University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Community Detection in Networks

Supervisor
Prof. Marco Formentin
University of Padova

Master Candidate
Lorenzo Caltran

Student ID
2097200

Academic Year
2023-2024

Abstract

Community detection plays a very important role in understanding the structure and dynamics of networks, this thesis addresses the challenge of identifying such communities. The first chapter of this dissertation introduces some preliminary concepts, such as the definition of graph, Markov chains and stationary distributions with all the necessary and sufficient conditions for their existence and uniqueness; these aspects are going to be crucial for the understanding of the following chapters. The next three sections deal with three different community detection algorithms: Louvain, Infomap and Girvan-Newman. After explaining in detail their functioning, their properties and their limitations, at the end of each chapter we are going to see some applications and see how different network partitioning approaches yield very different results. In the final chapter we are going to study how the community structure in a complex network of users influences the efficiency of communication by e-mail between them. The aspect that we want to analyze is the response time defined as the number of units of activity of the receiving user pertaining to the intervals between a message and its response. The goal is to assess whether the modular structure of the network influences the probability distribution of the response times between different communities of users, where said communities are going to be detected using one of the three methods described in the previous chapters.

Contents

1 Preliminary concepts

In this first chapter, we are going to introduce a series of tools that are necessary for a better understanding of the algorithms.

1.1 Introduction to networks and community detection

This section focuses on the different types of graphs and the concept of community detection. The material was taken from [1] and [4].

Definition 1.1.1 *An undirected graph is a pair $G = (V, E)$, where V is a set whose elements are called vertices, and E is a set of unordered pairs (u, v) of vertices, whose elements are called edges (or links). The vertices u and v of an edge (u, v) are called the edge's endpoints. The edge is said to join u and v and to be incident on them. A vertex may belong to no edge, in which case it is not joined to any other vertex and is called isolated. When an edge (u, v) exists, the vertices u and v are called adjacent.*

A graph can also be called network and from now on we are going to use the two terms interchangeably.

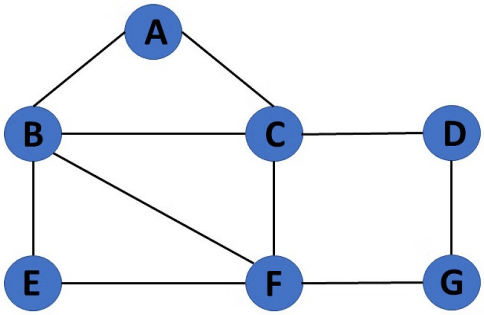

Figure 1.1: Example of an undirected graph. Picture taken from [2].

Definition 1.1.2 *In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. An undirected graph is said to be connected if every pair of vertices in the graph is connected .*

Definition 1.1.3 *A directed graph or digraph is a graph in which edges have orientations, in particular it is a pair G = (V, E) comprising:*

- *V, a set of vertices (or nodes)*
- *E, a set of edges (also called directed edges or directed links), which are ordered pairs of distinct vertices:* $E \subseteq \{(x, y) \mid (x, y) \in V^2, \text{ and } x \neq y\}$

In the edge (x, y) *directed from x to y, the vertices x and y are called the endpoints of the edge, x the tail of the edge and y the head of the edge.*

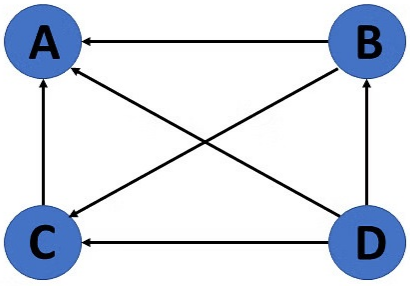

Figure 1.2: Example of a directed graph. Picture taken from [2].

Definition 1.1.4 *A directed graph G is strongly connected if, for every pair of nodes u and v, it contains a directed path from u to v and a directed path from v to u for every pair of vertices u, v.*

Definition 1.1.5 *A weighted graph or a network is a graph in which a number (the weight) is assigned to each edge. Weighted graphs can also be directed or undirected.*

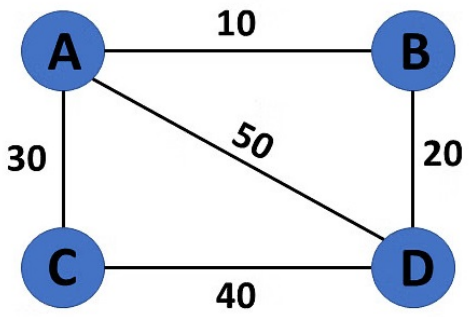

Figure 1.3: Example of a weighted undirected graph. Picture taken from [2].

Definition 1.1.6 *Given an unweighted graph* $G = (V, E)$*, the adjacency matrix A corresponding to G is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. More precisely:*

$$A_{ij} = \begin{cases} 1 \text{ if there is an edge from node } i \text{ to node } j \\ 0 \text{ otherwise} \end{cases}$$

Proposition 1.1.1 *If the graph is undirected, then the adjacency matrix is symmetric.*

PROOF In an undirected graph, if there is an edge from vertex i to vertex j, then there is also an edge from vertex j to vertex i. Therefore, $A_{ij} = 1$ if and only if $A_{ji} = 1$.
So, $\forall\, i, j \in V\ A_{ij} = A_{ji}$. Hence, the adjacency matrix A is symmetric. □

Definition 1.1.7 *Given an weighted graph* $G = (V, E)$*, the adjacency matrix A corresponding to G is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. More precisely:*

$$A_{ij} = \begin{cases} w_{ij} \text{ if there is an edge from node } i \text{ to node } j \\ 0 \text{ otherwise} \end{cases}$$

Where w_{ij} is the weight assigned to the edge (i, j).

Definition 1.1.8 *In graph theory, a loop (also called a self-loop or a buckle) is an edge that connects a vertex to itself. A graph is simple if doesn't contain any loops.*

Now we introduce the main concept of this dissertation, which is the concept of community in a network.

Definition 1.1.9 *A network is said to have community structure if the nodes of the network can be easily grouped into (potentially overlapping) sets of nodes such that each set of nodes is densely connected internally.*

We say that a set of nodes is densely connected if the number of edges $|E| \gtrsim log(|V|)$, where $|E|$ is the number of edges and $|V|$ the number of nodes.

Definition 1.1.10 *When the entire graph is densely connected then the graph is dense, on the contrary if it's not, the graph is sparse.*

Communities (also called modules or clusters), are often defined in terms of the partition of the set of vertices, that is each node is put into one and only one community. However, in some cases a better representation could be one where vertices are in more than one community, so in this case we are dealing with overlapping communities.

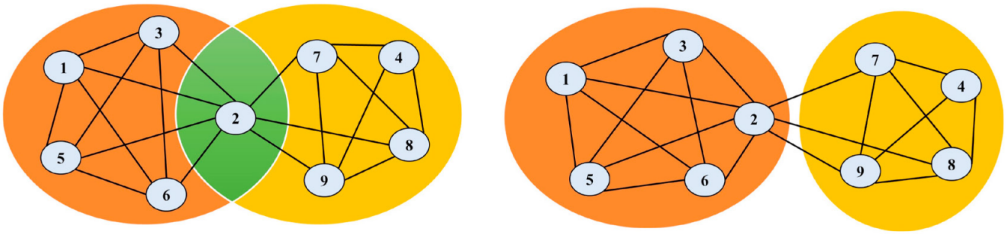

Figure 1.4: Example of overlapping and non-overlapping community detection. Picture taken from [3].

Not all graphs have a community structure, one example of such network is the following.

Example 1.1.1 (Random graphs) *A random graph, also known as Erdős–Rényi model, indicated as $G(n, p)$, is an undirected graph in which we have n nodes and the probability of connection between two nodes is equal to p. So we can say that the connection between two nodes can be expressed as $1_{\{\exists \text{ an edge from node } i \text{ to } j\}}$ and it's a Bernoulli random variable with parameter p.*

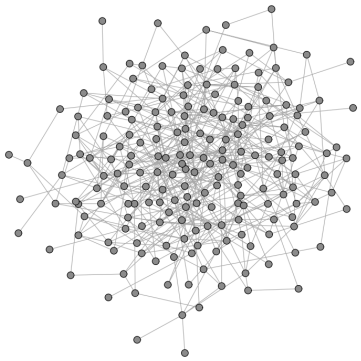

Figure 1.5: Example of a Erdős–Rényi graph G(n, p) with n = 200 and $p = \frac{1}{40}$ Picture taken from [4].

Definition 1.1.11 *In a random graph, a vertex i has degree d_i if it has d_i links.*
The degree d_i is a random variable, in fact $d_i = \sum_{j=1, j\neq i}^{n} 1_{\{\exists \text{ a link from } i \text{ to } j\}}$.
We have that $\mathbb{E}[d_i] = \sum_{j=1, j\neq i}^{n} \mathbb{E}\left[1_{\{\exists \text{ a link from } i \text{ to } j\}}\right] = \sum_{j=1, j\neq i}^{n} p = (n-1)p = d$.
So d_i is the sum of n-1 Bernoulli random variables.

In the case of random graphs, we consider them dense if d $\gtrsim$ log(n). With the next proposition we are going to show that dense graphs are almost regular with probability close to 1, which means that the degrees of all vertices approximately equal d.

Proposition 1.1.2 *Consider an Erdős–Rényi–Gilbert graph $G(n, p)$. For n large enough it holds that $\mathbb{P}(\{|d_i - d| < \delta d \ \forall i \in \{1, \dots, n\}\}) \geq 1 - \varepsilon \ \forall \delta, \varepsilon > 0$.*

Before proving this Proposition, we need to introduce an inequality necessary for the proof.

Theorem 1.1.1 (Chernoff's inequality for small deviations) *Let $X_1, \dots, X_N$ be independent Bernoulli random variables with parameters p_i. Consider their sum $S_N = \sum_{i=1}^{N} X_i$ and denote its mean $\mu = \mathbb{E}[S_N]$, then for $\delta \in (0, 1]$ we have that $\mathbb{P}(|S_N - \mu| < \delta\mu) \leq 2e^{-c\mu\delta^2}$*

We can now proceed with the proof of Proposition 1.1.2

Proof We know that $G(n, p)$ is a dense graph, which means that $\exists\, D_n > 0$ such that $d = (n-1)p \geq D_n log(n)$ for n large enough, where $D_n = min\{D \in \mathbb{R} \,|\, d \leq Dlog(n)\}$. In our case we can choose D_n such that $lim_{n\to+\infty} D_n = +\infty$, so we are going to take $D_n = log(n)$. Now we apply Theorem 1.1.1 with $X_j = 1_{\{\exists \text{ a link from i to j}\}} \sim Ber(p)$ and $d_i = S_n = \sum_{j=1, j\neq i}^{n} X_j$. For fixed i we have that $\mathbb{P}(|d_i - d| \geq \delta d) \leq e^{-cd\delta^2}$, with c > 0.
Now we write $\mathbb{P}(\{|d_i - d| < \delta d \ \forall i \in \{1, \dots, n\}\}) = 1 - \mathbb{P}(\{\exists \text{ i such that } |d_i - d| \geq$

$\geq \delta d\}) \leq 1 - \sum_{i=1}^{n} \mathbb{P}(|d_i - d| \geq \delta d) \leq 1 - 2ne^{-cd\delta^2}$.
We have to show that for n large enough $2ne^{-cd\delta^2} < \varepsilon$. In fact $2ne^{-cd\delta^2} < 2ne^{-cD_n log(n)\delta^2} < \varepsilon$ because $e^{-cD_n log(n)\delta^2} = O(n^{-\alpha})$ for $\alpha > 1$, so $lim_{n\to+\infty} e^{-cD_n log(n)\delta^2} = 0$. □

Observation 1.1.1 *The fact that in regular graphs all the nodes have a similar amount of connections it less likely for distinct communities to form because there are no nodes with significantly higher or lower degrees that could act as connectors between communities.*

The goal of community detection is to identify these clusters for a better understanding of the structure and dynamics of the network that we are dealing with.

1.2 Introduction to Markov chains and stationary distributions

In this section we're going to introduce the concept of Markov chains, specifically on a type of Markov chain, which is the random walk. The random walk is going to be one of the focal points in one of the algorithms that we're going to discuss. The material was taken from [5].

Definition 1.2.1 *A Markov chain is a sequence $\{X_n\}_{n\in\mathbb{N}}$ of a random variables taking value in a finite or countable set S that satisfy the Markov property, which is: $\forall\, X_0^{n+1} = (X_0, \ldots, X_{n+1}) \in \in S^{n+1}$ and $\forall\, n \geq 1$ $\mathbb{P}(X_{n+1} = x_{n+1} \,|\, X_0^n = x_0^n) = \mathbb{P}(X_{n+1} = x_{n+1} \,|\, X_n = x_n)$, where $X_n^m = (X_n, X_{n-1}, \ldots, X_m)$ $\forall\, m < n$. This property is also called memorylessness.*
The conditional probabilities $\mathbb{P}(X_{n+1} = x_{n+1} \,|\, X_n = x_n)$ are called the 1-step transition probabilities.

Observation 1.2.1 *The full law of the chain can be reconstructed if we know the transition probability and the initial distribution, i.e. $\mathbb{P}(X_0 = x_0)$ $\forall\, x_0 \in S$*

Definition 1.2.2 *A Markov chain is homogeneous if $\mathbb{P}(X_{n+1} = x_{n+1} \,|\, X_n = x_n)$ doesn't depend on n, so $\mathbb{P}(X_{n+1} = x \,|\, X_n = y) = \mathbb{P}(X_n = x \,|\, X_{n-1} = y)$ $\forall\, x, y \in S$ and $\forall\, n \geq 1$*

Definition 1.2.3 *Given a stochastic process, at each step, the transition matrix P is a square matrix given by $P = (p_{yx})_{y,x\in S} = (\mathbb{P}(X_{n+1} = y \,|\, X_n = x))_{y,x\in S}$. If the chain is homogeneous, the transition matrix is going to be the same at each step.*

Observation 1.2.2 *For a finite state homogeneous Markov chain, the transition matrix is always stochastic, i.e. all of its rows sum up to 1 and all of its entries are non-negative, this is because all of the entries represent probabilities.*

Let's now give an example of an homogeneous Markov chain.

Example 1.2.1 (Random walk) *A random walk is a random process that describes a path that consists of a succession of random steps on some mathematical space. An elementary example of a random walk is the random walk on the integer number line $\mathbb{Z}$ which starts at 0, and at each step moves $+1$ with probability p and -1 with probability $1 - p$. So the sequence of random variables becomes: $\{U_n\}_{n\in\mathbb{N}}$, where*

$$U_i = \begin{cases} 1 & \text{with probability } p \\ -1 & \text{with probability } 1 - p \end{cases}$$

We then define recursively the states:

$$\begin{cases} S_0 = 0 \\ S_1 = S_0 + U_1 \\ \vdots \\ S_n = S_{n-1} + U_n \\ \vdots \end{cases}$$

It is also possible to have a random walk on a finite set, but in this case we are only allowed to move forward when we are in the initial state and backwards when we are in the final state.

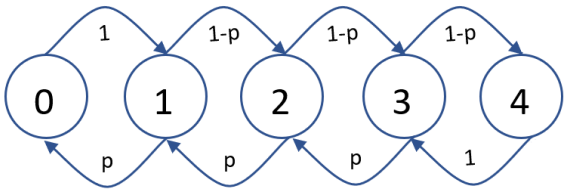

Figure 1.6: Example of random walk on the set $\{0,\ 1,\ 2,\ 3,\ 4\}$. Picture taken from [6].

From now on we are only going to consider homogeneous Markov chains.

