R.R. Geerling

Community detection in networks

Bachelorscriptie

Scriptiebegeleiders: **dr. A.J. Schmidt-Hieber**
S.L. van der Pas, MSc MA

Datum Bachelorexamen: 5 juli 2015

Mathematisch Instituut, Universiteit Leiden

Contents

1 Introduction

Recently there has been much interest in networks in various disciplines, such as computer science, sociology, economics and physics. Networks are used to indicate links between objects, such as friendships between people or partnerships between countries. Many networks have a community structure, which means that the vertices are divided into a number of classes, such that the vertices within the same class share the same connection probabilities to each other as well as to vertices in other classes. Community structure gives a lot of information about a network. For instance, the communities in a social network may denote groups of friends or acquaintances, in the World Wide Web the communities may denote sets of websites on the same topic, et cetera. Therefore community detection is of great importance in the study of properties of networks.

In this thesis we study a number of community detection methods. The networks we consider are generated by the Stochastic block model, which is described in Section 2. In Section 3 we study several methods to identify the communities when the number of communities is known. In Section 4 we examine a number of methods to estimate the number of communities, when this number is not known. A simulation study on small networks to compare the methods in different cases is given in Section 5. Finally, in Section 6 a conclusion is given.

2 Model

All networks we consider in this thesis are generated according to the Stochastic block model. This model divides the vertices of a network into a number of blocks and subsequently vertices are connected by edges with probabilities depending on the blocks the vertices are in. This way a community structure is created, with the blocks representing the communities, which we also call classes. We mainly focus on networks generated according to the Planted Clustering model with two classes, which is a special case of the Stochastic model.

2.1 Stochastic block model

For any $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. A network with n vertices is defined by the pair $([n], A)$, where A is an $n \times n$-matrix with, for $i, j \in [n]$,

$$A_{ij} = \begin{cases} 1 & \text{if the edge } (i,j) \text{ is contained in the network} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix A is called the adjacency matrix of the network.

In the Stochastic block model the set of vertices is divided into $K \geq 1$ classes. This means that each vertex $i \in [n]$ gets a label $Z_i \in [K]$ that indicates what class vertex i is in. Let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_K)$, with $\sum_{i=1}^{K} \alpha_i = 1$, be the vector of block proportions, such that

$$Z = (Z_i)_{i \in [n]} \overset{\text{i.i.d.}}{\sim} \mathcal{M}(1; \vec{\alpha}).$$

Here $\mathcal{M}$ denotes the multinomial distribution, i.e. $Z_i = k$ with probability α_k for all $i \in [n]$, $k \in [K]$.

Let $P = (P_{kl})_{k,l \in [K]}$ be a $K \times K$-matrix of probabilities, such that

$$\mathbb{P}(A_{ij} = 1 \mid Z_i = k, Z_j = l) = P_{kl}.$$

This way the probability of an edge between two vertices only depends on the classes the vertices are in. We assume that the probabilities on the diagonal of P are the largest, such that the classes form communities with relatively many connections within them.

2.2 Planted Clustering model

We study a special case of the stochastic block model, namely the Planted Clustering model with one cluster (Chen and Xu, 2015). In this model we have $K = 2$ communities, with the vertices of one class forming a cluster in the network and with the other class consisting of the remaining vertices.

Let $[n]$ be a set of vertices and let $\alpha \in (0, 1)$. We generate a subset S of $[n]$ by putting each vertex in S with probability α. This divides the n vertices into two classes, 1 and 2, which are associated with S and its complement S^c respectively. Two vertices $i, j \in [n]$ are connected by an edge with probability

$$\begin{cases} p & \text{if } i, j \in S \\ q & \text{otherwise,} \end{cases}$$

for certain $0 < q < p < 1$. The probability matrix P is now given by

$$P = \begin{pmatrix} p & q \\ q & q \end{pmatrix}.$$

In section 3.1.5 we will consider the case when p and q are functions of n. Until then we assume they are constants. As $p > q$, the fraction of edges within S is expected to be higher than the fraction of edges in the rest of the network. Therefore the vertices in S are expected to form a cluster in the network.

In general the Planted Clustering model can be defined for any number of classes K, such that there are $K-1$ clusters. In this thesis, unless noted otherwise, 'Planted Clustering model' refers to the model with two classes.

3 Community detection for known number of communities

In this section we study a number of methods to identify the communities when the number of classes is known. In Section 3.1 we examine the Largest Gaps algorithm. This method only makes use of the degrees of the vertices and is therefore computable for very large networks. We will see that, under strong assumptions, the Largest Gaps algorithm is consistent, which means that it finds the true communities

almost surely when n tends to infinity. In Section 3.2 we study the Newman-Girvan algorithm. This method is harder to compute, but is generally considered more reliable than the LG algorithm. Finally, in Section 3.3, we study two modularities and a way to use these quantities to detect communities. This method is however only computable for very small networks.

3.1 Largest Gaps algorithm

We assume that we know the network is generated by the Planted Clustering model and therefore we know there are two classes. The first method we consider to find these classes is the Largest Gaps (LG) algorithm. This method makes use of the degrees of the vertices and is based on the fact that the vertices of S are expected to have a larger degree than the other vertices.

The LG algorithm was introduced by Channarond, Daudin and Robin (2012). All of the following theory in this section about the LG algorithm, with the exception of Section 3.1.6, is based on this paper.

3.1.1 Degree distribution

Definition 3.1. For all $i \in [n]$, the *degree* of vertex i is

$$D_i^n := \sum_{j \neq i} A_{ij}.$$

Proposition 3.2. *D_i^n is a binomially distributed random variable conditionally on $Z_i = k$ with parameters $(n-1, \overline{P}_k)$, where $\overline{P}_k = \alpha P_{k1} + (1-\alpha)P_{k2}$.*

$\overline{P}_k$ can be interpreted as the average probability that a vertex in class k is connected to a uniformly at random picked vertex in the network. We have

$$\begin{aligned} \overline{P}_1 &= \alpha p + (1-\alpha)q, \\ \overline{P}_2 &= \alpha q + (1-\alpha)q = q. \end{aligned}$$

Because of $p > q$ and $\alpha > 0$ it follows that $\overline{P}_1 > \overline{P}_2$ holds.

Definition 3.3. The *normalized degree* of vertex $i \in [n]$ is

$$T_i^n := \frac{D_i^n}{n-1}.$$

We expect that, for n large enough, T_i^n is close to $\overline{P}_k$ given $Z_i = k$. To show that this is indeed true we can use Hoeffding's inequality (Hoeffding, 1963).

Theorem 3.4 (Hoeffding's inequality)**.** *Let $n \geq 1$, $r \in (0,1)$ and $(X_i)_{i\in[n]} \overset{\text{i.i.d.}}{\sim} \text{Ber}(r)$. Then for any $t > 0$*

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - r\right| > t\right) \leq 2e^{-2nt^2}.$$

Conditionally on $Z_i = k$, D_i^n is distributed $\mathcal{B}(n-1, \overline{P}_k)$, where $\mathcal{B}$ denotes the binomial distribution. Therefore it follows from Theorem 3.4 that

$$\mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k) \leq 2e^{-2(n-1)t^2}. \tag{1}$$

Hence the normalized degrees of k-labeled vertices are concentrated around $\overline{P}_k$. This means that, in the degree distribution, the vertices are with high probability divided into two groups; the normalized degrees of vertices in S are centered around $\overline{P}_1$ and those of the other vertices around $\overline{P}_2$. Therefore, the following quantity will play an important role in the theory about the LG algorithm.

Definition 3.5. The *smallest discrepancy* is

$$\delta := \min_{k \neq l} |\overline{P}_k - \overline{P}_l.|$$

This is the general definition for any number of classes K. In the Planted Clustering model we have $\delta = \overline{P}_1 - \overline{P}_2 > 0$. Because of the concentration inequality (1) the gap between the normalized degrees of vertices from different classes is expected to be larger than that of vertices within the same class. Hence it could be possible to identify the classes by comparing the normalized degrees of all vertices. This is done in the Largest Gaps algorithm.

3.1.2 Largest Gaps algorithm

For a sequence $(u_i)_{i\in[n]}$ of real numbers, $(u_{(i)})_{i\in[n]}$ denotes the same sequence, but sorted in increasing order.

Algorithm 3.6.

1. *Sort the normalized degrees in increasing order:*

$$T_{(1)}^n \leq T_{(2)}^n \leq \ldots \leq T_{(n)}^n.$$

2. *Compute the gap between each pair of consecutive normalized degrees:*

$$T_{(i+1)}^n - T_{(i)}^n, \qquad i \in [n-1].$$

3. *Determine the index of the largest gap, say j, such that for all $i \neq j$:*

$$T_{(j+1)}^n - T_{(j)}^n \geq T_{(i+1)}^n - T_{(i)}^n.$$

4. *Take $\{\{(1), \ldots, (j)\}, \{(j+1), \ldots, (n)\}\}$ as estimate for the true partition.*

As the vertices in S have a larger expected degree than the other vertices, we expect the class $\{(j+1), \ldots, (n)\}$ to be S if our estimate is the true partition.

3.1.3 Main result

Let $\{C_k^n\}_{k\in\{1,2\}}$ denote the true partition of $[n]$ into classes (thus $C_1^n = S$ and $C_2^n = S^c$) and let $\{\widehat{C}_k^n\}_{k\in\{1,2\}}$ denote the estimated partition. Write N_k^n and $\widehat{N}_k^n$ for the cardinality of the true and estimated k-labeled class respectively. Let E_n be the event 'the Largest Gaps algorithm makes at least one mistake', i.e.

$$E_n = \left\{ \{\widehat{C}_k^n\}_{k\in\{1,2\}} \neq \{C_k^n\}_{k\in\{1,2\}} \right\}.$$

Definition 3.7. $\{\widehat{C}_k^n\}_{k\in\{1,2\}}$ is called *consistent* if

$$\mathbb{P}(E_n) \underset{n\to\infty}{\longrightarrow} 0.$$

The following theorem gives an upper bound for $\mathbb{P}(E_n)$, from which the consistency of the LG algorithm follows when $\delta > 0$ is fixed.

Theorem 3.8.

$$\mathbb{P}(E_n) \leq 2ne^{-\frac{1}{8}(n-1)\delta^2} + (1-\alpha)^n + \alpha^n.$$

The proof of Theorem 3.8 is delayed until section 3.1.4. When δ and α are constants and $\delta > 0$, the upper bound in Theorem 3.8 clearly tends to 0. Therefore the LG algorithm is consistent in that case. In section 3.1.5 we consider the case when δ is a function of n that converges to 0, which is more realistic from an applied point of view. In this case the upper bound does not necessarily go to 0 and we will need to make some assumptions for the LG algorithm to be consistent.

3.1.4 Proof of the consistency of the LG algorithm

When none of the classes is empty and the spreading of the normalized degrees is small compared to the smallest discrepancy δ, we can be sure that the LG algorithm returns the true partition. Let A_n be the event 'no true class is empty', i.e.

$$A_n = \{C_1^n \neq \emptyset\} \cap \{C_2^n \neq \emptyset\} = \{N_1^n \neq 0\} \cap \{N_2^n \neq 0\}.$$

Define

$$d_n = \max_{k\in\{1,2\}} \sup_{i\in C_k^n} |T_i^n - \overline{P}_k|.$$

This is the maximum, over all vertices and classes, of the distance between the normalized degree of a vertex and its expected value.

Proposition 3.9. *For any $\epsilon > 0$,*

$$A_n \cap \left\{ d_n \leq \frac{\delta}{4+\epsilon} \right\} \subset E_n^c.$$

Proof. Assume $A_n \cap \left\{ d_n \leq \frac{\delta}{4+\epsilon} \right\}$ is true. For vertices $i, j \in [n]$ with the same label

$k \in \{1, 2\}$, we have, using the triangle inequality,

$$\begin{aligned} |T_j^n - T_i^n| &\leq |T_j^n - \overline{P}_k| + |T_i^n - \overline{P}_k| \\ &\leq 2d_n \\ &\leq \frac{2\delta}{4+\epsilon}. \end{aligned}$$

When i and j have different labels, say 1 and 2 respectively, we find, using the triangle inequality again,

$$\begin{aligned} |T_j^n - T_i^n| &\geq |T_j^n - \overline{P}_1| - |T_i^n - \overline{P}_1| \\ &\geq |T_j^n - \overline{P}_1| - \frac{\delta}{4+\epsilon} \\ &\geq |\overline{P}_1 - \overline{P}_2| - |T_j^n - \overline{P}_2| - \frac{\delta}{4+\epsilon} \\ &\geq \delta - \frac{\delta}{4+\epsilon} - \frac{\delta}{4+\epsilon} \\ &= \frac{(2+\epsilon)\delta}{4+\epsilon} \\ &> \frac{2\delta}{4+\epsilon}. \end{aligned}$$

We see that i and j are in the same class if and only if $|T_i^n - T_j^n| \leq \frac{2\delta}{4+\epsilon}$. Note that the sequence $\{|T_{(i+1)}^n - T_{(i)}^n|\}_{i\in[n-1]}$ contains exactly one interval length strictly greater than $\frac{2\delta}{4+\epsilon}$. Hence the Largest Gaps algorithm returns the true partition in this case. □

Proposition 3.10. *For any $t > 0$*

$$\mathbb{P}(d_n > t) \leq 2ne^{-2(n-1)t^2}.$$

Proof. Recall inequality (1):

$$\mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k) \leq 2e^{-2(n-1)t^2}.$$

Hence, using conditional expectation and the union bound, we obtain

$$\begin{aligned} \mathbb{P}(d_n > t) &= \mathbb{E}(\mathbb{P}(d_n > t \mid Z)) \\ &= \mathbb{E}\left(\mathbb{P}(\cup_{k\in\{1,2\}} \cup_{i\in C_k^n} \{|T_i^n - \overline{P}_k| > t\} \mid Z)\right) \\ &\leq \mathbb{E}\left(\sum_{k\in\{1,2\}} \sum_{i\in C_k^n} \mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z)\right) \\ &\leq \mathbb{E}\left(\sum_{k\in\{1,2\}} \sum_{i\in C_k^n} \mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k)\right) \\ &\leq 2ne^{-2(n-1)t^2}. \end{aligned}$$

□

We are now able to prove Theorem 3.8.

