is a simple eigenvalue of Δ_i.

Let us assume conversely that G_i is a strongly connected component of G that is not isolated. Then, there exist $k \in V_i$ and $l \in V \setminus V_i$ such that $W_{kl} > 0$. Moreover, we have $d_k^{out} > 0$. Hence, the matrix Δ^i fulfils the three following characteristics:

1. for vertex k,

$$|\Delta_{kk}^i| = 1 - \frac{W_{kk}}{d_k^{out}} > \frac{1}{d_k^{out}} \sum_{j \in V_i \setminus \{k\}} w_{kj} = \sum_{j \in V_i \setminus \{k\}} |\Delta_{kj}^i|$$

2. for any vertex $j \in V_i \setminus \{k\}$,

$$|\Delta^i_{jj}| = 1 - \frac{W_{jj}}{d^{out}_j} \geq \frac{1}{d^{out}_j} \sum_{l \in V_i \setminus \{j\}} w_{jl} = \sum_{l \in V_i \setminus \{j\}} |\Delta^i_{jl}|$$

3. the matrix Δ^i is irreducible as G_i is strongly connected.

Here we make use of a theorem by Taussky (theorem 4.1 in [28]) that states that a complex $n \times n$ matrix A is non-singular if A is irreducible and $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$ with equality in at most $n-1$ cases. Hence, Δ^i is non-singular and 0 is not an eigenvalue of Δ^i. ■

This proposition is used to prove the following general result.

Theorem 4

Let $G = (V, E, W)$ be a positive weighted directed graph with associated graph Laplacian Δ. If the multiplicity of 0 as an eigenvalue of G is k then there exists at most k isolated strongly connected components in G.

In order to prove this theorem, we need the following lemma.

Lemma 1

Let $G = (V, E, W)$ be a directed graph with graph Laplacian Δ and $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{W})$ be an isolated component of G with graph Laplacian $\tilde{\Delta}$, then

$$spec(\tilde{\Delta}) \subseteq spec(\Delta).$$

Proof: Let us note $k = |\tilde{V}|$. Without loss of generality, we assume the vertices in $\tilde{G}$ are labelled from $n-k+1$ to n. As $\tilde{G}$ is isolated, $w_{ij} = 0$ for all $i \in \tilde{V}$, $j \in V \setminus \tilde{V}$. Hence, Δ can be rewritten in a block-triangular form.

$$\Delta = \begin{bmatrix} \Delta_{11} & \Delta_{12} \\ 0 & \Delta_{22} \end{bmatrix}.$$

Where the labels $n-k+1$ to k correspond to the nodes in $\tilde{V}$. As a consequence, $spec(\Delta) = spec(\Delta_{11}) \bigcup spec(\Delta_{22})$.

As $\tilde{G}$ is isolated, the out-degree of each vertex of $\tilde{V}$ is not affected by vertices in $V \setminus \tilde{V}$. Hence, Δ_{22} corresponds to the graph Laplacian $\tilde{\Delta}$ of $\tilde{G}$, which terminates the proof. ■

Proof of theorem 4: Let $G = (V, E, W)$ be a positive weighted directed graph with associated graph Laplacian Δ, that contains l connected components $G_1, ..., G_l$ with associated Laplacians $\Delta_1, ..., \Delta_l$. Then by proposition 4, 0 is a simple eigenvalue of each matrix Δ_i for $1 \leq i \leq l$. Moreover, by lemma 1,

$$\bigcup_{i=1}^{l} spec(\Delta_i) \subseteq spec(\Delta).$$

Hence, 0 is an eigenvalue of Δ of geometric multiplicity at least l, which terminates the proof. ■

One can make a parallel between theorem 4 and the multiplicity of eigenvalue zero of the Laplacian of an undirected graph. Connected components of an undirected graph are also the

isolated components of the graph as any edge is bidirectional. Hence, the theorem showing that for any undirected graph G, the multiplicity of zero as an eigenvalue of the Laplacian of G equals the number of connected components of G is a particular case of theorem 4.

The purpose of this section was simply to show how a fundamental property of the graph Laplacian can be extended from undirected graphs to directed graphs. However, we could explore the possibility of designing a clustering algorithm based on this relation between the multiplicity of eigenvalue 0 and the existence of isolated strongly connected components. We recall that in section 1.4 of chapter 1, we presented an intuitive way of understanding why spectral clustering algorithm works. Following the same intuition, let us consider a directed graph $G = (V, E, W)$ with Laplacian Δ that consists of k strongly connected components $V_1, ..., V_k \subset V$ with the particular property that all these components are isolated or, equivalently, all k strongly connected components are also weakly connected components of G. Then, theorem 4 says that the multiplicity of 0 as an eigenvalue of Δ is at least k and the corresponding eigenvectors $v_1, ..., v_k$ act as indicators of the components $V_1, ..., V_k$. Now, if we add a few directed edges connecting the components of G to form a graph $\tilde{G}$, we can hope that there will still be k eigenvalues of the Laplacian close to zero, namely k complex eigenvalues with small magnitude. If we store the associated eigenvectors $v_1, ..., v_k$ as the columns of a matrix $\Gamma \in \mathbb{R}^{|V| \times k}$, we may expect that clustering the rows of Γ recovers the components $V_1, ..., V_k$ that were isolated in the original graph G.

Hence, we could believe that such algorithm would detect density-based clusters in directed graphs, with clusters that tend to be strongly connected. To our knowledge, there is no exploration of such algorithm in the literature. However, we insist on the fact that this idea is highly heuristic and further work should be done to validate our intuition.

1.5 Spectrum of the Laplacian of a block-cycle

In this section, we define a new type of directed graph: block-cycles which can roughly be described as cycles where vertices of the cycle are replaced by groups of vertices. Figure 2.1 is a typical example of such block-cycle. We first give a formal definition of block-cycles and then we characterize the spectrum of the Laplacian of such graphs.

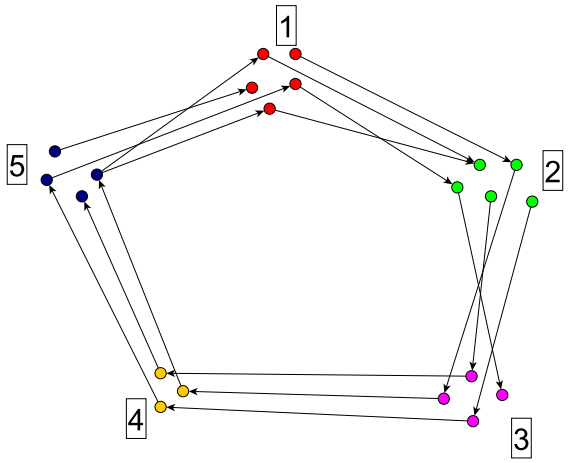


Figure 2.1: Block-cycle of 5 blocks.

Definition 7 (*Block-cycle*)

A weighted directed graph $G = (V, E, w)$ is a block-cycle if there exists a partition of V into k non-empty subsets $V_1, ..., V_k$ such that

$$E \subseteq \{(u, v) \in V \times V : u \in V_k \text{ and } v \in V_1 \text{ or } u \in V_l \text{ and } v \in V_{l+1} \text{ for some } l \in \{1, ..., k-1\}\}.$$

The subsets of vertices $V_1, ..., V_k$ are called the cyclic components of G.

The definition above is a formal description of the intuitive idea of what a block-cycle should look like: nodes are partitioned into groups, these groups are ordered and each edge connects nodes that belong to consecutive groups.

The following theorem characterizes the spectrum of block-cycles. These properties provide the basis of the spectral algorithm of block-cycle detection described in next section

Theorem 5

Let $G = (V, E, W)$ be a block-cycle with k cyclic components $V_1, ..., V_k$ such that $d_i^{out} > 0$ for all $i \in V$. Then $1 - e^{-2\pi i \frac{l}{k}} \in spec(\Delta)$ for all $0 \leq l \leq k - 1$.

Proof: For the sake of simplicity, we show instead that $e^{-2\pi i \frac{l}{k}} \in spec(P)$ for all $0 \leq l \leq k - 1$. For the l-th eigenvalue $e^{-2\pi i \frac{l}{k}}$, we consider the following eigenvector:

$$u_j^l = \begin{cases} e^{2\pi i \frac{lk}{k}} & j \in V_1 \\ e^{2\pi i \frac{l(k-1)}{k}} & j \in V_2 \\ \vdots & \\ e^{2\pi i \frac{l}{k}} & j \in V_k \end{cases}.$$

Then, $\forall j \in V_s$ with $1 \leq s \leq k - 1$:

$$[Pu^l]_j = \frac{1}{d_j^{out}} \sum_{r \in V_{s+1}} w_{jr} u_r^l.$$

If $j \in V_s$ and $r \in V_{s+1;}$, clearly, $u_r^l = e^{-2\pi i \frac{l}{k}} u_j^l$, hence

$$\begin{aligned} [Pu^l]_j &= \tfrac{1}{d_j^{out}} \textstyle\sum_{r \in V_{s+1}} w_{jr} u_r^l \\ &= \tfrac{1}{d_j^{out}} \textstyle\sum_{r \in V_{s+1}} w_{jr} e^{-2\pi i \frac{l}{k}} u_j^l \\ &= e^{-2\pi i \frac{l}{k}} u_j^l \tfrac{1}{d_j^{out}} \textstyle\sum_{r \in V_{s+1}} w_{jr} \\ &= e^{-2\pi i \frac{l}{k}} u_j^l \end{aligned}$$

which terminates the proof. ■

Hence, the Laplacian a block-cycle with k blocks has k eigenvalues that form a circle in the complex plane as shown in figure 2.2. In next section, this particular observation will provide a basis for the detection of block-cycles in directed graphs.

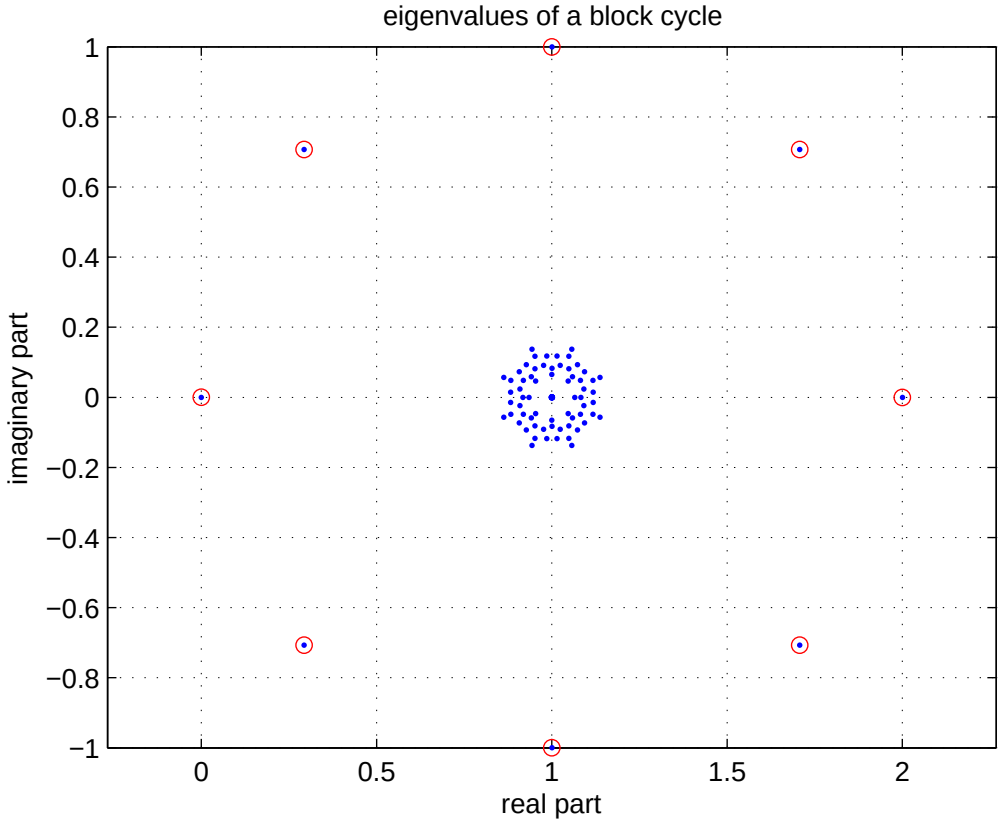


Figure 2.2: Eigenvalues of a block-cycle of 8 blocks and 100 nodes in the complex plane. Nodes are assigned with the same probability (0.125) to each of the 8 blocks and the probability of having an edge connecting two nodes belonging to consecutive blocks is 0.8. The eigenvalues that form a circle are circled in red.

2 Spectral algorithms for the detection of block-cycles

In last section, we provided the basis of a spectral theory of directed graphs based on a non-symmetric definition of the Laplacian. In the last paragraphs, we focused on the case of block-cycles and the spectral properties of such graphs. In this section, we show how these theories can be applied to design new spectral clustering algorithms.

More specifically, we wish to design algorithms that detects blocks in a block-cycle. We also expect the algorithms to work in the presence of slight perturbations, namely for a block-cycle with a few perturbing edges. We did not find any real-world networks with a block-cyclic structure. However, in next chapter, we show how the block-cycle detection algorithms may be used to detect other types of structures.

This section is organized as follows. We first formulate the problem of block-cycle detection as an optimization problem and show this problem is NP-complete in the general case. Secondly, we provide the theoretical basis of our algorithms and we give a formal description of them. Then we describe some extensions of these algorithms including some technical details or methods that may improve the solution returned by our algorithms. Finally, we analyse the performances of the algorithms on synthetic data.

2.1 An optimization problem for the detection of block-cycles

Given a block-cycle $G = (V, E, W)$, we address the problem of finding the block membership of each node. However, we also want to be able to recover the blocks of a slightly perturbed block-cycle, namely a block-cycle with a few perturbing edges. Formally, given an integer k, we wish to find a block membership function $f : V \to \{1, ..., k\}$ that maximizes a certain measure of the cyclicity of the graph when nodes are partitioned into blocks according to f. This can be stated as the following optimization problem:

$$\max_{f:V\to\{1,...,k\}} C(f)$$

where C is the measure of cyclicity. Several definitions of criterion $C(f)$ could be proposed depending on what kind of cycle we wish to detect. We choose what seems to be an intuitive measure of the cyclicity of the graph:

$$C(f) = \sum_{(u,v)\in E\cup D(f)} W_{uv} \tag{2.3}$$

where $D(f) = \{(u, v) \in V \times V : f(u) + 1 = f(v) \text{ or } (f(u), f(v)) = (k, 1)\}$.

Maximization of this criterion yields a partition of nodes into an ordered sequence of blocks such that most of the edges of the graph connect nodes belonging to consecutive blocks of the cycle.

We prove that this problem is NP-complete as shown by propositions 5 and 6. We propose a reduction from the partition problem. We first formulate both problems as decision problems.

Definition 8 (*Max-Cycle problem* $MC(G = (V, E, W), m, k)$)

Given a weighted directed graph $G = (V, E, W)$, $m \in \mathbb{R}^+$ and a positive integer $k \leq |V|$, is there a function $f : V \to \{1, ..., k\}$ partitioning nodes into k blocks such that $C(f) \geq m$?

Definition 9 (*Partition problem P(S)*)

Given a multiset of positive integers S is there a partition of S into two subsets S_1 and S_2 such that the sum of the elements S_1 is equal to the sum of elements in S_2.

Proposition 5

Max-Cycle is in NP.

Proof: Given a weighted directed graph $G = (V, E, W)$, $m \in \mathbb{R}^+$, $k \leq |V|$ and a partition of nodes into k blocks $f : V \to \{1, ..., k\}$, going through all edges, we may calculate $C(f)$ in polynomial time (which is done in $O(|E|)$ if accessing the block membership $f(u)$ of a node $u \in V$ can be done in constant time). Hence the problem is in NP. ■

Proposition 6

Max-Cycle is NP-hard.

Proof: We reduce the partition problem to the Max-cycle problem. Given an instance $P(S = (a_1, ..., a_n))$ of partition problem, we build an instance of max-cycle problem with the following parameters:

1. weighted graph $G = (V, E, W)$ such that $V = \{1, ..., n\}$, $E = V \times V$ and $w_{ij} = a_i a_j$ for all $i, j \in V$,
2. $k = 2$,
3. $m = \frac{1}{2}(\sum_{i=1}^{n} a_i)^2$.

Let us first assume $P(S)$ is a positive instance of partition problem, then there exists a partition of S into $S_1 = \{a_{i_1}, ..., a_{i_s}\}$ and $S_2 = \{a_{j_1}, ..., a_{j_t}\}$ such that

$$\sum_{l=1}^{s} a_{i_l} = \sum_{l=1}^{t} a_{j_l} = \frac{1}{2} \sum_{i=1}^{n} a_i.$$

We define a block membership function $f : V \to \{1, 2\}$ such that for any $u \in V$,

$$f(u) \begin{cases} = & 1 \text{ if } u \in \{i_1, ..., i_s\} \\ = & 2 \text{ otherwise} \end{cases}.$$

The associated criterion is given by

$$\begin{array}{rcl} C(f) & = & 2 \sum_{p=1}^{s} \sum_{q=1}^{t} a_{i_p} a_{j_q} \\ & = & 2 \sum_{p=1}^{s} a_{i_p} \sum_{q=1}^{t} a_{j_q} \\ & = & 2 \frac{1}{2} \sum_{i=1}^{n} a_i \frac{1}{2} \sum_{i=1}^{n} a_i \\ & = & \frac{1}{2} (\sum_{i=1}^{n} a_i)^2 \end{array}.$$

hence $C(f) \geq m$ and $MC(G, m, k)$ is a positive instance of maximum cycle problem.

Conversely let us assume $MC(G, m, k)$ is a positive instance of maximum cycle problem. There exists a partition $f : \{1, ..., n\} \to \{1, 2\}$ into two blocks such that $C(f) \geq \frac{1}{2}(\sum_{i=1}^{n} a_i)^2$. We introduce the following notations:

$$\begin{array}{rcl} I_1 & = & \{i \in \{1, ..., n\} : f(i) = 1\} \\ I_2 & = & \{i \in \{1, ..., n\} : f(i) = 2\} \\ S_1 & = & \sum_{i \in I_1} a_i \\ S_2 & = & \sum_{i \in I_2} a_i \\ S_t & = & \sum_{i=1}^{n} a_i \end{array}.$$

Clearly,

$$S_2 = S_t - S_1.$$

The relation between S_t, S_1, S_2 and $C(f)$ is given by

$$\begin{array}{rcl} C(f) & = & 2\sum_{i\in I_1}\sum_{j\in I_2} a_i a_j \\ & = & 2S_1S_2 \\ & = & 2S_1(S_t - S_1) \\ & \leq & \frac{S_t^2}{2} = \frac{1}{2}(\sum_{i=1}^n a_i)^2 \end{array} .$$

equality is verified only if $S_1 = \frac{S_t}{2}$ namely if $S_1 = S_2 = \frac{S_t}{2}$. Hence $P(S)$ is a positive instance of partition problem. Moreover, the reduction we proposed is clearly polynomial in n which terminates the proof. ■

In the following sections, we often use a relative measure of the criterion, namely

$$C_r(f) = \frac{C(f)}{\sum_i \sum_j W_{ij}} \tag{2.4}$$

The spectral method described in next section is expected to find a block membership function that is a good approximation of the one maximizing the criterion. We test the efficiency of our method in optimizing the criterion in section 4.

2.2 Theoretical foundation of the spectral detection of block-cycles

In this section, we give the theoretical foundations of the detection of a block-cyclic structure in directed graphs.

Firstly let us consider the simplest case of an unperturbed block-cycle $G = (V, E, W)$. We assume that the number k of blocks is known and we want to find a block membership function $f : V \to \{1, ..., k\}$ such that $C_r(f) = 1$. Two observations provide the basis of a spectral algorithm for the detection of the blocks, these observations are both based on theorem 5:

1. $1 - e^{-2\pi i \frac{l}{k}} \in spec(\Delta)$ for all $0 \leq l \leq k-1$ where Δ is the Laplacian of G,
2. the eigenvector associated to eigenvalue $1 - e^{-2\pi i \frac{l}{k}}$ is

$$u_j^l = \begin{cases} e^{2\pi i \frac{lk}{k}} & j \in V_1 \\ e^{2\pi i \frac{l(k-1)}{k}} & j \in V_2 \\ \vdots & \\ e^{2\pi i \frac{l}{k}} & j \in V_k \end{cases} .$$

We denote these particular eigenvalues and eigenvectors as the *cycle eigenvalues* and the *cycle eigenvectors* of G. Based on these two notions, we design two simple procedures to recover the block membership of each node. These two methods are referred to as the *Multiple Cycle Eigenvalues* (or MCE) clustering algorithm and the *Single Cycle Eigenvalue* (or SCE) clustering algorithm.

MCE clustering algorithm

The first method we suggest for the detection of blocks in a block-cycle consists of the following steps:

1. find the k cycle eigenvectors, store them as the columns of a matrix $\Gamma \in \mathbb{C}^{n\times k}$,

2. cluster each row $y \in C^k$ of Γ with k-means algorithm based on Euclidean distance,

3. assign nodes to blocks according to the clustering of each row of V,

This algorithm is based on the observation that for a block-cycle $G = (V, E, W)$,

$$\text{vertices } i, j \in \{1, ..., n\} \text{ belong to the same block } \Leftrightarrow \Gamma_{i*} = \Gamma_{j*}$$

where Γ_{i*} denotes the i-th row of Γ.

For instance, in the case $n = 10$, $k = 3$ and blocks $C_1 = \{1, 2\}$, $C_2 = \{3, 4, 5, 6, 7\}$ and $C_3 = \{8, 9, 10\}$, Γ has the form

$$\Gamma = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ e^{\frac{4\pi}{3}i} & e^{\frac{2\pi}{3}i} & 1 \\ e^{\frac{4\pi}{3}i} & e^{\frac{2\pi}{3}i} & 1 \\ e^{\frac{4\pi}{3}i} & e^{\frac{2\pi}{3}i} & 1 \\ e^{\frac{4\pi}{3}i} & e^{\frac{2\pi}{3}i} & 1 \\ e^{\frac{4\pi}{3}i} & e^{\frac{2\pi}{3}i} & 1 \\ e^{\frac{2\pi}{3}i} & e^{\frac{4\pi}{3}i} & 1 \\ e^{\frac{2\pi}{3}i} & e^{\frac{4\pi}{3}i} & 1 \\ e^{\frac{2\pi}{3}i} & e^{\frac{4\pi}{3}i} & 1 \end{bmatrix}.$$

One can see that rows corresponding to nodes that lie in the same block are identical. Hence, in slightly perturbed cases, we expect the rows to stay similar to each other and this similarity is captured by the steps of MCE algorithm where rows of Γ are clustered. A parallel can be made between this approach and the one presented at section 1.4 of chapter 1 for undirected graphs.

SCE clustering algorithm

The second method is only based on one eigenvector of the Laplacian of G. If $G = (V, E, W)$ is a block-cycle with Laplacian Δ, then $1 - e^{-i\frac{2\pi}{k}} \in spec(\Delta)$ and the associated eigenvector is:

$$u_j = \begin{cases} e^{2\pi i \frac{k}{k}} & j \in V_1 \\ e^{2\pi i \frac{(k-1)}{k}} & j \in V_2 \\ \vdots & \\ e^{2\pi i \frac{1}{k}} & j \in V_k \end{cases}.$$

We call this eigenvalue and the associated eigenvector *first cycle eigenvalue* and *first cycle eigenvector*.

This suggests the following method for the detection of blocks in a block-cycle,

1. find the eigenvector associated to the eigenvalue $1 - e^{-\frac{2\pi i}{k}}$,

2. cluster the (complex) components of this eigenvector according to their phase,

3. assign nodes to blocks according to the clustering of component.

As an illustration, let us consider the same case as for MCE algorithm, namely $n = 10$, $k = 3$ and blocks $C_1 = \{1, 2\}$, $C_2 = \{3, 4, 5, 6, 7\}$ and $C_3 = \{8, 9, 10\}$, then the first cycle eigenvector is

$$u = \begin{bmatrix} 1 \\ 1 \\ 1 \\ e^{\frac{2\pi}{3}i} \\ e^{\frac{2\pi}{3}i} \\ e^{\frac{2\pi}{3}i} \\ e^{\frac{2\pi}{3}i} \\ e^{\frac{2\pi}{3}i} \\ e^{\frac{4\pi}{3}i} \\ e^{\frac{4\pi}{3}i} \\ e^{\frac{4\pi}{3}i} \end{bmatrix}.$$

We see that components of u corresponding to the same blocks are identical, moreover only the phase of components is changed from one block to the other. This justifies the second step of SCE clustering algorithm where components of the first cycle eigenvector are clustered according to their phase. A parallel can be made between this method and the method based on random walks described in section 3.3 of chapter 1 where clusters are also identified by exploiting the piecewise constant form of the eigenvector associated to the second smallest eigenvalue in magnitude. However the algorithm of section 3.3 is expected to detect density-based clusters in a directed graphs whereas SCE clustering algorithms detects blocks in block-cycle.

What happens if we add small perturbations to the perfect block-cycle (namely, if the graph has still a global cyclic structure but if this structure is slightly perturbed)? Observations show that both algorithms give satisfactory results: the positions of the cycle eigenvalues are slightly perturbed but they stay at the boundary of the region occupied by the eigenvalues of Δ and there is still an eigenvalue that is close to the first cycle eigenvalue $1 - e^{-\frac{2\pi i}{k}}$. Further details about the sensitivity of the algorithms to perturbations are given in sections 3 (for a theorem of consistency) and 4 (for tests on synthetic data).

Figures 2.3 and 2.4 show the behaviour of the eigenvalues in the complex plane for different directed graphs with an underlying cyclic structure. In each case, all blocks contain an equal number of nodes. Figures 2.3 and 2.4 respectively illustrate the case of a non-perturbed block-cycle and a block-cycle perturbed with random edges. In both cases, the complex eigenvalues of the Laplacian are displayed in the complex plane and the first cycle eigenvalue is circled in red (while the other cycle eigenvalues form a cicle in the complex plane). We also display the components of the first cycle eigenvector (colouring of points correspond to the block membership of nodes estimated by SCE algorithm).

These two figures give the intuition behind SCE clustering algorithm for the detection of block-cycles, namely, finding the first cycle eigenvalue and clustering the components of the first cycle eigenvector. It is harder to illustrate MCE clustering algorithm with a graphical visualization as it involves clustering n points in $\mathbb{C}^k$ but the intuition is still the same: clustering the component of cycle eigenvectors gives an estimation of the block membership function.

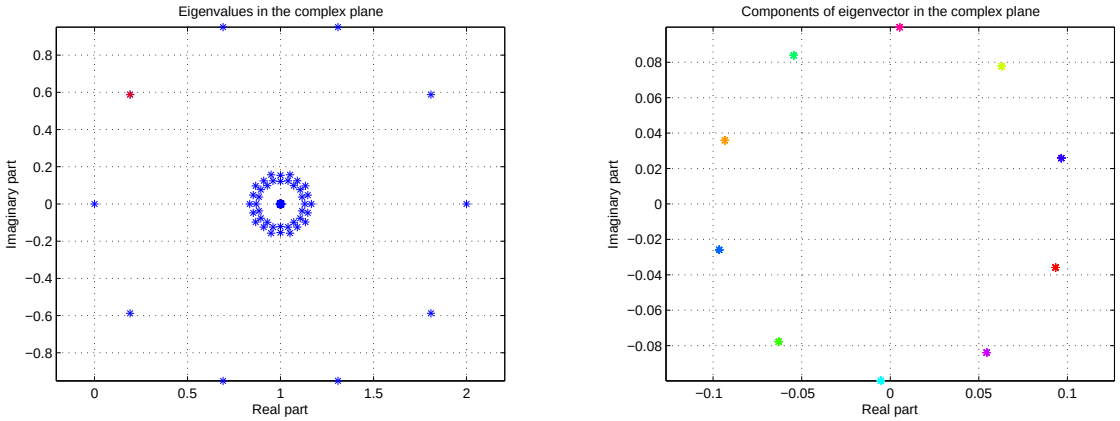


Figure 2.3: For a block-cycle of 10 blocks $V_1, ..., V_{10}$ with probability 0.8 of having an edge (u, v) for u and v belonging to consecutive blocks: (left, figure 2.3(a)) eigenvalues of the Laplacian in the complex plane (the first cycle eigenvalue is in red) and (right, figure 2.3(b)) components of the first cycle eigenvector in the complex plane (colouring of points correspond to the result of SCE clustering algorithm).

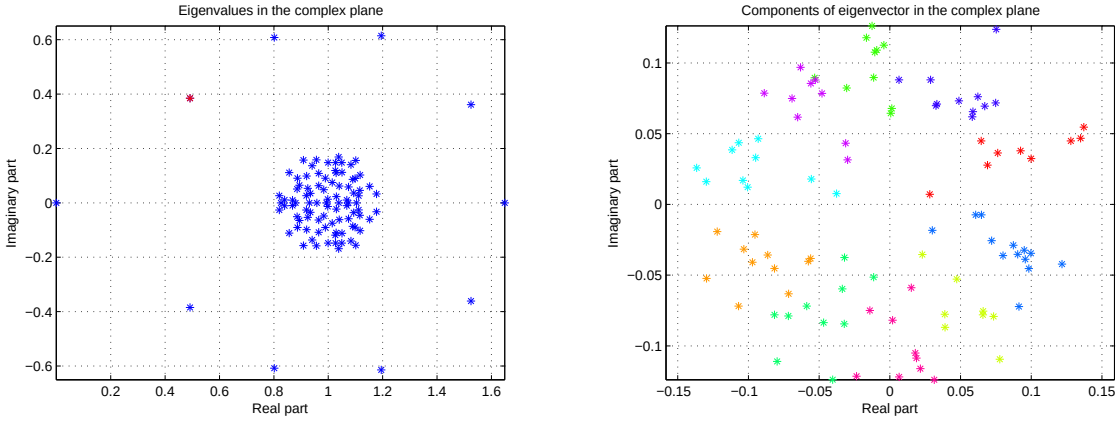


Figure 2.4: Perturbed block-cycle with adjacency matrix $W + \tilde{W}$: W is the adjacency matrix of a block-cycle of 10 blocks $V_1, ..., V_{10}$ with probability 0.8 of having an edge (u, v) for u and v belonging to consecutive blocks, $\tilde{W}$ is a random matrix of zeros and ones with probability 0.05 for any entry to be equal to 1: (left, figure 2.4(b)) eigenvalues of the Laplacian in the complex plane (the first cycle eigenvalue is in red) and (right, figure 2.4(b)) components of the first cycle eigenvector in the complex plane (colouring of points correspond to the result of SCE clustering algorithm).

2.3 Spectral clustering algorithm for the detection of block-cycles

In this section, we formalize the intuitive method presented in last section and formulate two spectral clustering algorithms for the detection of block-cycles.

Algorithm 2.1: MCE clustering algorithm

Input: Adjacency matrix $W \in \{0,1\}^{n\times n}$;
Parameters: $k \in \{2,3,...,n\}$;
Step 1: Compute the graph Laplacian $L = I - D^{-1}W$;
Step 2: Find the k cycle eigenvalues (the k eigenvalues located furthest from $(1,0)$ in the complex plane) and store the associated cycle eigenvectors as the columns of a matrix $\Gamma \in \mathbb{C}^{n\times k}$.;
Step 3: Consider each row of Γ as a point in $\mathbb{C}^k$ and cluster these points using a k-means algorithm. Let $\phi : \{1,...,n\} \to \{1,...,k\}$ be the function assigning each row of Γ to a cluster;
Step 4: Compute the estimation of block membership function f: $f(u) = \phi(u)$ for all $u \in \{1,...,n\}$;
Output: estimation of block membership f;

Steps 2 and 3 are the key steps of the algorithm. In step 2, we compute the cycle eigenvalues. Choosing the k eigenvalues located furthest from the point $(1,0)$ in the complex plane is a rather intuitive method for finding the cycle eigenvalues that is based on the two following observations:

- cycle eigenvalues tend to be located at the boundary of the region occupied by eigenvalues in the complex plan, even for slightly perturbed block-cycles as we observe in figure 2.4,
- the region occupied by the eigenvalues of the Laplacian of a graph is always enclosed in the circle of radius 1 centered in $(1,0)$ in the complex plane (theorem 3 of this chapter).

Refer to the benchkmark tests for more details about the efficiency of this method. In step 3, the rows of the matrix Γ of cycle eigenvectors are clustered using k-means algorithm which seems to be the natural clustering method by analogy with spectral clustering algorithm for undirected graphs.

