LUISS

Department of Economics and Finance
Chair of Financial and Economic Networks

Community Detection Algorithms for Network Data: a Markovian Approach

SUPERVISOR
Dr. Matteo Quattropani

CO-SUPERVISOR
Prof. Sara Biagini

CANDIDATE
Roberto Calò
Matr. 727251

Academic Year 2021/2022

Abstract

The last few years have been characterized by an ever increasing amount of data: we have been overwhelmed by a digital revolution that has deeply changed our lives.
In our everyday life we interact with each other and more and more the interaction involves electronic devices, interlinked and connected to each other.
As a consequence, each type of interaction generates large volumes of data, constantly collected and analysed. This led to the need to be able to study the so called Big Data, and hence light has been shed on new fields of study, so to be able to fulfill the task.
In this thesis we are going to present one of these recent fields called *Network Analysis*, and in particular we will focus on one of its branches that comes with the name of *Community Detection*. What we will do, in short, is to investigate different methods to identify the community structure in network-type data. Then we will focus on a particular and recently proposed technique based on the so called *2-Choices Dynamics*. Ultimately, we will test this method by some simulations over both computer-generated networks and a real-world one, to test its efficiency.

Contents

1 Introduction

The last few decades have been characterized by a shocking technological and digital development. Internet allowed the exchange of information instantly regardless of the distance. Computers and smartphones helped to make this even easier, just with a "click" or with a "tap".

As a result, large amount of data are constantly generated by anyone of us: when buying something online, when chatting with friends, when we send an email, when we download an app, etc. All these various data are being continuously exchanged, collected and stored in massive databases and that is the reason why we commonly call them *Big Data*.

Due to their complexity and extent, both in terms of volume and of type, to be studied they require proper methods of analysis, capable of putting into relation heterogeneous data. Being able to perform this task successfully means to be able to uncover trends and patterns, evaluate and forecast and, in general, to make data-driven decisions.

In this context, new fields of studies emerged and became popular in the recent years with the purpose of improving big data analysis. One of these fields is the so called *Graph Theory*, a branch of mathematics, that saw his birth in 1736 with Euler solving the problem of the "Seven Bridges of Königsberg". Since then, some mathematicians explored that field and laid the foundations for its development, up to the second half of 20th century, when the interest toward the subject grew considerably thanks to the always increasing availability of suitable data.

At the beginning of the new century, the growing need to interpret network-type data, gave a strong impulse to the implementation of a practical application. This led to the development of what we call *Network Analysis*, a field which studies complex networks data, based on the theoretical concepts of graph theory. It is a mix of the mathematical notions with the practical approach of statistics, physics and algorithms. Its ultimate aim is, beside the theoretical study of the topology of a network, to extract both qualitative and quantitative information from network data.

In this thesis we are going to present first some general concepts of graph theory and then we will focus on a particular branch of network analysis, called *Community Detection*.

Structure of the Thesis

Before going into details, we briefly present the structure of this thesis.
In the following sections of this Chapter 1, we will provide the theoretical framework and basic definitions of networks needed to properly understand the subsequent parts.
In Chapter 2 we will explore the community detection issue, first by giving some widely accepted definition of communities (Section 2.1) and then by investigating some famous and interesting methods and algorithms to perform community detection (Section 2.2 to Section 2.4).
In Chapter 3, we start in Section 3.1 by introducing Markovian processes on networks and then we focus in particular on the 2-choices dynamics [1] and on the community detection method based on it proposed by Cruciani et al. [2]. We will provide first an informal and gross description of the functioning of the process (Section 3.2) and then we report the formal mathematical and theoretical proofs in Section 3.3 and in its subsections.
The Chapter 4 is focused on testing the efficiency of the just described method. These simulations were carried out both on computer-generated networks and on a real-world network. The results are reported respectively in Section 4.1 and in Section 4.2.
Finally the Conclusion sums up the whole work. In addition, in Chapter 7 we report the Matlab code used to create the computer-generated networks (Section 7.1) and the code of the algorithm based on the 2-choices dynamics (Section 7.2).

1.1 What is a Network?

A *network* or a *graph* is a mathematical structure consisting of two entities: a set of agents, which we call *nodes* (or *vertices* or *points*) and a set of pairwise interactions among them. The presence of a relationship is shown up by the presence of an *edge* (or *link*) between two nodes.
Graph theory and network analysis have been recently employed to better understand, describe, optimize and sometimes also to try to predict social, biological, virtual, technological, economic and any other type of relationship. They provide helpful tools that try to give an explanation to any kind of interaction that any kind of agent, which is part of a network, has with the others. To give practical examples, when we talk about networks we refer for instance to the network of friends on a social network, the network of web pages interlinked with each other, biological networks like the food chain or economic networks like the so called World Trade Web, the international web made of import-export relationships among countries.
Almost any activity that involves entities interacting with each other can be represented by a graph. Obviously, each network presents its own features, but there are some of them which deserve to be mentioned since they are shared by a remarkable set of real-world networks:

- to be *sparse*: they have few links compared to the maximum possible;
- to be *small worlds*: the average distance between two random-picked nodes is small;
- to present *inhomogeneity* in the edge distribution: they have many vertices with few links and few vertices with many links;
- to present a *clustered structure*: it is common to observe groups of vertices with a dense concentration of links within them, and few edges connecting to other groups.

This last feature of real-world networks is called *community structure* (or clustering) and can be explained by the fact that groups of vertices that share common characteristics are probably linked to each other and play a similar role in the whole system [3] (grossly, "friends of friends are usually friends"). These groups are called communities (or clusters) and their identification can play an important role in all those fields whose structure can be represented by a graph.

In practice this last concept can have a wide range of application domains: in a generic network the classification of vertices according to their structural position may help to find the central nodes of a community, which are most likely to be important for the stability or well-functioning of the whole, while vertices on the boundaries of clusters may play an important role in mediation with other groups.

For an online platform, for example, it could be convenient to identify clusters of customers with similar interests and preferences. This would allow to set up a smart and targeted advertising campaign and more importantly to create an efficient recommendation system, that suggests the user something that probably is glad to meet. A further possible implementation is in the field of epidemiology, really well known nowadays, with the analysis of social communities to help the tracking of the spread of an infectious disease. Furthermore, also studies on how to set up an optimized vaccination campaign is part of this field, for instance, giving the priority to be protected to most central vertices in the social structure [4].

The activity of identification of communities has been faced with some methods that have been proposed through the recent years and others continue to be proposed. The task is not so easy as one may imagine, and still today there is no a generally recognized efficient method to perform community detection in practice.

We will give a brief illustration of the most widely used techniques both simple and more complex, and then we will present and focus on an interesting and recent method proposed by Cruciani et al. [2], that combines classic techniques with new ideas. We will then present the results of the efficacy of the method, resulting from running some simulations on networks whose community structure is known a priori.

The aim of this work is first to provide a good definition and understanding of communities and of the issue related to how to identify them, and to report some interesting solutions.

1.2 Basics of Graph Theory

First, let's give a proper definition to the aforementioned objects with the support of an example and figures.
The nodes or vertices or points are the fundamental elements of a graph that represent the agents. They are identified by a label and, if there is a relationship between them, they are connected by an edge or link. Usually, a graph G is associated to a pair of sets: $G(V, E)$, where V is the set of vertices and E is a subset of V^2 and is made of a list of the pairs of connected nodes, i.e. the edge list.

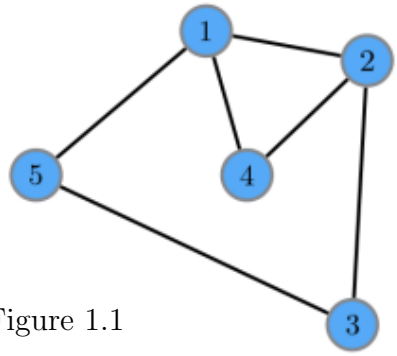


Figure 1.1

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(1, 2), (4, 1), (4, 2), (1, 5), (2, 3), (3, 5)\}$$

The most convenient way to represent graphs' structure is by using matrices. The most natural choice is the *Adjacency matrix*, which is a square matrix that takes as entries 0 or 1 in the position i, j depending on the presence of a link between the node i corresponding to the i-th row and the j one corresponding to the j-th column. In the case of the graph G of Figure 1.1, the adjacency matrix A would be:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Actually, the adjacency matrix can assume any non negative number as entry, not only 0 and 1. If so, we are dealing with a *weighted* graph, in which the weights are given by the number appearing in the corresponding entry and it represents the intensity of the relationship between the two nodes under consideration: the higher the entry, the higher it is the intensity. As it becomes clear, the case considered in the example above is an *unweighted* graph.
Depending on whether the link has a direction, namely one of the two nodes is the source and the other the target and not necessarily viceversa, we distinguish between *directed* and *undirected* graphs. The figure above represents an undirected graph, in which the pairs of nodes are unordered ($\{1, 2\} = \{2, 1\}$) and the interaction between them is reciprocal. In the

directed case, the links will no more be just lines, but arrows pointing vertices depending on the orientation of the relationship. As a result, in the undirected case, the adjacency matrix will be symmetric ($A_{ij} = A_{ji}$). A typical clarifying example of the difference between undirected and directed graphs is based on the comparison of online social networks: Facebook versus Twitter and Instagram. In the first case the relationship between nodes is reciprocal, since, after accepting the friend request, both the individuals will appear in the friends' list of the other. Clearly, in this case, the underlying network will be undirected. On the contrary, in social networks like Twitter or Instagram, in which you decide to follow one account that will not necessarily follow you back, the newly created relationship will be one-way and hence directed.

It is also possible to have nodes with *self-loops*, which corresponds to vertices linked with themselves. In this case, on the main diagonal of the adjacency matrix there will be some entries different from 0.

In this thesis we are going to deal with the simplest form of graphs, namely unweighted, undirected and with no self-loops, but note that any result can be extended to any type of graph with the proper adjustments.

Finally, two nodes connected by a link are called *neighbours* and the *degree* of a vertex is the number of neighbours that it has. The degree of the vertex i can be easily obtained from the adjacency matrix by just summing over the i-th row or column. The *degree sequence* is the list of the degrees ordered with respect to the vertices' label.

These are just basic definition to help an initial approach to the subject, further theoretical explanations will be given later on when required.

2 Community Detection

2.1 Communities and Partitions

Now that the main principles of graph theory have been provided, we can start focusing on the main topic of community detection. Before starting, it is important to make two considerations: first, the concept of community does not have a rigorous definition, due to its complexity, hence we will use some degree of arbitrariness and common-sense for our purpose. Second, community detection can be performed only in graphs which are sparse, which means that the number of nodes and the number of edges needs to have the same order of magnitude [3]. As mentioned, this is the case for many real-world networks.
We now present different and widely accepted definitions of community to become more familiar and have a better understanding of the issue.
A quantitative definition, even if not so rigorous, considers a community a group of nodes in which we observe considerably more edges connecting vertices of the same group compared to the edges connecting with the rest of the nodes of the graph. Defining the *intra-cluster density* as the ratio between the internal edges of a cluster and the total possible number of internal edges, and the *inter-cluster density* as the ratio between the number of edges that link nodes belonging to different clusters and the total possible external edges, we expect to have a significantly larger value of the intra-cluster density compared to inter-cluster one. Regarding this definition, problems may arise in the comparison between the two densities and in the quantification of "significantly larger value": it does not exist a benchmark for this, as it strongly depends on the case we are studying. These are the type of problems we can face and in which the above mentioned arbitrariness comes into play.
An alternative definition can be based on the *robustness* of the clusters. It relies on the idea that the higher the number of edges that need to be removed to disconnect two vertices, the larger the nodes are likely to belong to the same community.
Another common type of definition is based on *vertex similarity* with respect to some property, for example considering two nodes with the same neighbours as belonging to the same community, even in absence of a link connecting them [5]. Still in this context it is interesting to mention vertex similarity based on random walks on graphs: the commute-time, i.e. the average number of steps taken by a random walker to pass from one vertex to another, can be a significant indicator for cluster identification: smaller the time, higher the chances to

be in the same community [6].

As it can be imagined, the same graph may have different combinations of communities depending on the purpose of the study to be performed. For example, a community may contain micro-communities within it, that would be interesting to analyse if we are conducting a specific study, rather than a general one in which we would be more interested in the macro-communities. Each division in clusters is called a *partition* of the graph.
Besides the raw and different definitions of community, it is as well important to assess quantitatively how much a certain partition of a graph represents its community structure. In order to do that we exploit a function called *modularity*, that quantifies the goodness of each partition by measuring the strength of a partition comparing the actual edge density of a graph with the expected density of the graph with the same degree sequence but with links attached at random. This idea is based on the fact that a graph with random edges is not expected to exhibit clustered structures. It is obviously crucial the choice of the model for the comparison; the decision is arbitrary, the only constraint is to choose a model that keeps unchanged the degree distribution of the original graph.
As it will be extensively discussed in Section 2.4, modularity is a function that sums over all the possible pairs of vertices, and that tells how much, in a certain partition, nodes are connected with respect to the average. In principle the goal should be to maximize its value, to get the best partition in terms of communities.
Even though the maximization of modularity provides a rigorous definition of "best" partition, there are some drawbacks. The question whether a partition is better than another strongly depends on the definition of community considered, hence it cannot be considered as an optimum. Furthermore, modularity does not give reliable results in case we are dealing with a network in which we expect communities which are significantly different in size and its maximization over the space of all the possible partitions of a graph can be a complex problem due to the fact that the number of all the possible partitions is exponential in the number of the vertices.
In any case, this is the most used method for a qualitative evaluation of partitions and we will use it as benchmark for our results.

2.2 Traditional Methods

After this brief overview, we will make another preliminary consideration regarding the real community detection process. To perform the task, we make use of *algorithms* which will deal with the large amount of information embedded in a network. It is important to introduce the definition of *computational complexity* as the estimate of the amount of resources needed by the algorithm to run; by saying so, we include both the number of computation

steps and the number of simultaneous operations performed at each step. Usually, the complexity is scaled with respect to some input parameters, for example the number of nodes of the network. Based on difficulty we can distinguish among P, NP and NP-hard problems. In our context, algorithms need to deal with NP-hard problems (most complex ones), hence it is common to use *approximation algorithms* that do not give an exact solution but provide a good approximation with the advantage of a significantly lower complexity. This means that the solution obtained will not probably be the optimal one, but a fairly good one. However, the algorithm maintains its efficiency.

The first and most simple class to be presented comes with the name of *graph partitioning algorithms*. The approach consists in creating a partition of nodes divided in a number of communities such that it is minimized the number of edges lying between clusters. The number of links between clusters is formally called *cut size*. Clearly with this method we do not expect to find optimal results as it is too rough and requires too many input information such as the number of communities K and the size of communities, since otherwise lowest degree nodes would be obviously isolated from the rest of the graph. It would be instead preferable to use an algorithm which takes as inputs the less information possible (that are presumably unknown at the beginning) and that produces them autonomously in the output results.

To overcome this issue, the class of *hierarchical clustering algorithms* has been introduced. Those are built to detect the multilevel structure of a graph. In fact, the *hierarchical structure* is a quite common property of many real-world networks: communities are included in larger communities, which in turn are included in larger communities and so on (like the human body, composed by organs, composed by tissues, composed by cells). The first step is to choose a similarity measure between vertices depending on the case under study. Then the algorithm can start:

1. Calculate the score of the similarity measure for each possible pair of nodes.
2. The pairs which are not connected, are ordered from the most up to the less similar ones.
3. Edges are added, one by one, following the ranking of the prior step.

By adding the edges, the graph should bring together similar vertices and create larger components which end up being our communities. Obviously if the process is performed up to the less similar pair of nodes, we will end up with just one component with all nodes connected. Therefore, it is usually imposed some stopping criterion, for example a given number of communities or a certain value of modularity.

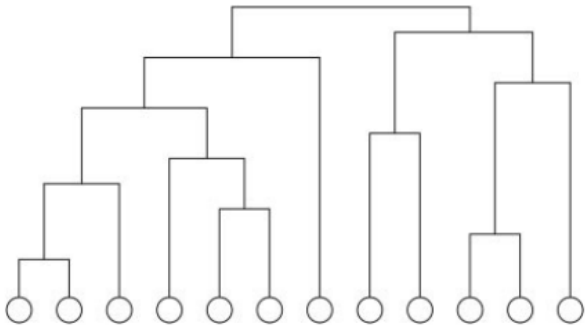


Figure 2.1: As the process goes on, we move from the bottom to the top of the figure.

The main advantages of this algorithm are that is quite easy, and it requires few inputs to work. On the other hand, it does not provide any information on the goodness of partitions obtained and may give unreasonable results if the network under consideration does not have any hierarchical structure. Furthermore, there is a problem with vertices with only one neighbour: they tend to be separated from the cluster they should belong. This can be explained by a low score in the similarity ranking (due to the little information they carry). All in all, this method can be surely useful, but can produce poor results and they may be difficult to interpret.

Opposed to this method, that focuses on creating larger clusters by adding edges, there is a class called *divisive algorithms*, which instead focuses on the removal of edges which are considered to lie between communities. As a consequence, clusters will tend to get disconnected from each other and the community structure would be highlighted. As before, also in this context it is crucial the choice of a betweeness measure. The most famous and used algorithm of this class is the one proposed by Girvan and Newman [7], that exploits the *edge centrality* [8] to quantify the betweeness of an edge. The definition of edge centrality of a link is the number of shortest paths between any pair of vertices that go through it. This quantity should reflect the importance that a link has in the communication between clusters. To be more clear, central edges of a community can be easily bypassed due to the high density of links. On the contrary, edges on the boundary of a cluster play a crucial role in inter-community communications. For this reason, we expect high values of edge betweeness for boundary edges and low values for central ones.
The algorithm works as follows:

1. Calculation of edge betweeness for all edges of the network.
2. Find the edge with the highest score and removal.
3. Recalculation of edge betweeness.
4. Repeat from step 2

The key passage of this algorithm is the recalculation step, which continuously updates the betweeness measure and allows to always make the best move. This algorithm turns out to be simple, intuitive and to give reliable results [9], the only drawback is its computational

complexity. It requires many computation steps, and it results to be efficient in networks of up to 10^4 nodes, which in our big data world can be limiting.