Proof of Theorem 3.8. According to Proposition 3.9 we have $A_n \cap \left\{ d_n \leq \frac{\delta}{4+\epsilon} \right\} \subset E_n^c$, which implies

$$\mathbb{P}(E_n) \leq \mathbb{P}\left(\left(A_n \cap \left\{ d_n \leq \frac{\delta}{4+\epsilon} \right\} \right)^c \right) \leq \mathbb{P}(A_n^c) + \mathbb{P}\left(d_n > \frac{\delta}{4+\epsilon} \right).$$

To find a bound for the first term, we note that A_n^c denotes the event 'there exists an empty class'. Recall that N_k^n denotes the cardinality of class k. As $N_1^n \sim \mathcal{B}(n, \alpha)$ and $N_2^n \sim \mathcal{B}(n, 1-\alpha)$, we obtain

$$\begin{aligned} \mathbb{P}(A_n^c) &= \mathbb{P}(\{N_1^n = 0\} \cup \{N_2^n = 0\}) \\ &\leq \mathbb{P}(N_1^n = 0) + \mathbb{P}(N_2^n = 0) \\ &= (1-\alpha)^n + \alpha^n. \end{aligned}$$

Using Proposition 3.10 we find a bound for the second term.

$$\mathbb{P}\left(d_n > \frac{\delta}{4+\epsilon} \right) \leq 2n \exp\left(-2(n-1) \left(\frac{\delta}{4+\epsilon} \right)^2 \right).$$

If we combine these two bounds, with ϵ tending to 0 in the second one, we find a bound for $\mathbb{P}(E_n)$:

$$\mathbb{P}(E_n) \leq 2ne^{-\frac{1}{8}(n-1)\delta^2} + (1-\alpha)^n + \alpha^n.$$

□

3.1.5 Consistency when P depends on n

In this section we make some assumptions that lead to a more natural setting for community detection. In this setting the LG algorithm is not necessarily consistent and therefore we study conditions under which the consistency still holds.

Assume that the probability matrix P depends on n. Because δ now also depends on n, we write δ_n instead. We assume

$$\delta_n \xrightarrow[n\to\infty]{} 0.$$

The upper bound for $\mathbb{P}(E_n)$ that we found in Theorem 3.8 now does not necessarily converge to 0. The following theorem gives a condition under which the LG algorithm is still consistent.

Theorem 3.11. *The Largest Gaps algorithm, applied to a Planted clustering modeled network, is consistent under the assumption*

$$\liminf_{n\to\infty} \delta_n \sqrt{\frac{n-1}{\log n}} > 2\sqrt{2}.$$

Proof. From the assumption it follows that there exists $C > 0$ such that, for n large enough, we have

$$\frac{(n-1)\delta_n^2}{\log n} - 8 \geq C.$$

This gives

$$\begin{aligned} 2n \exp\left(-\tfrac{1}{8}(n-1)\delta_n^2\right) &= 2\exp\left(-\tfrac{1}{8}\log n\left(\frac{(n-1)\delta_n^2}{\log n} - 8\right)\right) \\ &\leq 2\exp\left(-\tfrac{1}{8}C\log n\right) \xrightarrow[n\to\infty]{} 0. \end{aligned}$$

Also as α is a constant lying in the interval $(0,1)$, we have

$$(1-\alpha)^n \xrightarrow[n\to\infty]{} 0 \quad \text{and} \quad \alpha^n \xrightarrow[n\to\infty]{} 0.$$

Hence, the bound in Theorem 3.8 converges to 0. □

3.1.6 Bernstein's inequality

The result obtained in Theorem 3.11 can be sharpened if we make the assumption $0 < q < p \leq \frac{1}{2}$. This is a very weak assumption, as almost all real networks have very small connection probabilities. To get to the new result we use one of Bernstein's inequalities (see e.g. Van der Vaart and Wellner, 1996).

Theorem 3.12 (Bernstein's inequality)**.** *For $n \geq 1$, let $(X_i)_{i\in[n]}$ be a sequence of i.i.d. random variables, with $\mathbb{E}X_i = 0$ and $|X_i| \leq 1$ for all $i \in [n]$. Then for any $t > 0$*

$$\mathbb{P}\left(\left|\sum_{i=1}^n X_i\right| > t\right) \leq 2\exp\left(-\frac{\frac{1}{2}t^2}{n\mathbb{E}[X_1^2] + \frac{t}{3}}\right).$$

Corollary 3.13. *Let $r \in [0,1]$ and let $(Y_i)_{i\in[n]}$ be a sequence of i.i.d random variables, with $\mathbb{E}Y_i = r$ and $Y_i \in [0,1]$ for all $i \in [n]$. Then for any $t > 0$*

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^n Y_i - r\right| > t\right) \leq 2\exp\left(-\frac{\frac{1}{2}nt^2}{\mathrm{Var}[Y_1] + \frac{t}{3}}\right).$$

Proof. Define $(X_i)_{i\in[n]}$ by $X_i := Y_i - r$. Now we have, for all $i \in [n]$, $\mathbb{E}X_i = 0$ and $X_i \in [-r, 1-r]$, and thus $|X_i| \leq 1$. With Theorem 3.12 we now find

$$\begin{aligned} \mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^n Y_i - r\right| > t\right) &= \mathbb{P}\left(\left|\sum_{i=1}^n X_i\right| > nt\right) \\ &\leq 2\exp\left(-\frac{\frac{1}{2}nt^2}{\mathbb{E}[X_1^2] + \frac{t}{3}}\right). \end{aligned}$$

We note that

$$\mathbb{E}[X_1^2] = \mathbb{E}[(Y_1 - r)^2] = \mathrm{Var}[Y_1],$$

which proves the claim. □

Since $D_i^n \mid Z_i = k \sim \mathcal{B}(n-1, \overline{P}_k)$, we have

$$\mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k) \leq 2\exp\left(-\frac{\frac{1}{2}(n-1)t^2}{\overline{P}_k(1-\overline{P}_k) + \frac{t}{3}}\right). \tag{2}$$

Proposition 3.14. *Assuming* $0 < q < p \leq \frac{1}{2}$, *for any* $t > 0$

$$\mathbb{P}(d_n > t) \leq 2n\exp\left(-\frac{\frac{1}{2}(n-1)t^2}{\overline{P}_1 + \frac{t}{3}}\right).$$

Proof. From $0 < q < p \leq \frac{1}{2}$ follows $0 < \overline{P}_2 < \overline{P}_1 < \frac{1}{2}$. Therefore, using inequality (2), we find for $k \in \{1, 2\}$

$$\mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k) \leq 2\exp\left(-\frac{\frac{1}{2}(n-1)t^2}{\overline{P}_1(1-\overline{P}_1) + \frac{t}{3}}\right) \leq 2\exp\left(-\frac{\frac{1}{2}(n-1)t^2}{\overline{P}_1 + \frac{t}{3}}\right).$$

Hence, using conditional expectation and the union bound, we obtain

$$\begin{aligned}
\mathbb{P}(d_n > t) &= \mathbb{E}(\mathbb{P}(d_n > t \mid Z)) \\
&= \mathbb{E}\left(\mathbb{P}(\cup_{k\in\{1,2\}} \cup_{i\in C_k} \{|T_i^n - \overline{P}_k| > t\} \mid Z)\right) \\
&\leq \mathbb{E}\left(\sum_{k\in\{1,2\}} \sum_{i\in C_k} \mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z)\right) \\
&\leq \mathbb{E}\left(\sum_{k\in\{1,2\}} \sum_{i\in C_k} \mathbb{P}(|T_i^n - \overline{P}_k| > t \mid Z_i = k)\right) \\
&\leq 2n\exp\left(-\frac{\frac{1}{2}(n-1)t^2}{\overline{P}_1 + \frac{t}{3}}\right).
\end{aligned}$$

□

Using this result, we can find a new upper bound for $\mathbb{P}(E_n)$.

Theorem 3.15. *Under the assumption* $0 < q < p \leq \frac{1}{2}$,

$$\mathbb{P}(E_n) \leq 2n\exp\left(-\frac{(n-1)\delta^2}{35\overline{P}_1}\right) + (1-\alpha)^n + \alpha^n.$$

Proof. As shown in the proof of Theorem 3.8, we have

$$\begin{aligned}
\mathbb{P}(E_n) &\leq \mathbb{P}\left(d_n > \frac{\delta}{4+\epsilon}\right) + \mathbb{P}(A_n^c) \\
&\leq \mathbb{P}\left(d_n > \frac{\delta}{4+\epsilon}\right) + (1-\alpha)^n + \alpha^n.
\end{aligned}$$

Using Proposition 3.14 and the fact that $\delta_n \leq \overline{P}_1$ always holds, we find

$$\begin{aligned}\mathbb{P}\left(d_n > \frac{\delta}{4}\right) &\leq 2n\exp\left(-\frac{\frac{1}{32}(n-1)\delta^2}{\overline{P}_1 + \frac{\delta}{12}}\right) \\ &\leq 2n\exp\left(-\frac{(n-1)\delta^2}{35\overline{P}_1}\right).\end{aligned}$$

Plugging this into the above inequality, where we let ϵ tend to 0, we find the claimed upper bound for $\mathbb{P}(E_n)$. □

As in the previous section, we consider the case where P, and therefore δ, depends on n. Therefore we write p_n, q_n, $\overline{P_k^n}$ and δ_n for p, q, $\overline{P}_k$ and δ respectively. We assume $0 < q_n < p_n \leq \frac{1}{2}$ for all n and

$$\delta_n \xrightarrow[n\to\infty]{} 0.$$

Theorem 3.16. *The Largest Gaps algorithm, applied to a Planted clustering modeled network, is consistent under the assumption*

$$\liminf_{n\to\infty} \delta_n \sqrt{\frac{n-1}{\overline{P_1^n}\log n}} > \sqrt{35}.$$

Proof. Since we have $0 < q_n < p_n \leq \frac{1}{2}$ for all n, we can use the upper bound of Theorem 3.15 and show that it converges to 0.

From the assumption it follows that there exists $C > 0$ such that, for n large enough, we have

$$\frac{(n-1)\delta_n^2}{\overline{P_1^n}\log n} - 35 \geq C.$$

This gives

$$\begin{aligned}2n\exp\left(-\frac{(n-1)\delta_n^2}{35\overline{P_1^n}}\right) &= 2\exp\left(-\frac{1}{35}\log n\left(\frac{(n-1)\delta_n^2}{\overline{P_1^n}\log n} - 35\right)\right) \\ &\leq 2\exp\left(-\frac{C}{35}\log n\right) \xrightarrow[n\to\infty]{} 0.\end{aligned}$$

Also, as α is a constant lying in the interval $(0,1)$, we have

$$(1-\alpha)^n \xrightarrow[n\to\infty]{} 0 \quad \text{and} \quad \alpha^n \xrightarrow[n\to\infty]{} 0.$$

Hence, the bound in Theorem 3.15 converges to 0. □

In real networks it is often the case that $0 < q_n < p_n \leq \frac{1}{2}$ for all n and that p_n and q_n converge to 0 as $n \to \infty$. Therefore it follows from Theorem 3.16 that $\frac{1}{\delta_n}$ only needs to be of order $O(\sqrt{\frac{n-1}{\overline{P_1^n}\log n}})$, instead of $O(\sqrt{\frac{n-1}{\log n}})$ in such networks. Since $\overline{P_1^n} \leq p_n$ always holds, we have $\overline{P_1^n} \xrightarrow[n\to\infty]{} 0$. Therefore this makes the restrictions on δ_n a lot weaker.

3.2 Newman-Girvan algorithm

In this section we explore another method to identify the communities of a network; the Newman-Girvan (NG) algorithm. This method was introduced by Newman and Girvan (2004). The method can be used when the number of classes is unknown, which we will see in Section 4.3. In that case the algorithm itself is used to find a partition in K classes for each $K \in \{1, \ldots, n\}$. To decide which of these partitions to pick as the estimate for the true partition the so called Newman-Girvan modularity is used.

In this section we study the algorithm when the true number of classes K is known, in the general case where K can be any number in $\{1, \ldots, n\}$. Because K is known we don't need to use the Newman-Girvan modularity. In Section 3.3.1 we explore a way to use the modularity for community detection, without using the algorithm first.

