Final Report

Random Numerical Simulation of Rare Events

Robustness of Spectral Clustering in Community Identification

Supervised by :
GOBET Emmanuel
GUYADER Arnaud

Authors :
AZNAG Abdellah
EL HAJOUJI Oualid

Abstract

The aim of this work is to study the algorithm of spectral clustering applied on graphs simulated with the Stochastic Block Model (SBM). We focus on the performance of the algorithm depending on the parameters of the SBM. We are particularly interested in computing probabilities of malfunction of the algorithm, on different sets of vertices.These probabilities are sometimes very low, malfunction being considered as a rare event. Naive methods such as Monte Carlo simulation (which is a simple implementation of the law of large numbers), are not efficient without a very high number of simulations, which is computationally costly. Therefore, we are using instead methods of importance sampling and splitting with Monte Carlo Markov chain, which are efficient methods for computing probabilities of rare events.

Contents

1 Theoretical frame

In this section, we introduce the theoretical background of the project. We begin (Section 1.1) by the Stochastic Block model that generates a specific random graph structure. It will be this structure that will serve as a study sample for community detection. Then (Section 1.2), we will describe the spectral clustering algorithm whose performance will be studied in the following sections.

1.1 The Stochastic Block model

A network is a set of elements interacting in a simple manner, either in a binary manner (two elements can be either connected or not connected), or according to an interaction function w (where $w_{i,j}$ represents the quality of the interaction of elements i and j).

A community of the network is a subset of elements such that the interactions between its elements is significantly higher than the interactions between its elements and the elements of its complementary. Note that this definition is only heuristic.

Generating a graph with k communities can be done backwardly, by creating a k-partition of the n elements network, and choosing the edges by linking with high probability two elements in the same class, and linking with low probability two elements in different classes. This is the fundamental idea behind the Stochastic Block model.

Definition 1. *The Stochastic Block Model*
Given a k-partition of n vertices (such that the class of vertex i is C_i), consider the random adjacency matrix A defined as followed:

$$\mathbb{A}_{i,j} \sim \mathbb{1}_{C_i=C_j} \times B(p_{in}) + \mathbb{1}_{C_i \neq C_j} \times B(p_{out})$$

Where $p_{in} \gg p_{out}$ are respectively the connection probabilities intra-communities and inter-communities. The resulting random graph (undirected and without loops) is said to follow the ***Stochastic Block Model*** *$G(n, p_{in}, p_{out})$.*

Figure 1 shows a simulation of a stochastic block model.

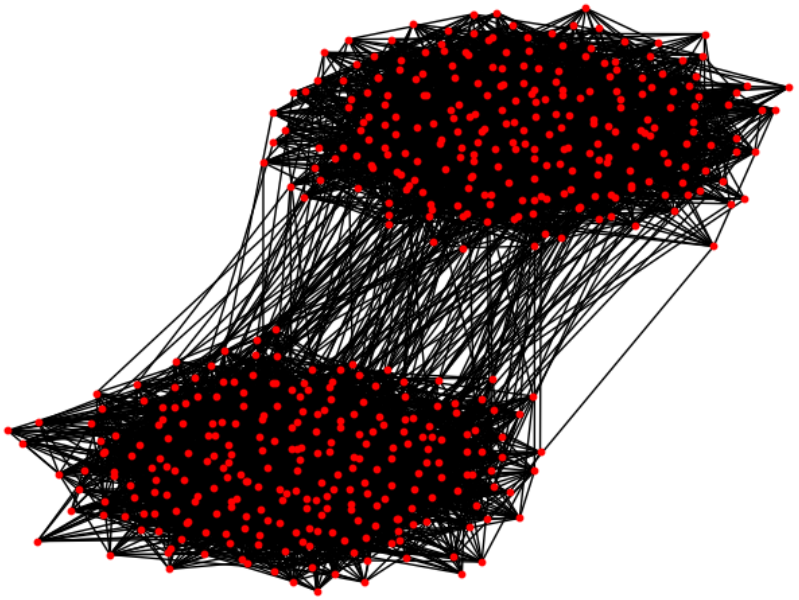

Figure 1: A network following the stochastic block model $G(200, 0.2, 0.04)$. The communities are distinguishable here.

1.2 Spectral Graph theory tools

Suppose now we are observing a graph following the stochastic block model. How can we reconstruct its communities? In other words, how can we find which vertices belong to which communities?
The Spectral Clustering Algorithm does this. Before describing the algorithm, let us introduce some objects from spectral graph theory:

Definition 2. *The degree and Laplacian matrices*

The degree matrix $\boldsymbol{D}$ *of a graph is the diagonal matrix where* D_{ii} *is the degree of vertex* i*,* $i \in [\![1, n]\!]$.
The unnormalized Laplacian matrix L *is defined as*

$$\boldsymbol{L := A - D}.$$

The normalized by division Laplacian matrix L_{norm} *is defined as*

$$\boldsymbol{L_{norm} := D^{-1}L}$$

The normalized by symmetrical division Laplacian matrix L_{snorm} *is defined as*

$$\boldsymbol{L_{snorm} := D^{-\frac{1}{2}}LD^{-\frac{1}{2}}}$$

These Laplacian matrices have important/common spectral properties giving us much insight about the graph itself, such as its connectivity (number of connected components, density of each component, ...). More precisely, it is the eigenvectors of such matrices that translate well how connected the data points are, and hence exhibits the different communities. In other words, performing a linear Laplacian transformation to the initial data set I outputs a new data set S that regroups communities in a compact manner (when well distinguishable) so that any unsupervised learning algorithm recognizing compactness (such as k-means) identifies them easily as clusters.
A more rigorous presentation and a proof of the statements above can be found in annex.

1.3 The algorithm (simplest form)

The discussions above allow us to derive the following algorithm, which is the spectral clustering algorithm in its simplest formulation:

Algorithm 1 Spectral clustering algorithm (general form)

Input **:** Adjacency matrix A
Step 1 : Compute L a chosen Laplacian matrix
Step 2 : Compute $v_1, v_2, ..., v_k$ the eigenvectors of L corresponding to the k greatest eigenvalues
Step 3 : Perform to the n columns of the matrix $S = (v_1|v_2|...|v_k)^T$ the k-means clustering algorithm Get the Output of Step 3 in $X_1, X_2, ..., X_k$ where X_i contains the indices of points in cluster i
Output **:** $C_1, C_2, ..., C_k$ such that $C_i = [\![y_j | j \in X_i]\!]$ where y is the initial data points vertex

As we will see, the performance (which will be defined and discussed in the following section), the time cost and the memory cost of the algorithm, depend on several parameters. In our stochastic block model, the theoretical dependencies (NOT implementation dependencies) we chose to examine are the following:

1.3.1 The parameters of the stochastic block model

The probabilities generating the adjacency matrix are an immediate factor. Intuitively, if for a given graph, p_{in} and p_{out} are close, then the resulting model will have its edges distributed almost independently from the classes choice, leaving no information about classes distribution to the Laplacian matrix. Conversely, if $p_{in} \gg p_{out}$, then there will be a high density of edges within a class, and almost zero density between two classes, making it easier for the Laplacian matrix to distinguish them.
Figures 2 and 3 illustrate this for $k = 3$:

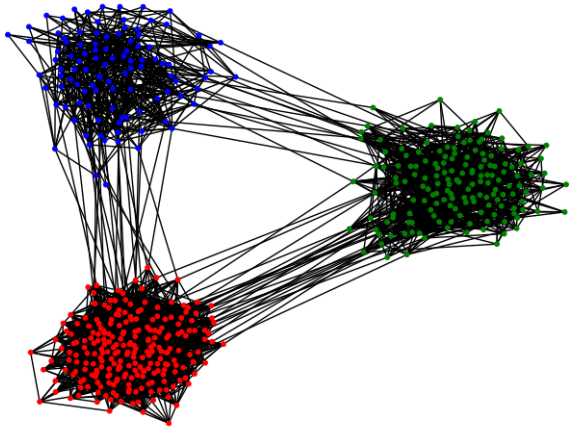

Figure 2: Average performance of Algorithm 1 on $G(500, 0.2, 0.02)$, with distribution $[0.2, 0.3, 0.5]$

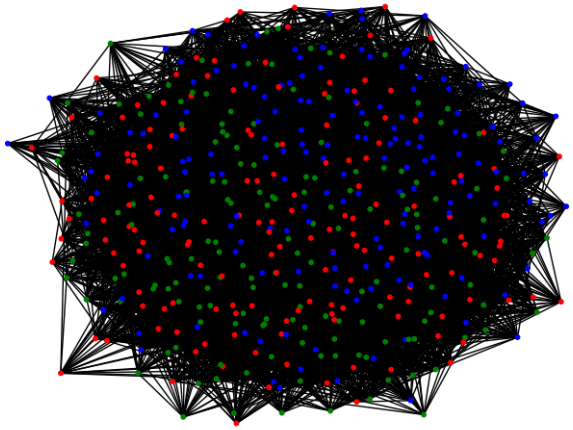

Figure 3: Average performance of Algorithm 1 on $G(200, 0.2, 0.18)$ with the same distribution $[0.2, 0.3, 0.5]$

1.3.2 The choice of the Laplacian matrix

Different Laplacian matrices transform differently the initial data sets, and one can expect different final clusters after performing the algorithm. It is clear that the algorithm will perform well regardless of the chosen Laplacian matrix when hypothesis around p_{in} and p_{out} are extreme (cf. 1.3.1). That is because the community blocks are easily distinguishable in all possible options after block-diagonalizing the Laplacian.
The performance differences are thus relevant when the graph parameters evolve around a "frontier" we have yet to define.

1.3.3 The distribution of the classes

A distribution of $[0.5, 0.5]$ -two co-dominant communities- is not expected to be clustered as hardly as $[0.001, 0.999]$ -two communities with one as a rare minority-. The reason why the latter is harder to solve is that the set of edges connected to the minority group is not enough to exhibit community traits.
One can point out that this last parameter involves directly the first two. The first one is because the graphs we're clustering in anomaly cases are much alike. The second one is because the Laplacian transformation magnifies the data according to the vertices degrees (through multiplication by D). Minorities tend to have a much smaller degree than majorities, making it hard for them to regroup after a Laplacian transformation. So adapting the algorithm can help distinguish such minorities.

1.4 Implementation choices

1.4.1 Python vs. C++

The execution when using NumPy objects is satisfying in most heavy simulations encountered in this work (a few minutes when simulating 50000 different graphs in the bad clustering vertices tests in the following sections). After discussing it with our advisors, we did not see any major interest in using a more powerful language for the moment -C++ for example-.

1.4.2 Code architecture

We tried as much as possible to follow a design pattern separating the model implementing from the tests and the graphical designing/representations. The functions are also as modular as possible, allowing for more flexibility when performing tests, and for the codes to be re-used for further works.

1.4.3 $k = 2$

We restricted our studies to the case of 2 communities. The aspects of the algorithm we're studying can be indeed thoroughly explored in the case of $k = 2$. Another reason is more theoretical, as the transition from the case $k = 2$ to $k \neq 2$ is only a generalization of already introduced tools, and does not change the study in a fundamental way.

1.4.4 Measuring the performance

One way to determine whether an algorithm performed well on a certain graph is to count the distances between the clusters and the classes. One first proposition might be:

$$Q = \frac{1}{n}\sum_{k=1}^{n} 1_{L(k)=C(k)}$$

where $L(k)$, $C(k)$ are respectively the class and the cluster of vertex k. The initial labeling can however be completely different from the algorithm's labeling, since the choice of these two is done independently (see *qlt.reEvaluate* in the code). The definition above does not take this into account, as it returns for example a performance of $Q = 0$ when clustering the list $[1, 0, 0, 0]$ into $[0, 1, 1, 1]$, while the community detection was done successfully. A more suitable definition would be

$$Q = \frac{1}{n}\max(\sum_{k=1}^{n} 1_{L(k)=C(k)}, \sum_{k=1}^{n} 1_{L(k)\neq C(k)})$$

This quantity will be called **the quality of the clustering** and is between 0.5 and 1. A quality of $Q \simeq 1$ means classes and clusters are similar, hence good clustering, while a quality of $Q \simeq 0.5$ means that there is no similarity in the way classes and clusters are distributed, hence bad clustering.
However, this definition is still unsatisfying is some extreme cases, for example, in the case of minority vs majority. Assume the distribution is $[a_1, a_2]$, with $a_1 \gg a_2$. Then an output where all vertices of class 2 are badly clustered will still give a quality near 1. Since we are not concerned with the study of minorities, we decided to keep the decision above.

2 Algorithm performance

2.1 Performance based on the graph ratio

2.1.1 The graph ratio

It is clear that as p_{out} comes closer to p_{in}, the quality of clustering Q decreases, as elements in different communities interact more and more frequently, making it harder to differentiate their interactions from intra-communities interactions. A question of interest is how the quality Q evolves as p_{out} gets bigger?
In the literature (1), it is proven that when:

$$(1) \qquad r := \frac{c_{in} - c_{out}}{\sqrt{\log n \times (c_{in} + c_{out})}} \gg 1$$

(where $c_{in} = np_{in}$ and $c_{out} = np_{out}$) the algorithm works extremely well with asymptotic error in $o(n)$. We chose to examine more precisely the behavior of this ratio, which we will call from

now on **the graph ratio**.
The simulation procedure is the following: Fixing n and c_{in}, and simulating for each ratio a reasonable number of graphs. For precision purposes, the distribution of the ratio values is denser in the zones where the curve changes drastically. Altogether, this study will be repeated for several distributions of classes, as the performance comparisons differ when distributions vary.
For each time, a precision step of $\frac{1}{80}$ is used and for each of the 80 graph ratio values 100 simulations were performed to compute the quality average.

2.1.2 Case of co-dominance

The first case deals with two classes of similar size.

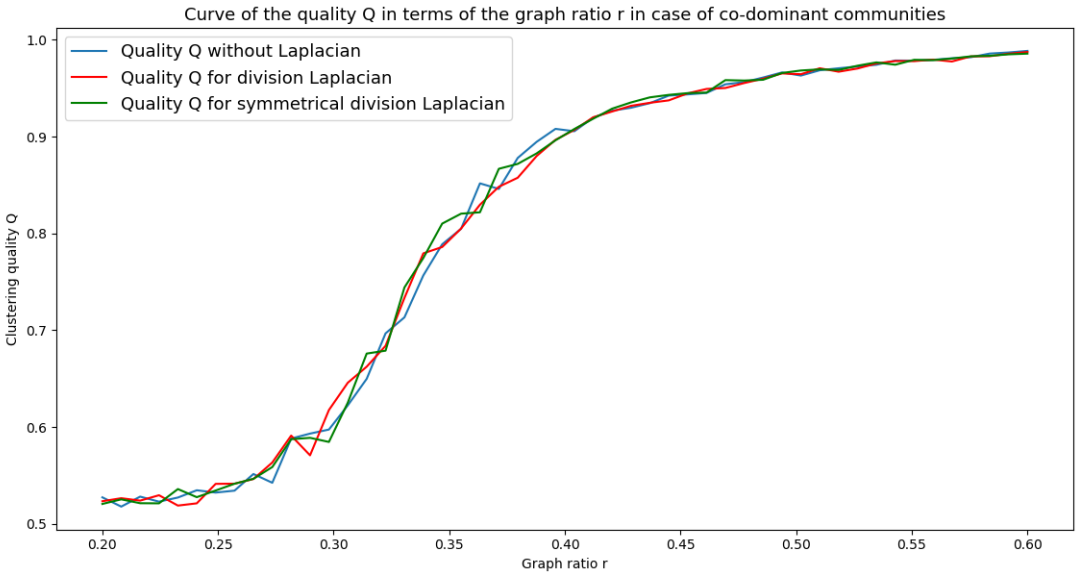

Figure 4: Clustering quality in the case of co-dominant communities: The performance remains the same when applying different Laplacian matrices