Algorithm 2.2: SCE clustering algorithm

Input: Adjacency matrix $W \in \{0,1\}^{n\times n}$;
Parameters: $k \in \{2,3,...,n\}$;
Step 1: Compute the graph Laplacian $L = I - D^{-1}W$;
Step 2: Find the first cycle eigenvalue (among the k eigenvalues of located furthest from $(1,0)$, choose the one nonzero imaginary part and smallest norm) and the associated first cycle eigenvector V;
Step 3: Cluster the components of V using a k-means algorithm with cosine distance. Let $\phi : \{1,...,n\} \to \{1,...,k\}$ be the function assigning each component of V to a cluster;
Step 4: Compute the estimation of block membership function f: $f(u) = \phi(u)$ for all $u \in \{1,...,n\}$;
Output: estimation of block membership f;

As for the MCE clustering algorithm, steps 2 and 3 are the key steps of the algorithm. In step 2, we compute the cycle eigenvalue (corresponding to the eigenvalue $1 - e^{-\frac{2\pi i}{k}}$ for an unperturbed block-cycle). To find the first cycle eigenvalue, we suggest to compute the k eigenvalues of located furthest from $(1,0)$, choose the one nonzero imaginary part and smallest norm. This is a heuristic method that allows to compute the cycle eigenvalue efficiently even in perturbed

case, as we can observe on figure 2.4[1]. In step 3, the components of the cycle eigenvector are clustered according to their phase: again, by analogy with the classical spectral clustering algorithm, k-means seems to be the natural clustering method of doing this, however we use cosine distance instead of Euclidean distance to only take the difference in phase between points into account.

For a comparison of efficiency of MCE clustering and SCE clustering algorithms based on tests of performances on perturbed block-cycles, we refer to section 4. However, we can already make a qualitative comparison of the two methods.

- MCE clustering algorithm requires finding k eigenvalues. Depending on the numerical method we choose to achieve this, computing k eigenvalues could be more expensive in time and memory than computing only one eigenvalue as does the SCE clustering algorithm. However, using our heuristic, finding the first cycle eigenvalues also requires to find the k eigenvalues located furthest from $(1, 0)$.
- SCE clustering algorithm is based on the clustering of points in a 2-dimensional space (ie the complex components of the first cycle eigenvector). Hence, SCE clustering algorithm could be interpreted as a projection of vertices in a 2-dimensional space where the projection of vertices belonging to the same blocks are close to each other in the 2-dimensional space. Thus, one can easily visualize the clustering step of SCE clustering method. In contrast, MCE clustering algorithm is based on the clustering of points in $\mathbb{C}^k$ which cannot be visualized in the plane.

Some questions still arise regarding the efficiency of this algorithm.

1. How do we recover the order in which blocks appear in the block-cycle?
2. How do we compute the eigenvalues efficiently at step 2 of both algorithms (especially for MCE clustering algorithm)?
3. How do we implement the k-means algorithm of step 3 of both algorithms?
4. How do we guarantee the quality of the result when the components of the cycle eigenvector do not cluster in a clear way for both algorithms (such as in figure 2.4 for SCE clustering algorithm)?

These four questions are treated in section 2.4.

2.4 Further developments of the algorithms

In this section, we treat the problems described at the end of section 2.3. As already mentioned, the problem of the detection of the number of blocks is not treated in this paper, hence the number k of blocks is a parameter of our algorithms.

2.4.1 Recovering the positions of blocks in the cycle

Let us consider the block-cycle of figure 2.5. In last section, we showed how to cluster the nodes and recover their block membership. However, we did not explain how to label the blocks: for the example of figure 2.4, we know how to classify nodes into five blocks but this does not give the order in which blocks appear if we go around the cycle (i.e. block 1 comes first, then block

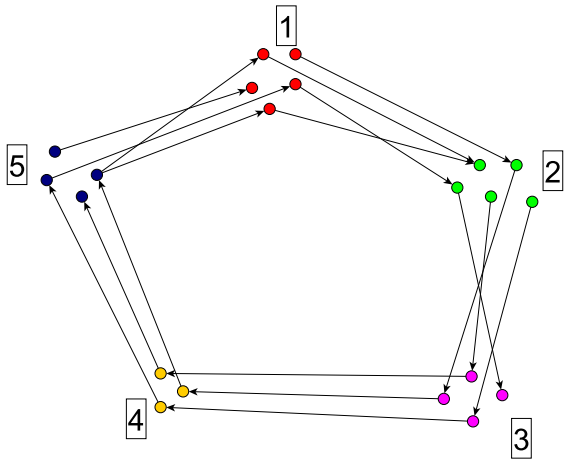


Figure 2.5: Block-cycle of 5 blocks labelled from 1 to 5.

2, then block 3, etc.).

To recover the ordering of blocks, we use the components of the first cycle eigenvector and we look at the "average phase" of each block, namely the phase of the average (or centroid) of the components associated to each block: if we order blocks with respect to the phase of their respective centroids, we recover the ordering of blocks in the block-cycle. This is a direct consequence of the definition of the first cycle eigenvector given in section 2.2. As an illustration, figure 2.6 displays the components of the first cycle eigenvector for a slightly perturbed block-cycle of five blocks and a hundred nodes.

Based on these observations, we formulate the following algorithm in order to recover the ordering of blocks in the cycle from the block membership function $f : V \to \{1, ..., k\}$ and the first cycle eigenvector $u = [u_1, ..., u_n]^T \in \mathbb{C}^n$.

Algorithm 2.3: Cyclic neighbours algorithm

Input: $f : V \to \{1, ..., k\}$, $u = [u_1, ..., u_n]^T$;
Step 1: Compute centroids: for $A_l = \{i : f(i) = l\}$,

$$c_l \leftarrow \frac{1}{|A_l|} \sum_{i \in A_l} u_i$$

for $1 \leq l \leq k$;
Step 2: $p_l \leftarrow \angle c_l$, $1 \leq l \leq k$;
Step 3: sort the components of vector p in ascending order and relabel the blocks from 1 to k according to this ordering;
Output: block membership function $f : V \to \{1, ..., k\}$ with relabelled blocks;

We note that this algorithm may be used to detect the ordering of blocks for the output of both MCE and SCE algorithm.

[1]Refer to section 4 about benchkmark tests for more details about the efficiency of this method.

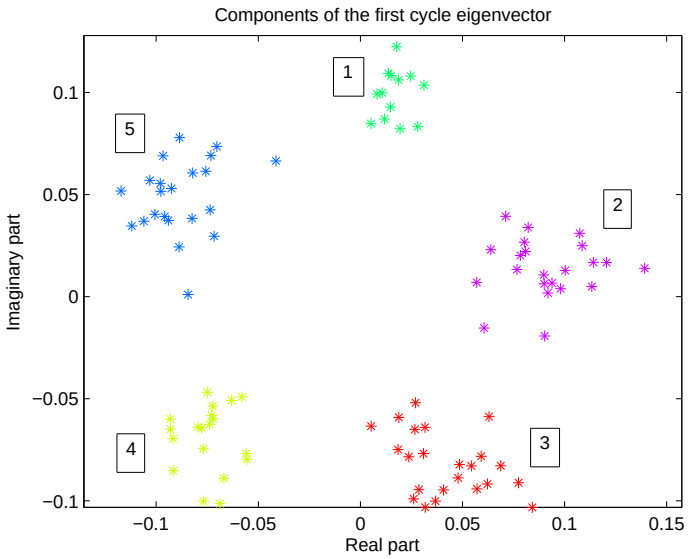


Figure 2.6: Components of the cycle eigenvectors for a slightly perturbed block-cycle of five blocks and a hundred nodes. The order of appearance of blocks in the cycle is $1, 2, 3, 4, 5$. This corresponds to the ordering of the phases of the centroid of each cluster in the complex plane.

2.4.2 Computing the cycle eigenvalues

A major drawback of spectral clustering algorithms (and spectral methods in general) is the high cost of computing eigenvalues of large matrices. However, in the case of MCE and SCE algorithms it potentially more complicate than in the case of classical spectral clustering algorithms for undirected graphs. Indeed, the Laplacian matrix of undirected graphs is symmetric positive definite and efficient algorithms exist for the computation of the extremal eigenvalues of such matrices (such as Lanczos algorithm for instance), in contrast our Laplacian matrix is not symmetric. We could use a built-in function such as the function `eigs.m` of `Matlab`. However, we want our method to be independent of such built-in function. The method we propose here can be implemented using any other software and it allows to compute the cycle eigenvalues of large graphs containing up to a hundred thousand nodes and ten millions of edges on a single computer in less than thirty seconds when implemented in `Matlab`.

We consider the problem of finding the cycle eigenvalues of the Laplacian Δ of a block-cycle. In preceding section we suggested that these eigenvalues should be chosen as the k eigenvalues of Δ located furthest from the point $(1, 0)$ in the complex plane. For the sake simplicity we can equivalently seek the k eigenvalues of $P = I - \Delta$ of largest magnitude.

We suggest to use Arnoldi's method for the computation of these eigenvalues. Roughly speaking, this method estimates the eigenvalues and eigenvectors of a matrix A by computing the spectrum of the orthogonal projection of A on an associated Krylov subspace (the complete pseudo-code of Arnoldi's algorithm is available in appendix 1). Arnoldi's algorithm has remarkable performances when it comes to compute eigenvalues at the boundary of the spectrum (typically eigenvalues with largest magnitude).

We denote Arnoldi's algorithm by $Arnoldi(A, m, v_0)$. The inputs of the algorithm are parameters A, m and v_0 which are set in the following way.

- m the dimension of the Krylov subspace $K_m(A, v_0)$ we consider: we do not suggest any particular method for a good estimation of m except that it must be larger than k. However, rather small values of m seem to give results that are as good as the ones

provided by `Matlab`'s `eigs.m` function (typically, we suggest $m = 20$ for $k = 10$, or, in general, $m = 2k$).

- v_0 the initial vector defining the Krylov subspace $K_m(A, v_0)$: this parameter is often hard to set, a good choice for it is the eigenvector associated to the largest eigenvalue in magnitude. In our case, we know that the vector of all ones is an eigenvector of P with eigenvalue 1, which is one of the largest eigenvalues in magnitude. We choose therefore $v_0 = [1, ..., 1]^T$. Knowing the exact value of this eigenvector makes the algorithm very efficient even for small values of m.

The outputs of Arnoldi's algorithm are an $m \times m$ Hessenberg matrix (from which we compute the estimation of eigenvalues and eigenvectors) and an $n \times m$ matrix U whose columns form an orthogonal basis of the Krylov subspace $K_m(A, v_0)$.

Given the transition matrix $P = I - \Delta$ of a directed graph, the full algorithm that computes the cycle eigenvalues is the following.

Algorithm 2.4: Computation of cycle eigenvalues and eigenvectors

Input: P, m;
Step 1: $[H, U] \leftarrow Arnoldi(P, m, [1, ..., 1]^T)$ where U is an orthogonal basis of $K_m(P, [1, ..., 1]^T)$ and H is the projection of P on $K_m(P, [1, ..., 1]^T)$;
Step 2: Compute $spec(H)$;
Step 3: Extract the set of k eigenvalues with largest magnitude $\tilde{\Lambda} \subseteq spec(H)$ and compute the associated eigenvectors, store them as the columns of a matrix V;
Step 4: Compute the cycle eigenvectors in the original space $\tilde{V} = UV$;
Output: cycle eigenvalues and eigenvectors $(\tilde{\Lambda}, \tilde{V})$.

The computation of the eigenvalues of H at step 2 can be done using QR-algorithm or `Matlab`'s `eig.m` function (this choice is not essential as matrix H has low dimension $m \times m$ and the computation of its eigenvalues is cheap).

This algorithm computes the cycle eigenvalues and eigenvectors needed for MCE clustering algorithm. We still need to show how to extract the first cycle eigenvalue and first cycle eigenvector required for SCE clustering algorithm. We run algorithm 2.4 to compute $\tilde{\Lambda}$ and $\tilde{V}$, then we extract the first cycle eigenvalue $\tilde{\lambda}$ by solving

$$\tilde{\lambda} = \arg\min_{\substack{\lambda \\ \lambda \in \tilde{\Lambda} \\ Im(\lambda) > \epsilon}} |1 - \lambda|$$

where the last condition ensures that $\tilde{\lambda}$ has nonzero imaginary part. Finally, the first cycle eigenvector is $U\tilde{u}$ where $\tilde{u}$ is the eigenvector associated to eigenvalue $\tilde{\lambda}$.

We do not give any theoretical guarantee that this algorithm always finds the eigenvalues we are looking for. However, tests on synthetic data show that it gives a good estimation of the cycle eigenvector for our purpose. Figure 2.7 shows, for a slightly perturbed block-cycle, the spectrum of the Laplacian found using `eigs.m` function, superposed with the cycle eigenvalues found by algorithm 2.4 and the components of the estimated first cycle eigenvector obtained by using `eigs.m` function superposed with the one found using algorithm 2.4. `eigs.m` and algorithm 2.4 find very similar values for the cycle eigenvalues and the components of the cycle eigenvectors tend to cluster according to their phases in both cases.

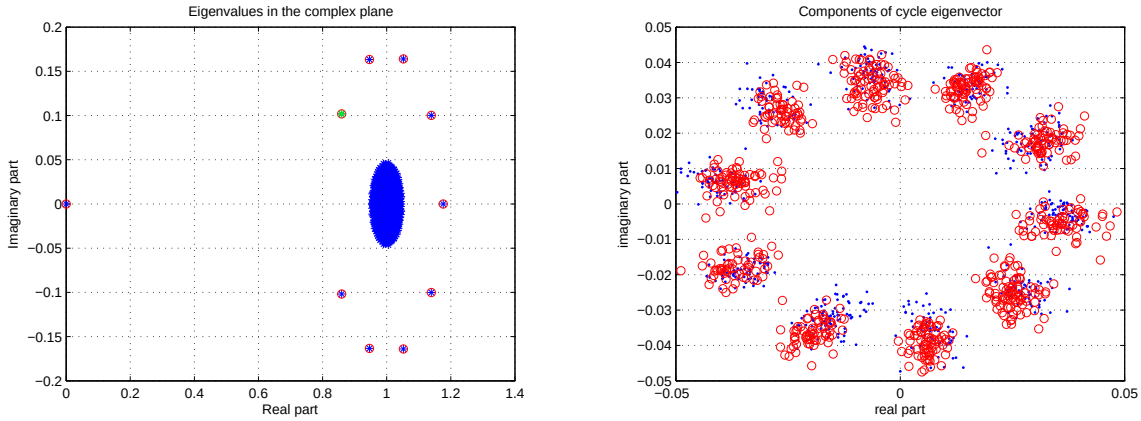


Figure 2.7: For a block-cycle of $n = 700$ nodes partitioned into 10 blocks $V_1, ..., V_{10}$ with probability 0.7 of having an edge (u, v) for u and v belonging to consecutive blocks, the block-cycle is perturbed by randomly adding $5m$ edges where m is the number of edges in the unperturbed block-cycle: (left, figure 2.7(a)) eigenvalues of the Laplacian in the complex plane (in blue, eigenvalues of the Laplacian obtained by `eig.m`, in red, the cycle eigenvalues obtained using Arnoldi's algorithm) and (right, figure 2.7(b)) components of the first cycle eigenvector in the complex plane (in blue the components of the first cycle eigenvector computed by `eigs.m` and in red, the components of the first cycle eigenvector computed by Arnoldi's algorithm).

The purpose of this section was essentially to provide the basis of an efficient algorithm for finding the cycle eigenvalues and cycle eigenvectors, further theoretical development (eg. about the convergence of Arnoldi's algorithm) are beyond the scope of this report (refer to [34] for more details about Arnoldi's algorithm).

2.4.3 Clustering the components of the cycle eigenvectors

Both MCE clustering and SCE clustering algorithms involve clustering points in the complex plane. In the former, we suggested to use a k-means method to partition n points in $\mathbb{C}^k$ into k clusters. In the latter, n points in $\mathbb{C}$ should be clustered according to their phase (namely using angle or cosine distance instead of classical Euclidean distance). In this section, we discuss what k-means related algorithm should be used in both situations.

General k-means problem (based on distance function d in $\mathbb{C}^m$) can be stated as follows:

Definition 10 (*k-means problem*)

Given a set of points $S = \{p_1, ..., p_n\} \subset \mathbb{C}^m$ *and an integer* k
Find a set of points $C = \{c_1, ..., c_k\} \in \mathbb{C}^m$ *such that*

$$\sum_{i=1}^{N} \left(\min_{1 \leq j \leq k} d(p_i, c_j) \right)^2$$

is minimized

K-means problem is NP-hard, even in dimension 2 (see [31] for a proof of NP-hardness in the planar case). Hence, there are heuristic algorithms such as Lloyd's algorithm and some variants. However, in dimension 1, there exists a polynomial algorithm solving k-means problem exactly (see [32] for instance). In the rest of this section, we first give a variant of Lloyd's heuristic algorithm that can be used for both MCE clustering and SCE clustering algorithms and then

we give a dynamic programming algorithm extending the algorithm in [32] to the case of SCE clustering algorithm, finally we compare both algorithms and argue which method should be preferred for SCE clustering algorithm.

Method 1: Lloyd's algorithm

Lloyd's algorithm is an iterative way of solving k-means problem. There are several variants for Lloyd's algorithm. The following version has the advantage of guaranteeing a strict decrease of the objective function at each iteration. We formulate the algorithm for a general distance function d on $\mathbb{C}^m$.

Algorithm 2.5: Lloyd's Algorithm

Input: $X = \{x_1, ..., x_n\} \subset \mathbb{C}^m$, $k \in \mathbb{Z}_0^+$;
Initialization: choose the k points in X maximizing the sum of pairwise distances, label them as $c_1, ..., c_k \in \mathbb{C}^m$ (coordinates of centroids);
Iterations: until centroids do not move significantly from one step to the next
Step 1: for $1 \leq j \leq n$,

$$f(j) \leftarrow \arg\min_{1 \leq i \leq k} d(x_j, c_i)$$

Step 2: let $A_i = \{x_l,\ f(l) = i\}$ and for $1 \leq i \leq k$

$$c_i \leftarrow \frac{1}{|A_i|} \sum_{x \in A_i} x$$

Output: cluster membership function f

Some comments are necessary regarding this algorithm.

Firstly, we must define the distance function that should be used.

For MCE clustering algorithm, the distance function we use is

$$\forall x, y \in \mathbb{C}^m : d(x, y) = \sqrt{(x-y)^*(x-y)}$$

where v^* denotes the adjoint (conjugate transpose) of a vector $v \in \mathbb{C}^m$. For SCE clustering algorithm, the distance function is the cosine distance noted d_c[2].

$$\forall x, y \in \mathbb{C} \setminus \{0\} : d_c(x, y) = 1 - \frac{Re(x)Re(y) + Im(x)Im(y)}{|x|.|y|}.$$

Hence, when cosine distance is used, the algorithm somehow mixes cosine and Euclidean distances (cosine distance is used at step 1 and Euclidean distance is implicitly used at step 2 when positions of centroids are chosen as the geometric centres of the Voronoi zones). We could instead only let the phases of centroids vary and fix their norm to 1. However, using geometric centres better takes the spreading of points within clusters into account (this spreading occurs both in term of phase and norm for a perturbed block-cycle).

[2]We note that this is not a proper distance but rather a measure of the cosine of the difference in angle between two points x, y in $\mathbb{C}$. Hence it is not defined if $x = 0$ or $y = 0$.

Secondly, the initialization step is crucial. Instead of choosing initial centroids randomly in the input data, we choose the k data points maximizing the sum of pairwise distances[3] which gives good convergence results in practice (see [33] for more details about k-means initialization).

Thirdly, one must mention a difficulty that might occur with such implementation of Lloyd's algorithm: we may end up with an empty cluster. A way to avoid this is to rather use the following algorithm.

Algorithm 2.6: Lloyd's algorithm with frequency sensitive learning

Input: $X = \{x_1, ..., x_n\} \subset \mathbb{C}^m$, $k \in \mathbb{Z}_0^+$;
Initialization: choose the k points in X maximizing the sum of pairwise distances, label them as $c_1, ..., c_k \in \mathbb{C}^m$ (coordinates of centroids), initialize centroids activity vector:

$$a \leftarrow [1, ..., 1]^T$$

;
Iterations: until centroids do not move significantly from one step to the next
Step 1: for $1 \leq j \leq n$,

$$\begin{aligned} f(j) &\leftarrow \arg\min_{1 \leq i \leq k} a(i) d(x_j, c_i) \\ a(f(j)) &\leftarrow a(f(j)) + 1 \end{aligned}$$

Step 2: let $A_i = \{x_l,\, f(l) = i\}$ and for $1 \leq i \leq k$

$$c_i \leftarrow \frac{1}{|A_i|} \sum_{x \in A_i} x$$

Output: cluster membership function f

In this version of Lloyd's algorithm, every time a node is assigned to a centroid c_i, the activity $a(i)$ of c_i is increased. As distance function $d(x_j, c_i)$ is multiplied by the activity $a(i)$ at step 1, centroids with low activity are chosen in priority, hence avoiding the empty cluster phenomenon. The problem of such approach is that it tends to find rather balanced clusters (namely clusters containing approximately the same number of points). Hence, depending on the application, one can choose either variant 2.5 or 2.6 of Lloyd's algorithm. From now, we always use Lloyd's algorithm with frequency sensitive learning as we mainly want to avoid empty clusters.

Method 2: Dynamic programming algorithm

In the case of SCE clustering algorithm, it is possible to solve k-means problem exactly with dynamic programming algorithm. Instead of solving k-means problem (definition 10), we solve a slightly different problem. Given n points in $\mathbb{C}$ the goal is to find an assignment of $f : \{1, ..., n\} \to \{1, ..., K\}$ such that the sum of distances between points in the same cluster is minimized:

$$f = \arg\min_{\phi} \sum_{\phi(u) = \phi(v)} d_c(u, v) \tag{2.5}$$

[3]This is not possible in practice (NP-hard problem) so we rather use the following heuristic: choose an initial centroid randomly in the data set, choose the point of the data set located furthest from the first centroid as the second centroid, choose the point minimizing the sum of distances to the first and the second centroid as the third centroid, etc.

where d_c is the cosine distance

$$d_c(u,v) = \frac{Re(u) * Re(v) + Im(u) * Im(v)}{\|u\|\|v\|}.$$

We denote the objective function of this optimization problem as the *total inner distance*. We consider the clustering of n points $x_1, ..., x_n$ into k clusters. We assume the points are ordered in increasing order of phase (in $[0, 2\pi[$). We use the following notations:

- sets X and I:

$$X(i,j) = \begin{cases} \{x_i, ..., x_j\} & \text{if } i \leq j \\ \{x_i, ..., x_n, x_1, ..., x_j\} & \text{if } i > j \end{cases}$$

$$I(i,j) = \begin{cases} \{i, ..., j\} & \text{if } i \leq j \\ i, ..., n, 1, ..., j & \text{if } i > j \end{cases}.$$

- $D(i,j,m)$: for $m \geq 2$ and $m < |X(i,j)|$ is the minimum total inner distance to cluster points in $X(i,j)$ into m clusters such that points x_i and x_j do not belong to the same cluster.
- $D(i,j,1)$: the total inner distance when clustering points in $X(i,j)$ into one cluster.

By convention, we note $X(i,0,m) = X(i,n,m)$.

The recurrence relation to find all entries $D(i,j,m)$ for $i,j \in \{1,...,n\}$ is

$$\begin{aligned} D(i,j,1) &= \sum_{u,v \in X(i,j)} d_c(u,v) \\ D(i,j,m) &= \min_{l \in I(i+m,j)} D(i, l-1, m-1) + \sum_{u,v \in X(l,j)} d_c(u,v) \text{ for } 1 < m \leq |X(i,j)| \end{aligned}.$$

Finally, to solve equation 2.5, we compute:

$$\min_{1 \leq i \leq n} D(i, i-1, k).$$

A simple inspection of table D allows to recover the optimal solution.

Compared to Lloyd's method, the dynamic programming algorithm has the advantage of giving an exact result (it gives the optimal solution of problem 2.5). However, the complexity of this method is $O(n^3)$ where n is the number of points considered (namely the number of nodes in the graph). Hence the dynamic programming algorithm quickly becomes prohibitive for large problems. In contrast the complexity of one iteration of Lloyd's algorithm is $O(n^2)$. With our initialization, i.e. taking the points points maximizing the sum of pairwise distances as initial centroids, experimental tests show that the convergence is very fast: in typical examples of slightly perturbed block-cycles, 3 to 4 iterations of Lloyd's algorithm are sufficient. For these reasons, we argue that Lloyd's algorithm should be preferred in practice, especially for very large graphs.

2.4.4 local search algorithm to improve the output of MCE and SCE algorithms

Our algorithm of block-cycle detection is based on the following empirical observation: in a block-cycle (or a slightly perturbed block-cycle), if two nodes i and j belong to the same block,

- according to MCE clustering approach, i-th and j-th rows of matrix Γ of cycle eigenvectors are close to each other in terms of Euclidean distance,
- according to SCE clustering approach, the difference in phase of i-th and j-th entries of the first cycle eigenvector is small.

However, if the perturbation is large, the components of the cycle eigenvector may not cluster properly according to block membership or even worse, components associated to nodes belonging to different blocks may overlap. Some of these situations are illustrated in figure 2.8.

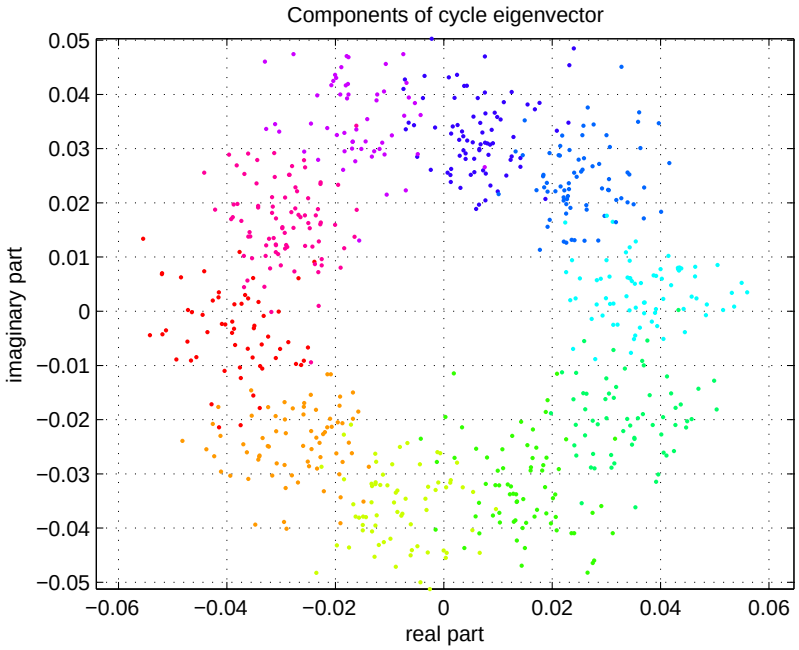


Figure 2.8: For a block-cycle of 700 nodes partitioned into 8 blocks $V_1, ..., V_8$ with probability 0.7 of having an edge (u, v) for u and v belonging to consecutive blocks, the block-cycle is perturbed by randomly adding $0.5n^2 \simeq 30000$ random edges to the graph: the graph shows the repartition of the components of the cycle eigenvector in the complex plane (colouring of points indicate the block membership of nodes), the blocks do not form clusters that are separable in an obvious way.

In section 2.1, we claimed MCE and SCE clustering algorithms approximates the maximum of $C_r(f)$. However, we do not provide any theoretical proof of the relation between the algorithms and the maximization of $C_r(f)$. Hence, for perturbed block-cycles, we do not have any guarantee about the quality of the result expressed as the ratio

$$\frac{C_r(f)}{C_r^{opt}}$$

where C_r^{opt} is the optimal value of criterion and $C_r(f)$ is the value of the objective function achieved with an estimated block assignment f using either MCE or SCE clustering algorithms.

In order to improve this quality measure of the result, one could suggest to use a local search process. This post-processing step is based on the following observation: for slightly perturbed

block-cycles, both MCE and SCE clustering algorithms tend to confuse blocks that are consecutive in the block-cycle. What we expect from this local search algorithm is schematically described in figure 2.9.

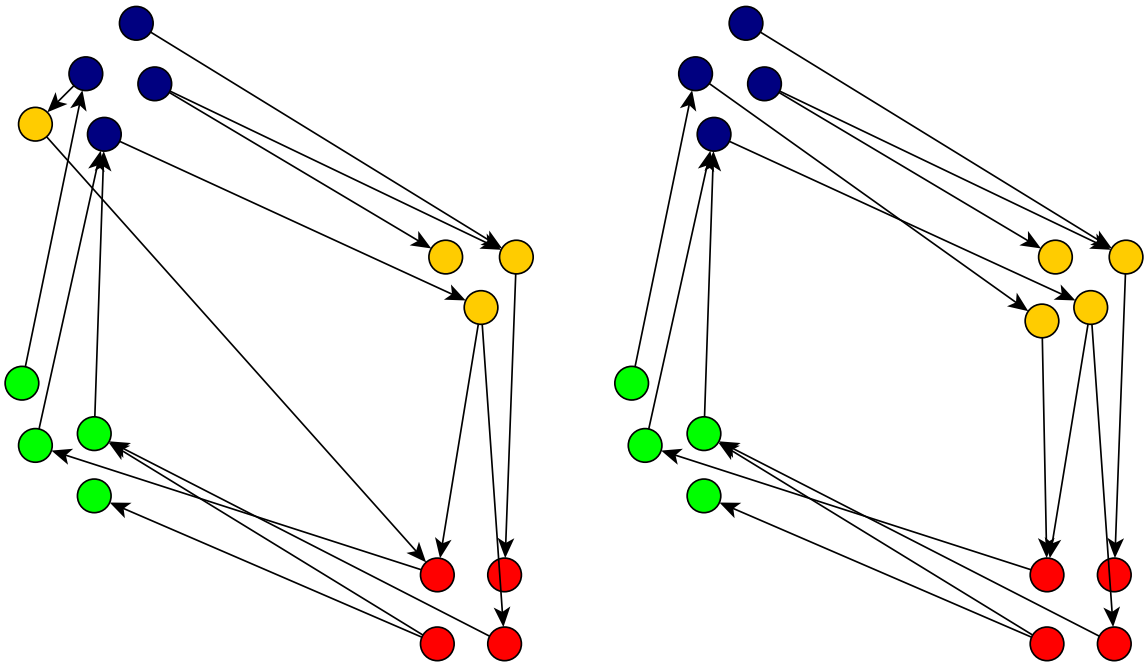


Figure 2.9: Figure illustrating the local search approach: (left, figure 2.9(a)) shows a typical output of MCE or SCE algorithm, one node belonging to yellow cluster is misclassified into blue cluster and yellow cluster and blue cluster are consecutive in the cycle, (right, figure 2.9(b)) shows what we want from the local search process, namely detect the misclassification of the yellow node and reclassify it in the yellow cluster. This example is rather artificial as both MCE and SCE clustering algorithm never misclassify nodes when the input is an unperturbed block-cycle, such a mis-clustering would occur for perturbed block-cycles only.

This observation suggests the following local search algorithm for the improvement of the output of both MCE and SCE clustering algorithms. In the following developments, we use the absolute cycle criterion $C(f)$ (equation 2.3) rather than the relative cycle criterion $C_r(f)$ (defined in equation 2.4), the latter being a scaled version of the former. We also denote by *neighbouring blocks* of block l the blocks

$$\begin{array}{ll} \{l-1, l+1\} & \text{if } 1 < l < k \\ \{1, k-1\} & \text{if } l = k \\ \{2, k\} & \text{if } l = 1 \end{array} .$$

Moreover, given an block membership function f, we denote by $A(i, \alpha, f)$ the contribution of node $i \in V$ to the value of the objective function if i is assigned to block α and the block membership of all other nodes is determined by function f:

$$A(i, \alpha, f) = \sum_{j:f(j)=\alpha+1} W_{ij} + \sum_{j:f(j)=\alpha-1} W_{ji}.$$

Algorithm 2.7: Local Search algorithm

Input: Adjacency matrix $W \in \{0,1\}^{n\times n}$, estimated block membership f, relative criterion $C_r(f)$, blocks are numbered from 1 to k in their order of appearance in the cycle;
Iteration: While it is still possible to increase the value of the criterion $C(f)$ by changing the membership of a node i to a neighbouring block of $f(i)$;
Step 1: pick a node $i \in V$;
Step 2: evaluate the current contribution of i to the objective value and its potential contribution if it is moved to a neighbouring block;

$$\begin{aligned} r &= A(i, f(i), f) \\ s^+ &= A(i, f(i)+1, f) \\ s^- &= A(i, f(i)-1, f) \end{aligned}$$

Step 3: change the block membership of i if it improves the objective function;

```
if max(s+, s−) > r then
    if s+ > s− then
        reassign i to block f(i) + 1;
    end
    else
        reassign i to block f(i) − 1;
    end
end
```

Output: (improved) estimation of block membership f;

There are several ways one could choose the node $i \in V$ to consider at each iteration. One of the following strategies can be applied.