Theorem 1.2.1 (Chapman-Kolmogorov equation) *Given an homogeneous Markov chain and the set of states S we have:*

$$p_{yx}^{n+m} = \mathbb{P}(X_{n+m} = x \,|\, X_0 = y) = \sum_{z \in S} p_{zx}^{m} p_{yz}^{n}$$

Proof We know that the probability space $\Omega = \bigsqcup_{z \in S} \{X_n = z\}$. By the total probability rule we get:

$$\begin{aligned}
\mathbb{P}(X_{n+m} = x \,|\, X_0 = y) &= \mathbb{P}(\{X_{n+m} = x\} \cap \Omega \,|\, X_0 = y) \\
&= \sum_{z \in S} \mathbb{P}(X_{n+m} = x, X_n = \ = z \,|\, X_0 = y) \\
&= \sum_{z \in S} \mathbb{P}(X_{n+m} = x, \,|\, X_n = z, X_0 = y) \mathbb{P}(X_n = z \,|\, X_0 = y)
\end{aligned}$$

Then by the Markovian and the homogeneous property

$$\begin{aligned}
&\sum_{z \in S} \mathbb{P}(X_{n+m} = x, \,|\, X_n = z, X_0 = y) \mathbb{P}(X_n = z \,|\, X_0 = y) \\
&= \sum_{z \in S} \mathbb{P}(X_{n+m} = x \,|\, X_n = z) \mathbb{P}(X_n = z \,|\, X_0 = y) \\
&= \sum_{z \in S} \mathbb{P}(X_m = x \,|\, X_0 = z) \mathbb{P}(X_n = z \,|\, X_0 = y) \\
&= \sum_{z \in S} p_{zx}^{m} p_{yz}^{n}
\end{aligned}$$

□

The Chapaman-Kolmogrov equation allows us to associate our n-step transition probabilities to a matrix of n-step transistion probabilities. Lets define the matrix $P^{(n)} = (p_{ij}^{(n)})$.

Corollary 1.2.1 $P^{(n)} = P^n$, *where* P^n *denotes the typical matrix multiplication.*

Proof We can rewrite the Chapaman-Kolmogrov equation in matrix form as $P^{(n)} = P^{(m)} P^{(n-m)}$ $\forall\, m \in \{0, 1, \ldots, n-1\}$. Now we can induct on n:

It's apparent that $P^{(1)} = P$, giving us $P^{(n+1)} = P^{(m)} P^{(n-m)}$.

□

Definition 1.2.4 *The vector π is called a stationary distribution of a Markov chain with matrix of transition probabilities P if π has entries π_j, where $j \in S$, such that*

- $\pi = \pi P$, *which is to say that* $\pi_j = \sum_{i \in S} \pi_i p_{ij} \; \forall j \in S$, *where p_{ij} are the entries of the transition matrix P. These are also called balance equations;*
- $\pi_j \geq 0 \; \forall j \in S$ *and* $\sum_{j \in S} \pi_i = 1$

Observation 1.2.3 *π being a stationary distribution implies that $\pi P^n = \pi \; \forall \, n \geq 0$.*

Proposition 1.2.1 *Every finite state Markov chain has a stationary probability distribution.*

This proposition is a consequence of the following Theorem.

Theorem 1.2.2 (Perron-Frobenius Theorem) *If $A \in \mathbb{R}^{n \times n}$ is non-negative, then:*

- *$\rho(A)$ is an eigenvalue of A*
- *There is a non-negative eigenvector x such that $Ax = \rho(A)x$*

Where $\rho(A)$ is the spectral radius of the matrix A.

For stochastic matrices we know that the spectral radius is always 1. In this case, Theorem 1.2.2 states that there exists a stationary distribution.

However, this theorem doesn't say anything about the uniqueness of the distribution. We are now going to introduce a series of concepts and results that allow us to study the uniqueness of such distribution.

Definition 1.2.5 *P is irreducible if $\forall \, x, y \in S \; \exists \, n \in \mathbb{N}$ such that $p_{xy}^n > 0$*

Definition 1.2.6 *Let $f_i = \mathbb{P}(\{\exists \, n \geq 1 \, \text{such that} \, X_n = i\} \,|\, X_0 = i)$.*

- *If $f_i = 1$, i is recurrent*
- *If $f_i < 1$, i is transient*

A Markov chain is recurrent if all of its states are recurrent and it's transient if they are transient.

Observation 1.2.4 *Let $N_i = |\{n \geq 0 \,|\, X_n = i\}|$ be the number of visits to the state i.*

- *If i is recurrent, then $\mathbb{P}(N_i = +\infty \,|\, X_0 = i) = 1$*

- *If i is transient, then* $\mathbb{P}(N_i = +\infty \,|\, X_0 = i) = 0$

Moreover, $N_i \sim Geo(1 - f_i)$, *so* $\mathbb{P}(N_i = m \,|\, X_0 = i) = f_i^m(1 - f_i)$

Definition 1.2.7 *Let* $T_i = inf\{n \geq 1 \,|\, X_n = i\}$ *be the first time visit to state* i *and* $m_i = = \mathbb{E}[T_i \,|\, X_0 = i]$.

- *If* $m_i < +\infty$, *then* i *is positive recurrent*
- *If* i *is recurrent and* $m_i = +\infty$, *then* i *is null recurrent.*

A Markov chain is positive recurrent if all of its states are positive recurrent and it's null recurrent if they are null recurrent.

Observation 1.2.5

- *If* i *is recurrent,* $\mathbb{P}(T_i < +\infty \,|\, X_0 = i) = 1$
- *If* i *is transient* $\mathbb{P}(T_i = +\infty \,|\, X_0 = i) > 1$, *then* $m_i = +\infty$
- *If* i *is recurrent,* $\sum_{n=0}^{+\infty} \mathbb{P}(T_i = n \,|\, X_0 = i) = 1$

Definition 1.2.8 *Let* $H_i = inf\{n \geq 0 \,|\, X_n = i\}$ *be the hitting to time state* i, *then the hitting probability of state* j *starting from state* i *is defined as* $h_{ij} = \mathbb{P}(\{\exists\, n \geq 0$ *such that* $X_n = = j\}|\, X_0 = i)$.
The expected hitting time η_{ij} *of the state* j *starting from state* i *is* $\eta_{ij} = \mathbb{E}\left[H_j \,|\, X_0 = i\right]$

Observation 1.2.6

- $m_k = 1 + \sum_{j \in S} p_{kj}\eta_{jk}$
- $\eta_{ik} = \begin{cases} 1 + \sum_{j \in S} p_{ij}\eta_{jk} & \text{if } i \neq k \\ 0 & \text{if } i = k \end{cases}$

Proof

- We can write $T_K = H_k + 1$. By using the Markovian property and conditioning over the first step and we get

$$\begin{aligned} m_k &= \mathbb{E}[T_k \,|\, X_0 = k] = 1 + \mathbb{E}[H_k \,|\, X_0 = k] \\ &= 1 + \sum_{j \in S} \mathbb{E}[H_k \,|\, X_1 = j, X_0 = k]\, \mathbb{P}(X_1 = j \,|\, X_0 = k) \\ &= 1 + \sum_{j \in S} \mathbb{E}[H_k \,|\, X_1 = j]\, p_{kj} \\ &= 1 + \sum_{j \in S} p_{kj}\eta_{jk} \end{aligned}$$

- If $i = k$, it's clear that $\eta_{ii} = 0$. For $i \neq k, H_k = H'_k$, where H'_k denotes the remaining time to reach k. So now we follow a similar procedure as in the previous proof

$$\begin{aligned}
\eta_{ik} &= \mathbb{E}\left[H_k \,|\, X_0 = i\right] = 1 + \mathbb{E}\left[H'_k \,|\, X_0 = i\right] \\
&= 1 + \sum_{j \in S} \mathbb{E}\left[H'_k \,|\, X_1 = j, X_0 = i\right] \mathbb{P}(X_1 = j \,|\, X_0 = i) \\
&= 1 + \sum_{j \in S} \mathbb{E}\left[H'_k \,|\, X_1 = j\right] p_{ij} \\
&= 1 + \sum_{j \in S} p_{ij} \eta_{jk}
\end{aligned}$$

□

Theorem 1.2.3 *If a Markov chain is irreducible and positive recurrent, then a stationary distribution π exists, is unique, and is given by $\pi_i = \frac{1}{m_i}$.*

Before proceeding with the proof we need to introduce a Lemma

Lemma 1.2.1 *Let $\{X_n\}_n$ be an irreducible and recurrent Markov chain. Then for any initial distribution and any state j, the $\mathbb{P}(H_j < +\infty) = 1$*

Proof Let's arbitrarily fix a state i, since the chain is irreducible, we have $p_{ij}^m > 0$, for some $m > 0$. Since the chain is recurrent, we know the return probability from j to j is 1, and we return infinitely many times with probability 1. Using those facts allows us to conclude that:

$$\begin{aligned}
1 &= \mathbb{P}(X_n = j \text{ for infinitely many n} \,|\, X_0 = j) \\
&= \mathbb{P}(X_n = j \text{ for some n > m} \,|\, X_0 = j) \\
&= \sum_{k \in S} \mathbb{P}(X_m = k \,|\, X_0 = j) \mathbb{P}(X_n = j \text{ for some n > m} \,|\, X_m = k, X_0 = j) \\
&= \sum_{k \in S} p_{jk}^m \mathbb{P}(H_j < +\infty \,|\, X_0 = k)
\end{aligned}$$

where the last line used the Markov property to treat the chain as starting over again when it reaches some state k at time m. Note that $\sum_{k \in S} p_{jk}^m = 1$, since that's the sum of the probabilities of going anywhere in m steps. This means we must have $\mathbb{P}(H_j < +\infty \,|\, X_0 = k) = 1$ whenever $p_{ij}^m > 0$, to ensure the final line does indeed sum to 1. But we stated earlier that $p_{ij}^m > 0$, so we indeed have $\mathbb{P}(H_j < +\infty \,|\, X_0 = j) = 1$, as required. □

We can now prove Theorem 1.2.3

Proof

- **Existence**: Suppose that the Markov chain $\{X_n\}_n$ is recurrent. Fix an initial state k, and let ν_i be the expected number of visits to i before we return back to k.

$$\begin{aligned}\nu_i &= \mathbb{E}\left[|\{n \geq 1 \,|\, X_n = i \text{ and } X_m \neq k \ \forall m \in \{1, \dots, n-1\}\}| \,|\, X_0 = k\right] \\ &= \mathbb{E}\left[\sum_{n=1}^{T_k} \mathbb{P}(X_n = i \,|\, X_0 = k)\right] \\ &= \sum_{n=1}^{+\infty} \mathbb{P}(X_n = i \text{ and } T_k \geq n \,|\, X_0 = k)\end{aligned}$$

T_k is the return time of Definition 1.2.7. Under this definition, $\nu_k = 1$, because the only visit to k is the return to k itself. Since ν is counting the number of visits to different states in a certain time, it seems plausible that ν suitably normalised could be a stationary distribution, meaning that ν itself could be a stationary vector.

$$\begin{aligned}\sum_{i \in S} \nu_i p_{ij} &= \sum_{i \in S} \sum_{n=1}^{+\infty} \mathbb{P}(X_n = i \text{ and } T_k \geq n \,|\, X_0 = k) p_{ij} \\ &= \sum_{n=1}^{+\infty} \sum_{i \in S} \mathbb{P}(X_n = i, X_{n+1} = j \text{ and } T_k \geq n \,|\, X_0 = k) \\ &= \sum_{n=1}^{+\infty} \mathbb{P}(X_{n+1} = j \text{ and } T_k \geq n \,|\, X_0 = k)\end{aligned}$$

We can exchange the order of the sums because thanks to Observation 1.2.5, we know that the recurrence of the chain implies that T_k is finite with probability 1. Now we swap the visit to k at time T_k with the visit to k at time 0. This means instead of counting the visits from 1 to T_k, we can count the visits from 0 to $T_k - 1$.

$$\begin{aligned}\sum_{i \in S} \nu_i p_{ij} &= \sum_{n=0}^{+\infty} \mathbb{P}(X_{n+1} = j \text{ and } T_k - 1 \geq n \,|\, X_0 = k) \\ &= \sum_{n+1=1}^{+\infty} \mathbb{P}(X_{n+1} = j \text{ and } T_k \geq n+1 \,|\, X_0 = k)\end{aligned}$$

$$= \sum_{n=1}^{+\infty} \mathbb{P}(X_n = j \text{ and } T_k \geq n \,|\, X_0 = k) = \nu_i.$$

We now want normalise ν into a stationary distribution by dividing through by $\sum_{i=1}^{n} \nu_i$. We can do this if $\sum_{i=1}^{n} \nu_i$ is finite, but $\sum_{i=1}^{n} \nu_i$ is the expected total number of visits to all states before return to k, which is precisely the expected return time m_k. Now we use the assumption that $\{X_n\}_n$ is positive recurrent, this means that μ_k is finite, so $\pi = \left(\frac{1}{m_k}\right) \nu$ is a stationary distribution.

- **Uniqueness**: Suppose the Markov chain is irreducible and positive recurrent, and suppose π is a stationary distribution. We want to show that $\pi_i = \frac{1}{m_i} \; \forall \, i \in S$. Using the two equations in Observation 1.2.6, we get:

$$\sum_{i \in S} \pi_i \eta_{ik} = \sum_{i \in S,\, i \neq k} \pi_i + \sum_{j \in S} \sum_{i \in S,\, i \neq k} \pi_i p_{ij} \eta_{jk} \tag{1.1}$$

The sum on the left can be over all i, since $\eta_{kk} = 0$. If we take the first equality of Observation 1.2.6 and multiply it by π_k we get

$$\pi_k m_k = \pi_k + \sum_{j \in S} \pi_k p_{kj} \eta_{jk} \tag{1.2}$$

If we sum Equations 1.1 and 1.2 we get:

$$\sum_{i \in S} \pi_k \eta_{ik} + \pi_k m_k = \sum_{i \in S} \pi_i + \sum_{j \in S} \sum_{i \in S} \pi_i p_{ij} \eta_{jk}$$

We can now use $\sum_{i \in S} \pi_i p_{ij} = \pi_j$ and $\sum_{i \in S} \pi_i = 1$ to get

$$\sum_{i \in S} \pi_k \eta_{ik} + \pi_k m_k = 1 + \sum_{j \in S} \pi_j \eta_{jk}$$

But the first term on the left and the last term on the right are equal, and because the Markov chain is irreducible and positive recurrent, thanks to Lemma 1.2.1, they are finite. Thus we're allowed to subtract them, and we get $\pi_k m_k = 1$, which is $\pi_k = \frac{1}{m_k}$.

□

Proposition 1.2.2 *For an irreducible Markov chain $\{X_n\}_n$ on a finite state space, all the states are positive recurrent, hence a stationary distribution exists and it's unique.*

PROOF Let's fix a state i, let's define

$$\begin{aligned} h_{ij}^{(m)} &= \mathbb{P}(X_n = i \text{ for some } 1 < n < m \,|\, X_0 = j) \\ &= \sum_{n=1}^{m} \mathbb{P}(X_n = i \,|\, X_0 = j \text{ and } X_k \neq i \ \forall k \in \{1, \dots, n-1\}) \end{aligned}$$

Since the Markov chain is irreducible, $lim_{m\to+\infty} h_{ji}^{(m)} = h_{ji} > 0 \ \forall j \in S$, where h_{ji} is the same as in Definition 1.2.8. Hence, since the state space is finite, we can find $m \in \mathbb{N}$ and $\delta > 0$ such that $h_{ji}^{(m)} \geq \delta \ \forall j \in S$.
But we must have that $1 - h_{ii}^{(n)} \leq (1-\delta)^{\lfloor \frac{n}{m} \rfloor}$, so letting T_i and m_i as in Definition 1.2.7 we have that

$$m_i = \sum_{n=0}^{+\infty} \mathbb{P}(T_i > n+1 \,|\, X_0 = i) = \sum_{n=0}^{+\infty} (1-h_{ii}^{(n)}) \leq \sum_{n=0}^{+\infty} (1-\delta)^{\lfloor \frac{n}{m} \rfloor} = \frac{m}{\delta} < +\infty$$

We have now that the chain $\{X_n\}_n$ is irreducible and positive recurrent, so thanks to Theorem 1.2.3, we have a unique stationary distribution given by $\pi_i = \frac{1}{m_i}$. □

2

Louvain algorithm

The Louvain algorithm is a community detection algorithm and its aim is to maximize modularity, which measures the quality of a network's partition. The material was taken from [7], [8], [9], [10], [11], [12], [13], [14] and [19].

