Community detection in graphs

Felipe Castro de Britto

Dissertation presented
to the
Institute of Mathematics and Statistics
of
University of São Paulo
to
obtain the degree
of
Master in science

Program: Statistics

Advisor: Prof. Dr. Florencia G. Leonardi

São Paulo, october 2020

Detecção de comunidades em grafos

Felipe Castro de Britto

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Estatística

Orientador: Prof. Dr. Florencia G. Leonardi

São Paulo, outubro de 2020

Community detection in graphs

This is the original version of the dissertation of the
Msc. candidate Felipe Castro de Britto,
as presented to the Judging Commitee.

Community detection in graphs

Comissão Julgadora:

- Prof. Dra. Florência Graciela Leonardi - IME-USP
- Prof. Dra. Andressa Cerqueira - DEs-UFSCar
- Prof. Dr. André Fujita - IME-USP

Resumo

BRITTO, F.C. **Detecção de comunidades em grafos**. 2020. Tese (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

O Modelo Estocástico de Blocos (SBM), do inglês, *Stochastic Block Model*, é um dos modelos mais famosos de grafos com estutura de comunidades, devido a sua facilidade em simular diversas estruturas diferentes. Neste trabalho é feita uma introdução a detecção de comunidades no modelo SBM, diferentes estratégias para essa detecção, e condições para que se obtenha consitência na detecção de comunidades. É feita também uma aplicação dessas estratégias, ou algoritmos, para saber sob quais condições, ou regimes, também conhecido como limites fundamentais, esses algoritmos obtém bons resultados, em grafos simulados com diferentes regimes.

Palavras-chave: Modelo estocástico de blocos, detecção de comunidades, limites fundamentais.

Abstract

BRITTO, F.C. **Community detection in graphs**. 2020. Dissertation (Master's Degree) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

The *Stochastic Block Model* (SBM), is one of the most famous models of graphs with community structure, due to its facility in simulating several different structures. In this work, an introduction to community detection in the SBM model is made, to different approaches to this detection, and conditions to obtain consistency in the detection of communities. An application of these strategies, or algorithms, is also made to know under which conditions, or regimes, also known as fundamental limits, these algorithms have good results, in simulated graphs with different regimes.

Keywords: Stochastic Block Model, community detection, fundamental limits.

Contents

Chapter 1

Introduction

Graph theory goes back to the 17-century (Euler, 1736). Due to the advancements of computational power and the flexibility of these models, graphs, or networks, became a popular topic in recent years. Graphs are used in many different areas to describe all types of datasets that describe relationships between observations, from websites and people to financial or healthcare data. Consider each observation as a node, like a person or a blog on the internet, and the relationship between observations as edges, like people that are friends to each other (or if two blogs have links that connect both of them), you can represent all these types of data in a graph form, just having to choose the "right" model for it.

The model studied in this dissertation is the Stochastic Block Model (SBM), introduced by (Holland *et al.*, 1983), which is seen as an extension to another popular model, the Erdős-Rényi model (Erdõs and Rényi, 1960). The SBM became highly used because of its ability to represent community structure in graphs, both simple and complex structures. One of the main areas of study about the SBM is the problem of parameter estimation. In our work, we want to focus mainly on two things, the latent variable that defines the "labels" of each node and the parameter that tells us how many labels, or communities, there are in the graph. When estimating the number of communities, we call *model selection*. When trying to identify the labels, we call *community detection*. However, in other areas, it is usual to see *community detection* as a way to identify both the number of communities as the labels of each node.

Different approaches were proposed to address the problem of model selection and community detection, using maximum likelihood (Bickel and Chen, 2009), variational methods (Côme and Latouche, 2013; Daudin *et al.*, 2008), semidefinite programming relaxation (Abbe *et al.*, 2014c), spectral clustering (Rohe *et al.*, 2011) and bayesian approaches like Pas and Vaart (2016), or McDaid *et al.* (2012). Some of those methods assume that we know the number of communities, while others try to estimate this number. These methods use statistics foundations to estimate the parameters, but this is not the only approach used to estimate graphs' labels. Methods like the one in Newman and Girvan (2004) based on *modularity* do not use statistical knowledge in their algorithms, Fortunato and Castellano (2007); Lee and Wilkinson (2019) gives us an overview of some existing methods in community detection.

Another aspect of community detection is to study the conditions under which our algorithms are able to recover correctly all the labels, which is called *exact recovery* or *strong consistency*, and these conditions are known as *fundamental limits*.

Works like Abbe and Sandon (2015); Bickel and Chen (2009); Jerrum and Sorkin (1998); Rohe *et al.* (2011) tries to understand under which conditions algorithms can achieve strong

consistency. In these articles, usually, it is used an SBM with two symmetric communities, also known as the planted bisection model.

Our thesis focuses on using some known methods in different situations and evaluates how they will perform in these situations. We implement a simple but effective method to evaluate the accuracy of these algorithms. Namely, we measure each model's accuracy by studying the number of labels that are correctly labeled. This measure will serve as the main benchmark for assessing algorithm accuracy. In one part of our simulations, we look into these different graph regimes to see how the algorithms work in those situations. In another part, our focus will be on how well they are estimating the number of communities and the labels together.

1.1 Organization

In Chapter 2, we define the Stochastic Block Model and the statistical background used by some of the methods studied in this thesis, such as the relaxations for the model, the KT distribution, and some conditions for recovery, or consistency, in the model. In Chapter 3, we will explain briefly the algorithms used in this dissertation. In Chapter 4, we discuss the simulations. This Chapter is divided into two parts. The first one using a symmetric Stochastic Block Model with two communities only, since one of the algorithms only works in situations with two communities, and our focus is on the regimes of the graph. In the other part, we will test a different number of communities and see how those algorithms estimate the labels and number of communities in these situations. Chapter 5 concludes our results and what could be done after that, with suggestions for future research.

1.2 Objectives

One of this thesis objectives is to test if the algorithm used in Abbe *et al.* (2014c) is good compared with more commonly used algorithms and how viable it could be in real networks. We used this algorithm in different situations according to a strong consistency limit found in the same article. With the results, we hope to see if the algorithm can not only be used "inside" the limit but how good it can be even if these limits are not achieved.

Another objective is to test two algorithms, based on the KT distribution, and see if the algorithm can not only achieve the consistency of estimating the correct number of communities but to achieve the same when estimating the labels of each node too. The difference between those two algorithms is that they estimate both the number of communities and the labels together, which is not usual to see in statistic based algorithms in community detection. These simulations could also indicate the consistency of these approaches.

Chapter 2

Stochastic Block Models

2.1 Stochastic Block Model

A graph, or network, is a pair (V, E) where V is the number of vertices, or nodes, and E is a set of edges connecting those vertices. A graph $G = (V, E)$ is commonly denoted by it's adjacency matrix $\mathbf{X}$, since a network is completely identified by this matrix. The adjacency matrix, in this dissertation, is a binary matrix where, if there is an edge between two nodes it receives the value 1, and if there isn't an edge, it receives the value 0 (or -1), that is: $x(i, j) = 1$ if $(i, j) \in E$, and $x(i, j) = 0$ if $(i, j) \notin E$.

We will also work with only undirected, non-weighted, and with no self-loops graphs, meaning that the adjacency matrix will always be symmetric. Each pair of vertices only have assigned 1 or 0 to represent the presence or not of an edge, and the diagonal of the graph is always 0, that is:

- $\forall i \in V, \quad (i, i) \notin E,$
- $(i, j) \in E \implies (j, i) \in E.$

One of the most popular models in random graphs, the Erdős Rényi (ER) model, can be seen as a particular case of the model that is presented next, the *Stochastic Block Model* (SBM). Unlike the ER model, where each edge is placed independently with probability p, the SBM has a community structure, making more realistic models and still maintaining simplicity. When working with the ER model, there is not community detection to apply since the model cannot represent community structures.

2.1.1 The general Stochastic Block Model

The Stochastic Block Model (SBM) with k communities is a probability model for a random graph where the latent variables Z_i, $i \in \{1, \ldots, n\}$, called sometimes *labels* or *communities*, are independent and identically distributed random variables over $[k]$, $[k] := \{1, 2, \ldots, k\}$, and the adjacency matrix $X_{n \times n}$ are independent Bernoulli random variables whose parameters depend only on the nodes label. This means that given the community of the vertices, edges are independently placed in the graph, that is:

$$X(i, j) | (Z_i = a, Z_j = b) \sim Bernoulli(\mathbf{P}_{a,b}), \qquad 1 \leq i < j \leq n, \ a, b \in [k],$$

where $\mathbf{P}_{a,b}$ is a symmetric matrix.

Since in this work there are only unweighted and non-oriented graphs, that will be defined as a set of nodes $1, 2, \ldots, n$, specified by a symmetric matrix $\mathbf{X}_{n\times n} \in \{0,1\}^{n\times n}$, and since there are no self-loops in this graph, the diagonal of the adjacency matrix will always have the value 0. For each node in this graph there is a latent variable Z_i, $i \in \{1, \ldots, k\}$.

A formal statistical definition is to assume that there exists $\pi = (\pi_1, \ldots, \pi_k)$, where π is the probability distribution of each community, and $\mathbf{P} \in [0,1]^{k\times k}$ a symmetric probability matrix such that the distribution of $(Z_n, X_{n\times n})$ is:

$$\mathbb{P}_{\pi,\mathbf{P}}(Z_n = z, X_{n\times n} = x) = \prod_{a=1}^{k} \pi_a^{n_a} \prod_{a,b=1}^{k} P_{a,b}^{O_{a,b}} (1 - P_{a,b})^{n_{a,b} - O_{a,b}} \tag{2.1}$$

where

$$n_a = n_a(z_n) = \sum_{i=1}^{n} \mathbb{I}\{z_i = a\}, \qquad 1 \le a \le k,$$

$$n_{a,b} = n_{a,b}(z_n) = \begin{cases} n_a(z_n) n_b(z_n), & 1 \le a, b \le k;\ a \ne b, \quad (2.2) \\ \frac{1}{2} n_a(z_n)(n_a(z_n) - 1) & 1 \le a, b \le k;\ a = b. \quad (2.3) \end{cases}$$

$$O_{a,b} = O_{a,b}(z_n, x_{n\times n}) = \begin{cases} \sum_{1\le i,j\le n} \mathbb{I}\{z_i = a, z_j = b\} x_{ij}, & a < b, \quad (2.4) \\ \sum_{1\le i<j\le n} \mathbb{I}\{z_i = a, z_j = b\} x_{ij}, & a = b. \quad (2.5) \end{cases}$$

In the first part of the simulation, the focus is on the symmetric SBM with two equal-size communities, that is, the SBM will have $k = 2$, and there will be a probability $p \in [0,1]$ of having an edge between vertices of the same community and a probability $q \in [0,1]$, of having an edge between two vertices from different communities, and $p > q$.

This model is called symmetric when the probability p is constant for all groups, so does the same hold for q. We will use the notation $SBM(n, k, p, q)$ when referring to a general SBM, with n being the number of nodes or vertices, k the number of communities or clusters, and p and q the probabilities for vertices in the same community or different communities. We could also use a notation $SBM(n, k, \mathbf{P})$ when having more than two communities, or non-symmetric SBM, where $\mathbf{P}_{k\times k}$ is the matrix with the probabilities of connection between members of each cluster.

Before discussing the estimation of an SBM, we present some of the recovery requirements for the SBM models.

2.1.2 Recovery in the SBM

Recovery in community detection is related to how many of the latent variables Z_i, that is, which community the node i belongs, can be "recovered" by applying some algorithm to an observed graph G. A way to define the types of recovery is to use a measure called agreement (Abbe, 2017). As we can see in the following definition, some types of recovery are related to what is known in statistics as consistency of estimators.

Definition 2.1.1. The agreement between two label vectors $\mathbf{x}, \mathbf{y} \in \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{k}\}^{\mathbf{n}}$ is obtained by maximizing the common components of $\mathbf{x}$ and any relabeling of $\mathbf{y}$, that is,

$$A(\mathbf{x}, \mathbf{y}) = \mathbf{max}_{\pi \in \mathbf{S_k}} \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbb{I}(\mathbf{x_i} = \pi(\mathbf{y_i})), \tag{2.6}$$

$$\bar{A}(\mathbf{x}, \mathbf{y}) = \mathbf{max}_{\pi \in \mathbf{S_k}} \frac{\mathbf{1}}{\mathbf{k}} \sum_{\mathbf{i=1}}^{\mathbf{k}} \frac{\sum_{\mathbf{u} \in [\mathbf{n}]} \mathbb{I}(\mathbf{x_u} = \pi(\mathbf{y_u}), \mathbf{x_u} = \mathbf{i})}{\sum_{\mathbf{u} \in [\mathbf{n}]} \mathbb{I}(\mathbf{x_u} = \mathbf{i})}, \tag{2.7}$$

where $\bar{A}(x, y)$ is the normalized agreement and S_k is the space with all the relabeling possibilities for π.

Definition 2.1.2. Let $X \sim SBM(n, k, P)$. The asymptotic recovery requirements are solved by any algorithm that takes G as an input and gives an output $\hat{\mathbf{z}} = \hat{\mathbf{z}}(G)$ such that:

- **Exact recovery:** $P\{A(\mathbf{z}, \hat{\mathbf{z}}) = 1\} = 1 - o(1)$,
- **Almost exact recovery:** $P\{A(\mathbf{z}, \hat{\mathbf{z}})\} = 1 - o(1)\} = 1 - o(1)$,
- **Partial recovery:** $P\{\hat{A}(\mathbf{z}, \hat{\mathbf{z}}) \geq \alpha\} = 1 - o(1)$, $\alpha \in (\frac{1}{k}, 1)$,
- **Weak recovery:** $P\{A(\mathbf{z}, \hat{\mathbf{z}}) \geq \frac{1}{k} + \epsilon\} = 1 - o(1)$, $\epsilon > 0$.