- Go through all vertices from 1 to n, when node n is reached, come back to node 1 and restart the process until the objective value cannot be increased anymore. For practical reasons, we also limit the number of times we go through all vertices, this limit number is called the **number of local search passes** and it is a parameter of the algorithm.
- Consider first the nodes that allow the largest increase in objective value if moved to a neighbouring block.
- For MCE clustering algorithm, considering matrix $\Gamma \in \mathbb{C}^{n\times k}$ of cycle eigenvectors, the clustering step of MCE assigns each row of Γ to a cluster which defines Voronoï zones in $\mathbb{C}^k$. We may consider moving nodes for which the associated row of Γ is at the boundary of its Voronoi zone. Such points should be more likely to be misclassified.

We choose the first strategy but further studies could examine the quality of the result obtained with the two other methods.

Now, we prove a theorem giving some guarantees regarding the quality of the result of the local search procedure regardless of the initial solution provided by SCE or MCE clustering algorithms.

Theorem 6

Given a positive weighted directed graph $G = (V, E, W)$ with adjacency matrix W and estimated block membership function $f : V \to \{1, ..., k\}$ partitioning the nodes into k

blocks, we assume the natural ordering of blocks suggested by f is $1, .., k$ (ie there is a great number of edges from block 1 to block 2, from block 2 to block 3 etc.). We let

$$\begin{array}{rcl} \tilde{E} & = & \{(u,v) \in E : f(u) = f(v) \text{ or } f(v) = f(u)+1 \text{ or } f(v) = f(u)+2\} \\ O & = & \sum_{(u,v)\in E\setminus\tilde{E}} W_{uv} \end{array} \tag{2.6}$$

If C^* denotes the absolute cycle criterion achieved with block membership function f and C^{opt} denotes the maximum absolute cycle criterion that can be achieved on graph G, and if C^* cannot be increased by local search algorithm 2.7 starting with membership function f, then

$$2C^* + O \geq C^{opt}.$$

Proof: We introduce the following notations. For any $\alpha \in V$,

$$\begin{array}{rcl} m^+_{\alpha j} & = & \sum_{v\in V_j} W_{\alpha v} \\ m^-_{\alpha j} & = & \sum_{v\in V_j} W_{v\alpha} \\ l_{ij} & = & \sum_{f(u)=i, f(v)=j} W_{uv} \\ l_i & = & \sum_{f(u)=f(v)=i} W_{uv} \end{array}.$$

We denote the k blocks partitioning V according to f by $V_1, ..., V_k$.

We consider any node $\alpha \in V_i$ and assume $2 < i < k-1$ for the sake of simplicity. Then,

- the contribution of α to C^* is $m^-_{\alpha,i-1} + m^+_{\alpha,i+1}$,
- the potential contribution of α if it is reassigned to block $i-1$ is $m^-_{\alpha,i-2} + m^+_{\alpha,i}$,
- the potential contribution of α if it is reassigned to block $i+1$ is $m^-_{\alpha,i} + m^+_{\alpha,i+2}$,

As C^* cannot be increased by a local search procedure starting with membership function f, we have the inequalities

$$\begin{array}{rcl} m^-_{\alpha,i-1} + m^+_{\alpha,i+1} - (m^-_{\alpha,i-2} + m^+_{\alpha,i}) & \geq & 0 \\ m^-_{\alpha,i-1} + m^+_{\alpha,i+1} - (m^-_{\alpha,i} + m^+_{\alpha,i+2}) & \geq & 0 \end{array}.$$

Summing over all vertices $\alpha \in V_i$, we obtain:

$$\begin{array}{rcl} l_{i-1,i} + l_{i,i+1} - l_i - l_{i-2,i} & \geq & 0 \\ l_{i-1,i} + l_{i,i+1} - l_i - l_{i,i+2} & \geq & 0 \end{array}.$$

Repeating the same procedures for all $i \in \{1, ..., k\}$ and summing all inequalities, we obtain

$$l_{12} + l_{23} + ... + l_{k-1,k} + l_{k1} - (l_1 + ... + l_k) - (l_{13} + l_{24} + ... + l_{k-1,1} + l_{k,2}) \geq 0 \tag{2.7}$$

Using the notation O defined in 2.6 and defining $m = \sum_{(u,v)\in E} W_{uv}$, we obtain

$$\begin{array}{rcl} l_{12} + l_{23} + ... + l_{k-1,k} + l_{k1} & = & C^* \\ l_1 + ... + l_k + l_{13} + l_{24} + ... + l_{k-1,1} + l_{k,2} & = & m - C^* - O \end{array}.$$

Putting these expressions back in 2.7, we obtain

$$2C^* + O \geq m \geq C^{opt}.$$

■

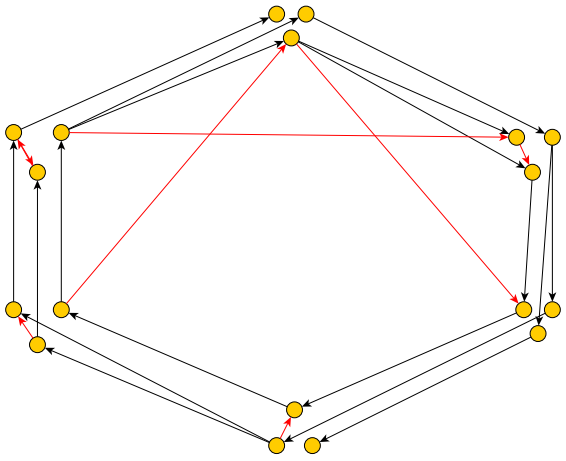


Figure 2.10: Typical perturbed block-cycle where $O = 0$. Perturbing edges are coloured in red. Each of these perturbing edges either lies within a block either appears to "jump over one block".

The preceding theorem gives a guarantee on the quality of the result obtained by the local search algorithm. However, this guarantee is of interest only when O is small which is the case when perturbation essentially consists of edges having both extremities in the same block or edges "jumping over one block". This situation is depicted in figure 2.10.

It is likely that the perturbations of block-cycles encountered in practice are more complex than the one depicted in figure 2.10, in which case the guarantee of quality provided by theorem 6 is of little interest. In order to give better guarantees on the quality of the result, one could enlarge the local search space. Instead of only considering the reassignment of a vertex to neighbouring blocks, one could try to reassign vertices to any block of the cycle in order to improve C^*. This is computationally more expensive but a straightforward generalization of the preceding theorem would show that the ratio between C^* and C^{opt} is

$$\frac{C^{opt}}{C^*} \leq 2$$

which provides a stronger guarantee on the quality of the result. In practice we prefer to use the algorithm 2.7 (based on neighbouring blocks). Indeed, as the initial solution provided by MCE or SCE clustering algorithm is assumed to be close to the optimal solution, algorithm 2.7 should give good results while being much cheaper. Tests on synthetic data regarding the efficiency of the Local Search algorithm are found in section 4.

3 Consistency of SCE algorithm

In this section, we provide a proof of a consistency theorem for SCE clustering algorithm. We must first clarify what we mean by consistency. We consider a block-cycle $G = (V, E, W)$ and a perturbed block-cycle $\tilde{G}$ obtained by appending a few perturbing edges to G. We denote by $f : V \to \{1, ...k\}$ the block membership of nodes in block-cycle G and by $\tilde{f}$ the block membership estimated by applying SCE clustering algorithm to $\tilde{G}$. We define the mis-clustering rate η as the proportion of nodes that are not classified into the same block according to f and $\tilde{f}$. What we mean by consistency of SCE algorithm is the fact that whenever the perturbation in $\tilde{G}$ tends to zero, the mis-clustering rate η also tends to zero. Obviously, we will have to define formally what we mean by the perturbation going to zero. We only prove the consistency of SCE clustering algorithm. In next section, tests on synthetic data will show that the classification error of MCE clustering algorithm is always smaller than the one of SCE clustering algorithm so we make the assumption that MCE clustering algorithm is also consistent.

For our proof of consistency, we restrict ourselves to a particular type of block-cycle $G = (V, E, W)$: the number of blocks in G is 2 and edges of G are undirected. The fact that we limit ourselves to undirected graphs can seem in contradiction with the definition of block-cycle which assumes the presence of a pattern of directed cycle. However, we remind that in the particular case of two blocks, the edges of a block-cycle can be undirected such as suggested by figure 2.11. In that case, the block-cycle actually corresponds to a bipartite graph.

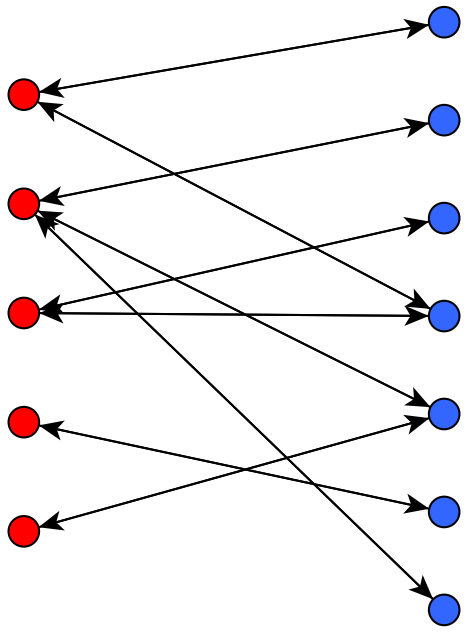


Figure 2.11: Block-cycle of 2 blocks with undirected edges. Colours correspond to block membership. In that case, the graph corresponds to an undirected bipartite graph.

The reason why we restrict ourselves to undirected graphs is that it yields a symmetric Laplacian. In that case, we can show that the steps followed by SCE clustering algorithm are very similar to the ones of classical spectral clustering algorithm for the partitioning the nodes of an undirected graph into two clusters (referred to as spectral bi-clustering algorithm), although the clusters found by both methods are totally different. The similarity is that both methods perform a clustering of the component of a particular eigenvector of the Laplacian, the difference is that they do not use the same eigenvector. A proof of consistency of spectral bi-clustering

algorithm can be found in [36]. Our contribution is that we adapt their consistency theorem to prove the consistency of SCE clustering algorithm for a completely different problem: the detection of two blocks in an undirected graph. A the end of the section, we discuss the extension of our consistency theorem to directed graphs.

3.1 Outline of the method

We limit ourselves to the case $k = 2$. The spectrum of the Laplacian a block-cycle with two blocks (not necessarily undirected) is in figure 2.12.

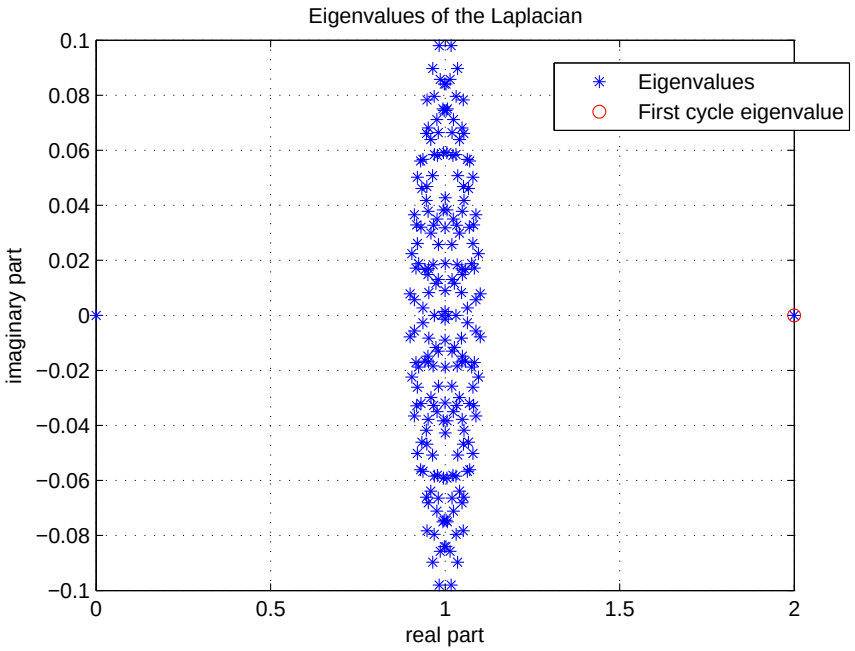


Figure 2.12: Eigenvalues of the Laplacian of a block-cycle of 2 blocks with 200 nodes. The first cycle eigenvalue is circled in red.

When considering a block-cycle of two blocks, the first cycle eigenvalue is simply the eigenvalue with largest norm and its value is 2. The first cycle eigenvector has a particular form. For instance, if we consider the case of 10 nodes with blocks $\{1, 2, 3, 4, 5, 6, 7\}$ and $\{8, 9, 10\}$, the first cycle eigenvector is

$$\mathbf{v}_c = [1, 1, 1, 1, 1, 1, 1, -1, -1, -1]^T.$$

Then, SCE clustering algorithm reduces to thresholding the components of $\mathbf{v}_c$, namely the sign of the components of $\mathbf{v}_c$ define the block membership of nodes.

The outline of our reasoning is the following. We consider a block-cycle G of two blocks and a perturbed version $\tilde{G}$ of G obtained by appending a few perturbing edges to G. We assume all edges in G have nonzero out-degree. We denote by η the mis-clustering rate, namely the proportion of nodes that do not belong to the same cluster when applying SCE clustering algorithm to G or to $\tilde{G}$. Our goal is to bound η in terms of the entries of $\Delta - \tilde{\Delta}$ where Δ and $\tilde{\Delta}$ are respectively the Laplacians of G and $\tilde{G}$. We will first bound η in terms of $\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2$, the norm of the difference between the cycle eigenvectors of G and $\tilde{G}$, then we provide a bound of $\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2$ in terms of the entries of $\Delta - \tilde{\Delta}$.

For the proof of our consistency theorem, we limit ourselves to the case where both G and $\tilde{G}$ are undirected. In that case, SCE algorithm is very similar to spectral clustering of undirected

graph into two clusters presented in section 1.3 of chapter 1: clusters are detected by extracting the sign of the second eigenvector of the Laplacian of an undirected graph. The clusters we obtain and the eigenvector we consider (respectively the second eigenvector for spectral clustering and the first cycle eigenvector for SCE clustering algorithm) are very different but both methods are based on thresholding the components of one eigenvector of the Laplacian. This similarity allows an extension of the consistency theorem for spectral bi-clustering in [36] to the case of SCE clustering algorithm.

3.2 Theorem of consistency

We use the following notations. G and $\tilde{G}$ are respectively the unpertubed and the perturbed undirected block-cycles of two blocks. Δ and $\tilde{\Delta}$ are the associated Laplacians. $\mathbf{v}_c$ and $\tilde{\mathbf{v}}_c$ are the first cycle eigenvectors of G and $\tilde{G}$ respectively. a and b are the clusters obtained by thresholding the components of $\mathbf{v}_c$, the sizes of which are k_a and k_b. a' and b' are the clusters obtained by thresholding the components of $\tilde{\mathbf{v}}_c$. We denote by $a \to a'$ the set of components that are clustered in a when considering $\mathbf{v}_c$ and clustered in a' when considering $\tilde{\mathbf{v}}_c$, similarly for $a \to b'$, $b \to a'$ and $b \to b'$. We denote by m the total number of components in $a \to b'$ and $b \to a'$, namely the total number of mis-clustered nodes.

We make the following assumptions regarding $\mathbf{v}_c$ and $\tilde{\mathbf{v}}_c$ (these assumptions are very similar to the one described in [36], p.4).

- A1. The blocks of G have sizes comparable to n.
- A2. The components of $\tilde{\mathbf{v}}_c$ form two clusters, the size of each cluster is comparable to n. The total number of mis-clusterings is small in comparison with the original clusters $m << \min(k_1, k_2)$.
- A3. The perturbation of the components of $\mathbf{v}_c$ in each set of $a \to a'$, $a \to b'$, $b \to a'$ and $b \to b'$ have identical distributions with bounded second moments and they are uncorrelated with the components in $\mathbf{v}_c$.

Assumptions A1 and A2 are easy to interpret: the sizes of the clusters are approximately equal and the number of mis-clustered nodes is small. Further explanations about assumption A3 and the cases in which it is verified are provided in appendix 2.

The relation between the mis-clustering rate η and $E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right]$ is given by theorem 7.

Theorem 7

Under assumptions A1, A2 and A3 and if we require $\|\mathbf{v}_c\|_2 = \|\tilde{\mathbf{v}}_c\|_2 = 1$, the mis-clustering rate η of SCE clustering algorithm under the perturbation satisfies

$$\eta \leq E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right].$$

The proof of theorem 7 is given in appendix 2.

The relation between $E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right]$ and $\tilde{\Delta} - \Delta$ is given by theorem 8. We denote by $\mathbf{v}_1$,..., $\mathbf{v}_n$ the n eigenvectors of Δ where the first cycle eigenvector is $\mathbf{v}_c = \mathbf{v}_n$, the associated eigenvalues are $\lambda_1, ..., \lambda_n$ and the first cycle eigenvalue is λ_n. Finally we denote by v_{pj} the p-th component of $\mathbf{v}_j$. We assume $\|\mathbf{v}_1\|_2 = ... = \|\mathbf{v}_n\|_2 = 1$.

Theorem 8

The relation between $E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right]$ *and* $\tilde{\Delta} - \Delta$ *is given by the following inequality*

$$E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right] \leq \sum_{p=1}^{n} \alpha_p \|\mathbf{u}_p\|_2^2$$

where

$$\begin{aligned} \mathbf{u}_p &= \textstyle\sum_{j=1}^{n-1} \frac{v_{pj}\mathbf{v}_j}{\lambda_n - \lambda_j} \\ \alpha_p &\leq \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} v_{i1}v_{jn} E\left[\tilde{\Delta}_{pi} - \Delta_{pi}\right] E\left[\tilde{\Delta}_{pj} - \Delta_{pj}\right] + |v_{in}v_{jn}|\sigma_{pi}\sigma_{pj} \end{aligned}$$

and σ_{pi}^2 *is the variance of* $\tilde{\Delta}_{pi} - \Delta_{pi}$.

The proof of theorem 8 is given in appendix 2.

What do these two theorems tell us? Theorem 7 states that the misclustering rate is controlled by the norm of the difference between the first cycle eigenvector respectively in the unperturbed and the perturbed cases. In particular, whenever $E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right]$ tends to zero, η tends to zero. Theorem 8 shows that a controlled variation of the mean and variance of the entries of $\tilde{\Delta} - \Delta$ yields a controlled variation of $E\left[\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2^2\right]$ and hence of η. The coefficients of $\mathbf{u}_p$ are inversely proportional to the differences $\lambda_n - \lambda_j$ between the first cycle eigenvalue of G and all other of its eigenvalues. For an undirected block-cycle with two blocks, $\lambda_n = 2$ and all other eigenvalues lie in $[0, 2[$, hence having all eigenvalues $\{\lambda_j, j \neq 2\}$ away from 2 yields a tighter bound.

In [36], the authors further give a relation between $\Delta - \tilde{\Delta}$ and $W - \tilde{W}$, the difference between the adjacency matrices of G and $\tilde{G}$.

This relation states that as long as G does not contain any node of zero out-degree, a controlled variation of the adjacency matrix yields a controlled variation of the Laplacian. The full expression is given in [36] (we do not provide it as we primarily wanted to give a relation between the mis-classification rate η and the difference between Laplacians $\tilde{L} - L$.

3.3 Relaxation of assumptions

What happens if we relax the assumption that G and $\tilde{G}$ are undirected. Tests on synthetic data show that theorem 7 stays valid. If G is a block-cycle with two blocks, it stays true that the first cycle eigenvalue of G is 2 and that the sign of the components of the first cycle eigenvector $\mathbf{v}_c$ yields the block membership of all nodes. For a slightly perturbed graph $\tilde{G}$, the first cycle eigenvector $\tilde{\mathbf{v}}_c$ is complex but the imaginary parts are largely negligible for small perturbations.

In contrast theorem 8 is not verified in the directed case as we have no guarantee that the Laplacian Δ is diagonalizable (which is an implicit requirement of theorem 8). It is not even true that the components of eigenvectors vary continuously with the entries of the Laplacian Δ. Indeed, perturbations on Δ could change the dimension of some eigenspaces of Δ. Further work should be done to verify in what cases we can guarantee a controlled variation of $\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2$ for directed graphs G and $\tilde{G}$.

Hence, the extensions of theorems to the directed case is not straightforward. However, as we will show in next section, tests on synthetic data are satisfactory.

For the sake of simplicity, we restricted ourselves to the case of 2 blocks. This choice is justified by the fact the consistency theorems for undirected spectral clustering in [36] is also restricted to the case of 2 clusters or spectral bi-clustering. It is commonly accepted that there are more theoretical guarantees of the quality of the result of bi-clustering than in the general case. However, as we will show in next section, our algorithm is also efficient in the detection of a larger number of blocks.

4 Tests on synthetic data

In this section, we use synthetic data in order to test the performances of MCE and SCE clustering algorithms and their sensitivity to perturbations. Our procedure is as follows: we generate some slightly perturbed block-cycle and we measure the evolution of the clustering error of MCE and SCE clustering algorithms for different values of some parameters (such as the number of nodes, the magnitude of the perturbation, the presence of unbalanced blocks...). We first clarify some notions including the way we compute the clustering error and the way we create perturbed block-cycles. Then, the rest of the section is dedicated to the description of tests and results. For a table summarizing these observations, we refer to section 4.7.

Both MCE clustering and SCE clustering algorithms are designed in a way that they always detect the blocks of an unperturbed block-cycle perfectly. In order to generate block-cycles automatically, we propose the following *cyclic stochastic blockmodel* defined as a particular *stochastic blockmodel* (section 3.2.1 in chapter 1). For the sake of simplicity, we restrict ourselves to the case of unweighted graphs.

Definition 11 (*Cyclic stochastic blockmodel*)

A cyclic stochastic blockmodel is a blockmodel with parameters (k, ρ, P) such that

$$P_{ij} \begin{cases} > 0 \text{ if } (i,j) \in \{(1,2),(2,3),...,(k-1,k),(k,1)\} \\ = 0 \text{ otherwise} \end{cases}$$

up to a permutations of block labels $\{1,...,k\}$.

Hence matrix P of probabilities of connexions has the form

$$P = \begin{bmatrix} & \times & & & \\ & & \times & & \\ & & & \ddots & \\ & & & & \times \\ \times & & & & \end{bmatrix}$$

where "$\times$" represents a non-zero entry.

We use this definition of cyclic stochastic blockmodel to define the following benchmark procedure.

1. Generate a block-cycle $G = (V, E)$ with adjacency matrix W_c using a cyclic stochastic blockmodel.
2. Generate a perturbed graph $\tilde{G}$ with adjacency matrix $W = W_c + W_p$ for some perturbation matrix W_p.
3. Run a block-cycle detection algorithm (MCE clustering or SCE clustering algorithm) on $\tilde{G}$.
4. Compute the error in the estimation of the block membership of nodes of $\tilde{G}$ compared to the ideal block membership of nodes of G.

Some details need to be clarified in this procedure.

Firstly, the parameters (k, ρ, P) of the cyclic blockmodel must be defined. In order to facilitate the description of a block-cycle, we refer to *consecutive blocks* as blocks that are consecutive in the block-cycle: for instance, if the order of appearance of blocks in a block-cycle of 5 blocks is $1, 3, 4, 2, 5$, the pairs $(1, 3)$, $(3, 4)$, $(4, 2)$, $(2, 5)$ and $(5, 1)$ are pairs of consecutive blocks. Unless otherwise specified, we assume $k = 10$, $\rho = [0.1, ..., 0.1]$ and the k nonzero entries of P have value 0.7. Hence blocks tend to be balanced (namely, they tend to contain approximately the same number of nodes) and the probability of having an edge between two nodes belonging to consecutive blocks is 0.7 regardless of the pair of consecutive blocks we consider. In sections 4.4 and 4.5, we analyse the behaviour of the algorithm if these two assumptions are relaxed.

Secondly, we must generate insightful perturbations of block-cycle G to form the adjacency matrix $W_c + W_p$ of graph $\tilde{G}$. We define two types of perturbations:

1. random perturbation: $\tilde{G}$ is obtained by appending s unweighted edges randomly to G, allowing to append the same edge several times which simply increases the weight of the perturbing edge,

2. perturbation of the blockmodel: perturbation of matrix P in the blockmodel before generating graph G,

In each of the following sections, we specify what type of perturbation is generated.

Thirdly, we must define the error in the estimation of block membership of nodes of $\tilde{G}$ compared to the ideal block membership of nodes of G.

Definition 12 (*Block membership error*)

Given a graph $G = (V, E)$ generated by a cyclic stochastic blockmodel with block membership function $f : V \to \{1, ..., k\}$ and a perturbed version $\tilde{G} = (V, \tilde{E})$ of graph G, if $\tilde{f} : V \to \{1, ..., k\}$ is an estimated block membership function based on $\tilde{G}$, then the block membership error is

$$\frac{1}{|V|} \min_{\pi \in \Pi(k)} |\{u \in V : f(u) \neq \pi(\tilde{f}(u))\}|$$

where $\Pi(k)$ is the set of permutations on $\{1, ..., k\}$.

In other words, the block membership error computes the number of differences in the entries of f and $\tilde{f}$ up to a permutation of block labels $\{1, ..., k\}$. Computation of the block membership error can be formulated as a minimum matching problem which can be solved by Hungarian algorithm.

4.1 Comparison between MCE clustering and SCE clustering algorithms

In this section, we compare the performances of MCE clustering and SCE clustering algorithms on a randomly perturbed block-cycle.

We generate an unperturbed block-cycle using a cyclic stochastic blockmodel with the following parameters

- 1000 nodes,
- 10 blocks,

- $\rho = [0.1, ..., 0.1]$,
- all nonzero entries of P are equal to 0.7.

The perturbation graph $\tilde{G}$ is obtained by appending r edges randomly to G for r varying from 0 to $60m$ perturbing edges where m is the number of edges in G. Every time m perturbing edges are appended, MCE clustering and SCE clustering algorithms are ran onto the perturbed graph and the block membership errors are computed. As we wish to compare the performances of MCE clustering and SCE clustering algorithms only, we do not include any local search post-processing at this stage.

Figure 2.13 shows the evolution of the error as a function of the ratio between the number of perturbing edges and the number m of edges in G.

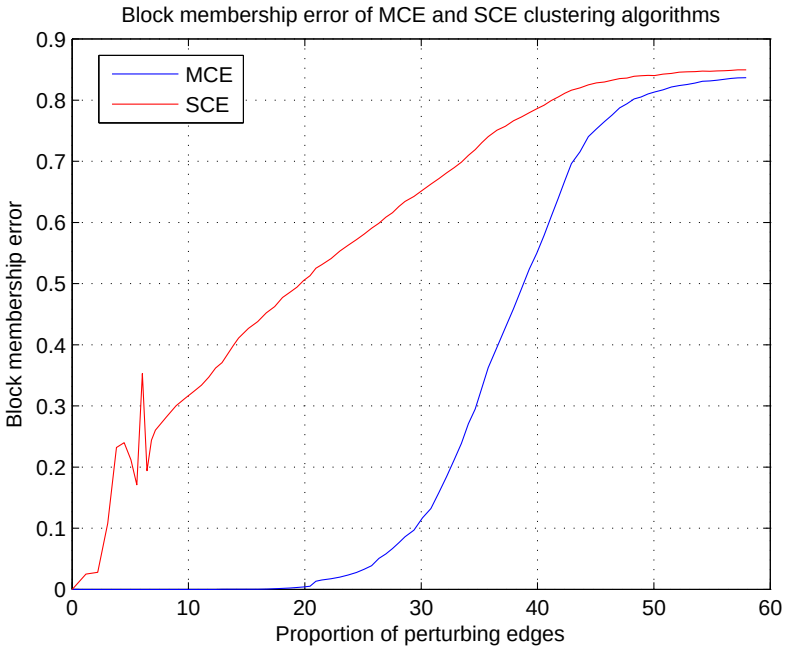


Figure 2.13: Evolution of the block membership error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 balanced blocks obtained and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending r edges randomly to G for r varying between m and $60m$ edges, where m is the number of edges in G. MCE clustering algorithm yields a smaller error than SCE clustering algorithm. If the perturbation is below $32m$ perturbing edges, the error of MCE clustering algorithm is approximately below 0.2.

We observe from this graph that MCE clustering algorithm even without local search post-processing performs well on randomly perturbed block-cycles: if there are m edges in the block-cycle and up to $32m$ edges are randomly appended to the graph, the block membership errors remains approximately below 0.2.

Although SCE clustering algorithm performs well on non-perturbed graphs, the error increases much faster then for MCE clustering algorithm when the graph is perturbed. Hence, the fact that MCE clustering algorithm takes all k cycle eigenvalues and cycle eigenvectors into account makes it less sensitive to perturbation than SCE clustering algorithm which is only based on the first cycle eigenvalue and eigenvector.

One could argue that the time of computation of SCE clustering algorithm should be significantly smaller than the time of computation of MCE clustering algorithm. However, as seen before, we can use Arnoldi algorithm to compute all cycle eigenvalues needed for MCE clustering algorithm and this algorithm is very efficient in time and quality of the result, even if the dimension of the underlying Krylov space is taken close to the number k of blocks. Hence, for small values of k (typically, $k = 10$), there is no significant difference between MCE clustering and SCE clustering for the computation of eigenvalues and eigenvectors.

Figure 2.14 shows how the times of computation evolve when the number n of nodes increases from 50 to 3000 and the number of edges is taken as approximately $\frac{n^2}{10}$. Implementation was done in `Matlab`. Measures of time reveal that about 90% of the time of computation is dedicated to the k-means step both for MCE clustering and SCE clustering algorithms. However, we do not observe a significant difference between SCE clustering and MCE clustering in terms of time of computation: both involve running Lloyd's algorithm on n points in $\mathbb{C}$ for the former and in $\mathbb{C}^k$ for the latter.

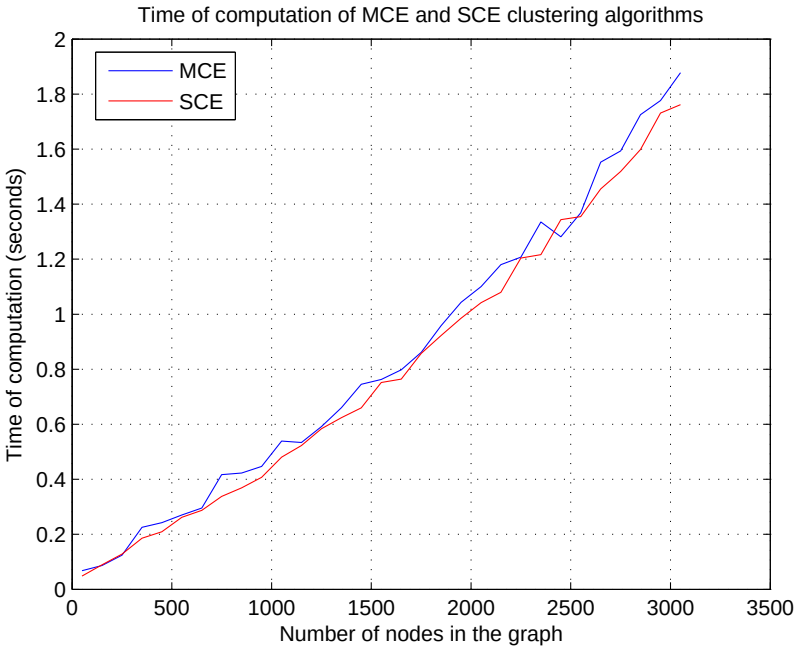


Figure 2.14: Evolution of the times of computation of MCE and SCE clustering algorithms applied to a block-cycle G. G is a block-cycle of 10 balanced blocks and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. The number of nodes in the graph varies from 50 to 3000. The time of computation is approximately the same for both MCE and SCE clustering algorithms.