Notice that these curves (in figure 4) are increasing, with asymptotic values of $\frac{1}{2}$ (the clustering is equivalent to a random classifier) and 1 (perfect clustering).The remarkable thing is that the curves are close to one another, that is, the gain from multiplying with the degree matrix D (either to the left or symmetrically) is null in the case of co-dominance.
A non-rigorous explanation of this phenomenon is the following: the effect of multiplying the matrix to the left (resp. to the right) by a diagonal matrix is to magnify each row (resp. each column) i with its i-th eigen-value. When this is applied to the multiplying Laplacian, each row

i is multiplied by the degree of the vertex i, which is on average $p_{in} \times |C_1| + p_{out} \times |C_2|$, with C_1 and C_2 the two communities. So in the case of co-dominance, $|C_1| = |C_2|$, which means all rows are multiplied by the same factor, which have no effect on the spectrum nor the eigenspaces. The case where of the symmetrical multiplying Laplacian is globally the same.

2.1.3 The case $|C_1| \neq |C_2|$

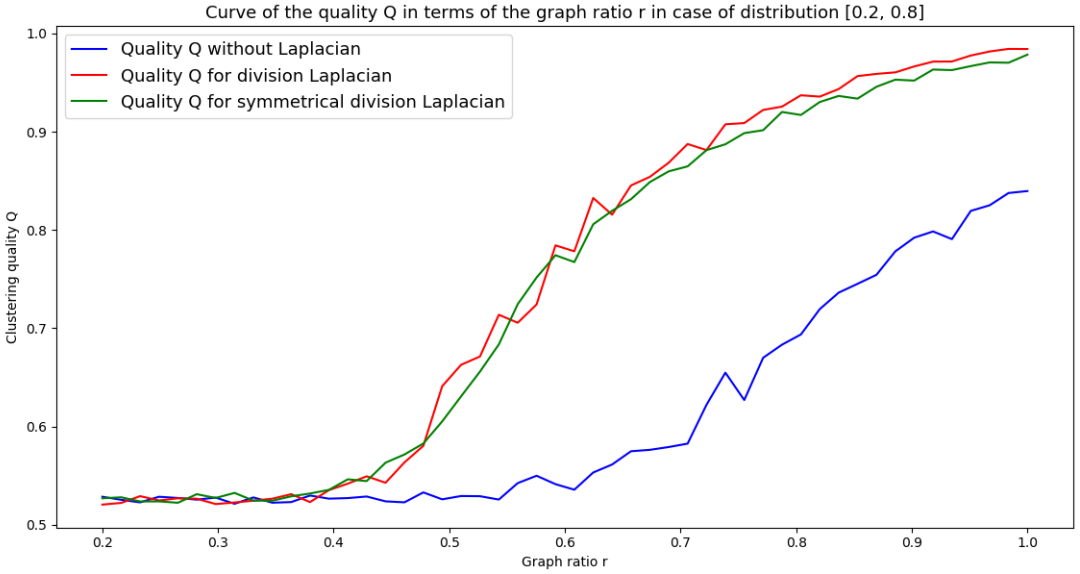

Figure 5: Clustering quality in the case of a disequilibrium: Using a non-trivial Laplacian increases the quality.

The power of Laplacian multiplication is seen when communities have different sizes: while multiplying symmetrically (or to the left) seems globally similar, the two are outperforming the basic clustering when the Laplacian is taken as the adjacency matrix.
Like in the previous subsection, we propose a non-rigorous statement to explain the phenomenon: In the case of the multiplying Laplacian, each row i is magnified on average by the average degree of i, which is the same for vertices in the same community and different for two vertices in different communities. This magnification tends to regroup highly linked vertices, that is, vertices which are more probable to be from the same community. More elaborate discussion regarding the effect on the spectrum and the eigenvectors will be in the following section.
Also, as we can observe, the first order differences between normal Laplacian division and symmetrical division do not appear to concern performance itself. Perhaps the differences lie in the spectral properties of each transformation and hence in the execution speed of the algorithm (See the next section on the study of the spectral properties)

2.1.4 Synthesis of behaviors and general remarks

After running these simulations, we can make the following conclusions:

- The asymptotic behaviors of the curve match the predictions: excellent clustering when $r \gg 1$ -and bad clustering when $r \ll 1$- . Surprisingly, it appears that the condition $r \gg 1$ is sufficient but not necessary, as it seems that $r \geq 1$ is enough. Moreover, the asymptotic validity of the algorithm discussed in the previous subsection is noted as well.
- In the general case where the communities are of different sizes, using a non-trivial Laplacian improves -significantly- the performance in the zone where the graph ratio is not big.

We noticed that the general behavior of the quality curve Q(r) is sigmoidal, regardless of the parameters of the simulation. The inflection point $r_{\frac{3}{4}}$ varies along with the size of the graph, the Laplacian used and the distribution. For a fixed Laplacian method and graph parameters, $r_{\frac{3}{4}}$ increases (decreasing the overall quality) when the distribution gets more unbalanced then rapidly decreases when it gets too unbalanced. The first movement is because the algorithm performs better when communities have the same size, while the second movement is due to the limits of our definition of Q, as it does not translate well bad clustering when a community is a minority (see the last section).
We did not find in the literature a proof -not even a mention- of this sigmoidal behavior. At the moment, we are trying to implement a suitable regression to confirm -or infirm- this behavior.

Conjecture 1. *Consider fixed parameters n, c_{in} and a Laplacian choice for Algorithm 1. Then $Q(r)$ is a sigmoid.*

2.2 Study of the eigen values

The expected adjacency matrix E is similar (after rearranging the order of the vertices in the matrix) to the following matrix:

$$\left(\begin{array}{c|c} p_{in}H_{|C_1|} & p_{out}H_{|C_1|,|C_2|} \\ \hline p_{out}H_{|C_2|,|C_1|} & p_{in}H_{|C_2|} \end{array}\right)$$

where $H_{i,j} = \begin{pmatrix} 1 & ... & 1 \\ .. & ... & .. \\ 1 & ... & 1 \end{pmatrix} \in \mathcal{M}_{i,j}$.
Its norm, which is also its largest eigenvalue, is also its expected degree:

$$\lambda_1 = d = \frac{p_{in}|C_1| + p_{out}|C_2|}{n}, v_1 = \begin{pmatrix} 1 & .. & 1 \end{pmatrix}^T$$

while

$$\lambda_2 = \frac{p_{in}|C_1| - p_{out}|C_2|}{n}, v_2 = \begin{pmatrix} 1 & .. & 1 & | & -1 & .. & -1 \end{pmatrix}^T$$

gives info on the community structure of the graph. Thus the knowledge of E gives perfect recovery of the communities. One natural question that arises is: how close is A from E?
The answer comes by using Bernstein's inequality on A viewed as a sum of independent random matrices. If we assume that $d \gg \log n$ we obtain:

$$||A - E|| \leq ||A||$$

with linear constraints on $\sqrt{d}, \sqrt{\log n}$. Approximation theory gives: $v_2(A) \simeq v_2(E)$, with the quality of the approximation (and hence the clustering) increasing as the sparsity of the network increases and the community structure (visible in the graph ratio as well) becomes clearer.
The purpose of these computations is to give a first insight on why the clustering algorithm works (for the Laplacian function taken as the adjacency matrix itself). For more sophisticated Laplacian choices, similar approximations exist but through different inequalities. From the