2.1 Modularity as quality measure of a network's partition

The idea behind this measure is that if the observed number of edges between the two groups is close to what you would expect if you were to randomly assign them, then there is no specific pattern between the groups.

In this section we are going to study the measure of modularity in the case of undirected graphs and then give a generalization for directed graphs.

First of all we need to introduce an example.

Example 2.1.1 (Molloy-Reed model) *The Molloy-Reed model (or configuration model or null model) is a method for generating random networks from a given degree sequence and it is obtained in the following way:*

- *Calculate the degree* $k_i = \sum_j A_{ij}$ *of each vertex, where* A_{ij} *is the adjacency matrix of the network;*

- *Unwire nodes by cutting each edge into two halves, called a stubs. The sum of stubs must be even in order to be able to construct a graph, so we say that the total number of stubs is* $2m = \sum_i k_i$;

- *Choose two stubs uniformly at random and connect them to form an edge. Choose another pair from the remaining* $2m - 2$ *stubs and connect them. Continue until you run out of stubs. The result is a network with the pre-defined degree sequence.*

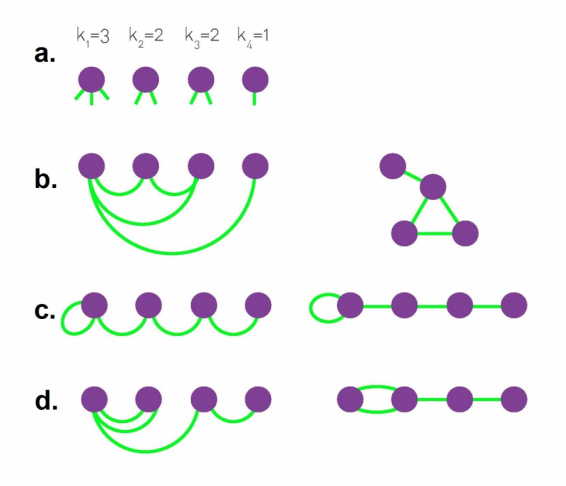

Figure 2.1: Degree sequence and different network realizations in the undirected configuration model. Picture taken from [7].

In the directed case, we unwire nodes by breaking edges but keep stubs and their direction so that nodes keep their in/out degree and then rewire stubs at random, linking output stubs to input stubs.

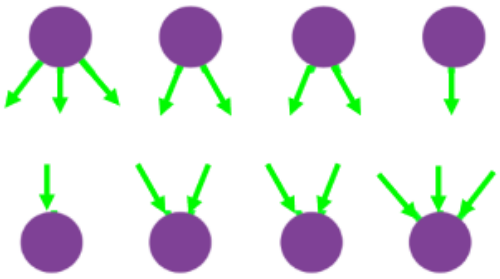

Figure 2.2: Degree sequence in the directed configuration model. Picture taken from [7].

Observation 2.1.1 *The realization of the network changes with the order in which the stubs are chosen, they might include cycles or self-loops.*

Observation 2.1.2 *This model assumes that each node can get attached to any other node of the network. This becomes almost impossible if the network is very large, in fact in reality, a node's connections are typically limited to a small fraction of all the other vertices.*

Theorem 2.1.1 *The probability of having an edge between node i and node j is given by* $p_{ij} = \frac{k_i k_j}{2m}$*, where* k_i *and* k_j *are the respective degrees of nodes i and j.*

PROOF Let us consider each of the k_i stubs of node i and create associated indicator variables $1_k^{(i,j)}$ for them, $k = 1, \ldots, k_i$ with $1_k^{(i,j)} = 1$ if the k-th stub happens to connect to one of the k_j stubs of node j in this particular random graph. If it does not, then $1_k^{(i,j)} = 0$. Since the k-th stub of node i can connect to any of the 2m − 1 remaining stubs with equal probability, and since there are k_j stubs it can connect to associated with node j, then

$$\mathbb{P}(1_k^{(i,j)} = 1) = \mathbb{E}\left[1_k^{(i,j)}\right] = \frac{k_j}{2m-1}$$

The total number of full edges N_{ij} between i and j is just $N_{ij} = \sum_{k=1}^{k_i} 1_k^{(i,j)}$, so the expected value of this quantity is

$$\mathbb{E}\left[N_{ij}\right] = \mathbb{E}\left[\sum_{k=1}^{k_i} 1_k^{(i,j)}\right] = \sum_{k=1}^{k_i} \mathbb{E}\left[1_k^{(i,j)}\right] = \sum_{k=1}^{k_i} \frac{k_j}{2m-1} = \frac{k_i k_j}{2m-1}.$$

When m is large, we can drop the subtraction of 1 in the denominator above and simply use the approximate expression $\frac{k_i k_j}{2m}$ for the expected number of edges between two nodes. Additionally, in a large random network, the number of self-loops and multi-edges is extremely small. Ignoring self-loops and multi-edges allows one to assume that there is at most one edge between any two nodes. In that case, N_{ij} becomes a binary indicator variable, so its expected value is also the probability that it equals 1, which means one can approximate the probability of an edge existing between nodes i and j as $\frac{k_i k_j}{2m}$. □

Observation 2.1.3 *For directed networks, we can prove in a similar way that the probability of having an edge between node i and node j is given by* $p_{ij} = \frac{k_i^{in} k_j^{out}}{m}$*, where* k_i^{in} *and* k_j^{out} *are the in-degree and out-degree of nodes i and j.*
The reason why we have m at the denominator instead of $2m$ *is because each stub of node i can connect only to the stubs with the opposite direction, so the number of available links is m and not* $2m$*.*

Definition 2.1.1 *Modularity is defined as the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random, like in Example 2.1.1. In particular it is defined as:*

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j) \tag{2.1}$$

Where A_{ij} represents the weight of the edge between i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function is defined as $\delta(u,v) = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$ For directed networks we get that the modularity is given by:

$$Q = \frac{1}{m}\sum_{i,j}\left(A_{ij} - \frac{k_i^{in} k_j^{out}}{m}\right)\delta(c_i, c_j) \tag{2.2}$$

Observation 2.1.4 *The modularity can be either positive or negative, in particular it's a scalar value between -1 and 1.*
Positive values indicate the possible presence of community structure, thus one can search for community structure precisely by looking for the divisions of a network that have positive, and preferably large, values of the modularity.

Observation 2.1.5 *Observation 2.1.2 implies that the expected number of edges between two groups of nodes decreases if the size of the network increases. So, if a network is large enough, the expected number of edges between two groups of nodes in modularity's configuration model may be smaller than one. What this implies is that modularity grows with the size of the graph and the number of clusters, so Q is not a good measure to compare graphs very different in size.*

Proposition 2.1.1 *The modularity Q for undirected networks can also be expressed in matrix form as:*

$$Q = \frac{1}{2m}Tr(CBC^T) \tag{2.3}$$

Where $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$ and C is community assignment matrix $C \in \mathbb{R}^{n_1 \times n_2}$, where n_1 is the number of communities and n_2 the number of nodes, is defined as

$$C_{ir} = \begin{cases} 1 & \text{if node } i \text{ belongs to community } r \\ 0 & \text{otherwise} \end{cases}$$

While for directed networks it can be expressed as:

$$Q = \frac{1}{m} Tr(CBC^T) \tag{2.4}$$

Where C is the same as in the previous equation and $B_{ij} = A_{ij} - \frac{k_i^{in} k_j^{out}}{m}$

Proof The proof is based on the fact that the δ-function can be seen as $\delta(c_i, c_j) = \sum_r C_{ir} C_{jr}$, allowing us to write $Q = \sum_{i,j} \sum_r \left(A_{ij} - \frac{k_i k_j}{2m} \right) C_{ir} C_{jr} = \frac{1}{2m} Tr(CBC^T)$.
We can prove the formula for the directed case in a similar way. □

Thanks to the previous Proposition we can introduce a method that allows us to calculate this measure in just a few steps.

Algorithm 2.1 Computation of the modularity Q for undirected networks

input Adjacency matrix A_0 of an undirected network G
Compute the sum of the entries $D_0 = 1^T A_0 1$
Compute the normalized adjacency matrix $A = \frac{A_0}{D_0}$
Compute the normalized degree vector $d = A1$
Compute the community assignment matrix C
Compute the modularity as $Q = Tr(C(A - dd^T)C^T)$
return Q

Observation 2.1.6 *Note that* $D_0 = 2m$*, so each entry of the normalized adjacency matrix A corresponds to the term* $\frac{a_{ij}}{2m}$*, where* a_{ij} *is the entry of the original adjacency matrix* A_0*.*
Also each entry of the vector d corresponds to the term $\frac{k_i}{2m}$*.*

We can compute the modularity measure for directed networks by following a similar procedure.

Algorithm 2.2 Computation of the modularity Q for directed networks

input Adjacency matrix A_0 of an directed network G
Compute the sum of the entries $D_0 = 1^T A_0 1$
Compute the normalized adjacency matrix $A = \frac{A_0}{D_0}$
Compute the normalized in-degree vector $d_{in} = A1$
Compute the normalized out-degree vector $d_{out} = A^T 1$
Compute the community assignment matrix C
Compute the modularity as $Q = Tr(C(A - d_{in} d_{out}^T)C^T)$
return Q

Observation 2.1.7 *Similarly we can note that $D_0 = m$, so each entry of the normalized adjacency matrix A corresponds to the term $\frac{a_{ij}}{m}$, where a_{ij} is the entry of the original adjacency matrix A_0. Also each entry of the vector d^{in} corresponds to the term $\frac{k_i^{in}}{m}$ and d^{out} to $\frac{k_j^{out}}{m}$.*

Another way to see modularity is by using a probabilistic approach.

Proposition 2.1.2 *The modularity Q can be calculated as:*

$$Q = Tr(P_{CC} - p_c p_c^T) \tag{2.5}$$

Where $P_{CC} = CAC^T$, where A and C are computed as in Algorithm 2.1 and 2.2. P_{CC} can be interpreted as a probability matrix linking communities, its entries are the sum of the links of A from community i to community j

2.2 Louvain's functioning and pseudo-code

Definition 2.2.1 *A greedy optimization algorithm is an algorithm that makes a locally optimal choice at each step with the hope that these local solutions will lead to a global minimum (or maximize) of the measure that it's trying to minimize (or maximize). At each step, the algorithm makes the best possible choice based on the current state without considering the overall problem.*

One example of a greedy maximization algorithm is Louvain.

Algorithm 2.3 Louvain algorithm

input Network G = (V, E), where V = set of n vertices, E = set of edges.
repeat
  Assign each node to its own community $M = \{m_i = \{v_i\} \mid v_i \in V\}$
  Compute the modularity Q_i for each single node community
  while some nodes are moved
    for i=0 to n - 1
      $m_{new} = bestNewModule(M, v_{N[i]})$, where $N[i]$ are the neighboring nodes of i
      Move $v_{N[i]}$ to m_{new} module
    end for
  end while
  if $\Delta Q > 0$
    Update M
    Update Q
  else
    return M
  end if
until

The algorithm is divided in two phases that are repeated iteratively. Assume that we start with a weighted network of n nodes. First, each node of the networks form its own community, so in the initial state the number of communities is the same as the number of nodes. Then, for each node i we consider the neighbours j of i and we evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i is then placed in the community for which this gain is maximum, but only if this gain is positive. If no positive gain is possible, i stays in its original community. This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved and the first phase is then complete.

Observation 2.2.1 *A node can be reconsidered multiple times during the local modularity optimization phase. This means after a node is moved to a new community, it may still be evaluated again in subsequent iterations to see if moving it again would further increase modularity.*

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual move can improve the modularity.

Theorem 2.2.1 *The gain in modularity ΔQ obtained by moving an isolated node i into a community C can easily be computed by:*

$$\Delta Q = \left(\frac{\sum_{in} + 2k_i^{in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right) - \left(\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right) \quad (2.6)$$

Where $\sum_{in}$ is the sum of the weights of the links inside C, $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C, k_i is the sum of the weights of the links incident to node i, k_i^{in} is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network.

Proof First, we compute the modularity of the isolated cluster of node i, which we will call c_i. Here we are assuming that there are no loops, and so $A_{\mu\mu} = 0 \ \forall \mu$

$$Q_i^{prev} = \frac{1}{2m} \sum_{\mu=1}^{N_i=1} \sum_{\nu=1}^{N_i=1} A_{\mu\nu} - \left(\sum_{\mu=1}^{N_i=1} \frac{k_\mu}{2m} \right)^2 = - \left(\frac{k_i}{2m} \right)^2$$

Next, we compute the modularity of the cluster c_j before we have added the new node i, which is

$$Q_j^{prev} = \frac{1}{2m} \sum_{\mu=1}^{N_j} \sum_{\nu=1}^{N_j} A_{\mu\nu} - \left(\sum_{\mu=1}^{N_j} \frac{k_\mu}{2m} \right)^2$$

Finally, we compute the modularity of the cluster c_j after we have added a new node i:

$$Q_j^{prev} = \frac{1}{2m} \sum_{\mu=1}^{N_j+1} \sum_{\nu=1}^{N_j+1} A_{\mu\nu} - \left(\sum_{\mu=1}^{N_j+1} \frac{k_\mu}{2m} \right)^2$$

We can rewrite the first term as follows:

$$\begin{aligned} \frac{1}{2m} \sum_{\mu=1}^{N_j+1} \sum_{\nu=1}^{N_j+1} A_{\mu\nu} &= \frac{1}{2m} \left(\sum_{\mu=1}^{N_j} \sum_{\nu=1}^{N_j} A_{\mu\nu} + 2 \sum_{\mu=1}^{N_j+1} A_{\mu,N_j+1} \right) \\ &= \frac{1}{2m} \left(\sum_{\mu=1}^{N_j} \sum_{\nu=1}^{N_j} A_{\mu\nu} + 2k_i^{in} \right) \end{aligned}$$

We can rewrite the second term as:

$$\begin{aligned}
\left(\sum_{\mu=1}^{N_j+1}\frac{k_\mu}{2m}\right)^2 &= \sum_{\mu=1}^{N_j+1}\sum_{\nu=1}^{N_j+1}\frac{k_\mu k_\nu}{(2m)^2} \\
&= \sum_{\mu=1}^{N_j}\sum_{\nu=1}^{N_j}\frac{k_\mu k_\nu}{(2m)^2} + 2k_{N_j+1}\sum_{\mu=1}^{N_j+1}\frac{k_\mu}{(2m)^2} + \frac{(k_{N_j+1})^2}{(2m)^2} \\
&= \frac{1}{(2m)^2}\left(\left(\sum_{\mu=1}^{N_j}k_\mu + k_{N_j+1}\right)\left(\sum_{\nu=1}^{N_j}k_\nu + k_{N_j+1}\right)\right) \\
&= \left(\frac{\sum_{\mu=1}^{N_j}k_\mu + k_{N_j+1}}{2m}\right)^2
\end{aligned}$$

Putting this together we have

$$Q_j^{updated} = \frac{1}{2m}\left(\sum_{\mu=1}^{N_j}\sum_{\nu=1}^{N_j}A_{\mu\nu} + 2k_i^{in}\right) - \left(\frac{\sum_{\mu=1}^{N_j}k_\mu + k_{N_j+1}}{2m}\right)^2$$

Putting together the equations for Q_i^{prev}, Q_j^{prev} and $Q_j^{updated}$, we can compute the change in modularity ΔQ for adding an isolated node i to the cluster c_j. This is sometimes referred to as the gain:

$$\begin{aligned}
\Delta Q &= Q_j^{updated} - Q_j^{prev} - Q_i^{prev} \\
&= \frac{1}{2m}\left(\sum_{\mu=1}^{N_j}\sum_{\nu=1}^{N_j}A_{\mu\nu} + 2k_i^{in}\right) - \left(\frac{\sum_{\mu=1}^{N_j}k_\mu + k_{N_j+1}}{2m}\right)^2 \\
&\quad - \left(\frac{1}{2m}\sum_{\mu=1}^{N_j+1}\sum_{\nu=1}^{N_j+1}A_{\mu\nu} - \left(\sum_{\mu=1}^{N_j+1}\frac{k_\mu}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right)
\end{aligned}$$

We know that $N_j + 1 = i$ and if we define $\sum_{\mu=1}^{N_j}k_\mu = \sum_{tot}$ and $\sum_{\nu=1}^{N_j}A_{\mu\nu} = \sum_{in}$ we get Equation 2.6. □

A similar expression is used in order to evaluate the change of modularity when i is removed from its community. In practice, one therefore evaluates the change of modularity by removing

i from its community and then by moving it into a neighbouring community.