If you say that a function $f(n) = o(1)$, then $\lim_{n \to \infty} f(n) = 0$.

Exact recovery means that the partition will be completely recovered with no errors, while *almost exact recovery* means that a small part of the graph can be misclassified. Our focus in this article will be on the *exact recovery* and *almost exact recovery* for the symmetric SBM with two communities since the algorithms can give an idea if the algorithm is indeed consistent in their estimations. In statistics *exact recovery* is known as *strong consistency* and *almost exact recovery* as *weak consistency*, and we will refer to *exact recovery* just as *recovery*.

While talking about recovery, it is important to remember some topological properties of the graph since they are necessary for the exact recovery in the SBM graphs. We can generalize the results of the ER model to the SBM. Considering the ER model $G(n, p)$ (Erdős and Rényi, 1960):

- $G(n; c\ log(n)/n)$ is connected with high probability if and only if $c > 1$,
- $G(n; c = n)$ has a giant component (i.e., a component of size linear in n) if and only if $c > 1$.

considering SBM(n,k,p,q):

- For $a > 0,\ b > 0$, SBM(n; k; a log n/n; b log n/n) is connected with high probability if and only if $\frac{a+(k-1)b}{k} > 1$ (if a or b is equal to 0, the graph is of course not connected),
- SBM(n; k; a/n; b/n) has a giant component (i.e., a component of size linear in n) if and only if $\frac{a+(k-1)b}{k} > 1$.

With these conditions Abbe *et al.* (2014c) defines a new threshold for exact recovery: if $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and impossible if $|\sqrt{a} - \sqrt{b}| \leq \sqrt{2}$. In the same paper it's defined a limit where the SDP algorithm will work best, that is: if $(a-b)^2 > (8(a+b) + (8/3)(a-b)$ exact recovery is possible, and impossible if $(a-b)^2 < (8(a+b) + (8/3)(a-b)$, and both limits will be used in the simulations with two communities. Before explaining the algorithms we need first to understand some of the approaches for parameter estimation in the SBM, since some algorithms are based in these approaches.

2.1.3 MAP in the SBM

Another approach for estimating the communities in the SBM is the maximum a posteriori estimator, using bayesian statistics to model a graph. Suppose that, given prioris to the parameters, we want to estimate Z, the community label of each node in an observed graph G. The probability of error when estimating a partition of this graph, $\Omega = \Omega(Z)$ is:

$$P\{\Omega \neq \hat{\Omega}(G)\} = \sum_{g} P\{\Omega \neq \hat{\Omega}(G)|G = g\}P\{G = g\},$$

An estimator $\hat{\Omega}_{map}(\cdot)$ minimizing the equation above must minimize $P\{\Omega \neq \hat{\Omega}(G)|G = g\}$ for every g, and to do that we must use a reconstruction ω that maximizes the posterior distribution:

$$P\{\Omega = \omega|G = g\} \propto P\{G = g|\Omega = \omega\}P\{\Omega = \omega\}.$$

Since we are working with the balanced SBM, where $P\{\Omega = \omega\}$ is the same for all partitions, the MAP estimator is equivalent to the Maximum Likelihood estimator, that is, maximize $P\{\Omega = \omega|G = g\}$ for all partitions ω of graph G.

And in the two community case, having n_{in} and n_{across} as the number of edges inside and edges between different communities, we have:

$$P\{\Omega = \omega|G = g\} \propto \left(\frac{q(1-p)}{p(1-q)}\right)^{n_{across}}.$$

Since $p > q$ the MAP for the SBM with two communities is equivalent to finding a min-bisection of the graph G, an equal partition with the minimum number of edges between partitions.

The MAP minimizes the probability of making an error for the reconstruction of the entire partition Ω, minimizing the error probability for exact recovery. Thus, if MAP fails in solving exact recovery, no other algorithm can succeed (Abbe and Sandon, 2015).

Direct maximization of a posteriori distribution in a graph model is expensive since it involves too many terms, parameters, and prioris. Because of this, relaxations have been proposed for the MAP estimator. The relaxations are done using the expression $\mathbf{z}^t\mathbf{X}\mathbf{z}$, that counts the number of edges inside the clusters minus the number of edges across the clusters, which is equivalent to the min-section problem, and is equivalent to maximize the MAP. Here, z is the vector of labels of each node, and X the adjacency matrix. Two relaxations are done using that expression, the *Spectral relaxation* and the *SDP relaxation*:

The MAP for the symmetric SBM maximizes

$$\max_{\substack{\mathbf{z}\in\{1,-1\}^n \\ \mathbf{z}^t\mathbf{1}^n=0}} \mathbf{z}^t\mathbf{X}\mathbf{z}, \tag{2.8}$$

that is an equivalent way to find the MAP estimator, that is, finding the partition with the minimum number of "crossing" edges. Here 1^n is a vector of length n with only the value 1, that is $\mathbf{z}^t 1^n = \sum_{i=1}^{n} \mathbf{z}_i$.

Spectral relaxations. The spectral relaxation uses a different constraint for z, resulting in the maximization of

$$\max_{\substack{\mathbf{z}\in\mathbb{R}^2:||\mathbf{z}||_2^2=n \\ \mathbf{z}^t\mathbf{1}^n=0}} \mathbf{z}^t\mathbf{X}\mathbf{z}, \tag{2.9}$$

Without the constraint $\mathbf{z}^t\mathbf{1}^n = 0$ the result of this maximization is the eigenvector corresponding to the largest eigenvalue of X, and $X1^n$ is the vector that contains the degrees of each node in g. This equation is easier to compute than the actual MAP estimator. We can also write the MAP estimator as a minimizer of

$$\underset{\substack{\mathbf{z}\in\{1,-1\}^n \\ \mathbf{z}^t1^n=0}}{min} \sum_{1\le i<j\le n} X_{ij}(z_i - z_j)^2. \tag{2.10}$$

This equation also looks for the best balanced cut between two partitions of the graph, and as in the equations before, it is an equivalent approach to the MAP estimator. With a few manipulations we get

$$\underset{\substack{\mathbf{z}\in\{1,-1\}^n \\ \mathbf{z}^t1^n=0}}{min} \mathbf{z}^t\mathbf{L}\mathbf{z}, \tag{2.11}$$

where $\mathbf{L}$ is the Laplacian of the graph, $\mathbf{L} = \mathbf{D} - \mathbf{X}$, $\mathbf{D}$ is the degree matrix of the graph, and $\mathbf{X}$ the adjacency matrix. The degree of an adjacency matrix $\mathbf{X}$ is

$$d_i = \sum_{i=1}^{n} x_{ij},$$

and the degree matrix is a diagonal matrix with the degrees on the diagonal. When using the Laplacian matrix, 1^n is an eigenvector of L with eigenvalue 0. The relaxation to a real-valued vector leads directly to the second eigenvector of L to determine the communities. A problem with this approach is that if the graph becomes sparse, the results no longer determine the communities since the second eigenvector may concentrate on nodes with large degrees.

SDP relaxations. *Semidefinite Programming* is a sub-field of convex optimization concerned with the optimization of a linear objective function. In the symmetric SBM with two equal communities, the SDP relaxation changes the MAP maximization into a linear maximization, similar to the min-bisection problem. We know that $tr(\mathbf{AB}) = tr(\mathbf{BA})$ for matrices with same dimension, and this results in

$$\mathbf{z}^t\mathbf{X}\mathbf{z} = \ Tr(\mathbf{z}^t\mathbf{X}\mathbf{z}) = \ Tr(\mathbf{X}\mathbf{z}\mathbf{z}^t). \tag{2.12}$$

Making $\hat{\mathbf{Z}}_{sdp}(g) = \mathbf{z}\mathbf{z}^t$ we have

$$\hat{\mathbf{Z}}_{sdp}(g) = \underset{\substack{\mathbf{Z}\succeq 0 \\ \mathbf{Z}_{ii}=1,\ \forall i\in[n] \\ \mathbf{Z}\mathbf{1}^n=0}}{argmax}\ tr(\mathbf{X}\mathbf{Z}). \tag{2.13}$$

The first two constraints force $\hat{\mathbf{Z}}$ to be $\mathbf{z}\mathbf{z}^t$ for a vector $\mathbf{z} \in \{1, -1\}^n$, and the last is a balance requirement. To handle the constraint $\mathbf{Z}\mathbf{1}^n = 0$ one could substitute the matrix $\mathbf{X}$ by a matrix that receives 1 if there is an edge between two nodes and -1 otherwise. Although the SDP relaxation is not as sensitive as the basic spectral relaxations, it is more complex, and the equation could not be solvable in an optimal time depending on the size of the graph.

2.2 Community Structure Problem

In real data that networks can represent, there is a high complexity that simple models, as the Erdős-Renyi model, can not represent, like community structures. These networks, like websites on the internet, companies in a country, and any other type of population, can have such structures, called communities, or clusters, that differentiate their members from the ones in another cluster. Understanding these structures, how many there are, and how they relate to each other is a good way to understand the data population and get precise information.

Identifying communities, or *community detection*, became popular in recent years with the advancements of computational power and statistics theory, getting attention from many areas, such as biological sciences, computer, sociology, psychology sciences, math, and statistics. There are numerous topics that are studied, such as making theories about the conditions that a community must have to retrieve some information, creating algorithms that can run in faster times, developing new theoretical models that can represent better real networks, or just applying these models to other networks and studying their findings. This range of studies usually arrives because when studying complex networks, we come upon questions like *"Does this graph have a community? How many of them"*, *"Does my model really represents the structure of this graph? Moreover, does my algorithm can be applied to large graphs?"*, and to answer all these questions, one must have knowledge of many different areas, such as statistics, math, and programming languages, and know about the data that's being studied; that is why many researchers focus on their areas of expertise to tackle this problem.

Usually, in a more philosophical way, the first thing that someone must do is define what a community is. Using this definition, the researcher will use a method that takes that into account. Since we will work with only unweighted graphs, our definitions of communities are based only on the structure of the network. The two most common types of communities, defined as in Caldarelli and Vespignani (2007b), are:

- **Self-referring communities:** this type of community is usually called a clique, that is, a community is a group of fully connected subgraph, meaning that all nodes inside the same group are connected to each other.
- **Comparative communities:** unlike the *self-referring community*, this type of definition is based on the sense that the nodes "inside" a community are more connected, while connections between nodes in different communities are less likely to occur. Radicchi *et al.* (2004) proposes to use the sum of degrees of the nodes in a community as a metric to define a good community.

The first definition usually has problems when applying to real-world problems since its algorithms are costly. With the growing size of the network, it may not be able to get optimal results, and it is not a common characteristic found in real networks. The *comparative communities* definition is easier to be used in large graphs since it is easier to tackle the problem from a mathematics/statistics point of view. It is a definition similar to the concept of the SBM model.

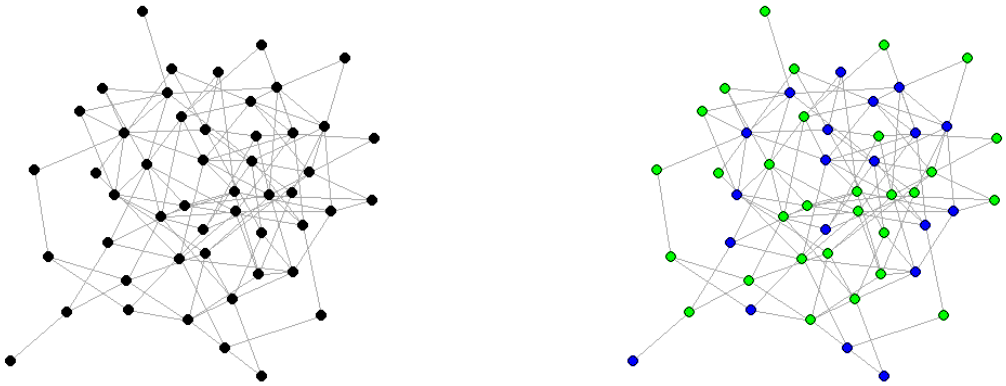

Figure 2.1: *The graph in the left represents the observed graph and the graph on the right represents the graph after we find the labels of each node, representing a community structure.*

Chapter 3

Algorithms

All the attention that the field of community detection is receiving results in many new algorithms to find community structure in networks that is, selecting the model and estimating the parameters in the graph model. There are many ways to tackle community detection, with divisive algorithms, agglomerative algorithms, or algorithms based on paths or statistics models. In this thesis, we will explain briefly the algorithms used in the simulations in this thesis and not make an overall analysis of all the types of algorithms available.

Next, we will explain five algorithms used in this dissertation, the *Newman-Girvan algorithm* is not based on any statistical model, both *Spectral Clustering algorithm* and *SDP algorithm* are based on the SBM. At the same time, the *Exact ICL algorithm* and *Allocation sampler algorithm* are too based on the SBM but with a bayesian approach.

3.1 Newman-Girvan Algorithm

Also known as *edge-betweenness algorithm*, it is a divisive algorithm based on each node's degree in the graph. It "erases" the nodes with the highest "betweenness" measure in the graph recursively, thus "creating" communities in the graph. The algorithm repeats the calculation of *betweenness* every time it removes a node from the graph. In the end, it uses a measure called *modularity* to select the best division possible. Thus, this algorithm's "real" objective is to maximize the *modularity*.