2.3 Spectral Clustering

One more interesting and elegant technique that needs to be mentioned is *spectral clustering*. This method creates partitions of graphs by making use of the eigenvectors of the matrix which represents the network, like the previously mentioned adjacency matrix (or the laplacian, as it will be discussed soon).
By this method the initial set of nodes is transformed in a set of points in the Euclidean space where the orthonormal basis is that defined by the eigenvectors of the matrix representing the network under consideration. The reason why this technique works, is due to the change of representation: the cluster properties of the network become much more evident.
The most used matrix of network structure is by far the *Laplacian matrix* L. It is defined as the difference between the *degree matrix* and the *adjacency matrix*. The latter has already been explained, regarding the degree matrix D, it is simply the diagonal matrix whose elements D_{ii} equal the degree of vertex i. An example can surely help to make it clear. Let's reuse the graph of Figure 1.1:

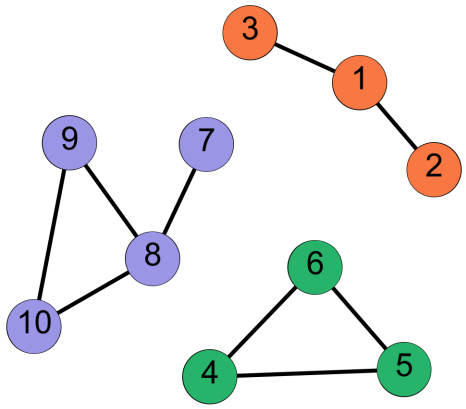
$$L = D - A$$

$$\begin{pmatrix} 3 & -1 & 0 & -1 & -1 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 2 & 0 & -1 \\ -1 & -1 & 0 & 2 & 0 \\ -1 & 0 & -1 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

By construction the laplacian is symmetric[1] and the sum over each row is 0. Therefore, we can surely say that there is at least one zero eigenvalue and it is surely the smallest one since it is a diagonally dominant matrix. In addition, the algebraic multiplicity of the zero eigenvalue equals the number of connected components[2] of the graph. Here we present a trivial example:

[1] Still if the graph is undirected and unweighted.

[2] Connected component: subgraph in which it is possible starting to any vertex to reach any other vertex. If the component is just one, it is directly called connected graph.



$$L: \begin{pmatrix} 2 & -1 & -1 & & & & & & & \\ -1 & 1 & 0 & & 0 & & & 0 & & \\ -1 & 0 & 1 & & & & & & & \\ & & & 2 & -1 & -1 & & & & \\ & 0 & & -1 & 2 & -1 & & 0 & & \\ & & & -1 & -1 & 2 & & & & \\ & & & & & & 1 & -1 & 0 & 0 \\ & 0 & & & 0 & & -1 & 3 & -1 & -1 \\ & & & & & & 0 & -1 & 2 & -1 \\ & & & & & & 0 & -1 & -1 & 2 \end{pmatrix}$$

In this case it is clear and evident the community structure from the block form of the laplacian. We can surely say there are three connected components and hence algebraic multiplicity of zero eigenvalue equal to 3. Clearly this is an extreme simplification, and it would be impossible in a real case to check for communities by just looking at the laplacian: the nodes would not be ordered like in the example and so the blocks would not be so clear. Still, we can exploit the relation in the other sense: retrieve the number of connected components by calculating the multiplicity of the zero eigenvalue. Furthermore, by observing the eigenvectors associated to the zero eigenvalue, the community structure is easily revealed. In the example above:

$$v_1 = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$$
$$v_2 = [0, 0, 0, 1, 1, 1, 0, 0, 0, 0]$$
$$v_3 = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]$$

Realistically speaking, the community detection problem is not as easy as in the case of an unconnected graph, since we rather expect a connected network having K subgraphs with fewer links among each other. In this case the zero eigenvalue will be only one and all the others will be positive. The associated eigenvector will be formed by only ones and

the laplacian would not be a block-diagonal matrix, even if it will still exhibit many zeros outside the diagonal blocks (but not everyone) [10].
However, there is still an interesting property to exploit: in a graph with K communities, the zero eigenvalue of the laplacian will have algebraic multiplicity equal to one, but the following K eigenvalues will be much more close to 0 compared to the $(K+1)$-th eigenvalue. This means that the number of clusters can be retrieved by looking at the eigenvalues, finding a relatively large gap in the spectrum. However, it can be hard to identify significant gaps, especially in graphs with overlapping or mixed communities.
So far, we described some of the interesting properties that make the laplacian matrix so suitable for our purpose, now we can take a step further and see how it works in practice.
The method by Donetti and Muñoz [11] uses the eigenvectors of the laplacian. As we just saw, for unconnected graphs the eigenvectors related to the zero eigenvalues will only take values 0 and 1 depending on the belonging to the cluster. This can be easily extended to connected graphs, since the values of the components of the lowest m eigenvectors will be close to each other for nodes belonging to the same community. It is then possible to plot the graph in an m-dimensional space and communities would appear as separated groups of points. The separation becomes more evident as the number of dimensions m increases.

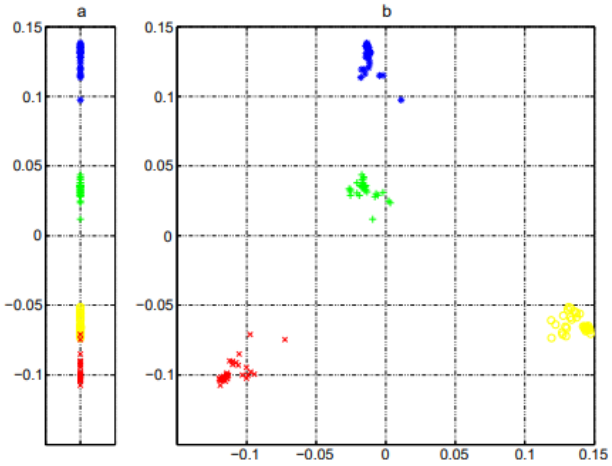


Figure 2.2: From Donetti and Muñoz, the diagram a on the left shows the spectral clustering by using one eigenvector, the right chart b shows the same result by using two eigenvectors.

By proceeding in this way, the solution would be visual, but it is not how we would deal with the problem of community detection of a graph, even because if we have more than three dimensions we would not be able to visualize the metric space. It is necessary to introduce a quantitative measure to solve the problem analytically.
The easiest and most direct way to approach the issue is to simply calculate the distances between vertices, for example the Euclidean one:

$$d(x,y) = \sqrt{\sum_{i=1}^{m}(y_i - x_i)^2}.$$

The distance can be seen as a measure of dissimilarity between vertices and we can exploit it to separate the points into clusters. To do so, we need to find a function to minimize based on distance.
The most common and widely used method is the *k-means*, in which the function to minimize is the following squared error function:

$$\sum_{i=1}^{K} \sum_{x_j \in S_i} ||x_j - c_i||^2,$$

where S_i is the subset of nodes in the $i-th$ cluster and c_i is its centroid, i.e. an imaginary location of the center of a cluster[3]. It is important to note that the number of clusters K has to be given as an input of the method.
The algorithm starts with a configuration in which the K centroids are as far as possible from each other and each node is assigned to the nearest centroid. After that the center of each cluster obtained is recalculated and vertices are reclassified. After few iterations, the centers will remain stable and the clusters will not change.
This technique is quite easy to implement and computationally efficient, the main issue is, again, that it is not able to derive itself the number of clusters K, since they need to be specified at the beginning. Nevertheless, it delivers really good results if combined with other techniques, possibly able to get themselves the number of clusters. The *k-means* is the method that will be used in Chapter 4 to group nodes together.

Coming back to the general spectral clustering methods, the main drawback is the computation of the eigenvectors. If the graph is large, an exact computation would be impossible, so also here, some approximation techniques are required. Beside this, the number m of eigenvectors that need to be calculated in order to obtain a clear separation of communities in a graph is not known at the beginning of the process. Usually the maximum number of eigenvectors m to be calculated is set a priori.
As it should be clear at this point, spectral clustering is not a stand-alone method for community detection, and it just partly works for the purpose. It simply provides, in an elegant way better input data for other algorithms[4].

2.4 Modularity Based Methods

In this section we will discuss some algorithms which aim directly at maximizing the modularity, that, we recall, is taken as a measure of the goodness of a partition from the point of view of community structure. The underlying assumption of this part is that high values

[3]It can be either an actual node or even an imaginary point in space.
[4]like $k - means$.

of modularity mean good partitioning.
First, let's give a precise mathematical formulation. Call Π a generic partition of a graph in K disjointed blocks:

$$\Pi = (\Pi_1, \ldots, \Pi_K)\,,$$

the modularity associated to the partition Π is defined as follows:

$$Q(\Pi) = \frac{1}{2L} \sum_{i}^{N} \sum_{j}^{N} (A_{ij} - P_{ij})\, \delta(\Pi_{(i)}, \Pi_{(j)})\,,$$

where L is the number of links, N the number of nodes, A the adjacency matrix, P the matrix of expected number of edges between vertices i and j in the null model, $\Pi_{(i)}$ the community of vertex i and δ a function that yields one if vertices i and j are in the same community ($\Pi_{(i)} = \Pi_{(j)}$) and zero otherwise.
The choice of the null model, as said before, is arbitrary. The most used one in practice is the *Chang-Lu model*, in which we consider a graph sampled at random fixing a priori an average degree sequence $(d_1, \ldots, d_N)$ and $2L = \sum_i^N d_i$.
The probability of having an edge between vertex i and vertex j equal to[5]:

$$\mathbf{P}_{ij} = \frac{d_i d_j}{2L}\,.$$

This implies that the expected degree of the vertex i equals d_i. The result is a random graph with the expected degree sequence equal to the actual degree sequence of the graph under analysis.
In this case the final expression of modularity would be:

$$Q(\Pi) = \frac{1}{2L} \sum_{ij}^{n} \left(A_{ij} - \frac{d_i d_j}{2L} \right) \delta(\Pi_i, \Pi_j)\,.$$

Since the contributions to Q come only from the pairs of vertices that belong to the same cluster (due to the δ function), it is possible to rewrite the equation above as a sum over the number of clusters:

$$Q(\Pi) = \sum_{k=1}^{K} \left(e_k - (f_k)^2 \right)\,, \tag{2.1}$$

where K is the total number of communities, e_k is the fraction of edges that join vertices of the same cluster k and f_k is the fraction of links that have one or both vertices inside the community k. Single contributions can be both positive and negative and the range of values that modularity can take is between -1 and 1. For the goal of community detection, we aim to maximize modularity, since we can surely say that the more the number of internal edges of a cluster exceeds the expected edges, the better defined is the community.

[5]We assume also that the graph does not present any vertex i such that $d_i > \sqrt{2L}$

This maximization problem is NP-hard and so, again, we will work with algorithms that provide a fairly good approximation of the modularity maximum, but not the real one.
The simplest approach is a greedy technique presented by Newman that starts with each vertex being a cluster and no edges between them. Communities are combined together following this criterion: from the original graph an edge is picked such that it gives the maximum increase in modularity with respect to the actual configuration. All other edges are added based on the same principle. At each step we will have a different partition and the final outcome of the algorithm will be the one having the maximum value of modularity among all partitions obtained. The main advantage of this algorithm is that it is quite fast and allows the analysis of large networks.
However, it tends to create large communities at the expenses of small ones, and it appears to give results less accurate compared to the divisive algorithm [12].

Another technique of the modularity class is the so-called *extremal optimization*. It is a heuristic technique that sees modularity as the sum of the individual contribution of each vertex. Hence from Equation (2.1) we can define the contribution of each vertex as:

$$q_i = l_{(i)} - d_i f_{(i)} \,,$$

where (i) is the community where the vertex i lies, $l_{(i)}$ is the number of links that the node i has with nodes in its same community and $f_{(i)}$ is defined as above [13]. This local measure depends on the degree of each node, so it needs to be re-scaled by dividing precisely for the degree. The quantity so obtained will be the fitness measure used in the algorithm. The steps are the followings:

1. The graph is randomly divided in 2 communities of the same size.
2. The fitness measure of each vertex is calculated.
3. The node with the lowest score is shifted to the other community.
4. Repeat step 2 until the global modularity of the graph can no longer be improved.
5. Consider the communities obtained as the graph of step 1 and restart from the beginning.

The iteration stops as soon as it no more possible to increase the total modularity. This method represents a good trade-off between accuracy and speed and also overcomes the issue of graph partitioning methods of the predetermined number and size of clusters: in this case the algorithm itself is able to determine both. The only limit lies in dealing with large networks with many communities: this bisection method seems to perform poorly.
Community detection with modularity is by far one of the best methods since it combines well both qualitative and quantitative aspects and it embeds in itself all the ingredients for the issue, from the definition of community to the choice of a null model, to the quantification of strength of a community. These peculiar aspects make it so widely used and common in this field.

However, it is important to mention also its drawbacks. For example, large value of modularity not necessarily indicate that a graph has a community structure. In fact it may happen that random graphs have partitions with high values of modularity or more in general to get not null values of modularity in graphs with no community structure. In addition, partitions associated to high modularity can result to be very different to each other, and since we work with local maxima, it would be difficult which one to choose among all the high valued partitions. There is no guarantee that the partition corresponding to the (hypothetical) global maximum is similar to the high modularity partition chosen.

3 The 2-Choices Dynamics

3.1 Dynamics on Networks

Now that the issue of community detection is more clear and both traditional and more complex methods have been presented, we can take a further step. We can now introduce another interesting community detection technique based on the concept of *dynamics* on networks. With dynamics we refer to simple stochastic processes on graphs aimed to highlight their community structure.
These dynamics on networks are part of the class of *Label Propagation Algorithms* and take their inspiration from epidemic processes: at each time step each node update its state/label according to the state of its neighbours.
The general concept can be described as follows: starting from a graph, an *initialization rule* assigns to each node a label from a finite set of possible states; nodes are then triggered by an *activation rule* and, by interacting with their neighbours, they update their label according to a predefined *update rule*. These rules are invariant with respect to time and network topology, the only inputs they require are the current state of the node and those of its neighbours. The final output is an evolution of the initial graph in which, if the dynamic processes are robust, efficient and well built, the community structure should be more evident and easier to find compared to the initial graph. These methods are usually not able to find the community themselves, they need to be combined with other techniques able to group nodes together, as it was the case for spectral clustering in Section 2.3.
In the following part, we will deeply examine and analyse one of these methods, proposed by Cruciani et al. [2] in 2019 based on a process that comes with the name of *2-Choices dynamics*. Then we will run some simulations on it, comment and compare the results.

3.2 Overview of the Process

In this section we are going to explain in simple words the main steps of the algorithm based on the 2-choices dynamics, while in Section 3.3 we will provide the formal mathematical definitions and proofs.
The algorithm we will illustrate can be applied to any undirected graph. In the proofs'

section[6] we will consider a specific and simplified graph satisfying certain assumptions, that allow us to perform computations. As a consequence, the efficiency of the algorithm is guaranteed only on that type of graph. Anyhow it is a good approximation of any general graph containing sufficiently clear community structure. In the subsequent part we will empirically test the robustness of the algorithm on graphs which are not as regular as the one we will use in the proofs, but that still exhibit a clear community structure.
Bearing this in mind, first of all we need to explain how the algorithm works: first, each node is assigned randomly with a label, which in our case, for the sake of simplicity, will be a color between *red* and *blue*. At each time step, each vertex samples at random two of its neighbours and updates its color according to the following rule: if the two nodes sampled have the same color, the initial vertex adopts it, otherwise it keeps its original one. It clearly is a Markovian process, since the probability of changing color just depends on the state of the network at the previous step. Obviously if the color of the initial node and the one of the two sampled neighbours is the same, nothing happens.
In the proofs' part, we will consider a graph G with specific features, namely to be composed by two communities of equal size and with nodes having all the same degree.
In this context, in the first step of the dynamic process each node of the graph G picks uniformly at random and independently from others, a color between *blue* and *red*. The Theorem 1 states that performing the dynamic process on G, with some constant probability, the graph will reach an almost-clustered configuration in few rounds and that it will keep this configuration for many further rounds with high probability.
Its proof is divided in several steps. We will prove that with a non vanishing probability, the initial distribution of the two colors will be slightly asymmetric with respect to the communities, i.e. in one of them there will be a slight majority of red nodes and of blue ones in the other (Lemma 1), just because of statistical noise. Then, that when the distribution of colors is slightly asymmetric as described above, there exist a significant probability that, as the process evolves, the bias toward the initial slight majority color in each community will become more and more evident (Lemma 2 and Lemma 3), until reaching an *almost-clustered* configuration, meaning that the vast majority of nodes in the first community supports one color, and same for the second community with the other color; furthermore, this configuration will be reached in few rounds of the dynamic process (Lemma 4 and Lemma 5). The final step is to prove that, when the process is in an almost-clustered configuration, it will remain in almost-clustered configurations for many rounds with high probability (Lemma 6).
With this framework, if we let the algorithm evolve in L parallel and independent runs of the 2-choices dynamics, we can obtain, for each vertex, a vector y containing the resulting color after a number of $tmax$ iterations of the dynamic process, for each of the L parallel runs.

[6]Subsection 3.3.2.

We will prove that for all the pairs of nodes but a small number, the vector y looks similar for nodes in the same community and different for nodes belonging to different communities (Theorem 2).
From the initial adjacency matrix, we end up having a matrix Y whose rows are the y vectors for each node of the graph. Coming back to our initial community detection purpose, we can use the output matrix Y as an input for other traditional methods, since the issue can now be seen as a community detection problem on a metric space, like we saw before in Section 2.3. In particular, for our simulations to test the efficacy of the algorithm we used the simple and explained above *k-means*.