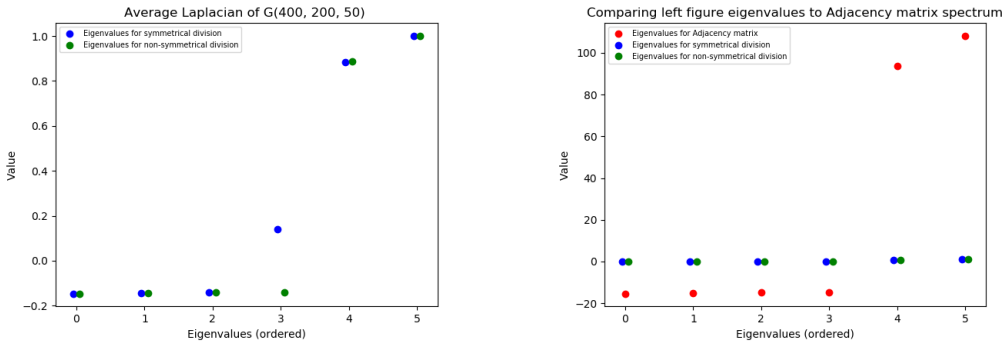

Figure 6: Left figure shows the distribution of eigenvalues for the Laplacian matrices and compare their scale to the spectrum of the Adjacency matrix

figure above, we notice mainly three things:

- The values are not exactly positive: this is due to algorithmic computation uncertainties (They are close to 0 in figure 6)
- The distribution of eigenvalues is globally the same for the two Laplacian matrices, except for the Kernel dimension: On average, normalising symmetrically reduces the Kernel dimension, or equivalently, increases the size of the other eigenspaces. These eigenspaces are usually strong indicators of the community structure, and thus increasing their dimensions means increasing the performance.
  In other words, symmetrical normalization gives better performance, even though it is only a second order improving.
- The scale of eigenvalues decreases considerably after a Laplacian performance.

3 Theoretical limits of the algorithm and insight on the sigmoid conjecture

In this section, we restate the community recovery problem differently. This version uses elements of information theory, as opposed to the former one, which uses elements of algebraic graph theory.

3.1 The ITCRP problem and the phase transition theorem

Definition 3. *Information Theory Community recovery problem (ITCRP)*
Let n be a positive integer and $x \in \{0,1\}^n$. Consider the input vertex $y \in \{0,1\}^N$ where $N = \binom{n}{2}$, where $y_{i,j}$ is the output of $x_i \oplus x_j$ through the channel $\begin{pmatrix} 1-p_{in} & p_{in} \\ 1-p_{out} & p_{out} \end{pmatrix}$ for $1 \leq i < j \leq n$.
The Information Theory Community recovery problem is to recover correctly x from the observation of y with high quality.

Proposition 1. *The ITCRP and the Stochastic Block Model community recovery are equivalent.*

The proof is immediate. The reason we restated the problem is because we can now use results from information theory which will give us deeper insight regarding theoretical possibilities given certain parameters. As a matter of fact, the ITCRP is a special case of a larger set of problems in information theory, called Low Density Generator Matrix (LDGM) decoding problems. The main result for such a set of problems is the existence of performance phases which depend only on the problem parameters (and are in particular independent of the recovery algorithm).
Precisely, the following theorem holds:

Theorem 1. *Consider a LDGM decoding problem and $M(a_1, ..., a_k)$ its Matrix. Then there exists a function $f(a_1, ..., a_k)$ and reals $\alpha > \beta$ such that:*

$$f > \alpha \iff \textbf{\textit{The problem has a feasible solution (algorithm)}}$$

and

$$f < \beta \iff \textbf{\textit{The problem can't be solved}}$$

In particular, for the ITCRP problem:

$$f(n, c_{in}, c_{out}) = \frac{\frac{c_{in}+c_{out}}{2} - \sqrt{c_{in}c_{out}}}{\log n}$$

and

$$\alpha = \beta = 1$$

Notice that

$$f \underset{r \to +\infty}{\longrightarrow} +\infty$$

$$f \underset{r \to -\infty}{\longrightarrow} -\infty$$

With r the graph ratio. Hence using the equivalence in proposition 1, Theorem 1 implies that:

$$Q \underset{r \to +\infty}{\longrightarrow} 1$$

$$Q \underset{r \to -\infty}{\longrightarrow} \frac{1}{2}$$

And most importantly: Q changes abruptly from $\frac{1}{2}$ to 1 around a critical value r_c, corresponding to the regime switch (or phase transition) in Theorem 1. We recover theoretically most of the behavior of $Q(r)$.
Conclusion: even if conjecture 1 is proven to be false (which is not the case yet), approximating Q with a sigmoid function turns out to be a good second order approximation.

4 Clustering badly a set of vertices

4.1 The problem

Consider a subset V of vertices, say $[\![1,\ m]\!]$. What is the probability that a part of these vertices (resp. all these vertices) is badly clustered? The answer is *very, very small*. In fact, such an event is considered a rare event, and as we will see, a naive Monte Carlo simulation will not be enough to estimate its probability. Moreover, the approach we will use varies depending on the size of m ($m \ll n$ and m reasonably big).

4.2 Importance sampling: theoretical frame

Importance sampling is a key concept when it comes to evaluating probabilities of rare events. It relies on a simple principle: reducing the problem to calculating, via Monte-Carlo, an expectation related to a non-rare event. Here is the main theorem we will be using, related to importance sampling for Bernoulli random variables.

Theorem 2. *Consider the random independent variables $X_1, \cdots, X_p$ with Bernoulli distributions $B(p_1), \ldots, B(p_p)$; and $Z_1, \ldots, Z_p$ independent variables with Bernoulli distributions $B(q_1)$, ..., $B(q_p)$. Then, for all $g : \{0,1\}^p \to \mathbb{R}$ bounded,*

$$\mathbb{E}\left[g(X_1, \ldots, X_p)\right] = \left(\prod_{i=1}^{p} \frac{1-p_i}{1-q_i}\right) \mathbb{E}\left[g(Z_1, \ldots, Z_p) \prod_{i=1}^{p} \left(\frac{p_i(1-q_i)}{q_i(1-p_i)}\right)^{Z_i}\right]$$

$(X_{i,j})_{1\leq i<j\leq n}$, the random coefficients of the adjacency matrix of the simulated graph, follow the Stochastic Block Model, and therefore have Bernoulli distributions $(B(p_{i,j}))_{i<j}$. Let's consider another independent adjacency matrix, of coefficients $(Y_{i,j})_{i<j}$ with distributions $(B(q_{i,j}))_{i<j}$.

Let E(x) be the event : "When the clustering algorithm is applied to the adjacency matrix $(x_{i,j})$, the set of vertices V is *well-clustered*". Note that *well-clustered* means that every vertex is in the right community. Let g be the bounded function defined as :

$$g : x = (x_{i,j})_{i<j} \to \mathbb{1}_{E(x)}.$$

Theorem 1 becomes, in this case, with p the probability of badly clustering the set V.

$$p = \left(\prod_{i<j} \frac{1-p_{i,j}}{1-q_{i,j}}\right) \mathbb{E}\left[\mathbb{1}_{E(Y)} \prod_{i<j}\left(\frac{p_{i,j}(1-q_{i,j})}{q_{i,j}(1-p_{i,j})}\right)^{Y_{i,j}}\right]$$

The goal is therefore to calculate the probability p by evaluating, through Monte-Carlo, the expectation. This is possible only when the distributions of the coefficients $(Y_{i,j})_{i<j}$ are chosen so that E(Y) is no longer a rare event, which brings us to the following part.