The second phase of the algorithm consists in building a new network whose nodes are now the communities found during the first phase. To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Links between nodes of the same community lead to self-loops for this community in the new network. Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate. Let us denote by "pass" a combination of these two phases. By construction, the number of meta-communities decreases at each pass, and as a consequence most of the computing time is used in the first pass. The passes are iterated until there are no more changes and a maximum of modularity is attained.

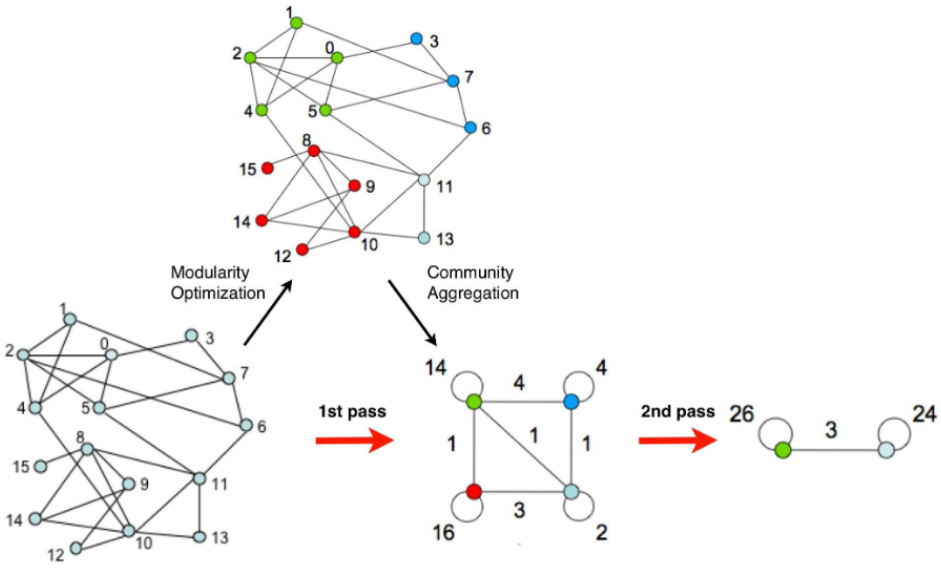

Figure 2.3: Visualization of the steps of our algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the found communities are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible. Picture taken from [9].

2.2.1 Louvain's complexity and limitations

Observation 2.2.2 *Note that the matrix C in Equation 2.3 and 2.4 is a binary matrix, hence Louvain produces only non-overlapping communities, which means that each node can belong to at most one community.*

Observation 2.2.3 *Note that the order of nodes is selected at random, hence different order of nodes might yield different results.*

One way to mitigate this problem is by using consensus clustering.

Algorithm 2.4 Consensus clustering for Louvain

input Network G = (V, E), where V = set of n vertices, E = set of edges. Threshold value ε
repeat
  for i=0 to P − 1
    Apply Algorithm 2.3 to the network G
  end for
  Compute the consensus matrix D, where D_{ij} is the fraction of partitions in which vertices i and j are assigned to the same cluster in C_P
  for i, j = $1, \ldots, n$
    if $D_{ij} < \varepsilon$
      Set $D_{ij} = 0$
    end if
  end for
  if $D_{ij} = 1 \ \forall i, j \in \{1, \ldots, n\}$
    return M
  else
    Update G by creating a new network, where its adjacency matrix is given by D
  end if
until

We apply the Louvain algorithm P times to a network and we get different partitions, but we expect that these are somehow related.

Observation 2.2.4 *The condition $D_{ij} = 1 \ \forall i, j \in \{1, \ldots, n\}$ is equivalent to saying that all the P partitions obtained by applying Louvain P times are the same.*

Theorem 2.2.2 (Scalability of Louvain) *Louvain is a scalable algorithm as it has complexity $O(nlog(n))$, where n is the number of nodes in the network.*

PROOF For each node, the algorithm tries to move the node to a neighboring community if the move results in an increase in modularity. This process is repeated iteratively for all nodes until no further improvement can be made. The time complexity for each move is $O(1)$ since it involves checking a constant number of communities. Each node is considered once per iteration. In the worst case, each node might be considered for moving O(n) times.

Once the modularity optimization step stabilizes, the algorithm creates a new network where communities from the previous step are treated as single nodes. This reduces the number of nodes in the network. The new network is then subjected to the same process of modularity optimization. If we assume the number of nodes is halved in each phase, the number of phases would be O(log(n)). □

Definition 2.2.2 *The resolution of a community detection algorithm refers to the level of detail or granularity with which the algorithm identifies communities within a network.*

Remark 2.2.1 (Resolution of Louvain) *Another implication of Observation 2.1.2, is that if a network is large enough, the expected number of edges between two groups of nodes in the configuration model may be smaller than one. If this happens, a single edge between the two clusters would be interpreted by modularity as a sign of a strong correlation between the two clusters, and optimizing modularity would lead to the merging of the two clusters, independently of the clusters' features. For this reason, optimizing modularity in large networks would fail to detect small communities, even when they are well defined.*

Definition 2.2.3 *The resolution limit can be mitigated by controlling the strength of the null model and we do that by introducing a parameter γ, so now instead of minimizing Equation 2.5 we minimize:*

$$Q = Tr(P_{CC} - \gamma p_c p_c^T)$$

Observation 2.2.5 $\gamma > 1$ *increases the number of communities, while* $\gamma < 1$ *decreases it.*

2.3 Application to four different types of networks

In this section we are going to see different applications of the Louvain algorithm on different types of graphs, more specifically, we are going to apply the algorithm to each type of network described in Section 1.1: unweighted undirected, weighted undirected, unweighted directed and weighted directed. The four networks are the Jazz musicians graph, the Windsurfers graph, the Macaque rhesus brain graph and the US Congress Twitter graph.

2.3.1 Unweighted undirected network

As a unweighted undirected graph we selected the Jazz musicians graph. This graph represents a collaboration network between Jazz musicians: each node is a Jazz musician and an edge denotes that two musicians have played together in a band. The network is composed by 198 vertexes and 2742 edges. The data was collected from [15].

The algorithm detected 4 communities, where the first one is made by 67 noes, the second one by 66, the third by 62 and the last by 3.

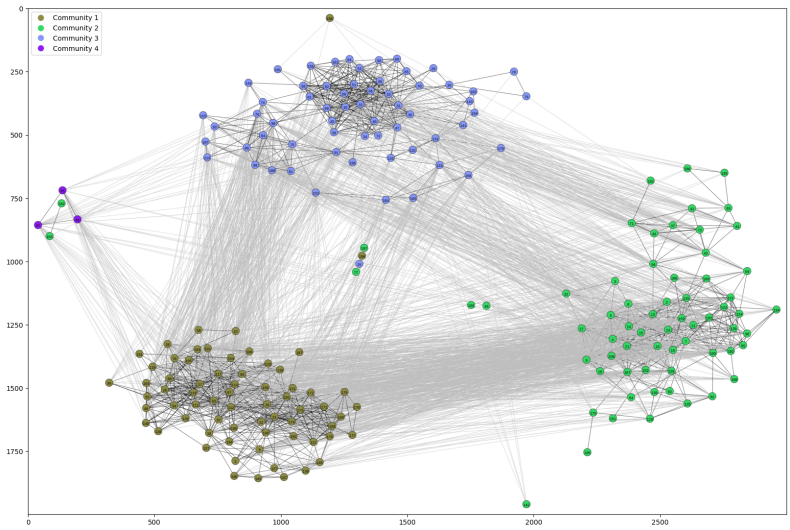

Figure 2.4: Graphical representation of the modules detected by the Louvain algorithm on the Jazz musicians network.

This partition yields a modularity of 0.43890781537538287.

2.3.2 Weighted undirected network

As a weighted undirected graph we selected the Windsurfers network. This graph represents the interpersonal contacts between windsurfers in southern California during the fall of 1986. A node represents a windsurfer and an edge between two windsurfers shows that there was a interpersonal contact and the edge weights indicate the perception of social affiliations majored by the tasks in which each individual was asked to sort cards with other surfer's name in the order of closeness. The network is composed by 43 vertexes and 336 edges. The data was collected from [16].

The algorithm detected 2 communities, where the first one has 22 noes and the second one 21.

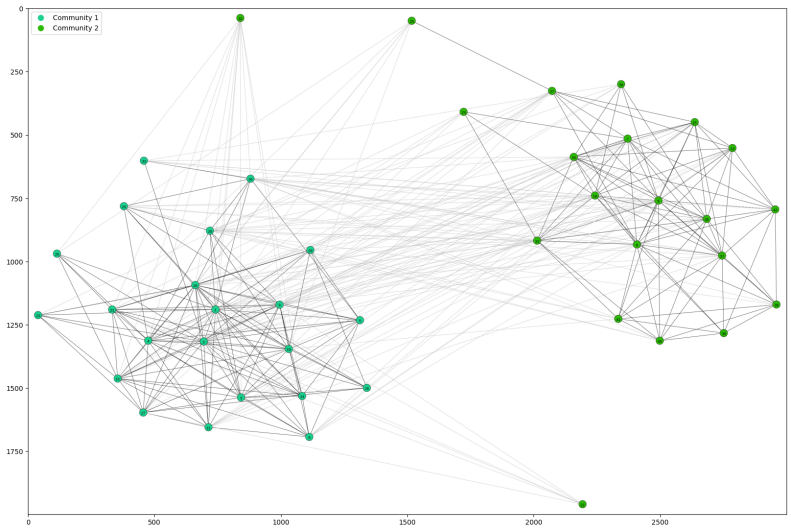

Figure 2.5: Graphical representation of the modules detected by the Louvain algorithm on the Windsurfers network.

This partition yields a modularity of 0.37121605900844046.

2.3.3 Unweighted directed network

As unweighted directed graph we used the Macaque rhesus brain graph, which represents the brain network of the rhesus macaque. Each node in the graph corresponds to different anatomical regions of the macaque brain. Each node is a specific area that has been anatomically defined and is part of the brain's network. The edges represent the presence of anatomical connections (white matter tracts) between these brain regions. The network is composed by 242 vertexes and 4090 edges. The data was collected from [17].

The algorithm detected 4 communities, where the first one is contains 88 nodes, the second one 76, the third 54 and the last 24.

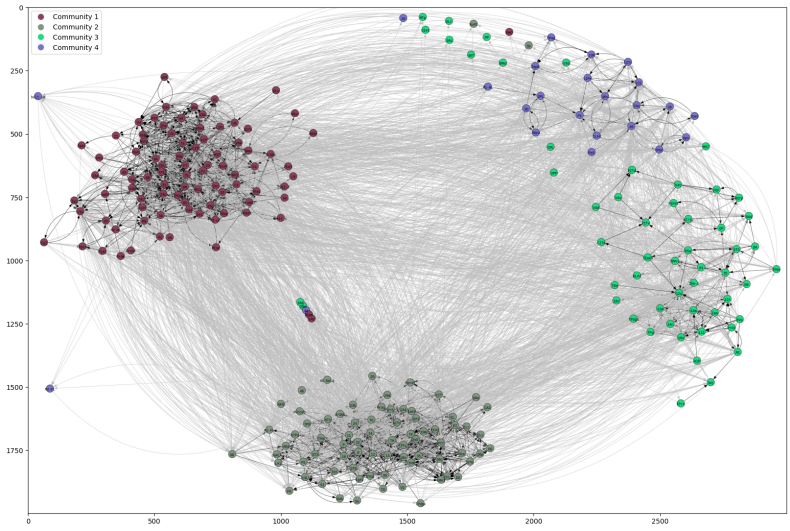

Figure 2.6: Graphical representation of the modules detected by the Louvain algorithm on the Macaque rhesus brain graph.

This partition yields a modularity of 0.3295535655573555.

2.3.4 Weighted directed network

As weighted directed graph the US Congress Twitter graph was selected: the network represents the Twitter interaction network for the 117*th* United States Congress, both House of Representatives and Senate. The base data was collected via the Twitter's API, then the empirical transmission probabilities were quantified according to the fraction of times one member retweeted, quote tweeted, replied to, or mentioned another member's tweet. These transmission probabilities represent the weights' edges. The network is composed by 475 vertexes and 4090 edges. The data was collected from [18].

The algorithm detected 4 communities, where the first one is made by 201 noes, the second one by 163, the third by 84 and the last by 27.

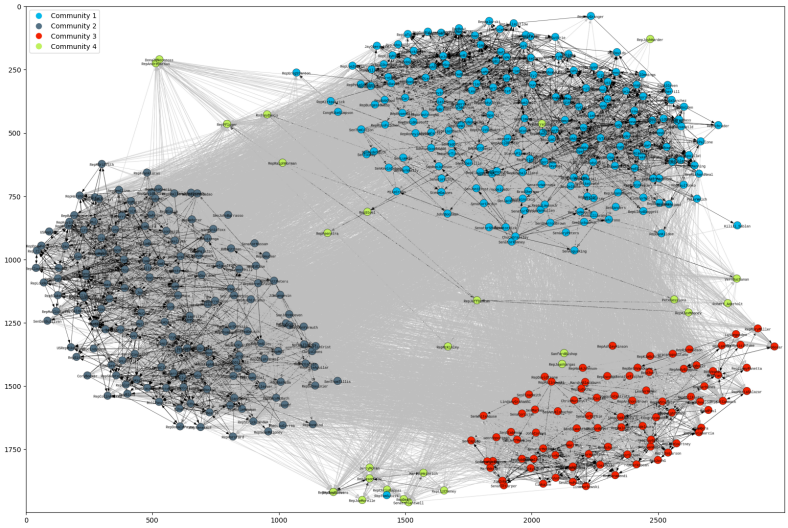

Figure 2.7: Graphical representation of the modules detected by the Louvain algorithm on the US Congress Twitter network.

This partition yields a modularity of 0.4399847144924596.

3
Infomap Algorithm

Infomap is a community detection algorithm and its aim is to mainimize the map equation, which leans on information theory to give a quality measure of a network's partition. The material is taken from [20], [21], [22], [23], [24], [25], [26], [27] and [28].

3.1 Huffman coding procedure

Definition 3.1.1 *A flow-based algorithm for community detection is an algorithm which aim is to model the flow of information, influence, or entities within a network to identify communities of nodes that have strong connections internally but weaker connections with nodes outside the community.*

One example of such algorithm is Infomap. The main idea behind the Infomap algorithm is a community detection algorithm that uses community partitions of the graph as a Huffman code that compresses the information about a random walker exploring the graph.
The first thing we need to do is introduce the Huffman coding procedure.

The central object of this procedure is a random walker exploring the network: just like it is possible to do a random walk on $\mathbb{Z}$ or on a subset, as we've seen in Example 1.2.1, it is also possible to do the same thing on a graph.

Definition 3.1.2 *A random walk on $G = (V, E)$ is the following sequence of moves of the process that starts in some initial node v_0 and then a neighbor v_1 of v_0, is chosen, randomly and*

independently and then the process moves from v_0 to v_1. We repeat this process until for some reasons, the process ends, if that doesn't happen, the random walk can go on forever.