Let $e_{k\times k}$ be a symmetric matrix with the fractions of all the vertices that connects the community i to community j. Let $a_i = \sum_j e_{ij}$. The modularity is then defined as:

$$Q = \sum_i (e_{ii} - a_i^2)$$

The first part of the algorithm is to assign weights and distances to the nodes in the graph. Consider one node v in the graph:

1. Node v is given distance $d_v = 0$ and weight $w_v = 1$.
2. Every node i adjacent to v is given distance $d_i = d_v + 1$ and weight $w_i = w_v$
3. For each node j adjacent to one of those neighbours in the last step we will do one of this:
    - if j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$;

- if j already have a distance, and $d_j = d_i + 1$, then we increase this node's weight by w_i;
- if j already have a distance, and $d_j < d_i + 1$, we do nothing.

4. Repeat step 3 until no nodes remain without having distances but whose neighbors don't have distance.

After this, we will begin the calculation of the edge-betweenness score:

1. Find every node t that no paths from node v to other nodes go through node t;
2. For each node i adjacent to node t, assign a score to the edge from t to i as w_i/w_t;
3. Starting with farthest node from the node v, go towards v assigning 1 plus the sum of the scores on the neighboring edges; immediately below it, all multiplied by w_i/w_j, where node j is far from v than node i;
4. Repeat the last step until you reach the node v.

This step is just for one node. After doing this for every node in the graph and adding all the scores, you will have the total betweenness of all the edges in the graph.

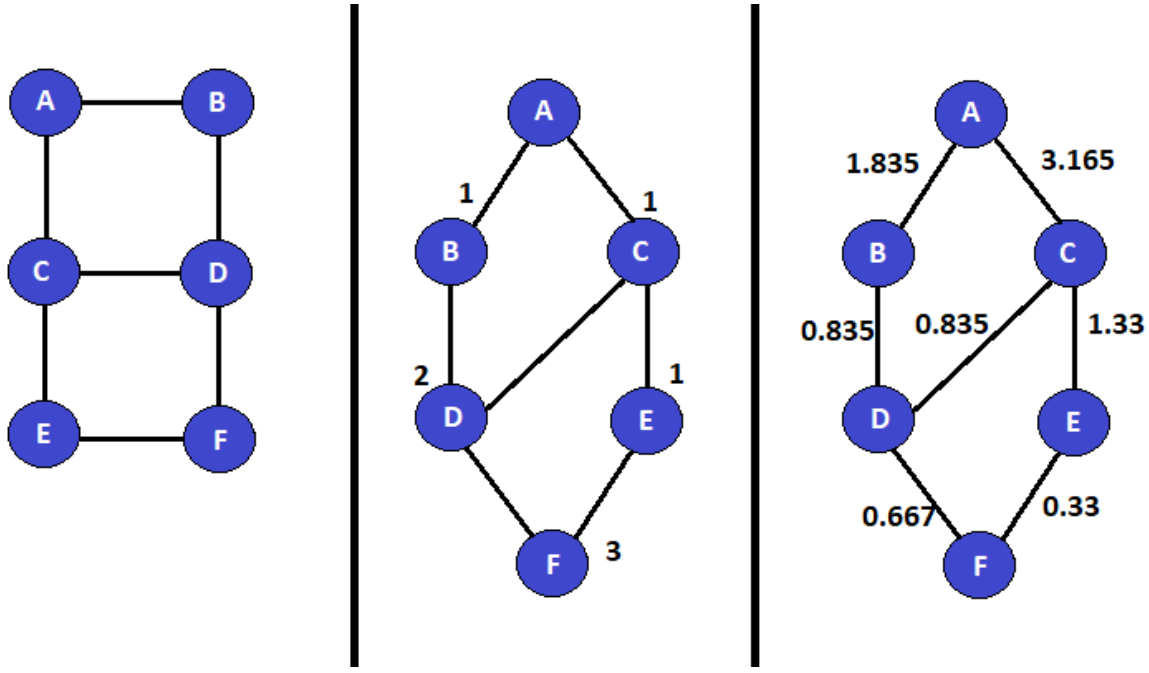

Figure 3.1: *An example of the NG algorithm being applied on the node A. First (middle graph) we go from the node A to the node F giving weights and distances, and in the right graph, we "go back" from node F to node A giving the scores.*

3.2 Spectral Clustering

With a vast literature, *spectral clustering* is one of the most used algorithms for community detection. There are many variations and ways to apply this method; the one used

in this dissertation is the same one presented in Lei and Rinaldo (2015). The choice of this algorithm is because the consistency for it is already proven.

This algorithm consists in applying the k-means method in the matrix $U_{n\times u}$, where the matrix $U_{n\times u}$ is the matrix with u leading eigenvectors of the adjacency matrix A, that is, the eigenvectors related with the u largest eigenvalues.

3.3 SDP algorithm

This algorithm uses the SDP relaxation explained before, in **2.2.4**, and this relaxation can be solved faster than solving the maximum likelihood equation using methods like the interior-point methods. After solving this equation, the vector that we have is not a vector of only $1's$ and $-1's$, so we take the sign function of this vector as our estimation of the communities, that is $x = sign(v)$, where v is the result of the SDP relaxation. In this dissertation, we used the package *CVXR* from software R to solve this problem, where it uses the "best" (chosen by the algorithm) way to solve a semidefinite constraint like the one used here. There are a few ways to solve this type of problem, so we will not explain in detail those methods.

This algorithm, in the way that it is built, only works with two community graphs. Thus we will only use this approach in the first part of the simulation, while the others are used in all simulations. Another point about this algorithm is that it is slower than other algorithms and demands a lot of computer processing, at least with the approach used in this dissertation.

3.4 Allocation Sampler

This algorithm is based on Nobile and Fearnside (2007) and was proposed by McDaid *et al.* (2012). Using priors to the parameters (and hyperparameters), the author uses the concept of *collapsing the SBM* to find a fast way to compute the posterior distribution of the number of clusters and labels at the same time and make inference about them. For this algorithm it has used a discrete or categorical distribution for the labels, that is:

$$z_i \sim CAT(\pi_1, \ldots, \pi_K),$$

a Dirichlet for the hyperparameters,

$$\pi \sim Dirichlet(\alpha_1, \ldots, \alpha_K), \quad \alpha_i = \alpha \ \ \forall i \in \{1, \ldots, K\},$$

and similarly to the KT distribution:

$$x_{ij}|z, K, \pi \sim Bernoulli(p_{z_i z_j})$$

with,

$$p_{kl} \sim Beta(\beta_1, \beta_2).$$

This formulation of the SBM is from Snijders and Nowicki (1997), and like them, we choose $\beta_1 = \beta_2 = 1$.

To collapse the SBM, it is used a Poisson prior for K, the number of clusters, that is, $K \sim Poisson(1)$, this gives us $P(K) \propto \frac{1}{K!}$. We only need a proportion of the expressions

based on z and K only since it is the focus of this algorithm. Then we "separate" the full distribution of the model in a way that our algorithm only needs to search over K and z:

$$\mathrm{p}(x, p, z, \pi, K) = P(K) \times \mathrm{p}(z, \pi|K) \times \mathrm{p}(x, p|z),$$

Consider Π and Θ the domains of π and p,

$$P(x, z, K) = P(K) \times \int_{\Pi} \mathrm{p}(z, \pi|K) d\pi \times \int_{\Theta} \mathrm{p}(x, p|z) dp,$$

using the block-by-block independence $x_{(kl)}|z, K$, we have

$$P(x, z, K) = P(K) \times \int_{\Pi} \mathrm{p}(z, \pi|K) d\pi \times \prod_{k,l} \int_{\Theta} \mathrm{p}(x_{(kl)}, p_{kl}|z) dp_{kl}.$$

Thus, we get the full posteriori expression as:

$$P(x, z, K) \propto \frac{1}{K!} \times \frac{\Gamma(\alpha K) \prod_{i=1}^{K} \Gamma(p_i + \alpha)}{\Gamma(\alpha)^K \Gamma(N + \alpha K)} \times \prod f(x_{kl}|z).$$

The function $f(x_{kl}|z)$ is given by

$$f(x_{kl}|z) = \frac{B(\beta_1 + O_{kl}, n_{kl} - O_{kl} + \beta_2)}{B(\beta_1, \beta_2)},$$

where O_{kl} and n_{kl} are the same defined in chapter 2, and $B(\cdot)$ is the beta function.

The algorithm will sample from the posteriors z and $K|x$ using four moves. Each iteration will choose one of those moves with the same probability and execute them. The four moves are:

- MK: A metropolis-hastings to add or remove an empty cluster.
- GS: Gibbs sampling on one single node, it will select a new cluster for only this node selected randomly.
- M3: It will choose two randomly clusters and reassign all the nodes in these clusters using a movement described in (Nobile and Fearnside (2007)).
- AE: This move will choose between merging two existing clusters or split one randomly selected cluster.

The name of each move, *MK*, *GS*, *M3* and *AE*, are names used by the creators of this algorithm, and we will continue to use them.

All the moves are a Metropolis-Hastings move, that is, given a state z and K, the algorithm will modify and propose a new state for $z^{'}$ or $K^{'}$, or both. The new move is accepted with probability:

$$min\left(1, \frac{P(x, z^{'}, K^{'})}{P(x, z, K)} \times \frac{P_{prop}((K^{'}, z^{'}) \to (K, z))}{P_{prop}((K, z) \to (K^{'}, z^{'}))}\right),$$

where $P_{prop}((K, z) \to (K^{'}, z^{'}))$, is the transition probability from state (K, z) to state $(K^{'}, z^{'})$.

The algorithm initializes by setting K =2 and randomly giving labels to every node in the graph. Next, the four moves are explained as in the original article.

3.4.1 MK

This move increases or decreases the number of clusters (only empty clusters), that is $z^{'} = z$ in all moves since only the number of clusters will change. The proposal probabilities if you add a cluster is:

$$P_{prop}((K, z) \to (K+1, z^{'})) = \frac{0.5}{K+1}$$

$$P_{prop}((K^{'}, z^{'}) \to (K-1, z)) = \begin{cases} \frac{0.5}{K^{'}} \text{ if } K^{'} > 1 \\ 0 \quad \text{otherwise.} \end{cases}$$

$$\frac{P(x, z^{'}, K^{'})}{P(x, z, K)} = \frac{\Gamma(\alpha(K+1))\Gamma(N+\alpha K)}{(K+1)\Gamma(\alpha K)\Gamma(N+\alpha(K+1))}$$

3.4.2 GS

This move selects a random node, and it will reassign a new cluster to it. Could it be the same cluster that the node is part of, or could it be any other cluster possible. This move only changes one node, and it does not change the number of clusters K. The acceptance probability is chosen proportional to $P(x, z^{'}, K)$, where $z^{'}$ is the new cluster assignment for the selected node.

This move is complex to do since it changes many factors in $P(x, z, K)$. Because of this, the algorithm only uses neighbors of the selected node to do all the calculation changes, that is, it will not test all the possible clusters for the selected node, avoiding problems with time when applying this method in larger networks.

3.4.3 M3

This move's name is from Nobile and Fearnside (2007), it will select two clusters j and k, remove all the nodes from those two clusters, put in a ordered list $A = (a_1, \ldots, a_{n_j+n_k})$. For each node in A it will assign to one of those two clusters, and this new assignments will be placed in a list $B_i = (b_1, \ldots, b_{i-1})$ where B_i means the assignment done at the iteration i of this move.

The nodes on list A are assigned to one of the two clusters satisfying

$$p_{B_i}^{a_i \to j} + p_{B_i}^{a_i \to k} = 1.$$

Those probabilities are conditional on the nodes already assigned, and any probability can be chosen, as long as it's not zero and sum to one. After all nodes have a new assignment, the proposal probability is:

$$P_{prop}(z \to z^{'}) = \prod_{i=1}^{n_j+n_k} p_{B_i}^{a_i \to b_i},$$

and,

$$P_{prop}(z^{'} \rightarrow z) = \prod_{i=1}^{n_j+n_k} p_{B_i^{'}}^{a_i \rightarrow b_i^{'}},$$

where $B^{'} = z_{a_1}, \ldots, z_{a_{i-1}}$. For the ratio probabilities, it's the same used in Nobile and Fearnside (2007), that is:

$$\frac{p_{B_i}^{a_i \rightarrow j}}{p_{B_i}^{a_i \rightarrow k}} = \frac{P(x^{'}, z_{\{a_i \rightarrow j, B_i\}}, K)}{P(x^{'}, z_{\{a_i \rightarrow k, B_i\}}, K)},$$

in this equation all the unassigned nodes and the edge of these nodes are ignored.

3.4.4 AE

This move will choose between split or merge two clusters, If it chooses to split, it will select one cluster, and if it chooses to merge, two clusters will be selected at random.

In the split move, for each node in the selected cluster, there will be a probability p_E of the node to be assigned to the new cluster, called *probability of ejection*, and it is chosen at random from a Uniform$(0, 1)$ distribution. The proposal probability is dependent on p_E, but the author decides to integrate over p_E. Similar to collapsing the SBM, then the transition probability of a given state (z, K) going to the new state $(z^{'}, K^{'} = K + 1)$ is

$$P_{prop}((z, K) \rightarrow (z^{'}, K^{'})) = \frac{\Gamma(n_1 + 1)\Gamma(n_2 + 1)}{K(K + 1)\Gamma(n + 2)},$$

where n is the number of nodes in the original cluster, and n_1, n_2 are the number of nodes in each new cluster after the split.