4.2 Effect of the number of nodes

In this section, we analyse how the performances of the algorithms on a perturbed graph change when the number of nodes is increased. Again, we consider 10 blocks, $\rho = [0.1, ..., 0.1]$ and all nonzero entries of P being equal to 0.7. The perturbation of the block-cycle G is obtained by appending $1.3m$ edges randomly to G, where m is the number of edges of G. Figure 2.15 shows how the error evolves when the number of nodes varies from 100 to 2000. For both MCE clustering and SCE clustering algorithms, the error tends to decrease when the number of nodes is increased and the perturbation is maintained in the same proportion.

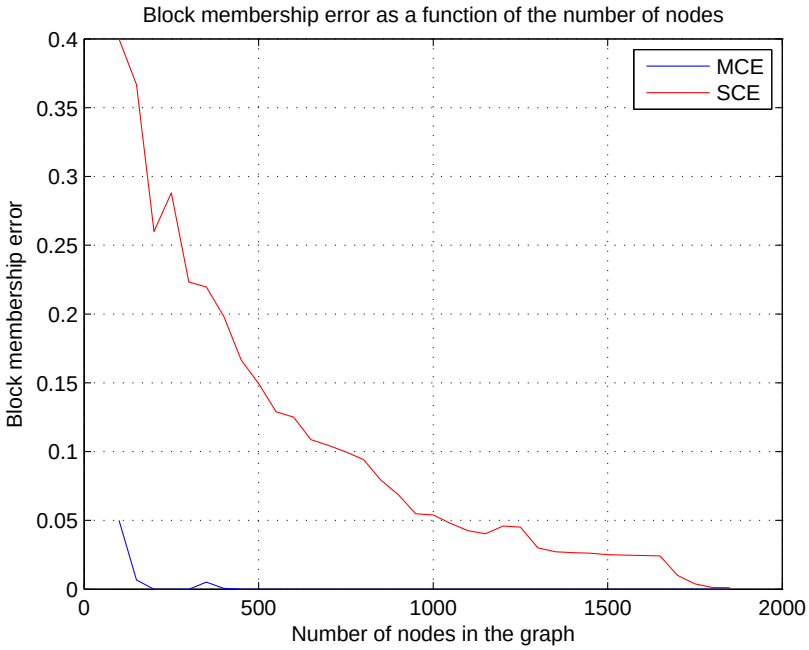


Figure 2.15: Evolution of the error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 10 balanced blocks and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending $1.3m$ edges randomly to G, where m is the number of edges of G. The number of nodes in the graph varies from 100 to 2000. We observe that the error of both algorithms tends to decrease when the number of nodes grows.

4.3 Efficiency of local search step

In section 2.4.4, we introduced a local search method in order to improve the solution returned by MCE clustering or SCE clustering algorithms justified by the fact that both algorithms tend to confuse blocks that are consecutive in the cycle. In this section, we analyse how efficient this method is in practice. As before, the graph consists of 10 blocks, ρ equals $[0.1, ..., 0.1]$ and all nonzero entries of P are equal to 0.7. We consider two different values for the number n of nodes and for the perturbation.

Firstly, we consider $n = 100$ nodes and a perturbation of $1.3m$ edges randomly added to the unperturbed block-cycle G, where m is the number of edges of G. Figure 2.16 shows the evolution of the block membership error and the cycle criterion when the number of local search passes varies from 0 to 10 passes. We see that the local search procedure significantly improves the output of the algorithms, especially for SCE clustering algorithm. These graphs also confirm the relation between the optimization of the cycle criterion and the problem of finding a block cyclic structure in a graph: when the cycle criterion is increased, the block membership error is decreased.

Secondly, we consider $n = 1000$ nodes. We append a perturbation of $8m$ perturbing edges. Figure 2.17 shows the evolution of the block membership error and the cycle criterion when the number of local search passes varies from 0 to 10 passes. Error is still decreased by the local search process for SCE clustering algorithm. However, for MCE clustering algorithm, the error is zero without local search process and it is slightly increased when adding a few local search passes. Hence, one should keep in mind that the local search procedure does not always decrease the block membership error, especially when the perturbation is large: indeed, the local search method does not explicitly look for a block-cycle, it only intends to improve the cycle criterion. Hence when the perturbation is large its result can be very different from the

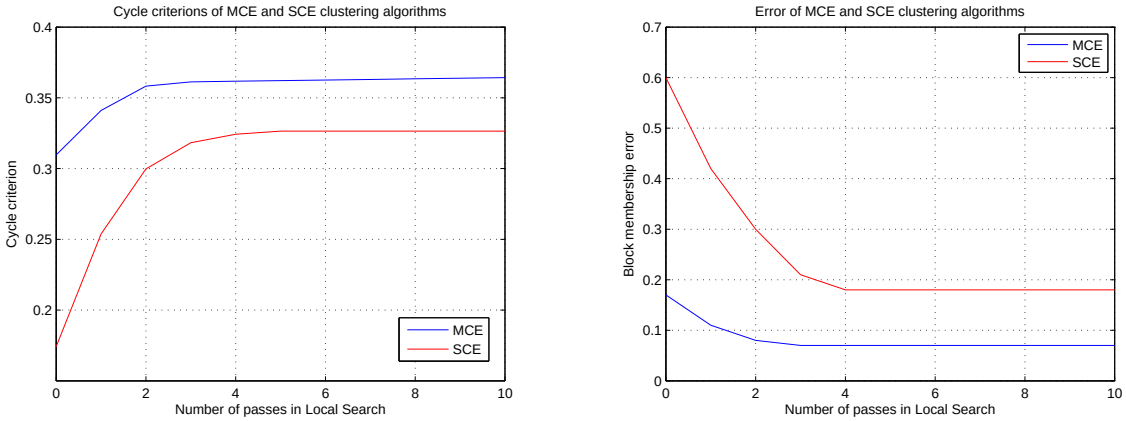


Figure 2.16: Evolution of the cycle criterion (left, figure 2.16(a)) and the block membership error (right, figure 2.16(b)) for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 100 nodes, 10 balanced blocks and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending $2.4m$ edges randomly to G, where m is the number of edges of G. The number of local search passes varies from 0 to 10 passes and the graph displays the error and the criterion after each pass. We observe that the error of both algorithms tends to decrease with a growing number of local search passes.

expected block-cycle.

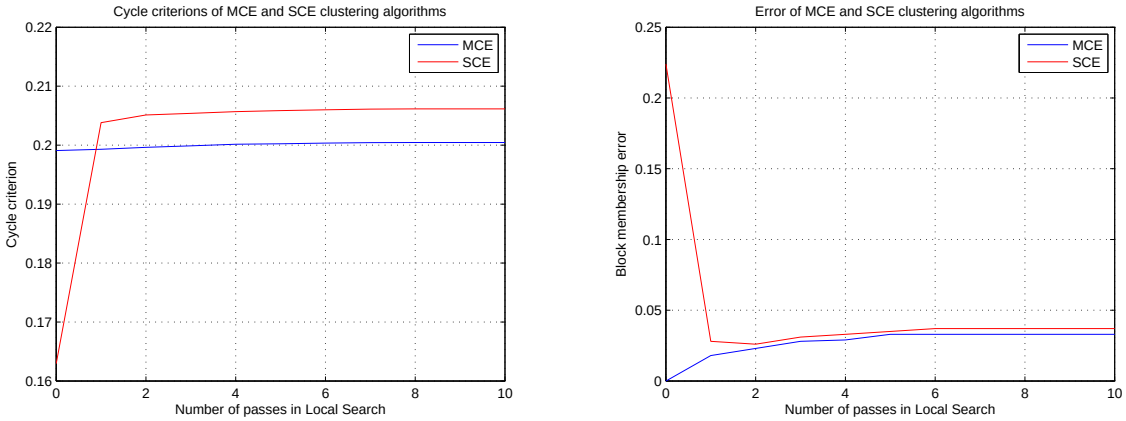


Figure 2.17: Evolution of the cycle criterion (left, figure 2.16(a)) and the block membership error (right, figure 2.16(b)) for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 balanced blocks and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending $8m$ edges randomly to G, where m is the number of edges of G. The number of local search passes varies from 0 to 10 passes and the graph displays the error and the criterion after each pass. We observe that the error of MCE clustering algorithm tends to increase with a growing number of local search passes. Hence, when the number of nodes is large and the perturbation is high (compared to the case of figure 2.16), the effect of the local search step is uncertain.

4.4 Effect of unbalanced blocks

Up to now we considered block-cycles where blocks are balanced, but what should happen if the blocks have very different sizes (commonly called unbalanced clusters)? Most of clustering algorithms fail to detect unbalanced clusters (such as spectral clustering algorithm for instance). Our algorithm perfectly detects the blocks of an unperturbed block-cycle regardless of the relative sizes of the blocks. But to avoid the appearance of empty clusters, we suggested

to use Lloyd's algorithm with frequency sensitive learning for the k-means step of both SCE and MCE clustering algorithms which could tend to find balanced clusters in the presence of perturbation. Let us test how the algorithm behaves on perturbed block-cycles with unbalanced blocks.

We generate an unperturbed block-cycle G by defining a cyclic stochastic blockmodel (definition 11) with the following parameter ρ:

$$\begin{aligned} \tilde{\rho}_i &= (1-\epsilon) + \epsilon S \text{ for } 1 \leq i \leq k \\ \rho &= \frac{\tilde{\rho}}{\sum_i \tilde{\rho}_i} \end{aligned}$$

where S is uniformly distributed on $[0,1]$ and $0 \leq \epsilon \leq 1$. Parameter ϵ is a measure of how unbalanced blocks are: when $\epsilon = 0$, the blocks tend to be perfectly balanced (uniform probability of membership to each block) and when $\epsilon = 1$, the blocks tend to be unbalanced (probabilities ρ_i's of membership are chosen randomly in $[0,1]$). Other parameters are set in the same way as before ($k = 10$ and all nonzero entries of P being equal to 0.7). We choose $n = 1000$ for the number n of nodes in the graph.

Figure 2.18 shows how the block membership error evolves for a fixed value $\epsilon = 0.1$ and by letting the perturbation vary from 0 to $60m$ perturbing edges, where m is the number of edges of G. Compared to figure 2.13, the graph of both MCE clustering and SCE clustering algorithms seem to be translated to the left, which means that the quality of the result is slightly deteriorated.

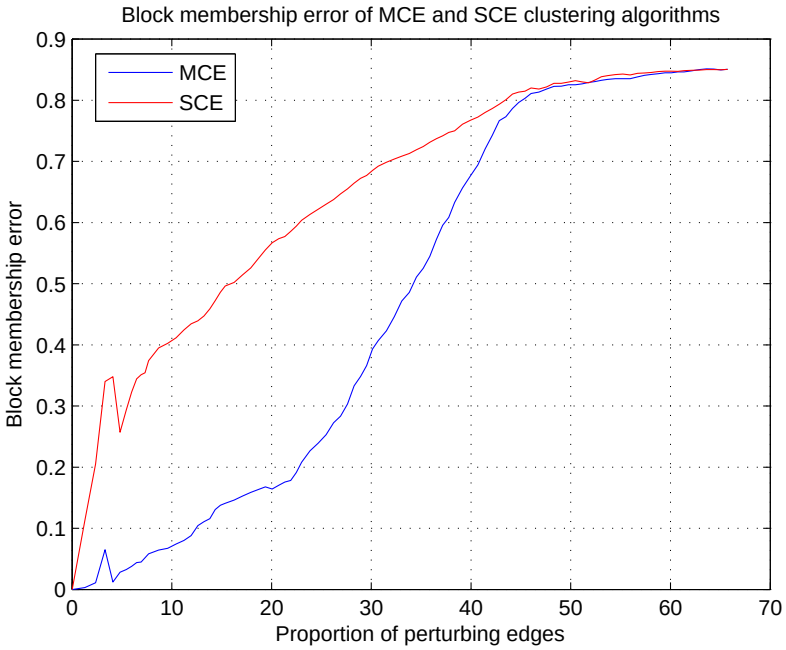


Figure 2.18: Evolution of the block membership error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 unbalanced blocks obtained with parameter $\tilde{\rho}_i = 0.9 + 0.1S$ for S uniformly distributed on $[0,1]$, and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending r edges randomly to G for r varying between m and $65m$ edges, where m is the number of edges in G. Compared to figure 2.13, the graphs of both MCE clustering and SCE clustering algorithms seem to be translated to the left, which means that the quality of the result is slightly deteriorated.

Figure 2.19 shows how the block membership error evolves for a fixed perturbation of $20m$ perturbing edges and ϵ varying from 0 to 1. With such perturbation, the quality of the result

of SCE clustering algorithm is very poor (error of 0.4) regardless of the value of ϵ). On the contrary, MCE clustering algorithm gives satisfying result (error below 0.2) even when $\epsilon = 0.8$. To realize how unbalanced the blocks are when $\epsilon = 0.8$, table 2.1 displays an example of sizes of the blocks obtained with a cyclic stochastic blockmodel with parameter $\epsilon = 0.8$.

Block	Relative size
1	2.4%
2	4.1%
3	4.7%
4	5.6%
5	9.6%
6	12%
7	13.8%
8	14.7%
9	15.7%
10	17.4%

Table 2.1: Typical sizes of blocks (as a percentage of the total number n of vertices) for a cyclic stochastic blockmodel with unbalanced blocks generated with a parameter $\tilde{\rho}_i = 0.8 + 0.2S$ for S uniformly distributed on $[0, 1]$.

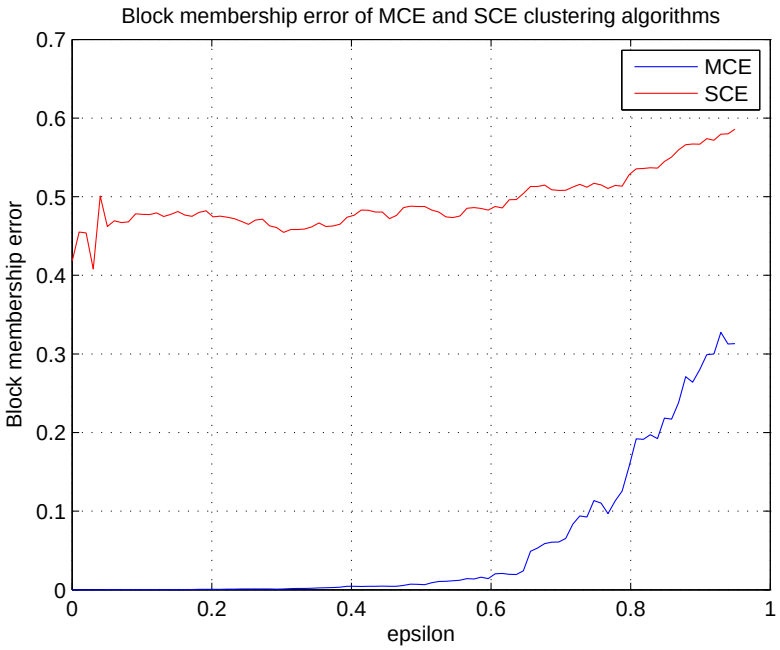


Figure 2.19: Evolution of the block membership error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 unbalanced blocks obtained with parameter $\tilde{\rho}_i = (1 - \epsilon) + \epsilon S$ for S uniformly distributed on $[0, 1]$, and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. The value of parameter ϵ varies from 0 to 1 (0 corresponding to balanced blocks and 1 to highly unbalanced blocks). $\tilde{G}$ is obtained by appending $20m$ edges randomly to G, where m is the number of edges in G. The result of SCE clustering algorithm is poor. In contrast, MCE clustering algorithms performs well on unbalanced blocks for a value of parameter ϵ up to 0.8.

4.5 Effect of unbalanced flow

If two blocks A and B are consecutive in a block-cycle, we denote by the flow between A and B the number of edges with origin in A and destination in B. Up to now, we generated block-cycles with a probability of 0.7 of having an edge (a, b) with a and b belonging to two consecutive blocks regardless of the pair of blocks considered. This means that flows between pair of consecutive blocks tend to be balanced.

If we order the blocks of a block-cycle from 1 to k in their order of appearance in the cycle, we may define a vector $p = [p_1, ..., p_k]$ where

- p_i is the probability of having an edge with origin in block i and destination in block $i+1$ for $1 \leq i < k$,
- p_k is the probability of having an edge with origin in block k and destination in 1.

Up to now, we considered $p = [0.7, ..., 0.7]$. What if entries in p are not constant? Whenever all entries of p are not constant but remain above 0.5, tests showed that the evolution of the error if we randomly perturb the corresponding block-cycle is similar to the case depicted in figure 2.13 (with $p = [0.7, ..., 0.7]$).

Another particular case we may expect to encounter is a vector p having all entries above 0.5 but one, namely there is "bottleneck" in the block-cycle.

Let us consider the case of a block-cycle with $n = 1000$ nodes, $k = 10$ blocks, $\rho = [0.1, ..., 0.1]$ and vector p of the form $p = [0.7, 0.7, ..., 0.05]$, then matrix P of the corresponding stochastic blockmodel has the form

$$P = \begin{bmatrix} & 0.7 & & & \\ & & 0.7 & & \\ & & & \ddots & \\ & & & & 0.7 \\ 0.05 & & & & \end{bmatrix}.$$

Figure 2.20 shows the evolution of the error when the perturbation varies from 0 to $60m$ perturbing edges for a block-cycle generated by this stochastic blockmodel. Compared to the balanced case depicted by graph 2.13, the efficiency is slightly deteriorated. However, if the perturbation is below $20m$ perturbing edges, the error of MCE clustering algorithm remains approximately below 0.2. In contrast, the error produced by SCE clustering algorithm is much larger.

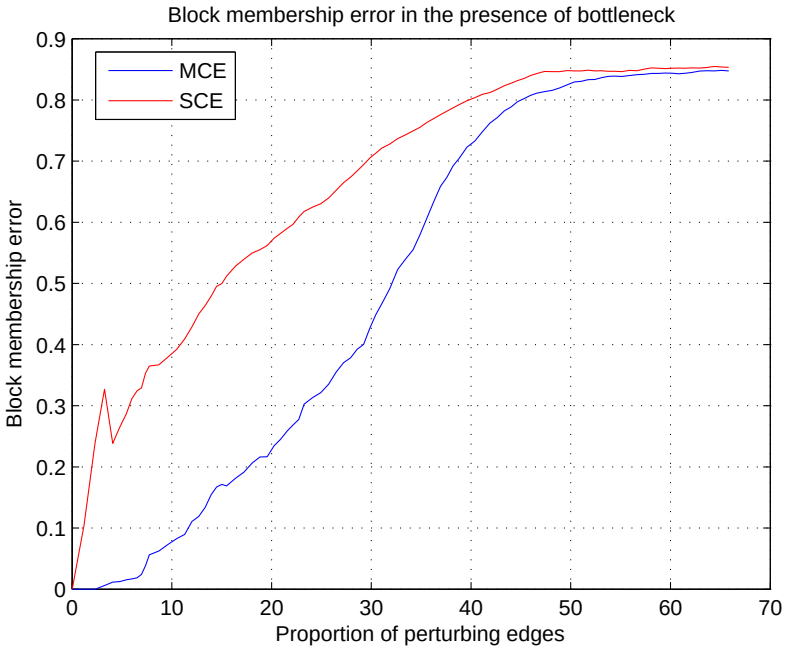


Figure 2.20: Evolution of the block membership error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 balanced blocks and parameter $p = [0.7, ..., 0.7, 0.05]$ which yields a probability of 0.7 for two nodes (a, b) in blocks $B_1 \times B_2, ..., B_9 \times B_{10}$ and a probability of 0.05 for two nodes (a, b) in blocks $B_{10} \times B_1$, where B_i denote the i-th block. $\tilde{G}$ is obtained by appending r edges randomly to G for r varying fom 0 to $65m$ perturbing edges, where m is the number of edges in G. The result of SCE clustering algorithm is poor. For MCE clustering algorithm, compared to the balanced case depicted by graph 2.13, the efficiency is deteriorated. Yet if the perturbation is below $20m$ the quality of the result is satisfying.

4.6 Perturbation of the stochastic blockmodel

Up to now, we only considered random perturbations in the form of edges that are randomly appended to a block-cycle. In this section, we analyse the case where we perturb the stochastic blockmodel. Instead of a random perturbation, we perturb the model itself: this choice illustrates the situation where the dynamic of the graph is more complex than a block-cycle, the underlying blockmodel may thus be a superposition of a cyclic stochastic blockmodel and other kinds of blockmodels.

We consider graphs G generated by a stochastic blockmodel with parameters

- $n = 1000$ nodes,
- $k = 10$ blocks,
- vector of probabilities of memberships $\rho = [0.1, 0.1, ..., 0.1]$ (balanced blocks).

We only let the matrix P of probabilities of connexions of the stochastic blockmodel vary.

Firstly, let us analyse what happens with $k = 10$ and a matrix of connexions of the form

$$P = \begin{bmatrix} \epsilon & \delta & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \delta & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \delta & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \delta & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \delta & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \delta & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \delta & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \delta & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \delta \\ \delta & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \end{bmatrix}$$

for δ and ϵ in $[0, 1]$. Given a graph G generated by this stochastic blockmodel, two nodes a and b of G have probability δ of being connected if they belong to consecutive blocks and probability ϵ otherwise. Figures 2.23 and 2.22 show the evolution of the error as a function of ϵ respectively for $\delta = 0.1$ and $\delta = 0.7$. In both cases, the error reaches a maximum when $\epsilon \simeq \delta$. Indeed, when $\epsilon \simeq \delta$, the graph generated is essentially a random graph with probability $\epsilon \simeq \delta$ of having an edge between any pair of nodes, therefore there is no obvious block cyclic structure to detect. Another interesting observation is the fact that the block membership error tends to 0 when $\delta >> \epsilon$ but also when $\delta << \epsilon$ especially for MCE clustering algorithm. This means that MCE clustering algorithm also succeeds in detecting the blocks of the "complement" of a block-cycle, namely for graphs G generated by a stochastic blockmodel with a particular matrix of connexions: two nodes a and b have high probability of being connected in all cases but when a and b belong to consecutive blocks in a cyclic structure (figure 2.21 depicts an example of such graph).

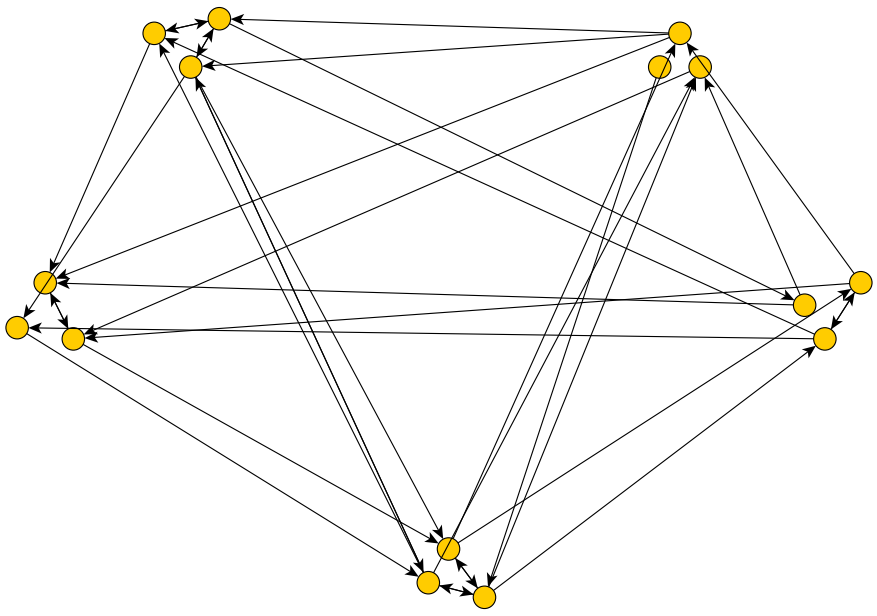


Figure 2.21: Example of "complement" of a block-cycle: blocks are arranged in a cycle such that there is a high probability of having an edge (a, b) only if a and b do not belong to consecutive blocks in the cycle. Namely there are edges pretty much everywhere except between consecutive blocks of the block-cycle in the clockwise orientation.

Secondly, let us analyse the effect of a random perturbation of matrix P. We proceed exactly in the same way as for random perturbations of the adjacency matrix of a block-cycle in previous sections but we append nonzero entries randomly to matrix P of a cyclic stochastic blockmodel.

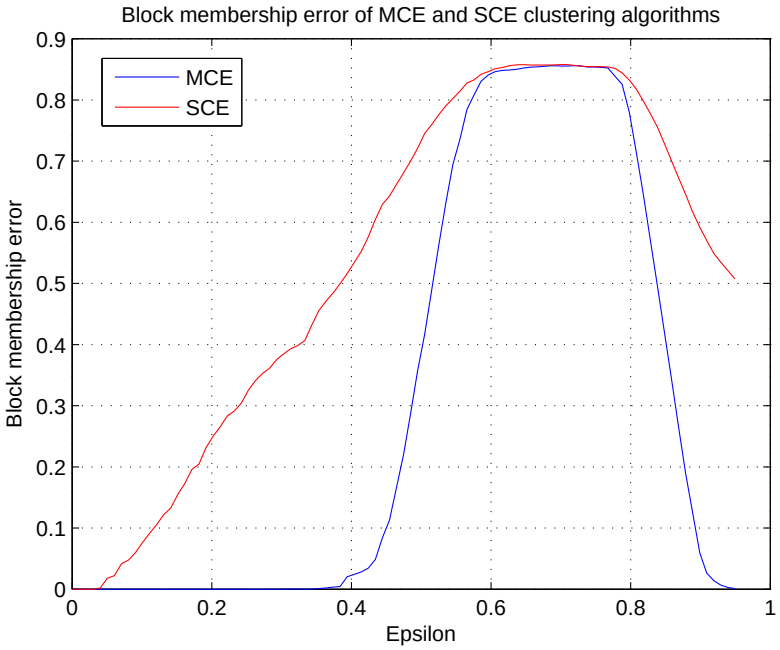


Figure 2.22: Evolution of the block membership error for a graph G of 1000 nodes, 10 balanced blocks and matrix P with parameter $\delta = 0.7$ and ϵ varying from 0 to 1. The error reaches a maximum when $\epsilon \simeq \delta$. The error is low both in cases where $\epsilon << \delta$ and $\epsilon >> \delta$ ("complement" of a block-cycle), especially for MCE clustering algorithm.

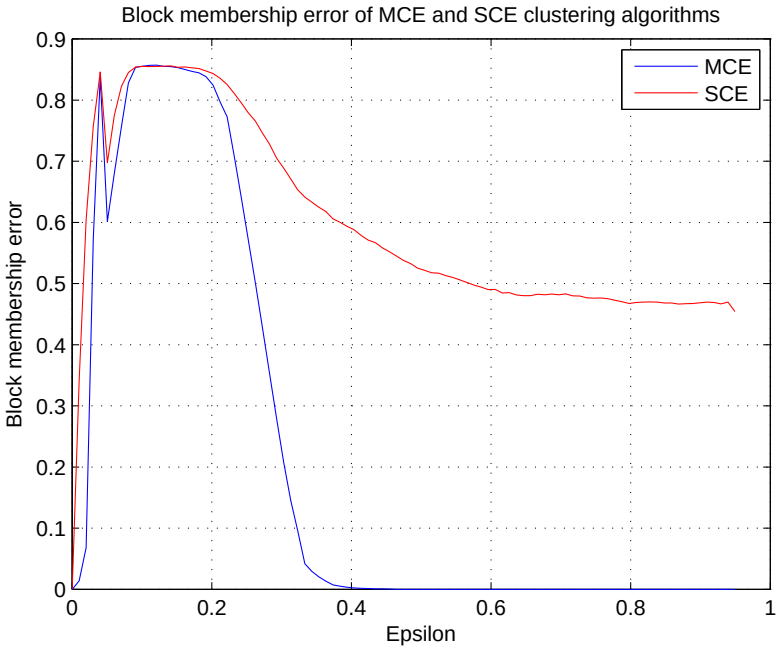


Figure 2.23: Evolution of the block membership error for a graph G of 1000 nodes, 10 balanced blocks and matrix P with parameter $\delta = 0.1$ and ϵ varying from 0 to 1. The error reaches a maximum when $\epsilon \simeq \delta$. The error is low both in cases where $\epsilon << \delta$ and $\epsilon >> \delta$ ("complement" of a block-cycle), especially for MCE clustering algorithm.

This situation can be interpreted in the following way: we start with a block-cycle and add a few shortcuts, namely flows going across the block-cycle and connecting blocks that are not consecutive. This situation is depicted by figure 2.25 for $k = 5$ blocks.

For our tests, we consider an initial block-cycle containing $k = 10$ blocks with a matrix P having

10 nonzero entries with value 0.7, as before. Then, we progressively append $1, 2, 3, ...k(k-1) = 90$ nonzero entries to P (with value 1) and measure the evolution of the block membership error which is depicted in graph 2.25. The result is of course deteriorated with growing number of perturbing entries in P. However, whenever the number of perturbing entries in P is below 20, the block membership error is below 0.2.

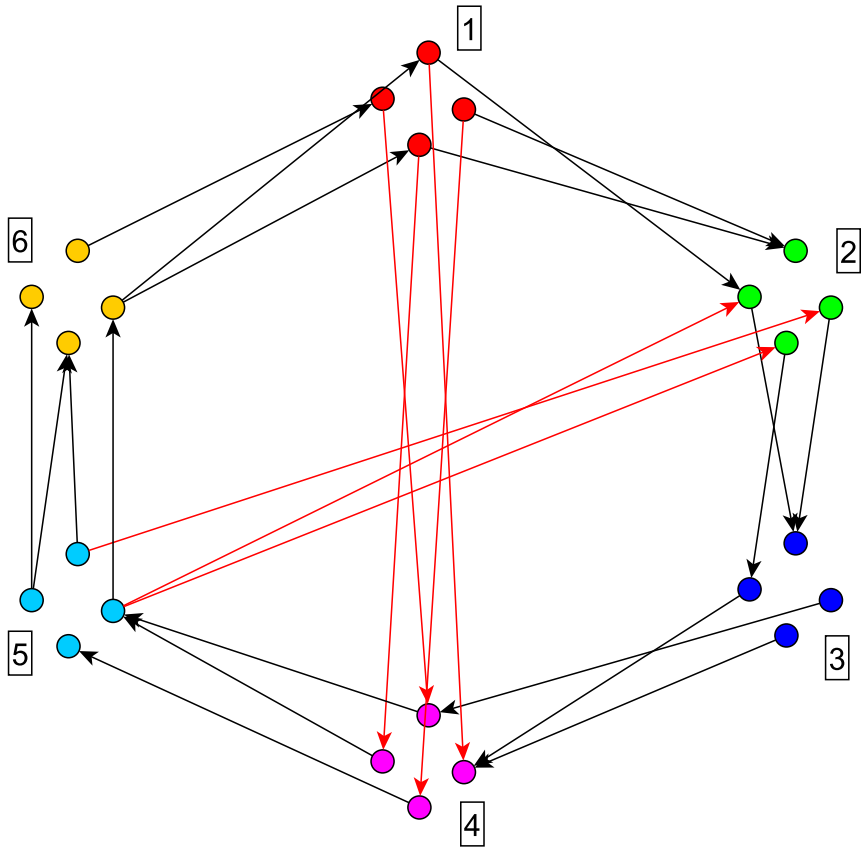


Figure 2.24: Block-cycle with 6 blocks, a flow of edges is appended from block 1 to block 4 and from block 5 to block 2, hence the perturbation on P is represented by entries $P_{14} > 0$ and $P_{52} > 0$.

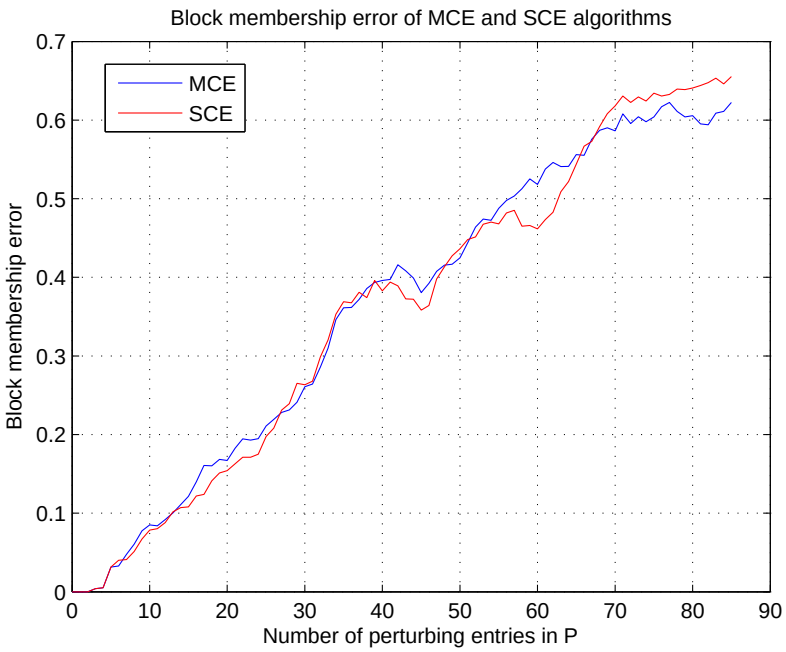


Figure 2.25: Evolution of the block membership error for a slightly perturbed block-cycle $\tilde{G}$. $\tilde{G}$ is a perturbed version of a block-cycle G of 1000 nodes, 10 balanced blocks and a probability of 0.7 for two nodes belonging to consecutive blocks of being connected by an edge. $\tilde{G}$ is obtained by appending r 1's randomly to matrix P of connexions of the stochastic blockmodel for r varying fom 0 to 85 perturbing entries. Whenever the number of perturbing entries in P is below 20, the block membership error is below 0.2.