Observation 3.1.1 *If the graph is unweighted, at each step, all the neighboring nodes all have the same probability of being chosen. For weighted graphs, the probability depends on the values of the assigned weights.*

In any case, the transition matrix for a random walk through a graph is computed in the following way:

Algorithm 3.1 Computation of the transition matrix P for a random walk through a graph

Adjacency matrix A
$d = A^T 1$
$P = A\, diag^{-1}(d)$

If A is a $n \times n$ matrix, then 1 is the vector $(1, \ldots, 1)^T$, where all the n entries are equal to 1, while $diag^{-1}(d) = \begin{bmatrix} \frac{1}{d_1} & 0 & \ldots & 0 \\ 0 & \frac{1}{d_2} & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & 0 & \frac{1}{d_n} \end{bmatrix}$, where d_i is the i-th entry of the vector d.

The probability that the walker transitions between two nodes given by the Markov transition matrix P.

In most real-world networks, there are regions of the network such that once the random walker enters a region, it tends to stay there for a long time, and movements between the regions are relatively rare. This allows us to combinatorially combine codewords into Huffman codes: we can use a prefix code for each region, and then use a unique codeword for each node within a community.

Definition 3.1.3 *A codeword is a label representing a nodes or a community of the network, common choices of codewords are sequences of* 0 *or* 1 *bits.*

Observation 3.1.2 *It is possible to reuse node level codewords for each community, this means that two nodes can have the same codeword, as long as they belong to two different communities.*

Definition 3.1.4 *The set of the region prefixes form the master codebook.*
The set of node codewords form the module codebook.

Remark 3.1.1 *A straightforward method of assigning codewords to nodes is using short codewords to common events or objects, and long codewords to rare ones. In the case of a random walk, we assign shorter codewords to nodes that are visited a lot of times and longer to nodes that are never or rarely visited.*

Let's look at some examples.

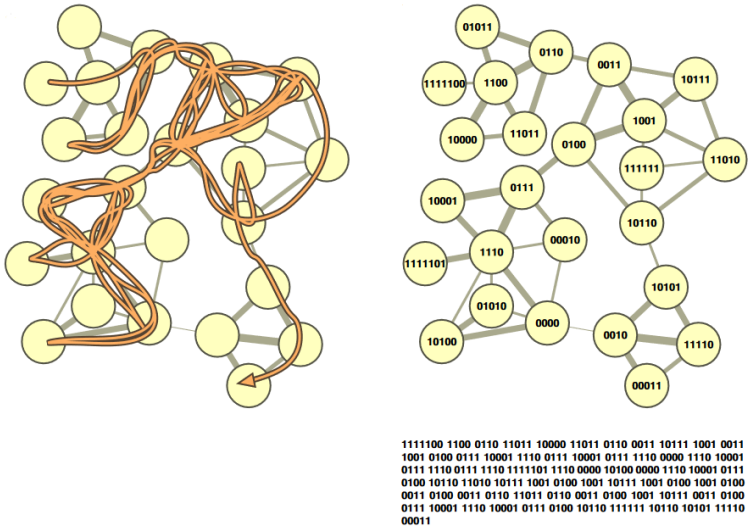

Figure 3.1: The orange line in the first picture shows one sample trajectory. In this example the network is represented as one big community, which means that the node codewords can't be repeated and as we can see, each node in the network is given a unique name. The bits shown under the network in the second picture describe the sample trajectory, starting with 1111100 for the first node on the walk in the upper left corner, 1100 for the second node, etc., and ending with 00011 for the last node on the walk in the lower right corner. In this case we are able to represent a 71-step random walk in 314 bits. Picture taken from [21].

Let's now look at a second partition of the same network.

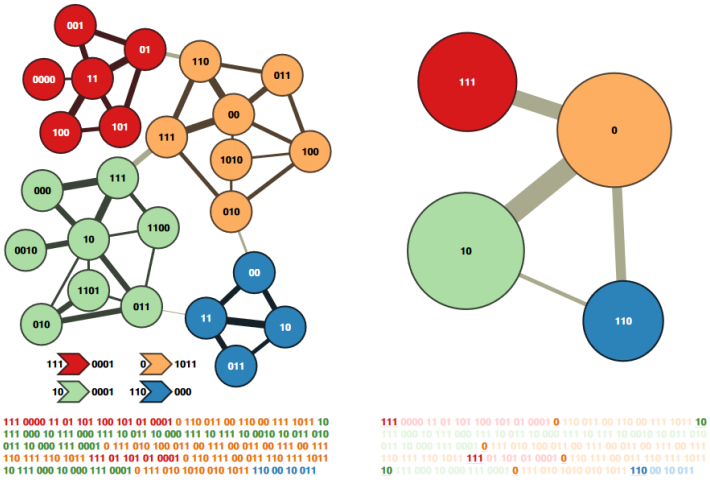

Figure 3.2: In this second partition of the same network, we can observe two-level description of the random walk, in which major clusters receive unique names, but the names of nodes within clusters are reused, this technique gives us on average a much shorter description for this network, in fact we can describe the walk by the 243 bits shown under the network, versus the 314 in the previous example. The codes naming the modules and the codes used to indicate an exit from each module are shown to the left and the right of the arrows under the network, respectively. The first three bits 111 represent the region prefix and they indicate that the walk begins in the red community, the code 0000 specifies the first node on the walk, etc. In the second picture we can see reported only the module names, and not the locations within the modules. Picture taken from [21].

These two examples show how the clustering of the network influence the number of bits needed to describe a random walk.

Definition 3.1.5 *The average number of bits used to describe each step of the random walk is called known as the average length encoding and it's denoted as $L_H(M(m))$, where M is a specific network partition and m the number of communities of such partition.*

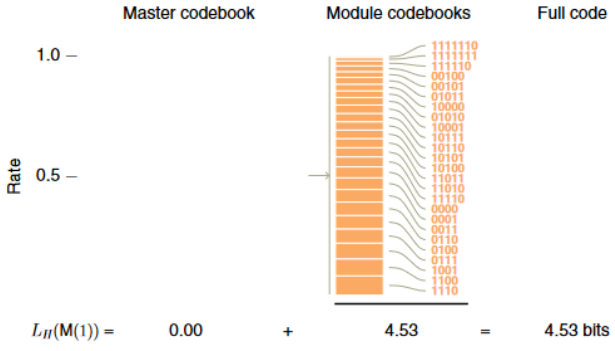

Figure 3.3: Average length encoding for the partition of the network in Figure 3.1. Picture taken from [22].

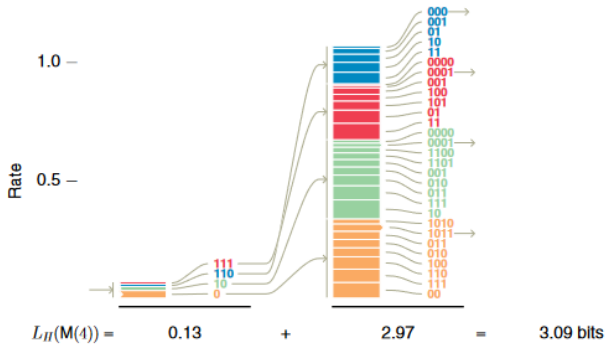

Figure 3.4: Average length encoding for the partition of the network in Figure 3.2. Picture taken from [22].

Observation 3.1.3 *We can clearly see that, out of the two partitions of the network, the second one is the optimal one: the four module codebooks are associated with smaller sets of nodes and shorter codewords, but still with long persistence time and few between-module movements. This compensates for the extra cost of switching modules and accessing the index codebook.*

When we use too few modules, we are effectively still back at the level of using an individual codeword for every node. On the other hand, when we use too many communities, the number of prefix codes becomes too large. So we need to find an optimal partition that assigns nodes to modules such that the information needed to compress the movement of our random walkers is minimized.

A very important result when it comes the representation of random walks is the Shannon's source coding theorem.

Theorem 3.1.1 (Shannon's source coding theorem) *Given n independent identically distributed random variables each with entropy $H(X)$ can be compressed into more than $nH(X)$ bits with negligible risk of information loss, as $n \longrightarrow +\infty$, but if they are compressed into fewer than $nH(X)$ bits it is very likely that information will be lost, this means that the probability that the original bits won't be recoverable from the binary bits is close to* 1.

Corollary 3.1.1 *This theorem states is that if we use n codewords to describe the n states of a random variable X that occur with frequencies p_i, the average length of a codeword can be no less than the entropy of the random variable X itself: $H(X) = -\sum p_i log(p_i)$ and that the average number of bits needed to describe a single step in the random walk is bounded below by the entropy $H(P)$, where P is the distribution of visit frequencies to the nodes on the network.*

Our goal is to find an optimal partition that assigns nodes to communities such that the information needed to compress the movement of our random walkers is minimized, in particular what we need to minimize the map equation, which is, as we are going to see in the next section, a concept that is strictly related to the average length encoding. This is when the minimzation algorithm comes into place.

3.2 The map equation as quality measure of a network's partition

The map equation takes advantage of the duality between finding community structure in networks and minimizing the description length of a random walker's movements on a network, which is why it's a good measure of how well a given network partition captures community structures within the network.

The goal of the Infomap algorithm is going to be minimizing the map equation over all possible partitions of the network.

Remark 3.2.1 *The random walk over a graph is a good approximation of the actual flow of information through the network, which is why the map equation measures the per-step theoretical lower limit of a modular description of a random walker on a network.*

Instead of measuring the average length encoding of a long walk and dividing by the number of steps, it is more efficient to derive the codelength from the stationary distribution of the random walker on the nodes and links.

From now on we are going to assume graph to be finite, now thanks to Proposition 1.2.1, the stationary distribution exists and it's given by the following Theorem.

Theorem 3.2.1 *The stationary distribution of a random walk over a graph G is given by:*

$$p_i = \sum_k p_k p_{ik} \tag{3.1}$$

Where the transition probabilities p_{ij} are computed in Algorithm 3.1

However, the solution given by the system in Equation 3.1 is not necessarily unique, to ensure a unique solution independent of where the random walker starts we need to introduce the concept of random walks with teleportation.

Definition 3.2.1 *A random walk through a graph is a Markov chain that works in the same way as in Definition 3.1.2, with the exception that with probability τ the random walker instead of moving to a neighboring node, it teleports to a random node.*

The transition matrix P is now given by:

Algorithm 3.2 Computation of the transition matrix P' for a random walk through a graph with teleportation

Adjacency matrix A
$d = A^T 1$
$P = A\, diag^{-1}(d)$
$P' = (1 - \tau)M + \tau q 1^T$

Where each entry q_i represents the probability that the random walker teleports to the node i

By introducing the teleportation parameter τ we add a small probability for the random walker to jump to any node in the network, regardless of the connectivity, this implies that now there is a path between every pair of nodes, making the transition matrix irreducible.

Now if we recall Proposition 1.2.2, we get that the stationary distribution not only exists, but it's also unique, so independent of the initial state.

Remark 3.2.2 *A typical value of the teleportation rate is $\tau = 0.15$, but in practice the clustering results show only small changes for teleportation rates in the range $\tau \in (0.05,\ 0.95)$.*

Observation 3.2.1 *In undirected networks the results are completely independent of the teleportation rate and identical to results given by Equation 3.1. For directed networks, a teleportation rate too close to 0 gives results that depend on how the random walker was initiated and should be avoided, but a teleportation value equal to 1 corresponds to using the link weights as the stationary distribution.*

For more robust results that depend less on the teleportation parameter τ, we most often use teleportation to a node proportional to the total weights of the links to the node.

Theorem 3.2.2 *The stationary distribution for a random walk through a graph with teleportation is given by*

$$p_i = \sum_j p_j p'_{ij} = (1 - \tau) \sum_j p_j p_{ji} + \tau \frac{\sum_j a_{ji}}{\sum_{k,j} a_{jk}} \tag{3.2}$$

Where p'_{ij} are computed in Algorithm 3.2 and a_{ij} are the entries of the original adjacency matrix A.

To make the results even more robust and independent from the teleportation parameter τ we a new kind of teleportation.

Definition 3.2.2 *Unrecorded teleportation is a type of teleportation scheme in which we do not explicitly track or record the specific nodes visited during teleportation events. Instead, only the fact that teleportation occurred is noted, but the individual steps taken during teleportation are not recorded and that the transitions between nodes only occurs when the walker moves along edges.*

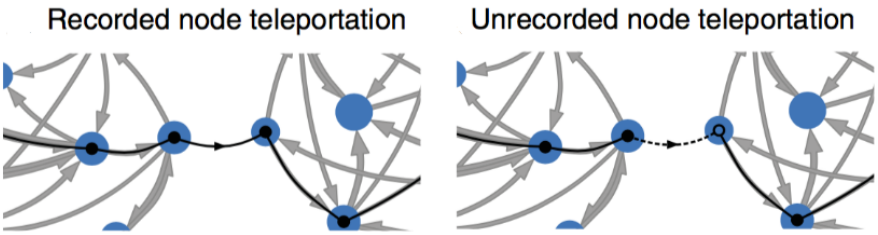

Figure 3.5: Example of recorded and unrecorded teleportation. Picture taken from [23].

Observation 3.2.2 *By only keeping tack on the movements along the edges, unrecorded teleportation reduces the noise and focuses more on the network structure.*

We capture these dynamics with an extra step without teleportation on a stationary solution similar to Equation 3.2, but with teleportation to a node proportional to the total weights of the links from the node, rather than being proportional to the total weights of the links into the node, like in Equation 3.2.

Theorem 3.2.3 *The stationary distribution for a random walk through a graph with unrecorded teleportation is given by:*

$$p_i^* = (1 - \tau) \sum_j p_j^* p_{ji} + \tau \frac{\sum_j a_{ij}}{\sum_{k,j} a_{jk}}$$

Where p_{ij} are computed in Algorithm 3.1 and a_{ij} are the entries of the original adjacency matrix A.

Definition 3.2.3 *The unrecorded visit rates on links q_{ji} and nodes p_i are defined as:*

$$q_{ji} = p_j^* p_{ji} \quad and \quad p_i = \sum_j q_{ji}$$

In the previous section we have seen the Huffman codes, which are optimal in the sense that there are no other binary codes as close to the theoretical limit given by the Theorem 3.1.1. However, for identifying the optimal partition of the network, we are only interested in the compression rate and not the actual codewords, which is why the Infomap algorithm only measures the theoretical limit given by the map equation.

Observation 3.2.3 *To take advantage of the modular structure of the network, the map equation uses the extra constraint that the only available information from one step to the next is the currently visited module, or that the random walk switches between modules, forcing independent and identically distributed events within and between modules.*

The map equation can be expressed in closed form by invoking Theorem 3.1.1 in for each of multiple codebooks, and by weighting them by their rate of use.

We can now define the module-transition rates at which the random walker enters and exits each module.

Definition 3.2.4

- *The probability of entering a module α is given by $q_{\alpha\curvearrowright} = \sum_{i\in\beta\neq\alpha, j\in\alpha} q_{ij}$*
- *The probability of exiting a module α is given by $q_{\alpha\curvearrowleft} = \sum_{i\in\alpha, j\in\beta\neq\alpha} q_{ij}$*

Where q_{ij} are the unrecorded visit rates given in Definition 3.2.3.

We know have all the objects necessary to define the map equation.