In the merge move, one of the clusters will maintain all the nodes and will receive the other nodes of the second cluster, and the transition probability is simply

$$P_{prop}((z, K) \rightarrow (z^{'}, K - 1)) = \frac{1}{K(K + 1)}$$

3.5 Exact ICL algorithm

This algorithm was created by Côme and Latouche (2013) based on Daudin *et al.* (2008) and is an alternative to the *allocation sampler* algorithm. Similar to McDaid *et al.* (2012), and the KT distribution, the SBM is defined as:

$$\pi_k = P(Z_{ik} = 1) \quad \text{with} \quad \sum_{k=1}^{K} \pi_k = 1.$$

Here Z represents the labels of each node. However, instead of being a vector with the number of the community that the node belongs, it is a matrix where the columns represent the communities, and the lines represent the nodes, in each line has a value 1 in the column representing the community of the node, and the value 0 in all the other columns. This does not change anything from the theoretical point of view, but the authors did this for computational calculations, and K is an upper case to differentiate the number of communities that we have in the algorithm and the community k that the algorithm will select in each step. The edges are given by:

$$X_{ij}|Z \sim Bernoulli(p_{kl}) \text{ for } i \neq j.$$

This gives:

$$p(Z|\pi) = \prod_{i=1}^{N} \prod_{k=1}^{K} \pi_k^{Z_{ik}},$$

and

$$p(X|Z,p) = \prod_{i \neq j}^{N} \prod_{k,l}^{K} \left(p_{kl}^{X_{ij}} (1-p_{kl})^{1-X_{ij}}\right)^{Z_{ik}Z_{jl}}.$$

For π we use a Dirichlet prior, that is,

$$\pi \sim Dirichlet(\alpha^0 = (\alpha_1^0, \ldots, \alpha_K^0)),$$

For p we have

$$p_{kl} \sim Beta(\eta_{kl}^0, \zeta_{kl}^0).$$

The integrated complete data log likelihood, or ICL, can be decomposed as:

$$\begin{aligned} ICL(Z,K) = log\ \mathrm{p}(X|Z,K) &= log\left(\int_{\pi,p} \mathrm{p}(X,Z,p,\pi|K)d\pi dp\right) \\ &= log\left(\int_p \mathrm{p}(X|Z,p,K)\mathrm{p}(p|K)dp \int_\pi \mathrm{p}(Z|\pi,K)p(\pi|K)d\pi\right) \\ &= log\ \mathrm{p}(X|Z,K) + log\ \mathrm{p}(Z|K) \end{aligned}$$

Usually, this equation does not have analytical form, but with using the chosen priors, both distributions $log\ p(X|Z,K)$, and $log\ p(Z|K)$ are known, thus making it possible to calculate the exact ICL to make inference about the clusters and labels.

The calculation of the ICL already penalizes the number of clusters, so the model complexity, in theory, will not be a problem since it will not estimate a higher number of communities. To compute this algorithm, the author uses a greedy algorithm to calculate the ICL multiple times and select an optimum K and Z simultaneously to see all the steps of the Exact algorithm, see Côme and Latouche (2013).

With the priors already given, the only thing that we need to do is give a K_{up} that is, the upper bound for K, or the maximum number of communities that your model will have, and Z, the labels of each node of the graph. K_{up} usually is known by the researchers that know about the Data, while the common approach for Z is to use another algorithm like *spectral clustering* for the initial guess since the algorithm is fast and can provide a good initial labeling for the labels.

The algorithm will select randomly every node (one by one) in the graph and will maximize an equation $\Delta_{g\to h}$, $\forall h \neq g$, that is, it will see if the ICL will increase if we change the label of the selected node, this equation, as some of the other calculations that the algorithm does will be explained next. After making the calculations for every single node in the graph, one by one, the algorithm makes calculations of the ICL based on merging some clusters to see if the ICL will increase.

The algorithm will calculate $\Delta_{g\to h}$ for every node in the graph, and each iteration will do the calculations for only one node. The term $\Delta_{g\to h}$ is defined as:

$$\Delta_{g\to h} = ICL_{ex}(Z^{'}, K^{'}) - ICL_{ex}(Z, K),$$

the ICL is given by

$$ICL_{ex}(Z,K) = \sum_{k,l}^{K} log\left(\frac{\Gamma(\eta_{kl}^0+\zeta_{kl}^0)\Gamma(\eta_{kl})\Gamma(\zeta_{kl})}{\Gamma(\eta_{kl}+\zeta_{kl})\Gamma(\eta_{kl}^0)\Gamma(\zeta_{kl}^0)}\right) + log\left(\frac{\Gamma(\sum_{k=1}^{K}\alpha_k^0)\prod_{k=1}^{K}\Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K}\alpha_k)\prod_{k=1}^{K}\Gamma(\alpha_k^0)}\right),$$

where

$$\alpha_k = \alpha_k^0 + \sum_{i=1}^{N} Z_{ik},\ \forall k \in \{1,\dots,K\},$$

$$\eta_{kl} = \eta_{kl}^0 + \sum_{i\neq j}^{N} Z_{ik}Z_{jl}X_{ij},\ \forall (k,l) \in \{1,\dots,K\}^2,$$

and

$$\zeta_{kl} = \zeta_{kl}^0 \sum_{i\neq j}^{N} Z_{ik}Z_{jl}(1-X_{ij}),\ \forall (k,l) \in \{1,\dots,K\}^2.$$

The equation $\Delta_{g\to h}$ takes two different forms depending on the cluster g that the selected node is part of, if g is empty after the changing of the node's label. Consider that the node i was selected:

Cluster g still have nodes after removal of i

If after changing the label of the node i, cluster g still have nodes, that is, $\sum_i Z_{ig}^{'} > 0$ the equation used is:

$$\Delta_{g\to h} = log\left(\frac{\alpha_h}{\alpha_g - 1}\right) + \sum_{l=1}^{K}\sum_{k\in\{g,h\}} log\left(\frac{B(\eta_{kl}+\delta_{kl}^{(i)}, \zeta_{kl}+\rho_{kl}^{(i)})}{B(\eta_{kl},\zeta_{kl})}\right) + \sum_{k\notin g,h}\sum_{lin\{g,h\}} log\left(\frac{B(\eta_{kl}+\delta_{kl}^{(i)}, \zeta_{kl}+\rho_{kl}^{(i)})}{B(\eta_{kl},\zeta_{kl})}\right.$$

with

$$\delta_{kl}^{(i)} = \mathbb{I}_{\{k=h\}}\sum_{j\neq i}^{N} Z_{jl}X_{ij} + \mathbb{I}_{\{l=h\}}\sum_{j\neq i}^{N} Z_{jk}X_{ji} - \mathbb{I}_{\{k=g\}}\sum_{j\neq i}^{N} Z_{jl}X_{ij} - \mathbb{I}_{\{l=g\}}\sum_{j\neq i}^{N} Z_{jk}X_{ji},$$

and

$$\rho_{kl}^{(i)} = \big(\mathbb{I}_{\{k=h\}} - \mathbb{I}_{\{k=g\}}\big)(\alpha_l - \alpha_l^0 - Z_{il}) + \big(\mathbb{I}_{\{l=h\}} - \mathbb{I}_{\{l=g\}}\big)(\alpha_k - \alpha_k^0 - Z_{ik}) - \delta_{kl}^{(i)}.$$

Cluster g is empty after the removal of i

If after removing node i from cluster g, this cluster becomes empty ($\sum_i Z_{ig} = 0$), the equation is defined as

$$\Delta_{g\to h} = log\left(\frac{\alpha_h}{\alpha^0}\frac{\Gamma((K-1)\alpha^0)\Gamma(K\alpha^0+N)}{\Gamma(K\alpha^0)\Gamma((K-1)\alpha^0+N)}\right) + \sum_{\substack{(k,l)\neq g \\ k=h \text{ or } l=h}} log\left(\frac{B(\eta_{kl}+\delta_{kl}^{(i)},\zeta_{kl}+\rho_{kl}^{(i)})}{B(\eta_{kl},\zeta_{kl})}\right) + \sum_{k=g \text{ or } l=g} log\left(\frac{B(\eta_{kl}^0,\zeta_{kl}^0)}{B(\eta_{kl},\zeta_{kl})}\right).$$

Merge movement

After making all the calculations using every node in the graph, we can start to look at full cluster and see if they can be put together. If there is a merge between two clusters, the equation used to calculate the ICL is given by

$$\Delta_{g\cup h} = log\left(\Gamma(\alpha^0)\frac{\Gamma((K-1)\alpha^0)\Gamma(K\alpha^0+N)\Gamma(\alpha_h+\alpha_g-\alpha^0)}{\Gamma(K\alpha^0)\Gamma((K-1)\alpha^0+N)\Gamma(\alpha_g)\Gamma(\alpha_h)}\right) + \sum_{\substack{(k,l)\neq g \\ k=h \text{ or } l=h}} log\left(\frac{B(\eta_{kl}+\delta_{kl}^{(i)},\zeta_{kl}+\rho_{kl}^{(i)})}{B(\eta_{kl},\zeta_{kl})}\right) + \sum_{k=g \text{ or } l=g} log\left(\frac{B(\eta_{kl}^0,\zeta_{kl}^0)}{B(\eta_{kl},\zeta_{kl})}\right).$$

In this case, $\delta_{kl}^{(i)}$ and $\rho_{kl}^{(i)}$ are defined as

$$\delta_{kl}^{(i)} = \mathbb{I}_{\{k=h\}}(\eta_{gl}-\eta_{gl}^0) + \mathbb{I}_{\{l=h\}}(\eta_{kg}-\eta_{kg}^0) + \mathbb{I}_{\{l=h \text{ and } k=h\}}(\eta_{gg}-\eta_{gg}^0)$$

$$\rho_{kl}^{(i)} = \mathbb{I}_{\{k=h\}}(\zeta_{gl}-\zeta_{gl}^0) + \mathbb{I}_{\{l=h\}}(\zeta_{kg}-\zeta_{kg}^0) + \mathbb{I}_{\{l=h \text{ and } k=h\}}(\zeta_{gg}-\zeta_{gg}^0)$$

Chapter 4

Simulation

In this section, we tested the algorithms explained in chapter 3 in different situations and then compared them to see if there is a better one. We used a few different probabilities for each graph, having more sparse graphs and more dense ones, and also different sizes for the simulated graphs to test if this would change the precision of the algorithms.

The simulations were done on a 3.6 GHz AMD Ryzen 5 2600X with 8GB RAM. Memory was a problem when applying the SDP algorithm since it could not handle larger graphs. All the simulations were made in R, besides the allocation sampler, since the author released the algorithm in another language. However, the same matrix created for the other algorithms was used in the allocation sampler.

For two communities the algorithms *Allocation Sampler* (**AS**), *Exact ICL* (**Exact ICL**), *Newman-Girvan Modularity* (**NG**), *Semidefinite Programming* (**SDP**), *Spectral Clustering* (**Spectral**), were used, while for more communities we didn't use the **SDP**. No algorithm was made by the author of this dissertation, instead every algorithms used here is available, or by the authors that created the algorithm (showed in the references) or by the package *igraph* from the software R, used here for the **NG** algorithm.

4.1 Simulation with two communities

The models used here are in the form $SBM(n, k, a * \frac{log(n)}{n}, b * \frac{log(n)}{n})$, we used three types of values for the probabilities, always satisfying one of the limits:

- Inside both limits $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and $(a-b)^2 > (8(a+b) + (8/3)(a-b)$,
- Inside the limit $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and not on $(a-b)^2 > (8(a+b) + (8/3)(a-b)$, that is: $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and $(a-b)^2 < (8(a+b) + (8/3)(a-b)$,
- Not on any limit, that is $|\sqrt{a} - \sqrt{b}| < \sqrt{2}$.

For each of the above items, we simulated with a graph of a fixed size of 100 nodes and fixed a, while changing b, and a model with fixed a and b, and an increasing size of the network, going from 100 to 300. This was done to test how the algorithms will behave in the known limits presented in Abbe's article, and increasing the size of the graph, maintaining the other parameters fixed will make the graph sparser as the graph grows.

The limits used here are fundamental limits in exact recovery, were $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ is a known limit when using maximum likelihood estimation, and $(a-b)^2 > (8(a+b)+(8/3)(a-b)$

was proposed by Abbe *et al.* (2014c) when using an SDP algorithm, and because of this we decided to test the algorithms in different situations with those limits. These limits are used only in balanced communities. Thus, even when increasing the sizer of the graph, it will always be balanced communities when working with two communities.

For each value of b (or size n), we created 100 graphs and applied the algorithms on each one of them. Then we took the mean of how much each algorithm got the correct nodes, that is, the number of nodes estimated in the right communities divided by the number of nodes in the graph, that is, similar to the agreement measure explained in chapter 2. We consider almost all the permutations of labels here, but, in this simulations with two communities only, estimating more than two communities were heavily penalized, and this particularly is a problem for the **Exact ICL** since, as our simulations will show, the algorithms overestimate the real number of communities in many simulations, and because of this if algorithms estimated a number large compared to the real number of communities (if estimated five or more communities), not all relabeling were tested.

The difference between the *agreement* and the measure that we call *precision* here is that the *agreement* consider that the algorithms got the real number of communities. Here we will consider the relabeling even if the algorithm got the wrong number of communities. The rest of the calculations are exactly like the *agreement*.