4.2.1 A first importance sampling model : probability inversion

E(Y) is no longer a rare event when the elements of V have a reasonable chance of getting linked to vertices of a different class. The probabilities of extra-class and intra-class links were therefore inverted as the latter is initially higher. That is to say:

$$For\ all\ i \in [\![1,\ m]\!], q_{i,j} = \begin{Bmatrix} \mathrm{p}_{out}\ if\ i \sim j \\ \mathrm{p}_{in} \qquad else \end{Bmatrix}$$

The factor outside the expectation becomes, when replacing with the correct values of p and q, where Cl(i) is the class of i :

$$\left(\frac{1-p_{in}}{1-p_{out}}\right)^{\sum_{i=1}^{m}(2\#Cl(i)\cap[\![i+1,\ m]\!]-n+i)}$$

The factor inside is, on the other hand ($i \sim j$ means i and j have the same class):

$$\prod_{i=1}^{m}\left(\frac{p_{in}(1-p_{out})}{p_{out}(1-p_{in})}\right)^{2\sum_{\substack{i<j\\ i\sim j}} Y_{i,j} - \sum_{i<j} Y_{i,j}}$$

This gives us the whole formula, that we will use for our probability calculation:

$$p = \left(\frac{1-p_{in}}{1-p_{out}}\right)^{\sum_{i=1}^{m}(2\#Cl(i)\cap[\![i+1,\ m]\!]-n+i)} \times \mathbb{E}\left[\mathbb{1}_{E(Y)} \prod_{i=1}^{m}\left(\frac{p_{in}(1-p_{out})}{p_{out}(1-p_{in})}\right)^{2\sum_{\substack{i<j\\ i\sim j}} Y_{i,j} - \sum_{i<j} Y_{i,j}}\right]$$

4.3 Computing probabilities using importance sampling

4.3.1 Case where $m \ll n$

All our simulations are done with $n = 100$, in the case of co-dominance and with m = 1 (for the time being). We are aiming to compute, with different graph parameters, the probability of vertex 1 being badly clustered.

First let's take $c_{in} = 90$ and $c_{out} = 72$, which gives a graph ratio r=0.66. This specific graph ratio makes our event a non-rare one. In fact, we computed a probability of 0.012 with a classical Monte-Carlo approach, with 10000 simulations. With the importance sampling approach, we obtain a probability of 0.01175 (with 10000 simulations). Even though we are interested in rare events, this calculation was necessary in order to check the accuracy of the importance sampling approach. In the figure below, and in all other figures, the red and green lines are the bounds of the 95 % confidence interval. This interval was obtained thanks to the central limit theorem: standard deviation was estimated with the common numpy std estimator.

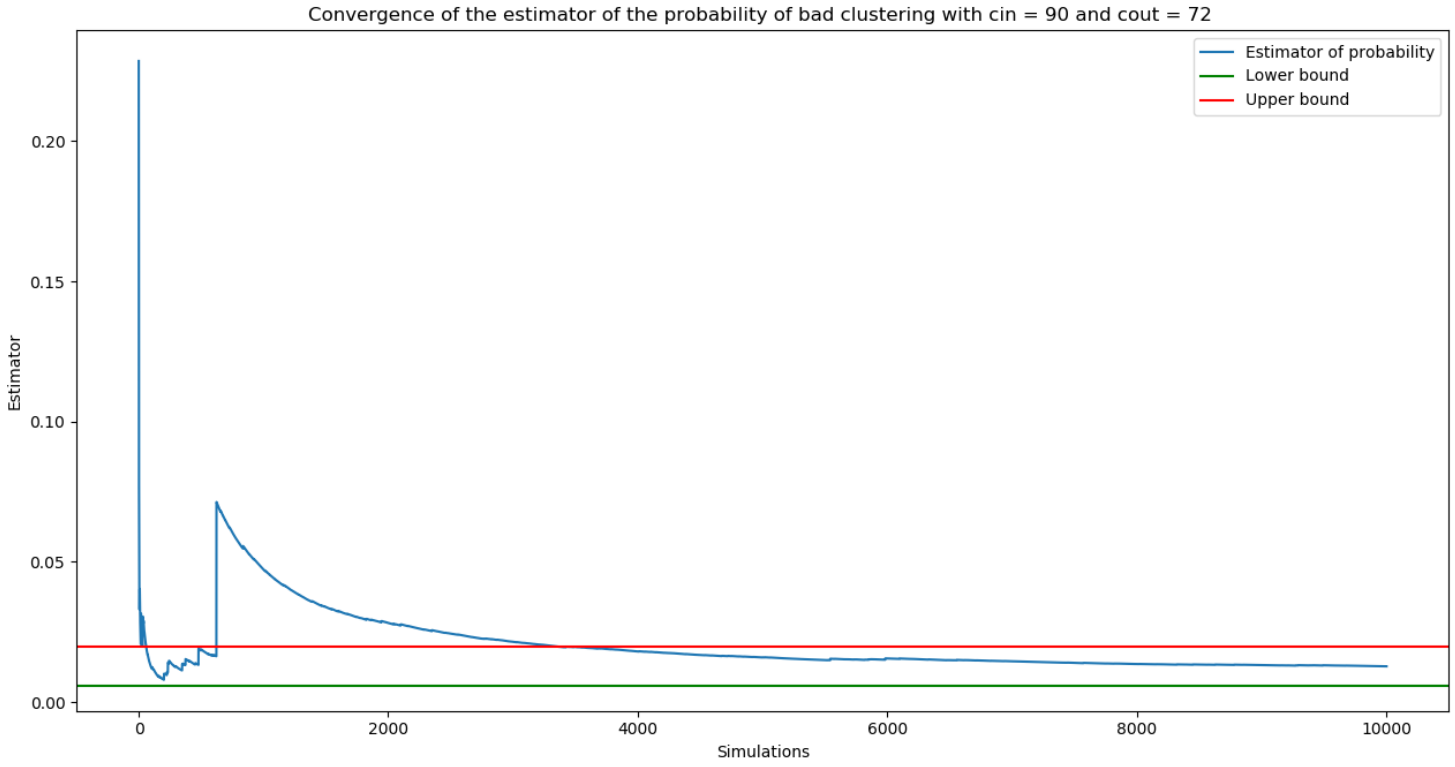

Figure 7: Value of the estimator depending on the number of simulations ($c_{in} = 90$, $c_{out} = 72$).

With $c_{in} = 90$ and $c_{out} = 50$ (r=1.56), the importance sampling algorithm returns a probability of 2.23×10^{-13}, which makes the event a rare one. It is coherent with the quality of clustering found in section 2. The Monte Carlo approach obviously doesn't work.