4.7 Summary of results

In this section, we summarize the observations of last sections based on tests on synthetic data.

In all cases, we have seen that MCE clustering algorithm has better performances than SCE clustering algorithm in terms of block membership error. Moreover, the difference in time of computation is not significant. Hence, we recommend to use MCE clustering algorithm.

Table 2.2 summarizes observations regarding MCE clustering algorithm only. For each characteristic (type of perturbation or efficiency of certain parts of the algorithm such as the local search step), we characterize the block membership error and/or the robustness of MCE clustering algorithm with respect to a certain type of perturbation.

Number of points	When n grows, the error decreases.
Local search	• particularly efficient when n is low (typically when $100 \leq n \leq 1000$, • uncertain effect when the perturbation is high.
Unbalanced blocks	robust
Unbalanced flows	• robust if probability of edge (a, b) for a and b belonging to consecutive blocks is above 0.5 for all pair of consecutive blocks, even if this yields unbalanced flows between different pairs of consecutive blocks in the block-cycle • satisfactory result in the presence of a bottleneck in the block-cycle.
Mixing several models	when considering the matrix of connexions P of the cyclic stochastic blockmodel of k blocks, if up to $2k$ perturbing 1's are appended to the matrix, creating up to $2k$ "short-cuts" in the block-cycle (as shown for instance in figure 2.25), the error stays below 0.2.

Table 2.2: Summary of observations based on tests on synthetic data

Chapter 3

Spectral algorithm for the detection of acyclic patterns in directed graphs

In previous section, we introduced the concept of block-cycles and a spectral algorithm to detect this kind of structure. However, it is hard to find any real world data where block-cycles clearly appear. The main reason is that databases may have a block structure but the connexions between blocks is governed by laws that are far more complicated than a cyclic dynamic. As we mentioned in previous sections, stochastic blockmodels can efficiently capture the structure of various real-world graphs. But the connections between blocks in such model rarely form cycles (we would rather have combinations of cycles and acyclic structures, such as paths or directed trees).

However, in this section we show how a slightly modified version of MCE clustering algorithm succeeds in detecting a particular kind of directed graphs where nodes can be clustered with an acyclic pattern of connexions between clusters. As we shall see this type of structure appears in various types of real-world database including trophic networks and web-based graphs.

In 1950, the American sociologist George Homans published *The Human Group* [37] which was one of the first books that deeply analysed the structure and the dynamic of small communities of human beings. One of his hypothesis is that small social groups inevitably tend to combine small communities and a ranking law. To make this notion clear, let us focus on the social groups of cinema and television actors. One can model a directed relation between actor i and actor j as "i likes j" (in the sense of admiration). Homans's assumption expresses the fact that the whole community of actors can be partitioned into groups. Actors within a group are only connected by mutual (symmetric) links (if i likes j then j likes i in return). However, bonds between groups are governed by asymmetry: the majority of links between any pair of groups A and B are oriented from A to B or from B to A but rarely in both directions. Homans's assumption even goes further: connections between groups form an acyclic network, hence there exists an underlying ranking law that governs the relations between communities. Coming back to the example of actors, although it seems a little caricatural, one could imagine that actors are partitioned into such communities. Worldwide famous actors (Hollywoodian in most cases) are on the top of the list and often like (or at least know) each other. Then come the actors that are famous in some important countries, such as French actors that are essentially famous in France, these actors know and may appreciate Hollywoodian actors but the opposite is less likely. Lower in the hierarchy, one would find some regional or local actors, such as some french speaking Belgian actors that are integrated in the French cinema industry

but are less renowned than French actors. Figure 3.1 summarizes the example of actors[1]

The example of actors may seem artificial but it captures Homans's idea: the presence of two types of connections in social groups: symmetric links (that denote an equal rank between two actors) and asymmetric links (that express a notion of hierarchy such as admiration or even some kind of prey-predator relations...).

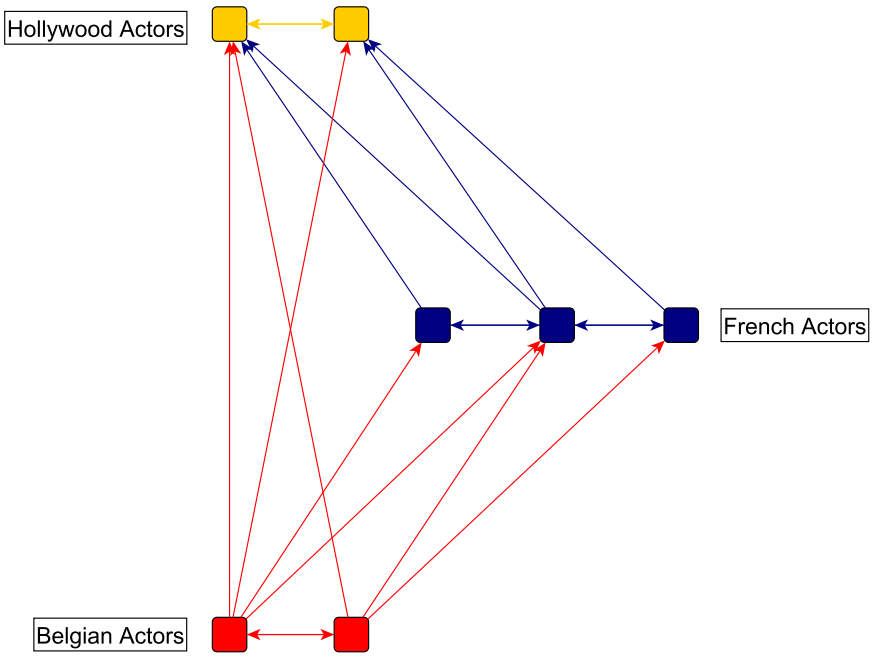


Figure 3.1: Network of relationships between Hollywoodian, French and Belgian actors. An edge (a, b) means actor a has heard of actor b.

Homans's book focuses on the social implications of this theory. Since then, several approaches intended to formalize and test his assumptions on real-world databases. Davis and Leinhardt (in [38], 1972) gave the first expression of Homans's proposition into graph theoretical terms where ideal communities are viewed as cliques and a ranking rule governs the existence of asymmetric links between these. Their goal was essentially to verify if Homans's conjecture was verified in practice. They find necessary and sufficient conditions for graphs to fulfil their ideal definition and suggest that these conditions are more likely to be satisfied in social groups than in random graphs. This assumption is verified on various acquaintance based networks (namely social graphs where edges represent relations of friendship, knowledge or admiration).

In 1998, pursuing Davis's and Leihnardt's work, Doreian et al. proposed a new approach in [39] defined as the *Symmetric-Acyclic Decomposition of Networks*. They define an ideal blockmodel similar to the one presented in [38] and formulate a quality function that tends to fit the ideal structure into a graph when minimized. A drawback of their method is the lack of a systematic algorithm solving their optimization problem, hence a difficulty to apply their approach to large graphs.

To our knowledge, since then there were no explicit reference to Homans's and Leinhardt's theories in the mathematical and computer science related literature. However some authors further

[1]This is of course an example intended to clarify the concept introduced by Homans, we do not mean that Belgian or French actors are not as good as American actors.

analysed the concept of hierarchy in directed graphs such as Gupte et al. in [40] who aims at *Finding Hierarchy in Directed Online Social Networks*. However, in their approach, they focus on the presence of asymmetric links in directed networks (penalising even the presence of undirected edges between nodes of equal rank) and they suggest a hierarchy built "node-by-node" rather than a community based ranking, which makes their approach essentially different from Homans's theories.

The ten last years have seen the emergence of new fields of applications for Homans's theory, for instance worldwide virtual social networks such as Twitter, Facebook or even Youtube in a sense. A new challenge is to understand what principles govern these new types of interaction and, in particular, to test if Homans's conjecture is also validated on social networks, suggesting a large-scale social hierarchy on virtual networks. Another challenge would be to check if the model cannot describe the dynamic of graphs that are not based on social networks. Hence, potential fields of application could include the following types of networks.

- Social networks that involve directed links (such as Twitter or Youtube for instance).
- Networks based on the interaction between animals (with trophic networks as the archetype of graphs governed by an acyclic model).
- Networks of transfer of cash between Internet Service Providers around the world.
- Networks based on sport competitions where nodes are teams and directed edges symbolise victory and defeat links. In particular, one could analyse the correlation between the ranked communities based on Homans's theory and the official hierarchy between groups of teams (eg, in football competitions, the Champions league, the Europa league...).

Hence, the growing number of fields of applications and the need for methods able to detect structures based on Homans's theory in large graphs motivates the design of a new clustering algorithm that is presented in this chapter. In a sense, this algorithm provides an automatic tool to test the impact of Homans's conjecture when dealing with very large graphs encountered in various domains. In this chapter we first describe an algorithm for the detection of symmetric-acyclic networks. Then we present some applications of the method to real-world data. As we shall see, this algorithm is closely related to MCE clustering algorithm.

1 Spectral algorithm for the detection of symmetric-acyclic networks

The purpose of this section is to formalize Homans's conjecture and to provide an automatic method to check if it is verified for a given network. We first give a formal definition of the structure described in Homans's theory: a network with an underlying decomposition into blocks with the presence of bidirectional edges inside blocks and unidirectional edges between blocks, yielding an acyclic pattern of connexions between blocks. We then formulate an optimization problem for the detection of blocks in such structure and we provide a spectral algorithm closely related to MCE clustering algorithm that computes an approximate solution to the optimization problem. Newt, we compare our algorithm to PageRank algorithm as both share similarities while being essentially different in the information they provide about networks. Finally, we describe some further extensions including technical details of the method and we also briefly describe the performance of the algorithm on synthetic data.

In next section, we will describe the performance of our algorithm on real-world data.

1.1 Block-symmetric-acyclic networks and block-acyclic networks

In this section, we give a mathematical description of the network structure defined by Homans. Such formulation can be found in [39] but our definition is more graph oriented.

Definition 13 (*block-symmetric-acyclic network*)

Given a positive weighted directed graph $G = (V, E, W)$, G is a block-symmetric-acyclic network if there exists an integer k and a block membership function $f : V \to \{1, ..., k\}$ such that

- *$\{u : f(u) = i\} \neq \emptyset$ for any $i \in \{1, ..., k\}$,*
- *for all $(u, v) \in E$,*

$$\begin{array}{l} \text{either } f(u) = f(v) \text{ and } (v, u) \in E \\ \text{or } f(u) < f(v) \end{array} .$$

Figure 3.2 shows an example of block-symmetric-acyclic network.

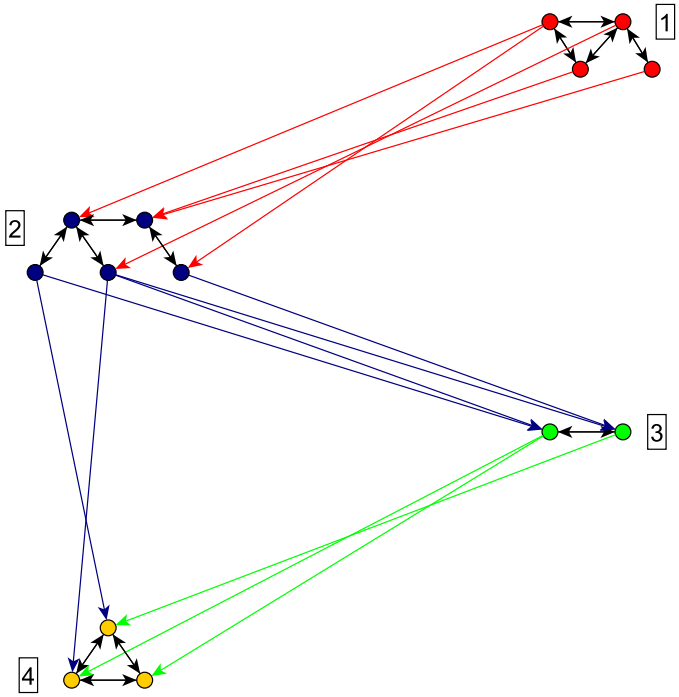


Figure 3.2: Abstract representation of a block-symmetric-acyclic network: there are four blocks respectively coloured in red, green, blue and yellow, the number next to each group is its rank according to definition 13.

As we will see further, in some cases, we wish to focus on the block-acyclic structure of the network, especially when it is not likely to encounter bidirectional intra-cluster edges. To encompass this case, we also give a definition of *block-acyclic network* (definition 14).

Definition 14 (*block-acyclic network*)

Given a positive weighted directed graph $G = (V, E)$, G is a block-acyclic network if there exists an integer k and a block membership function $f : V \to \{1, ..., k\}$ such that

- $\{u : f(u) = i\} \neq \emptyset$ *for any* $i \in \{1, ..., k\}$,
- *for all* $(u, v) \in E$, $f(u) < f(v)$

In order to generate block-symmetric-acyclic networks automatically, we define a block-symmetric-acyclic stochastic blockmodel.

Definition 15 (*block-symmetric-acyclic stochastic blockmodel*)

A block-symmetric-acyclic stochastic blockmodel is a stochastic blockmodel with the following parameters:

- *a positive integer* k
- *a vector* $\rho \in [0, 1]^k$ *such that* $\sum_i \rho_i = 1$,
- *a strictly upper triangular matrix* $P \in [0, 1]^{k \times k}$,
- *a vector* $q \in [0, 1]^k$.

An unweighted directed graph $G = (V, E)$ is generated by a block-symmetric-acyclic stochastic blockmodel if there exist a block membership function $f : V \to \{1, ..., K\}$ satisfying the following condition,

- *block membership:* $\forall u \in V$,
$$\mathcal{P}\left[f(u) = i\right] = \rho_i$$
- *symmetric links:* $\forall u, v \in V$
$$\mathcal{P}\left[(u, v) \in E | f(u) = f(v) = i\right] = q_i$$
$$\mathcal{P}\left[(u, v) \in E | f(u) = f(v),\, (v, u) \in E\right] = 1$$
- *asymmetric links:* $\forall u, v \in V$,
$$\mathcal{P}\left[(u, v) \in E | i = f(u) \neq f(v) = j\right] = p_{ij}$$

It is clear that any graph generated by a block-symmetric-acyclic stochastic blockmodel is a block-symmetric-acyclic network. For the sake of simplicity, graphs used to test our algorithm are generated by a block-symmetric-acyclic stochastic blockmodel with chosen parameters (k, ρ, P, q). We will attach more importance to asymmetric links than bidirectional connections: in particular, we want our algorithm to be able to detect the ranked blocks even when there are no internal links ($q = [0, ..., 0]^T$).

Regarding a block-symmetric-acyclic network, two kinds of perturbations may occur.

1. Violation of symmetry if the adjacency matrix restricted to a block is not symmetric.
2. Violation of ranking law if there exists some link from block A to block B where B is ranked lower than A.

To generate slightly perturbed versions of block-symmetric-acyclic networks, we generate a graph by the block-symmetric-acyclic stochastic blockmodel and then add a few symmetry or ranking errors.

Regarding block-acyclic networks, only one kind of perturbation may occur: violation of the ranking law, namely when $(a, b) \in E$ but $f(a) \geq f(b)$.

1.2 Optimization problem for the detection of symmetric-acyclic networks

In this section, we formulate the problem of detection of symmetric-acyclic networks. Given a block-symmetric-acyclic network $G = (V, E, W)$ with adjacency matrix W (weighted or unweighted) and a positive integer k, we seek the block membership function $f : V \to \{1, ..., k\}$ that recovers the blocks of G. Obviously, we also want to be able to detect the blocks in slightly perturbed block-symmetric-acyclic network (namely a block-symmetric-acyclic network with a few perturbing edges).

We propose two different criteria: the *acyclic criterion* for block-acyclic networks and the *symmetric-acyclic criterion* for block-symmetric-acyclic networks. We first define the absolute and relative acyclic criterion.

Definition 16 (*Acyclic criterion*)

$$\begin{array}{rcl} C(f) & = & \sum_{u,v:f(u)<f(v)} W_{uv} - W_{vu} \\ C_r(f) & = & \frac{C(f)}{|E|} \end{array} .$$

This criterion is increased by the presence of a directed edge (u, v) such that $f(u) < f(v)$ but it is penalized by the existence of an edge (u, v) such that $f(u) > f(v)$. Clearly for an unperturbed block-acyclic network, the maximum criterion is achieved when the block membership function exactly captures the blocks of the network. This objective function does not explicitly take bidirectional intra-cluster links into account. To take the effect of bidirectional links into account in the **unweighted** case, we suggest to use another criterion, the *symmetric-acyclic criterion*.

Definition 17 (*Symmetric-acyclic criterion*)

$$\begin{array}{rcl} \tilde{C}(f) & = & \sum_{u,v:f(u)<f(v)} W_{uv} - W_{vu} + \sum_{u,v:f(u)=f(v)} W_{uv}W_{vu} \\ \tilde{C}_r(f) & = & \frac{\tilde{C}(f)}{|E|} \end{array} .$$

Like $C(f)$, criterion $\tilde{C}(f)$ is increased by the presence of an edge (u, v) such that $f(u) < f(v)$ but it is penalized by the existence of an edge (u, v) such that $f(u) > f(v)$. It also takes into account the presence of intra-cluster edges in the following way: if $f(u) = f(v)$, the contribution to the criterion is strictly positive if $W_{uv} = W_{vu} = 1$ and it is zero if $W_{uv} = W_{vu} = 0$ or $W_{uv} \neq W_{vu}$. This criterion is only suitable for unweighted graphs.

Depending on the application, one can suggest any of the two criteria. If the dataset is expected to contain strong bidirectional intra-cluster ties, the second criterion should be used whereas the first one is suited for datasets where the proportion of bidirectional intra-cluster edges is negligible compared to the contribution of asymmetric inter-cluster bonds. Unless otherwise specified, we rather use the criterion $C(f)$ which gives the following optimization problem.

Definition 18 (*Symmetric-acyclic problem*)

Given a directed and possibly positive weighted graph $G = (V, E, W)$ and a positive integer k, find a block membership function $f : V \to \{1, ..., k\}$ solving the maximization problem

$$f = \arg\max_{\hat{f}} K_r(\hat{f}).$$

where $K_r(f)$ can either denote acyclic criterion C_r or symmetric-acyclic criterion $\tilde{C}_r$.

1.3 Block-cycle based algorithm for the detection of symmetric-acyclic networks

In this section, we present a heuristic algorithm solving problem 18. This heuristic is essentially based on the detection of a block-cyclic structure in a modified version of a graph.

Let us consider a block-symmetric-acyclic network $G = (V, E, W)$. We first focus on the case where there are no intra-cluster bidirectional links, hence E only consists of directed edges between clusters. Figure 3.3 shows an abstract representation of such network. The basic idea behind our algorithm is to transform this acyclic network to a block-cycle shaped network. Figure 3.4 shows what we intend to do in the ideal case. Adding a link from the cluster with highest rank to the one with lowest rank would clearly create a loop and highlight a block-cyclic structure in the graph. Hence, we obtain a perturbed block-cycle such as the one in figure 3.4 and, as we showed that MCE clustering algorithm is robust to perturbations, we may hope that running MCE clustering algorithm on such modified graph recovers the block membership

of nodes.

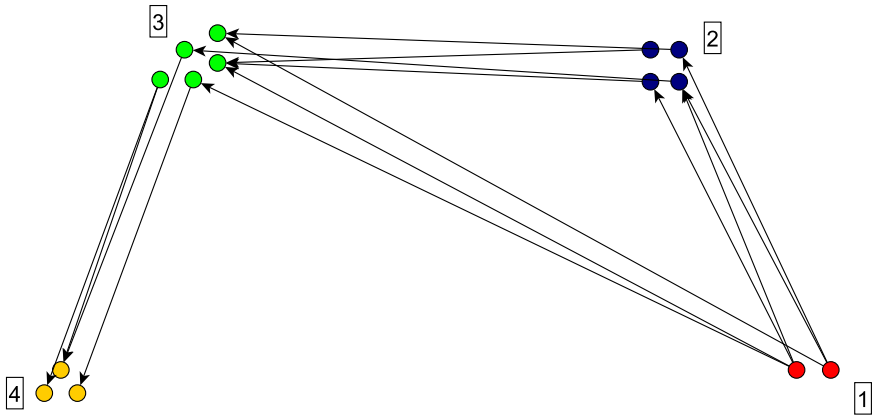


Figure 3.3: Block-acyclic network with four blocks (without bidirectional intra-cluster edges).

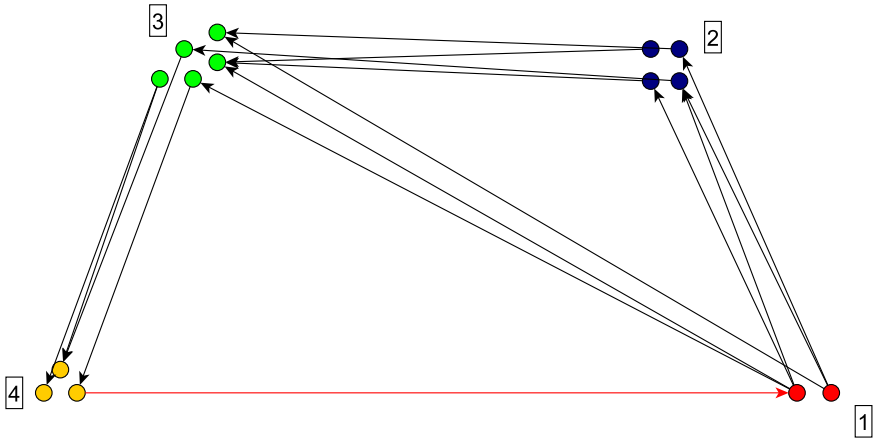


Figure 3.4: Block-acyclic network with four blocks (without bidirectional intra-cluster edges). The edge in red closes the cycle.

Appending an edge from the cluster with highest rank to the one with lowest rank is not an easy task as it requires to know the ranking in advance. So we rather use a method suggested by the Google matrix which can be summarized by the following modified definition of Laplacian.

Definition 19 (*Generalized Laplacian*)

Given a directed, possibly positive weighted, graph $G = (V, E, W)$ with $|V| = n$ and a parameter $\epsilon \in [0, 1[$, the generalized Laplacian $\Delta_G \in \mathbb{R}^{n \times n}$ is defined in the following way. We first define matrix $P_G \in \mathbb{R}^{n \times n}$

$$(P_G)_{ij} = \begin{cases} \frac{1}{d_i^{out}} w_{ij} & \text{if } d_i^{out} > 0 \\ \frac{1}{n} & \text{otherwise} \end{cases}.$$

Then Δ_G is defined as

$$\Delta_G = I - ((1 - \epsilon) P_G + \frac{\epsilon}{n} \mathbf{1}\mathbf{1}^T)$$

where $\mathbf{1}\mathbf{1}^T$ is the $n \times n$ matrix of all one's.

We note that $(1-\epsilon)P_G + \frac{\epsilon}{n}\mathbf{1}\mathbf{1}^T$ is the transpose of the Google matrix A_G associated to graph G with parameter ϵ, hence $\Delta_G = I - A_G^T$. There are two main differences with the classical definition of Laplacian.

1. If a node is out-isolated (zero out-degree), we artificially add edges connecting it to all other nodes of the graph.

2. We also virtually add edges with small weight from any vertex to all other vertices so that the Laplacian is a weighted average of the Laplacian of the graph and the Laplacian of a complete graph:
$$\Delta_G = (1-\epsilon)(I - P_G) + \epsilon(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T).$$

The first modification artificially adds out-edges from isolated nodes to any other vertex including edges from the cluster with highest rank to the cluster with lowest rank which is what we intended to do. The presence of the second modification is justified by robustness considerations in the presence of strong perturbations but choosing $\epsilon = 0$ often gives the same result.

Hence, the procedure we suggest for the detection of blocks in a block-acyclic network (with no bidirectional intra-cluster edges) is

1. Compute the Laplacian matrix Δ_G,

2. Run MCE clustering algorithm on Δ_G.

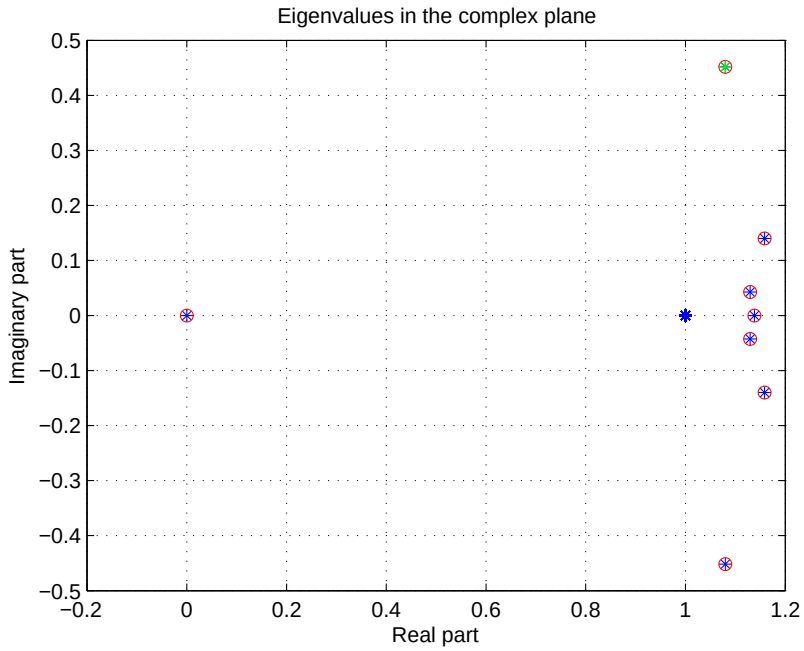


Figure 3.5: Eigenvalues of the generalized Laplacian Δ_G of a block-symmetric-acyclic network (without bidirectional intra-cluster edges) with parameter $\epsilon = 10^{-3}$, cycle eigenvalues found by Arnoldi algorithm are circled in red, the first cycle eigenvalue is in green. The graph is generated with $n = 400$ nodes, $k = 8$ blocks, an equal probability of membership of each vertex to any block and a probability 0.7 of having an edge from block i to any block j with $j > i$.

What should be done when bidirectional intra-cluster links are also present in the graph? The presence of these links strongly affects the spectrum of the Laplacian Δ_G so we remove them

by first taking the anti-symmetric part of the adjacency matrix and removing negative entries:

$$\tilde{W} \leftarrow \frac{1}{2}(W - W^T)_+$$

One could wonder if we could as well take advantage from the presence of bidirectional intra-cluster edges. MCE clustering algorithm is specifically designed to detect block-cycles in graphs and its results are deteriorated with the presence of intra-cluster links. However, some local search post-processing can improve the quality of the result by taking advantage of the presence of intra-cluster bidirectional links (see section 1.5.2).

Below is a formal description of this algorithm that we define as BAS clustering algorithm for *Block-Acyclic Spectral* clustering algorithm.

Algorithm 1.1: BAS clustering algorithm

Input: Adjacency matrix $W \in \{0,1\}^{n \times n}$;
Parameters: $k \in \{2, 3, ..., n\}, \epsilon \in [0, 1[$;
Step 1: $W \leftarrow \frac{1}{2}(W - W^T)_+$;
Step 2: Compute the generalized graph Laplacian $(1-\epsilon)(I - P_G) + \epsilon(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)$;
Step 3: Find the k cycle eigenvalues (the k eigenvalues located furthest from $(1, 0)$ in the complex plane) and store the associated cycle eigenvectors as the columns of a matrix $\Gamma \in \mathbb{C}^{n \times k}$.;
Step 4: Consider each row of Γ as a point in $\mathbb{C}^k$ and cluster these points using a k-means algorithm. Let $\phi : \{1, ..., n\} \to \{1, ..., k\}$ be the function assigning each row of Γ to a cluster;
Step 5: Compute the estimation of block membership function f: $f(u) = \phi(u)$ for all $u \in \{1, ..., n\}$;
Output: estimation of block membership f;

1.4 Comparison between BAS clustering algorithm and PageRank

In this section, we briefly analyse the similarities between PageRank and BAS clustering algorithm that we introduced in last section. PageRank algorithm is the ranking algorithm used by Google for sorting webpages. Generally speaking, this method computes a ranking of nodes of a directed graph such that a node a has a high rank if there are some edges of the form (b, a) with b of high rank. The algorithm relies on the definition of Google matrix

$$A_G = (1-\epsilon)P_G^T + \frac{\epsilon}{n}\mathbf{1}\mathbf{1}^T$$

which turns out to be the transpose of the Generalized Laplacian defined in previous section. Using Perron-Frobenius theory (see [33] for more details), one can show that the unique dominant eigenvalue of A_G is 1. The ranks of nodes are then given by the entries of the associated eigenvector.

PageRank and BAS clustering algorithms are related to the extent that they both make use of the spectrum of Google matrix and they both compute some ranking of the nodes of a directed graph. The major difference is that PageRank algorithm uses the dominant right eigenvector of A_G whereas BAS clustering uses the k dominant left eigenvectors of A_G. There is also a difference in terms of goal: PageRank finds a random walk based rank that is well defined

for any graph, BAS clustering finds a ranking of blocks of nodes and hence it is well defined only if the network has a block-acyclic structure. The ranks computed by both algorithms is not even expected to be related in a way. Indeed, nodes within a block of a block-acyclic or a block-symmetric-acyclic network can have very different ranks according to PageRank algorithm depending on their connections with other nodes inside or outside of the block.

1.5 Further extensions of BAS clustering algorithm

As we did for MCE clustering and SCE clustering algorithms, we introduce in this section some further extensions making the result of BAS clustering algorithm more accurate. We do not develop the steps that are already included in the cycle detection part of BAS clustering algorithm (namely the computation of cycle eigenvalues using Arnoldi algorithm as described in section 2.4.2 of chapter 2 and the k-means step explained in section 2.4.3 of chapter 2). We first focus on the extraction of the cluster ranking from the result of BAS clustering algorithm, then we show how to improve this solution by a local search procedure.

1.5.1 Recovering the ranking of blocks

In this section, we show how to recover the ranks of blocks in a block-acyclic network where the rank is defined by a sort of global topological order between blocks in a block-acyclic network such as suggested in figure 3.2.

As we have seen in section 2.4.1 of chapter 2, one can recover the order in which blocks appear in a block-cycle using algorithm 2.3 of chapter 2.

Given a directed graph $G = (V, E, W)$, possibly with positive weights, and a positive integer k, we assume we estimated the block membership function $f : V \to \{1, ..., k\}$ with BAS clustering algorithm. We also assume we computed the cyclic order of blocks returned by algorithm 2.3 (in section 2.4.1 of chapter 2). This ordering is stored in vector $c = [c_1, ..., c_k] \in \{1, ..., k\}^k$. We define the directed flow between two blocks as

$$Flow(i, j) = \sum_{u,v:f(u)=i,f(v)=j} W_{uv}$$

for $i, j \in \{1, ..., k\}$. We also define the out-flow of class c_i at i-th position in vector c as

$$out(c_i) = \begin{cases} flow(c_i, c_{i+1}) & \text{if } i < k \\ flow(c_k, c_1) & \text{otherwise} \end{cases}.$$

The algorithm recovering the ranking between blocks is formulated as follows.