3.3 Detailed Analysis

3.3.1 Notation and Assumptions

Now that an overview of the process has been given, it is possible to go more in depth with the formal mathematical aspects of the analysis of the process: we will briefly provide some definitions and assumptions and then continue with the proofs.
First of all, consider a graph G composed by two a-regular communities connected by a b-regular cut, namely two communities V_1 and V_2 of n vertices with a neighbours in their own community, and b in the other one. Let V be the set of nodes and E the set of edges, we define $G = (V, E)$ to be a $(2n, d, b)$-*clustered regular graph*, by meaning that:

- $V = V_1 \cup V_2$, $\quad V_1 \cap V_2 = \emptyset$, $\quad |V_1| = |V_2| = n$;
- each node has degree $d = a + b$;
- each node in V_1 has exactly b neighbours in V_2 and viceversa.

Clearly when $a > b$, the graph G exhibits a well defined community structure.
In our case we assume that $\frac{b}{d} \leq c_1\, n^{-1/2}$, for some positive constant c_1. The latter has to be intended as the maximum number of neighbours that a node can have in the other community for every $\sqrt{n}$ neighbours in its own one.

In the first step of the 2-choices dynamics, each node of G is assigned with a label indicating its state from a set of possible states of size 2. For simplicity the possible states will be denoted by colors: *red* and *blue*.
The vector containing the states of nodes at time t will be called *configuration vector* and it will be denoted by $c^{(t)}$. Thus, the state of a generic node $u \in V_1 \cup V_2$ at time t is expressed by $c_u^{(t)} \in \{red, blue\}$.
The set of blue nodes of the whole G at time t will be called $B^{(t)}$ and the red one $R^{(t)}$. The color sets of each community $i \in \{1, 2\}$ are instead: $B_i^{(t)} = V_i \cap B^{(t)}$ and $R_i^{(t)} = V_i \cap R^{(t)}$. Hence we can further define $s_i^{(t)} = |R_i^{(t)}| - |B_i^{(t)}|$ as the *bias* in the community i toward the

color *red*.
A configuration $c^{(t)}$ will be called *almost-clustered* if $\forall i \in \{1,2\}$ and for $s_1 s_2 < 0$:

$$|s_i| \geq n - O\left(\frac{\log n}{\log \log n}\right) \quad .$$

Which, in other words, means that a configuration has a community with a vast majority of nodes supporting one color, and the other community supporting the other color.

The second technical assumption we are going to work with has a spectral flavour. For this reason it is necessary to introduce some definitions concerning the *spectrum* of the graph G. In particular we will focus on the *transition matrix*, i.e., the square matrix used to represent the transitions of a *simple random walk* on the graph G. Each of its row represents a probability distribution.
Let $P = \frac{1}{d}A^7$ be such a transition matrix. The matrix P can be decomposed as follows:

$$P = \begin{pmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{pmatrix} = \begin{pmatrix} P_{1,1} & 0 \\ 0 & P_{2,2} \end{pmatrix} + \begin{pmatrix} 0 & P_{1,2} \\ P_{2,1} & 0 \end{pmatrix} ,$$

where the first addend is the transition matrix if the communities were disconnected and the second is the transition matrix involving only the edges connecting the two communities. Since the graph G is undirected and regular[8], then $P_{1,2}^{\top} = P_{2,1}$.
Let $\lambda_1 \geq \cdots \geq \lambda_n$ denote the eigenvalues of the transition matrix induced by the first community $\bar{P}_{1,1} := \frac{d}{a} P_{1,1}$ and $v_1, \ldots, v_n$ denote the corresponding orthonormal eigenvectors. Analogously, for the second community, we will have respectively μ and w instead of λ and v, corresponding to eigenvalues and eigenvectors of the transition matrix induced by the second community $\bar{P}_{2,2} := \frac{d}{a} P_{2,2}$.
Since both $\bar{P}_{1,1}$ and $\bar{P}_{2,2}$ are stochastic matrices, by the *Perron-Frobenius theorem* we have that $\lambda_1 = \mu_1 = 1$ and that $v_1 = w_1 = \frac{1}{\sqrt{n}}\mathbf{1}$, where $\mathbf{1}$ is the vector of all ones. Furthermore, since we are assuming that the 2 subgraphs are connected and not bipartite, it holds $\lambda_2 < 1$ and $\lambda_n > -1$, $\mu_2 < 1$ and $\mu_n > -1$.
In our analysis we define $\lambda := \max(|\lambda_2|, |\lambda_n|, |\mu_2|, |\mu_n|)$ and let $\lambda \leq c_2\, n^{-1/4}$, for some positive constant c_2.

We are now able to better understand the 2-choices dynamics: let G be a clustered regular graph and let each node pick a color $c_u^{(0)} \in \{red, blue\}$ uniformly at random and independently from other nodes (this is what we call *initialization rule*). Then the algorithm starts: at each round, each node $u \in V_1 \cup V_2$ chooses uniformly at random two of its neighbours, r and s say. If r and s support the same color, then u updates to their color, otherwise u keeps its original color (the so-called *update rule*). The random sequence of configurations $c^{(t)}$ generated by the iterations of the dynamics on G is a *Markov Chain*, since the configuration

[7] A is still the adjacency matrix
[8] i.e., every vertex has the same degree

at time t just depends on the configuration at time $t-1$.

Before moving on, let us define the constant $h := 4(2\sqrt{2}\,c_1 + c_2^2)$ which will be used to simplify calculations and let us assume that, when needed, without loss of generality, the first community has more nodes supporting the *red* color ($s_1 > 0$), while the second is unbalanced toward color *blue* ($s_2 < 0$).

3.3.2 Proofs

Our aim is to prove that with some constant probability, during the initialization phase, the distributions of the two colors in the communities will be slightly asymmetric, meaning that a community will exhibit a bias toward a color and the other one toward the other color.
When we have a "lucky" initialization, meaning what we just described above, there is a high probability that running the dynamic process, it will make the distributions more and more asymmetric until converging to an almost-clustered configuration, with a clear differentiation of colors between communities.
The above behavior is formalized in:

Theorem 1. Let G be as in Section 3.3.1.
Let us define two events about 2-choices dynamics on G:
$\xi = \{$*Starting from a random initialization, the process reaches an almost-clustered configuration within* $O(\log n)$ *rounds*$\}$.
For any $c \in \mathbb{N}$ fixed constant, define:
$\xi_c : \{$*Starting from an almost-clustered configuration, the process stays in almost-clustered configurations for* n^c *rounds*$\}$.
For two suitable positive constants γ_1 and γ_2 it holds:

$$\mathbb{P}(\xi) \geq \gamma_1 \quad \text{and} \quad \mathbb{P}(\xi_c) \geq 1 - n^{-\gamma_2}.$$

The proof will be divided into six lemmata and they will refer, w.l.o.g., only to the first community, by obvious symmetry reasons.

Lemma 1 (Lucky initialization)**.** Let $G = (V, E)$ be as in Section 3.3.1.
Let each node $u \in V$ choose a color $c_u^{(0)} \in \{red, blue\}$ uniformly at random and independently from the others.
Then it exists a constant γ_1 such that:

$$\mathbb{P}\left(s_1^{(0)} \geq h\sqrt{n} \wedge -s_2^{(0)} \geq h\sqrt{n}\right) \geq \gamma_1\,.$$

Proof. The first step is the initialization phase. Our purpose is to show that there is a

probability bounded away from zero that we are "lucky", namely, that the biases in the two communities are $\Theta(\sqrt{n})$ toward different colors.

The initial bias in the first community is: $s_1^{(0)} = |R_1^{(0)}| - |B_1^{(0)}|$ and it can be seen as a sum of *Rademacher random variables*, i.e., $s_1^{(0)} = \sum_{i \in V_1} X_i$, where $X_i = 1$ if node i is *red* and $X_i = -1$ if node i is *blue*. The mean is equal to 0, the variance equals 1 and third moment equals 1, hence we can exploit the *Barry-Essen theorem*, which states in our case:

$$\left| \mathbb{P}\left(\frac{\sum_{i \in V_1} X_i}{\sqrt{n}} \leq h \right) - \Phi(h) \right| \leq \frac{C}{\sqrt{n}},$$

where Φ is the cumulative distribution function of the standard normal distribution and C is a universal positive constant ($0 < C < 0.4748\ldots$). The Berry-Essen theorem can be read as a more quantitative version of the well known *Central Limit theorem*. Hence:

$$\begin{aligned} \mathbb{P}\left(\sum_{i \in V_1} X_i < 4\left(2\sqrt{2}\, c_1 + c_2^2\right)\sqrt{n} \right) &\leq \Phi\left(4\left(2\sqrt{2}\, c_1 + c_2^2\right)\right) + \frac{C}{\sqrt{n}} \\ &\leq \Phi\left(4\left(2\sqrt{2}\, c_1 + c_2^2\right)\right) + \epsilon \leq \sigma, \end{aligned}$$

where ϵ is a suitably small positive constant and the last inequality holds for a positive constant $\sigma < 1$.
The same inequality holds for $-s_2$ and since both s_1 and s_2 are independent random variables:

$$\mathbb{P}\left(s_1^{(0)} \geq h\sqrt{n}\right) \cdot \mathbb{P}\left(-s_2^{(0)} \geq h\sqrt{n}\right) \geq (1-\alpha)^2 = \gamma_1,$$

where $\gamma_1 > 0$ is a suitable constant. The latter shows the validity of the claim in Lemma 1. □

Lemma 2 (Expected decrease of the minority color)**.** Let $G = (V, E)$ be as in Section 3.3.1. For any configuration $c^{(t)}$ we have:

$$\mathbb{E}\left[|B_1^{(t+1)}| \middle| c^{(t)}\right] < |B_1| \left[1 - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2} - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}}\right)}\right]$$

and

$$\mathbb{E}\left[|R_2^{(t+1)}| \middle| c^{(t)}\right] < |R_2| \left[1 + \frac{s_2}{2n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|R_1|}{n|R_2|} + \frac{2c_1}{\sqrt{n}}\sqrt{\frac{|R_1|}{|R_2|}\left(\frac{1}{2} + \frac{s_2}{2n} + \frac{c_2^2}{\sqrt{n}}\right)}\right].$$

Proof. As mentioned above, w.l.o.g., we consider the case in which the minority color of community 1 is blue.
Conditioning on the configuration at time t, the expected number of nodes supporting the minority color (blue) in community 1 at time $t+1$ can be computed as follows:

- each red node can become blue by picking two blue neighbours;
- each blue node can remain blue by picking at least a blue neighbour.

Therefore, called $B(x)$ the set of neighbors of x having color blue at time t and B_1 the set of blue nodes in community 1 at time t[9], we have

$$\begin{aligned}
\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] &= \sum_{x\in R_1}\left(\frac{|B(x)|}{d}\right)^2 + \sum_{x\in B_1}\left(1-\left(\frac{|R(x)|}{d}\right)^2\right) \\
&= \sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 - \sum_{x\in B_1}\left(\frac{|B(x)|}{d}\right)^2 + \sum_{x\in B_1}\left(1-\left(1-\frac{|B(x)|}{d}\right)^2\right) \\
&= \sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 - \sum_{x\in B_1}\left(\frac{|B(x)|}{d}\right)^2 + \sum_{x\in B_1}\left(1-1+2\frac{|B(x)|}{d}-\left(\frac{|B(x)|}{d}\right)^2\right) \\
&= \sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 + 2\sum_{x\in B_1}\left(\frac{|B(x)|}{d}-\left(\frac{|B(x)|}{d}\right)^2\right) \\
&= \sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 + 2\sum_{x\in B_1}\left(\frac{|B(x)|}{d}\left(1-\frac{|B(x)|}{d}\right)\right).
\end{aligned}$$

Since the function inside the last sum is concave and it satisfies

$$\frac{|B(x)|}{d}\left(1-\frac{|B(x)|}{d}\right) \leq \frac{1}{4},$$

we can conclude that

$$\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] \leq \sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 + \left(\frac{|B_1|}{2}\right).$$

Now our goal is to bound the right hand side of the inequality. Since we already bounded the second term, we now focus on obtaining an upper bound for the first. Start by noting that

$$\begin{aligned}
\sum_{x\in V_1}\left(\frac{|B(x)|}{d}\right)^2 &= \sum_{x\in V_1}\left(\frac{|B_1(x)|}{d}+\frac{|B_2(x)|}{d}\right)^2 \\
&= \sum_{x\in V_1}\left(\frac{|B_1(x)|}{d}\right)^2 + \sum_{x\in V_1}\left(\frac{|B_2(x)|}{d}\right)^2 + 2\sum_{x\in V_1}\frac{|B_1(x)|}{d}\cdot\frac{|B_2(x)|}{d},
\end{aligned}$$

and then bound each one of the three addends, that for ease of presentation, we will call `a1`, `a2`, `a3`.

`a1`: By the assumptions made in Section 3.3.1, since G is $(2n, d, b)$- clustered, the subgraph induced by the first community is a-regular and thus $\bar{P}_{1,1}$ is symmetric and the eigenvectors of $\bar{P}_{1,1}$ form an orthogonal basis of the space. We can thus write the spectral

[9]This notation without the time indicator at the apex to denote the time t, will be used through all proofs.

decomposition of the matrix: $\bar{P}_{1,1} = \sum_{i=1}^{n} \lambda_i v_i v_i^\top$.
Let $\mathbf{1}_{B^{(t)}}$ be the indicator vector of the set $B^{(t)}$:

$$\mathbf{1}_{B^{(t)}}(v) \begin{cases} 1 & \text{if } v \in B^{(t)}\,, \\ 0 & \text{if } v \notin B^{(t)}\,. \end{cases}$$

This vector can be rewritten as a linear combination of the eigenvectors of $\bar{P}_{1,1}$, i.e. $\mathbf{1}_{B_1}^{(t)} = \sum_{i=1}^{n} \alpha_i v_i$, where $\alpha_i = \langle v_i\,, \mathbf{1}_{B_1} \rangle$. We can write:

$$\begin{aligned}
\sum_{x \in V_1} \left(\frac{|B_1(x)|}{d} \right)^2 &= (\,||P_{1,1}\,\mathbf{1}_{B_1}||_2\,)^2 = \mathbf{1}_{B_1}^\top\, P_{1,1}^\top \cdot P_{1,1}\,\mathbf{1}_{B_1} \\
&= \mathbf{1}_{B_1}^\top\, P_{1,1}^2\,\mathbf{1}_{B_1} = \frac{a^2}{d^2}\,\mathbf{1}_{B_1}^\top\, \bar{P}_{1,1}^2\,\mathbf{1}_{B_1} \\
&\leq \mathbf{1}_{B_1}^\top\, \bar{P}_{1,1}^2\,\mathbf{1}_{B_1} = \mathbf{1}_{B_1}^\top \cdot \sum_{i=1}^{n} \lambda_i^2\, v_i\, v_i^\top \cdot \sum_{i=1}^{n} \alpha_i v_i \\
&= \mathbf{1}_{B_1}^\top \cdot \sum_{i=1}^{n} \lambda_i^2 \alpha_i v_i \\
&= \mathbf{1}_{B_1}^\top \left(\lambda_1^2 \alpha_1 v_1 + \sum_{i=2}^{n} \lambda_i^2 \alpha_i v_i \right) \\
&\leq \mathbf{1}_{B_1}^\top \left(\lambda_1^2 \alpha_1 v_1 + \lambda^2 \sum_{i=2}^{n} \alpha_i v_i \right) \\
&\leq \mathbf{1}_{B_1}^\top \left(\lambda_1^2 \alpha_1 v_1 + \lambda^2 \sum_{i=1}^{n} \alpha_i v_i \right) \\
&= \mathbf{1}_{B_1}^\top \left(\lambda_1^2 \alpha_1 v_1 + \lambda^2 \mathbf{1}_{B_1} \right) = \frac{|B_1|^2}{n} + \lambda^2 |B_1|\,.
\end{aligned}$$

a2: The second quantity can be bounded using *Cauchy-Schwarz inequality*:

$$\begin{aligned}
\sum_{x \in V_1} \left(\frac{|B_2(x)|}{d} \right)^2 &= (\,||P_{1,2}\,\mathbf{1}_{B_2}||_2\,)^2 \leq (\,||P_{1,2}||_2 \cdot ||\mathbf{1}_{B_2}||_2\,)^2 \\
&= \left(||P_{1,2}||_2\, \sqrt{|B_2|} \right)^2 \leq \left(\sqrt{||P_{1,2}||_1 \cdot ||P_{1,2}||_\infty}\, \sqrt{|B_2|} \right)^2\,,
\end{aligned}$$

where in the last passage, we exploited *Hölder's inequality*. Since each node has exactly b neighbours in the other community:

$$||P_{1,2}||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |P_{1,2}(i,j)| = \frac{b}{d}\,,$$

and, by the symmetry of $P_{1,2}$:

$$||P_{1,2}||_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |P_{1,2}(i,j)| = \frac{b}{d}\,.$$

Thus we can write:

$$\left(\sqrt{||P_{1,2}||_1 \cdot ||P_{1,2}||_\infty}\,\sqrt{|B_2|}\right)^2 = \left(\frac{b}{d}\,\sqrt{|B_2|}\right)^2 = \frac{b^2}{d^2}\,|B_2|\,.$$

a3: The third and last term can be easily bounded by the previous results:

$$\begin{aligned} 2\sum_{x\in V_1} \frac{|B_1(x)|}{d}\frac{|B_2(x)|}{d} &= 2\,||P_{1,1}\,b_1||_2 \cdot ||P_{1,2}\,b_2||_2 \\ &\le 2\,\frac{b}{d}\,\sqrt{|B_2|\left(\frac{|B_1|^2}{n} + \lambda^2|B_1|\right)}\,. \end{aligned}$$

Combining all the results we get:

$$\begin{aligned} \mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] &\le \frac{|B_1|^2}{n} + \lambda^2|B_1| + \frac{b^2}{d^2}\,|B_2| + 2\,\frac{b}{d}\,\sqrt{|B_2|\left(\frac{|B_1|^2}{n} + \lambda^2|B_1|\right)} + \left(\frac{B_1}{2}\right) \\ &= \frac{|B_1|^2}{n} + \lambda^2|B_1| + \frac{b^2}{d^2}\,|B_2| + 2\,\frac{b}{d}\,\sqrt{|B_1|\,|B_2|\,\left(\frac{|B_1|}{n} + \lambda^2\right)} + \left(\frac{B_1}{2}\right)\,. \end{aligned}$$