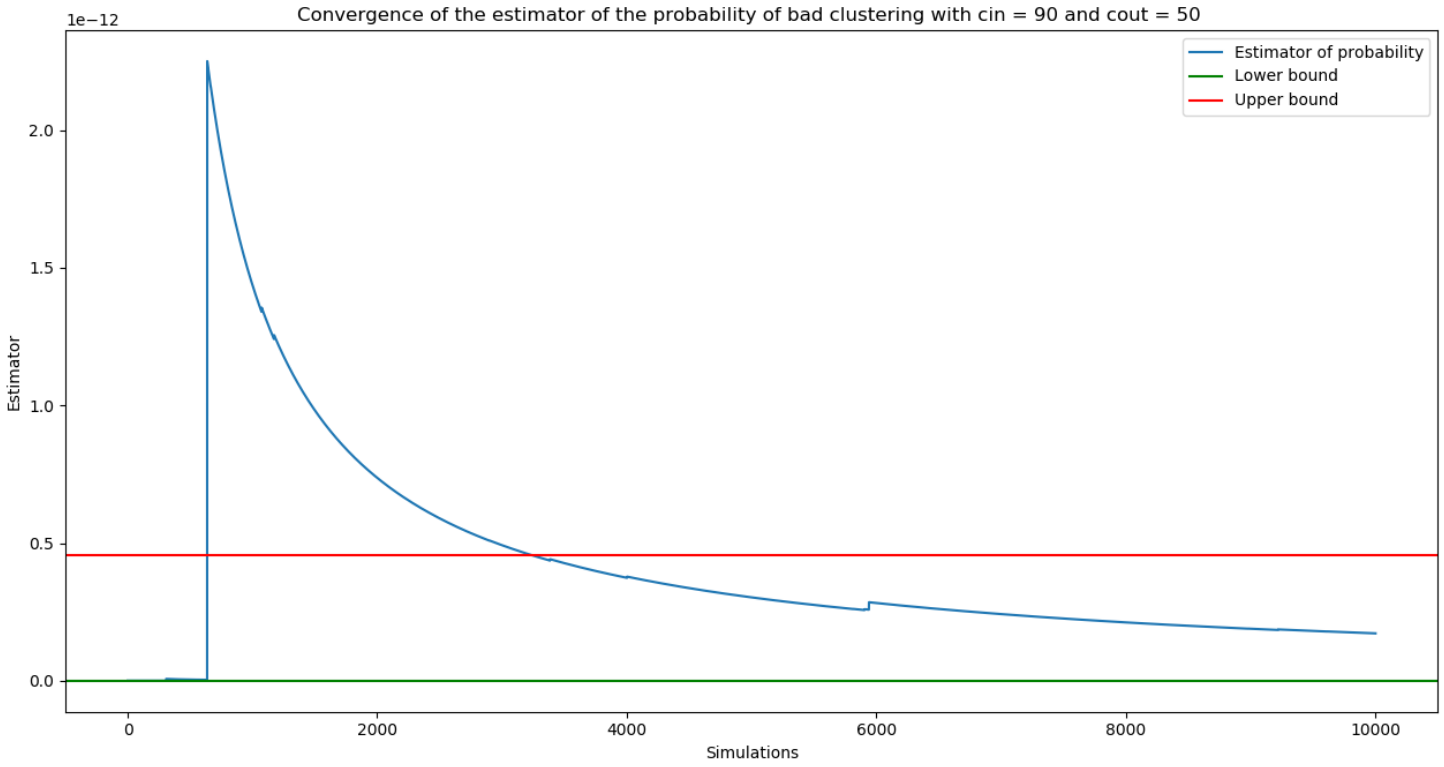

Figure 8: Value of the estimator depending on the number of simulations ($c_{in} = 90$, $c_{out} = 50$).

Finally, with $c_{in} = 90$ and $c_{out} = 20$ (r=3.11), the probability computed is 6.8×10^{-71}, which is impressively low. It is not surprising that the convergence of the estimator is quite slow (it cannot really be seen with 50000 simulations). The result obtained is hower satisfying, as the confidence interval is not as wide as we could have expected for such a rare event, especially with a non-negative lower bound. It only seems wide because of the vertical scale of the graph.

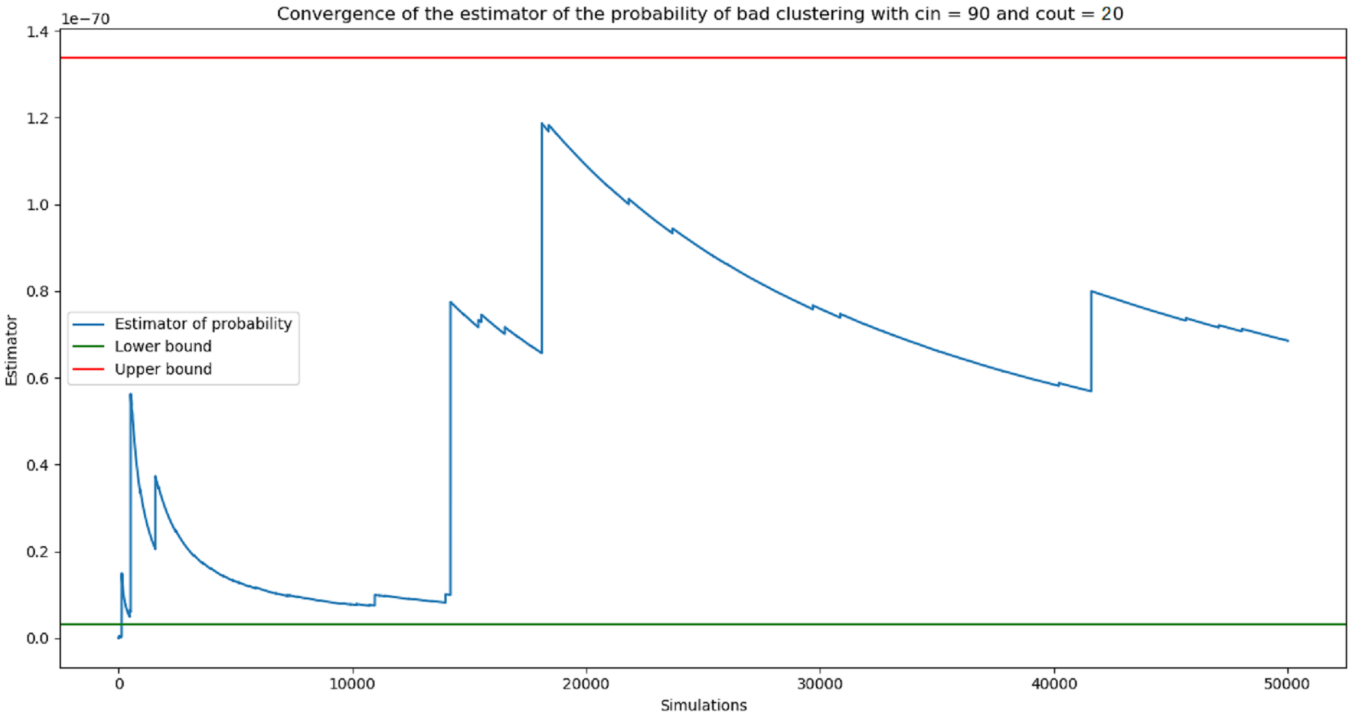

Figure 9: Value of the estimator depending on the number of simulations ($c_{in} = 90$, $c_{out} = 20$).

4.3.2 Case where m is gets bigger

Here, we are still working with n = 100, and same cardinal classes. As for m, we chose a value of 25. We compute the probability of badly clustering a set of 25 vertices using importance sampling with probability inversion, . We systematically choose parameters that make the graph ratio inferior to one. The considered event would otherwise be not only rare, but almost impossible.
With $c_{in} = 90$ and $c_{out} = 72$ (r=0.66), a probability of 0 was computed: this event, in these conditions, is too rare for the algorithm.
With $c_{in} = 40$ and $c_{out} = 38$ (r=0.11), a probability of 0 was computed also.

It seems that this model of importance sampling won't work for big values of m. That is because of the factor $\mathbb{1}_{E(Y)}$, which seems to always be equal to 0. Indeed, even though we invert the probabilities of extra-class and intra-class links, the event $E(Y)$ is still rare: it is difficult to cluster badly all the 25 vertices considered.