Algorithm 1.2: Ranking algorithm

Input: Adjacency matrix $W \in \{0,1\}^{n \times n}$, cyclic order vector $c = [c_1, ..., c_k]$, estimated block membership function $f : V \to \{1, ..., k\}$;

Step 1: Compute the out flow of each block in c as

$$out(c_i) = \begin{cases} flow(c_i, c_{i+1}) & \text{if } i < k \\ flow(c_k, c_1) & \text{otherwise} \end{cases}$$

;

Step 2: Find the block with minimum out-flow

$$i \leftarrow \arg\min_l out(c_l)$$

;

Step 3: Compute the ranks of blocks as

$$\begin{array}{rcl} rank(c_i) & \leftarrow & k \\ rank(c_{i-1}) & \leftarrow & k-1 \\ & \vdots & \\ rank(c_1) & \leftarrow & k-i+1 \\ rank(c_k) & \leftarrow & k-i \\ & \vdots & \\ rank(c_{i+1}) & \leftarrow & 1 \end{array}$$

or synthetically $rank(c_l) \leftarrow ((l + k - i - 1) \mod k) + 1$;

Output: estimation of block membership f;

The principle of this algorithm is described by both figure 3.6 and 3.7. We consider the block-acyclic network G of figure 3.6. When running BAS algorithm on this instance, we append edges to G to form a graph $\tilde{G}$ similar to the one of figure 3.7. Running MCE clustering algorithm on $\tilde{G}$ allows to recover blocks 1, 2, 3 and 4 and we know that the order of appearance of blocks in the cyclic structure of $\tilde{G}$ is $1, 2, 3, 4$ (or equivalently, $3, 4, 1, 2$, etc.). In order to recover the ranks of blocks (in this case, block 1 has rank 1, block 2 has rank 2 etc.), we come back to G and measure the flow between each pair of consecutive blocks in the cycle (namely, $(1, 2)$, $(2, 3)$, $(3, 4)$ and $(4, 1)$). Clearly, the flow from 4 to 1 is very small (zero in this case): indeed, when appending edges to G to create graph $\tilde{G}$, the goal was to "close the cycle", namely to add edges from the block with highest rank to the block with lowest rank. Hence, detecting between which pair of consecutive blocks the flow is small in G allows to recover the ranks of blocks.

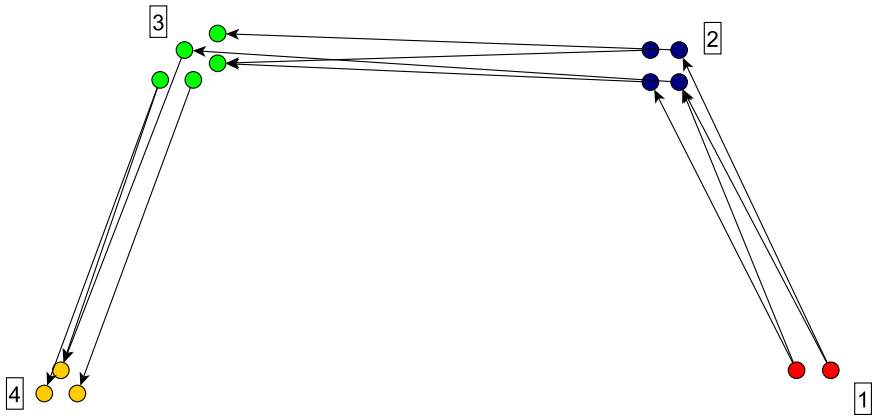


Figure 3.6: Block-acyclic network with four blocks.

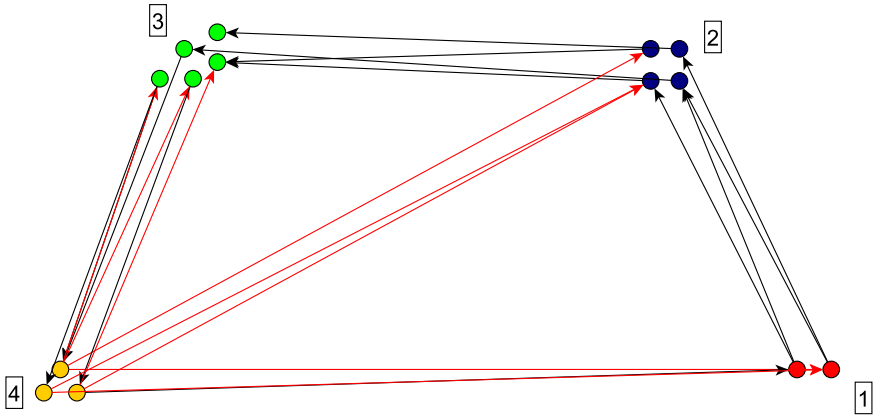


Figure 3.7: Block-acyclic network with four blocks where we appended a few out-going edges (in red) to nodes whose out-degree is zero in the initial graph of figure 3.6.

1.5.2 Local search procedure to improve the output of BAS algorithm

As pointed out in section 2.4.4 of chapter 2, a major problem of MCE clustering algorithm (and a fortiori of BAS clustering algorithm) is that it relies on k-means algorithm for the clustering step and hence for the estimation of the block membership function. As such, it may easily confuse consecutive blocks in a block-cycle. Hence the necessity of improving the solution returned by BAS clustering algorithm by a local search procedure. We follow the same logic as in section 2.4.4 of chapter 2, hence we only give an outline of the algorithm.

Considering either the acyclic criterion $C(f)$ or the symmetric-acyclic criterion $\tilde{C}(f)$ described in section 1.2, we go through all nodes of G and for each node v, we check if the criterion is improved by assigning v to block $f(v)+1$ or $f(v)-1$, if this is the case, we reassign v to block $f(v)+1$ (or to block $f(v)-1$ if it increase more the criterion). After reaching the last node (namely after one *pass*), we come back to the first one and restart the whole process until either the criterion is no longer improved or the number of passes reached a certain limit.

1.6 Test on synthetic data

In this section, we test BAS clustering algorithm on synthetic data. As the goal is rather to test the algorithm on real-world data, we do not test the method in details as we did for SCE and MCE clustering algorithms in section 4 of chapter 2.

We generate a block-symmetric-acyclic network with a block-symmetric-acyclic stochastic block-model with parameters

- $n = 800$ nodes,
- $k = 8$ blocks,
- vector $\rho = [1/8, ..., 1/8]$,
- matrix

$$P = \begin{bmatrix} 0 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0 & 0 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0 & 0 & 0 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0 & 0 & 0 & 0 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0.7 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- vector $q = [0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7]^T$.

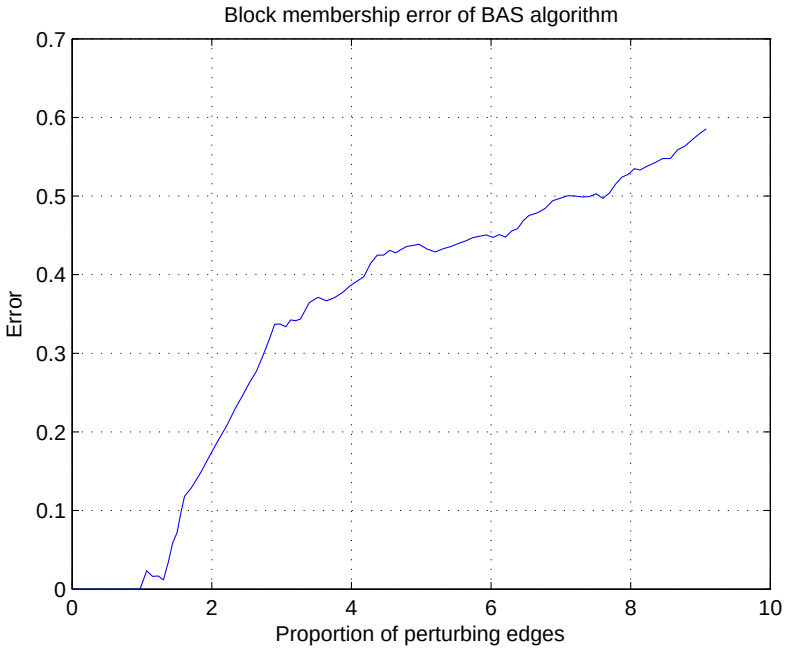


Figure 3.8

We append random edges of weight 1 to the graph and measure the block membership error as defined in section 4 of chapter 2. Graph 3.8 shows the evolution of the error when the number of perturbing edges varies from 0 to $10m$ where m is the number of edges in the original unperturbed graph. As long as the number of perturbing edges is below $2m$, the block membership error stays below 0.2.

2 Acyclic and Symmetric-Acyclic networks in real-world data

In this section, we analyse the performances of BAS clustering algorithm on real-world data. As we have seen in previous sections, the idea of the presence of an underlying hierarchical structure in large networks was first suggested by Homans in the case of social networks. As we shall see such block-acyclic structures are actually present in various types of networks. We do not test our algorithm on social graphs but further work could be done in that area, in particular to test if Homans's assumptions is also verified on social networks involving directed connexions such as `Twitter`.

We present the results obtained on four types of graphs:

1. football competition network: a graph based on the English League Championship of football ([42]),
2. trophic network: a network of predator-prey relationships in Florida bay ([43]),
3. AS network (or Autonomous Systems network): a network of peering relationships between internet service providers around the globe ([49]).

In each case, we intend to detect either block-acyclic networks or block-symmetric-acyclic networks depending on the existence of intra-community bidirectional connexions in the networks considered.

For the first and the second applications, we also give the result obtained by applying the SVD based algorithm for the detection of stochastic blockmodels (described in section 3.2 of chapter 1) and we give a qualitative comparison with the information obtained by applying BAS clustering algorithm.

Apart from the design of the method a large part of our work was to design an efficient implementation of the algorithm both in time and memory, which allows to handle graphs with a hundred thousand nodes and ten million edges in about a minute on a single computer. This suggests that this algorithm could be applied to larger graphs using parallelization.

2.1 Football competition network

In this section, we analyse a graph generated by the scores of games involving the twenty teams of Barclays Premier League in 2009. The full excel file is available in [41], it is based on official scores of games played in August 2009 which are published in [42]. The table of scores for each game is given in appendix 3.

A weighted directed graph $G = (V, E, W)$ is generated in the following way. Nodes are football teams and edges are based on scores for each game: whenever a game takes place between team A and team B,

1. if team A beats team B with a score of $4-1$ for instance, we create a directed edge (B, A) with weight 4 and a directed edge (A, B) with weight 1,
2. if the game is a draw ($2-2$ for instance), we create an edge (A, B) and an edge (B, A) both of weight 1 (which corresponds to an undirected edge of weight 1).

We expect that the nodes of such graph tend to form a block-symmetric-acyclic network where teams within a cluster have approximately the same level of football: if two teams A belonging

to the same cluster, the teams that were beaten by A or B should all belong to the same cluster (or family of clusters) and similarly teams that have beaten A or B should all belong to another family of clusters, disjoint from the first one. Moreover, by applying the local search algorithm based on the symmetric-acyclic criterion (as described in section 1.5.2 of this chapter), we also take the presence of draws into account: if a game between A and B results in a draw, then A and B are likely to fall in the same cluster.

The ranked clusters obtained by applying BAS clustering algorithm are compared to the total number of points obtained by the team. According to Barclays Premier League, the official points of teams are computed in the following way for any game of the season: if the game is a draw, both teams get one point, else the winner gets three points and the loser does not get any point. The ranking of teams published at the end of each season is essentially based on the points accumulated by each team.

We give several quantities to assess the performance of BAS clustering algorithm: the block membership of each team, the value of the objective function, the computational time and the two following $k \times k$ matrices.

1. The relative flow matrix

$$F_{ij} = Flow(i,j) = \frac{100}{m} \sum_{u,v:f(u)=i,f(v)=j} W_{uv}$$

where $f : V \to \{1, ..., k\}$ is the estimated block membership function, V is the set of vertices, W is the adjacency matrix and m is the sum of weights of all edges or

$$m = \sum_{u \in V} \sum_{v \in V} W_{uv}$$

hence, F_{ij} is the weighted percentage of the total number of edges having origin in block i and destination in block j.

2. The matrix of undirected edges S:

$$S_{ij} = \frac{100}{m} |\{(u,v) \in E : W_{uv} = W_{vu} \neq 0,\ f(u) = i,\ f(v) = j\}|.$$

As the undirected edges of the graph all have weight 1, S_{ij} is also the weighted percentage of all undirected edges having origin in block i and destination in block j.

For an unperturbed block-symmetric-acyclic network, we expect matrix F to be upper triangular, matrix S to be diagonal and matrix $F - S$ to be strictly upper triangular.

2.1.1 Parameters of the algorithm

We choose the following values for the parameters of the algorithm.

- In the definition of the generalized Laplacian (according to 19), we choose $\epsilon = 0$ as tests showed that the results do not vary significantly when choosing a nonzero value for ϵ (such as for instance $\epsilon = 10^{-3}$).

- We choose the number k of blocks to be 3. This assumption is confirmed by the graph of complex eigenvalues of the Laplacian (figure 3.9) where three eigenvalues are clearly away from $(1, 0)$ in the complex plane.

- We seek a block-symmetric-acyclic structure rather than a block-acyclic structure: considering two teams A and B, if there was a draw between A and B we want A and B to have more chances to be classified in the same block than if they had not played against each other at all. Hence, we take undirected edge into account and use a local search procedure based on definition 17 of symmetric acyclic criterion.
- We do only one pass of local search as any further pass does not increase the objective value significantly.

2.1.2 Results of the algorithm

Before presenting the result returned by the algorithm, we display the eigenvalues of the Laplacian in the complex plane, as well as the components of the first cycle eigenvector. There are only three eigenvalues that are significantly different from $(1, 0)$ in the complex plane. Components of the first cycle eigenvector also tend to cluster into three groups[2].

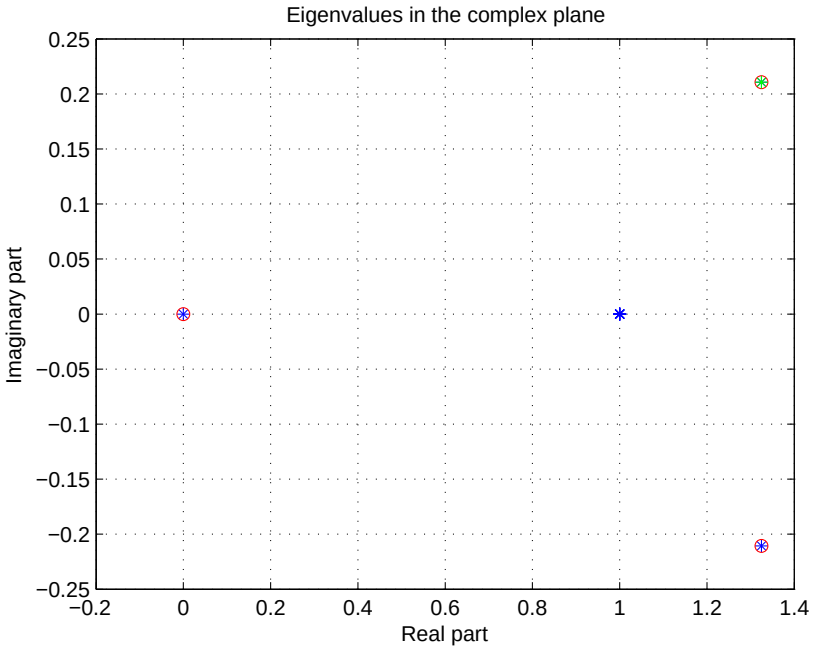


Figure 3.9: Eigenvalues of the Laplacian. Proposed cycle eigenvalues are circled in red, the first cycle eigenvalues is coloured in green.

[2]We remind that BAS clustering algorithm is based on MCE clustering algorithm and hence on the computation of all k cycle eigenvalues and cycle eigenvectors, we display the components of the first cycle eigenvector only for the purpose of visualization.

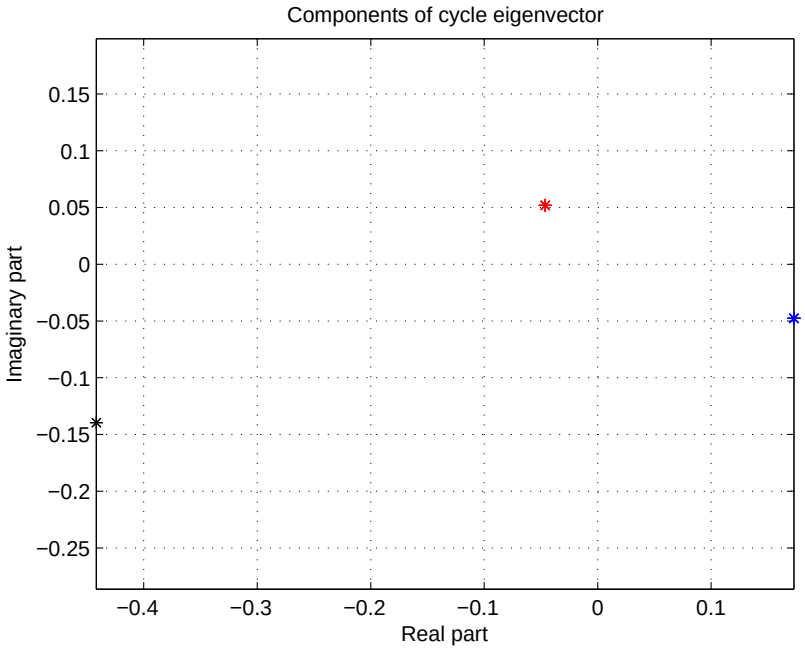


Figure 3.10: Components of the first cycle eigenvector of the Laplacian in the complex plane. Colours represent the clusters obtained by applying BAS clustering algorithm where i-th and j-th components are coloured in the same colour if the associated teams are classified in the same cluster. Several components coincide: there are 10 superposed components coloured in red, 3 coloured in black and 7 coloured in blue.

The following paragraph summarizes the results of the algorithm on the dataset of 20 teams partitioned into 3 ranked clusters.

- **Table of block membership of teams:** block 1 is the weakest and block 3 is the strongest, the number displayed next to the name of each team is the official number of points accumulated by the team in the whole season.

Block 1		**Block 2**		**Block 3**	
Team	Points	Teams	Points	Teams	Points
Sunderland	0	Bolton	6	Burnley	18
Man United	0	Wolves	6	Man City	18
Birmingham	0	West Ham	8	Liverpool	18
		Everton	8	Chelsea	18
		Arsenal	12	Fulham	18
		Tottenham	12		
		Hull City	8		
		Portsmouth	2		
		Stoke City	2		
		Wigan Ath	0		
		Black Burn	6		
		Aston Villa	8		

- **Objective value:**
  - relative acyclic criterion C_r^0 without local search post-processing:

$$C_r^0 = 0.5758$$

  - relative acyclic criterion C_r^1 with a one pass local search post-processing:

$$C_r^1 = 0.6364$$

- **Relative flow matrix:**

$$F = \begin{bmatrix} 0 & 21 & 0 \\ 9 & 6 & 55 \\ 0 & 9 & 0 \end{bmatrix}$$

- **Matrix of undirected edges:**

$$S = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- **Computational time** with a one pass local search post-processing: 0.7 seconds.

Figure 3.11 shows a simplified version of the directed graph G corresponding to the competition: to highlight the symmetric-acyclic structure, we build the graph $G = (V, E)$ where V is the set of teams and for any game between A and B,

- if A wins, $(B, A) \in E$,
- if B wins, $(A, B) \in E$,
- if the game is a draw, $(A, B) \in E$ and $(B, A) \in E$.

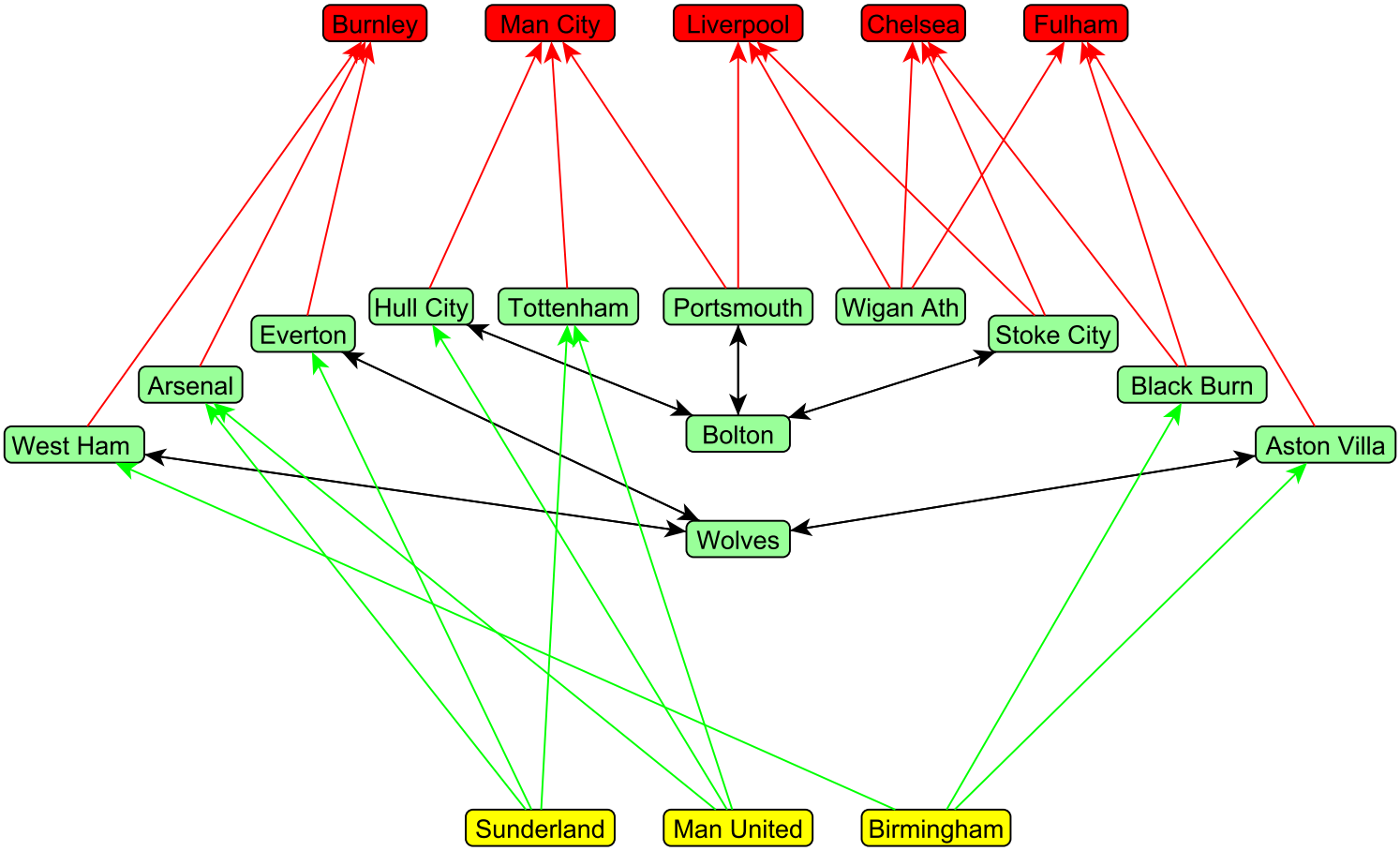


Figure 3.11: simplified version of the directed graph G corresponding to the competition: to highlight the symmetric-acyclic structure of the graph, if team A played a game against team B, we draw an edge from team B to team A if team A won the game and we draw a bidirectional edge between A and B if the game was a draw. Colours represent blocks, yellow for the weakest teams, green for the teams of medium level and red for the strongest teams. Red edges represent a victory of a strong team against a medium level team, green edges represent a victory of a medium level team against a weak team, bidirectional black edges represent draws.

We coloured nodes according to the blocks obtained by BAS clustering algorithm.

Let us make some comments regarding these results. First of all, we observe that matrix S is diagonal, matrix F is nearly upper triangular (percentage of entries that fall below the diagonal is 18%) and $F - S$ is nearly strictly upper triangular.

We see that the block membership of each team is consistent with its official number of points except for Wigan Ath: teams with 0 points belong to block 1, teams with scores strictly between 0 and 18 belong to block 2 and teams having 18 points belong to block 3. Concerning Wigan Ath, one can observe that although the club lost all its games, it played 2 games respectively against Fulham, Chelsea and Liverpool wich all belong to block 3 and hence have a very high level of football. Hence, Wigan Ath could equally belong to block 1 or block 2 because it only lost games against teams of block 3 which does not make it weaker than other teams of block 2. This is confirmed by the graph of figure 3.11 where we see that Wigan Ath was only beaten by teams of block 3.

Figure 3.11 showing the directed graph and the block membership illustrates these observations. The overall shape of the graph respects the definition of symmetric-acyclic networks: for

instance teams in block 2 beat teams in block 1, are beaten by teams in block 3 and/or have draws with teams in block 2.

2.1.3 Comparison with the result returned by the SVD algorithm

In section 3.2 of chapter 1, we introduced the SVD algorithm for the detection of a stochastic blockmodel in a graph. One of the strengths of this algorithm is its ability to detect any form of stochastic blockmodel in theory. However this characteristic is also its major weakness: user cannot give a priori information about this stochastic blockmodel, for this reason, the algorithm sometimes fails to detect a blockmodel that would be suggested by intuition. This is also the case for block-symmetric-acyclic networks. Let us now examine how the SVD algorithm performs on the example of football teams. The following table gives the block membership returned by the SVD algorithm.

Block 1		**Block 2**		**Block 3**	
Teams	Points	Teams	Points	Teams	Points
Sunderland	0	Burnley	18	Liverpool	18
Man United	0	Bolton	6	Chelsea	18
Man City	18	Birmingham	0	Fulham	18
		Wolves	6		
		West Ham	8		
		Everton	8		
		Arsenal	12		
		Tottenham	12		
		Hull City	8		
		Portsmouth	2		
		Stoke City	2		
		Wigan Ath	0		
		Blackburn	6		
		Aston Villa	8		

From this table one can see that there are three differences in comparison with the result of BAS clustering algorithm. Two good teams (Man City and Burnley) are respectively classified in block 1 and 2 instead of block 3 and one weak team (Birmingham) belongs to block to 2 instead of block 1.

2.2 Trophic network

The concept of trophic network (or food web) was first introduced in [45] by Charles Elton in 1927 and further developed in 1942 by Raymond Lindeman in [46]. The basic idea was to represent all transfers of carbon in an ecosystem as a single directed graph: nodes are living beings or any other agent that stores carbon (such as animals, carbon dioxide in the air, etc.) and a directed edge (A, B) represents a steady transfer of carbon from A to B (such as for instance a predator-prey relationship where B is the predator and A is the prey). Moreover, the weight of an edge (A, B) represents the mass of carbon transferred from A to B during a certain period of time.

As suggested by intuition, a food web is not governed by randomness. Hence, considering the trophic network of an isolated ecosystem, we should be able to partition nodes into clusters such that nodes belonging to the same clusters have roughly the same predators and preys (more specifically, they should consume and be consumed by the same families of clusters). Empirical observations often lead to the following five categories (see [44] for more details about the characterization of each group):

1. primary producers: plants and algae that produce organic material by photosynthesis,
2. primary consumers: herbivores (e.g. rabbits)
3. secondary consumers: carnivores that hunt herbivores (e.g. badgers),
4. tertiary consumers: carnivores that eat other carnivores (e.g. hawks),
5. top-level predators or apex predators: carnivores that have no other predator in their ecosystem (e.g. bears).

Decomposers are sometimes associated to primary producers to form level 1 as neither of them hunts or kills other living beings for its livelihood. Other types of "carbon holders" may be included in the network such as some input of carbon contained in the soil and absorbed by producers or the atmosphere that receives carbon dioxide from all consumers and gives carbon to the producers.

Figure 3.12 illustrates the general shape of a food web and its partition into four groups (all groups defined except top-level predators).

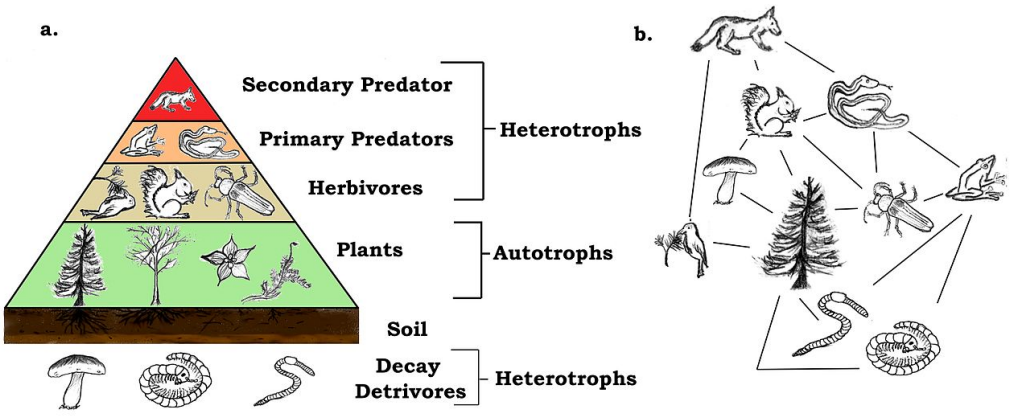


Figure 3.12: Basic structure of a food web with four groups, the figure was taken from [47].

2.2.1 Definition and computation of trophic level

A popular way to estimate the position of a species into a food web is the computation of its trophic level. The trophic level can be viewed as the level at which a species can be found in a food chain. The trophic level T_i of a species i present in a trophic network is defined in the following way (according to [48]):

$$T_i = 1 + \sum_j T_j p_{ji}$$

where p_{ji} denotes the fraction of j in the diet of i. Hence, if we define P as the transition matrix of a food network with associated adjacency matrix W namely

$$P_{*j} = \begin{cases} \frac{1}{\sum_i W_{ij}} W_{*j} & \text{if } \sum_i W_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

where P_{*j} denotes the j-th column of P.

Then the vector T of trophic levels is

$$T = (I - P)^{-1}\mathbf{1}$$

where $\mathbf{1}$ is the column vector of all ones.

The interpretation of the notion of trophic level is straightforward. If all preys of an animal are at level 2, this animal will be at level 3. If its preys belong to different trophic levels, the animal's trophic level is a weighted average of the levels of its preys plus one.

2.2.2 Trophic network of Florida Bay

In next section, we show the block membership returned by BAS clustering algorithm applied to a specific trophic network. Florida Bay is located in the south of the state of Florida in the US. It is characterized by a very rich ecosystem which is due in particular to the abundance of seagrasses which host a wide variety of species. [43] provides the data of a food web of Florida Bay consisting of 128 species or other relevant agents in the trophic network. We focused on data provided for the wet season.

Apart from living beings (with representatives from all five kingdoms), data also include the following agents:

- *respiration* which represent the carbon stored in the atmosphere in the form of carbon dioxide,
- *input* and *output* which represents all causes that introduces or removes carbon that is introduced or removed from the ecosystem (for instance due to human actions),
- other agents stocking carbon such as *DOC* or *Dissolved Organic Carbon*.

Having information about the feeding relations in the food web, we build a directed graph $G = (V, E, W)$ where a directed unweighted edge connects node A to node B if B feeds on A. For the sake of simplicity, we keep the edges unweighted but the algorithm could also be tested with weighted edges.

2.2.3 Results of BAS clustering algorithm

We choose the following values for the parameters of the algorithm.

- In the definition of the generalized Laplacian (according to 19), we choose $\epsilon = 10^{-3}$.
- We choose the number k of blocks to be 5 as suggested by the traditional number of trophic levels present in a trophic network.
- We seek a block-acyclic structure rather than a block-symmetric-acyclic structure as it is rather rare to encounter bidirectional edges in a trophic network.
- We do a five pass local search procedure as doing a sixth one does not increase the objective value significantly.

Graphs 3.13 and 3.14 respectively show the eigenvalues and the components of the first cycle eigenvector of the Laplacian of G in the complex plane. Unlike for the football competition network, it is rather unclear which eigenvalue is significantly away from $(1, 0)$ in the complex plane, hence the choice of the five eigenvalues located furthest from $(1, 0)$ is arbitrary but gives consistent results in practice as we show further. Moreover, components of the first cycle eigenvector do not cluster in an obvious way in the complex plane.