Recall that the assumptions in Section 3.3.1 ensure that $\frac{b}{d} \le c_1\,n^{-1/2}$ and $\lambda \le c_2\,n^{-1/4}$. Substituting:

$$\begin{aligned} &\le \frac{|B_1|^2}{n} + c_2^2\,\frac{|B_1|}{\sqrt{n}} + c_1^2\,\frac{|B_2|}{n} + \frac{2c_1}{\sqrt{n}}\,\sqrt{|B_1|\,|B_2|\,\left(\frac{|B_1|}{n} + \frac{c_2^2}{\sqrt{n}}\right)} + \left(\frac{B_1}{2}\right) \\ &= |B_1|\left(\frac{|B_1|}{n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\,\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{|B_1|}{n} + \frac{c_2^2}{\sqrt{n}}\right)} + \frac{1}{2}\right)\,. \end{aligned}$$

We can rewrite $\frac{|B_1|}{n}$ as: $\frac{1}{2} - \frac{s_1}{2n}$, thus:

$$\begin{aligned} &= |B_1|\left(\frac{1}{2} - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\,\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2} - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}}\right)} + \frac{1}{2}\right) \\ &= |B_1|\left(1 - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\,\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2} - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}}\right)}\right)\,. \end{aligned}$$

We finally get:

$$\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] < |B_1|\left[1-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{c_1^2|B_2|}{n|B_1|}+\frac{2c_1}{\sqrt{n}}\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2}-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}\right)}\right],$$

for community 1, and analogously for community 2:

$$\mathbb{E}\left[|R_2^{(t+1)}|\Big|c^{(t)}\right] < |R_2|\left[1+\frac{s_2}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{c_1^2|R_1|}{n|R_2|}+\frac{2c_1}{\sqrt{n}}\sqrt{\frac{|R_1|}{|R_2|}\left(\frac{1}{2}+\frac{s_2}{2n}+\frac{c_2^2}{\sqrt{n}}\right)}\right].$$

□

Lemma 3 (Multiplicative growth of the bias)**.** Let $c^{(t)}$ be a configuration such that $h\sqrt{n} \le s_1 \le \frac{n}{2}$ and $h\sqrt{n} \le -s_2 \le \frac{n}{2}$.
It holds

$$\mathbb{P}\left(s_1^{(t+1)} \ge \left(1+\frac{1}{32}\right)s_1 \,\Big|\, c^{(t)}\right) \ge 1-\exp\left(-2\frac{s_1^2}{64^2}n\right)$$

and

$$\mathbb{P}\left(s_2^{(t+1)} \le \left(1+\frac{1}{32}\right)s_2 \,\Big|\, c^{(t)}\right) \ge 1-\exp\left(-2\frac{s_2^2}{64^2}n\right).$$

Proof. By imposing $s_1 \le \frac{n}{2}$, we can rearrange the terms and write:

$$s_1 = |R_1| - |B_1| \le \frac{n}{2} \tag{3.1}$$

and by adding and subtracting $|B_1|$ we get

$$n - 2|B_1| \le \frac{n}{2}$$

which implies

$$|B_1| \ge \frac{n}{4}.$$

Similarly, by the same trick we used above, we can rewrite $|B_1|$ as:

$$|B_1| = \frac{n-s_1}{2}.$$

Since s_1 is positive by Lemma 1, we can bound $|B_1|$:

$$\frac{n}{4} \le |B_1| = \frac{n-s_1}{2} \le \frac{n}{2}. \tag{3.2}$$

Furthermore, since $|B_1| \ge \frac{n}{4}$ and $|B_2| \le n$, we get:

$$\frac{|B_2|}{|B_1|} \le 4. \tag{3.3}$$

By plugging (3.2) and (3.3) into the last display of Lemma 2, we obtain:

$$\mathbb{E}\left[|B_1^{(t+1)}|\middle|c^{(t)}\right] < |B_1|\left[1-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{c_1^2|B_2|}{n|B_1|}+\frac{2c_1}{\sqrt{n}}\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2}-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}\right)}\right]$$
$$\mathbb{E}\left[|B_1^{(t+1)}|\middle|c^{(t)}\right] < |B_1|\left[1-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{4c_1^2}{n}+\frac{2c_1}{\sqrt{n}}\sqrt{4\left(\frac{1}{2}-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}\right)}\right].$$

By Lemma 1, we got $s_1 \geq h\sqrt{n}$, hence:

$$\begin{aligned} s_1 &\geq h\sqrt{n} \\ s_1 &\geq \sqrt{n}\left[4\left(2\sqrt{2}c_1+c_2^2\right)\right] \\ s_1 &\geq \sqrt{n}\left(8\sqrt{2}c_1+4c_2^2\right) \\ \frac{s_1}{2n} &\geq \frac{4\sqrt{2}c_1}{\sqrt{n}}+\frac{2c_2^2}{\sqrt{n}}. \end{aligned}$$

Therefore:

$$\frac{s_1}{2n} \geq \frac{c_2^2}{\sqrt{n}}. \tag{3.4}$$

Substituting:

$$\begin{aligned} \mathbb{E}\left[|B_1^{(t+1)}|\middle|c^{(t)}\right] &< |B_1|\left(1-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{4c_1^2}{n}+\frac{2c_1}{\sqrt{n}}\sqrt{4\left(\frac{1}{2}-\frac{s_1}{2n}+\frac{s_1}{2n}\right)}\right) \\ &= |B_1|\left(1-\frac{s_1}{2n}+\frac{c_2^2}{\sqrt{n}}+\frac{4c_1^2}{n}+\frac{2c_1}{\sqrt{n}}\sqrt{4\cdot\frac{1}{2}}\right) \\ &= |B_1|\left(1-\frac{s_1}{2n}+\frac{4c_1^2}{n}+\frac{2\sqrt{2}c_1}{\sqrt{n}}+\frac{c_2^2}{\sqrt{n}}\right) \\ &\leq |B_1|\left(1-\frac{s_1}{2n}+\frac{4c_1^2}{n}+\frac{s_1}{4n}\right). \end{aligned}$$

Since $\frac{4c_1^2}{n} = O\left(\frac{1}{n}\right)$ and by Equation (3.4): $\frac{s_1}{n} = \Omega\left(\frac{1}{\sqrt{n}}\right)$, for sure, if n is sufficiently large: $\frac{4c_1^2}{n} \leq \frac{s_1}{8n}$. Hence:

$$\mathbb{E}\left[|B_1^{(t+1)}|\middle|c^{(t)}\right] < |B_1|\left(1-\frac{s_1}{8n}\right).$$

Now we exploit the *additive form of Chernoff Bound* and that $|B_1| \geq \frac{n}{4}$, we get:

$$\begin{aligned}
\mathbb{P}\left(|B_1^{(t+1)}| > |B_1|\left(1-\frac{s_1}{16n}\right) \,\middle|\, c^{(t)}\right) &= \mathbb{P}\left(|B_1^{(t+1)}| > |B_1|\left(1-\frac{s_1}{8n}\right) + \frac{s_1|B_1|}{16n} \,\middle|\, c^{(t)}\right) \\
&\leq \mathbb{P}\left(|B_1^{(t+1)}| > |B_1|\left(1-\frac{s_1}{8n}\right) + \frac{s_1}{64} \,\middle|\, c^{(t)}\right) \\
&\leq \mathbb{P}\left(|B_1^{(t+1)}| > \mathbb{E}\left[|B_1^{(t+1)}| \,\middle|\, c^{(t)}\right] + \frac{s_1}{64} \,\middle|\, c^{(t)}\right) \\
&\leq \exp\left(-2\frac{s_1^2}{64^2}n\right).
\end{aligned} \tag{3.5}$$

Recalling that $s_1 = n - 2|B_1|$ and that $s_1 \leq \frac{n}{2}$, we can conclude that with probability $1 - \exp\left(-2\frac{s_1^2}{64^2}n\right)$ it holds:

$$\begin{aligned}
s^{(t+1)} \geq n - 2|B_1|\left(1-\frac{s_1}{16n}\right) &= n - (n-s_1)\left(1-\frac{s_1}{16n}\right) \\
&= n - n\left(1-\frac{s_1}{16n}\right) + s_1\left(1-\frac{s_1}{16n}\right) \\
&= \frac{s_1}{16} + s_1 - \frac{s_1^2}{16n} \\
&\geq \frac{s_1}{16} + s_1 - \frac{s_1}{32} = s_1\left(1+\frac{1}{32}\right).
\end{aligned}$$

In other words, the latter means that with high probability the bias toward the majority color will increase by a multiplicative factor from t to $t+1$. □

We now show that, with a non-vanishing probability, after a lucky initialization (see Lemma 1), the process will within few rounds will increase this biased by a logarithmic factor.

Lemma 4 (Clustering - symmetry breaking)**.** Starting from an initial configuration as in Section 3.3.1, it holds that: with non-vanishing probability, within $O(\log\log n)$ rounds, the process reaches a configuration $c^{(t)}$ such that:

$$s_1^{(t)} \geq \sqrt{n}\log n \quad \text{and} \quad -s_2^{(t)} \geq \sqrt{n}\log n\,.$$

Proof. Let I be the event "the initial configuration in such that $s_1^{(0)} \geq h\sqrt{n}$ and $-s_2^{(0)} \geq h\sqrt{n}$." In Lemma 1 we proved that I happens with a non-vanishing probability. Then from that configuration, by iterating the application of Lemma 3, we want to show that s_1 (and $-s_2$) becomes greater than $\sqrt{n}\log n$ within $O(\log\log n)$ rounds with non-vanishing probability.

Let us define a round t *to be successful* for community 1 if one of the following holds:

- the process is in a configuration in which the bias is multiplicatively increasing but is not yet large enough, namely $s_1^{(t)} \geq s^{(t-1)}\left(1+\frac{1}{32}\right)$ and $s_1^{(t-1)} < \sqrt{n}\log n$;
- the bias is already large enough, namely it exists $t' < t$ such that $s_1^{(t')} \geq \sqrt{n}\log n$.

Let $\alpha = 2(h/64)^2$ and $\beta = (1+1/32)$ and let us define two events:

- $R_i^{(t)}$: the round t is successful with respect to community i.
- K_i: the first $\log_\beta \log n$ rounds are successful with respect to community i.

Note that after T consecutive successful rounds, the process reaches a configuration $\bar{c}$ such that $\bar{s_1} \geq h\sqrt{n}(1+1/32)^T$ and that the probability that also the following round is successful is at least $1 - \exp\left[-2h^2(1+1/32)^T\right]$.

We can write the probability to have $\log_\beta \log n$ successful rounds, conditional to the event I as

$$\begin{aligned}
\mathbb{P}\left(K_1 \mid I\right) &= \mathbb{P}\left(\bigcap_{i=1}^{\log_\beta \log n} R_1^{(i)} \,\middle|\, I\right) \\
&= \prod_{i=1}^{\log_\beta \log n} \mathbb{P}\left(R_1^{(i)} \,\middle|\, \bigcap_{j=1}^{i-1} R_1^{(j)},\, I\right) \\
\text{Lemma 3} \Rightarrow \quad &\geq \prod_{i=1}^{\log_\beta \log n} \left(1 - \exp\left[-2h^2 \frac{\left(1+\frac{1}{32}\right)^{2i}}{64^2}\right]\right) \\
&= \prod_{i=1}^{\log_\beta \log n} \left(1 - e^{-\alpha\beta^{2i}}\right) \\
&= \exp\left[\log\left(\prod_{i=1}^{\log_\beta \log n} \left(1 - e^{-\alpha\beta^{2i}}\right)\right)\right] \\
&= \exp\left[\sum_{i=1}^{\log_\beta \log n} \log\left(1 - e^{-\alpha\beta^{2i}}\right)\right].
\end{aligned}$$

Since $\left(1 - e^{-\alpha\beta^{2i}}\right) < 1$, each addend of the sum will be negative. Thus we can bound the probability:

$$\mathbb{P}\left(K_1 \mid I\right) > \exp\left[\sum_{i=1}^{\infty} \log\left(1 - e^{-\alpha\beta^{2i}}\right)\right].$$

Then, by expanding using the *Taylor series* and the fact that[10] $e^{-\alpha\beta^{2i}} < \alpha^{-1}\beta^{-2i} \Rightarrow \alpha\beta^{2i}$, we get:

$$\begin{aligned}
\mathbb{P}\left(K_1 \mid I\right) &> \exp\left[-\sum_{i=1}^{\infty}\left(e^{-\alpha\beta^{2i}} + O\left(e^{-2\alpha\beta^{2i}}\right)\right)\right] \\
&> \exp\left[-\frac{1}{\alpha}\sum_{i=1}^{\infty}\left(\beta^{-2i} + o\left(\beta^{-2i}\right)\right)\right] \\
&= \exp\left[-\frac{1}{\alpha}\left(\frac{1}{1-\beta^{-2}}\right)(1+C)\right] = e^{-\mu}.
\end{aligned}$$

The term C is a constant deriving from smaller order terms coming from the Taylor approx-

[10] This inequality is always true in our case since we always have $\alpha\beta^{2i} > 0$.

imation.
The result means that the bias s_1 reaches a value of at least $\sqrt{n}\log n$ within $O(\log\log n)$ rounds with a probability at least $e^{-\mu}$.
In a completely symmetric fashion, the same holds for community 2. Now we can compute the probability that both K_1 and K_2 happen, still conditioning on I. Note that the two events are independent:

$$\begin{aligned}
\mathbb{P}\left(K_1, K_2 \,|\, I\right) &= \mathbb{P}\left(\bigcap_{i=1}^{\log_\beta \log n} R_1^{(i)} \cap R_2^{(i)} \,\Big|\, I\right) \\
&= \prod_{i=1}^{\log_\beta \log n} \mathbb{P}\left(R_1^{(i)} \cap R_2^{(i)} \,\Big|\, \bigcap_{j=1}^{i-1} R_1^{(j)} \cap R_2^{(i)},\, I\right) \\
&= \prod_{i=1}^{\log_\beta \log n} \mathbb{P}\left(R_1^{(i)} \Big|\, \bigcap_{j=1}^{i-1} R_1^{(j)} \cap R_2^{(j)}, I\right) \cdot \prod_{i=1}^{\log_\beta \log n} \mathbb{P}\left(R_2^{(i)} \Big|\, \bigcap_{j=1}^{i-1} R_1^{(j)} \cap R_2^{(j)}, I\right) \\
&= \mathbb{P}\left(\bigcap_{i=1}^{\log_\beta \log n} R_1^{(i)} \,|\, I\right) \cdot \mathbb{P}\left(\bigcap_{i=1}^{\log_\beta \log n} R_2^{(i)} \,|\, I\right) \\
&\geq e^{-2\mu} = \gamma_3 \,,
\end{aligned}$$

where γ_3 represents the non-vanishing probability that both the biases $|s_1|$ and $|s_2|$ in the two communities will reach a value of at least $\sqrt{n}\log n$ within $O(\log\log n)$ rounds. □

Lemma 5 (Convergence)**.** Starting from a configuration $c^{(t)}$ such that $|s_i| \geq \sqrt{n}\log n$, for each $i \in \{1,2\}$, with high probability, there exist two rounds $\tau_1, \tau_2 = O(\log n)$ such that:

$$|s_1^{(\tau_1)}| \geq n - \log n \quad \text{and} \quad |s_2^{(\tau_2)}| \geq n - \log n \,.$$

Proof. The goal of this lemma is to prove that both the biases will continue to increase and will preserve their sign, up to a configuration in which the minority color nodes in each community have at most a logarithmic size. As for Lemma 4, we first focus on one of the biases and then we show that the result holds for both of them.
Using Lemma 3 and the fact that $s_i \geq \sqrt{n}\log n$ we get:

$$\begin{aligned}
\mathbb{P}\left(s_i^{(t+1)} \geq \left(1+\frac{1}{16}\right) s_i \,\Big|\, s_i \geq \sqrt{n}\log n\right) &\geq 1 - \exp\left(-2\frac{s_1^2}{32^2} n\right) \\
&\geq 1 - \exp\left[-\Omega\left(n^2 \log^2 n\right)\right] > 1 - n^{-g_1},
\end{aligned}$$

for any positive constant g_1.
Let τ_i' be the first round such that $s_i \geq \frac{n}{2}$, for any $i \in \{1,2\}$. By iteratively applying the union bound, we get that with high probability we have $O(\log n)$ consecutive rounds of

multiplicative growth (Lemma 3) and thus it holds:

$$\mathbb{P}\left(\tau_i' > f_2 \log n\right) < n^{-g_2},$$

for two suitable positive constants f_2 and g_2.
Thanks to Lemma 2, we know that the expected number of blue nodes in the first community is:

$$\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] < |B_1|\left[1 - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\sqrt{\frac{|B_2|}{|B_1|}\left(\frac{1}{2} - \frac{s_1}{2n} + \frac{c_2^2}{\sqrt{n}}\right)}\right].$$

Since $s_1 \geq \frac{n}{2}$, it implies $\frac{s_1}{2n} \geq \frac{1}{4}$, and since $\frac{s_1}{2n} \geq \frac{c_2^2}{\sqrt{n}}$ from (3.4), we can rewrite:

$$\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] < |B_1|\left(1 - \frac{1}{4} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\sqrt{\frac{|B_2|}{|B_1|}\cdot\frac{1}{2}}\right),$$

assuming $\log n \leq |B_1| \leq \frac{n}{4}$ and $|B_2| < n$:

$$\begin{aligned}
\mathbb{E}\left[|B_1^{(t+1)}|\Big|c^{(t)}\right] &\leq |B_1|\left(1 - \frac{1}{4} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{n}}\sqrt{\frac{n}{2\log n}\cdot\frac{1}{2}}\right) \\
&= |B_1|\left(1 - \frac{1}{4} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2|B_2|}{n|B_1|} + \frac{2c_1}{\sqrt{4\log n}}\right) \\
&\leq |B_1|\left(1 - \frac{1}{4} + \frac{c_2^2}{\sqrt{n}} + \frac{c_1^2}{\log n} + \frac{c_1}{\sqrt{\log n}}\right) \\
&\leq |B_1|\left(1 - \frac{1}{5}\right).
\end{aligned}$$