4.1.1 $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and $(a-b)^2 > (8(a+b) + (8/3)(a-b)$

As we can see in (**Figure 4.1**), with fixed a, for the value of a = 3, giving a probability of 0.13 of nodes in the same community having an edge, the **Exact ICL** algorithm and the **Spectral Clustering** was the two algorithms that had the worst performance between all algorithms used. Still, no algorithm got lower precision than 80%. With a higher value of a, having a probability of an edge between nodes of the same community equal to 0.92, the **Exact ICL** algorithm performed better, always having a precision above 90%, but still being the algorithm with the worst results in general, with the spectral clustering being the other algorithm with a lower precision at some values of B. The **SDP** algorithm was the slowest one but had a high consistency of estimating well the communities of each node. The **NG** and the **AS** algorithms were the ones that performed better, having near-perfect precision in all samples.

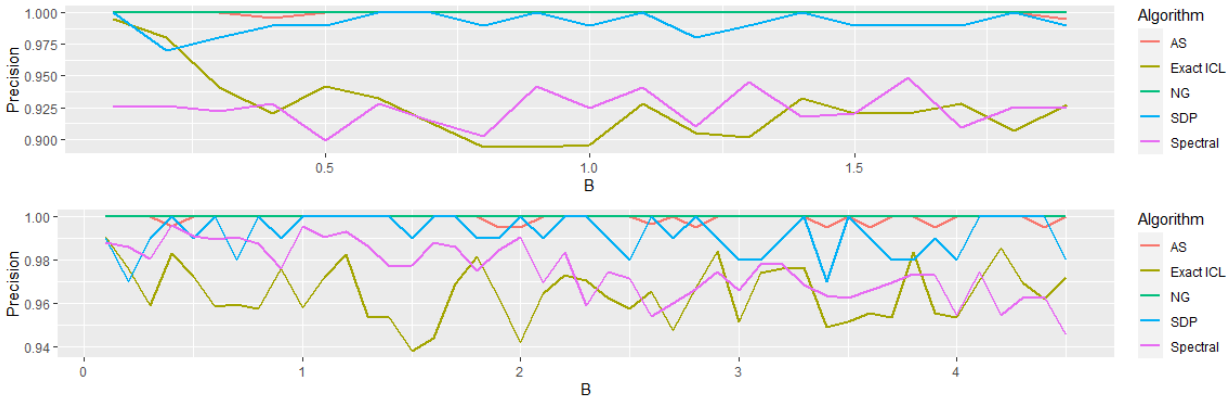

Figure 4.1: *Precision of algorithms when changing the term b in the SBM. In the upper graph we have a = 5. and in the bottom one a = 20. Both have n = 100.*

When increasing the size of the graphs (**Figure 4.2**), while maintaining the terms a and b fixed, the graph becomes sparser as n goes higher. However, most algorithms did

not perform differently, maintaining a high precision as the graph grew sparser. In these simulations, **Exact ICL** started to perform worse each time that the graph increased, again probably having a problem dealing with sparse graphs, probably due to estimating more than two communities when the graph becomes sparser. All the other algorithms, besides the **Spectral**, had again an almost perfect precision in those simulations.

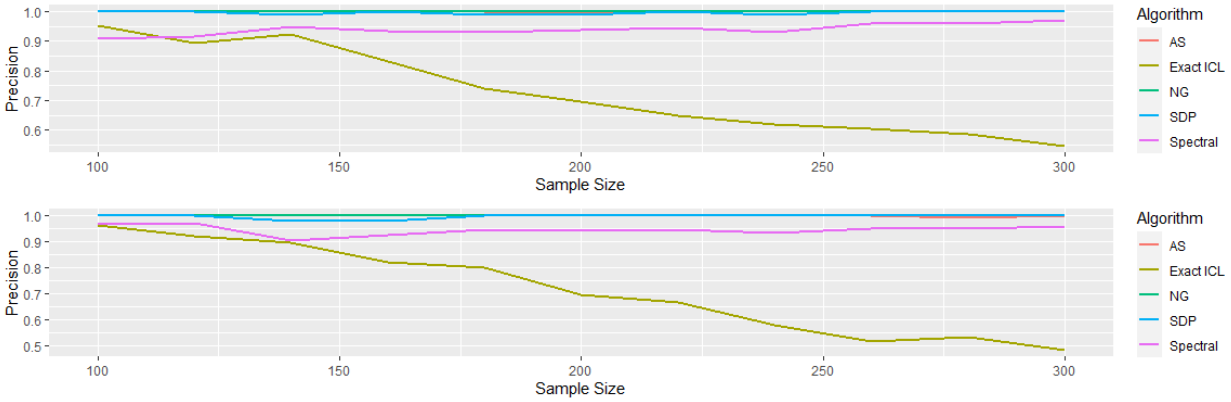

Figure 4.2: *Precision of algorithms when increasing the size of the graph. In the upper graph we have $a = 11$ and $b = 1.3$. and in the bottom one $a = 20$ and $b = 4$.*

4.1.2 $|\sqrt{a} - \sqrt{b}| > \sqrt{2}$ and $(a-b)^2 < (8(a+b) + (8/3)(a-b)$

When fixing a and the size of the graph (**Figure 4.3**), with a low value of a the **Exact ICL** had the worst result, and overall bad results, suggesting that it is not a good option to use this algorithm in this interval of parameters. For a higher value of a, the **Spectral** was the algorithm that had the worst result but still having a precision above 90%. All the other three algorithms had precision above 95%.

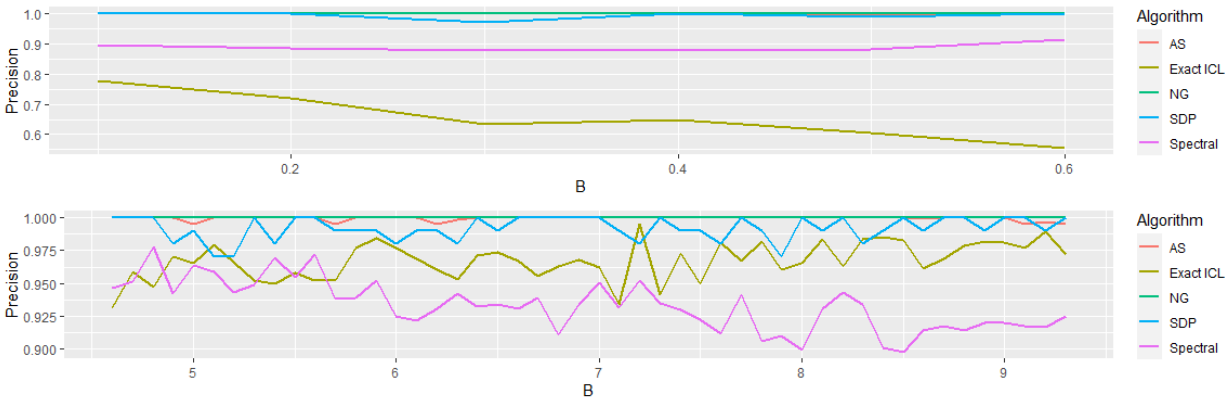

Figure 4.3: *Precision of algorithms when changing the term b in the SBM. In the upper graph we have $a = 5$. and in the bottom one $a = 20$. Both have $n = 100$.*

When fixing a and b (**Figure 4.4**), we decide in both simulations to use small values of a and b, and again the **Exact ICL** had problems estimating the labels as the size of the graph increased. The **Spectral** algorithm had a lower precision than the other three algorithms but did not have any precision below 80%.

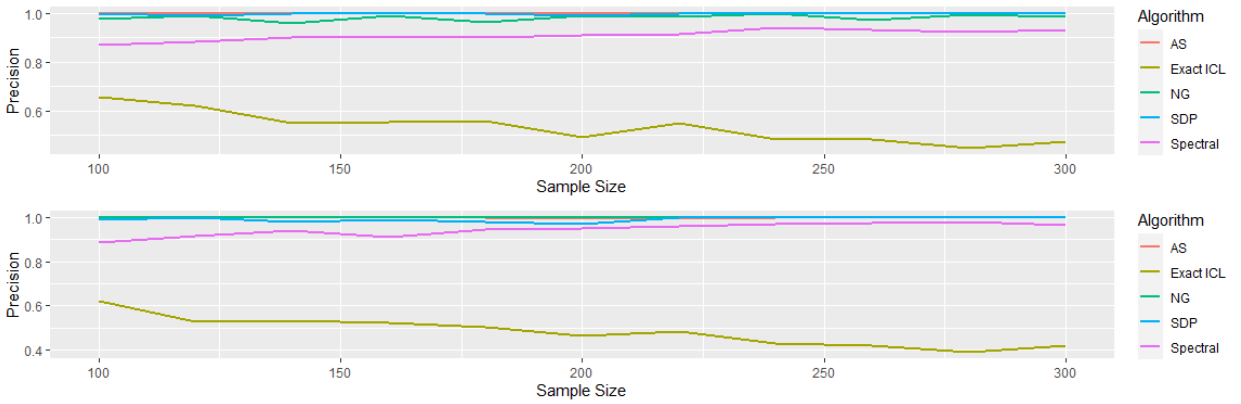

Figure 4.4: *Precision of algorithms when increasing the size of the graph. In the upper graph we have $a = 3$ and $b = 0.1$. and in the bottom one $a = 7$ and $b = 1$.*

4.1.3 $|\sqrt{a} - \sqrt{b}| < \sqrt{2}$

In these simulations, we should expect for the algorithms not to be able to get consistent high precision in the simulations, but besides the **Exact ICL** algorithm when $a = 5$, and the spectral clustering that did not get high precision, all the algorithms got above 95% precision in most of the simulations. When $a = 20$ the **NG** and **SDP** have a poor performance when the value of the parameter b(B in the image) get near the parameter a, but that can be explained because the model becomes unidentifiable in this situation.

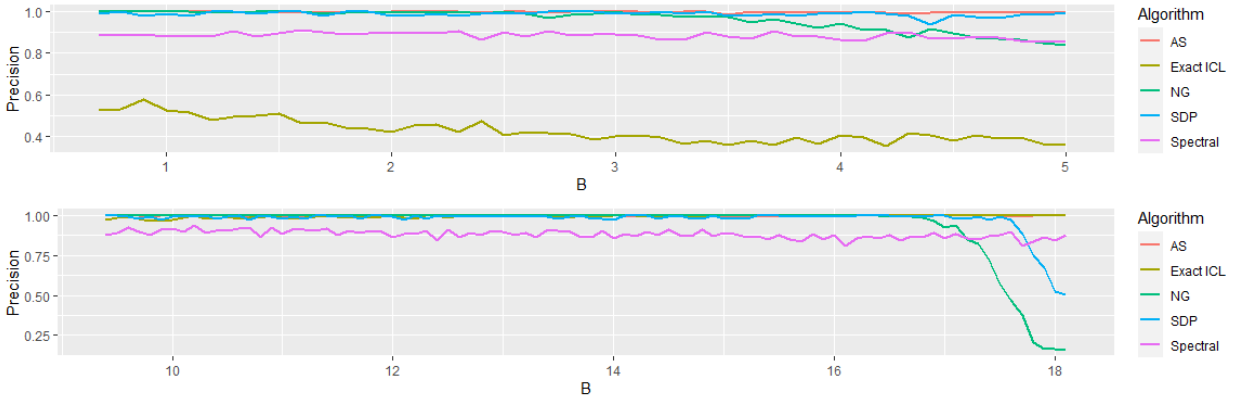

Figure 4.5: *Precision of algorithms when changing the term b in the SBM. In the upper graph we have $a = 5$. and in the bottom one $a = 20$. Both have $n = 100$.*

When fixing both probabilities and increasing the size of the graph (**Figure 4.6**), the **Exact ICL** got the worse results in both cases, and the **Spectral** algorithm again had lower results when compared to the other three algorithms that had almost near perfection results.

It seems that the **Exact ICL** have problems estimating the correct number of communities and labels when the graph has two communities only, and this is probably because this algorithm overestimates the real number of communities many times, as we will see in the next section.

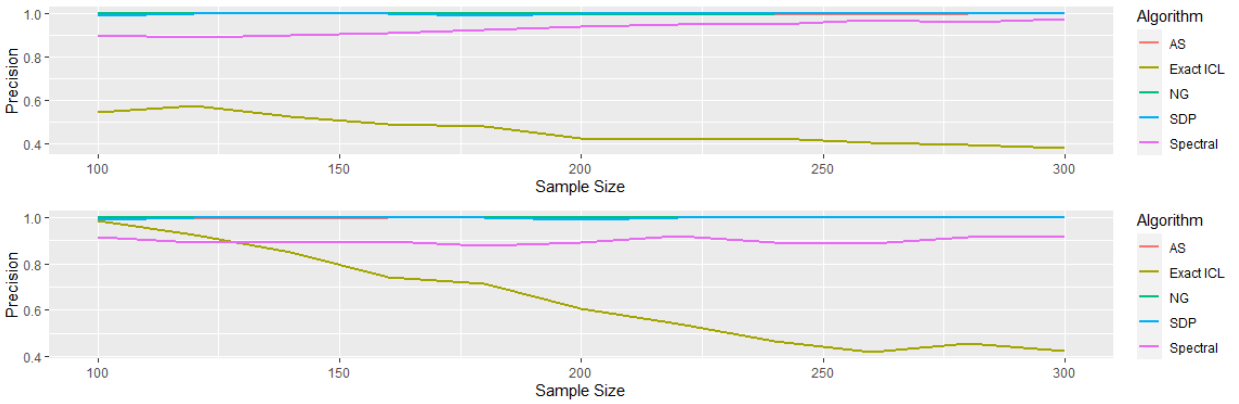

Figure 4.6: *Precision of algorithms when increasing the size of the graph. In the upper graph we have $a = 5$ and $b = 0.7$. and in the bottom one $a = 10$ and $b = 5$.*

4.2 Simulations with more than two communities

In this section, we work with more than two communities ($K = 4, 5, 6, 7$), and unlike when working with two communities, we will not focus on the different regimes of probabilities. Instead, we will change the probabilities and see if the algorithms can achieve exact recovery with these changes.