The Newman-Girvan algorithm makes use of the betweenness measures of the edges in the network. The betweenness of an edge can roughly be described as the number of shortest paths between all pairs of vertices that pass through this particular edge. In a network with community structure we expect the edge betweenness to be largest for inter-community edges (i.e., edges that connect two vertices in different communities), as a lot of shortest paths from one community to the other pass through these edges. Therefore the edge betweenness scores could help us identify the classes of the network.

Algorithm 3.17 (Newman-Girvan algorithm)**.**

1. *Calculate the edge betweenness for all edges in the network.*
2. *Remove the edge with the highest betweenness. If this edge is not unique, choose one of the edges with the highest betweenness at random and remove it.*
3. *Recalculate the betweenness for all edges in the new network.*
4. *Repeat from step 2 until no edges are left.*

The idea of the algorithm is the following. After a number of edges is removed the network will split into two components, representing a partition into two classes. As we proceed with removing edges the network will split into more components, such that partitions into more classes are found. Since we assume that K is known, we could stop removing edges when there are K components, to make the algorithm shorter. The found components now give a unique partition into K communities.

3.2.1 Calculation of the betweenness

As said before, the betweenness of an edge is roughly the number of shortest paths in the network that pass through this edge. For instance, if the shortest path between vertices i and j passes through edge e, this adds 1 to the betweenness score of e. However, in general there can be several shortest paths between a pair of vertices. Therefore, if there are, say, three shortest paths between i and j, of which two pass

through e, this adds $\frac{2}{3}$ to the betweenness score of e.

Newman and Girvan described a way to compute all betweenness scores in a network with n vertices and m edges in time $O(mn)$, which was introduced by Newman (2001) and independently by Brandes (2001). In this method the algorithm Breadth-first search is used to find all shortest paths from a single vertex s (the source vertex) to all other vertices. In this algorithm all vertices j are assigned a distance d_j to s and a set F of shortest paths is created. A queue Q is used to keep track of the vertices that have been assigned a distance, but whose attached edges have not yet been followed. In the algorithm below, $front(Q)$ denotes the first element of Q, $dequeue(Q)$ means the first element of Q is removed from Q and $enqueue(j, Q)$ means the vertex j is added to the back of Q. The algorithm is as follows.

Algorithm 3.18 (Breadth-first search)**.**

1. $F := \emptyset$; $Q := \{s\}$; $d_s = 0$.
2. *While* $Q \neq \emptyset$, *do:*

   $i := front(Q)$; $dequeue(Q)$.

   For each vertex j *adjacent to* i*;*

   (a) *if* j *has not yet been assigned a distance, do:*
   $d_j := d_i + 1$; $F := F \cup \{(i, j)\}$; $enqueue(j, Q)$.

   (b) *if* $d_j = d_i + 1$*: do* $F := F \cup \{(i, j)\}$.

   (c) *else: do nothing.*

The output of the algorithm is F, which is the set of shortest paths from s to every other vertex. Using F betweenness scores $B_s(e)$ associated with s can be calculated for all edges e. First a weight w_i and a new distance d_i are assigned to every vertex i in F using the following algorithm.

Algorithm 3.19.

1. $d_s := 0$; $w_s := 1$.
2. *For every vertex* i *adjacent to* s*, do* $d_i := 1$ *and* $w_i := 1$.
3. *For each vertex* j *adjacent to one of those vertices* i*;*

   (a) *if* j *has not yet been assigned a distance, do* $d_j := d_i + 1$ *and* $w_j := w_i$.

   (b) *if* $d_j = d_i + 1$*, do* $w_j := w_j + w_i$.

   (c) *if* $d_j < d_i + 1$*, do nothing.*
4. *Repeat from step 3 until all vertices have been assigned a distance.*

The weight w_i of a vertex i is now equal to the number of shortest paths between s and i. Using these weights we can calculate the betweenness scores as follows.

Algorithm 3.20.

1. *Find every vertex t that has maximal distance to s.*
2. *For each vertex i neighboring t, do $B_s((i,t)) := \frac{w_i}{w_t}$.*
3. *Working up towards s, for each edge (i,j), with j being farther away from s than i, let $b_{(i,j)}$ be the sum of the betweenness scores of the edges directly below (i,j) and do $B_s((i,j)) := \frac{w_i}{w_j}(1 + b_{(i,j)})$.*
4. *Repeat from step 3 until s is reached.*

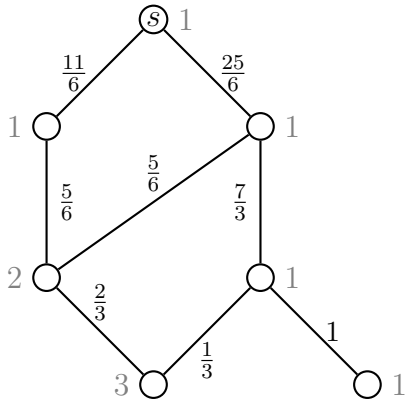

Figure 1: Calculation of the betweenness scores in the set of shortest paths from a source vertex s. The numbers on the vertices indicate the weights. The numbers on the edges are the betweenness scores $B_s(e)$.

Figure 1 shows an example of a set of shortest paths from some source vertex s, where the betweenness scores have been calculated. After all betweenness scores have been calculated for all n vertices as source vertex, the total betweenness $B(e)$ for each edge e can be calculated:

$$B(e) = \tfrac{1}{2} \sum_{s \in [n]} B_s(e).$$

Here we take half the sum because otherwise every shortest path would contribute twice.

3.2.2 Computation time

The computation time of Breadth-first search is $O(m)$. Therefore the calculation of $B_s(e)$ for all edges e also takes total time $O(m)$. These betweenness scores have to be calculated for every possible source vertex s to find the total betweenness scores. Therefore this takes time $O(mn)$. Since the total betweenness scores are recalculated in each iteration of the NG algorithm (Algorithm 3.17), the worst case time is $O(m^2 n)$.

3.3 Modularity maximization

In this section a community detection method is described that uses modularity maximization. A modularity is a measure of the strength of the community structure in a network. We study two modularities; the Newman-Girvan modularity (Section 3.3.1) and the Likelihood modularity (Section 3.3.2). Both modularities are described for the general Stochastic Block model with K communities.

3.3.1 Newman-Girvan modularity

We will now examine a method introduced in (Bickel and Chen, 2009) that makes use of the Newman-Girvan modularity, which was initially used by Newman and Girvan (2004) in combination with the NG algorithm. Let $C_n = \{C_k^n\}_{k\in[K]}$ be a partition into K communities of a network with adjacency matrix A. For $k, l \in [K]$, define

$$\mathrm{O}_{kl}(C_n, A) := \sum_{i,j\in[n]} A_{ij} \mathbb{1}_{\{i\in C_k^n, j\in C_l^n\}},$$

such that, for $k \neq l$, $\mathrm{O}_{kl}(C_n, A)$ is the number of edges between vertices in communities k and l and $\mathrm{O}_{kk}(C_n, A)$ is twice the number of edges between vertices in community k. Let $\Delta_k(C_n, A) := \sum_{l=1}^K \mathrm{O}_{kl}(C_n, A)$ be the sum of the degrees of all vertices in community k. Define $\Lambda(A) := \sum_{k=1}^K \Delta_k(C_n, A)$ as the sum of all degrees, which is equal to twice the number of edges in the network. For convenience we will just write O_{kl}, Δ_k and Λ without the parameters C_n and A. The Newman-Girvan modularity is then defined by

$$Q_{NG}(C_n, A) := \sum_{k=1}^K \left(\frac{\mathrm{O}_{kk}}{\Lambda} - \left(\frac{\Delta_k}{\Lambda} \right)^2 \right).$$

Notice that in a network with the same vertex degrees but in which the edges are randomly generated uniformly among all pairs of vertices, the number of edges among vertices in community k is expected to be $\Lambda^{-1}\Delta_k^2$. Thus, $Q_{NG}(C_n, A)$ measures the fraction of all edges in the network that connect vertices in the same communities (the so called within-community edges) minus the expected value of the same quantity in a network with the same communities but with random vertex connections. This means that the higher the value of Q_{NG} is, the stronger the community structure is. Therefore we calculate the Newman-Girvan modularity for all possible partitions into K communities and pick the one that delivers the maximal value as our estimate, i.e.

$$\widehat{C}_n = \underset{C_n\in\Omega}{\operatorname{argmax}}\, Q_{NG}(C_n, A),$$

where Ω denotes the set of all partitions into K classes.

3.3.2 Likelihood modularity

An alternative modularity, also introduced by Bickel and Chen (2009), is based on the maximum likelihood approach. For $k \in [K]$, let N_k^n be the number of vertices in community k. Define $N_{kk}^n := N_k^n(N_k^n - 1)$ as twice the highest possible number of

edges among vertices in community k and, for $l \neq k$, $N_{kl}^n := N_k^n N_l^n$ as the highest possible number of edges between communities k and l. The probability of having x edges between communities k and l is ${P_{kl}}^x (1-P_{kl})^{N_{kl}^n - x}$ and the probability of having y edges within community k is ${P_{kk}}^y (1-P_{kk})^{\frac{1}{2}N_{kk}^n - y}$. Since the observed numbers for x and y are O_{kl} and $\frac{1}{2}O_{kk}$ respectively, the likelihood function is given by

$$\begin{aligned} & \prod_{k<l} P_{kl}^{\mathrm{O}_{kl}} (1-P_{kl})^{N_{kl}^n - \mathrm{O}_{kl}} \prod_{k\in[K]} P_{kk}^{\frac{1}{2}\mathrm{O}_{kk}} (1-P_{kk})^{\frac{1}{2}(N_{kk}^n - \mathrm{O}_{kk})} \\ = & \prod_{k\neq l} P_{kl}^{\frac{1}{2}\mathrm{O}_{kl}} (1-P_{kl})^{\frac{1}{2}(N_{kl}^n - \mathrm{O}_{kl})} \prod_{k\in[K]} P_{kk}^{\frac{1}{2}\mathrm{O}_{kk}} (1-P_{kk})^{\frac{1}{2}(N_{kk}^n - \mathrm{O}_{kk})} \\ = & \prod_{k,l\in[K]} P_{kl}^{\frac{1}{2}\mathrm{O}_{kl}} (1-P_{kl})^{\frac{1}{2}(N_{kl}^n - \mathrm{O}_{kl})}. \end{aligned}$$

The log-likelihood is

$$\tfrac{1}{2} \sum_{k,l\in[K]} \left[\mathrm{O}_{kl} \log(P_{kl}) + (N_{kl}^n - \mathrm{O}_{kl}) \log(1-P_{kl}) \right].$$

Note that each term is maximal for $P_{kl} = \frac{\mathrm{O}_{kl}}{N_{kl}^n}$. Thus, maximizing over P we find

$$Q_{LM}(C_n, A) := \tfrac{1}{2} \sum_{k,l\in[K]} \left[\mathrm{O}_{kl} \log\left(\frac{\mathrm{O}_{kl}}{N_{kl}^n}\right) + (N_{kl}^n - \mathrm{O}_{kl}) \log\left(1 - \frac{\mathrm{O}_{kl}}{N_{kl}^n}\right) \right],$$

which we call the Likelihood modularity. Since we replaced P_{kl} by $\frac{\mathrm{O}_{kl}}{N_{kl}^n}$ this is not a true likelihood, but a profile likelihood. As our estimate for the true partition we take

$$\widehat{C}_n = \operatorname*{argmax}_{C_n \in \Omega} Q_{LM}(C_n, A),$$

where Ω denotes the set of all partitions into K classes.

3.3.3 Consistency

Bickel and Chen (2009) claim that the Newman-Girvan modularity is not consistent for $K > 2$. They give the following counterexample for $K = 3$.

Let $\alpha = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ and

$$P = \begin{pmatrix} 0.66 & 0.04 & 0 \\ 0.04 & 0.12 & 0.04 \\ 0 & 0.04 & 0.06 \end{pmatrix}.$$

Let $n \to \infty$. For the true partition Q_{NG} approaches 0.3. However, when classes 2 and 3 are merged together Q_{NG} is about 0.34 (see Appendix A for the calculations). Hence, maximizing Q_{NG} does not return the true partition in this network.

The reason for the failure of the Newman-Girvan modularity in the above model is the large difference between the within-community connection probabilities. Because class 1 is very dense and classes 2 and 3 are sparse, the community structure is stronger when the latter two classes are counted as one community.

This counterexample however does not disprove the consistency of the Newman-Girvan modularity in the way we use it, as it is not taken into account that the number of classes is known. Since we assume that we know the number of classes, we do not consider a partition into two classes, when the true number of classes is three. However, when in this partition a single vertex from true class 2 or 3 is classified as a separate class, a partition into three classes is derived and the Newman-Girvan modularity of this partition is probably also about 0.34. Therefore the counterexample does make the consistency very unlikely.

As for the Likelihood modularity, Bickel and Chen (2009) claim consistency under the assumption $\frac{\lambda_n}{\log n} \xrightarrow[n\to\infty]{} \infty$, where λ_n is the expected degree of a randomly chosen vertex in the network.