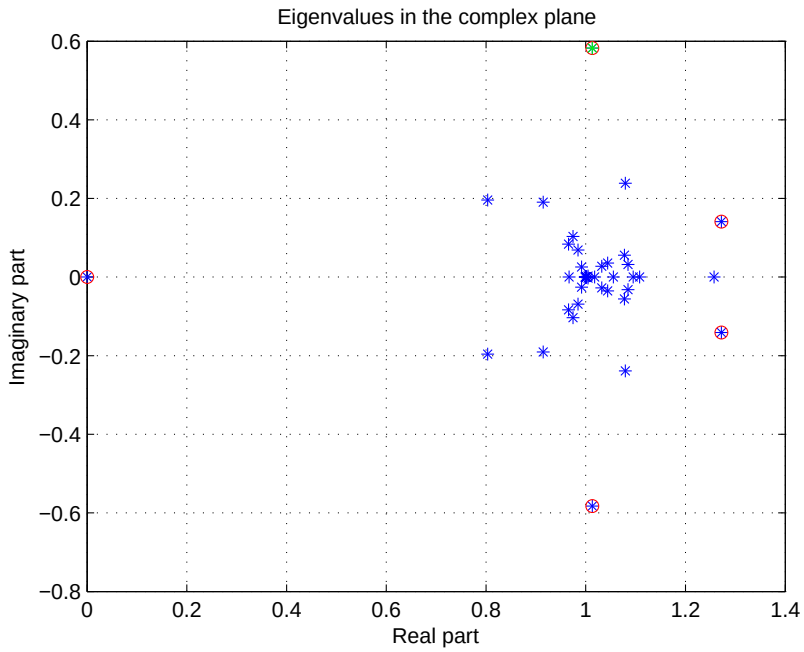


Figure 3.13: Eigenvalues of the Laplacian. Proposed cycle eigenvalues are circled in red, the first cycle eigenvalue is coloured in green.

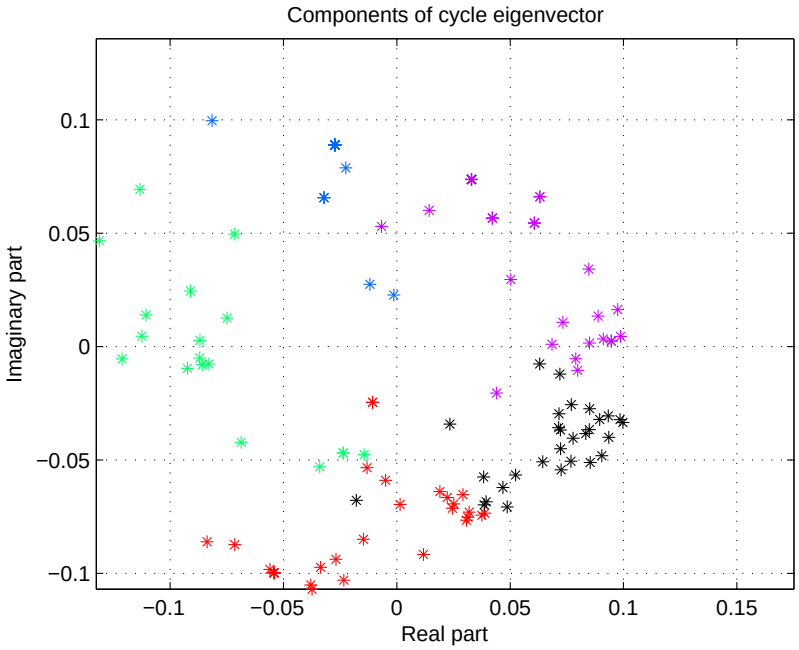


Figure 3.14: Components of the first cycle eigenvector of the Laplacian in the complex plane. Colours represent the clusters obtained by applying BAS clustering algorithm.

The following paragraph summarizes the results of the algorithm. As for the case of football competition, we give the final objective value, the time of computation and the relative flow matrix. Tables 3.1 and 3.2 show the blocks extracted by BAS clustering algorithm. The tables also include the trophic levels of each actor and the average trophic level of each cluster.

- **Objective value:**
  - relative acyclic criterion C_r^0 without local search post-processing:

$$C_r^0 = 0.6543$$

  - relative acyclic criterion C_r^1 with a five pass local search post-processing:

$$C_r^1 = 0.8898$$

- **Relative flow matrix:**

$$F = \begin{bmatrix} 1 & 7 & 3 & 1 & 1 \\ 0 & 2 & 16 & 18 & 6 \\ 0 & 0 & 1 & 14 & 13 \\ 0 & 0 & 0 & 1 & 12 \\ 0 & 1 & 0 & 0 & 2 \end{bmatrix}$$

- **Computational time** with a five pass local search post-processing: 5 seconds.

Block 1		**Block 2**		**Block 3**	
Agents	Levels	Agents	Levels	Agents	Levels
Average	**1.65**	**Average**	**3.13**	**Average**	**4.03**
Input	0	Roots	1	Coral	3.43
2um Spherical Phytoplankt	1	Water Cilitaes	2.9	Other Cnidaridae	4.26
Synedococcus	1	Acartia Tonsa	2.91	Echinoderma	3.7
Oscillatoria	1	Oithona nana	2.91	Lobster	4.58
Small Diatoms (<20um)	1	Paracalanus	2.91	Predatory Crabs	4.49
Big Diatoms (>20um)	1	Other Copepoda	3.36	Callinectus sapidus	4.49
Dinoflagellates	1	Meroplankton	3.63	Stone Crab	4.31
Other Phytoplankton	1	Other Zooplankton	2.91	Sardines	4.18
Benthic Phytoplankton	1	Sponges	3	Anchovy	4.25
Thalassia	1	Bivalves	3	Bay Anchovy	3.95
Halodule	1	Detritivorous Gastropods	4.13	Toadfish	4.85
Syringodium	1	Epiphytic Gastropods	2	Halfbeaks	3.26
Drift Algae	1	Predatory Gastropods	5.12	Other Killifish	3.7
Epiphytes	1	Detritivorous Polychaetes	3.88	Goldspotted killifish	4.3
Free Bacteria	2.92	Predatory Polychaetes	4.44	Rainwater killifish	3.82
Water Flagellates	3.19	Suspension Feeding Polych	3.32	Silverside	3.95
Benthic Flagellates	3.77	Macrobenthos	4.13	Other Horsefish	3.97
Benthic Ciliates	4.1	Benthic Crustaceans	3.88	Gulf Pipefish	4.06
Meiofauna	4.35	Detritivorous Amphipods	3.88	Dwarf Seahorse	3.97
		Herbivorous Amphipods	2.59	Mojarra	4.32
		Isopods	2	Porgy	4.26
		Herbivorous Shrimp	2	Pinfish	3.82
		Predatory Shrimp	3.9	Mullet	3.86
		Pink Shrimp	3.43	Blennies	4.12
		Thor Floridanus	2	Code Goby	4.41
		Detritivorous Crabs	3.72	Clown Goby	4.41
		Omnivorous Crabs	3.79	Other Demersal Fishes	4.21
		Sailfin Molly	2	DOC	1.92
		Green Turtle	2		

Table 3.1: Blocks 1 to 3 returned by BAS clustering algorithm, 1 is at the bottom of the hierarchy.

Block 4		Block 5	
Agents	Levels	Agents	Levels
Average	**4.69**	**Average**	**5.17**
Rays	4.89	Sharks	5.12
Bonefish	4.81	Tarpon	5.23
Lizardfish	4.98	Grouper	5.06
Catfish	4.93	Mackerel	5.2
Eels	4.93	Barracuda	5.21
Brotalus	4.85	Loon	5.21
Needlefish	4.72	Greeb	5.05
Snook	4.64	Pelican	5.2
Jacks	4.71	Comorant	5.18
Pompano	4.83	Big Herons and Egrets	5.14
Other Snapper	4.69	Predatory Ducks	5.14
Gray Snapper	4.78	Raptors	5.66
Grunt	4.36	Crocodiles	5.39
Scianids	4.63	Dolphin	5.27
Spotted Seatrout	4.69	Water POC	5.04
Red Drum	4.77	Benthic POC	4.55
Spadefish	4.58	Output	5.22
Parrotfish	3.86	Respiration	5.19
Flatfish	4.58		
Filefishes	4.78		
Puffer	4.77		
Other Pelagic Fishes	4.74		
Small Herons and Egrets	4.85		
Ibis	4.8		
Roseate Spoonbill	4.91		
Herbivorous Ducks	4.19		
Omnivorous Ducks	4.28		
Gruiformes	4.91		
Small Shorebirds	4.93		
Gulls and Terns	4.91		
Kingfisher	4.86		
Loggerhead Turtle	4.79		
Hawksbill Turtle	4.71		
Manatee	3.9		

Table 3.2: Continuation of table 3.1: blocks 4 and 5 returned by BAS clustering algorithm, 5 being at the top of the hierarchy.

Concerning the performances of the algorithm, we observe that

- edges in the upper-triangular part of F represent 99% of all edges,
- edges in the strict upper triangular part of F represent 92% of all edges.

The local search step slightly improves the result of BAS clustering algorithm but the result of BAS clustering algorithm alone is already satisfactory.

We can also observe that our classification is globally consistent with trophic levels. Firstly, average of trophic levels within a block increases from one block to the next. Secondly, one can compute an estimation of the error produced by BAS clustering algorithm with respect to trophic levels. Let us denote by V the set of 128 agents, $f : V \to \{1, ..., 5\}$ the function assigning each agent to a block and $l : V \to \mathbb{R}^+$ such that $l(x)$ is the trophic level of agent x. We compute a particular type of error that we refer to as the *inversion error*

$$\frac{2}{n(n-1)}|\{(i,j) \in V \times V : (f(i) - f(j))(l(i) - l(j)) < 0\}|.$$

We can interpret the inversion error in the following way. For any pair $(i, j) \in V \times V$, we have $(f(i) - f(j))(l(i) - l(j)) < 0$ if there is a contradiction between the trophic levels and the block memberships of i and j (namely $l(i) < l(j)$ but $f(i) > f(j)$ or $l(i) > l(j)$ but $f(i) < f(j)$). This can be interpreted as a misclassification of either i or j. We count the number of such pairs and we divide it by the total number of pairs $\frac{n(n-1)}{2}$. In our case, we obtain an inversion error of 7% which means that the block membership is almost fully consistent with the trophic levels. We also ran the SVD clustering algorithm based on blockmodelling approach of section 3.2 on the same trophic level and we obtain an inversion error of 25%. This confirms the fact that our algorithm is more efficient for the detection of block-acyclic networks.
Finally, we observe that we roughly recover the classical trophic levels we expected:

- **block 1:** mostly autotrophs, bacteria and input of carbone,
- **block 2:** herbivorous and small omnivorous (such as gastropods and shrimps),
- **block 3:** larger omnivorous and carnivorous (Killifish, crabs, lobsters...),
- **block 4:** carnivorous (such as kingfishers, catfish, snappers...),
- **block 5:** top-level predators (such as sharks, crocodiles or cormorants) and output of carbon.

It may seem unrealistic to classify cormorants, pelicans and loons in the category of top-level predators. However, in such marine ecosystem, it is likely that these hunting birds have no predators which undoubtedly makes them belong to the highest level of the pyramid. In order to provide a visual intuition that the algorithm succeeds in detecting the five classical trophic levels, we display the whole graph of the trophic network at figure 3.15. Colours of nodes denote the five clusters returned by our algorithm. An edge (a, b) is coloured in the colour of a which means for instance that all edges coloured in blue connect nodes coloured in blue to all their predators. The graph clearly highlights the block-acyclic structure of the trophic network.

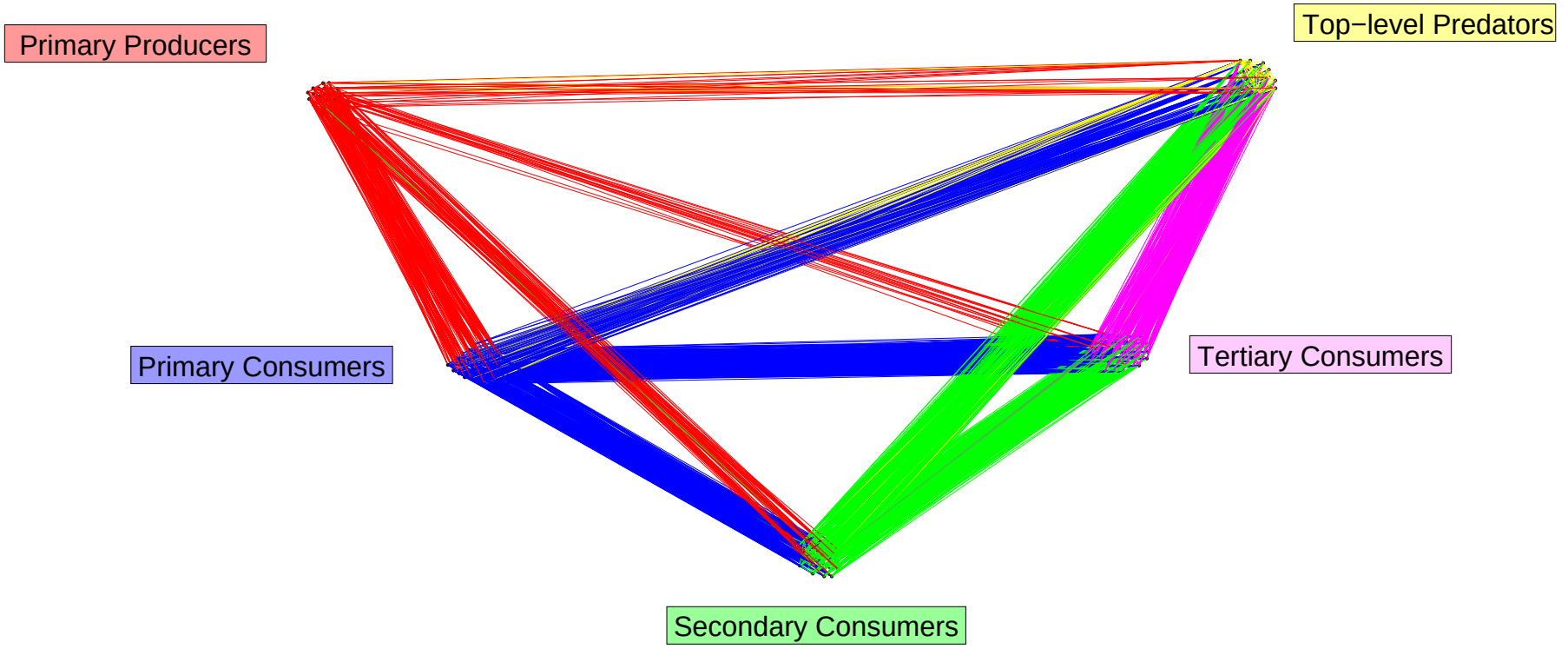


Figure 3.15: Graph of the trophic network. Colours of nodes denote the five clusters returned by BAS clustering algorithm. An edge (a, b) is coloured in the colour of a which means for instance that all edges coloured in blue connect nodes coloured in blue to all their predators.

2.3 Network of autonomous systems

Every month since 1998, the Center for Applied Internet Data Analysis (CAIDA) publishes the Autonomous Relationships Dataset [49]. An autonomous system can be viewed as a set of computers with IP address that are connected to the internet according to a common routing protocol (this protocol defines for instance the routes that should be chosen in priority on the internet, the routes that should be avoided, etc.). Autonomous systems often correspond to sets of computers that get their internet connection from a common Internet Service Provider or ISP (such as Belgacom in Belgium or Deutsche Telekom in Germany).

2.3.1 The AS graph

In order to collectively manage the traffic of information on the internet, ISPs maintain relationships usually in the form of business agreements: for example, Belgacom may have a deal with Deutsche Telekom to use the German network (that have connections with the whole world) for the transfer of information between Belgium and the United States or Asia. These relationships are often maintained secret. However these agreements result in traffic flows between ISPs that can be measured[3]. These traffic flows provide the basis for CAIDA's datasets.

In [54], Gao proposes a model for the large network of ISPs and relationships in the forms of cash flows as shown on figure 3.16. In this network, A, B, C, D and F represent ISPs. Connections between ISPs represent transfers of cash resulting from business agreements. Two types of connections appear:

1. peer-to-peer connections (connection B-C): these are undirected connections between two networks of equal strategical importance that do not pay for the traffic between each other, typically B uses C's network but C uses B's network in the same proportion so it is a benefit for both B and C not to pay or to get paid for these connections. Another way to interpret this type of connection is to view B and C as ISPs of equal rank in terms of their number of customers, meaning that B and its customers have an equal need for C's network than C and its customer have a need for B's network (this could for instance be the case for ISPs like Deutsche Telekom in Germany and Orange in France).

2. customer-to-provider connections (connections B-A, C-A, D-B, E-B and F-C): these are directed connections from a lower ranked ISP (customer) to an ISP of greater importance, typically all traffic leaving D must transit in B but the opposite is not true, hence it is likely that D must pay B for the transit of data in B's network. This could be the case for companies like Belgacom and Deutsche Telekom in Germany: a large part of the traffic leaving Belgacom's network must first transit through Deutsche Telekom before being sent elsewhere in the world.

Clearly the network depicted by figure 3.16 is a symmetric-acyclic network where the three blocks are $\{A\}$, $\{B, C\}$ and $\{D, E, F\}$. Our goal is now to test BAS clustering algorithm on a dataset provided in [50]. We ran BAS clustering algorithm on the data provided by CAIDA in [49] for the 1st of October 2013. The database involves 45427 ISPs (nodes) and 230194 connections (edges). The connexions are labelled as peer-to-peer connexions (undirected) or customer-to-provider connexions (directed) inferred from the traffic measured in October 2013. In [49], nodes are denoted by their *AS number*, the correspondence between AS numbers and official names of ISPs can be found in [50]. We use this dataset to create an unweighted directed graph $G = (V, E)$ where V is the set of ISPs and an edge $(A, B) \in E$ can either be an undirected

[3]For more details about how relations are inferred by CAIDA, the reader is referred to [49]

edge if there exists a peer-to-peer relation between A and B or directed if the relation is of the type customer-to-provider. We hope that BAS clustering algorithm can help us understand the topology of CAIDA's graph by classifying ISPs according to their strategical importance on a global scale.

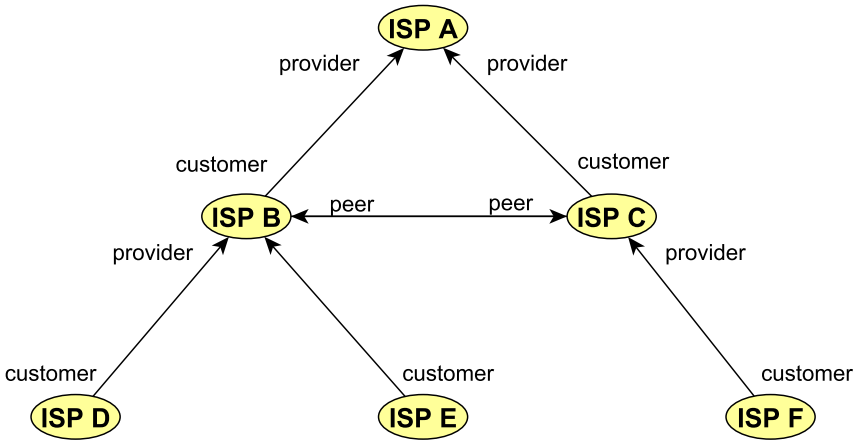


Figure 3.16: Abstract representation of the network formed by ISPs, edges represent cash flows. Similar figures can be found on CAIDA's official website [49].

In the following sections, we first present the output of BAS clustering algorithm as we did for the two previous applications (relative flow matrix, objective value...). Then, we check if the output of the algorithm is consistent with two well known rankings of ISPs: the ranking based on the *transit degree* and the *classification into tiers*.

2.3.2 Results of BAS clustering algorithm

We choose the following values for the parameters of BAS clustering algorithm.

- In the definition of the generalized Laplacian (according to 19), we choose $\epsilon = 0$ as tests showed that result does not vary significantly when choosing a nonzero value for ϵ (such as for instance $\epsilon = 10^{-3}$).
- We choose a number of blocks $k = 3$. This choice is rather arbitrary. However, this assumption is confirmed by the graph of complex eigenvalues of the Laplacian (figure 3.17) where three eigenvalues are clearly away from $(1, 0)$ in the complex plane. Moreover, the official ranking of ISPs into tiers partitions the ISPs into three groups (called *Tier 1*, *Tier 2* and *Tier 3*, see section 2.3.4 for more details) which supports the choice of $k = 3$ blocks for BAS clustering algorithm.
- As mentioned earlier, we seek a block-symmetric-acyclic structure rather than a block-acyclic structure because of the presence of undirected peer-to-peer connexions.
- We do a one-pass local search post-processing as a two-pass local search post-processing does not improve significantly the objective value.

We first display the thirty eigenvalues of the Laplacian located furthest from $(1, 0)$ in the complex plane (figure 3.17).

In the following paragraph, we briefly present the results of BAS clustering algorithm. In addition to the final value of the objective function and the time of computation, two information are given regarding the result of BAS clustering algorithm, given the resulting block membership function $f : V \to \{1, ..., K\}$:

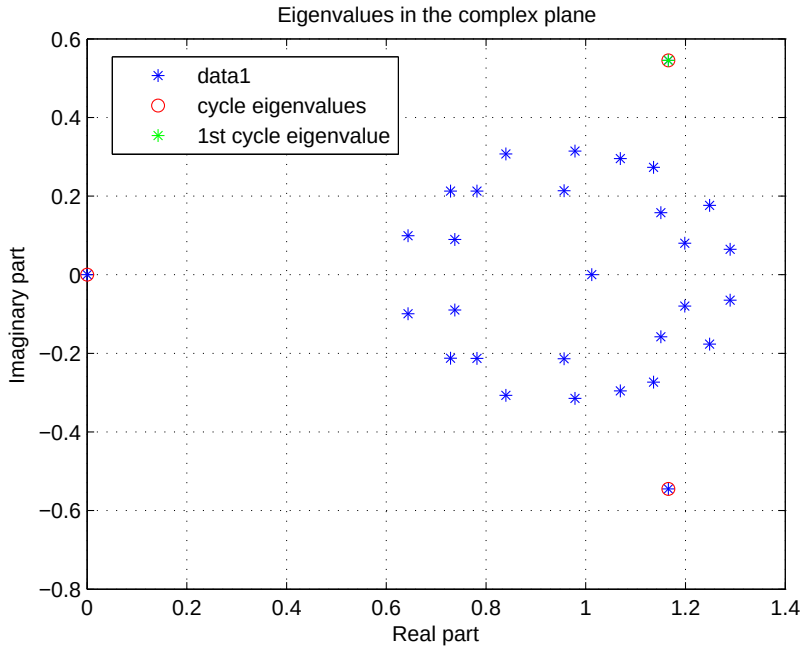


Figure 3.17: 30 eigenvalues of the Laplacian located furthest from $(1, 0)$ in the complex plane. The 3 proposed cycle eigenvalues are circled in red, the first cycle eigenvalue is coloured in green.

1. the number of vertices in each block (as a percentage of the total number of nodes in the graph):

$$s_i = \frac{100}{n}|\{u \in V : f(u) = i\}|$$

where $n = |V|$.

2. the relative flow matrix

$$F_{ij} = Flow(i, j) = \frac{100}{m} \sum_{u,v:f(u)=i,f(v)=j} W_{uv}$$

where m is the total number of edges.

As previously, we expect F to be upper triangular in the ideal case. Blocks of ISPs are referred to as Block 1, Block 2 and Block 3 where Block 1 is at the top of the hierarchy than comes block 2 and finally block 3, at the bottom. The results of BAS clustering algorithm are the following.

- **Objective value:**
  - relative symmetric-acyclic criterion C_r^0 without local search post-processing:

$$C_r^0 = 0.3896$$

  - relative symmetric-acyclic criterion C_r^1 with a one pass local search post-processing:

$$C_r^1 = 0.8542$$

- **Vector of sizes of each block**

Block	**Relative size (percentage)**
3	40
2	46
1	13

- **Relative flow matrix:**

$$F = \begin{bmatrix} 1 & 7 & 8 \\ 0.5 & 54 & 19 \\ 0.08 & 4 & 5.5 \end{bmatrix}$$

- **Computational time** with a one pass local search post-processing: 90 seconds.

The results given above confirm the assumption that the graph formed by AS relationships is a symmetric-acyclic network as shown by the value achieved by the objective function and the upper-triangular shape of matrix F. We may highlight three facts out of the results given above.

1. regarding the relative sizes of ISPs, we expect that the sizes of the blocks would decrease when moving from the bottom to the top of the hierarchy, namely block 3 (bottom) should contain the majority of ISPs which are small and have a very limited influence whereas block 1 (top) should consist of a small number of powerful ISPs. In our case, block 1 is significantly smaller than other blocks but blocks 2 and 3 have approximately the same size which suggests that BAS clustering algorithm tends to erroneously classify into block 2 nodes that should actually be in block 3.

2. We observe that the flow inside block 2 and inside block 1 is large. We could expect this result. Indeed, there should be much more peer-to-peer connexions inside block 1 and 2 than inside block 3 as these connexions are essential to ensure that the overall network is connected and that any two ISPs in block 3 can communicate by transiting via block 2 and block 1.

3. We observe that flows from block 3 to block 2, from block 2 to block 1 and from block 3 to block 1 are essentially unidirectional which confirms the fact that ISPs in block 3 are customers of ISPs in block 2 and block 1, ISPs in block 2 are customer of ISPs in block 1 and providers of ISPs in block 3 and ISPs in block 1 are providers of blocks 2 and 3.

2.3.3 Comparison with the AS transit degree ranking

The AS transit degree provided by CAIDA is one of the measurements of the importance of an ISP. The transit degree of an ISP x can roughly be viewed as the number of ISPs for which x may provide a transit of flow through customer-to-provider connexions. In figure 3.16 for instance, B has a transit degree of 2 and A has a transit degree of 5. Transit degrees of all 45427 ISPs of our graph were computed by CAIDA and can be found in [51]. We now test if our classification into three blocks is consistent with AS transit degrees.

Firstly, we compute the average transit degree within each block (table 3.3).

Block	Average transit degree
3	0.3114
2	5.1602
1	16.4924

Table 3.3

As expected, the average transit degree is the lowest for block 3 and the highest for block 1.

Secondly, we compute the number of inversions when comparing the output of BAS clustering algorithm and AS transit degrees. We compute the *inversion error* as defined in preceding sections where the trophic level of each agent of the trophic network was compared to its block membership.

The total inversion error is 6% which confirms the consistency of the output of BAS clustering algorithm with the AS transit degrees.

2.3.4 Comparison with the classification into tiers

The most popular classification of ISPs is the classification into *Tier 1*, *Tier 2* and *Tier 3* as follows.

- **Tier 1:** ISPs that can reach any network around the world without purchasing any IP transit, these ISPs have peer-to-peer connexions with other ISPs in Tier 1 and are providers of some ISPs in Tiers 2 and 3. In Europe Tier 1 companies are typically Deutsche Telekom, Orange (France), Telecom Italia, Telefonica (Spain) and TeliaSonera (Finland-Sweden).
- **Tier 2:** ISPs that have some peer-to-peer connexions with other ISPs in Tier 2 but they must purchase IP transit for traffic in some part of the Internet by having customer-to-provider connexions with ISPs in Tier 1. ISPs in Tier 2 are often customers of some ISPs in Tier 1 and providers of some ISPs in Tier 3. Examples of networks in Tier 2 networks in Europe are Belgacom or British Telecom.
- **Tier 3:** ISPs that must purchase IP transit from other ISPs for there participation in the internet. These ISPs have no peer-to-peer connexion with other ISPs, they are customers of some ISPs in Tier 2 and Tier 1.

The problem encountered with this classification scheme is that it is quite vague. As pointed out in [52], a consistent classification into Tiers takes many elements into account, including

information that cannot be inferred from the network of ISPs (such as the ownership of international fiberoptic transport, the presence on several continents...). For this reason, there is no unique classification of ISPs into tiers and the ones that are available often only include ISPs in Tier 1 and a few ISPs in Tier 2 but rarely a complete version of Tier 2 and Tier 3.

For this reason, we only compare the output of BAS clustering algorithm with the most popular version of Tier 1 (available in [53]). These ISPs are displayed in table 3.4 along with the block in which they were classified by BAS clustering algorithm.

ISP	Headquarters	Block membership
AT& T	United States	1
CenturyLink	United States	1
Cogent	United States	1
Deutsche Telekom	Germany	1
GTT	United States - Italy	1
Level 3 Communications	United States	1
NTT Communications	Japan	1
OpenTransit (Orange)	France	1
Sprint	United States	1
Tata Communications	India	1
Seabone (Telecom Italia Sparkle)	Italy	1
Telefonica	Spain	1
TeliaSonera International Carrier	Sweden - Finland	1
Verizon Enterprise Solutions	United States	1
XO Communications	United States	1
Zayo Group	United States	1

Table 3.4: Networks of Tier 1 and block membership computed with BAS clustering algorithm.

We observe that our algorithm is fully consistent with this classification of ISPs in Tier 1. However, we cannot say that blocks returned by BAS clustering algorithm are equivalent to Tiers. Indeed, block 1 contains approximately four thousand ISPs which is far more than the classical definition of Tier 1 which rarely contains more than fifty ISPs. To illustrate this observation, we may look at the classification of two ISPs that are commonly classified into Tier 2: Belgacom (Belgium) and British Telecom (United Kingdom). According to BAS clustering algorithm, Belgacom is classified into block 2 and British Telecom is classified into block 1. British Telecom is classified into Tier 2 because its presence outside of Europe and its international influence is more limited than the one of ISPs such as Deutsche Telekom. However, when focusing on the topology of the graph of ISPs connections, we can expect that British Telecom has an important role, being the major ISPs of the United Kingdom. For this reason, BAS clustering algorithm classifies British Telecom into block 1.

2.3.5 Discussion and further developments

As we showed, the output of BAS clustering algorithm confirms the intuition that the graph of Autonomous Systems has a block-symmetric-acyclic structure, it is also consistent with the classification of ISPs according to transit degrees. Hence, we expect that BAS clustering algorithm classifies nodes according to their importance on the internet as providers of other ISPs.

However, several difficulties arise when dealing with CAIDA's dataset.

Firstly, the graph is highly sparse: with 45427 nodes and 230194 edges, the average in- and out-degree is about 5 but more than half of the nodes have out-degree 1. The sparsity of the graph makes it hard to provide a reliable classification of ISPs. CAIDA publishes the list of connexions between ISPs every month. Hence, one way to avoid sparsity is to use the data published by CAIDA for several months (and not only the month of October 2013 as we did) hoping that it will add enough edges to make the result more reliable.

Secondly, the number of Automous Systems taken into account is very large. The graph we analysed contains 45427 out of the 49874 Autonomous Systems identified by CAIDA worldwide. Perhaps, it would be interesting to focus on a more limited number of AS's and apply CAIDA's algorithm to the corresponding subgraph. To that end, one must find a clever method to choose the Autonomous Systems that should be extracted.

Thirdly, after having partitioned the graph into three blocks using BAS clustering algorithm, in order to better extract world-leading ISPs, we could try to re-cluster the subgraph corresponding to block 1 with BAS clustering algorithm. This method follows the principle of hierarchical clustering and might be able to extract the small group of world-leading ISPs corresponding to Tier 1.

Conclusion and perspectives

This paper was dedicated to the description of spectral clustering algorithms for directed graphs. In the first chapter we presented some spectral clustering algorithms found in the literature, the type of clustering problem they aim to solve and their failure to detect some types of pattern-based clusters, in particular graphs where the connections between clusters form a pattern of cycle or an acyclic pattern. In chapters 2 and 3, we presented two algorithms (mainly MCE and BAS clustering algorithms, as SCE clustering algorithm is outperformed by MCE clustering algorithm). These algorithms overcome some of the difficulties encountered with the existing spectral approaches. In this conclusion, we first provide an overview of our contributions, then we give some further perspectives related to our work that could be treated in future papers.

Summary of contributions

We classify our contributions into two categories: major contributions that are directly related to the design and application of our algorithms and minor contributions.

1. **Major contributions:**

    - Definition of the concept of block-cycle in graph theoretical terms.
    - Proof of np-completeness of the optimization problem related to the detection of blocks in a slightly perturbed block-cycle.
    - Design of two spectral clustering algorithms (MCE and SCE clustering algorithms) for the detection of blocks in a slightly perturbed block-cycle.
    - Proof of consistency of SCE clustering algorithm for the detection of two blocks in an undirected graph.
    - Tests on synthetic data assessing the performances of MCE and SCE clustering algorithms in different cases: random perturbation of the adjacency matrix, growing number of nodes, presence of unbalanced blocks, presence of unbalanced flows, etc.
    - Formal definition of block-acyclic and block-symmetric-acyclic network based on Homans's conjecture concerning social interactions.
    - Design of a spectral clustering algorithm (BAS clustering algorithm) for the detection of blocks in slightly perturbed block-acyclic and block-symmetric-acyclic networks.
    - Detection of block-acyclic and block-symmetric-acyclic networks in real-world data applying BAS clustering algorithm to a network based on football competition, a trophic network and a network of Autonomous Systems. In each case, the output of BAS clustering algorithm is shown to be consistent with official rankings that are available. This proves that Homan's conjecture is verified in networks that are not directly related to social science.