In the first part, like in Côme and Latouche (2013), we fix q, that is, the probability of having an edge between nodes of different communities, and we increase the value of p, the probability of having an edge between nodes from the same community. In this part of the algorithm, for simplicity, we consider the SBM in the form $SBM(k, n, p, q)$. Like in the simulations with two communities, we also fix both probabilities p and q and increase the size of the graph n, for these simulations, we consider again the SBM in the form $SBM(n, k, a * \frac{log(n)}{n}, b * \frac{log(n)}{n})$ because we want the graph to become sparser as it increases the number of nodes.

In **Figure 4.7** shows the simulations done with q fixed at 0.01 and a graph with 100 nodes. We started the probability of connection between nodes of the same edges p at 0.07 and increased the value 0.03 each time until p was equal to 0.82, that is, we started with a model that is almost impossible to identify and finished with a model with a considerable difference between probabilities, being easier in theory to identify the communities. As we can see in the graph, the **Spectral** was never able to estimate the communities fully, while the other algorithms at some point got the exact communities of each node. The **NG** and **Exact ICL** algorithms were the best ones in these simulations, having good results with minimal values of p. In contrast, the **AS** algorithm was only able to increase his precision at higher values of p.

In all simulations, we made bar-plots of the number K estimated, that is, the number of communities estimated. For every parameter or size of the graph, we estimated 100 different graphs, and the bar-plots consist of the number that each algorithm estimated K communities. We did not use the **Spectral** algorithm here since he does not estimate the number of communities. All the label changes were tested when looking at the precision. However, the **Exact ICL** sometimes estimated a high number of communities. When the algorithm estimated more than three communities above the real number, not all the label permutations were used. The precision is probably lower than it should be if all permutations were tested if this happens.

In **Figure 4.8** we see that while the **AS** underestimate the right number of communities, the **Exact ICL** and **NG** consistent overestimate the right number of communities. The **AS**

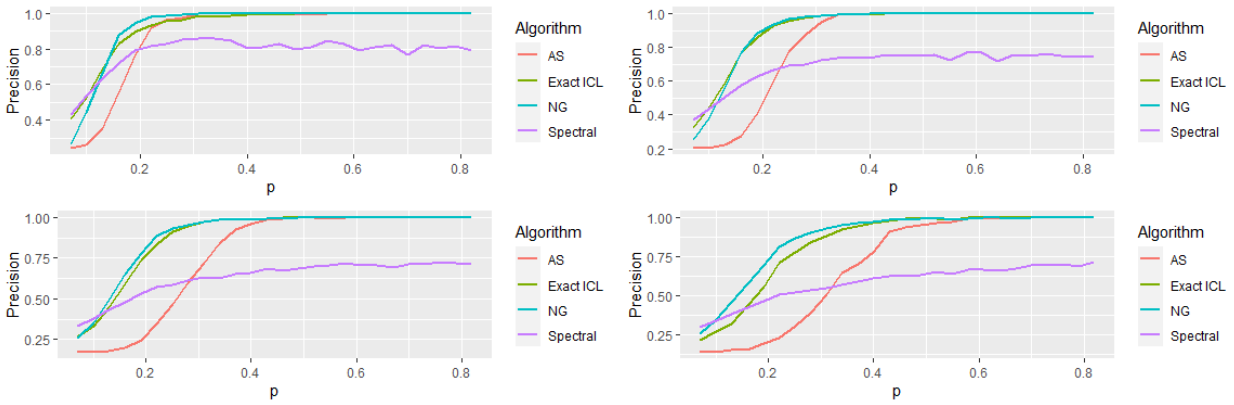

Figure 4.7: *Precision of algorithms when increasing p with $q = 0.01$. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $k = 6$ and bottom right $k = 7$. All graphs have $n = 100$.*

only got the right number of communities more times than the other algorithms when $K = 4$ while being equal with $K = 5$ and having worse results when there were 6 and 7 communities in those graphs.

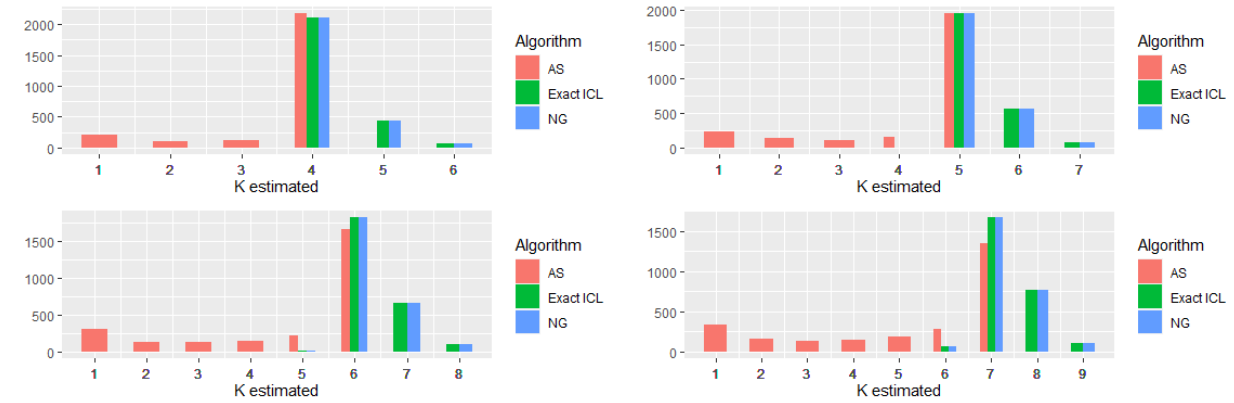

Figure 4.8: *Number of times that each algorithm estimated K when increasing p with $q = 0.01$. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $k = 6$ and bottom right $k = 7$. All graphs have $n = 100$.*

With a value of q equal to 0.1, that is, a probability of connection between two nodes of different communities equal to 10%, we intended to have a less sparse graph and see if this would change something in the estimation. In these simulations, p will start in 0.17 and again will increase 0.03 each time until p reaches 0.82.

In **Figure 4.9** we can see that for a graph with more nodes, the algorithms have worse results for low values of p, needing higher values (higher than the simulations with $q = 0.01$) to get higher precision, and the only algorithm to increase the precision earlier was the **Spectral** algorithm. However, while his precision gets better earlier than other algorithms, he does not achieve at any point precision higher than 90%. The **AS** and **Exact ICL** algorithms look like the ones with better precision overall in the simulations.

In **Figure 4.10**, in all cases, the **Exact ICL** and the **NG** algorithms got the wrong number of communities more than getting it right. While the **AS** algorithm only had this poor results when $K = 7$, but still had problems estimating the correct number of communities.

When doing simulations increasing the size of the graph, it begins with graphs of 100 nodes, and we increase the size of the graph by 25 nodes until it reaches the size 400, that

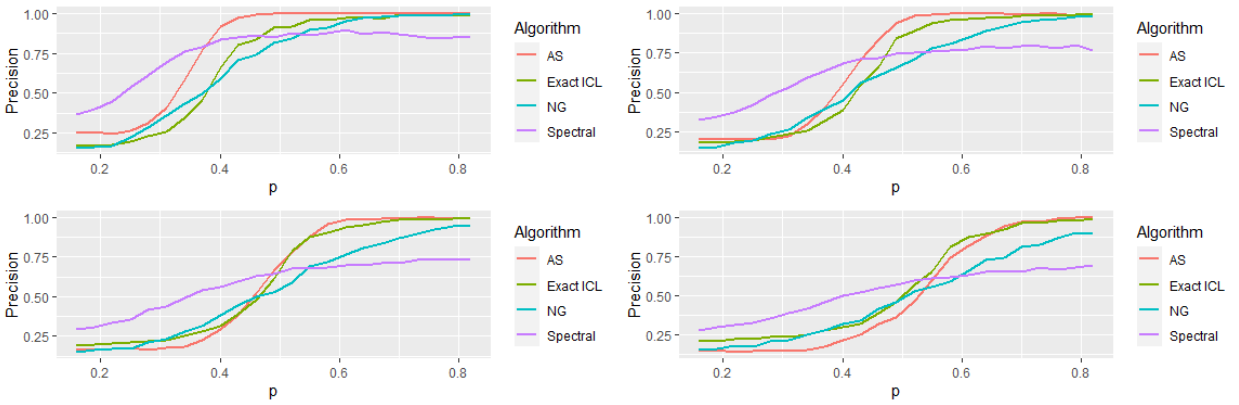

Figure 4.9: *Precision of algorithms when increasing p with $q = 0.1$. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $K = 6$ and bottom right $K = 7$. All graphs have $n = 100$.*

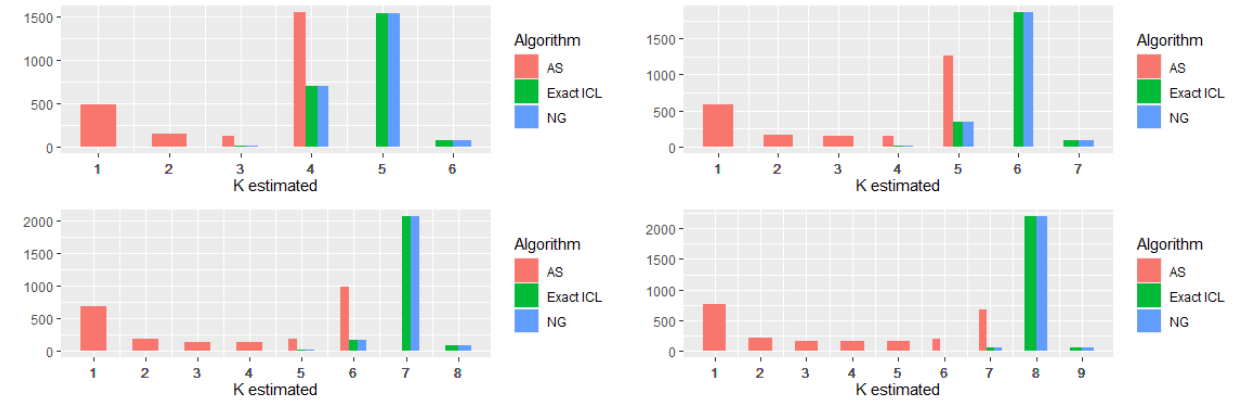

Figure 4.10: *Number of times that each algorithm estimated K when increasing p with $q = 0.1$. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $k = 6$ and bottom right $k = 7$. All graphs have $n = 100$.*

is, $n = \{100, 125, 150, \ldots, 400\}$. We fixed $a = 13$ and $b = 2$, that gives us a probability p, where $p = a * \frac{log(n)}{n}$, of almost 60% with a graph of size 100, and a probability a little lower than 20% when the graph reaches the maximum size in the simulations, and a probability $q = b * \frac{log(n)}{n}$ of almost 10% in the beginning, and a probability near 3% when the graph reaches 400 nodes. As the graph increases, it will become sparser and, in theory, harder to estimate the right communities.

In **Figure 4.11** we can see that almost all the algorithms followed the "trend", having lower precision when the size of the graph got bigger, and the number of edges got lower. For lower values of K the **AS** algorithm got excellent results while the other algorithms could not maintain a precision above 90%. But for a higher value of K, with 7 communities, the **Exact ICL** got better results being consistent in estimate the right communities of each node, while the **Spectral** algorithm, even having always the right number of communities, could not have a high precision, and the **NG** algorithm only had good results when $K = 4$.

The following figure (**Figure 4.9**) shows us that for a lower number of communities ($k = 4$, $k = 5$ and $k = 6$), the **AS** algorithm got better results estimating most of the times the correct number of communities, while with $K = 7$, the AS was not able to keep the same good results, underestimating many times the right number of communities. The **NG** and **Exact ICL** algorithms had poor results overall, almost always estimating the wrong number of communities.