3.4 Comparison of the methods

Now that we have studied the methods, we are able to compare them on computational complexity, consistency and robustness.

The method studied in Section 3.3, where we use the Newman-Girvan modularity or the Likelihood modularity, takes a long time to compute, because the number of possible partitions in K classes grows exponentially with n. For all of these partitions the modularity has to be computed, which is not very efficient even in the Planted Clustering model where $K = 2$. Therefore these methods are only computable for really small networks.

The Newman-Girvan algorithm is easier to compute. The worst case time for the algorithm is $O(m^2 n)$. However, the complexity will mostly not be that large since the number of edges decreases and since we can stop the algorithm when we are left with K components.

Nevertheless, the NG algorithm is not as easy to compute as the Largest Gaps algorithm, since the LG algorithm only uses the degrees of the vertices.

This is, however, also the weakness of the LG algorithm. Because it uses so little information the LG algorithm is only consistent under strong assumptions. For the consistency of the Likelihood modularity a weaker assumption is needed according to (Bickel and Chen, 2009). We do not know if, or under which conditions, the Newman-Girvan algorithm and the Newman-Girvan modularity are consistent.

The LG algorithm is not a robust method. In the simulations in Section 5.2.2 we find that the LG algorithm often fails when a network contains an outlier, which is a vertex with many more connections than the other vertices. In this case the LG algorithm mostly classifies the outlier as one class and all other nodes as the other

class. The NG algorithm is not much influenced by an outlier, since this does not affect the betweenness scores much.

4 Community detection for unknown number of communities

So far, we have used the fact that there are two classes. However, in general the number of classes is not known. In this section we examine two ways to use the Largest Gaps algorithm to detect the communities when the number of classes is unknown. The first one (Section 4.1) was introduced by Channarond, Daudin and Robin (2012), but is not ideal for the Planted Clustering model. Therefore we also study an own innovation, in Section 4.2, in which we combine the Largest Gaps algorithm with the Newman-Girvan modularity, which we studied in Section 3.3.1. In Section 4.3 we study the general Newman-Girvan algorithm, as introduced by Newman and Girvan (2004), which uses the Newman-Girvan modularity to estimate the number of classes.

4.1 Largest Gaps algorithm with f_K^n

The algorithm as described for $K = 2$ in section 3.1.2 can be generalized for $K \in \{2, \dots, n\}$ by looking at the $K-1$ largest gaps (in the third step of the algorithm) to find a partition in K classes. Thus, when we do not know the number of classes, we could use the LG algorithm to find a partition in K classes for every $2 \leq K \leq n$. In this section we study a method that was introduced by Channarond, Daudin and Robin (2012) for estimating the right value of K after the LG algorithm has been used to find partitions into each possible number of classes.

In this method a function f_K^n is used to estimate the number of classes. In this function the largest gaps between normalized degrees are compared to the gaps between the average normalized degrees of the estimated classes. When the estimated number of classes is right this difference is expected to be smaller than when the number of classes is underestimated.

4.1.1 Study of the largest gaps between normalized degrees

From now on, K denotes the true number of classes (i.e. $K = 2$ in the Planted Clustering model) and $\widehat{K}$ denotes the estimated number of classes. Let $(G_k^n)_{k\in[n-1]}$ be the sequence of lengths of gaps between consecutive normalized degrees, i.e. $(T_{(i+1)}^n - T_{(i)}^n)_{i\in[n-1]}$, but sorted in decreasing order, such that $G_1^n, \dots, G_{K-1}^n$ are the lengths of the $K-1$ largest gaps in the LG algorithm.

Lemma 4.1. *For all $k < K$, $\liminf_{n\to\infty} G_k^n > 0$.*

Proof. Let $k < K$. Recall the definition of A_n in Section 3.1.4. Assume that the event $B_n := A_n \cap \{d_n \leq \frac{\delta}{5}\}$ is true. According to Proposition 3.9 (with $\epsilon = 1$) the $K-1$ largest gaps lie between normalized degrees of vertices in different classes.

Therefore we have $G_k^n = |T_i^n - T_j^n|$ for some $i \in C_k^n$ and $j \in C_l^n$ where $k \neq l$. Using the triangle inequality, we obtain

$$\begin{aligned} G_k^n &= |T_i^n - T_j^n| \\ &\geq |\overline{P}_k - \overline{P}_l| - |T_i^n - \overline{P}_k| - |T_j^n - \overline{P}_l| \\ &\geq \delta - d_n - d_n \\ &\geq \tfrac{3}{5}\delta, \end{aligned}$$

which implies $B_n \subset \{G_k^n \geq \frac{3}{5}\delta\}$. This gives

$$\mathbb{P}\left(G_k^n < \tfrac{3}{5}\delta\right) \leq \mathbb{P}(B_n^c) \leq \mathbb{P}(A_n^c) + \mathbb{P}(d_n > \tfrac{\delta}{5}).$$

Using Proposition 3.10 we find

$$\mathbb{P}(d_n > \tfrac{\delta}{5}) \leq 2ne^{-\frac{2}{25}(n-1)\delta^2}.$$

Since, for all $l \in [K]$, N_l^n has a binomial distribution with parameters (n, α_l), we have

$$\mathbb{P}(A_n^c) \leq \sum_{l\in[K]} \mathbb{P}(N_l^n = 0) = \sum_{l\in[K]} (1-\alpha_l)^n \leq K(1-\alpha_{\min})^n,$$

where $\alpha_{\min} = \min_{l\in[K]} \alpha_l$. Hence

$$\mathbb{P}\left(G_k^n < \tfrac{3}{5}\delta\right) \leq 2ne^{-\frac{2}{25}(n-1)\delta^2} + K(1-\alpha_{\min})^n.$$

This upper bound is summable, so we can use the Borel-Cantelli lemma to find

$$\mathbb{P}(\limsup_{n\to\infty}\{G_k^n < \tfrac{3}{5}\delta\}) = 0,$$

which implies $\liminf_{n\to\infty} G_k^n \geq \frac{3}{5}\delta > 0$ almost surely. □

All further gaps lie between normalized degrees of vertices of the same class and converge to 0.

Lemma 4.2. *For any $\beta \in (0,1)$ and $K \leq k \leq n-1$,*

$$n^{\frac{1-\beta}{2}} G_k^n \xrightarrow[n\to\infty]{a.s.} 0.$$

Proof. It is enough to prove $n^{\frac{1-\beta}{2}} G_K^n \xrightarrow[n\to\infty]{a.s.} 0$, as all further gaps are smaller or equal to G_K^n. We know that, on the event $B_n = A_n \cap \{d_n \leq \frac{\delta}{5}\}$, the gap G_K^n lies between the normalized degrees of two vertices in the same class. As both of these normalized degrees are at most d_n away from their conditional mean, we have $G_K^n \leq 2d_n$. We find, for any $0 < t < \frac{\delta}{5}$,

$$\begin{aligned} \mathbb{P}\left(n^{\frac{1-\beta}{2}} G_K^n > t\right) &= \mathbb{P}\left(n^{\frac{1-\beta}{2}} G_K^n > t \cap B_n\right) + \mathbb{P}\left(n^{\frac{1-\beta}{2}} G_K^n > t \cap B_n^c\right) \\ &\leq \mathbb{P}\left(2n^{\frac{1-\beta}{2}} d_n > t\right) + \mathbb{P}\left(B_n^c\right). \end{aligned}$$

As we have seen in the proof of Lemma 4.1, the second term converges to 0:

$$\mathbb{P}\left(B_n^c\right) \leq 2ne^{-\frac{2}{25}(n-1)\delta^2} + K(1-\alpha_{\min})^n \xrightarrow[n\to\infty]{} 0.$$

For the first term we can use Proposition 3.10 to find

$$\begin{aligned}\mathbb{P}\left(2n^{\frac{1-\beta}{2}}d_n > t\right) &= \mathbb{P}\left(d_n > \tfrac{t}{2}n^{\frac{\beta-1}{2}}\right) \\ &\leq 2n\exp\left(-2(n-1)\tfrac{t^2}{4}n^{\beta-1}\right) \xrightarrow[n\to\infty]{} 0.\end{aligned}$$

Hence $n^{\frac{1-\beta}{2}}G_K^n \xrightarrow[n\to\infty]{} 0$ almost surely. □

4.1.2 Study of the gaps between estimated classes

Let $2 \leq \widehat{K} \leq n-1$ be our current estimate for K and suppose we have used the LG algorithm to find a partition $\{\widehat{C}_k^n\}_{k\in[\widehat{K}]}$ into $\widehat{K}$ classes. Recall that $\widehat{N}_k^n$ denotes the cardinality of estimated class k. For $1 \leq k \leq \widehat{K}$, let m_k be the average of the normalized degrees of estimated class k, i.e.

$$m_k := \frac{1}{\widehat{N}_k^n}\sum_{i\in\widehat{C}_k^n} T_i^n.$$

Now, let $(H_k^n)_{k\in[\widehat{K}-1]}$ denote the sequence of gaps between consecutive averages $(m_{(k+1)} - m_{(k)})_{k\in[\widehat{K}-1]}$, sorted in order of decreasing length. When $\widehat{K} = K$ we expect H_k^n to be close to G_k^n for $k \leq \widehat{K}-1$. However, when $\widehat{K} < K$, there is at least one k for which H_k^n stretches over more than one class and thus includes more than one G_k. Hence, when looking at the difference between the H_k^n's and the G_k^n's we will be able to see if we took $\widehat{K}$ too small, when n is large enough.

Lemma 4.3.

1. *If* $\widehat{K} = K$, *then* $\sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n) \xrightarrow[n\to\infty]{a.s.} 0$.

2. *If* $\widehat{K} < K$, *then* $\liminf_{n\to\infty}\sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n) > 0$.

Proof.

1. Assume $\widehat{K} = K$. Let $(J_k)_{k\in[K-1]}$ be the $K-1$ largest intervals between consecutive normalized degrees, such that $|J_k| = G_k^n$ for all k. Moreover, let $J_0' = [0, m_{(1)})$ and $J_K' = [m_{(K)}, 1)$ and define $H_0^n := |J_0'|$ and $H_K^n := |J_K'|$. Recall that $(H_k^n)_{k\in[K1]}$ consists of the gaps $m_{(k+1)}m_{(k)}$ between the normalized degrees. Hence $\sum_{k=0}^{K} H_k^n = 1$. The union of $J_0', J_1, \ldots, J_{K-1}, J_K'$ partially covers the interval $[0,1)$ and the gaps between the intervals are at most $2d_n$. Therefore we have

$$1 - 2Kd_n \leq \sum_{k=1}^{K-1} G_k^n + H_0^n + H_K^n \leq 1.$$

Subtracting $\sum_{k=0}^{K} H_k^n$ (which equals 1) in the above inequalities, we obtain

$$-2Kd_n \leq \sum_{k=1}^{K-1}(G_k^n - H_k^n) \leq 0.$$

Using Proposition 3.10 we find

$$\begin{aligned}\mathbb{P}\left(\sum_{k=1}^{K-1}(H_k^n - G_k^n) > t\right) &\leq \mathbb{P}(2Kd_n > t) \\ &\leq 2n\exp\left(-2(n-1)(\tfrac{t}{2K})^2\right) \\ &= 2n\exp\left(-\frac{(n-1)t^2}{2K^2}\right) \xrightarrow[n\to\infty]{} 0.\end{aligned}$$

This gives $\sum_{k=1}^{K-1}(H_k^n - G_k^n) \xrightarrow[n\to\infty]{a.s.} 0$.

2. A sketch of the proof is given.

   Assume $\widehat{K} < K$. According to equation (1) in Section 3.1.1 the normalized degrees of vertices in class $k \in [K]$ converge to $\overline{P}_k$ almost surely. This implies, for $k < K$,

   $$G_K^n \xrightarrow[n\to\infty]{a.s.} \overline{P}_{(l+1)} - \overline{P}_{(l)}$$

   for some $l < K$. Since we have $\widehat{K} < K$, at least two classes are merged together in the estimated partition. Therefore there is at least one m_q, with $q \in [\widehat{K}-1]$, that is the average of the normalized degrees of at least two true classes, say classes k and l. This means that m_q lies somewhere in between $\overline{P}_k$ and $\overline{P}_l$ as n tends to infinity. It follows that there is a $k \in [\widehat{K}-1]$ for which H_k^n stretches over more than a G_l^n as n tends to infinity. Hence, $\liminf_{n\to\infty} \sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n) > 0$.

□

4.1.3 Estimation of the number of classes

Looking at the result of Lemma 4.3 we could think that minimizing the quantity $\sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n)$ over all $\widehat{K} \in \{2, \ldots, n\}$ might give the right number of classes, as it converges to 0 for the right $\widehat{K}$, and to a positive value when $\widehat{K}$ is too small. However, for $\widehat{K} > K$ the quantity becomes smaller and eventually becomes 0 when $\widehat{K} = n$, as $(H_k^n)_{k\in[\widehat{K}-1]}$ is then equal to $(G_k^n)_{k\in[n-1]}$. Hence the minimum of $\sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n)$ is not attained at $\widehat{K} = K$, but at $\widehat{K} = n$. To deal with this, we add a penalty term to the quantity that penalizes overly small gaps. For all $\widehat{K} \in \{2, \ldots, n\}$ we define

$$f_{\widehat{K}}^n := \sum_{k=1}^{\widehat{K}-1}(H_k^n - G_k^n) + \frac{1}{n^{\frac{1-\beta}{2}} G_{\widehat{K}-1}^n} \text{ where } \beta \in (0,1).$$

Note that $f_{\widehat{K}}^n \in [0, \infty]$, since $G_{\widehat{K}-1}^n$ can be 0.