Definition 3.2.5 *The map equation is given by:*

$$L(M) = q_{\curvearrowright} H(Q) + \sum_{\alpha=1}^{m} p_{\alpha\circlearrowleft} H(P^{\alpha}) \tag{3.3}$$

Let's explain each element in the equation:

- *L(M) is the per-step description length for module partition M. That is, for module partition M of n nodes into m modules, the lower bound of the average length of the code describing a step of the random walker.*

- $q_{\curvearrowleft} = \sum_{\alpha=1}^{m} q_{\alpha\curvearrowleft}$ *is the rate at which the index codebook is used. The per-step use rate of the index codebook is given by the total probability that the random walker enters any of the m communities.*

- $H(Q) = \sum_{\alpha=1}^{m} \frac{q_{\alpha\curvearrowleft}}{q_{\curvearrowleft}} log(\frac{q_{\alpha\curvearrowleft}}{q_{\curvearrowleft}})$ *is the frequency-weighted average length of codewords in the index codebook. The entropy of the relative rates to use the module codebooks measures the smallest average codeword length that is theoretically possible.*

- $p_{\alpha\circlearrowleft} = \sum_{i\in\alpha} p_i + q_{\alpha\curvearrowright}$ *is the rate at which the module codebook α is used, which is given by the total probability that any node in the module is visited, plus the probability that the random walker exits the module and the exit codeword is used.*

- $H(P^{\alpha}) = -\frac{q_{\alpha\curvearrowright}}{p_{\alpha\circlearrowleft}} log(\frac{q_{\alpha\curvearrowright}}{p_{\alpha\circlearrowleft}}) - \sum_{i\in\alpha} \frac{p_i}{p_{\alpha\circlearrowleft}} log(\frac{p_i}{p_{\alpha\circlearrowleft}})$ *is the frequency-weighted average length of codewords in module codebook α. The entropy of the relative rates at which the random walker exits module α and visits each node in module α measures the smallest average codeword length that is theoretically possible.*

Observation 3.2.4 *As we can see, unlike in Theorem 3.1.1, we are now able to describe the per-step description length of a random walk through a graph by taking advantage of the community structure of the network.*

3.3 Infomap's functioning and pseudo-code

3.3.1 Two-level algorithm

The goal of the Infomap algorithm is to optimize the partition of the graph in a way that would generally favor shorter random walk descriptions. The algorithm doesn't target a specific random walk, but the resulting community structure should make it more likely that a random walk description through the graph would be shorter on average.

Infomap works very similarly to the Louvain method: neighboring nodes are joined into modules, which subsequently are joined into supermodules and so on. A supermodule is a collection of communities that are strongly interconnected or have significant flow of information between them.

Algorithm 3.3 Infomap algorithm

input Network G = (V, E), where V = set of n vertices, E = set of edges. Minimum quality improvement threshold ε
Calculate the node visit rate for each node $v \in V$
$M = \{m_i = \{v_i\} \mid v_i \in V\}$
Compute $L = L(M)$ in Equation 3.3
repeat
  $L_{prev} = L$
  R = random sequence of integers 1 to n
  for i=0 to n - 1
    $m_{new} = bestNewModule(M, v_{R[i]})$
    Move $v_{R[i]}$ to m_{new} module
    Update M
    Update L
  end for
until $L_{prev} - L < \varepsilon$
return M

First, each node is assigned to its own community. Then, in random sequential order, each node is moved to the neighboring module that results in the largest decrease of the map equation. If no move results in a decrease of the map equation, the node stays in its original module. This procedure is repeated, each time in a new random sequential order, until no move generates a decrease of the map equation. Then the network is rebuilt by creating a new network where the detected communities become the new nodes, once the network is rebuilt with communities as nodes, the algorithm repeats the process of identifying communities or modules. This process continues iteratively, with each iteration producing a new level of the hierarchical structure. At each level, the communities detected in the previous level become the nodes for the next level, and the process of identifying communities within these nodes is repeated.

With this algorithm, a fairly good clustering of the network can be found in very few iterations. It is also possible to improve the algorithm by making sure that the nodes assigned to the same module are forced to move jointly when the network is rebuilt. As a result, what was an optimal move early in the algorithm might have the opposite effect later in the algorithm. Two or more modules that merge together and form one single module when the network is rebuilt can never be separated again in this algorithm. Therefore, the accuracy can be improved

by breaking the modules of the final state of the core algorithm in either of the two following ways:

- Submodule movements: first, each cluster is treated as a network on its own and the main algorithm is applied to this network. This procedure generates one or more submodules for each module. Then all submodules are moved back to their respective modules of the previous step. At this stage, with the same partition as in the previous step but with each submodule being freely movable between the modules, the main algorithm is reapplied.

- Single-node movements: first, each node is re-assigned to be the sole member of its own module, in order to allow for single- node movements. Then all nodes are moved back to their respective modules of the previous step. At this stage, with the same partition as in the previous step but with each single node being freely movable between the modules, the main algorithm is reapplied.

In practice, we repeat the two extensions to the core algorithm in sequence and as long as the clustering is improved. Moreover, we apply the submodule movements recursively. That is, to find the submodules to be moved, the algorithm first splits the submodules into subsubmodules and so on until no further splits are possible. Since the algorithm is stochastic, we can restart the algorithm from scratch every time the clustering cannot be improved further and the algorithm stops. By repeating the search more than once the final partition is less likely to correspond to a local minimum. For each iteration, we keep track of the clustering if the description length is shorter than the shortest description length recorded before.

3.3.2 Multilevel algorithm

In the two-level algorithm the network is partitioned into only two levels of organization: communities and nodes. It is possible to generalize our search algorithm for the two-level map equation to recursively search for multilevel solutions. The recursive search operates on a module at any level; this can be all the nodes in the entire network, or a few nodes at the finest level. For a given module, the algorithm first generates submodules if this gives a shorter description length. If not, the recursive search does not go further down this branch. But if adding submodules gives a shorter description length, the algorithm tests if movements within the module can be further compressed by additional index codebooks. To test for all combinations, the al-

gorithm calls itself recursively, both operating on the network formed by the submodules and on the networks formed by the nodes within every submodule.

In this way, the algorithm successively increases and decreases the depth of different branches of the multilevel code structure in its search for the optimal hierarchical partitioning. For every split of a module into submodules, we use the two-level search algorithm described above.

3.3.3 Infomap's complexity and limitations

Infomap handles both unweighted and weighted, undirected and directed links, it also can identify overlapping communities, which means that one particular node can belong to multiple modules.

Remark 3.3.1 (Scalability of Infomap) *Infomap is a scalable algorithm for the following reasons:*

- *This hierarchical approach of clustering used by the algorithm allows Infomap to handle large networks because it can reduce the size of the problem by aggregating nodes into modules*

- *Since the goal of the algorithm is to represent the network in the most compact way, it is able to store information about the network structure without requiring excessive memory resources.*

- *Infomap can be parallelized, which makes sure that the algorithm is able to process large-scale networks more efficiently.*

Remark 3.3.2 *Infomap is able to recognize the random networks described in Example 1.1.1. This means that the algorithm is capable of distinguishing between networks that have clear community structures and networks that exhibit a random structure, hence lacking a clear community organization, just like we stated in Proposition 1.1.2.*

Remark 3.3.3 (Resolution of Infomap) *Infomap is that suffers from a sort of resolution limit, this means that when a network exhibits strong community structure at a small scale, some algorithms may fail to detect these smaller communities accurately, instead merging them into larger communities.*

For Infomap, this resolution limit happens because the algorithm's optimization process is primarily focused on optimizing the description length of the overall network structure. As a result, in networks with where there are many strongly connected communities, Infomap may tend to merge smaller communities into larger ones in order to achieve better overall compression of information flow.

This can be problematic if the goal is to accurately identify and characterize communities at all scales within the network. In such cases, the resolution limit of Infomap means that it may not be able to capture the full extent of the community structure, particularly at finer scales.

3.4 Application to four different types of networks

In this section we are going to see different applications of the Infomap algorithm on different types of graphs, more specifically, we are going to apply it to the same networks introduced in Section 2.3 in order to compare the results.

3.4.1 Unweighted undirected network

For the Jazz musicians network, 5 communities were detected by Infomap, where the first one is made by 135 nodes, the second one by 55, the third by 2, the fourth by 4 and the last by 2.

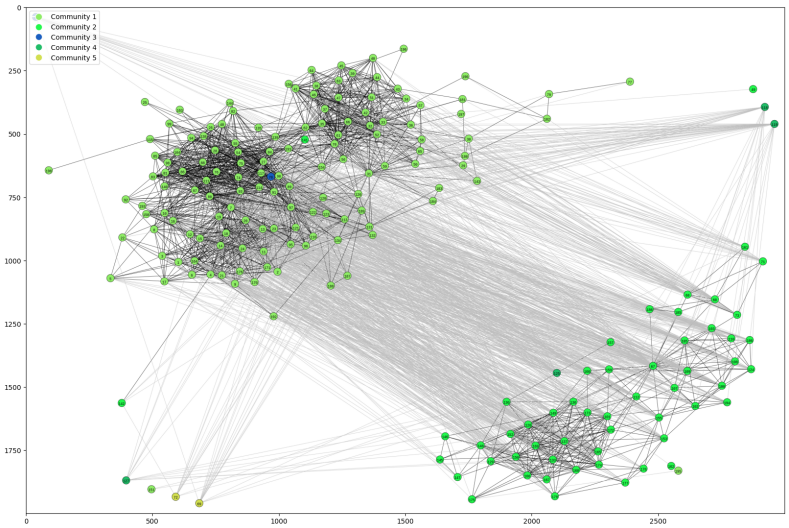

Figure 3.6: Graphical representation of the modules detected by the Infomap algorithm on the Jazz musicians network.

We are now going to compare the results obtained with the Infomap algorithm to the ones obtained with Louvain.

Algorithm	Number of modules	Modularity	Map equation
Louvain	4	0.43890781537538287	11.761273616050756
Infomap	5	0.038870920032069954	6.910117006606936

Table 3.1: Results of the Infomap and Louvain algorithms on the Jazz musicians network.

We can observe that for the number of clusters detected is stable for both algorithms and as expected, Louvain yields the highest modularity value, while Infomap yields the lowest value for the map equation.

3.4.2 Weighted undirected network

For the Windsurfers network, 3 communities were detected by Infomap, where the first one has 22 nodes, the second one 19 and the third one 2.

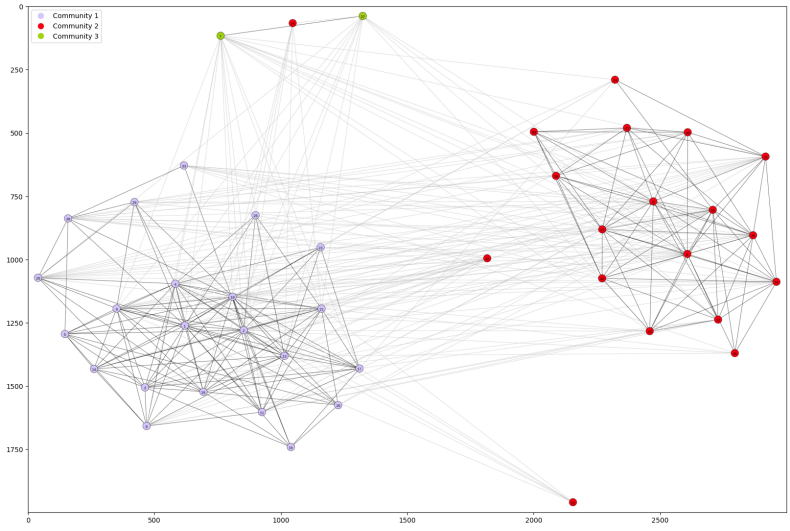

Figure 3.7: Graphical representation of the modules detected by the Infomap algorithm on the Windsurfers network.

We are now going to compare the results obtained with the Infomap algorithm to the ones obtained with Louvain.

Algorithm	Number of modules	Modularity	Map equation
Louvain	2	0.37121605900844046	5.535010448869063
Infomap	3	0.20192741378788756	4.5052902422623315

Table 3.2: Results of the Infomap and Louvain algorithms on the Windsurfers network.

We can observe that for the number of communities detected by the algorithms is similar and once again, from Louvain to Infomap we can observe a decrease for both the and the modularity value and the map equation.

3.4.3 Unweighted directed network

For the Macaque rhesus brain graph, Infomap detected 6 clusters made by 94, 25, 33, 40, 46 and 4 vertices each.

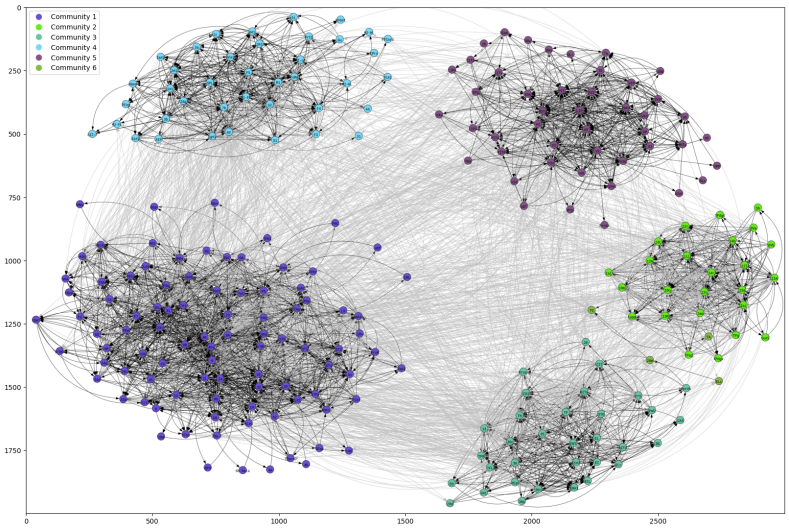

Figure 3.8: Graphical representation of the modules detected by the Infomap algorithm on the Macaque rhesus brain graph.

We are now going to compare the results obtained with the Infomap algorithm to the ones obtained with Louvain.

Algorithm	Number of modules	Modularity	Map equation
Louvain	4	0.3295535655573555	11.610916375907639
Infomap	6	0.30617633953173806	6.999063756957538

Table 3.3: Results of the Infomap and Louvain algorithms on the Macaque rhesus brain graph.

Again, the numbers of communities are close and, from Louvain to Infomap, we can observe a decrease for both the and the modularity value and the map equation, although the decrease in map equation is not as large as in the other networks.

3.4.4 Weighted directed network

For the US Congress Twitter network, Infomap detected 15 clusters with the following sizes.

Cluster	Number of nodes
1	165
2	200
3	48
4	11
5	16
6	10
7	4
8	4
9	5
10	4
11	3
12	2
13	1
14	1
15	1

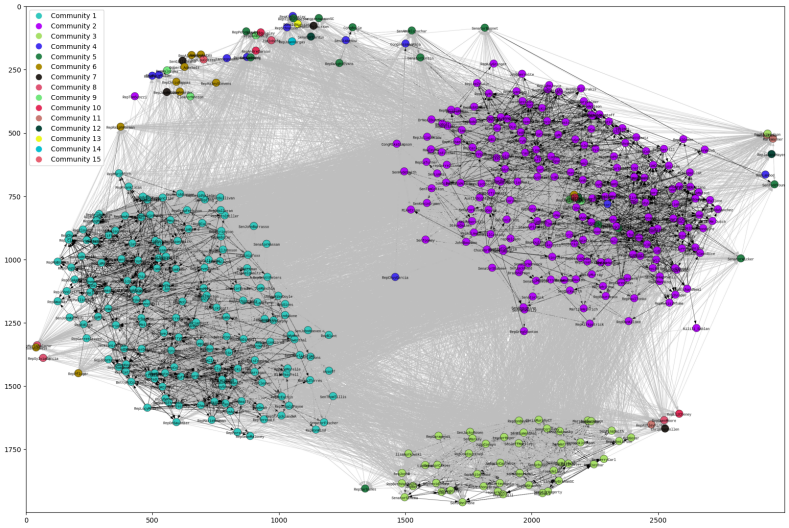

Figure 3.9: Graphical representation of the modules detected by the Infomap algorithm on the US Congress Twitter network.

We are now going to compare the results obtained with the Infomap algorithm to the ones obtained with Louvain.

Algorithm	Number of modules	Modularity	Map equation
Louvain	4	0.4399847144924596	11.614856746923584
Infomap	15	0.007693580298894281	7.6239123302314455

Table 3.4: Results of the Infomap and Louvain algorithms on the US Congress Twitter network.