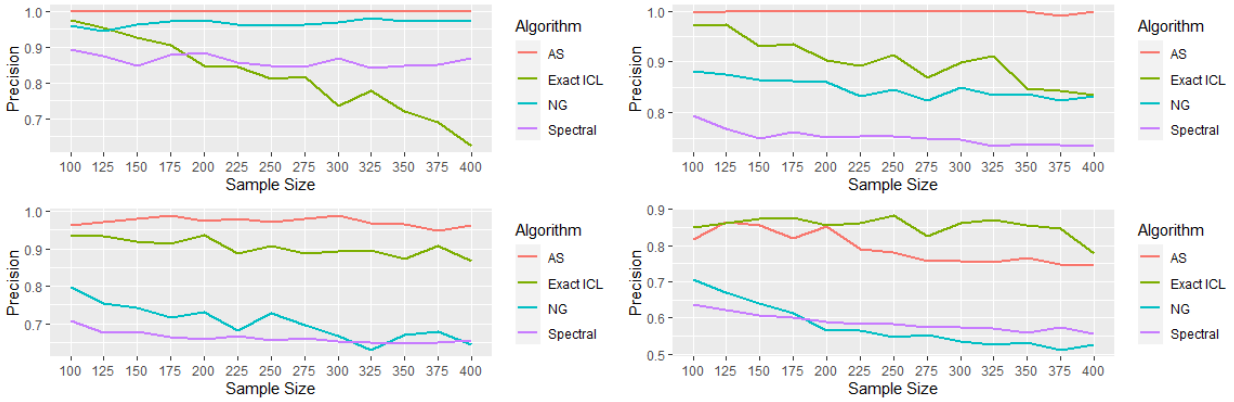

Figure 4.11: *Precision of algorithms when increasing the size of the graph. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $K = 6$ and bottom right $K = 7$.*

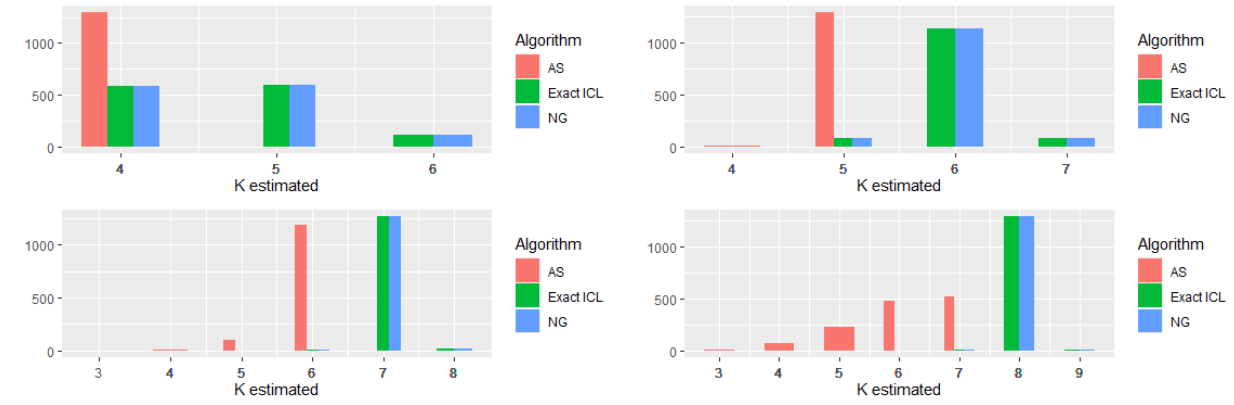

Figure 4.12: *Precision of algorithms when increasing the size of the graph. In the top left we have 4 communities in the graph, 5 communities in the top right, in the bottom left we have $K = 6$ and bottom right $K = 7$.*

Chapter 5

Conclusion

In the simulations where the graphs had two communities, we saw that, besides having good results, the SDP algorithm is not the best choice since other algorithms achieve similar results with a low computational cost. We saw that the allocation sampler had better results overall than the exact ICL algorithm, despite having similar approaches. The Spectral algorithm and the Newman-Girvan modularity continues to be a good choice of algorithm depending on the graph that you will study since they had good performance when using two communities. However, it seems that there are better options in the case of the Spectral clustering.

Besides the exact ICL algorithm, all the other algorithms could estimate exactly the nodes in each community in all regimes tested, meaning that they could achieve exact recovery even in regimes where in theory, it should be impossible.

In the simulations with more than two communities, there was no best algorithm. However, in most cases, the allocation sampler was the most consistent one, having good results in all simulations at some point. Unlike the other algorithms, the Spectral clustering, even with the exact number of communities, did not perform well. In most cases, it should be avoided since there are better algorithms available. When estimating the number of communities, we saw that the allocation sampler tends to underestimate the number of communities, while the exact ICL overestimates the value of K. The same problem happened with the Newman-Givan algorithm. The non-statistic based algorithm Newman-Girvan continues to be a good choice for community detection if you have a low number of communities and the graph is not very sparse.

Another aspect noted in the simulations is that not only sparser graphs cause problems for detecting the clusters, but the difference between the probabilities of having edges seems to have a significant impact on the precision of algorithms, something already studied when using two balanced communities.

5.1 Suggestions for future research

For future research, one could try to generalize the semidefinite programming algorithm for more than two communities and see if it will maintain consistent good results when estimating the right communities.

Still in the computational field, one of the paths to take is to look for more defined "regimes" of graphs where each algorithm is more consistent than others, as it was shown by the figures, there is not a clear better algorithm since in different situations all the algo-

rithms were the "best" at some point, understanding these situations is a way to make good advancements in the area of community detection. Looking at the Allocation Sampler and the Exact ICL algorithms to find out why both algorithms have consistently underestimated or super-estimated the number of communities in the simulations could result in a more consistent and better algorithm for community detection.

From a more theoretical perspective, the allocation sampler, exact ICL, and the Newman-Girvan algorithms do not have any proof of consistency. However, the simulations suggest that they could be consistent. Looking at these models more carefully could give a stronger base on when to use these algorithms and the confidence that those algorithms will achieve the exact recovery or at least almost exact recovery.

Appendix A

Code

```
1
2 ##### Packages
3
4 library(igraph) # Graphs and Clusters
5 library(CVXR) # Convex optmization
6 library(greed) # Come e Latouche (2015) ICL exact
7 library(Matrix) # Making a matrix for the ICL
8
9 ##### Code for installing the package for the AS algorithm
10
11 setwd(path) #Path were the graphs will be saved
12 system("git clone --recursive -o github git://github.com/aaronmcdaid/
    collapsedSBM.git")
13 system("cd collapsedSBM")
14 system("make")
15
16 ##### Support Functions
17
18 quiet <- function(x) { #Function to avoid warningsS
19     sink(tempfile())
20     on.exit(sink())
21     invisible(force(x))
22 }
23
24 makegraph.as <- function(x){ #x adjacency matrix
25     edges <- c()
26     for(i in 1:nrow(x)){
27         for(j in 1:ncol(x)){
28             if(x[i,j]==1){
29                 edges <- rbind(edges,c(i,j))
30             }
31         }
```

```
32      }
33      return(edges)
34  }
35
36  permut.labels <- function(k,xest,xreal){ #k number of communities, x the
        label vectors
37      perm <- function(v) { #Permutations
38          n <- length(v)
39          if (n == 1) v
40          else {
41              X <- NULL
42              for (i in 1:n) X <- rbind(X, cbind(v[i], perm(v[-i])))
43              X
44          }
45      }
46      comlabels <- 1:k
47      vetordeperms <- perm(1:k)
48      novolabel <- c()
49      precis <- 0
50      for(i in 1:nrow(vetordeperms)){
51          for(j in 1:k){
52              novolabel[xest == j] <- vetordeperms[i,j]
53          }
54          novolabel[is.na(novolabel)] <- k + 1
55          if((sum(novolabel==xreal)/length(xreal))>precis){
56              precis <- (sum(novolabel==xreal)/length(xreal))
57          }
58      }
59      return(precis)
60  }
61
62  spectral_clustering <- function(X, # matrix of data points
63                                  nn = 10, # the k nearest neighbors to
                                        consider
64                                  n_eig = 3) # m number of eignenvectors
                                        to keep
65  {
66
67      ei = eigen(X, symmetric = TRUE) # 3. Compute the eigenvectors and
            values of L
68      return(ei$vectors[,1:n_eig]) # return the eigenvectors of the n_eig
            largest values eigenvalues
69
70  }
71
```

```
72 sample.z <- function(n,q,k){ #Sample the labels given the probabilities
       pi (q)
73     k <- length(q)
74     return(sample(x = seq(1:k), n, replace = T, prob = q))
75 }
76
77 sample.g <- function(P,z,n,k){ #Sample the adjacency matrix given the
       labels and matrix P
78     mat.adj <- matrix(0,n,n)
79     k <- dim(P)[1]
80     for(i in 1:(n-1)){
81         for(j in (i+1):n){
82             mat.adj[i,j] <- rbinom(1,1,P[z[i],z[j]])
83             mat.adj[j,i] <- mat.adj[i,j]
84         }
85     }
86     #diag(mat.adj) <- 0
87     return(mat.adj)
88 }
89
90 convertstring <- function(setence){ #Converting a string into numeric
91     result <- strsplit(setence,",")
92     result <- as.numeric(result[[1]])
93     return(result)
94 }
95
96 ### Main Functions
97
98 sim_inc_size <-function(comsize = 2,n,p,q){
99     SDPmean <- c()
100    NGmean <- c()
101    Specmean <- c()
102    ExactICLmean <- c()
103    AllocSamplermean <- c()
104
105    num_communities <- c()
106    sample_size <- c()
107
108    k_NGmean <- c()
109    k_specmean <- c()
110    k_iclmean <- c()
111    k_AS <- c()
112
113    for(k in comsize){
114        for(i in seq(100,400,25)){
115            n <- i
```

```
116             p <- p*(log(n)/n)
117             q <- q*(log(n)/n)
118
119             kmat <- matrix(rep(q,k*k),ncol = k, nrow = k)
120             diag(kmat) <- p
121
122             tempNG <- c()
123             tempspecclust <- c()
124             tempExactICL <- c()
125             tempSDP <- c()
126
127             k_ng_temp <- c()
128             k_spec_temp <- c()
129             k_icl_temp <- c()
130
131                 for(j in 1:100){
132                     z <- sample.z(n,rep(1/k,k),k)
133                     g <- sample.g(kmat,z,n,k)
134                     gx <- igraph::graph_from_adjacency_matrix(g,mode="
                            undirected",diag=FALSE) # Grafo
135
136                     # Saving the graph to use on the AS algorithm
137                     locsave <- paste0("~/IncSample/collapsedSBM/n",n,"k"
                            ,k,"iter",j,".txt")
138                     write.table(makegraph.as(g), file = locsave, sep = "
                            \t", row.names = FALSE, col.names = FALSE)
139
140                     # AS calculations
141                     locread <- paste0("./sbm --iterations=5000 n",n,"k",
                            k,"iter",j,".txt")
142                     x <- system(locread, intern = TRUE)
143                     vetorresult <- stringi::stri_extract(x,regex = '
                            [0-9]+(,[0-9]+)+')
144                     vetorresult <- na.omit(vetorresult)[1]
145                     vetorresult <- convertstring(vetorresult)
146                     if(length(unique(vetorresult)) == 1){
147                         tempallocsampler[j] <- sum(((vetorresult + 1) ==
                                z))/n
148                         k_as_temp[j] <- 1
149                     }
150                     else{
151                         k_as_temp[j] <- ifelse(length(unique(vetorresult
                                )) < (k +3), length(unique(vetorresult)), (k
                                +1) )
152                         tempallocsampler[j] <- permut.labels(k_as_temp[j
                                ],(vetorresult + 1),z)
```

```
153                     }
154
155                     # SDP Calculations (For two communities only)
156                     new_g <- g
157                     new_g[new_g == 0] <- -1
158                     xestimated <- Variable(rows = n, cols =  n,PSD =
                            TRUE)
159                     objective <- (matrix_trace( new_g  \% * \%
                            xestimated))
160                     constraints <- list(diag(xestimated) == 1)
161                     problem <- Problem(Maximize(objective),constraints)
162                     result <- solve(problem)
163                     SDPcluster <- sign(eigen(result[[1]])$vectors[,1])
164                     tempSDP <- c(tempSDP,permut.labels(k,SDPcluster,z))
165
166                     # NG calculations
167                     ngtemp <- cluster_edge_betweenness(gx)
168                     kngtemp <- ifelse(length(unique(ngtemp$membership))
                            <= (k + 2),length(unique(ngtemp$membership)),k+1
                            )
169                     NGcluster <- permut.labels(kngtemp,ngtemp$membership
                            ,z)
170                     tempNG  <- c(tempNG,NGcluster)
171                     k_ng_temp <- c(k_ng_temp,kngtemp)
172
173                     # Spectral Clustering Calculations
174                     kmeantemp <- kmeans(spectral_clustering(g), centers
                            = k)
175                     X_temp_kmeans <- permut.labels(length(unique(
                            kmeantemp$cluster)),kmeantemp$cluster,z)
176                     k_spec_temp <- c(k_spec_temp,length(unique(kmeantemp
                            $cluster)))
177                     tempspecclust <- c(tempspecclust,X_temp_kmeans)
178
179                     # Exact ICL calculations
180                     tempicl <- NULL
181                     while(is.null(tempicl)){
182                          try(
183                               tempicl <- quiet(greed(g)),
184                               silent = TRUE
185                          )
186                     }
187                     k_icl_temp <- c(k_icl_temp,tempicl@K)
188                     kicltemp <- ifelse(tempicl@K <= (k+2),tempicl@K,k+1
                            )
189                     X_temp_icl <- permut.labels(kicltemp,tempicl@cl,z)
```

```
190                     tempExactICL <- c(tempExactICL,X_temp_icl)
191                 }
192
193             # Precision means
194             NGmean <- c(NGmean, mean(tempNG))
195             Specmean <- c(Specmean,mean(tempspecclust))
196             ExactICLmean <- c(ExactICLmean, mean(tempExactICL))
197             SDPmean <- c(SDPmean, mean(tempSDP))
198             AllocSamplermean <- c(AllocSamplermean, mean(
                    tempallocsampler))
199
200             # Checking the values used
201             num_communities <- c(num_communities, k)
202             sample_size <- c(sample_size, n)
203
204             # Means of estimated number of communities
205             k_iclmean <- c(k_iclmean, mean(k_icl_temp))
206             k_specmean <- c(k_specmean, mean(k_spec_temp))
207             k_NGmean <- c(k_NGmean, mean(k_ng_temp))
208             k_AS <- c(k_AS, mean(k_as_temp))
209         }
210     }
211     table <- data.frame(Ncom = num_communities,SampleSize = sample_size
            , NG = NGmean, Spec = Specmean, ExactICL = ExactICLmean,
212                                             SDP = SDPmean, AS =
                                                    AllocSamplermean, KNG = k_
                                                    NGmean, KSPEC = k_specmean,
                                                    KICL = k_iclmean, KAS = k_AS)
213     return(table)
214 }
215
216 plot_dif_probs <- function(p_vetor, q_vetor, comsize = 2){
217     SDPmean <- c()
218     NGmean <- c()
219     Specmean <- c()
220     ExactICLmean <- c()
221     AllocSamplermean <- c()
222
223     num_communities <- c()
224     p_used <- c()
225     q_used <- c()
226
227     k_NGmean <- c()
228     k_specmean <- c()
229     k_iclmean <- c()
230     k_AS <- c()
```

```
231
232     for(k in comsize){
233         for(i in 1:length(p_vetor)){
234             n <- 100
235             p <- p_vetor[i]
236             q <- q_vetor[i]
237
238             kmat <- matrix(rep(q,k*k),ncol = k, nrow = k)
239             diag(kmat) <- p
240
241             tempNG <- c()
242             tempspecclust <- c()
243             tempExactICL <- c()
244             tempSDP <- c() # Only for two communities
245
246             k_ng_temp <- c()
247             k_spec_temp <- c()
248             k_icl_temp <- c()
249
250                 for(j in 1:100){
251                     # Creating the graph
252                     z <- sample.z(n,rep(1/k,k),k)
253                     g <- sample.g(kmat,z,n,k)
254                     gx <- igraph::graph_from_adjacency_matrix(g,mode="
                            undirected",diag=FALSE) # Grafo
255
256                     # Saving the graph to use on the AS algorithm
257                     locsave <- paste0("~/DifProbs/collapsedSBM/p",p,"q",
                            q,"k",k,"iter",j,".txt")
258                     write.table(makegraph.as(g), file = locsave, sep = "
                            \t", row.names = FALSE, col.names = FALSE)
259
260                     # AS calculations
261                     locread <- paste0("./sbm --iterations=5000 p",p,"q",
                            q,"k",k,"iter",j,".txt")
262                     x <- system(locread, intern = TRUE)
263                     vetorresult <- stringi::stri_extract(x,regex = '
                            [0-9]+(,[0-9]+)+')
264                     vetorresult <- na.omit(vetorresult)[1]
265                     vetorresult <- convertstring(vetorresult)
266                     if(length(unique(vetorresult)) == 1){
267                         tempallocsampler[j] <- sum(((vetorresult + 1) ==
                                z))/n
268                         k_as_temp[j] <- 1
269                     }
270                     else{
```

```
271                     k_as_temp[j] <- ifelse(length(unique(vetorresult
                            )) < (k +3), length(unique(vetorresult)), (k
                            +1) )
272                     tempallocsampler[j] <- permut.labels(k_as_temp[j
                            ],(vetorresult + 1),z)
273                 }
274
275                 # SDP Calculations (For two communities only)
276                 new_g <- g
277                 new_g[new_g == 0] <- -1
278                 xestimated <- Variable(rows = n, cols =  n,PSD =
                        TRUE)
279                 objective <- (matrix_trace( new_g  \% * \%
                        xestimated))
280                 constraints <- list(diag(xestimated) == 1)
281                 problem <- Problem(Maximize(objective),constraints)
282                 result <- solve(problem)
283                 SDPcluster <- sign(eigen(result[[1]])$vectors[,1])
284                 tempSDP <- c(tempSDP,permut.labels(k,SDPcluster,z))
285
286                 # NG calculations
287                 ngtemp <- cluster_edge_betweenness(gx)
288                 kngtemp <- ifelse(length(unique(ngtemp$membership))
                        <= (k + 2),length(unique(ngtemp$membership)),k+1
                        )
289                 NGcluster <- permut.labels(kngtemp,ngtemp$membership
                        ,z)
290                 tempNG  <- c(tempNG,NGcluster)
291                 k_ng_temp <- c(k_ng_temp,kngtemp)
292
293                 # Spectral Clustering calculations
294                 kmeantemp <- kmeans(spectral_clustering(g), centers
                        = k)
295                 X_temp_kmeans <- permut.labels(length(unique(
                        kmeantemp$cluster)),kmeantemp$cluster,z)
296                 k_spec_temp <- c(k_spec_temp,length(unique(kmeantemp
                        $cluster)))
297                 tempspecclust <- c(tempspecclust,X_temp_kmeans)
298
299                 # Exact ICL calculations
300                 tempicl <- NULL
301                 while(is.null(tempicl)){
302                     try(
303                         tempicl <- quiet(greed(g)),
304                         silent = TRUE
305                     )
```

```
306                     }
307                     k_icl_temp <- c(k_icl_temp,tempicl@K)
308                     kicltemp <- ifelse(tempicl@K <= (k+2),tempicl@K,k+1
                            )
309                     X_temp_icl <- permut.labels(kicltemp,tempicl@cl,z)
310                     tempExactICL <- c(tempExactICL,X_temp_icl)
311                 }
312
313             # Getting the means for the precision
314             NGmean <- c(NGmean, mean(tempNG))
315             Specmean <- c(Specmean,mean(tempspecclust))
316             ExactICLmean <- c(ExactICLmean, mean(tempExactICL))
317             SDPmean <- c(SDPmean, mean(tempSDP))
318             AllocSamplermean <- c(AllocSamplermean, mean(
                    tempallocsampler))
319
320             # Just checking the values used in each step
321             num_communities <- c(num_communities, k)
322             p_used <- c(p_used,p)
323             q_used <- c(q_used,q)
324
325             # Getting the means for the number of communities
326             k_iclmean <- c(k_iclmean, mean(k_icl_temp))
327             k_specmean <- c(k_specmean, mean(k_spec_temp))
328             k_NGmean <- c(k_NGmean, mean(k_ng_temp))
329             k_AS <- c(k_AS, mean(k_as_temp))
330         }
331     }
332     table <- data.frame(Ncom = num_communities, p = p_used, q = q_used,
            NG = NGmean, Spec = Specmean, ExactICL = ExactICLmean,
333                                         SDP = SDPmean, AS =
                                                AllocSamplermean, KNG = k_
                                                NGmean, KSPEC = k_specmean,
                                                KICL = k_iclmean, KAS = k_AS)
334     return(table)
335 }
```