Theorem 4.4.

1. *If* $\widehat{K} = K$, *then* $f^n_{\widehat{K}} \xrightarrow[n\to\infty]{a.s.} 0$.

2. *If* $\widehat{K} < K$, *then* $\liminf_{n\to\infty} f^n_{\widehat{K}} > 0$ *a.s.*

3. *If* $\widehat{K} > K$, *then* $f^n_{\widehat{K}} \xrightarrow[n\to\infty]{a.s.} \infty$.

Proof.

1. Let $\widehat{K} = K$. From Lemma 4.3 it follows that the first term of $f^n_{\widehat{K}}$ converges to 0 almost surely. According to Lemma 4.1, $\liminf_{n\to\infty} G^n_{K-1} > 0$ almost surely. Therefore the penalty term almost surely converges to 0, hence $f^n_{\widehat{K}}$ does too.

2. Let $\widehat{K} < K$. According to Lemma 4.3, $\liminf_{n\to\infty} \sum_{k=1}^{\widehat{K}-1}(H^n_k - G^n_k) > 0$. As the penalty term is always non-negative, we find $\liminf_{n\to\infty} f^n_{\widehat{K}} > 0$.

3. Let $\widehat{K} > K$. Since $\sum_{k=1}^{\widehat{K}-1} H^n_k$ and $\sum_{k=1}^{\widehat{K}-1} G^n_k$ both lie in the interval $[0, 1]$, $\sum_{k=1}^{\widehat{K}-1}(H^n_k - G^n_k)$ is lower bounded by -1. From Lemma 4.2 it follows that the penalty term converges to ∞ almost surely, and therefore $f^n_{\widehat{K}}$ does too.

□

From Theorem 4.4 follows

$$\operatorname{argmin}_{2\le\widehat{K}\le n} f^n_{\widehat{K}} \xrightarrow[n\to\infty]{a.s.} K.$$

Therefore we take $\widehat{K} = \operatorname{argmin}_{2\le K\le n} f^n_K$ as our estimate for the true number of classes K.

4.2 Largest Gaps algorithm using modularity

The method by Channarond, Daudin and Robin (2012), which we studied in the previous section, mainly focuses on how not to underestimate the number of classes. Therefore their method is most suitable for networks with many communities and not so much for the Planted Clustering model with two classes (especially because the method excludes the possibility $K = 1$). In this section we study another way to use the Larges Gaps algorithm to identify the communities when K is unknown which, as we will see in the simulations (see Section 5.3.2), works better for the Planted Clustering model.

Again, we use the Largest Gaps algorithm to find a partition in $\widehat{K}$ classes for each $\widehat{K} \in \{2, \dots, n\}$. We also consider the possibility $K = 1$, where we have a single community that consists of all vertices. Now that we have found a unique partition in $\widehat{K}$ classes for each $\widehat{K} \in [n]$, we use the Newman-Girvan modularity (see Section 3.3.1) to determine the best estimate $\widehat{K}$ for K, i.e. we pick

$$\widehat{C}_n = \operatorname*{argmax}_{C_n\in\Pi} Q_{NG}(C_n, A),$$

where Π denotes the set of the n possible partitions we found.

For the modularity we use Q_{NG} because, if we would use Q_{LM} instead we would always find $\widehat{K} = n$. This is because the Likelihood modularity is always non positive and it equals 0 for the partition into n single-vertex communities. In general the value of Q_{LM} becomes larger as the number of classes becomes larger. Therefore it might be possible to use Q_{LM} if we add a term that penalizes overly small gaps, similar to the penalty term in f_K^n (see Section 4.1.3). However, this adaption is out of the scope of this thesis.

4.3 Newman-Girvan algorithm

The Newman-Girvan algorithm (Newman and Girvan, 2004) as described in section 3.2, can also be used when the number of classes is unknown. In this case we keep removing edges until no edges are left, such that the network falls apart into more and more communities until we end up with n single-vertex communities. This process can be displayed by a dendrogram. This is a tree whose leaves represent the vertices of the network. Every split in the tree corresponds to a component of the network splitting into two components after the removal of an edge. Therefore each horizontal cut in the dendrogram represents a partition (see Figure 2 for an example). As there are n possible cuts, associated with partitions into $1 \leq K \leq n$ classes, we have to decide which cut delivers the best partition. To do this, we use the Newman-Girvan modularity to do this. We compute Q_{NG} for each of the n partitions and choose the one that gives the maximal value as our estimate.

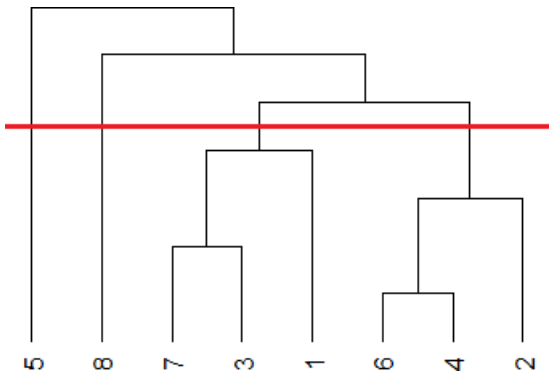

Figure 2: Dendrogram of a network with eight vertices. The red line shows a possible cut. This cut corresponds to the partition $\{\{1, 3, 7\}, \{2, 4, 6\}, \{5\}, \{8\}\}$.

5 Simulation study

To further study and compare the discussed methods we do simulations. Because of the computational complexity we only study small networks, mostly with 50 vertices. Therefore we must keep in mind that the results are not fully reliable. We will however get an indication of the strength of the methods in different situations.

In our simulations we let the parameters vary so we can study their influence and to see which method works better in what cases. To be able to tell how well a method performs, we need to somehow compare the found partitions to the true one. For this we use the Rand index, which was introduced by Rand (1971).

5.1 Rand index

Let $C_n = \{C_k^n\}_{k\in[K]}$ be the true partition and $\widehat{C}_n = \{\widehat{C}_k^n\}_{k\in[\widehat{K}]}$ the estimated one (whether $\widehat{K}$ is equal to K does not matter). Let $i, j \in [n]$, $i \neq j$, be two vertices in the network. There are four possibilities:

1. Vertices i and j are in the same class in C_n, and in the same class in $\widehat{C}_n$.
2. Vertices i and j are in different classes in C_n, and in different classes in $\widehat{C}_n$.
3. Vertices i and j are in the same class in C_n, but in different classes in $\widehat{C}_n$.
4. Vertices i and j are in different classes in C_n, but in the same class in $\widehat{C}_n$.

When 1 or 2 is the case, we say that the pair (i, j) is assessed correctly. Let r be the total number of correctly assessed pairs of vertices. The Rand index $I_R(C_n, \widehat{C}_n)$ of C_n and $\widehat{C}_n$ is defined

$$I_R(C_n, \widehat{C}_n) := \frac{r}{\binom{n}{2}},$$

which is the fraction of vertices which are assessed correctly, as $\binom{n}{2}$ is the total number of pairs. Obviously, $I_R(C_n, \widehat{C}_n)$ is in the interval $[0, 1]$, with $I_R(C_n, \widehat{C}_n) = 1$ if and only if $\widehat{C}_n = C_n$. Also, we could say that the higher $I_R(C_n, \widehat{C}_n)$ is, the better the estimate $\widehat{C}_n$ is for C^n. Therefore, we use the Rand index in our simulations to find out how well every method works in different cases.

5.2 Community detection for known number of communities

In Section 3 we studied four methods to detect communities when the number of communities is known; the Largest Gaps algorithm, the Newman-Girvan algorithm, the Newman-Girvan modularity and the Likelihood modularity. We don't do simulations for the modularities, because these are only computable for very small networks.

Therefore we only compare the LG algorithm and the NG algorithm. We also study their robustness, by introducing an adaption of the Planted Clustering model that contains an outlier.

5.2.1 Planted Clustering model

For our simulations we consider networks with $n = 50$ vertices. Because we want to see how the cardinality of S affects the results, we try three different values, namely $|S| = 12$, 25 and 38. Here it must be noted that in the real model $|S|$ is a random

variable that depends on a given parameter α. However, we choose to fix $|S|$ as this makes the simulations a lot easier. Besides, from Theorem 3.4 follows that $|S|$ is expected to be close to αn in the real model and therefore fixing $|S|$ will not affect the results much.

In order to study the influence of the connection probabilities p and q, we try three different combinations; one where p and q are far apart ($\frac{6}{7}$ and $\frac{1}{7}$), one where both are rather large ($\frac{6}{7}$ and $\frac{5}{7}$) and one where both are rather small ($\frac{2}{7}$ and $\frac{1}{7}$).

As we try three different values for $|S|$ and three inputs for the pair (p, q), we do a total of nine simulations, each consisting of 10000 iterations. In each iteration a network is generated with respect to the input parameters, and the Largest Gaps algorithm is applied to this network. Then the Rand index of the found partition with respect to the true partition is calculated. The red bars in Figure 3 show the mean of the Rand index in each case.

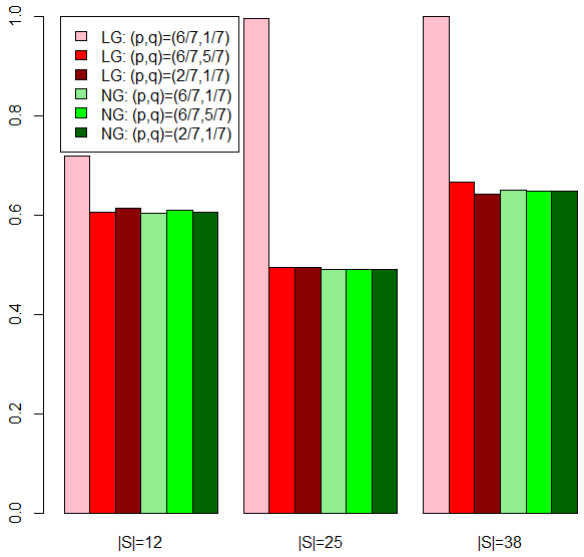

Figure 3: Mean Rand index in the Planted Clustering model. The red and green bars show the mean of the Rand index found using the Largest Gaps algorithm and the Newman-Girvan algorithm respectively.

When we evaluate these results, we keep in mind that the LG algorithm is expected to work the best when δ_n is large. In general, we have

$$\delta_n = \overline{P}_1 - \overline{P}_2 \approx \left(\tfrac{|S|}{n}p + \left(1 - \tfrac{|S|}{n}\right)q\right) - q = \tfrac{|S|}{n}(p - q).$$

Therefore, as for p and q, it is only the difference $p - q$ that matters. In Figure 3 we see that indeed $(p, q) = (\frac{6}{7}, \frac{5}{7})$ and $(p, q) = (\frac{2}{7}, \frac{1}{7})$ give equal results. When we look at the cardinality of S, we see that δ_n gets larger as $|S|$ gets larger. In the figure we see that, for $(p, q) = (\frac{6}{7}, \frac{1}{7})$, the results indeed get better for larger $|S|$. However, in the other two cases, we see that the results are worst when $|S| = 10$. It could be that identifying two communities of equal size is harder than one small and one large community.

The same simulations were done for the NG algorithm. The green bars in Figure 3 show the results. The mean Rand index does not differ much for different values of p and q. Upon studying the output of the algorithm closer, we noticed that a lot of times the NG algorithm finds a partition into classes of sizes 1 and 49. Apparently there is not enough community structure, which causes the NG algorithm to only cut off a single vertex most of the time.

We thus see that when p and q are close the performance of both the LG algorithm and the NG algorithm leaves something to be desired. For $(p, q) = (\frac{6}{7}, \frac{1}{7})$ the LG algorithm performs much better. In this case it finds the true partition in 3%, 95% and 99.99% of the time for $|S| = 12$, $|S| = 25$ and $|S| = 38$ respectively, whereas the NG algorithm never finds the true partition.

The exact results can be found in Appendix B.1.

5.2.2 Planted Clustering model with an outlier

To study the robustness of the methods, we consider an adapted version of the Planted Clustering model that includes an outlier. We pick a vertex $*$ in S and introduce a probability $p < p^* < 1$. The probability of an edge between vertices i and j is

$$\begin{cases} p^* & \text{if } i, j \in S \text{ and } i = * \text{ or } j = * \\ p & \text{if } i, j \in S \text{ and } i \neq * \neq j \\ q & \text{otherwise.} \end{cases}$$

The probability of an edge between the outlier $*$ and another vertex in S is now larger than the other probabilities. Therefore $*$ has a larger expected degree, which can cause the LG algorithm to classify $*$ as a single-vertex class and all other vertices as the other class.

In our simulations we take $q = \frac{1}{7}$, $p = \frac{4}{7}$ and $p^* = 1$, such that the outlier is always connected to all other vertices in S. Again we consider networks with $n = 50$ vertices and $|S| = 12$, 25 or 38.