Using a *multiplicative form of Chernoof Bound* [14], we get:

$$\begin{aligned}
\mathbb{P}\left[|B_1^{(t+1)}| \geq \left(1 - \frac{1}{25}\right)|B_1|\,\Big|\,c^{(t)}\right] &= \mathbb{P}\left[|B_1^{(t+1)}| \geq \left(1 + \frac{1}{5}\right)\left(1 - \frac{1}{5}\right)|B_1|\,\Big|\,c^{(t)}\right] \\
&\leq \exp\left[-\frac{1}{75}\left(1 - \frac{1}{5}\right)|B_1|\right] = n^{-\gamma_4},
\end{aligned}$$

for a positive constant γ_4.
Now, let τ_i'' be the first round such that $s_i \geq n - \log n$, starting from a configuration $s_1 \geq \frac{n}{2}$, again by a union bound:

$$\mathbb{P}\left(\tau_i'' > f_3 \log n\right) < n^{-g_3},$$

for two suitable positive constants f_3 and g_3.
Finally, let τ_i be the first round such that $s_i \geq n - \log n$ starting from a configuration such that $s_i \geq \sqrt{n}\log n$, we get:

$$\mathbb{P}\left[\tau_1 > (f_2 + f_3)\log n \bigcup \tau_2 > (f_2 + f_3)\log n\right] <$$

$$< \mathbb{P}\left(\tau_1^{'} > f_2 \log n \bigcup \tau_1^{''} > f_3 \log n \bigcup \tau_2^{'} > f_2 \log n \bigcup \tau_2^{''} > f_3 \log n\right) <$$
$$< \mathbb{P}\left(\tau_1^{'} > f_2 \log n\right) + \mathbb{P}\left(\tau_1^{''} > f_3 \log n\right) + \mathbb{P}\left(\tau_2^{'} > f_2 \log n\right) + \mathbb{P}\left(\tau_2^{''} > f_3 \log n\right) <$$
$$< n^{-g_4}.$$

for a suitable positive constant g_4.
This last display means that the number of rounds needed to obtain a value for both biases of $n - \log n$, is with high probability smaller than logarithmic. Furthermore, a bias of this size implies that the vast majority of nodes in each community support one color. □

Lemma 6 (Metastability)**.** Let $\delta \in \mathbb{N}$ be any constant. Starting from a configuration $c^{(t)}$ such that $|s_i| \geq n - \log n$ for each $i \in \{1, 2\}$, for the next n^δ rounds the process lies in the set of configurations such that:

$$|s_i| \geq n - O\left(\frac{\log n}{\log \log n}\right)$$

and the sign of the bias is preserved with high probability.

Proof. This last lemma tells us that once the process reaches a configuration such that each community has most of its vertices of the same color, then it will remain stably in this situation for many rounds. This means that even if few nodes in each community will continue to change color in favour of the minority one, they will not be enough to affect the majority color for many rounds.
Let X_u be an indicator random variable such that:

$$X_u = \begin{cases} 1 & \text{if } u \text{ supports the minority color at the next round}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Let p_u be the probability of having $X_u = 1$, namely the probability that the node u will support the minority color in the next round.
By *Le Cam's Theorem*, we can approximate $\sum_{u \in V} X_u$ with a Poisson random variable. More precisely, if $\sum_{u \in V_i} p_u^2 \leq \frac{1}{n^\epsilon}$, for some positive constant ϵ, then $\sum_{u \in V} X_u$ is well approximated by a Poisson random variable of mean $\sum_{u \in V_i} p_u$.
In order to exploit the Theorem, we need to prove $\sum_{u \in V_i} p_u^2 \leq \frac{1}{n^\epsilon}$. In particular, we need to show that:

$$\sum_{u \in V_i} p_u^2 = O\left(\frac{\log^5 n}{n}\right).$$

Let σ_i^- be the set of nodes supporting the minority color in community i and σ_i^+ the set supporting the majority one. Let then z_u be the number of neighbours of node u, that support the minority color of community i, i.e., those belonging to σ_i^- and that, at the same time, belong to the same community of node u.

Note that $|\sigma_i^-| \leq \log n$ and $|\sigma_i^+| \geq n - \log n$. It is possible to write:

$$\sum_{u \in V_i} p_u^2 = \sum_{u \in \sigma_i^+} p_u^2 + \sum_{u \in \sigma_i^-} p_u^2 \,. \tag{3.6}$$

Let us analyse the two addends separately. Regarding the first, it represents the sum of probabilities for each node supporting the majority color of its own community, to change its state in the next round. It can be bounded as follows:

$$\sum_{u \in \sigma_i^+} p_u^2 \leq \sum_{u \in \sigma_i^+} \left(\frac{z_u + b}{a + b} \right)^4 ,$$

since, for a node supporting the majority color, in order to change state, it needs to select 2 neighbours of its opposite color or from the set σ_i^- or from the other community. It follows:

$$\begin{aligned} \sum_{u \in \sigma_i^+} \left(\frac{z_u + b}{a + b} \right)^4 &\leq n \left(\frac{\log n + b}{a} \right)^4 \\ &\leq c_3 \, n \left(\frac{\log n}{\sqrt{n}} + \frac{b}{\sqrt{n}} \right)^4 \\ &\leq O\left(\frac{\log^5(n)}{n} \right), \end{aligned}$$

for some suitable constant c_3. In the second inequality we used that:

$$\frac{b}{a+b} \leq \frac{c_1}{\sqrt{n}} \Rightarrow \frac{a}{b} + 1 \geq \frac{\sqrt{n}}{c_1} \Rightarrow a \geq b \left(\frac{\sqrt{n}}{c_1} - 1 \right), \tag{3.7}$$

and since $b \geq 1$ we finally get:

$$a \geq c_3' \sqrt{n} \,,$$

for some constant c_3'.

The second term, instead, is the sum of probabilities for each node of the minority color, to keep its state also in the next round. This means to choose at least one neighbour that supports its own color:

$$\begin{aligned} \sum_{u \in \sigma_i^-} p_u^2 &\leq \sum_{u \in \sigma_i^-} \left(\frac{z_u + b}{a + b} \right)^2 \\ &\leq \sum_{u \in \sigma_i^-} \left(\frac{\log n + b}{a + b} \right)^2 = \log n \left(\frac{\log n + b}{a + b} \right)^2 \\ &\leq \frac{\log^3 n}{a^2} + \frac{2b \log^2 n}{a^2} + \frac{b^2 \log n}{a^2} \\ &\leq \frac{\log^3 n}{n} + \frac{2b \log^2 n}{n} + \frac{b^2 \log n}{n} \end{aligned}$$

$$= O\left(\frac{\log^3 n}{n}\right).$$

By combining the two terms, we get:

$$\sum_{u \in V_i} p_u^2 = O\left(\frac{\log^5(n)}{n}\right) + O\left(\frac{\log^3 n}{n}\right) = O\left(\frac{\log^5(n)}{n}\right).$$

This latter result allows us to approximate our random variable X_u with a Poisson with mean $\sum_{u \in V_i} p_u$.
We now recall the results for a Poisson random variable. Let $X \sim Poisson(\eta)$, where η is a positive real number such that:

$$\mathbb{P}(X = i) = \frac{\eta^i}{i!} e^{-\eta}.$$

It holds that if $g = c_4 \frac{\log n}{\log \log n}$, for some $c_4 > 0$, and η be constant with respect to n, then:

$$\mathbb{P}(X > g) \leq n^{-c_4 + o(1)}.$$

Applying this last result to our initial random variable, we get that with high probability, the number of nodes supporting the minority color in the next round will be really small, i.e., for every $\delta > 0$

$$\mathbb{P}\left(\forall t' \in [t, t + n^\delta],\ |s_i^{(t')}| \geq n - O\left(\frac{\log(n)}{\log\log(n)}\right) \mid c^{(t)}\right) = 1 - o(1).$$

In order to exploit the above result we just need to prove that η is constant with respect to n.
In our case η is represented by $\sum_{u \in V_i} p_u$, thus we can rewrite:

$$\eta = \sum_{u \in V_i} p_u = \sum_{u \in \sigma_i^+} p_u + \sum_{u \in \sigma_i^-} p_u\,,$$

using the same logic we used in (3.6). Again we analyse the two addends separately. The first can be written as follows, since at most a nodes can have all the $\log n$ nodes belonging to σ_i^- as neighbours:

$$\begin{aligned}
\sum_{u \in \sigma_i^+} p_u &\leq \sum_{u \in \sigma_i^+} \left(\frac{z_u + b}{a + b}\right)^2 \\
&\leq a \log(n) \left(\frac{\log n + b}{a + b}\right)^2 + (n - a \log n) \left(\frac{b}{a + b}\right)^2 \\
&= \frac{a \log n (\log^2 n + b^2 + 2b \log n) + (n - a \log n) b^2}{(a + b)^2}
\end{aligned}$$

By (3.7) we know that $a \geq c_3'\sqrt{n}$; furthermore, we can obtain $\frac{b}{a} \leq \frac{c_1}{\sqrt{n}}$, hence for some constant c_5' we have $\frac{b}{a} \leq \frac{c_5'}{\sqrt{n}}$. It follows that the quantity in the latter display is $O(1)$. The second term instead:

$$\begin{aligned}\sum_{u\in\sigma_i^-} p_u &\leq \sum_{u\in\sigma_i^-}\left(\frac{z_u+b}{a+b}\right) \leq \sum_{u\in\sigma_i^-}\left(\frac{\log n+b}{a+b}\right)\\ &\leq \log n\left(\frac{\log n+b}{a+b}\right) \leq \frac{\log^2 n}{a}+\frac{b\log n}{a}\\ &\leq \frac{\log^2 n}{c_3\sqrt{n}}+\frac{c_5\log n}{\sqrt{n}} = o(1)\,.\end{aligned}$$

Combining:

$$\eta = \sum_{u\in V_i} p_u = \sum_{u\in\sigma_i^+} p_u + \sum_{u\in\sigma_i^-} p_u = O(1)\,.$$

Since we showed that η does not diverge with n, we can fully exploit *Le Cam's Theorem*, and apply the result we got on the Poisson random variable. □

Here we conclude the proofs for Theorem 1, thus we can now continue with the part concerning Theorem 2. The goal of this second part is to obtain a model for an algorithm that works with high probability with the results from the previous part.
Note that in order to get an *almost-clustered* configuration, we need a lucky initialization (Lemma 1), but it exists also a constant probability that the two communities will converge to the same color and following the steps from Lemma 2 to Lemma 6, we will end up with a graph having all nodes, but a small number, supporting one color. As a consequence, we need an algorithm able to perform the task even in this case.

Theorem 2. Let G be as in Section 3.3.1.
Consider $L = \Theta(\log n)$ independent parallel runs of 2-choices dynamics after $tmax = \Theta(\log n)$ rounds (for each $\ell \in L$) over G.
Let $y_v(\ell)$ be the color associated to vertex v in the ℓ-th parallel run after $tmax$ iterations of the 2-choices dynamics.
Let y_v be the vector of length L associated to v whose elements are $y_v(\ell)\,\forall\,\ell \in L$. We have:

$$\forall\, v \in V_1,\ w \in V_1 \Rightarrow \sum_{\ell=1}^{L}\Big|y_v(\ell) - y_w(\ell)\Big| = O(1),$$

while

$$\forall\, v \in V_1,\ w \in V_2 \Rightarrow \sum_{\ell=1}^{L}\Big|y_v(\ell) - y_w(\ell)\Big| = \Omega(L).$$

Lemma 7. Starting from any initial configuration, within $O(\log n)$ rounds the system reaches a configuration such that with high probability:

$$|s_1^{(t)}| \geq \sqrt{n}\log n \text{ and } |s_2^{(t)}| \geq \sqrt{n}\log n\,.$$

Proof. Our interest is to bound the times τ_3 and τ_4 defined as the first round such that respectively $|s_1^{(t)}| \geq \sqrt{n}\log n$ and $|s_2^{(t)}| \geq \sqrt{n}\log n$.
To do so, we use a general tool for *Markov chains* [15]. Let $\{Y_t\}_{t\in\mathbb{N}}$ be a Markov chain, Ω be the configuration space of the process and $\sqrt{n}\log n$ the target value. In order to exploit the tool, we need to satisfy two conditions:

- For any positive constant m, there exist a positive constant $c_6 < 1$ such that for every $\omega \in \Omega : s_i < \sqrt{n}\log n$ we have:

$$\mathbb{P}\left(s_i^{(t+1)} < m\sqrt{n} \,\middle|\, Y_t = \omega\right) < c_6\,.$$

- There exist two positive constants ι and c_7 such that for every $\omega \in \Omega : s_i < \sqrt{n}\log n$ we have:

$$\mathbb{P}\left(s_i^{(t+1)} < (1+\iota)\, s_i \,\middle|\, Y_t = \omega\right) < \exp\left[-c_7\, s_i^2\, n\right]\,.$$

The proof for the first condition is analogous to the one of Lemma 1, while the second equals the proof of Lemma 3 with $\iota = \frac{1}{16}$ and $c_7 = \frac{2}{32^2}$.
This is enough to conclude that the process reaches with high probability a configuration ω in which $|s_i| \geq \sqrt{n}\log n$ within $O(\log n)$ rounds. □

With this Theorem 2, we end up having for each vertex v a vector y_v of length L, resulting from the L parallel and independent runs after $tmax$ iterations of the 2-choices dynamics. If two of these vectors are equal in their entries, but a small number of outliers, the two associated nodes belong to the same community. Note that even if we are not lucky in the initialization phase, we should still be able to get significant results, since we can count on a high number of independent runs.

4 Simulations

4.1 Tests on Computer-Generated Networks

Basing on the theory described in the previous chapter, we run some simulations to test the results obtained and to check the efficiency of the 2-choices dynamics.
The tests have been run on networks created by a Matlab function[11], written for the purpose, that yield a random network sampled according to the so called stochastic block model. In this framework we obtain a graph which is not exactly the one used in the proofs of the paper, but that on average satisfies its conditions. More precisely, to generate such random graphs we took as inputs the number of clusters and their size. Then, for each couple of vertices, we decided whether or not to place and edge between them, according to the communities to which the two vertices belong. This required to set an internal probability of having a link inside vertices of the community and an external probability of a link for vertices of different communities, for each possible couple. Obviously the internal probability had to be way higher than the external one, to create well defined communities. On average the resulting graphs had similar characteristics to the one of the paper, and knowing a priori the community structure allowed us to check whether the algorithm succeeded or not.
We have been able to test the effectiveness and robustness of the method on many random graphs with different features, like the number of nodes, the size of communities and the number of links. Furthermore, for networks with the same features, we run 100 different stochastic block models, to have more statistically significant results.
Before reporting the results of the simulations, we think it is worth to provide a quick explanation of the inputs that the function takes in the code we used, to help the understanding of the following lines. The first input of the SBM function is pi, namely the vector containing the probability of presence of links between vertices that belong to the same community. Second, the pe vector contains the probabilities of having a link between nodes in different communities, for each possible couple. The vector c is simply composed by the size of each community, and the two final parameters $Graph\_type$ and $Self\_loops$ just make the graph directed or undirected and with or without self-loops. For our simulations we will just work with undirected and with no self-loops graphs, hence we can just ignore them.
Beside the function to create artificial networks, we wrote as well the code for the 2-choices

[11] Section 7.1

dynamics following the steps of the paper examined. It can be found in Section 7.2.

After few trials, we decided that a good trade off between computational complexity of the algorithm and statistical significance, was to run simulations on networks with $N = 2000$ nodes, divided in two communities of the same size, namely $n = 1000$. What we changed in each simulation were the probabilities of presence of links between nodes, respectively pi and pe, the number of L parallel runs and the number of iterations $tmax$ of the dynamic process.
As it could be expected, by increasing the number of parallel runs L the overall efficiency of the algorithm grew, since it can be seen as an increase in the dimensions of the metric space of the *k-means* method[12], that makes the distance between clusters more and more evident. Following the paper by Cruciani et al., $L = c \log n$, with c a constant greater than 0. In particular we decided to test three different values of L, namely $L = 10$, 20 and 30 and here the results:

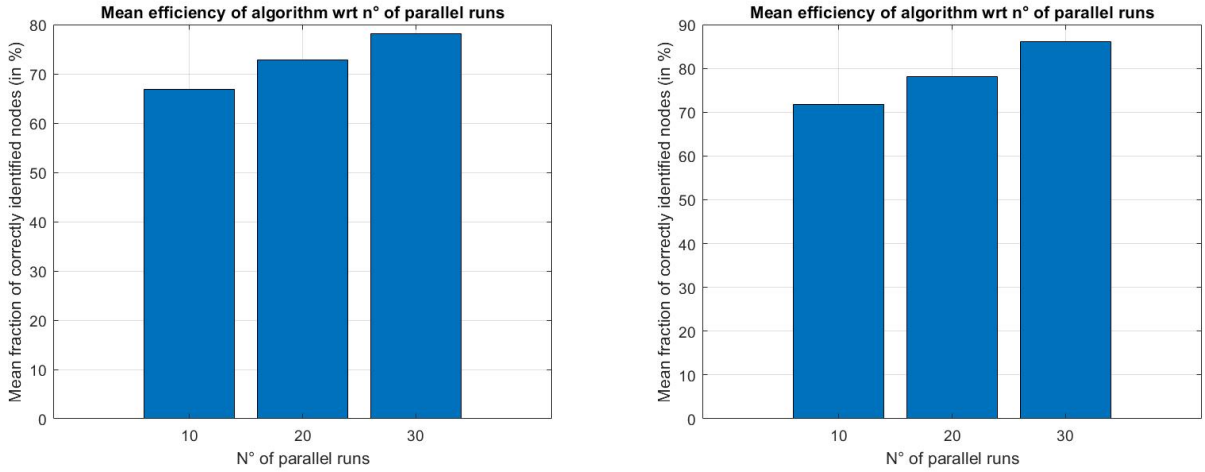


Figure 4.1: On the x-axis there is the number of the L parallel runs, while on the y-axis the fraction of correctly identified nodes over 100 simulations for each bar. Here we consider networks with a $pi = 0.7$ (left) and 0.8 (right), a $pe = 0.1$ (both) and number of iterations $tmax = 13$.