Bibliography

Abbe(2018) Emmanuel Abbe. Community detection and stochastic block models: Recent developments. *Journal of Machine Learning Research*, 18(177):1–86. URL http://jmlr.org/papers/v18/16-480.html. Quoted on page

Abbe(2017) Emmanuel Abbe. Community detection and stochastic block models: recent developments, 2017. Quoted on page 4

Abbe and Sandon(2015) Emmanuel Abbe and Colin Sandon. Community detection in general stochastic block models: fundamental limits and efficient recovery algorithms, 2015. Quoted on page 1, 6

Abbe *et al.***(2014a)** Emmanuel Abbe, Afonso S. Bandeira, Annina Bracher and Amit Singer. Linear inverse problems on erdõs-rényi graphs: Information-theoretic limits and efficient recovery. *2014 IEEE International Symposium on Information Theory*, páginas 1251–1255. Quoted on page

Abbe *et al.***(2014b)** Emmanuel Abbe, Afonso S. Bandeira, Annina Bracher and Amit Singer. Decoding binary node labels from censored edge measurements: Phase transition and efficient recovery, 2014b. Quoted on page

Abbe *et al.***(2014c)** Emmanuel Abbe, Afonso S. Bandeira and Georgina Hall. Exact recovery in the stochastic block model, 2014c. Quoted on page 1, 2, 5, 22

Bickel and Chen(2009) Peter Bickel and Aiyou Chen. A nonparametric view of network models and newman-girvan and other modularities. *Proceedings of the National Academy of Sciences of the United States of America*, 106:21068–73. doi: 10.1073/pnas.0907096106. Quoted on page 1

Caldarelli and Vespignani(2007a) Guido Caldarelli and Alessandro Vespignani. Large scale structure and dynamics of complex networks: From information technology to finance and natural science, 2007a. Quoted on page

Caldarelli and Vespignani(2007b) Guido Caldarelli and Alessandro Vespignani. *Large Scale Structure and Dynamics of Complex Networks: From Information Technology to Finance and Natural Science*. World Scientific Publishing Co., Inc., USA. ISBN 9789812706645. Quoted on page 8

Cerqueira(2018) Andressa Cerqueira. *Statistical inference on random graphs and networks*. Tese de Doutorado, Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil. Quoted on page

Cerqueira and Leonardi(2018) Andressa Cerqueira and Florencia Leonardi. Strong consistency of krichevsky-trofimov estimator for the number of communities in the stochastic block model, 2018. Quoted on page

Chen *et al.***(2018)** Shi Chen, Zhi-Zhong Wang, Liang Tang, Yan-Ni Tang, Yuan-Yuan Gao, Hui-Jia Li, Ju Xiang and Yan Zhangyan. Global vs local modularity for network community detection. *PLOS ONE*, 13:e0205284. doi: 10.1371/journal.pone.0205284. Quoted on page

Clauset *et al.***(2004)** Aaron Clauset, M. E. J. Newman and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6). ISSN 1550-2376. doi: 10.1103/physreve.70.066111. URL http://dx.doi.org/10.1103/PhysRevE.70.066111. Quoted on page

Côme and Latouche(2013) E. Côme and P. Latouche. Model selection and clustering in stochastic block models with the exact integrated complete data likelihood, 2013. Quoted on page 1, 16, 17, 25

Daudin *et al.***(2008)** J. Daudin, F. Picard and S. Robin. A mixture model for random graphs. *Statistics and Computing*, 18:173–183. Quoted on page 1, 16

Erdõs and Rényi(1960) P. Erdõs and A. Rényi. On the evolution of random graphs. Em *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, páginas 17–61. Quoted on page 1, 5

Euler(1736) Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140. Quoted on page 1

Fortunato and Castellano(2007) Santo Fortunato and Claudio Castellano. Community structure in graphs, 2007. Quoted on page 1

Freund(2004) Robert M. Freund. Introduction to semidefinite programming (sdp). Quoted on page

Hastie *et al.***(2001)** Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA. Quoted on page

Holland *et al.***(1983)** Paul W. Holland, Kathryn Blackmond Laskey and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109 – 137. ISSN 0378-8733. doi: https://doi.org/10.1016/0378-8733(83)90021-7. URL http://www.sciencedirect.com/science/article/pii/0378873383900217. Quoted on page 1

Hu *et al.***(2016)** Jianwei Hu, Hong Qin, Ting Yan and Yunpeng Zhao. Corrected bayesian information criterion for stochastic block models, 2016. Quoted on page

Jerrum and Sorkin(1998) Mark Jerrum and Gregory B. Sorkin. The metropolis algorithm for graph bisection. *Discrete Applied Mathematics*, 82(1):155 – 175. ISSN 0166-218X. doi: https://doi.org/10.1016/S0166-218X(97)00133-9. URL http://www.sciencedirect.com/science/article/pii/S0166218X97001339. Quoted on page 1

Krichevsky and Trofimov(1981) R. Krichevsky and V. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207. Quoted on page

Lee and Wilkinson(2019) Clement Lee and Darren Wilkinson. A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4. doi: 10.1007/s41109-019-0232-2. Quoted on page 1

Lei and Rinaldo(2015) Jing Lei and Alessandro Rinaldo. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1):215–237. ISSN 0090-5364. doi: 10.1214/14-aos1274. URL http://dx.doi.org/10.1214/14-AOS1274. Quoted on page 13

McDaid *et al.***(2012)** Aaron F. McDaid, Thomas Brendan Murphy, Nial Friel and Neil J Hurley. Clustering in networks with the collapsed stochastic block model, 2012. Quoted on page 1, 13, 16

Newman and Girvan(2004) M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2). ISSN 1550-2376. doi: 10.1103/physreve.69.026113. URL http://dx.doi.org/10.1103/PhysRevE.69.026113. Quoted on page 1

Newman(2004) Mark Newman. Detecting community structure in networks. *Eur Phys J*, 38. Quoted on page

Nobile and Fearnside(2007) Agostino Nobile and Alastair Fearnside. Bayesian finite mixtures with an unknown number of components: The allocation sampler. *Statistics and Computing*, 17:147–162. doi: 10.1007/s11222-006-9014-7. Quoted on page 13, 14, 15, 16

Pas and Vaart(2016) Stéphanie Pas and Aad Vaart. Bayesian community detection. *Bayesian Analysis*, 13. doi: 10.1214/17-BA1078. Quoted on page 1

Radicchi *et al.***(2004)** Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663. ISSN 0027-8424. doi: 10.1073/pnas.0400054101. URL https://www.pnas.org/content/101/9/2658. Quoted on page 8

Rohe *et al.***(2011)** Karl Rohe, Sourav Chatterjee and Bin Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics*, 39(4):1878–1915. ISSN 0090-5364. doi: 10.1214/11-aos887. URL http://dx.doi.org/10.1214/11-AOS887. Quoted on page 1

Snijders and Nowicki(1997) Tom Snijders and Krzysztof Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14:75–100. doi: 10.1007/s003579900004. Quoted on page 13

von Luxburg(2007) Ulrike von Luxburg. A tutorial on spectral clustering, 2007. Quoted on page