In this case the difference between the number of modules detected by the two algorithms is very large, also the decrease in modularity from Louvain to Infomap is considerably larger than with the previous networks, while the values for the map equation are in line with the results obtained in the other networks.

4

Girvan–Newman algorithm

The Girvan-Newman algorithm is a hierarchical method used to detect communities in networks. The material was taken from [31], [32], [33], [34], [35], [35], [36] and [37].

4.1 Hierarchical community detection methods

Definition 4.1.1 *A hierarchical community detection algorithm (or hierarchical clustering) is a method used to detect communities in a network by building a hierarchy (or tree structure) of communities, either by iteratively merging smaller communities into larger ones (agglomerative approach) or by recursively splitting larger communities into smaller ones (divisive approach).*

The first thing we need to do in these type of methods is calculating a weight W_{ij} for every pair i, j of vertices in the network, which represents how closely connected the vertices are.

Observation 4.1.1 *Note that the weight between two vertices i, j is not necessarily the weight of the adjacency matrix.*

Then we take the n vertices in the network, with no edges between them, and add edges between pairs one by one in order of their weights, starting with the pair with the strongest weight and progressing to the weakest.

As edges are added, the resulting graph shows a nested set of increasingly large components, which are represent the communities. The communities can be represented by using a tree in

which the lowest level at which two vertices are connected represents the strength of the edge that resulted in their first becoming members of the same community. A "slice" through this tree at any level gives the communities that existed just before an edge of the corresponding weight was added, this type of trees are called dendrograms.

Definition 4.1.2 *A dendrogram is a diagram representing a tree. In hierarchical clustering, it illustrates the arrangement of the clusters produced by the corresponding analyses.*

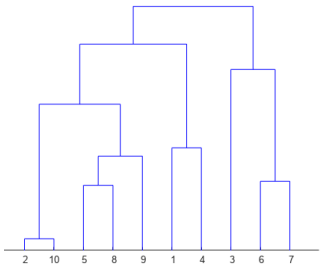

Figure 4.1: Example of a hierarchical clustering tree (or dendrogram). The numbers at the bottom represent the vertices in the network, and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs. In this example the communities identified in the last "slice" are $\{2,10\}$, $\{5,8\}$, $\{9,1\}$, $\{4\}$, $\{3\}$, $\{6,7\}$. Picture taken from [30].

Algorithm 4.1 Hierarchical clustering

input Network G = (V, E), where V = set of n vertices, E = set of edges.
for i, j = 0 to n - 1
  Calculate a weight W_{ij} for every pair of vertices
end for
Create an empty graph G' with the same set of vertices as G and no edges
Sort all pairs (i, j) of vertices by their weights W_{ij} in descending order
for i, j = 0 to n - 1
  Add edge (i, j) to the graph G'
end for
return Network G' = (V, E')

One possible definition of the weight is the number of node-independent paths between vertices.

Definition 4.1.3 *Two paths that connect the same pair of vertices are said to be node-independent if they share none of the same vertices other than their initial and final vertices. One can similarly also count edge-independent paths.*

Observation 4.1.2 *The number of node-independent (edge-independent) paths between two vertices i and j in a graph is equal to the minimum number of vertices (or edges) that must be removed from the graph to disconnect i and j from one another. Thus these numbers are in a sense a measure of the robustness of the network to deletion of nodes (or edges).*

Another possible way to define weights between vertices is to count the total number of paths that run between them. However, because the number of paths between any two vertices is infinite (unless it is zero), one typically weights paths of length l by a factor α^l, with α small, so that the weighted count of the number of paths converges. Thus long paths contribute exponentially less weight than those that are short.

Definition 4.1.4 *This definition of weight is given by:*

$$W = \sum_{l=0}^{\infty} (\alpha A)^l = (\boldsymbol{I} - \alpha A)^{-1}$$

Proposition 4.1.1 *For the sum to converge, we must choose α smaller than the reciprocal of the largest eigenvalue of A.*

PROOF For the series to converge, the spectral radius $\rho(\alpha A)$ must be less than 1. This means $max_i |\alpha\lambda_i|$, therefore $|\alpha| < \frac{1}{|max_i \lambda_i|}$.
The largest eigenvalue in absolute value of A is $\lambda_{max} = \max_i |\lambda_i|$ Thus, we must have $|\alpha| < \frac{1}{\lambda_{max}}$. □

Remark 4.1.1 *Both definitions of weight have a tendency to separate single peripheral vertices from the communities to which they should rightly belong. This means that if a vertex is, for example, connected to the rest of a network by only a single edge then, to the extent that it belongs to any community, it should clearly be considered to belong to the community at the other end of that edge. Both the numbers of independent paths and the weighted path counts for such vertices are small and hence single nodes often remain isolated from the network when the communities are constructed.*

4.2 Edge and vertex betweenness centrality measures

One way to solve this problem is by introducing a new centrality measure called betweenness centrality. Before doing that we need to introduce the concept of shortest path in a graph.

Definition 4.2.1 *A path in a graph is a finite or infinite sequence of edges which joins a sequence of vertices which are all distinct.*

Observation 4.2.1 *A path in an graph is a sequence of vertices* $P = (v_1, v_2, \ldots, v_n)$ *such that* v_i *is adjacent to* v_{i+1} *(or there is a directed edge from* v_i *to* v_{i+1} *if we are dealing with a directed graph) for* $1 \leq i \leq n$. *Such a path P is called a path of length n* 1 *from* v_1 *to* v_n.
Let $E = \{e_{ij}\}$ *where* e_{ij} *is the edge incident to both* v_i *and* v_j. *Given a real-valued weight function* $f : E \longrightarrow \mathbb{R}$, *and a graph G, the shortest path from v to v' is the path P that over all possible n minimizes the sum* $\sum_{i=1}^{n-1} f(e_{i,i+1})$. *When each edge in the graph has unit weight or* $f : E \longrightarrow \{1\}$, *this is equivalent to finding the path with fewest edges.*

Observation 4.2.2 *A path can be either undirected or directed, in the latter case we also call it a dipath.*

Definition 4.2.2 *The shortest path between two nodes is a path such that the sum of the weights of its constituent edges is minimized.*

Definition 4.2.3 *Betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.*
The betweenness centrality of a node v in an unweighted graph is given by the expression:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where $\sigma_{st}(v)$ *is the total number of shortest paths from node s to node t and* σ_{st} *is the number of those paths that pass through v (not where v is an end point).*

Observation 4.2.3 *Note that the betweenness centrality of a node scales with the number of pairs of nodes as suggested by the summation indices. Therefore, the calculation may be rescaled by*

dividing through by the number of pairs of nodes not including v, so that $g \in [0, 1]$.
The division is done by the total number of shortest paths between all pairs of nodes, so $(n-1)(n-2)$ *for directed graphs and* $\frac{(n-1)(n-2)}{2}$ *for undirected graphs, where n is the number of nodes in the network. Note that this scales for the highest possible value, where one node is crossed by every single shortest path. This is often not the case, and a normalization can be performed without a loss of precision:* $g^{normalized}(v) = \frac{g(v) - min(g)}{max(g) - min(g)}$. *Which results in* $max(g^{normalized}(v)) = 1$ *and* $min(g^{normalized}(v)) = 0$. *Note that this will always be a scaling from a smaller range into a larger range, so no precision is lost.*

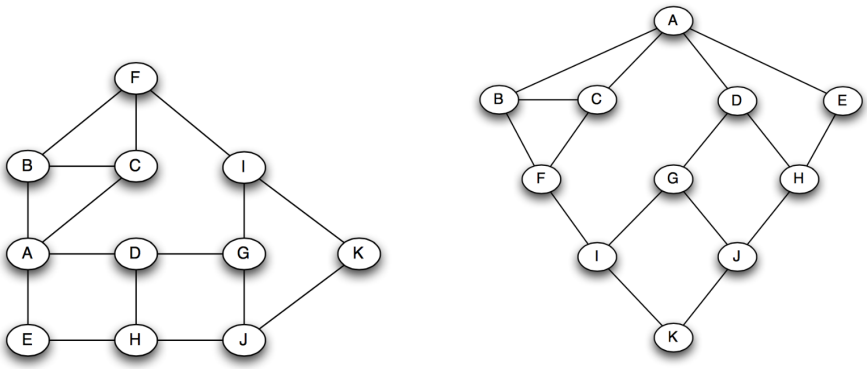

Figure 4.2: Calculation of betweenness centrality of node A. In the first phase we use breadth-first search to find all the shortest paths between A and all other nodes. Picture taken from [33].

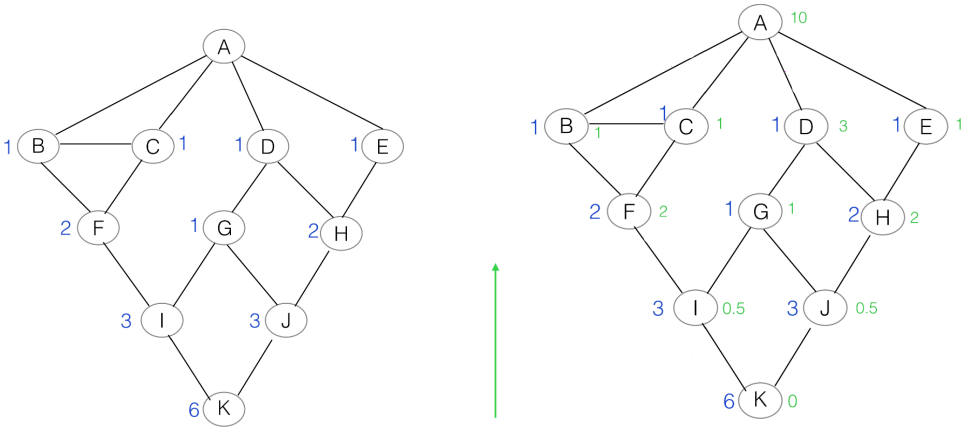

Figure 4.3: Calculation of betweenness centrality of node A. For each node s we calculate the number of shortest paths between s and A $\sigma(s, A)$. Then calculate $\delta(s \mid A)$, the dependency of s on A, we then repeat all the steps for each node. Picture taken from [33].

Algorithm 4.2 Brandes' algorithm for the computation of the vertex betweenness centrality in an unweighted network

input Unweighted network G = (V, E), where V = set of n vertices, E = set of edges
Betweenness centrality $C_B(v) = 0,\ v \in V$
Inizialize S as an empty stack and Q as an empty queue
for $s \in V$
  Initialize the list of predecessors of shortests paths from s: P(w) is an empty list for all $w \in V$
  Initialize the list of shortest paths from s to every other node: $\sigma(t) = 0,\ t \in V,\ \sigma(s) = 1$
  Initialize the distance of each node from s: $d(t) = +\infty,\ t \in V,\ d(s) = 0$
  Enqueue s in Q
  while $Q \neq \varnothing$
    Dequeue v from Q, Push v in S
    for $v' \in N(v) = \{v' \mid v' \text{ is a neighbor of v}\}$
      if $d(v') = +\infty$
        Enqeue v' in Q
        $d(v') = d(v) + 1$
      else
        $\sigma(v') = \sigma(v') + \sigma(v)$
        Append v to P(v')
      end if
    end for
  end while
  $\delta(v) = 0,\ v \in V$
  while $S \neq \varnothing$
    Pop v' from S
    for $v \in P(v')$
      $\delta(v) = \delta(v) + \frac{\sigma(v)(1+\delta(v'))}{\sigma(v')}$
    end for
    if $v' \neq s$
      $C_B(v') = C_B(v') + \delta(v')$
    end if
  end while
end for
for $v \in V$
  $C_B(v) = \frac{C_B(v)}{(n-1)(n-2)}$ #for undirected networks
  $C_B(v) = 2\frac{C_B(v)}{(n-1)(n-2)}$ #for directed networks
end for
return $C_B(v)$

Algorithm 4.3 Computation of the vertex betweenness centrality in a weighted network

input Weighted network G = (V, E), where V = set of n vertices, E = set of edges
Betweenness centrality $C_B(v) = 0,\ v \in V$
Inizialize S as an empty stack and Q as an empty priority queue
for $s \in V$
Initialize the list of predecessors of shortests paths from s: P(w) is an empty list for all $w \in V$
Initialize the list of shortest paths from s to every other node: $\sigma(t) = 0,\ t \in V,\ \sigma(s) = 1$
Initialize the distance of each node from s: $d(t) = +\infty,\ t \in V,\ d(s) = 0$
Enqueue (s, 0) in Q
while $Q \neq \varnothing$
Dequeue v from Q
for $v' \in N(v) = \{v' \mid v' \text{ is a neighbor of v}\}$
if $d(v') > d(v) + e_{vv'}$
Enqeue v' in Q
$d(v') = d(v) + e_{vv'}$
$\sigma(v') = \sigma(v)$
Push (v', d(v')) from Q
else
$\sigma(v') = \sigma(v') + \sigma(v)$
Append v to P(v')
end if
end for
end while
$\delta(v) = 0,\ v \in V$
while $S \neq \varnothing$
Pop v' from S
for $v \in P(v')$
$\delta(v) = \delta(v) + \frac{\sigma(v)(1+\delta(v'))}{\sigma(v')}$
end for
if $v' \neq s$
$C_B(v') = C_B(v') + \delta(v')$
end if
end while
end for
for $v \in V$
$C_b(v) = \frac{C_b(v)}{(n-1)(n-2)}$ #for undirected networks
$C_b(v) = 2\frac{C_b(v)}{(n-1)(n-2)}$ #for directed networks
end for
return $C_B(v)$

Observation 4.2.4 *Note that Brandes' algorithm works for unweighted networks, in fact this algorithm has the assumption that an edge between two nodes has to be either present or absent. In weighted networks the transaction between two nodes might be quicker along paths with more intermediate nodes that are strongly connected than paths with fewer weakly-connected intermediate nodes. To extend the algorithm to weighted graphs we introduce Algorithm 4.3, where the shortest paths are calculated using the Dijkstra algorithm, instead of just counting the number of edges in each path.*

Remark 4.2.1 *Algorithm 4.2 has complexity $O(|V|^2 log(|V|) + |V||E|)$., while Algorithm 4.3 has complexity $O(|V||E|)$, where $|V|$ is the number of nodes N and $|E|$ is the number of edges.*

4.3 Girvan-Newman's functioning and pseudo-code

The Girvan-Newman algorithm, instead of trying to construct a measure that tells us which edges are most central to communities, it focuses instead on those edges that are least central. Rather than constructing communities by adding the strongest edges to an initially empty vertex set, we construct them by progressively removing edges from the original graph, so by using a divisive approach.

This algorithm extends Definition 4.2.3 to the case of edges.

Definition 4.3.1 *The edge betweenness of an edge is the number of shortest paths between pairs of nodes that run along it. If there is more than one shortest path between a pair of nodes, each path is assigned equal weight such that the total weight of all of the paths is equal to unity. If a network contains communities or groups that are only loosely connected by a few inter-group edges, then all shortest paths between different communities must go along one of these few edges. We can express the edge betweenness as:*

$$b_{ij} = \sum_{(k,l)\in V^2} \frac{\sigma_{kl}(i,j)}{\sigma_{kl}}$$

Where σ_{kl} is the number of shortest paths connecting k to l, and $\sigma_{kl}(i,j)$ is the subset of these including edge (i,j).