It is evident how the increase of the parallel runs improves the effectiveness of the algorithm. Setting higher values, for example $L = 50$, made the algorithm recognize on average more than the 90% of the nodes. In our tests we did not use these high values due to the significant increase in the running time of the algorithm, and to highlight that already with lower numbers of parallel runs the algorithm has a good performance.
Shifting to the number of iterations of the 2-choices dynamics that the algorithm performed, we tested a $tmax$ between 7 and 13. The choice of this interval derives from the combination of Lemma 4 and Lemma 5, that state that within $\log n + \log(\log n)$ rounds of the dynamics, the process will reach an *almost-clustered* configuration. In our case, with $N = 2000$, $tmax$ should be equal to 10, hence we chose to run tests on an interval of ± 3. Below we report

[12]Section 2.3

two histograms that compare the efficiency of the algorithm with 7 iterations versus 13:

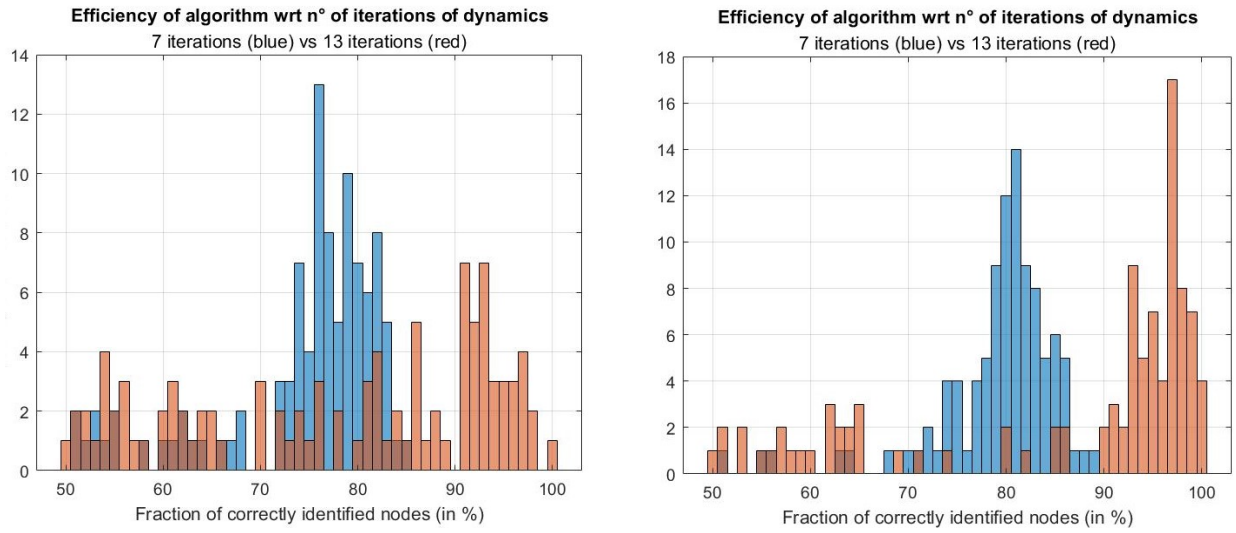


Figure 4.2: Here we consider networks with a $pi = 0.7$ and 0.8, a $pe = 0.1$ and $L = 30$. The total number of simulations of each histogram is 100.

It is evident how the density function of the simulation with our maximum number of iterations is shifted toward the higher values on the right. However, even with 7 iterations, we obtain good results. The significant difference between the two is the ability to achieve efficiency values higher than 90%: with fewer iterations, even if the mean of correctly identified nodes is around 80%, it never exceeded that threshold.
In general, also in this case, we observe a positive correlation between the number of iterations and the mean efficiency of the algorithm. Anyway, it is important to note there is no significant difference in results between 10 and 13 iterations.

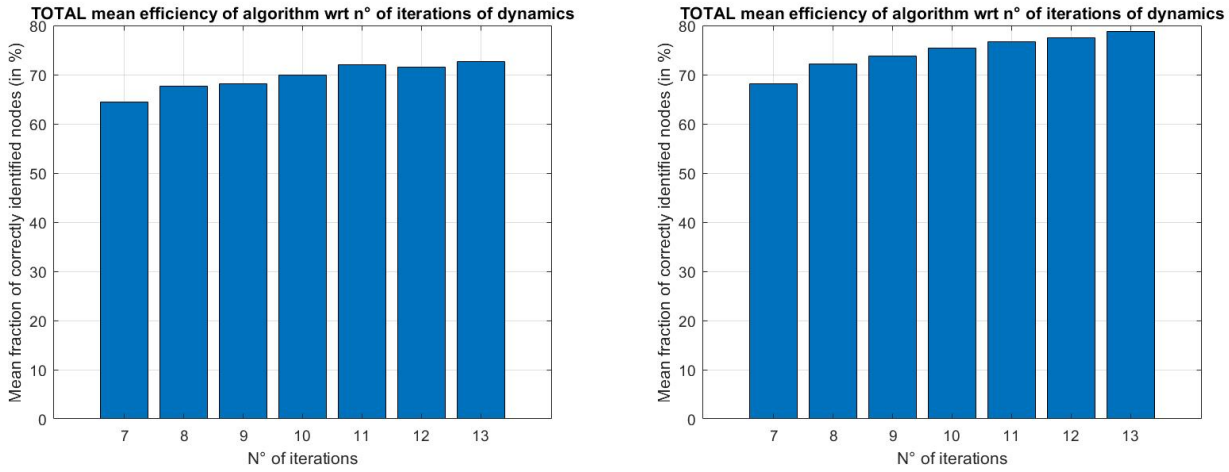
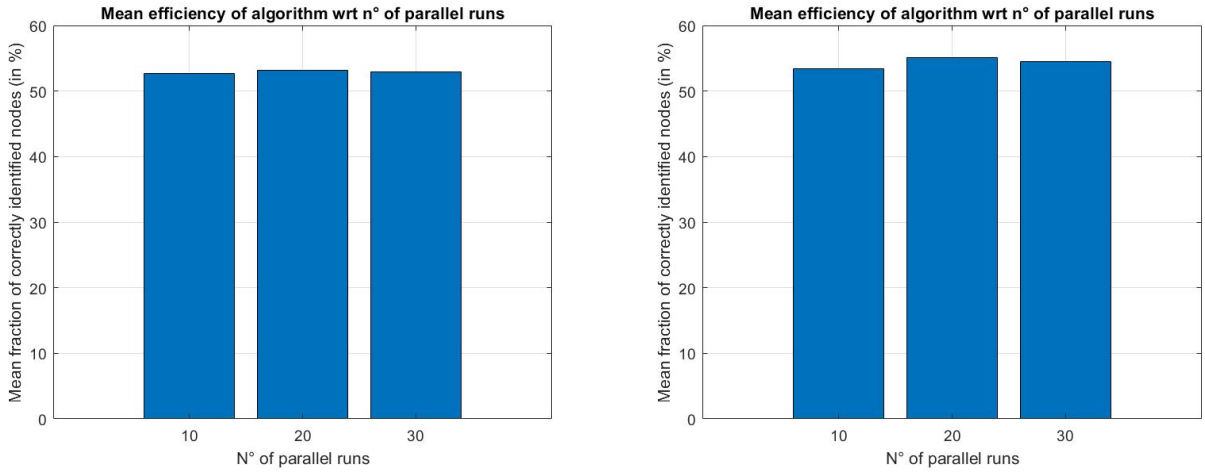


Figure 4.3: On the x-axis there is the number of iterations $tmax$, while on the y-axis the fraction of correctly identified nodes. Each column averages 300 simulations with different values of L. Moreover, we consider networks with a $pi = 0.7$ (left) and 0.8 (right) and a $pe = 0.1$ (both).

Considering this, it would be better to set the $tmax$ value as low as possible, since the algorithm would give good results and would gain in computational time.

In general the algorithm works well, especially if we increase the number of L parallel runs, that for a single or few runs of the algorithm would not imply a long computing time. Similar results as the ones we reported were obtained with communities of different size.
Issues and weaknesses were detected when dealing with higher values of pe, namely with different communities having many links connecting them, regardless of the value of pi. The results were quite poor and the fraction of total nodes identified on average never exceeded the 60%. Below we report the same plot of Figure 4.1, with the same characteristics except for the value of $pe = 0.2$:



The performance dropped significantly and the efficiency of the algorithm could be compared to a random choice.
For types of networks with many links connecting different communities other clustering methods would be preferable. The standard benchmark method, as mentioned before, is the modularity maximization[13], that still with all its advantages and disadvantages, appears to be the most robust technique suitable for any kind of network. Its results keep always high values of identified nodes, around $80\% - 90\%$.
Running modularity maximization on the networks that produced us poor results, we obtained a fraction of correctly identified nodes of about 80%, despite the not so clear community structure.
Performing as well modularity maximization on the first graphs we reported, like the ones of Figure 4.1, we got almost the same results of the algorithm, and when increasing the number of parallel runs our method performed even better with fractions of properly identified nodes of more than 95%, making it a reliable method for some types of networks.

For the sake of completeness, we decided to test the algorithm also on networks with three communities and even of different size. We kept the number of nodes unaltered, namely $N = 2000$, but we increased the number of parallel runs L to 50. This decision is due to the fact that the algorithm requires more information to produce good results, since the network

[13]Section 2.4

is more complex.
Furthermore, we changed also the order of magnitude of the probabilities to check whether this fact would have affected the overall efficiency. Regarding the probabilities of having a link for vertices belonging to the same community pi, we used respectively 0.5, 0.4 and 0.5. Instead, we set the values of the external probabilities pe to 0.05, 0.06 and 0.04. Finally the size of the communities: 700, 800 and 500.
The results over the 100 runs were quite good, with a mean fraction of corrected nodes of more than the 80%.

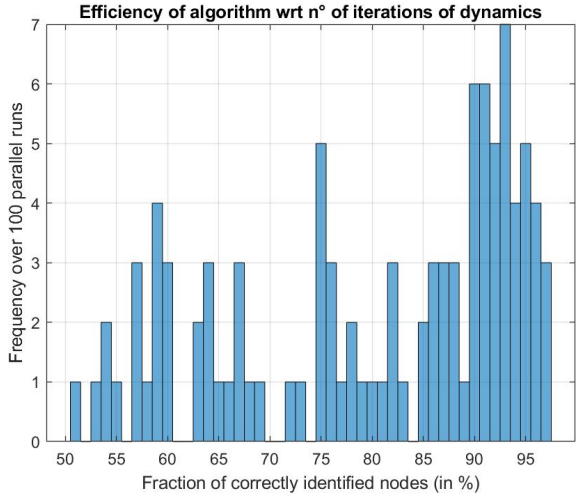


Figure 4.4: Histogram of the 100 simulations.

Even in this case modularity maximization performed better, with a fraction of identified nodes of more than 85%.
However, even if a bit more imprecise, the results continue to be good and the algorithm showed to be able produce fairly good and robust results even with slight modifications to the size of communities and to the probabilities of link.

4.2 Tests on a Real-World Network

After these general considerations on the overall efficiency of the algorithm based on computer-generated networks, it is interesting to test how it performs over a real-world network.
In networks analysis literature there are few well-known graphs regularly used as benchmark to test algorithms. In the context of community detection, the most used benchmark graphs represent real social networks, due to the natural tendency of people to create groups.
For our test, we will use one of these famous networks called Zachary's karate club [16]. It is composed by 34 nodes representing the members of a university karate class in the United States. Each link between nodes means that the two members had interactions also outside the activities of the club.
During the period in which the researcher was collecting data, by chance, there was a conflict

between the administrator of the course and the instructor, which made the club split into two groups. Half of the members followed the instructor, the others just found a new one or gave up. Basing on this, the author of the study has been able to correctly assign all but one member to the group they actually joined after the separation.
What we did is exactly the same: we made our algorithm run on the data collected just before the argument. For the simulations, we run the algorithm for 200 times, setting the parameters: $L = 50$ and $tmax = 5$. Obviously, the low value for $tmax$ derives from the reduced size of the network and we just used the number obtained by the formula contained the paper.
The results are highly satisfying: over the 200 simulations, 94 times the algorithm recognized perfectly the communities, while 50 times it misplaced just one node.

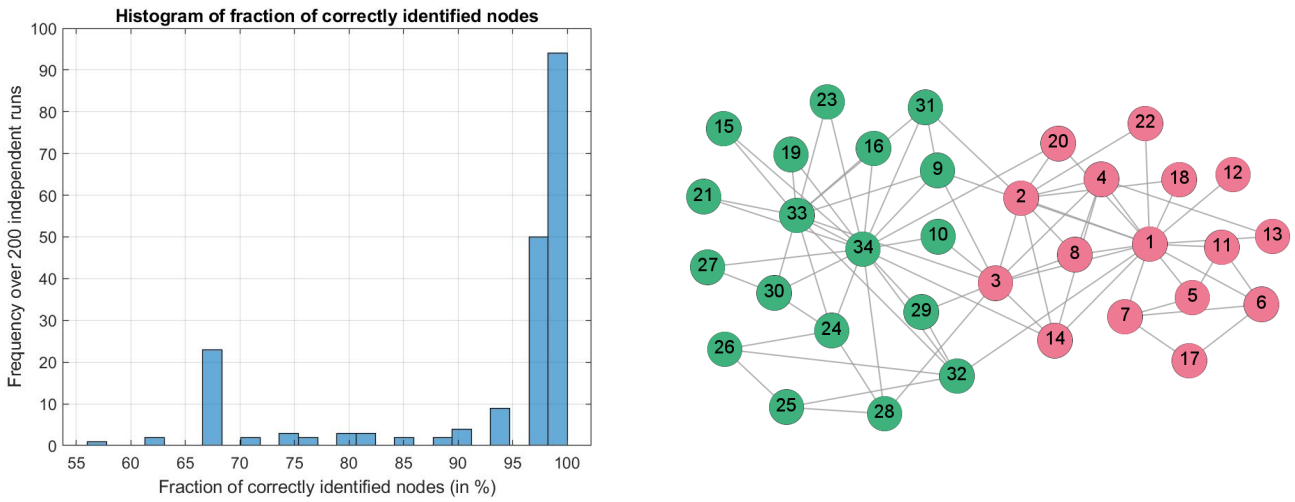


Figure 4.5: On the left the histogram of the 200 simulations. On the right the representation of the network with the two communities.

The network representation on the right of Figure 4.5 shows the resulting communities. The node labeled with number 1 represents the administrator, while the number 34 is the instructor: it is visually evident their centrality, both in the entire structure and in the two after-split communities. The sometimes misplaced node is the number 9: it has few links with both communities and understandably can create problems. Indeed, that node is the one that Zachary was not able to put in the right community.
We can safely conclude that the algorithm based on 2-choices dynamics performed really good on this real-world network, since its efficiency has been higher than the 97% in more than the 70% of the simulations and since the results would have surely led us to conclude for a community structure which is the actual one.

5 Conclusion

Nowadays, we have at our disposal so many data and being able to process them in an efficient way can be useful and a key driver for decision making policies in any sector.
Network analysis is a relatively new field of studies that adds a tool in the context of big data analysis, extracting different types of information hidden and embedded in complex network data, useful for any kind of study or research.
In this thesis we presented first the problem of community detection and its extension, then we reported some classic techniques to identify them and then finally we chose a recent and fascinating method to perform the task. We got to the heart of it, by understanding its mathematical structure and theoretical validity and then by coding the algorithm proposed in the paper to perform some simulations.
Community detection, in general, is not an easy task and it presents many issues deriving from the extent of its possible applications. To rely on an efficient method to perform this type of studies for any reason, can be important and useful and is surely an extra tool for big data analysis.
We believe that the method we reported offers an interesting solution to the issue, combining together different theories and techniques.
It certainly has some issues, especially when dealing with networks that exhibit a weak form of community structure. Nonetheless, this problem affects almost every community detection method.
Proceeding on this path, some improvements to the 2-choices dynamics may be performed. It could be interesting to slightly modify it by, for example, changing the update rule: instead of having a node sampling two neighbours and take their color if both the vertices have the same one, the size of the sample can be enlarged, choosing then the majority color among the sampled neighbours.
Furthermore, a quick remark on the decision to combine the 2-choices dynamics based algorithm with the k-means: the reason why we did so, is due to its extreme simplicity and speed in computing time. It is not essential for the well functioning of the algorithm and, in general, any other clustering method of points over a Euclidean space will do the trick, maybe also with better results.
It is clear that there is a long way to go to improve both this method and, in general, there is room for the development of several other community detection techniques. Luckily today

for these type of algorithm and skills in data analysis are in high demand and we expect they will experience a strong development in future years.

6 Bibliography

[1] Cooper, C. and R. Elsässer and T. Radzik, 2014, *The Power of Two Choices in Distributed Voting*, International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, Vol 8573.

[2] Cruciani, E. and E. Natale and G. Scornavacca, 2019, *Distributed Community Detection via Metastability of the 2-Choices Dynamics*, 33^{rd} AAAI Conference on Artificial Intelligence.

[3] Fortunato, S., 2010, *Community detection in graphs*, Physics Reports, Vol. 486, Issues 3-5, pp. 75-174.

[4] Pastor-Satorras, R. and A. Vespignani, 2001, *Immunization of Complex Networks*, Physical Review E, Vol. 65.

[5] Lorrain, F. and H. C. White, 1971, *Structural Equivalence of Individuals in Social Networks*, The Journal of Mathematical Sociology, Vol. 1, pp. 49-80.

[6] Fouss, F. et al., 2007, *Random-walk computation of similarities between nodes of a graph, with application to collaborative recommendation*, IEEE Transactions on Knowledge and Data Engineering, Vol. 19, Issue 3, pp. 355-369.