The darkred and darkgreen bars in Figure 4 show the resulting mean Rand index using the LG algorithm and the NG algorithm respectively. To study the influence of the outlier we did the same simulations with $p^* = \frac{4}{7} = p$, such that the original model is obtained. The results are displayed by the lighter red and green bars in Figure 4.

We see that the outlier does not affect the results when we use the NG algorithm. This is not very surprising since the NG algorithm already performs poorly in the model without the outlier.

When $|S| = 12$, the LG algorithm is not much affected by the outlier. It even performs a little better. This may be because the average connection probability of vertices in S is increased because of the outlier. Therefore the expected gap between the normalized degrees of vertices in S and the normalized degrees of the other vertices is larger than in the original model.

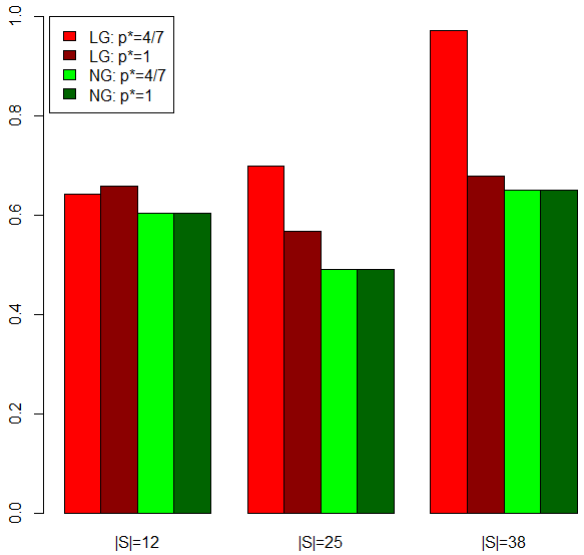

Figure 4: Mean Rand index in the Planted Clustering model with an outlier. The red and green bars show the mean of the Rand index found using the Largest Gaps algorithm and the Newman-Girvan algorithm respectively.

For $|S| = 25$ and $|S| = 38$ we see that the results are more like we expected, as there is a clear drop in the mean Rand index when the outlier is added. It turns out that the LG algorithm separates the outlier from the other vertices in over 67% of the time when $|S| = 25$. For $|S| = 38$ this happens 81% of the time even. Nevertheless, concerning the mean Rand index, the LG algorithm still performs better than the NG algorithm.

The exact results can be found in Appendix B.2.

5.3 Community detection for unknown number of communities

In Section 4 we studied three methods to detect communities when the number of communities is not known; the Largest Gaps algorithm with f_K^n, the Largest Gaps algorithm with the Newman-Girvan modularity Q_{NG} and the Newman-Girvan algorithm.

In Section 5.3.1 we study the performance of the LG algorithm with Q_{NG} and the NG algorithm on the Planted Clustering model. We do not include the Largest Gaps algorithm with f_K^n in these simulations because this method is not suitable for networks with only two classes (recall Sections 4.1 and 4.2). Therefore, in Section 5.3.2, we study a Planted Clustering model with three classes and do simulations using the LG algorithm with f_K^n and the LG algorithm with Q_{NG}.

5.3.1 Planted Clustering model

Again, we consider networks generated by the Planted Clustering model with parameters $n = 50$, $|S| = 12$, 25 or 38 and $(p, q) = (\frac{6}{7}, \frac{1}{7})$, $(\frac{6}{7}, \frac{5}{7})$ or $(\frac{2}{7}, \frac{1}{7})$.

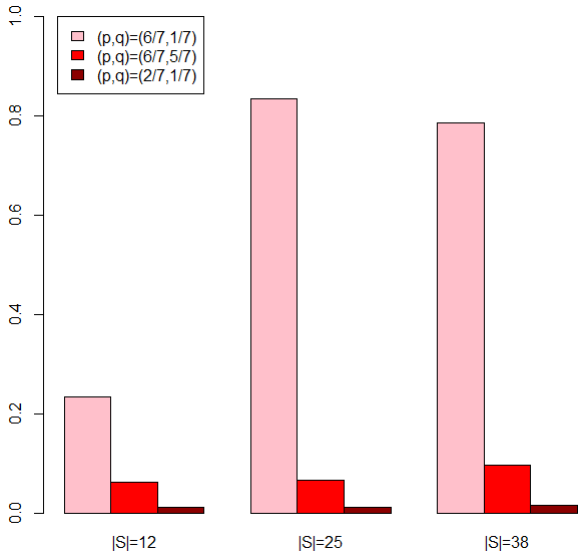

Figure 5: Fraction of estimated partitions, found using the Largest Gaps algorithm with the Newman-Girvan modularity Q_{NG}, in which the number of classes is $\widehat{K} = 2$.

Figure 5 shows how often the LG algorithm with Q_{NG} finds the true number of classes. When p and q are close the method rarely succeeds. Upon studying the results closer, we noticed that in the case $(p, q) = (\frac{2}{7}, \frac{1}{7})$ is mostly overestimated. However, for $(p, q) = (\frac{6}{7}, \frac{5}{7})$, the method fails mostly because $\widehat{K} = 1$ is found. In this case the networks often are dense and do not have a strong community structure and therefore Q_{NG} is often negative for all found partitions in two or more classes. Therefore $\widehat{K} = 1$, for which the Newman-Girvan modularity is always 0, maximizes Q_{NG}.

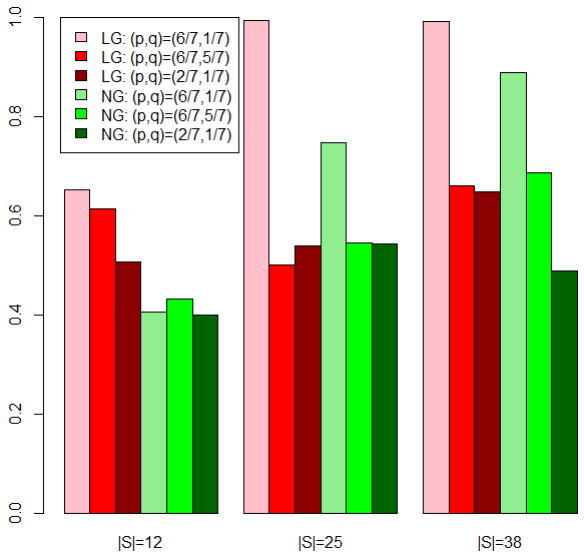

Figure 6: Mean Rand index in the Planted Clustering model. The red and green bars show the mean of the Rand index found using the Largest Gaps algorithm with the Newman-Girvan modularity Q_{NG} and the Newman-Girvan algorithm respectively.

The Newman-Girvan algorithm never finds the right number of classes for any of the input parameters. We found that it often considerably overestimates the number of classes. In many cases S is identified as one class, but all other vertices are put in separate classes. When this is the case all vertex pairs containing one or more vertices of S are assessed correctly. Therefore the mean Rand index, which is shown in Figure 5, is mostly larger when $|S|$ is larger, for the NG algorithm.

Concerning the mean Rand index, the LG algorithm performs better than the NG algorithm when $(p, q) = (\frac{6}{7}, \frac{1}{7})$, as was the case when K was unknown. When p and q are close the results vary for both methods.

The exact results can be found in Appendix B.3.

5.3.2 Planted Clustering model with three classes

To study the LG algorithm with f_K^n we consider a model with three classes that is based on the Planted Clustering model that was discussed by Chen and Xu (2015). The model we use is described below. In the simulations we only consider one input value for the connection probabilities and the sizes of the classes, because the influence of these parameters is not our main interest here. We are more interested in how large the network has to be until the the method performs well. Therefore we vary the number of vertices n. Secondly, recall that the function f_K^n depends on a parameter $\beta \in (0, 1)$ (see Section 4.1.3). To study the influence of this parameter we try the values $\beta = \frac{1}{10}$, $\frac{1}{2}$ and $\frac{9}{10}$.

The model we use is as follows. We consider networks with $n = 20$, 50, 80 or 100 vertices. In the set of vertices we define two disjoints subsets S_1 and S_2, satisfying $|S_1| = |S_2| = \frac{2n}{5}$. The probability of an edge between vertices i and j is

$$\begin{cases} \frac{9}{10} & \text{if } i, j \in S_1 \\ \frac{1}{2} & \text{if } i, j \in S_2 \\ \frac{1}{10} & \text{otherwise.} \end{cases}$$

It must be noted that these probabilities are very favorable and not realistic, since the connection probabilities in real networks are mostly small and not that far apart. However, because our main interest here is to study if the right number of classes can be identified, we choose parameters on which the LG algorithm works well.

The blue bars in Figure 7 show how often the true number of classes is found, for different values of n and β. Recall

$$f_{\widehat{K}}^n := \sum_{k=1}^{\widehat{K}-1} (H_k^n - G_k^n) + \frac{1}{n^{\frac{1-\beta}{2}} G_{\widehat{K}-1}^n} \text{ where } \beta \in (0, 1).$$

For small β the penalty term is small and therefore partitions into many classes are expected. In our simulations the number of classes is indeed considerably overestimated when $\beta = \frac{1}{10}$. Figure 7 shows that in this case the true number of classes, except for a few times when $n = 20$, is never found. However, also for the other

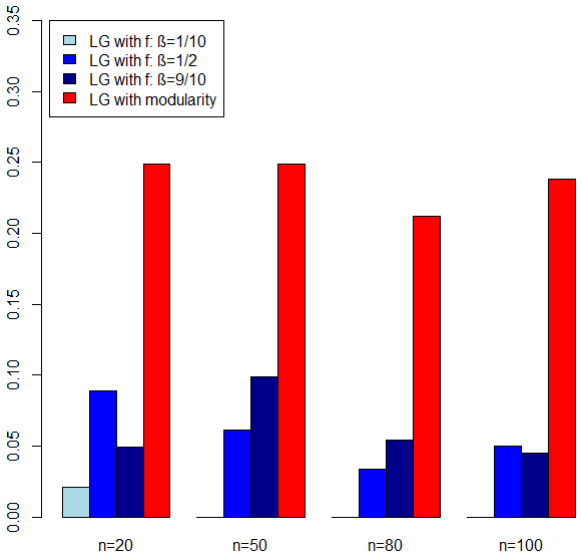

Figure 7: Fraction of estimated partitions in which the number of classes is $\widehat{K} = 3$. The blue bars show the results when the Largest Gaps algorithm with f_K^n is used, for three different values of β. The red bars show the results for the Largest Gaps algorithm with the Newman-Girvan modularity.

values of β the method performs poorly, as the true number of classes is never found in more than 10% of the time. Also, whether $\beta = \frac{1}{2}$ or $\beta = \frac{9}{10}$ works better, varies for different values of n, which means it is difficult to know what choice of β is best.

Therefore we did the same simulations for the Largest Gaps algorithm with the Newman-Girvan modularity, of which the results are displayed by the red bars in Figure 7. These results are clearly better, as the true number of classes is found in between 20% and 25% of the time. However, the results do not improve for increasing n.

Concerning the mean Rand index, the LG algorithm with Q_{NG} does perform better as n increases. This is shown in Figure 8. The LG algorithm with f_K^n performs poorly for all values of β and n. Remarkably, the mean Rand index is even the largest when $\beta = \frac{1}{10}$, in which case the number of classes is considerably overestimated.

The exact results can be found in Appendix B.4.

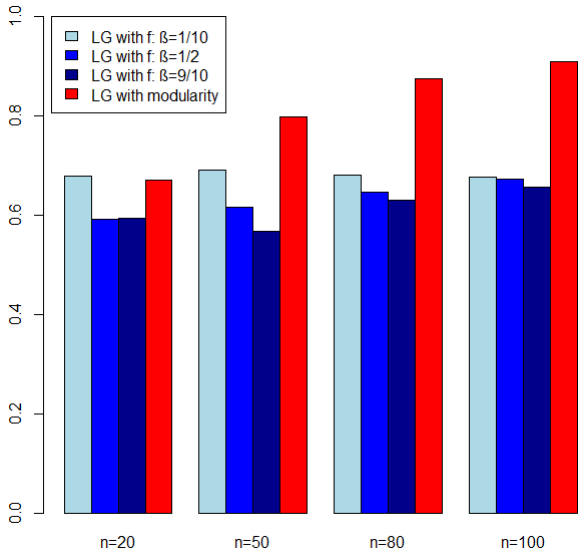

Figure 8: Mean Rand index in the Planted Clustering model with three classes. The blue bars show the mean of the Rand index found using the Largest Gaps algorithm with f_K^n, for three different values of β. The red bars show the mean of the Rand index found using the Largest Gaps algorithm with the Newman-Girvan modularity.

6 Conclusion

In this thesis we discussed several methods for community detection. In Section 3.4 we already compared the methods that can be used when the number of classes is known. We found that the computation time of modularity maximization grows exponentially with n, which makes the method only computable for very small networks. Therefore we were not able to include modularity maximization in our simulation study.