2. **Minor contributions:**

   - Brief survey about clustering algorithms for directed graphs and formal description of three existing spectral clustering algorithms for directed graphs.
   - Formulation and proof (adapted from [28]) of properties giving the relation between the spectrum of the Laplacian of a directed graph and the existence of isolated components. Based on these properties, we provide the intuition for a spectral algorithm for the detection of density-based clusters in directed graphs.
   - Dynamic programming algorithm solving k-means problem exactly in the case of points in $\mathbb{C}$ clustered according to their phase.
   - Design of a local search algorithm improving the output of MCE or SCE clustering algorithms and proof of a guarantee on the quality of the result obtained.
   - Design of a local search algorithm improving the output of BAS clustering algorithm.

We may as well mention that MCE, SCE and BAS clustering algorithms are efficient in terms of time of computation. In particular, a clever implementation of MCE clustering algorithm in `Matlab` allows to process a graph of a hundred thousand of nodes with average in and out-degree of 100 in less than a minute. This is firstly due to the efficiency of methods both for the computation of eigenvalues (Arnoldi's algorithm) and for k-means step (Lloyd's algorithm with clever initialization). It is secondly due the high sparsity of graphs treated in terms of number of edges. Sparsity seems to be a reasonable assumption as most of the networks encountered in practice (for instance, social networks, trophic networks or networks of ISPs) involve nodes with low degrees compared to the total number n of nodes, which yields a number of edges much smaller than n^2 and hence, sparse adjacency matrix and sparse Laplacian. We may take advantage of this sparsity to speed up the computation of eigenvalues and eigenvectors of the Laplacian.

Further perspectives

Further works on the theory and algorithms developed in this report could include the following points.

Detection of the number of blocks

We defined the number k of blocks as a parameter of MCE, SCE and BAS clustering algorithms. When dealing with real-world data, we choose an arbitrary value for k that seems reasonable for the network considered. It would be interesting to design an automatic method to determine what number of blocks should be preferred. We could for instance choose the value of k that yields the largest value of the objective function (cyclic, acyclic or symmetric-acyclic criterion depending on the type of network considered). Another idea would be to automatically detect how many eigenvalues are located significantly further from $(1, 0)$ in the complex plane than other eigenvalues of the Laplacian.

Parallelization

Another area of research would be to analyse how our methods could be parallelized in order to handle large graphs containing more than a million nodes (such as some social networks

for instance). Concerning the step of the algorithm where eigenvalues of the Laplacian are computed, the operations that are performed are essentially matrix-vector products which could be parallelized using MapReduce algorithm.

Other domains of application for BAS clustering algorithm

We used BAS clustering algorithm to detect the presence of block-acyclic and block-symmetric-acyclic networks in a network based on football competition, a trophic network and a network of Autonomous Systems. However, Homans's conjecture, namely the existence of a block-symmetric-acyclic structure, was initially stated for the case of social groups. It would thus be interesting to check if block-symmetric-acyclic networks can also be detected in large online social networks that appeared during the past ten years. For instance, the directed graph associated to Twitter's network might have a block-symmetric-acyclic structure where popular profiles with many followers lie at the top of the hierarchy. Hence, BAS clustering algorithm could be used in a systematic way to verify whether Homans's conjecture is a general law applicable to various types of of social networks.

Detection of more complex structures

Our spectral algorithms detect clusters in graphs with a cyclic or an acyclic pattern of connections between clusters. These algorithms rely on the particular shape of the spectrum of the Laplacian of a block-cycle. A similar approach could be followed to detect more complex structures including combinations of block-cycles, block-trees (directed trees where nodes are replaced by groups of nodes), etc. As an example, let us consider the case of a graph consisting of the superposition of the edges of two block-cycles, one consisting of ten blocks and the other consisting of three blocks. A graph is then formed by including all edges of the first and the second block-cycle. The question is then to verify whether it is possible to detect the two original block-cycles. Figure 3.18 displays the eigenvalues of the Laplacian of the graph. We observe that there are twelve eigenvalues located significantly further from $(1, 0)$ than other eigenvalues. If we apply MCE clustering algorithm using the eigenvectors associated to the three eigenvalues located furthest from $(1, 0)$, we may recover the block-cycle of three blocks. In contrast, if we apply MCE clustering algorithm with the eigenvectors associated to the other nine eigenvalue plus the eigenvalue zero, we recover the block-cycle of ten blocks. This example shows that our method could be generalized for the detection of more complex structures. In all cases, we would expect that some particular eigenvalues of the Laplacian give a specific information about the structure of a graph when nodes are partitioned into blocks (the presence of a block-cycle for instance) and clustering the components of the corresponding eigenvectors yields the block membership of nodes.

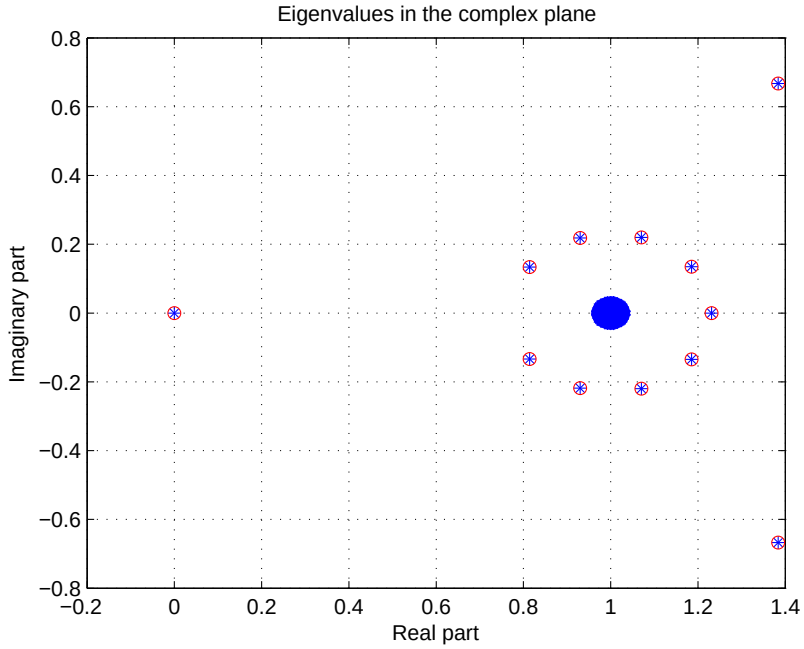


Figure 3.18: Eigenvalues of the Laplacian of a graph of 1000 nodes obtained by combining the edges of two block-cycles, one of 10 blocks and one of 3 blocks. The twelve cycle eigenvalues are circled in red.

Bibliography

[1] A. L. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (5439) 509-512, 1999.

[2] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, in: SIGCOMM '99: Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communication, pp. 251-262, 1999.

[3] S. Milgram, The small-world problem, Psychology Today 1 (1), 61-67, 1967.

[4] R. Albert, H. Jeong, A.-L. Barabasi, The diameter of the world wide web, Nature 401, 130-131, 1999.

[5] M. Girvan, M. E. Newman, Community structure in social and biological networks., Proc Natl Acad Sci 99 (12), 7821-7826, 2002.

[6] A. K. Jain, M. N. Murty, P. J. Flynn, Data clustering: a review, ACM Comput. Surv. 31 (3) 264-323, 1999.

[7] W. E. Donath, A. J. Hoffman, Lower bounds for the partitioning of graphs. IBM J. Res. Develop., 17, 420 – 425, 1973.

[8] M. Henzinger, "Algorithmic Challenges in Web Search Engines." Internet Mathematics 1:1, 115—126, 2003.

[9] M. Fiedler, Algebraic connectivity of graphs. Czechoslovak Math. J., 23, 298 – 305, 1973.

[10] J. Cheeger, A lower bound for the smallest eigenvalue of the Laplacian, Problems in Analysis (R. C. Gunning, ed.), Princeton Univ. Press, 195-199, 1970.

[11] D. Spielman, S. Teng, Spectral partitioning works: planar graphs and finite element meshes. In 37th Annual Symposium on Foundations of Computer Science (Burlington, VT, 1996) (pp. 96 – 105). Los Alamitos, CA: IEEE Comput. Soc. Press. (See also extended technical report.)

[12] U. von Luxburg, M. Belkin, O. Bousquet, Consistency of spectral clustering, in The Annals of Statistics, Vol. 36, No. 2, 555–586, Institute of Mathematical Statistics, 2008.

[13] U. von Luxburg, A tutorial on spectral clustering, Statistics and Computing 17(4): 395–416, 2007.

[14] S. Fortunato, Community Detection in graphs, Physics Reports 486 (3-5), 75-174, 2010.

[15] F. Malliaros, M. Vazirgiannis, Clustering and Community, Detection in Directed Networks: A Survey. Physics Reports, 2013.

[16] V. Satuluri, S. Parthasarathy, Symmetrizations for clustering directed graphs, in: EDBT '11: Proceedings of the 14th International Conference on Extending Database Technology, pp. 343-354, 2011.

[17] E. A. Leicht, M. E. J. Newman, Community structure in directed networks, Phys. Rev. Lett. 100 - 118703, 2008.

[18] Y. Kim, S.-W. Son, H. Jeong, Finding communities in directed networks, Phys. Rev. E 81 - 016103, 2010

[19] P. W. Holland, K. Laskey, S. Leinhardt, Stochastic blockmodels: First steps. Social Networks, 5:109-137, 1983.

[20] Y. J. Wang, G. Y. Wong, Stochastic Blockmodels for Directed Graphs. Journal of the American Statistical Association, 82:8-19, 1987.

[21] D. L. Sussman, M. Tang, D. E. Fishkind, C. E. Priebe, A consistent adjacency spectral embedding for stochastic blockmodel graphs. J. Amer. Statist. Assoc., 107(499):1119–1128, 2012.

[22] C. Aicher, A. Z. Jacobs, A. Clauset, Adapting the stochastic block model to edge-weighted networks. ICML Workshop on Structured Learning (SLG 2013), 2013.

[23] F. R. K. Chung, Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92). Cbms Regional Conference Series in Mathematics, 2006.

[24] F. R. K. Chung, Laplacians and the cheeger inequality for directed graphs. Annals of Combinatorics, 9:1–19, 2005.

[25] D. Gleich, Hierarchical directed spectral graph partitioning, Tech. rep., Stanford University, 2006.

[26] W. Pentney, M. Meila, Spectral clustering of biological sequence data, in: AAAI '05: Proceedings of the 20th national conference on Artificial intelligence - Volume 2, pp. 845-850, 2005.

[27] A. Capocci, V. D. P. Servedio, G. Caldarelli, F. Colaiori, Detecting communities in large networks, Physica A: Statistical and Theoretical Physics 352 (2-4) 669-676, 2005.

[28] F. Bauer, Normalized graph Laplacians for directed graphs. Linear Algebra Appl. 436, 4193–4222, 2012.

[29] F. Bauer, F. Atay, J. Jost, Synchronized chaos in networks of simple units, Europhysics Letters, 2(20002), 2010.

[30] J. Aljadeff, D. Refrew, M. Stern, Eigenvalues of block structured asymmetric random matrices, arXiv:1411.2688v1 [math.PR] 11 Nov 2014.

[31] M. Mahajan, P. Nimbhorkar, K. Varadarajan, The planar k-means problem is NP-hard. WALCOM: Algorithms and Computation, pages 274–285, 2009.

[32] H. Wang, M. Song, optimal k-means clustering in one dimension by dynamic programming The R Journal, 2011, cs.kent.edu.

[33] J. Leskovec, A. Rajaraman, J. D. Ullman, Mining of massive datasets, Cambridge University Press, Cambridge, UK, second edition, ISBN 1-107-07723-0 (hardcover), pp 254-256, 2014.

[34] Y. Saad, Numerical methods for large eigenvalue problems, Algorithms and Architectures for Advanced Scientific Computing, Manchester University Press, Manchester, U.K., 1992.

[35] P. Van Dooren, P.-A. Absil, Analyse numérique, syllabus of the course INMA1170, Université Catholique dde Louvain, academical year 2014-2015.

[36] L. Huang, D. Yan, M. I. Jordan, N. Taft, "Spectral clustering with perturbed data," in Proc. Neural Information Process. Syst. (NIPS), 2008.

[37] G. C. Homans, The human group. New York: Harcourt, Brace, 1950.

[38] J.A. Davis, S. Leinhardt, "The Structure of Positive Interpersonal Relations in Small Groups," in Sociological Theories in Progress, Volume 2, Ed., J. Berger, Boston: Houghton Mifflin, 1972.

[39] P. Doreian, V. Batagelj, A. Ferligoj, Symmetric-acyclic decompositions of networks. J. classif., 2000.

[40] M. Gupte, P. Shankar, J. Li, Muthukrishnan, S., Iftode, L.: Finding hierarchy in directed online social networks. In Proceedings of the 20th International Conference on World Wide Web. pp. 557-566, 2011.

[41] D. Williamson, ExcelMaster, 2009,
http://www.excelmaster.co.uk/support_files/excel_files/fixtures_league_table.xlsx,

[42] Barclays Premier League official website, 2015,
http://www.premierleague.com/en-gb/matchday/league-table.html

[43] R. E. Ulanowicz, C. Bondavalli, M. S. Egnotovich, Network Analysis of Trophic Dynamics in South Florida Ecosystem, FY 97: The Florida Bay Ecosystem, University of Maryland System, Chesapeake Biological Laboratory, Solomons, MD 20688-0038, May 29, 1998, data available at http://www.cbl.umces.edu/ atlss/FBay001.html

[44] S. D. Butz, Science of Earth Systems, ed. Thomson-Delmare Learning, New York, 2004.

[45] E. Charles, Animal Ecology, The Macmillan company, New York, 1927.

[46] R. L. Lindeman, "The trophic-dynamic aspect of ecology" (PDF). Ecology 23 (4): 399–417, 1942.

[47] E. P. Odum, G. W. Barrett, Fundamentals of Ecology (5th ed.). Brooks/Cole, a part of Cengage Learning. ISBN 0-534-42066-4, 2005.

[48] D. Pauly, M. L. Palomares, "Fishing down marine food webs: it is far more pervasive than we thought" (PDF), Bulletin of Marine Science 76 (2): 197–211, 2005

[49] Center for Applied Internet Data Analysis, CAIDA 2013 Annual Report, http://www.caida.org/data/as-relationships/

[50] Center for Applied Internet Data Analysis, CAIDA 2013 as relationships dataset, http://data.caida.org/datasets/as-relationships/serial-1/

[51] Center for Applied Internet Data Analysis, As Rank: AS Ranking, http://as-rank.caida.org/

[52] M. Winther, White Paper tier 1 ISPs, IDC Information and Data, 2006, paper available at https://www.us.ntt.net/downloads/papers/IDC_Tier1_ISPs.pdf

[53] Anonymous, Tier 1 network, Last update on 3rd of May 2015, http://en.wikipedia.org/wiki/Tier_1_network

[54] L. Gao, On Inferring Autonomous System Relationships in the Internet, IEEE/ACM Transactions on Networking, December 2001.

[55] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification. John Wiley & Sons, 2000.

Appendix

1 Arnoldi's algorithm

In this section, we briefly describe Arnoldi's algorithm for finding eigenvectors and eigenvalues of a matrix. For more details about the theoretical background and the convergence of this method, the reader is referred to [34] and [35]. We consider a matrix $A \in \mathbb{C}^{n\times n}$ and a vector $u \in \mathbb{C}^n$. Our goal is to find the eigenvectors and eigenvalues of an orthogonal projection of A onto the Krylov space $K_m(A,u)$. By definition, $\{u, ..., A^{m-1}u\}$ forms a basis of $K_m(A,u)$. However, this basis tends to be ill-conditioned when m grows (see [35] p.103 for more details). Arnoldi's algorithm tackles this difficulty by computing an orthonormal basis of $K_m(A,u)$. Orthonormalization is done by a modified Gram-Schmidt process.

Algorithm 1.1: Arnoldi's algorithm

Input: matrix $A \in \mathbb{C}^{n\times n}$, $m \in \{1, ..., n\}$;
Initialization: vector $u_1 \in \mathbb{C}^1$ such that $\|u\|_2 = 1$;
for $j = 1, 2, ..., m$ **do**
  $w \leftarrow Au_j$;
  for $i = 1, 2, ..., j$ **do**
    $h_{ij} \leftarrow u_i^* w$;
    $w \leftarrow w - h_{ij}u_i$;
  end
  $h_{j+1,j} \leftarrow \|w\|_2$;
  $u_{j+1} \leftarrow w/h_{j+1,j}$;
end
Output: matrices h and U;

The outputs of the algorithm are a matrix U whose columns form an orthonormal matrix of $K_m(A,u_1)$ and H which verifies $H = U^*AU$ (proof in [35] p. 104). Hence, H is a projection of A onto $K_m(A,u_1)$. Moreover, as elements of H are zero under its first subdiagonal, H is called a Hessenberg matrix. For small m, computing the eigenvalues of H is easy and one can show that these eigenvalues are good approximations of the eigenvalues of A located at the outskirts of the spectrum. Once we computed an eigenvalue λ of H for eigenvector $v \in \mathbb{C}^m$, the estimated corresponding eigenvector of A is Uv.

2 Proof of consistency theorems

In this section we prove the theorems of section 3 of chapter 2. Both proofs are slight variations of the proofs of theorems described in [36]. For some heuristic about assumption A3 described in section 3 of chapter 2, the reader is referred to the appendix of [36] which depicts cases where the assumptions is verified in the case of spectral bi-clustering, a similar reasoning leads

to similar conclusions in the case of SCE clustering algorithm.

Proof of theorem 7 of chapter 2 : G and $\tilde{G}$ are respectively the unpertubed and the perturbed undirected block-cycle of two blocks. Δ and $\tilde{\Delta}$ are the associated Laplacians. $\mathbf{v}_c$ and $\tilde{\mathbf{v}}_c$ are the first cycle eigenvectors of G and $\tilde{G}$ respectively. We denote by δ the norm of the difference $\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2$. a and b are the clusters obtained by thresholding the components of $\mathbf{v}_c$, the sizes of which are k_a and k_b. a' and b' are the clusters obtained by thresholding the components of $\tilde{\mathbf{v}}_c$. We denote by $a \to a'$ the set of components that are clustered in a when considering $\mathbf{v}_c$ and clustered in a' when considering $\tilde{\mathbf{v}}_c$, similarly for $a \to b'$, $b \to a'$ and $b \to b'$. We denote by m the total number of components in $a \to b'$ and $b \to a'$, namely the total number of mis-clustered nodes. m_- is the number of mis-clustered components in $a \to b'$ and m_+ is the number of mis-clustered components in $b \to a'$.

We let A_1,...,A_{k1} and B_1,...,B_{k2} denote the zero mean fluctuations around a and b respectively and we let U_1,...,$U_{k_1-m_-}$ and $V_1, ..., V_{k_2-m_+}$ denote the zero mean fluctuations around a' and b', respectively. We also denote the jumps from a to b' by Y_1,..., $Y_{m_-} \geq 0$ and from b to a' by Z_1,..., $Z_{m_+} \geq 0$. Hence, $Z_j = a' + U_j - b - B_j$. Figure 3.19 illustrates these notations. To compute the maximum number of mis-clustering m is obtained by solving the optimization problem

$$
\begin{array}{l}
\max m = m_+ + m_- \\
\text{subject to} \\
\sum_{i=1}^{k_1}(a + A_i)^2 + \sum_{j=1}^{k_1}(b + B_j)^2 = 1 \\
\sum_{i=1}^{k_1-m_-}(a' + U_i)^2 + \sum_{j=1}^{k_2-m_+}(b' + V_j)^2 + \sum_{i=1}^{m_-}(a + A_i - Y_i)^2 + \sum_{j=1}^{m_+}(b + B_j + Z_j)^2 = 1 \\
\sum_{i=1}^{k_1-m_-}(a + A_i - a' - U_i)^2 + \sum_{j=1}^{k_2-m_+}(b + B_j - b' - V_j)^2 + \sum_{i=1}^{m_-} Y_i^2 + \sum_{j=1}^{m_+} Z_j^2 = \delta^2
\end{array}
.
$$

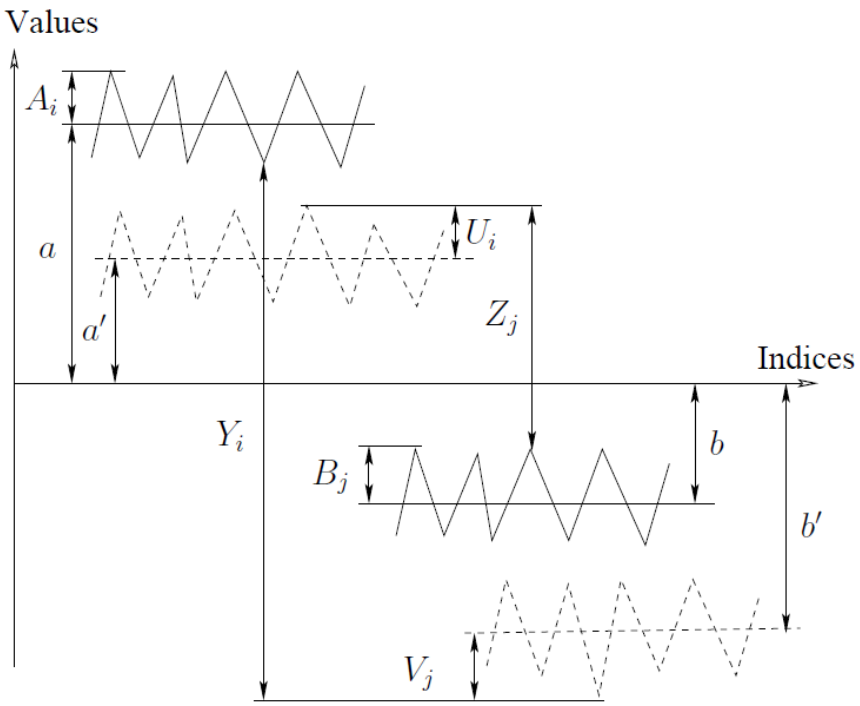


Figure 3.19: Notations in the proof.

The two first constraints enforce $\|\mathbf{v}_c\|_2 = 1$ and $\|\tilde{\mathbf{v}}_c\|_2 = 1$, the last constraint enforces $\|\mathbf{v}_c - \tilde{\mathbf{v}}_c\|_2 = \delta$.

Without loss of generality we assume $a, a' > 0$, $b, b' \leq 0$ and $(a' - b)^2 < (a - b')^2$. Assumption A3 ensures $E[Y_i] = a - b'$ and $E[Z_j] = a' - b$ for all i and j. Then

$$
\begin{array}{rcl}
E[Y_1^2] + ... + E\left[Y_{m_-}^2\right] & \leq & \frac{1}{m_-}(E[Y_1] + ... + E[Y_{m_-}])^2 = m_-(a - b')^2 \\
E[Z_1^2] + ... + E\left[Z_{m_+}^2\right] & \leq & \frac{1}{m_+}(E[Z_1] + ... + E[Z_{m_+}])^2 = m_+(a' - b)^2
\end{array}
.
$$

summing both inequalities and substituting the terms in the third constraint of the optimization problem,

$$m_+(a'-b)^2 + m_-(a-b')^2 \leq E\left[\sum_{i=1}^{m_-} Y_i^2\right] + E\left[\sum_{j=1}^{m_+} Z_j^2\right] \leq \delta^2 \tag{3.1}$$

Maximizing m under the constraint $(a'-b)^2 < (a-b')^2$, we must set $m_- = 0$ and $m = m_+$. Hence equation 3.1 becomes $m(a'-b)^2 \leq \delta^2$, under which m is maximized if $b = 0$.

We denote by σ_Z^2 the variance of Z_i. Choosing $b = 0$ and $m = m_+$ simplifies the first constraint of the optimization problem into

$$\sum_{i=1}^{k_1}(a'+U_i)^2 + \sum_{j=1}^{k_2-m}(b'+V_j)^2 + \sum_{j=1}^{m}(B_j+Z_j)^2 = 1.$$

If we take the expectations of both sides and use the assumptions that the U_i's and the V_j's are identically distributed with zero mean, we obtain

$$k_1 a'^2 + k_1 E\left[U_1^2\right] + (k_2-m)b'^2 + (k_2-m)E\left[V_1^2\right] + m(a'^2 + \sigma_Z^2 + E\left[B_1^2\right]) \tag{3.2}$$

hence $\sigma_Z^2 \leq \frac{1}{m}$ or $\sigma_Z^2 = \frac{\beta}{m}$ for some $0 \leq \beta \leq 1$. Setting $b'^2 = a'^2 - E\left[V_1^2\right] + \sigma_Z^2 + E\left[B_1^2\right]$, equation 3.2 becomes

$$k_1 a'^2 + k_1 E\left[U_1^2\right] + k_2 b'^2 + k_2 E\left[V_1^2\right] = 1.$$

Replacing $b'^2 = a'^2 - E\left[V_1^2\right] + \sigma_Z^2 + E\left[B_1^2\right]$, we have

$$k_1 a'^2 + k_1 E\left[U_1^2\right] + k_2(a'^2 - E\left[V_1^2\right] + \sigma_Z^2 + E\left[B_1^2\right]) + k_2 E\left[V_1^2\right] = na'^2 + k_2\sigma_Z^2 + k_1 E\left[U_1^2\right] + k_2 E\left[B_1^2\right] = 1.$$

Hence

$$na'^2 = 1 - k_1 E\left[U_1^2\right] - \frac{k_2\beta}{m} - k_2 E\left[B_1^2\right]. \tag{3.3}$$

Choosing $b = 0$ and $m = m_+$ also simplifies the third constraint of the optimization problem

$$\sum_{i=1}^{k_1}(a + A_i - a' - U_i)^2 + \sum_{j=1}^{k_2-m}(B_j - b' - V_j)^2 + \sum_{j=1}^{m}(a' - B_j - U_j)^2 = \delta^2.$$

Taking the expectation on both sides

$$k_1((a-a')^2 + E\left[A_1^2\right]) + k_1 E\left[U_1^2\right] + k_2 E\left[B_1^2\right] + (k_2-m)(b'^2 + E\left[V_1^2\right]) + m(a'^2 + E\left[U_1^2\right]) = \delta^2$$

which implies

$$k_1 E\left[U_1^2\right] + k_2 E\left[B_1^2\right] \leq \delta^2 - \beta - ma'^2 \tag{3.4}$$

or

$$(n-m)a'^2 \leq \frac{(n-m)(\delta^2-\beta)}{m}. \tag{3.5}$$

If we substitute 3.4 into 3.3 and combine it with 3.5, we obtain

$$1 - \delta^2 + \beta - \frac{k_2}{m}\beta \leq (n-m)a'^2 \leq \frac{(n-m)(\delta^2-\beta)}{m}.$$

Rearranging terms, we get

$$\begin{array}{rcl} m & \leq & n\delta^2 - n\beta + k_2\beta \leq n\delta^2 \\ \eta = \frac{m}{n} & \leq & \delta^2 = \|\tilde{\mathbf{v}}_2 - \mathbf{v}_2\|_2^2 \end{array}.$$

■

Proof of theorem 8 of chapter 2 : We define $d\Delta = \tilde{\Delta} - \Delta$ From the theory of perturbation of eigenvalues of a symmetric matrix, we have

$$\begin{array}{rcl} \tilde{\mathbf{v}}_c - \mathbf{v}_c & = & \sum_{j=1}^{n-1} \frac{\mathbf{v}_j^T d\Delta \mathbf{v}_n}{\lambda_n - \lambda_j} \mathbf{v}_j + O(d\Delta^2) \\ & = & \sum_{j=1}^{n-1} \frac{\mathbf{v}_j}{\lambda_n - \lambda_j} \sum_{p=1}^{n} \sum_{q=1}^{n} v_{pj} v_{qn} d\Delta_{pq} \\ & = & \sum_{p=1}^{n} \left[\left(\sum_{q=1}^{n} v_{qn} d\Delta_{pq} \right) \left(\sum_{j=1}^{n-1} \frac{v_{pj} \mathbf{v}_j}{\lambda_n - \lambda_j} \right) \right] \\ & = & \sum_{p=1}^{n} \beta_p \mathbf{u}_p \end{array} \tag{3.6}$$

where

$$\begin{array}{rcl} \beta_p & = & \sum_{q=1}^{n} v_{qn} d\Delta_{pq} \\ \mathbf{u}_p & = & \sum_{j=1}^{n-1} \frac{v_{pj} - \mathbf{v}_j}{\lambda_n - \lambda_j} \end{array} .$$

Taking the expectation of both members in 3.6 we obtain

$$\begin{array}{rcl} E\left[\|\tilde{\mathbf{v}}_c - \mathbf{v}_c\|_2^2\right] & \simeq & E\left[\| \sum_{p=1}^{n} \beta_p \mathbf{u}_p \|_2^2\right] \\ & = & \sum_{p=1}^{n} E\left[\|\beta_p \mathbf{u}_p\|_2^2\right] + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left[\beta_i \beta_j \mathbf{u}_i^T \mathbf{u}_j\right] \end{array} .$$

Our experimental works showed that the second term of the left member of the equality above is either close to zero or negative, which means that $\beta_i \mathbf{v}_i$ and $\beta_j \mathbf{v}_j$ are either very weakly correlated or negatively correlated. This observation yields the following empirical inequality

$$E\left[\|\tilde{\mathbf{v}}_c - \mathbf{v}_c\|_2^2\right] \lesssim \sum_{p=1}^{n} E\left[\beta_p^2\right] \|\mathbf{u}_p\|_2^2.$$

As shown in [36], $E\left[\beta_p^2\right]$ is related to $d\Delta$ by the following inequality

$$E\left[\beta_p^2\right] = E\left(\sum_{q=1}^{n} v_{qn} d\Delta_{pq} \right)^2 \leq \sum_{i=1}^{n} \sum_{j=1}^{n} \left[v_{in} v_{jn} E\left(dL_{pi}\right) E\left(dL_{pj}\right) + |v_{in} v_{jn}| \sigma_{pi} \sigma_{pj} \right]$$

where σ_{pi} is the variance of $d\Delta_{pi}$. ■

3 Barclays Premier League Scores

The following table contains the score of each game played by the teams of the Barclays Premier League of football. Each team played two games against three other teams for a total of six games per team.

Home Team	Away Team	Home Score	Away Score
Burnley	West Ham	2	0
Sunderland	Everton	1	3
Man United	Arsenal	1	2
Man City	Tottenham	3	2
Bolton	Hull City	0	0
Liverpool	Portsmouth	4	0
Chelsea	Stoke City	6	1
Fulham	Wigan Ath	3	0
Birmingham City	Blackburn	1	2
Wolves	Aston Villa	1	1
West Ham	Burnley	0	2
Everton	Sunderland	3	1
Arsenal	Man United	2	1
Tottenham	Man City	2	3
Hull City	Bolton	0	0
Portsmouth	Liverpool	0	4
Stoke City	Chelsea	1	6
Wigan Ath	Fulham	0	3
Blackburn	Birmingham City	2	1
Aston Villa	Wolves	1	1
Burnley	Everton	2	0
Sunderland	Arsenal	1	3
Man United	Tottenham	1	2
Man City	Hull City	3	2
Bolton	Portsmouth	0	0
Liverpool	Stoke City	4	0
Chelsea	Wigan Ath	6	1

Home Team	Away Team	Home Score	Away Score
Fulham	Blackburn	3	0
Birmingham City	Aston Villa	1	2
Wolves	West Ham	1	1
Everton	Burnley	0	2
Arsenal	Sunderland	3	1
Tottenham	Man United	2	1
Hull City	Man City	2	3
Portsmouth	Bolton	0	0
Stoke City	Liverpool	0	4
Wigan Ath	Chelsea	1	6
Blackburn	Fulham	0	3
Aston Villa	Birmingham City	2	1
West Ham	Wolves	1	1
Burnley	Arsenal	2	0
Sunderland	Tottenham	1	3
Man United	Hull City	1	2
Man City	Portsmouth	3	2
Bolton	Stoke City	0	0
Liverpool	Wigan Ath	4	0
Chelsea	Blackburn	6	1
Fulham	Aston Villa	3	0
Birmingham City	West Ham	1	2
Wolves	Everton	1	1
Arsenal	Burnley	0	2
Tottenham	Sunderland	3	1
Hull City	Man United	2	1
Portsmouth	Man City	2	3
Stoke City	Bolton	0	0
Wigan Ath	Liverpool	0	4
Blackburn	Chelsea	1	6
Aston Villa	Fulham	0	3
West Ham	Birmingham City	2	1
Everton	Wolves	1	1