4.3.3 General importance sampling model

Instead of inverting the probabilities of extra-class links and intra-class links, we could choose any parameters q_{in} and q_{out} for the new probabilities. We would have the more general definition:

$$For\ all\ i \in [\![1,\ m]\!], q_{i,j} = \begin{Bmatrix} \mathrm{q}_{in}\ if\ i \sim j \\ \mathrm{q}_{out} \qquad else \end{Bmatrix}$$

A first approach would be to consider that the model can be tuned by finding values of q_{in} and q_{out} that minimize the standard deviation. The following plot shows standard deviation depending on the value of q_{in}, with n=100, $c_{in} = 90$, $c_{out} = 72$, $q_{in} = 0.9$ and 5000 simulation per computation of standard deviation.

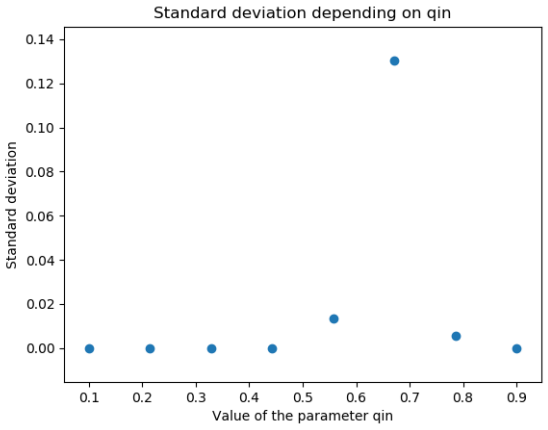

Figure 10: Standard deviation depending on q_{in} parameter of the importance sampling model.

Interestingly, the maximum value of standard deviation is at $q_{in} = 0.7$, which corresponds to $q_{in} = p_{out}$, that is to say to the probability inversion model (given that $q_{out} = p_{in} = 0.9$). It seems that, for other values of q_{in}, the standard deviation is very low, and that is because the likelihood ratio is abnormally low. We have proven that the probability inversion algorithm gives a good estimation of the probability, so that would be astonishing that this would be the worst possible choice of parameters for importance sampling.

Therefore, the criteria of minimizing standard deviation cannot be trusted here.

4.4 Splitting and MCMC for large sets of vertices

The probability inversion importance sampling model is not suitable for large values of m. It may be possible to use the general importance sampling model with correct values of the parameters

q_{in} and q_{out}, but this approach would demand a non-trivial tuning phase each time we want to compute a probability. We will be using instead a splitting method that relies on MCMC simulation (Markov chain Monte Carlo).

4.4.1 Splitting

For a given graph $\mathbf{A}$, we define $\Phi(\mathbf{A})$ as the total number of vertices badly clustered among the subset $[\![1,\ m]\!]$. Our goal is to compute the probability of $\{\Phi(\mathbf{A}) \geq m\}$. The splitting method consists in:
- Choosing K thresholds: $m = m_K > \cdots > m_2 > m_1 > m_0 = 0$
- For all $k = 1, \cdots, K$, approach $P\left(\Phi(\mathbf{A}) \geq m_k | \Phi(\mathbf{A}) \geq m_{k-1}\right)$ with an estimator $\hat{\pi}_k$.
- Estimate $P(\Phi(\mathbf{A}) \geq m)$ with $\prod_{k=1}^{K} \hat{\pi}_k$.

4.4.2 MCMC for conditional probabilities

We use Markov chain Monte Carlo for computing the conditional probability $P\left(\Phi(\mathbf{A}) \geq m_k | \Phi(\mathbf{A}) \geq m_{k-1}\right)$. We consider $p \in]0,1[$, $m_{k-1} \in R$, $M \in N$. We create a Markov chain of graphs $\mathbf{A}^\ell$ with the following algorithm:

- We start with $\mathbf{A}^0$, a simulated graph with $\Phi(\mathbf{A}^0) \geq a_{k-1}$
- For $l \in [\![0,\ M-1]\!]$, given $\mathbf{A}^\ell$, we simulate again a randomly chosen proportion p of the edges of $\mathbf{A}^\ell$. The remaining edges are not modified. If , for the resulting graph $\widetilde{\mathbf{A}}^{\ell+1}$,$\Phi(\widetilde{\mathbf{A}}^{\ell+1}) \geq m_{k-1}$, then we define $\mathbf{A}^{\ell+1} = \widetilde{\mathbf{A}}^{\ell+1}$. Else, we define $\mathbf{A}^{\ell+1} = \mathbf{A}^\ell$.

The ergodic theorem allows us to define an estimator $\hat{\pi}_k$ of $P\left(\Phi(\mathbf{A}) \geq m_k | \Phi(\mathbf{A}) \geq m_{k-1}\right)$ insofar as:

$$\frac{1}{M} \sum_{\ell=1}^{M} \mathbf{1}_{\Phi(\mathbf{A}^\ell) \geq m_k} \quad \underset{M \to \infty}{\overset{\text{a.s.}}{\longrightarrow}} \quad P\left(\Phi(\mathbf{A}) \geq m_k \mid \Phi(\mathbf{A}) \geq m_{k-1}\right).$$

4.4.3 Parameter tuning

We have to choose the thresholds $m = m_K > \cdots > m_2 > m_1 > m_0 = 0$, and the parameter p, proportion of modified edges. The dilemma in the choice of the thresholds is that, when we choose too few of them, the conditional probabilities are too small, and when we choose too many of them, the computational time cannot be handled.

Let's take n=100, m=25 and M=10000. We choose an iterative dichotomy approach to identify the correct thresholds. We start with a large split of the subset $[\![1,\ m]\!]$, for instance (0, 10, 20, 25). If one of the probabilities computed is < 0.001, we split the corresponding subset into two subsets. Let's say that we find $P\left(\Phi(\mathbf{A}) \geq 20 | \Phi(\mathbf{A}) \geq 10\right) < 0.001$, then the next

thresholds we will be trying are (0, 10, 15, 20, 25). On the other hand, if two probabilities when multiplied, give a result ≥ 0.1, we "merge" the thresholds to reduce computational time without affecting the computation itself. In the case $c_{in} = 90$, $c_{out} = 85$, we ended up with the following thresholds: (0, 15, 20, 21, 22, 23, 24, 25)

We choose $p = 0.6$, knowing that a too high or too low value of p would not give satisfying results.

4.4.4 Computing probabilities with MCMC splitting

First, let's define n=20, $m = 5$, $M = 5000$. With $c_{in} = 14$ and $c_{out} = 12$ ($r = 0.23$), using the splitting algorithm with the thresholds (0, 2, 4, 5), we compute a probability of misclustering the 5 vertices equal to 0.0055. Similarly to what has been done in the importance sampling part, we can check the accuracy of the model by computing this probability using a simple Monte Carlo approach (10000 simulations), as the event considered is not rare. With Monte Carlo, a probability of 0.0051 was computed, which confirms the accuracy of the model.