The other two methods we studied are the Largest Gaps algorithm and the Newman-Girvan algorithm. The former has the advantage that it is computable for very large networks. It is however only consistent under strong conditions. We do not know if, or under which conditions, the NG algorithm is consistent, but in general it is considered a better method than the LG algorithm. However, our simulation study shows that the NG algorithm performs poorly on the Planted Clustering model, while the LG algorithm gives good results provided that the connection probabilities p and q are far apart. Our simulations on the model with an outlier confirmed that the LG algorithm is not robust.

To detect the number of classes, when this number is not known, we studied three methods. The LG algorithm with f_K^n showed poor results in our simulation study. Our own invention, the LG algorithm combined with the Newman-Girvan modularity, works much better according to our simulations.

The Newman-Girvan algorithm, that also uses Q_{NG} to detect the number of classes, showed very poor results concerning the estimated number of classes. However, this is mainly because all vertices that are not in S are often classified as single-vertex classes. S itself is often identified as being one class.

This leads to a new idea for the case when it is known that there are two classes. In the way we used the Newman-Girvan algorithm in this case, which was stopping the algorithm as soon as the network splits into two components, it failed because mostly only one vertex was seperated from the others. It may work better to use the NG algorithm as if the number of classes is unknown and merge all single-vertex communities together afterwards. However, this idea is out of the scope of this thesis.

Overall it turns out that, even though it is not considered a very reliable method in general, the LG algorithm is, of all methods we considered, the most convenient for the Planted Clustering model. When the number of classes is unknown, the LG algorithm shows the best results when it is combined with the Newman-Girvan modularity.

References

Bickel, P.J., Chen, A. (2009). 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** (50), 21068-21073.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology* **25**, 163.

Channarond, A., Daudin, J.-J., Robin, S. (2012). Classification and estimation in the Stochastic Blockmodel based on the empirical degrees. *Electronic Journal of Statistics* **6**, 2574-2601.

Chen, Y., Xu, J. (2015). Statistical-Computational Tradeoffs in Planted Problems and Submatrix Localization with a Growing Number of Clusters and Submatrices. *arXiv* 1402.1267v3.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* **58** (301), 13-30.

Hubert, L., Arabie, P. (1985). Comparing partitions. *Journal of Classification* **2** (1), 193-218.

Newman, M.E.J. (2001). Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical Review E* **64**, 016132.

Newman, M.E.J., Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E* **69**, 026113.

Rand, W.M. (1971), Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association* **66**, 846-850.

Van der Vaart, A.W., Wellner, J.A. (1996). *Weak Convergence and Empirical Processes*. Springer (New York).

A Appendix A

Let $\alpha = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ and

$$P = \begin{pmatrix} 0.66 & 0.04 & 0 \\ 0.04 & 0.12 & 0.04 \\ 0 & 0.04 & 0.06 \end{pmatrix}.$$

First we calculate Q_{NG} for the true partition C_n as $n \to \infty$. For n large enough there will be about $\alpha_k n = \frac{n}{3}$ vertices in each class k. Therefore we have $\mathrm{O}_{kl} \approx \frac{n^2}{9} \cdot P_{kl}$ for all k, l. We find

$$\begin{aligned} \mathrm{O}_{11} &\approx \tfrac{n^2}{9} \cdot 0.66, \quad \mathrm{O}_{22} \approx \tfrac{n^2}{9} \cdot 0.12, \quad \mathrm{O}_{33} \approx \tfrac{n^2}{9} \cdot 0.06, \\ \Delta_1 &\approx \tfrac{n^2}{9} \cdot 0.7, \quad \Delta_2 \approx \tfrac{n^2}{9} \cdot 0.2, \quad \Delta_3 \approx \tfrac{n^2}{9} \cdot 0.1, \\ \Lambda &\approx \tfrac{n^2}{9}. \end{aligned}$$

We obtain

$$\lim_{n\to\infty} Q_{NG}(C_n, A) = (0.66 - 0.7^2) + (0.12 - 0.2^2) + (0.06 - 0.1^2) = 0.3 \ a.s.$$

Let $\widehat{C}_n$ be the partition into two classes, with class 1 being the true class 1 and class 2 being true classes 2 and 3 merged together. For n large enough there will be about $\frac{n}{3}$ vertices in class 1 and $\frac{2n}{3}$ vertices in class 2. The connection probabilities in the estimated classes are

$$\begin{aligned} \widehat{P}_{11} &= P_{11} = 0.66, \\ \widehat{P}_{12} &= \tfrac{1}{2}(P_{12} + P_{13}) = 0.02, \\ \widehat{P}_{22} &= \tfrac{1}{4}(P_{22} + 2P_{23} + P_{33}) = 0.065. \end{aligned}$$

Therefore we have

$$\begin{aligned} \mathrm{O}_{11} &\approx \tfrac{n^2}{9} \cdot 0.66, \quad \mathrm{O}_{12} \approx \tfrac{2n^2}{9} \cdot 0.02, \quad \mathrm{O}_{22} \approx \tfrac{4n^2}{9} \cdot 0.065, \\ \Delta_1 &\approx \tfrac{n^2}{9} \cdot 0.7, \quad \Delta_2 \approx \tfrac{n^2}{9} \cdot 0.3, \quad \Lambda \approx \tfrac{n^2}{9}. \end{aligned}$$

We obtain

$$\lim_{n\to\infty} Q_{NG}(\widehat{C}_n, A) = (0.66 - 0.7^2) + (4 \cdot 0.065 - 0.3^2) = 0.34 \ a.s.$$

B Appendix B

B.1 Planted Clustering model for known number of classes

The table below shows the results of the simulations discussed in Section 5.2.1. The number of vertices $n = 50$ is fixed. In the table $\mathbb{E}(I_R)$ denotes the mean Rand index and $\widehat{C}_n = C_n$ denotes the fraction of estimated partitions which are equal to the true partition.

$\lvert S\rvert$	(p,q)	LG algorithm		NG algorithm	
		$\mathbb{E}(I_R)$	$\widehat{C}_n = C_n$	$\mathbb{E}(I_R)$	$\widehat{C}_n = C_n$
12	$(\frac{6}{7}, \frac{1}{7})$	0.72	0.03	0.60	0
	$(\frac{6}{7}, \frac{5}{7})$	0.61	0	0.61	0
	$(\frac{2}{7}, \frac{1}{7})$	0.61	0	0.61	0
25	$(\frac{6}{7}, \frac{1}{7})$	0.997	0.95	0.49	0
	$(\frac{6}{7}, \frac{5}{7})$	0.49	0	0.49	0
	$(\frac{2}{7}, \frac{1}{7})$	0.49	0	0.49	0
38	$(\frac{6}{7}, \frac{1}{7})$	0.99999	0.9999	0.65	0
	$(\frac{6}{7}, \frac{5}{7})$	0.67	0	0.65	0
	$(\frac{2}{7}, \frac{1}{7})$	0.64	0	0.65	0

B.2 Planted Clustering model with an outlier for known number of classes

The table below shows the results of the simulations discussed in Section 5.2.2. The number of vertices $n = 50$ and the connection probabilities $q = \frac{1}{7}$ and $p = \frac{4}{7}$ are fixed. In the table $\mathbb{E}(I_R)$ denotes the mean Rand index, $\widehat{C}_n = C_n$ denotes the fraction of estimated partitions which are equal to the true partition and 'Outlier' denotes the fraction of estimated partitions in which the outlier is separated from the other vertices.

$\lvert S\rvert$	p^*	LG algorithm			NG algorithm		
		$\mathbb{E}(I_R)$	$\widehat{C}_n = C_n$	Outlier	$\mathbb{E}(I_R)$	$\widehat{C}_n = C_n$	Outlier
12	$\frac{4}{7}$	0.64	0.0001	-	0.60	0	-
	1	0.66	0.0004	0.21	0.60	0	0
25	$\frac{4}{7}$	0.70	0.10	-	0.49	0	-
	1	0.57	0.06	0.67	0.49	0	0
38	$\frac{4}{7}$	0.97	0.77	-	0.65	0	-
	1	0.68	0.18	0.81	0.65	0	0

B.3 Planted Clustering model for unknown number of classes

The table below shows the results of the simulations discussed in Section 5.3.1. The number of vertices $n = 50$ is fixed. In the table $\mathbb{E}(I_R)$ denotes the mean Rand index, $\widehat{K} = 2$ denotes the fraction of estimated partitions into the true number of classes and $\widehat{C}_n = C_n$ denotes the fraction of estimated partitions which are equal to the true partition.

		LG algorithm			NG algorithm		
$\lvert S\rvert$	(p,q)	$\mathbb{E}(I_R)$	$\widehat{K}=2$	$\widehat{C}_n=C_n$	I_R	$\widehat{K}=2$	$\widehat{C}_n=C_n$
12	$(\frac{6}{7},\frac{1}{7})$	0.65	0.23	0.03	0.41	0	0
	$(\frac{6}{7},\frac{5}{7})$	0.61	0.06	0	0.61	0	0
	$(\frac{2}{7},\frac{1}{7})$	0.51	0.01	0	0.61	0	0
25	$(\frac{6}{7},\frac{1}{7})$	0.99	0.84	0.84	0.49	0	0
	$(\frac{6}{7},\frac{5}{7})$	0.50	0.07	0	0.49	0	0
	$(\frac{2}{7},\frac{1}{7})$	0.54	0.01	0	0.49	0	0
38	$(\frac{6}{7},\frac{1}{7})$	0.99	0.79	0.77	0.65	0	0
	$(\frac{6}{7},\frac{5}{7})$	0.66	0.10	0	0.65	0	0
	$(\frac{2}{7},\frac{1}{7})$	0.65	0.02	0	0.65	0	0

B.4 Planted Clustering model with three classes

The table below shows the results of the simulations discussed in Section 5.3.2. The cardinalities $|S_1| = |S_2| = \frac{2n}{5}$ and the connection probabilities $p_1 = \frac{9}{10}$, $p_2 = \frac{1}{2}$ and $q = \frac{1}{10}$ are fixed. In the table $\mathbb{E}(I_R)$ denotes the mean Rand index, $\widehat{K}=3$ denotes the fraction of estimated partitions into the true number of classes, $\mathbb{E}(\widehat{K})$ denotes the average number of classes in the estimated partitions and $\widehat{C}_n = C_n$ denotes the fraction of estimated partitions which are equal to the true partition.

n	LG algorithm with	$\mathbb{E}(I_R)$	$\widehat{K}=3$	$\mathbb{E}(\widehat{K})$	$\widehat{C}_n=C_n$
20	f_K^n with $\beta=\frac{1}{10}$	0.68	0.02	7.98	0
	f_K^n with $\beta=\frac{1}{2}$	0.59	0.09	4.60	0
	f_K^n with $\beta=\frac{9}{10}$	0.59	0.05	4.64	0
	Q_{NG}	0.67	0.25	3.60	0.001
50	f_K^n with $\beta=\frac{1}{10}$	0.69	0	20.3	0
	f_K^n with $\beta=\frac{1}{2}$	0.62	0.06	9.26	0
	f_K^n with $\beta=\frac{9}{10}$	0.57	0.10	2.9	0
	Q_{NG}	0.79	0.25	5.13	0.003
80	f_K^n with $\beta=\frac{1}{10}$	0.68	0	30.2	0
	f_K^n with $\beta=\frac{1}{2}$	0.65	0.03	10.78	0
	f_K^n with $\beta=\frac{9}{10}$	0.63	0.05	2.26	0
	Q_{NG}	0.88	0.21	4.55	0.02
100	f_K^n with $\beta=\frac{1}{10}$	0.68	0	35.7	0
	f_K^n with $\beta=\frac{1}{2}$	0.67	0.05	6.56	0
	f_K^n with $\beta=\frac{9}{10}$	0.66	0.05	2.13	0
	Q_{NG}	0.91	0.24	4.00	0.05

C Appendix C (R code)

C.1 LG algorithm on PC model when K is known

```
#LARGEST GAPS ALGORITHM APPLIED TO THE PLANTED CLUSTERING MODEL WHEN
#THE NUMBER OF CLASSES IS KNOWN

library(igraph)

#PARAMETERS
t <- 10000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
p <- 6/7 #Connection probability p
q <- 1/7 #Connection probability q

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)

for(k in 1:t){
    x <- matrix(0, n, n) #Adjecency matrix

    for(i in 1:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    d <- sort(degree(graph), decreasing = FALSE, index.return = TRUE)
```

```
    #Sorted degree sequence

    gaps <- d$x[2:n] - d$x[1:(n-1)] #Gaps between consecutive degrees

    lg <- which.max(gaps) #Position of largest gap

    comm <- vector(length = n) #Estimated partition

    for(i in 1:lg){
        comm[d$ix[i]] <- 1
    }
    for(i in (lg+1):n){
        comm[d$ix[i]] <- 0
    }

    Rand[k] <- compare(z, comm, method = "rand") #Rand index
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
```

C.2 NG algorithm on PC model when K is known

```
#NEWMAN-GIRVAN ALGORITHM APPLIED TO THE PLANTED CLUSTERING MODEL WHEN
#THE NUMBER OF CLASSES IS KNOWN

library(igraph)

#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
p <- 6/7 #Connection probability p
q <- 1/7 #Connection probability q

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)
minN <- vector(length = t)

for(k in 1:t){
    x <- matrix(0, n, n) #Adjecency matrix

    for(i in 1:(s-1)){
        for(j in (i+1):s){
```

```
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    comm <- cutat(edge.betweenness.community(graph),2) #Estimated
            #partition

    minN[k] <- min(sum(comm==1),sum(comm==2)) #Cardinality of
    #smallest estimated class
    Rand[k] <- compare(z, comm, method = "rand") #Rand index
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
mean(minN) #Mean cardinality of smallest class
sum(minN==1)/t #Fraction of partitions with a class of size 1
```