[7] Girvan, M. and M. E. J. Newman, 2002, *Community structure in social and biological networks*, PNAS, June 11, 2002, Vol. 99, no. 12, pp. 7821-7826.

[8] Freeman, L. C., 1977, *A Set of Measures of Centrality Based on Betweenness*, Sociometry, 1977, Vol. 40, no. 1, pp. 35-41.

[9] Girvan, M. and M. E. J. Newman, 2004, *Finding and evaluating community structure in networks*, Physical Review E, Vol. 69.

[10] Yang, B. and J. Liu, 2008, *Discovering Global Networks Communities Based on Local Centralities*, ACM Transactions on the Web, Vol. 2, Issue 1, Article 9, pp. 1-32.

[11] Donetti, L. and M. A. Muñoz, 2004, *Detecting Network Communities: a new systematic and efficient algorithm*, Journal of Statistical Mechanic: Theory and Experiment, October 27, 2004.

[12] Newman, M. E. J., 2004, *Detecting community structure in networks*, The European Physical Journal B, Vol. 38, pp. 321-330.

[13] Duch, J. and A. Arenas, 2005, *Community detection in complex networks using Extremal Optimization*, Physical Review E, Vol. 72.

[14] Dubhashi, D. and A. Panconesi, 2009, *Concentration of Measure for the Analysis of Randomised Algorithms*, Cambridge University Press, pag. 1-15, ex. 1.1.

[15] Clementi, A. E. F. et al., 2018, *A Tight Analysis of the Parallel Undecided-State Dynamics with Two Colors*, 43^{rd} International Symposium on Mathematical Foundations of Computer Science, Article no. 28.

[16] Zachary, W., 1977, *An Information Flow Model for Conflict and Fission in Small Groups*, Journal of Anthropological Research, Vol. 33, no. 4, pp. 452-473.

7 Matlab Codes

7.1 Stochastic Block Model Function

```
function [A, E] = SBM(pi, pe, c, Graph_type, Self_loops)
%INPUTS:
%N: number of communities
%pi: vector of probabilities of presence of links inside each
   community - [1xN]
%pe: vector of probabilities of presence of links between each
   couple of communities** - [1x{(N^2-N)/2}]
%c: vector containing the size of each community - [1xN]
%Graph_type: 'Directed' or 'Undirected'
%Self_loops: 'Y' or 'N' (Yes or No)

%**pe needs to be written as: [peAB peAC peAD peBC peBD peCD]


%OUTPUT:
%A: adjacency matrix
%E: edge list (list containing in each row a couple of vertices
    connected by an edge)

%EXPORT: copy and paste the following lines to export the
   output
%Excel file of the adjacency matrix:
%xlswrite('SBM_Adj.xlsx', A);

%.csv file of the edge list (suitable for Gephi import):
%header = {'Source' 'Target'};
%E_tab = table(E(:,1), E(:,2), 'VariableNames', header);
%writetable(E_tab, 'SBM_EdgeList.csv');
```

```
N = size(c,2);

%MATRIX OF PROBABILITIES:
m = 0;
p = zeros();
for i = 1:N
    for j = 1:N
        if i == j
            p(i,j) = pi(1,i);
        else
            if j > i
                m = m+1;
                p(i,j) = pe(1,m);
                p(j,i) = pe(1,m);   %NxN matrix
            end
        end
    end
end

%Cycle to identify row and columns in the adjacency matrix:
pos = zeros(1,N+1);
for i = 1:N
    pos(1,i+1) = sum(c(1,1:i));
end

%CREATION OF MEAN ADJACENCY MATRIX:
n = sum(c);
A_bar = ones(n);
for i = 1:N
    for j = 1:N
        B = ones(c(1,i),c(1,j));
        A_bar(pos(1,i)+1:pos(1,i+1),pos(1,j)+1:pos(1,j+1)) =
           kron(p(i,j),B);
    end
end

%ADJACENCY MATRIX:
A = A_bar;
for i = 1:n
```

```
    for j = 1:n
        if A(i,j) > rand
            A(i,j) = 1;
        else
            A(i,j) = 0;
        end
    end
end

if strcmpi(Graph_type,'Undirected')    %Directed/Undirected
   option
    for i = 1:n
        for j = i:n
            A(j,i) = A(i,j);
        end
    end
elseif strcmpi(Graph_type,'Directed')
end

if strcmpi(Self_loops,'N')    %Self-loops option
    for i = 1:n
         A(i,i) = 0;
    end
elseif strcmpi(Self_loops,'Y')
end

%EDGE LIST:
s = 0;
for i = 1:n
    e2 = find(A(i,:)==1)';
    sz = size(e2,1);
    e1 = i*ones(sz,1);
    E(s+1:s+sz,:) = [e1 e2];
    s = s+sz;
end
```

7.2 2-Choices Dynamics Algorithm

Note that the following code is for a single run of the algorithm. A simple `for` cycle covering the entire code is needed to get the 100 (or any other number) simulations we used in Chapter 4.

```
pi = [0.8 0.8];
pe = [0.1];
c = [1000 1000];
[A] = SBM(pi, pe, c, 'Undirected', 'N');
fprintf('SBM created \n')

%INPUTS:
%A: adjacency matrix
%L: number of parallel runs [l = c log n]

L = 30;

N = size(A,1);
neigh = zeros();   %List of neighbours for each vertex
for i = 1:N
    count = 0;
    for j = 1:N
        if A(i,j) == 1
            count = count + 1;
            neigh(i,count) = j;
        end
    end
end

deg = zeros(N,1);   %Degree of each vertex
for i = 1:N
    deg(i,1) = sum(neigh(i,1:end)> 0);
end

tmax = round(log(N) + log(log(N)));   %Number of iterations
                                       of the dynamics
fprintf('Number of parallel runs: %d \nNumber of iterations in
   each run: %d \n',L,tmax)
```

```
Y = zeros(N,L);   %Will contain the vector of colors of each
                   node in each one of the l parallel runs
for k = 1:L

%INITIALIZATION PHASE: The nodes are colored with a probability
    "p".
    p = 0.5;
    colors = zeros(N,1);   %Vector containing the color of each
        node
    for i = 1:N
           if rand(1,1) < p
            colors(i,1) = 1;
           end
    end

%2-CHOICES DYNAMICS:
    colorsnew = colors;
     for t = 1:tmax
         for i = 1:N
         sample = randsample(neigh(i,1:deg(i)),2);
             if colors(sample(1)) == colors(sample(2))
             colorsnew(i,1) = colors(sample(1));
             end
         end
         colors = colorsnew;
     end
  fprintf('Run  n  : %d / %d \n',k,L)

%STORING RESULTS:
 Y(1:end,k) = colorsnew(1:end);
end

comm_algo = kmeans(Y,size(c,2));


%FEEDBACK: If the communities are known a priori (and the
   adjacency matrix "A" is well ordered), use this code to
   identify how many nodes have been correctly assigned to
   their real community.
```

```
comm_real = zeros();
for i = 1:size(c,2)
    comm_real(1:c(1,i),i) = i;
end
comm_real = nonzeros(reshape(comm_real,[],1));

right = 0;
for i = 1:N
    if comm_algo(i,1) == comm_real(i,1)
        right = right + 1;
    end
end
right = [right N-right];
comm_right = max(right);
fprintf('The number of nodes correctly assigned to their real
   community is %d / %d \n',comm_right,N)

right_percent = round ( comm_right / N *100);
fprintf('SUMMARY: \n N  of nodes: %d \n N  of runs: %d \n N
    of iterations for each run: %d \n N  of nodes in correct
   community: %d (%d%%) \n',N,L,tmax,comm_right,right_percent)
```

LUISS

Department of Economics and Finance
Chair of Financial and Economic Networks

Community Detection Algorithms for Network Data: a Markovian Approach

(Summary)

SUPERVISOR
Dr. Matteo Quattropani

CO-SUPERVISOR
Prof. Sara Biagini

CANDIDATE
Roberto Calò
Matr. 727251

Academic Year 2021/2022

Abstract

The last few years have been characterized by an ever increasing amount of data: we have been overwhelmed by a digital revolution that has deeply changed our lives.
In our everyday life we interact with each other and more and more the interaction involves electronic devices, interlinked and connected to each other.
As a consequence, each type of interaction generates large volumes of data, constantly collected and analysed. This led to the need to be able to study the so called Big Data, and hence light has been shed on new fields of study, so to be able to fulfill the task.
In this thesis we are going to present one of these recent fields called *Network Analysis*, and in particular we will focus on one of its branches that comes with the name of *Community Detection*. What we will do, in short, is to investigate different methods to identify the community structure in network-type data. Then we will focus on a particular and recently proposed technique based on the so called *2-Choices Dynamics*. Ultimately, we will test this method by some simulations over both computer-generated networks and a real-world one, to test its efficiency.

1 Introduction

The last few decades have been characterized by a shocking technological and digital development. Internet allowed the exchange of information instantly regardless of the distance. Computers and smartphones helped to make this even easier, just with a "click" or a "tap".
As a result, large amount of data are constantly generated by anyone of us: when buying something online, when chatting with friends, when we send an email, when we download an app, etc. All these various data are being continuously exchanged, collected and stored in massive databases and that is the reason why we commonly call them *Big Data*.
Due to their complexity and extent, both in terms of volume and of type, to be studied they require proper methods of analysis, capable of putting into relation heterogeneous data. Being able to perform this task successfully means to be able to uncover trends and patterns, evaluate and forecast and, in general, to take data-driven decisions.

In this context, new fields of studies emerged and became popular in the recent years with the purpose of improving big data analysis. One of these fields is the so called *Graph Theory*, a branch of mathematics, that saw his birth in the 18th century. Since then, some mathematicians explored that field and laid the foundations for its development.
At the beginning of the new century, the growing need to interpret network-type data, gave a strong impulse to the implementation of a practical application. This led to the development of what we call *Network Analysis*, a field which studies complex networks data, based on the theoretical concepts of graph theory. It is a mix of the mathematical notions with the practical approach of statistics, physics and algorithms. Its ultimate aim is, beside the theoretical study of the topology of a network, to extract both qualitative and quantitative information from network data.
We are going to present first some general concepts of graph theory and then we will focus on a particular branch of network analysis, called *Community Detection*. Some preliminary theoretical notions that are needed, will be provided in the following sections.

1.1 What is a Network?

A network or a graph is a mathematical structure consisting of two entities: a set of agents, which we call nodes (or vertices or points) and a set of pairwise interactions among them, shown up by the presence of an edge (or link) between two nodes.
Graph theory and network analysis have been recently employed to better understand, describe, optimize and sometimes also to try to predict social, biological, virtual, technological, economic and any other type of relationship. To give practical examples, when we talk about networks we refer for instance to the network of friends on a social network, the network of web pages interlinked with each other, biological networks like the food chain or economic networks like the World Trade Web, the international web made of import-export relationships among countries. Almost any activity that involves entities interacting with each other can be represented by a graph. Obviously, each network presents its own features, but there are some of them which deserve to be mentioned since they are shared by a remarkable set of real-world networks:

- to be sparse: they have few links compared to the maximum possible;
- to be small worlds: the average distance between two random-picked nodes is small;
- to present inhomogeneity in the edge distribution: many vertices with few links and few vertices with many links;
- to present a clustered structure: it is common to observe groups of vertices with a dense concentration of links within them, and few edges connecting to other groups.

This last feature is called community structure (or clustering) and can be explained by the fact that groups of vertices with common characteristics are probably linked to each other and play a similar role in the whole system. These groups are called communities and their identification can play an important role in all those fields whose structure can be represented by a graph.
In practice this last concept can have a wide range of application domains: in a generic network

the classification of vertices according to their structural position may help to find the central nodes of a community, which are most likely to be important for the stability or well-functioning of the whole, while vertices on the boundaries of clusters may play an important role in mediation with other groups. For an online platform, for example, it could be convenient to identify clusters of customers with similar interests and preferences. This would allow to set up a smart and targeted advertising campaign and more importantly to create an efficient recommendation system, that suggests the user something that probably is glad to meet. A further possible implementation is in the field of epidemiology, with the analysis of social communities to help the tracking of the spread of an infectious disease.
The activity of identification of communities has been faced with some methods that have been proposed through the recent years and others continue to be proposed. The task is not so easy as one may imagine, and still today there is no a generally recognized efficient method to perform community detection in practice.
The aim of this work is first to provide a good definition and understanding of communities and of the issue related to how to identify them, and to report some interesting solutions.

1.2 Basics of Graph Theory

The nodes or vertices or points represent the agents. They are identified by a label and, if there is a relationship between them, they are connected by an edge or link.
The most convenient way to represent graphs' structure is by using matrices. The most natural choice is the adjacency matrix, a square matrix that takes as entries 0 or 1 in the position i, j depending on the presence of a link between the node i corresponding to the i-th row and the j one corresponding to the j-th column. Actually, the adjacency matrix can assume any non negative number as entry. If so, we are dealing with a weighted graph, in which the weights are given by the number appearing in the corresponding entry and it represents the intensity of the relationship between the two nodes under consideration.
Depending on whether the link has a direction, namely one of the two nodes is the source and the other the target and not necessarily viceversa, we distinguish between directed and undirected graphs. In the directed case, the links will no more be just lines, but arrows pointing vertices depending on the orientation of the relationship. In the undirected case, the adjacency matrix will be symmetric. A typical clarifying example of the difference between undirected and directed graphs is the comparison of online social networks: Facebook versus Twitter and Instagram. In the first case the relationship between nodes is reciprocal, since, after accepting the friend request, both the individuals will appear in the friends' list of the other. Clearly, in this case, the underlying network will be undirected. On the contrary, on Twitter or Instagram, in which you decide to follow one account that will not necessarily follow you back, the newly created relationship will be one-way and hence directed.
It is also possible to have nodes with self-loops, which corresponds to vertices linked with themselves. In this case, on the main diagonal of the adjacency matrix there will be some entries

different from 0.
Two nodes connected by a link are called neighbours and the degree of a vertex is the number of its neighbours.
In this thesis we are going to deal with the simplest form of graphs, namely unweighted, undirected and with no self-loops, but note that any result can be extended to any type of graph with the proper adjustments.

2 Community Detection

2.1 Communities and Partitions

Since the concept of community does not have an exact definition, due to its complexity, we will present some of the most agreed ones making use of some degree of arbitrariness and commonsense.
A rough quantitative definition considers a community a group of nodes in which we observe a significantly larger value of the intra-cluster density compared to inter-cluster one, namely we have considerably more edges connecting vertices of the same group compared to the edges connecting with the rest of the nodes of the graph. Problems may arise in the comparison between the two densities in the quantification of "significantly larger value": it strongly depends on the case under study. With these type of problems the above mentioned arbitrariness comes into play.
Another definition is based on the robustness of the clusters. It relies on the idea that the higher the number of edges that need to be removed to disconnect two vertices, the larger the nodes are likely to belong to the same community.
A further common definition is based on vertex similarity with respect to some property, for example considering two nodes with the same neighbours as belonging to the same community, even in absence of a link connecting them.
It is important to highlight that the same graph may have different combinations of communities. Each of these divisions in clusters is called a *partition* of the graph.
Besides the raw and different definition of community we choose, it is as well important to assess quantitatively how much a certain partition of a graph represents its community structure. This is measured by *modularity*, a function that quantifies the quality of a partition by comparing the actual edge density of a graph with the expected density of a graph with the same degree sequence but with links attached at random. This idea is based on the fact that a graph with random edges is not expected to exhibit a clustered structure.

In principle, maximizing modularity would give the best partition in terms of communities. However, we must consider that the question whether a partition is better than another strongly depends on the community definition we choose, and that the maximization of modularity over the space of all the possible partitions of a graph can be a complex problem due to the fact that the number of all the possible partitions is exponential in the number of the vertices.
In any case, this is the most used method for a qualitative evaluation of partitions and we will use it as benchmark for our results.

2.2 Traditional Methods

The first and most simple class of algorithms to be presented are the *graph partitioning algorithms*. The approach consists in creating a partition of nodes divided in a number of communities such that it is minimized the number of edges lying between clusters. Due to its roughness, we do not expect to find optimal results as it requires too many input information such as the number of communities and their size.
To overcome this issue, the class of *hierarchical clustering algorithms* has been introduced, built to detect the multilevel structure of a graph. In fact, the hierarchical structure is a quite common property of many real-world networks: communities are included in larger communities, which in turn are included in larger communities and so on (like the human body, composed by organs, composed by tissues, composed by cells). The first step consists in choosing a similarity measure between vertices depending on the case under study. Edges are added one by one following the decreasing order of the similarity score, calculated for each possible pair of nodes. By adding the edges, the graph should bring together similar vertices and create larger components which end up being our communities. Usually a stopping criterion is set, such as a given number of communities or a certain value of modularity. The main advantages of this algorithm are that is quite easy, and it requires few inputs to work. On the other hand, it does not provide any information on the goodness of partitions obtained and may give unreasonable results if dealing with a network without any hierarchical structure.
Opposed to this method there is the class of *divisive algorithms*, which instead focuses on the removal of edges which are considered to lie between communities. As a consequence, clusters will tend to get disconnected from each other and the community structure would be highlighted. As before, also in this context we need to choose a betweeness measure. The most famous and used one is the edge centrality associated to a link, defined as the number of shortest paths between any pair of vertices that go through it that should reflect the importance that a link has in the communication between clusters. To be more clear, central edges of a community can be easily bypassed due to the high density of links. On the contrary, edges on the boundary of a cluster play a crucial role in inter-community communications. For this reason, we expect high values of edge betweeness for boundary edges and low values for central ones.
The algorithm consists in the calculation of the centrality for each edge, the removal of the link

associated to the highest score and the recalculation of the new edge centrality. These steps are then iteratively repeated. This algorithm turns out to be simple, intuitive and to give reliable results, the only drawback is its computational complexity: it requires many computation steps and it results to be efficient only in networks of limited size.