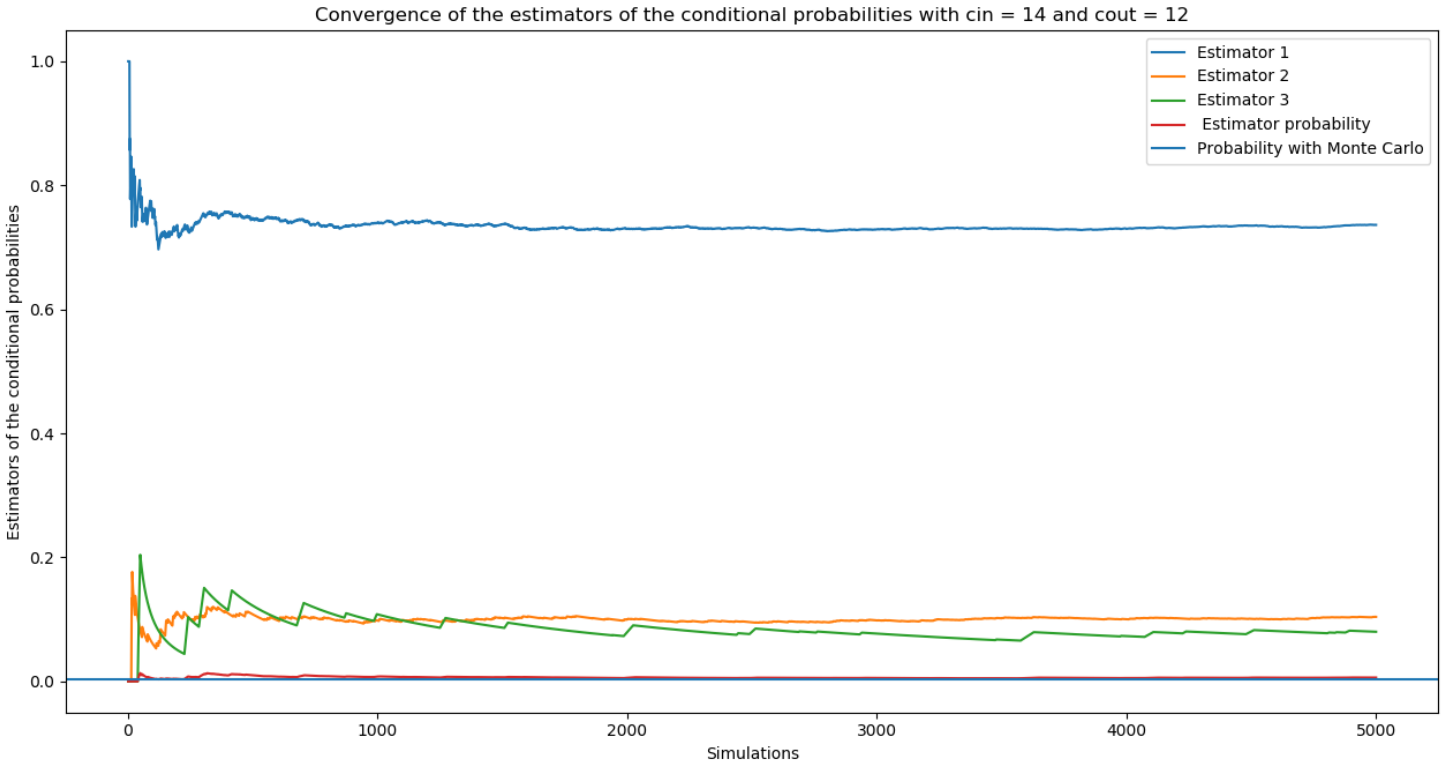

Figure 11: Value of the estimators depending on the number of simulations (n=20, m=5).

Let us now come back to the standard parameters n=100, $m = 25$, $M = 10000$.

With $c_{in} = 90$ and $c_{out} = 85$ (r=0.18), and thresholds (0, 15, 20, 21, 22, 23, 24, 25), we compute a probability of 2.01×10^{-7}. Most estimators (of the conditional probabilities) are consistent for this value of M, except estimator 6 and 7 which would probably give a better estimation with a greater value of M.

It is not obvious to find a confidence interval because the estimator we are considering is a product of empirical means, instead of being an empirical mean itself. Moreover, it is very difficult to compute the variance of the estimator because each simulation of it, with M=10000, takes more than 30 minutes, and computing the variance of the estimator requires a lot of simulations.

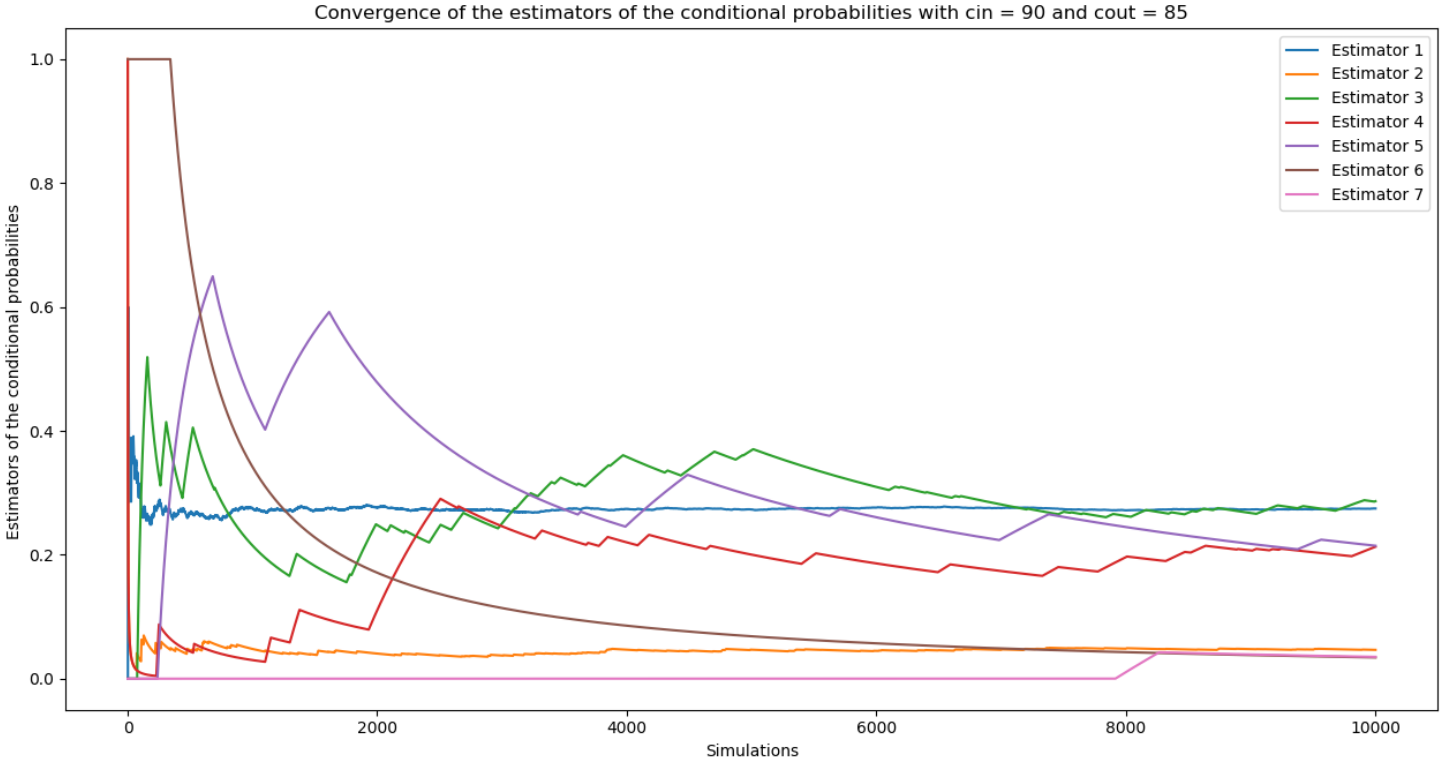

Figure 12: Value of the estimators depending on the number of simulations ($c_{in} = 90, c_{out} = 85$).

References

[1] R. Vershynin
Four lectures on probabilistic methods for data science, 2016 PCMI Summer School, AMS.