C.3 LG algorithm on PC model with outlier when K is known

```
#LARGEST GAPS ALGORITHM APPLIED TO THE PLANTED CLUSTERING MODEL WITH
#AN OUTLIER WHEN THE NUMBER OF CLASSES IS KNOWN

library(igraph)

#PARAMETERS
t <- 10000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
pp <- 1 #Connection probability p*
p <- 4/7 #Connection probability p
q <- 1/7 #Connection probability q
```

```
v <- rep(1,n) #Partition that separates the outlier from the rest
v[1] <- 0

z <- vector(,n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)
Rand2 <- vector(length = t)

for(k in 1:t){
    x <- matrix(0, n, n) #Adjecency matrix

    for(j in 2:s){
        x[1,j] <- rbinom(1, 1, pp)
        x[j,1] <- x[1,j]
    }
    for(i in 2:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    d <- sort(degree(graph), decreasing = FALSE, index.return = TRUE)
    #Sorted degree sequence

    gaps <- d$x[2:n] - d$x[1:(n-1)] #Gaps between consecutive degrees

    lg <- which.max(gaps) #Position of largest gap
```

```
    comm <- vector(length = n) #Estimated partition

    for(i in 1:lg){
        comm[d$ix[i]] <- 1
    }
    for(i in (lg+1):n){
        comm[d$ix[i]] <- 0
    }

    Rand[k] <- compare(z, comm, method = "rand") #Rand index
    Rand2[k] <- compare(v, comm, method = "rand")
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
sum(Rand2>0.9999)/t #Fraction partitions that separate outlier
```

C.4 NG algorithm on PC model with outlier when K is known

```
#NEWMAN-GIRVAN ALGORITHM APPLIED TO THE PLANTED CLUSTERING MODEL WITH
#AN OUTLIER WHEN THE NUMBER OF CLASSES IS KNOWN

library(igraph)

#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
pp <- 1 #Connection probability p*
p <- 4/7 #Connection probability p
q <- 1/7 #Connection probability q

v <- rep(1,n) #Partition that separates the outlier from the rest
v[1] <- 0

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)
Rand2 <- vector(length = t)

for(k in 1:t){
    x <- matrix(0, n, n)

    for(j in 2:s){
```

```
        x[1,j] <- rbinom(1, 1, pp)
        x[j,1] <- x[1,j]
    }
    for(i in 2:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    comm <- cutat(edge.betweenness.community(graph),2) #Estimated
            #partition

    Rand[k] <- compare(z, comm, method = "rand") #Rand index
    Rand2[k] <- compare(v, comm, method = "rand")
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
sum(Rand2>0.9999)/t #Fraction partitions that separate outlier
```

C.5 LG algorithm with Q_{NG} on PC model when K is unknown

```
#LARGEST GAPS ALGORITHM WITH Q_NG APPLIED TO THE PLANTED CLUSTERING
#MODEL WHEN THE NUMBER OF CLASSES IS UNKNOWN

library(igraph)

#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
```

```
p <- 6/7 #Connection probability p
q <- 1/7 #Connection probability q

z <- vector(,n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)
Khat <- vector(length = t)

for(r in 1:t){
    x <- matrix(0, n, n) #Adjacency matrix

    for(i in 1:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    d <- sort(degree(graph), decreasing=FALSE, index.return=TRUE)

    gaps <- d$x[2:n] - d$x[1:(n-1)] #Gaps between consecutive degrees

    lg <- sort.int(gaps, decreasing = TRUE, method = "quick",...
          index.return = TRUE)$ix #Positions of gaps sorted by size

    comms <- matrix(0,n,n) #Estimated partition for each possible K

    for(k in 1:(n-1)){
        lgk <- sort(lg[1:k], decreasing = FALSE)
        lgk[k+1] <- #Positions of k largest gaps
```

```
        for(i in 1:k){
            for(j in (lgk[i]+1):lgk[i+1]){
                comms[k+1,d$ix[j]] <- i
            }
        }
    }

    QNGs <- vector(length = n)

    for(l in 1:n){
        k <- max(comms[l,])+1 #Current estimated number of classes

        O <- matrix(0,k,k) #O_kl values
        D <- vector(length = k) #D_k values
        Q <- vector(length = k) #Terms of Q_NG

        for(i in 1:k){
            for(j in i:k){
                O[i,j] <- sum(x[which(comms[l,] == (i-1)),...
                        which(comms[l,] == (j-1))])
                O[j,i] <- O[i,j]
            }
        }
        D <- rowSums(O)
        L <- sum(D)
        for(i in 1:k){
            Q[i] <- O[i,i]/L - (D[i]/L)^2
        }
        QNGs[l] <- sum(Q) #Q_NG
    }
    comm <- comms[which.max(QNGs),] #Estimated partition
    Rand[r] <- compare(z, comm, method = "rand") #Rand index
    Khat[r] <- length(unique(comm)) #Estimated number of communities
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
mean(Khat) #Mean number of classes
sum(Khat==2)/t #Fraction true number of classes
```

C.6 NG algorithm on PC model when K is unknown

```
#NEWMAN-GIRVAN ALGORITHM APPLIED TO THE PLANTED CLUSTERING MODEL WHEN
#THE NUMBER OF CLASSES IS UNKNOWN

library(igraph)
```

```
#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 12 #Cardinality of S
p <- 6/7 #Connection probability p
q <- 1/7 #Connection probability q

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S are labeled 0
z[(s+1):n] <- 1 #Other vertices are labeled 1

Rand <- vector(length = t)
Khat <- vector(length = t)

for(k in 1:t){
    x <- matrix(0, n, n) #Adjacency matrix

    for(i in 1:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }

    graph <- graph.adjacency(x, mode="undirected")

    comm <- edge.betweenness.community(graph)[[6]] #Estimated
          #partition

    Rand[k] <- compare(z, comm, method = "rand") #Rand index
    Khat[k] <- length(unique(comm)) #Estimated number of communities
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
```

```
mean(Khat) #Mean number of classes
sum(Khat==2)/t #Fraction true number of classes
```

C.7 LG algorithm with f_K^n on PC model with 3 classes when K is unknown

```
#LARGEST GAPS ALGORITHM WITH f^n_K APPLIED TO THE PLANTED CLUSTERING
#MODEL WITH 3 CLASSES WHEN THE NUMBER OF CLASSES IS UNKNOWN

library(igraph)

#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 2*n/5  #Cardinality of S_1 and S_2
p1 <- 9/10 #Connection probability p_1
p2 <- 5/10 #Connection probability p_2
q <- 1/10 #Connection probability q
beta <- 1/10

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S_1 are labeled 0
z[(s+1):(2*s)] <- 1 #Vertices in S_2 are labeled 1
z[(2*s+1):n] <- 2 #Other vertices are labeled 2

Rand <- vector(length = t)
Khat <- vector(length = t)

for(r in 1:t){
    x <- matrix(0, n, n) #Adjacency matrix

    for(i in 1:(n-1)){
        for(j in (i+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in 1:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p1)
            x[j, i] <- x[i, j]
            x[i+s, j+s] <- rbinom(1, 1, p2)
            x[j+s, i+s] <- x[i+s, j+s]
        }
    }
```

```
    graph <- graph.adjacency(x, mode="undirected")

    d <- sort(degree(graph), decreasing = FALSE, index.return = TRUE)
    #Sorted degree sequence

    gaps <- d$x[2:n] - d$x[1:(n-1)] #Gaps between consecutive degrees

    G <- sort(gaps, decreasing = TRUE, method = "quick") #Gaps sorted
         #by size
    lg <- sort.int(gaps, decreasing = TRUE, method = "quick",...
        index.return = TRUE)$ix #Positions of gaps sorted by size

    comms <- matrix(0,n,n) #Estimated partition for each possible K

    for(k in 1:(n-1)){
        lgk <- sort(lg[1:k], decreasing = FALSE)
        lgk[k+1] <- n
        for(i in 1:k){
            for(j in (lgk[i]+1):lgk[i+1]){
                comms[k+1,d$ix[j]] <- i
            }
        }
    }
    f <- vector(,n) #f^n_K for each possible K
    f[1] <- Inf

    for(k in 2:n){
        m <- vector(,k) #m_k values
        for(l in 1:k){
            classl <- comms[k,] == (l-1)
            m[l] <- sum(degree(graph)[classl]) / sum(classl)
        }
        m <- sort(m, decreasing = FALSE, method = "quick")
        H <- vector(,k-1)

        H <- m[2:k] - m[1:(k-1)]

        H <- sort(H, decreasing = TRUE, method = "quick")
        f[k] <- (sum(H-G[1:k-1]))/(n-1) + 1/(n^((1-beta)/2)*G[k-1])
    }

    comm <- comms[which.min(f),] #Estimated partition
    Rand[r] <- compare(z, comm, method = "rand") #Rand index
    Khat[r] <- length(unique(comm)) #Estimated number of communities
}
mean(Rand) #Mean Rand index
```

```
sum(Rand>0.9999)/t #Fraction true partitions
mean(Khat) #Mean number of classes
sum(Khat==3)/t #Fraction true number of classes
```

C.8 LG algorithm with Q_{NG} on PC model with 3 classes when K is unknown

```
#LARGEST GAPS ALGORITHM WITH Q_NG APPLIED TO THE PLANTED CLUSTERING
#MODEL WITH 3 CLASSES WHEN THE NUMBER OF CLASSES IS UNKNOWN

library(igraph)

#PARAMETERS
t <- 1000 #Iterations
n <- 50 #Vertices
s <- 2*n/5  #Cardinality of S_1 and S_2
p1 <- 9/10 #Connection probability p_1
p2 <- 5/10 #Connection probability p_2
q <- 1/10 #Connection probability q

z <- vector(length = n) #True partition
z[1:s] <- 0 #Vertices in S_1 are labeled 0
z[(s+1):(2*s)] <- 1 #Vertices in S_2 are labeled 1
z[(2*s+1):n] <- 2 #Other vertices are labeled 2

Rand <- vector(length = t)
Khat <- vector(length = t)

for(r in 1:t){
    x <- matrix(0, n, n) #Adjacency matrix

    for(i in 1:(s-1)){
        for(j in (i+1):s){
            x[i, j] <- rbinom(1, 1, p1)
            x[j, i] <- x[i, j]
            x[i+s, j+s] <- rbinom(1, 1, p2)
            x[j+s, i+s] <- x[i+s, j+s]
        }
    }
    for(i in 1:s){
        for(j in (s+1):n){
            x[i, j] <- rbinom(1, 1, q)
            x[j, i] <- x[i, j]
        }
    }
    for(i in (s+1):(2*s)){
```

```
    for(j in (2*s+1):n){
        x[i, j] <- rbinom(1, 1, q)
        x[j, i] <- x[i, j]
    }
}
for(i in (2*s+1):(n-1)){
    for(j in (i+1):n){
        x[i, j] <- rbinom(1, 1, q)
        x[j, i] <- x[i, j]
    }
}

graph <- graph.adjacency(x, mode="undirected")

d <- sort(degree(graph), decreasing = FALSE, index.return = TRUE)
#Sorted degree sequence

gaps <- d$x[2:n] - d$x[1:(n-1)] #Gaps between consecutive degrees

lg <- sort.int(gaps, decreasing = TRUE, method = "quick",...
      index.return = TRUE)$ix #Positions of gaps sorted by size

comms <- matrix(0,n,n) #Estimated partition for each possible K

for(k in 1:(n-1)){
    lgk <- sort(lg[1:k], decreasing = FALSE)
    lgk[k+1] <- n
    for(i in 1:k){
        for(j in (lgk[i]+1):lgk[i+1]){
            comms[k+1,d$ix[j]] <- i
        }
    }
}

QNGs <- vector(length = n)

for(l in 1:n){
    k <- max(comms[l,])+1 #Current estimated number of classes

    O <- matrix(0,k,k) #O_kl values
    D <- vector(length = k) #D_k values
    Q <- vector(length = k) #Terms of Q_NG

    for(i in 1:k){
        for(j in i:k){
            O[i,j] <- sum(x[which(comms[l,] == (i-1)),...
```

```
                        which(comms[l,] == (j-1))])
                O[j,i] <- O[i,j]
            }
        }
        for(i in 1:k){
            D[i] <- sum(O[i,])
        }
        L <- sum(D)
        for(i in 1:k){
            Q[i] <- O[i,i]/L - (D[i]/L)^2
        }
        QNGs[l] <- sum(Q) #Q_NG
    }
    comm <- comms[which.max(QNGs),] #Estimated partition
    Rand[r] <- compare(z, comm, method = "rand") #Rand index
    Khat[r] <- length(unique(comm)) #Estimated number of communities
}
mean(Rand) #Mean Rand index
sum(Rand>0.9999)/t #Fraction true partitions
mean(Khat) #Mean number of classes
sum(Khat==3)/t #Fraction true number of classes
```