2.3 Spectral Clustering

Spectral clustering is an interesting and elegant technique that creates partitions of graphs by making use of the eigenvectors of matrices which represent the network structure. The initial set of nodes is transformed in a set of points in the Euclidean space where the orthonormal basis is that defined by the eigenvectors of the matrix used to represent the network. Because of the change of representation, the cluster properties of the network become much more evident. The most used matrix in this context is the laplacian matrix L, defined as the difference between the diagonal matrix of the degrees and the adjacency matrix. By construction the laplacian is symmetric and the sum over each row is 0. Therefore, we can surely say that there is at least one zero eigenvalue and it is surely the smallest one since it is a diagonally dominant matrix. The interesting property regarding its eigenvalues is that in a graph with K communities, the zero eigenvalue of the laplacian will have algebraic multiplicity equal to one, but the following K eigenvalues will be much more close to 0 compared to the $(K+1)$-th eigenvalue. This means that the number of clusters can be retrieved by looking at the eigenvalues, finding a relatively large gap in the spectrum. However, it can be hard to identify significant gaps, especially in graphs with a not so clear community structure.
In practice, the values of the components of the lowest m eigenvectors will be close to each other for nodes belonging to the same community. This allows to plot the graph in an m-dimensional space, in which communities would appear as separated groups of points. The separation becomes more evident as the number of dimensions m increases.
To solve the problem analytically, it is necessary to introduce a function based on the distance to be minimized, since it can be seen as a measure of dissimilarity between vertices. The most common and widely used method is the *k-means*, whose algorithm starts with a configuration in which K centroids[1] are as far as possible from each other; each node is then assigned to the nearest centroid, and then the centroid of each cluster so obtained is recalculated and vertices are reclassified. After few iterations, the centroids will remain stable and the clusters will not change.
This technique is quite easy to implement and computationally efficient, the main issue is that it is not able to derive itself the number of clusters K, but they need to be specified at the beginning. Nevertheless, it delivers really good results if combined with other techniques, possibly able to get themselves the number of clusters.
Back to the general spectral clustering methods, the main drawback is the computation of the eigenvectors. If the graph is large, an exact computation would be impossible, so some

[1] an imaginary location of the center of a cluster.

approximation techniques are required. Usually, the maximum number of eigenvectors m to be calculated is set a priori.
As it should be clear, spectral clustering is not a stand-alone method for community detection, and it just partly works for the purpose. It simply provides, in an elegant way better input data for other algorithms.

2.4 Modularity Based Methods

We now discuss some algorithms which aim directly at maximizing the modularity. The underlying assumption of this part is that high values of modularity mean good partitioning.
Calling Π a generic partition of a graph in K disjointed blocks: $\Pi = (\Pi_1, \ldots, \Pi_K)$, the modularity associated to the partition Π is defined as:

$$Q(\Pi) = \frac{1}{2L} \sum_{i}^{N} \sum_{j}^{N} (A_{ij} - P_{ij})\, \delta(\Pi_{(i)}, \Pi_{(j)}) \,,$$

where L is the number of links, N the number of nodes, A the adjacency matrix, P the matrix of expected number of edges between vertices i and j in the null model, $\Pi_{(i)}$ the community of vertex i and δ a function that yields one if vertices i and j are in the same community ($\Pi_{(i)} = \Pi_{(j)}$) and zero otherwise.
The choice of the null model is arbitrary. In practice, the most used one is the Chang-Lu model, which creates a random graph with the expected degree sequence equals the actual degree sequence of the graph under analysis. The final expression of modularity so becomes:

$$Q(\Pi) = \frac{1}{2L} \sum_{ij}^{n} \left(A_{ij} - \frac{d_i d_j}{2L} \right) \delta(\Pi_i, \Pi_j) \,.$$

The simplest approach to maximize the function is a greedy technique that starts with each vertex being a cluster and no edges between them. Communities are combined together following this criterion: from the original graph an edge is picked such that it gives the maximum increase in modularity with respect to the actual configuration. All other edges are added based on the same principle. At each step we will have a different partition and the final outcome of the algorithm will be the one having the maximum value of modularity among all partitions obtained. The main advantage of this algorithm is that it is quite fast and allows the analysis of large networks.
Community detection with modularity is by far one of the best methods since it combines both qualitative and quantitative aspects and it embeds in itself all the ingredients for the issue, from the definition of community to the choice of a null model, to the quantification of strength of a community. These peculiar aspects make it so widely used and common in this field.
However, it is important to mention also its drawbacks: large values of modularity not necessarily mean that a graph has a community structure. It may happen to get not null values of

modularity in graphs in which there is no community structure. In addition, partitions associated to high modularity can result to be very different to each other. There is no guarantee that the partition corresponding to the (hypothetical) global maximum is similar to the high modularity partition chosen.

3 The 2-Choices Dynamics

3.1 Dynamics on Networks

With dynamics on networks we refer to simple stochastic processes on graphs aimed to highlight its community structure. These dynamics are part of the class of *Label Propagation Algorithms* and take their inspiration from epidemic processes: at each time step each node update its state/label according to the state of its neighbours.
The general concept can be described as follows: starting from a graph, an *initialization rule* assigns to each node a label from a finite set of possible states; nodes are then triggered by an *activation rule* and, by interacting with their neighbours, they update their label according to a predefined *update rule*. These rules are invariant with respect to time and network topology, they just require the current state of the node and those of its neighbours. The final output is an evolution of the initial graph in which, if the process is robust, efficient and well built, the community structure should be more evident and easier to find.
These methods are usually not able to find the community themselves, they need to be combined with other techniques able to group nodes together. In the next sections, we will deeply examine and analyse one of these methods, based on a process comes with the name of *2-Choices dynamics*.

3.2 Overview of the Process

Here we present in simple words the main steps of the algorithm based on 2-choices dynamics. It works as follows: each node is assigned randomly with a label, which in our case, will be a color between red and blue. At each time step, each vertex samples at random two of its neighbours: if the two nodes sampled have the same color, the initial vertex adopts it, otherwise it keeps its original one. It clearly is a Markovian process, since the probability of changing color just depends on the state of the network at the previous step.
In the very first step of the dynamic process each node of the graph picks uniformly at random and independently from others, a color between blue and red. The Theorem 1 states that performing the dynamic process on G, with some constant probability, the graph will reach an

almost-clustered configuration in few rounds and that it will keep this configuration for many further rounds with high probability.
Its proof is divided in several steps. We will prove that with a non vanishing probability, the initial distribution of the two colors will be slightly asymmetric with respect to the communities, i.e. in one of them there will be a slight majority of red nodes and of blue ones in the other (Lemma 1), just because of statistical noise. Then, when the distribution of colors is slightly asymmetric, there exist a significant probability that, as the process evolves, the bias toward the initial slight majority color in each community will become more and more evident (Lemma 2 and Lemma 3), until reaching an *almost-clustered* configuration, meaning that the vast majority of nodes in the first community supports one color, and same for the second community with the other color; furthermore, this configuration will be reached in few rounds of the dynamic process (Lemma 4 and Lemma 5). The final step is to prove that, when the process is in an almost-clustered configuration, it will remain in almost-clustered configurations for many rounds with high probability (Lemma 6).
With this framework, if we let the algorithm evolve in L parallel and independent runs of the 2-choices dynamics, we can obtain, for each vertex, a vector y containing the resulting color after a number of $tmax$ iterations of the dynamic process, for each of the L parallel runs.
We will prove that for all the pairs of nodes but a small number, y is equal for nodes in the same community and different for nodes belonging to different communities (Theorem 2).
After the process, we end up having a matrix Y whose rows are the y vectors for each node of the graph. Coming back to our initial community detection purpose, we can use the output matrix Y as an input for other traditional methods, since the issue can now be seen as a community detection problem on a metric space, like we saw in Section 2.3. In particular, for our simulations to test the efficacy of the algorithm we used the already discussed *k-means*.

3.3 Detailed Analysis

In this section we report the proofs for the validity of the just described process. Here we will present just the statements, the proofs can be found in the body of the thesis.
Consider a graph G composed by two a-regular communities connected by a b-regular cut, namely two communities V_1 and V_2 of n vertices with a neighbours in their own community, and b in the other one. Let V be the set of nodes and E the set of edges, we define $G = (V, E)$ to be a $(2n, d, b)$-clustered regular graph, by meaning that:

- $V = V_1 \cup V_2$, $\quad V_1 \cap V_2 = \emptyset$, $\quad |V_1| = |V_2| = n$;
- each node has degree $d = a + b$;
- each node in V_1 has exactly b neighbours in V_2 and viceversa.

Assume that $\frac{b}{d} \leq c_1\, n^{-1/2}$, for some positive constant c_1.
The Theorem 1 defines the evolution of the Markov chain associated to the 2-choices dynamics:

Theorem 1. Let G be as above.
Let us define two events about 2-choices dynamics on G:

$\xi = \{$*Starting from a random initialization, the process reaches an almost-clustered configuration within* $O(\log n)$ *rounds*$\}$.
For any $c \in \mathbb{N}$ fixed constant, define:
$\xi_c : \{$*Starting from an almost-clustered configuration, the process stays in almost-clustered configurations for* n^c *rounds*$\}$.
For two suitable positive constants γ_1 and γ_2 it holds:

$$\mathbb{P}(\xi) \geq \gamma_1 \quad \text{and} \quad \mathbb{P}(\xi_c) \geq 1 - n^{-\gamma_2}.$$

In Theorem 2 we exploit the result of the previous Theorem 1 to build an algorithm for community detection: if we run the 2-choices dynamics over L parallel and independent runs, each one for $tmax$ iterations, we end up with a vector y_v of length L, for each node, containing the final colors. If two of these vectors are equal in their entries, but a small number of outliers, the two associated nodes belong to the same community.

Theorem 2. Let G be as above.
Consider $L = \Theta(\log n)$ independent parallel runs of 2-choices dynamics after $tmax = \Theta(\log n)$ rounds (for each $\ell \in L$) over G.
Let $y_v(\ell)$ be the color associated to vertex v in the ℓ-th parallel run after $tmax$ iterations of the 2-choices dynamics.
Let y_v be the vector of length L associated to v whose elements are $y_v(\ell)\ \forall\, \ell \in L$. We have:

$$\forall\, v \in V_1,\ w \in V_1 \ \Rightarrow \sum_{\ell=1}^{L} \Big| y_v(\ell) - y_w(\ell) \Big| = O(1),$$

while

$$\forall\, v \in V_1,\ w \in V_2 \ \Rightarrow \sum_{\ell=1}^{L} \Big| y_v(\ell) - y_w(\ell) \Big| = \Omega(L).$$

4 Simulations

4.1 Tests on Computer-Generated Networks

Basing on the theory, we run some simulations to test the real efficiency of the 2-choices dynamics.
The tests have been run on networks created by a Matlab function, that yields a random

network sampled according to the stochastic block model: the result are random graphs in which we set the number of clusters, their size and, for each couple of communities, an internal probability pi of presence of a link between vertices in the same community and an external probability pe of presence of a link between vertices of different communities. Knowing a priori the community structure allowed us to check whether the algorithm succeeded or not. We also wrote the code for the algorithm, setting as inputs the number of L parallel runs and the number of iterations $tmax$ of the dynamic process. Finally, we used the aforementioned k-means to aggregate points in the metric space.

We tested networks of 2000 nodes, divided in two communities, with $L = 10$, 20 and 30, $tmax$ between 7 and 13 and different values of pi and pe. Each simulation was repeated over 100 different stochastic block models, to have more statistical significance.

The results displayed a positive correlation between the efficiency of the algorithm and the number of L parallel runs. Setting for example $L = 50$, made the algorithm recognize on average way more than the 90% of the nodes, but due to the significant increase in the running time we used lower values, also to highlight that the algorithm already delivers good results with less parallel runs.

Almost the same holds for the number of iterations $tmax$, with a remark: good results have already been achieved with $tmax = 7$, what it really differed with the tests with higher values is the ability to reach an efficiency higher than the 90%. Anyway, it is important to note that there is no significant difference in results between 10 and 13 iterations. Considering this, it would be better to set the $tmax$ value as low as possible, since the algorithm would give good results and would gain in computational time.

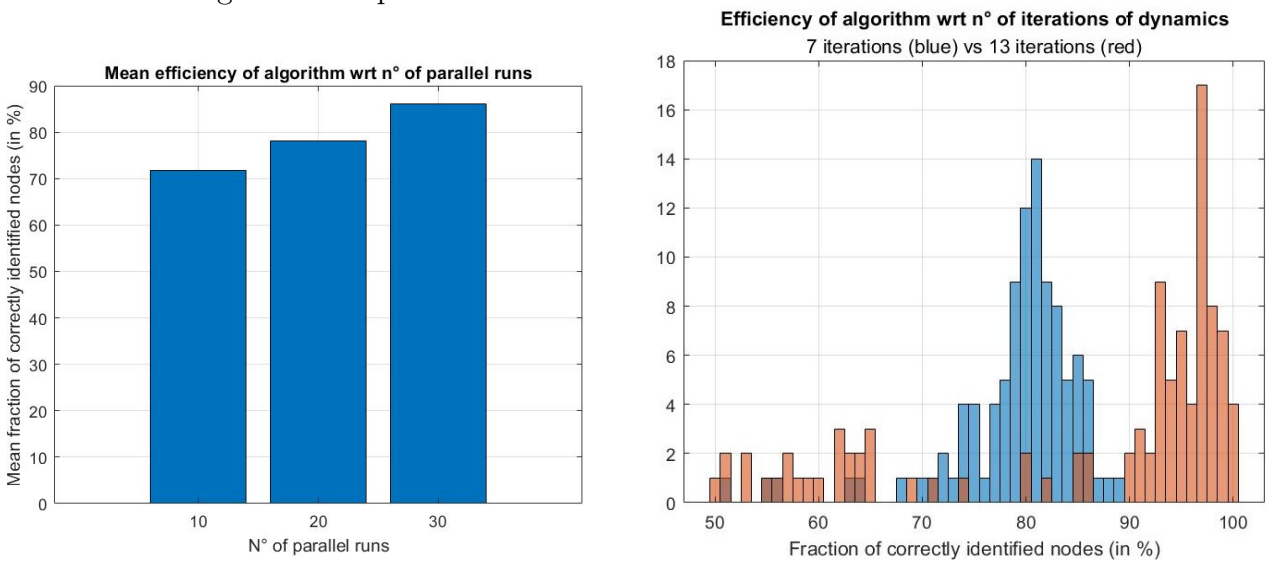


Figure 4.1: On the left, the mean fraction of correctly identified nodes with respect to the number of L parallel runs. On the right the histogram comparing the results of simulations with $tmax = 7$ versus $tmax = 13$.

Issues and weaknesses were detected when increasing the value of pe, namely with different communities having many links connecting them, regardless of the value of pi. The results were quite poor and the average fraction of total nodes identified never exceeded the 60%. The performance dropped significantly and the efficiency of the algorithm could be compared

to a random choice. For types of networks with many links connecting different communities other clustering methods would be preferable. The standard benchmark method is modularity maximization, that appears to be the most robust technique suitable for any kind of network. Its results keep always high values of identified nodes, around $80\% - 90\%$. Running modularity maximization on the networks that produced us poor results, we obtained a fraction of correctly identified nodes of about 80%, despite the not so clear community structure.
Performing as well modularity maximization on the previous networks, we got almost the same results of the algorithm. However, when increasing the number of parallel runs L our method performed even better with fractions of properly identified nodes of more than 95%, making it a reliable method for some types of networks.
We tested the algorithm also on networks with three communities and of different size. We kept the number of nodes unaltered and we increased the number of parallel runs L to 50, due to the higher complexity.
The results over 200 runs were quite good, with a mean fraction of correctly identified nodes of more than the 80%. Even in this case modularity maximization performed better, with a fraction of identified nodes of more than 85%.
Even if a bit imprecise, the results continue to be good and the algorithm showed to be able produce fairly good and robust results even with slight modifications to inputs.

4.2 Tests on a Real World Network

In networks analysis literature there are few well-known graphs regularly used as benchmark to test algorithms. In the context of community detection, the most used benchmark graphs represent real social networks, due to the natural tendency of people to create groups.
For our test, we will use one of these famous networks called Zachary's karate club. It is composed by 34 nodes representing the members of a university karate class. Each link between nodes means that the two members had interactions also outside the activities of the club.
During the period in which the researcher was collecting data, by chance, there was a conflict between the administrator of the course and the instructor, which made the club split into two groups. Half of the members followed the instructor, the others just found a new one or gave up. Basing on this, the author of the study has been able to correctly assign all but one member to the group they actually joined after the separation.
What we did is exactly the same: we made our algorithm run on the data collected just before the argument. For the simulations, we run the algorithm for 200 times, setting the parameters: $L = 50$ and $tmax = 5$. The results are highly satisfying: over the 200 simulations, 94 times the algorithm recognized perfectly the communities, while 50 times it misplaced just one node.
We can safely conclude that the algorithm based on 2-choices dynamics performed really good on this real-world network, since its efficiency has been higher than the 97% in more than the 70% of the simulations and since the results would have surely led us to conclude for a community structure which is the actual one.

5 Conclusion

Nowadays, we have at our disposal so many data and being able to process them in an efficient way can be useful and a key driver for decision making policies in any sector.
Network analysis is a new field of studies that adds a tool in the context of big data analysis, extracting different types of information hidden and embedded in complex network data.
Community detection, in general, is not an easy task and it presents many issues deriving from the extent of its possible applications. To rely on an efficient method to perform this type of studies can be important and useful and is surely an extra tool for big data analysis.
We believe that the method we reported offers an interesting solution to the issue, combining together different theories and techniques. It certainly has some issues, especially when dealing with networks that exhibit a weak form of community structure. Nonetheless, this problem affects almost every community detection method.
It is clear that there is a long way to go to improve both this method and, in general, other community detection techniques. Luckily today for these type of algorithm and skills in data analysis are in high demand and we expect they will experience a strong development in future years.

7 Matlab Codes

In this last part we just report the Matlab codes we produced to perform the simulations, namely the function that creates the stochastic block models and the algorithm based on the 2-choices dynamics.