Definition 4.3.2 *Just like the vertex betweenness, the edge betweenness can be normalized to*

range $[0,1]$ *by using the formula:*

$$b_{ij}^{normalized} = \frac{b_{ij} - b_{min}}{b_{max} - b_{min}}$$

Observation 4.3.1 *Betweenness centrality is a good fit for community detection tasks because if a network contains communities or groups that are connected only by a few intergroup edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness. By removing these edges, we separate groups from one another and so reveal the underlying community structure of the graph.*

Observation 4.3.2 *If a network contains communities or groups that are only loosely connected by a few intergroup edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness. By removing these edges, we separate groups from one another and so reveal the underlying community structure of the graph.*

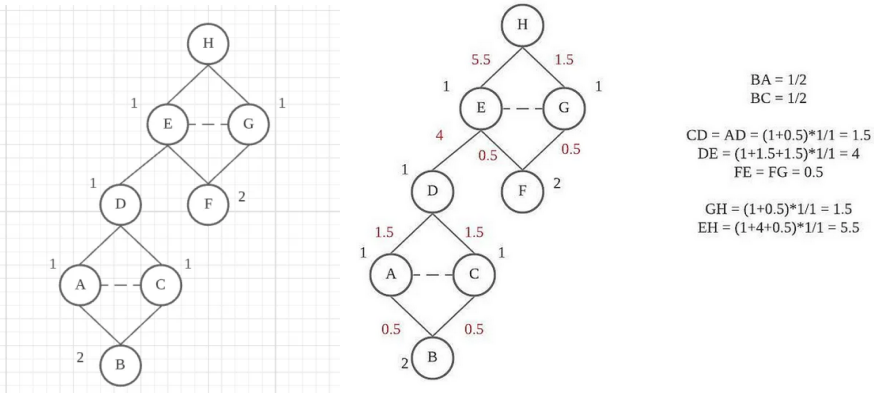

Figure 4.4: Calculation of edge betweenness. The first steps are the same as in Figure 4.2 and 4.3: we select node H and find the number of shortest path from the node H to each of the node. Then we to calculate the credit of each edge starting from the leaf node. The credit of an edge is given by the formula $(1 + \sum IncomingEdgeCredit) \frac{ScoreOfDestination}{ScoreOfStart}$, where the score of a vertex is the number of shortest paths from the node and H. Picture taken from [37].

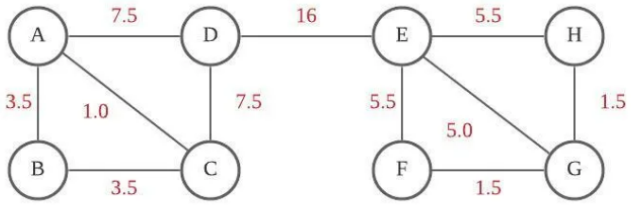

Figure 4.5: Calculation of edge betweenness. The steps in Figure 4.4 are repeated for all the nodes, after that we sum up all of the edge credit we computed in the previous step and then divide by 2. Picture taken from [37].

Observation 4.3.3 *If there is more than one shortest path between a pair of vertices, each path is given equal weight such that the total weight of all of the paths is unity.*

The Girvan Newman algorithm for community detection has the following steps:

- The betweenness of all existing edges in the network is calculated first.
- The edges with the highest betweenness are removed.
- The betweenness of all edges affected by the removal is recalculated.
- Steps 2 and 3 are repeated until no edges remain

The end result of the Girvan–Newman algorithm is a dendrogram.

Algorithm 4.4 Girvan-Newman algorithm

input Network G = (V, E)
Community assignment $M = V$
while |E| > 1
  for $e \in E$
    Calculate edge betweenness
  end for
  Find the edges with the maximum betweenness CB_{max}
  for $e \in CB_{max}$
    Remove e from G
  end for
  Update M as the set of the connected components of G
end while
return M

Observation 4.3.4 *The fact that the only betweennesses being recalculated are only the ones which are affected by the removal, may lessen the running time of the process. However, the betweenness centrality must be recalculated with each step, the reason is that the network adapts itself to the new conditions set after the edge removal. For instance, if two communities are connected by more than one edge, then there is no guarantee that all of these edges will have high betweenness. According to the method, we know that at least one of them will have, but nothing more than that is known. By recalculating betweenness centralities after the removal of each edge, it is ensured that at least one of the remaining edges between two communities will always have a high value.*

Remark 4.3.1 *Remark 4.2.1 states the calculation of betweenness for all edges (and for all nodes) has complexity $O(|V|^2 log(|V|) + |V||E|)$ for unweighted graphs and $O(|V||E|)$ for weighted graphs. Because this calculation has to be repeated once for the removal of each edge, the entire algorithm runs in worst-case time $O(|V|^3 log(|V|) + |V|^2|E|)$ for unweighted graphs and $O(|V|^2|E|)$ for weighted graphs. However, after the removal of each edge, we only have to recalculate the betweennesses of those edges that were affected by the removal, which is at most only those in the same component as the removed edge. This means that running time may be better than worst-case for networks with strong community structure (those that rapidly break up into separate components after the first few iterations of the algorithm).*

4.4 Application to four different types of networks

In this section we are going to see different applications of the Girvan-Newman algorithm on different types of graphs, more specifically, we are going to apply it to the same networks analyzed in Section 2.3 and 3.4 in order to compare the results.

4.4.1 Unweighted undirected network

Applying the Girvan-Newman algorithm to the Jazz musicians graph we obtain 40 communities: most of the communities are of smaller sizes, containing between 1 and 4 nodes, while three of them are made by respectively 49, 59 and 44 nodes.

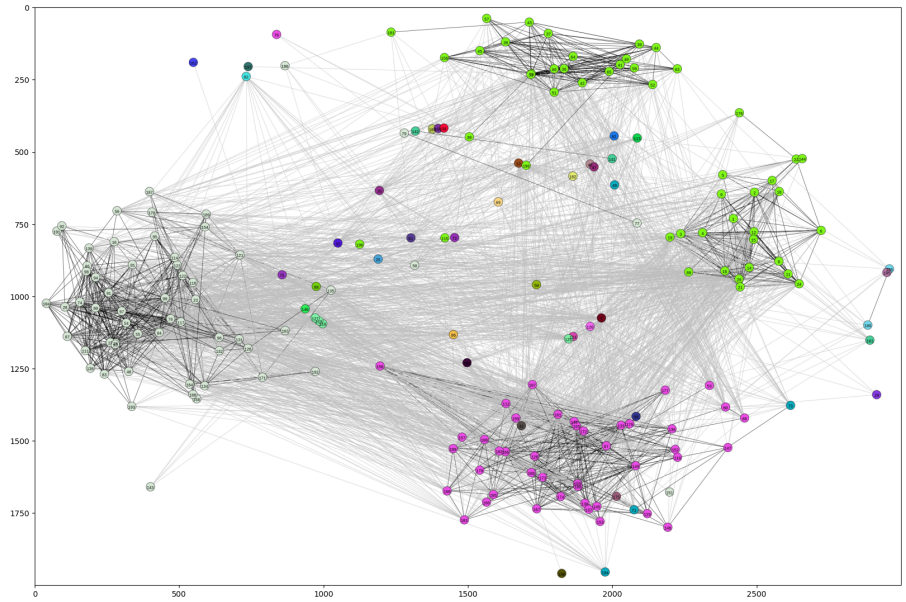

Figure 4.6: Graphical representation of the modules detected by the Girvan-Newman algorithm on the Jazz musicians network.

We are now going to compare the results with the two previous algorithms.

Algorithm	**Number of modules**	**Modularity**	**Map equation**
Louvain	4	0.43890781537538287	11.761273616050756
Infomap	5	0.038870920032069954	6.910117006606936
Girvan-Newman	40	0.4050988992046884	119.07819455516743

Table 4.1: Results of the three algorithms on the Jazz musicians network.

We can observe a much higher value of number of communities and map equation for the Girvan-Newman algorithm, while in terms of modularity, the algorithm gives better results than Infomap but worse than Louvain.

4.4.2 Weighted undirected network

For the Windsurfers network 6 communities were detected: 21 nodes for the first community, 18 for the second and 1 for the remaining ones

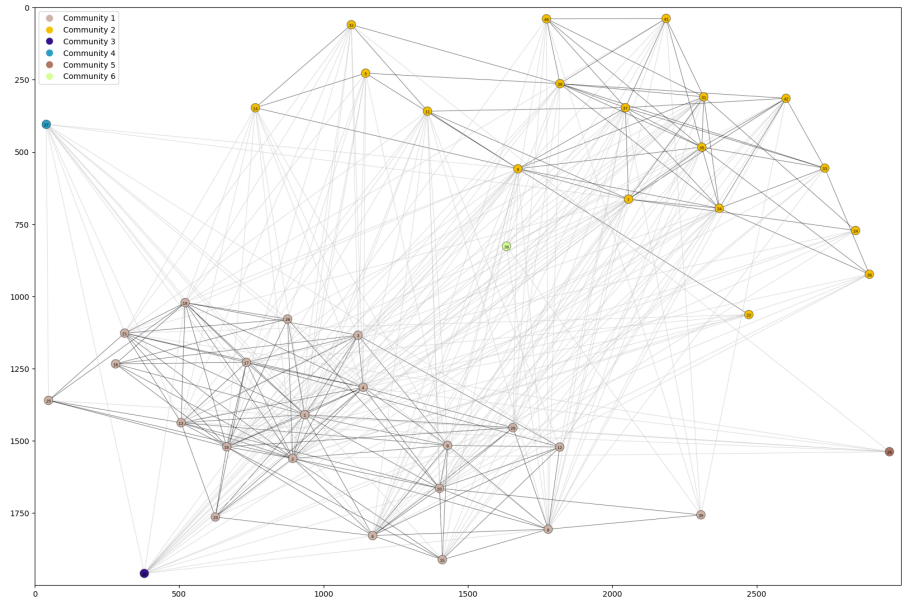

Figure 4.7: Graphical representation of the modules detected by the Girvan-Newman algorithm on the Windsurfers network.

We are now going to compare the results with the two previous algorithms.

Algorithm	**Number of modules**	**Modularity**	**Map equation**
Louvain	2	0.37121605900844046	5.535010448869063
Infomap	3	0.20192741378788756	4.5052902422623315
Girvan-Newman	6	0.3517397809185141	11.301612762000586

Table 4.2: Results of the three algorithms on the Windsurfers network.

In this case, we also have a higher number of communities and map equation value for the Girvan-Newman algorithm, although not as high as in the previous case. Once again, it shows a modularity value almost as good as the Louvain algorithm.

4.4.3 Unweighted directed network

For the Macaque rhesus brain graph 145 communities were obtained, although the two main communities are two, with one having 84 and the other 15 nodes, while the remaining ones are singular node clusters.

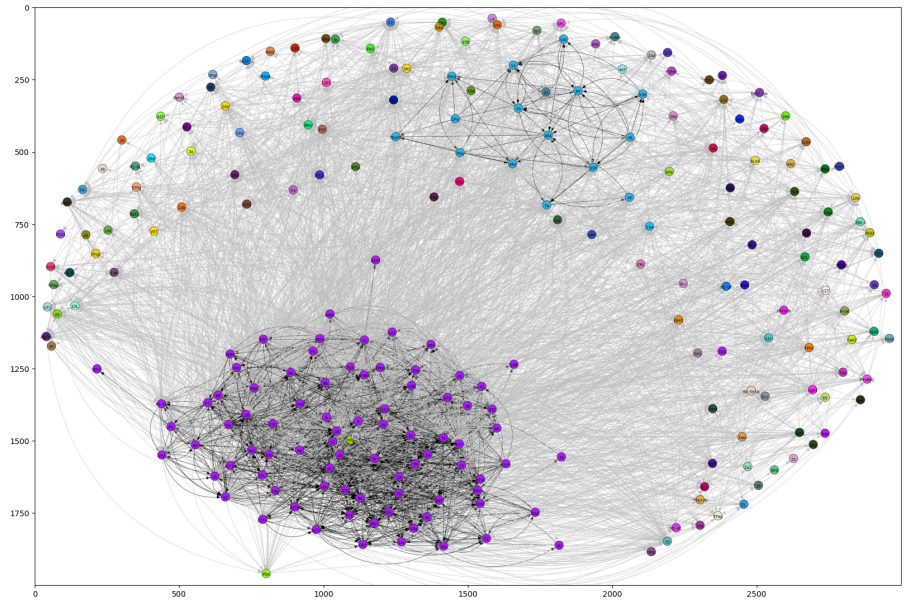

Figure 4.8: Graphical representation of the modules detected by the Girvan-Newman algorithm on the Macaque rhesus brain graph.

We are now going to compare the results with the two previous algorithms.

Algorithm	Number of modules	Modularity	Map equation
Louvain	4	0.3295535655573555	11.610916375907639
Infomap	6	0.30617633953173806	6.999063756957538
Girvan-Newman	145	0.07902224400858437	531.3086583722825

Table 4.3: Results of the three algorithms on the Macaque rhesus brain graph.

In this case we can see that Girvan-Newman has the worst performance between the three algorithms, in fact we can observe a low value for modularity and high value for the map equation.

4.4.4 Weighted directed network

For the US Congress Twitter network we got 15 resulting communities, although the two main communities are two, with one having 280 and the other 183 nodes, while the remaining ones are singular node clusters.

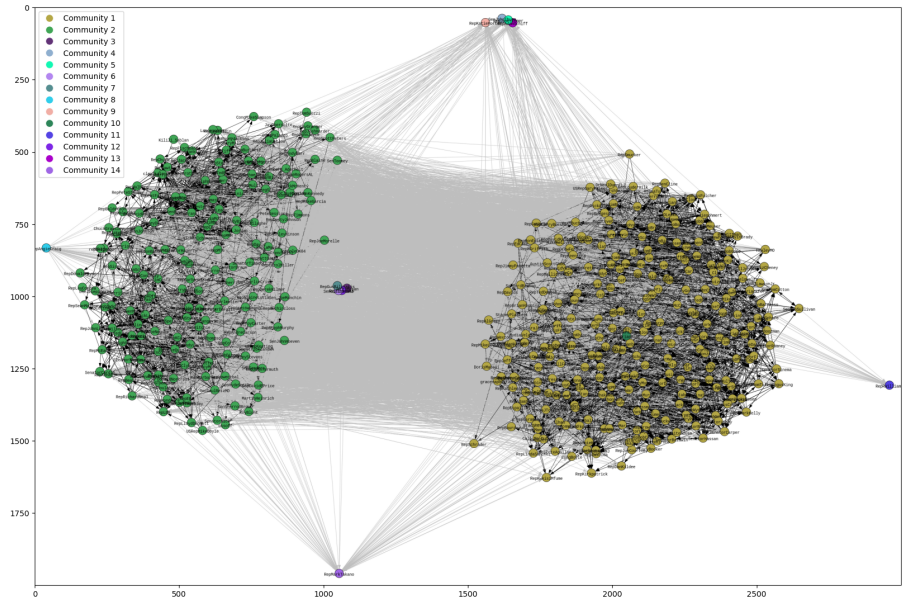

Figure 4.9: Graphical representation of the modules detected by the Girvan-Newman algorithm on the US Congress Twitter network.

We are now going to compare the results with the two previous algorithms.

Algorithm	Number of modules	Modularity	Map equation
Louvain	4	0.4399847144924596	11.614856746923584
Infomap	15	0.007693580298894281	7.6239123302314455
Girvan-Newman	15	0.38012164654971503	19.591171591664526

Table 4.4: Results of the three algorithms on the US Congress Twitter network.

With the weighted directed network, Infomap and Girvan-Newman yield the same number of communities, although of different sizes. In terms of modularity the best algorithm is once again Louvain and for the map equation is Infomap, although Girvan-Newman yields a pretty high modularity value.

Overall, we can observe that the number of communities detected by Louvain and Infomap is pretty stable in most cases, while it tends to grow considerably for Girvan-Newman. This is due to the fact that, as we saw in Remark 2.2.1 and Remark 3.3.3, both Louvain and Infomap suffer from a resolution problem, which means that they have the tendency of detecting larger sized communities, while Girvan-Newman does not have this limitation, hence the fact that a

lot a lot of the detected communities by the algorithm are very small, which of course influences the overall number of modules.

As expected, in all of the analyzed networks, the best results in terms of modularity are given by Louvain. With the exeption of the unweighted directed case, Girvan-Newman detects communities with higher modularity values than Infomap.

In terms of map equation, as expected, Infomap yields the best results, followed by Louvain and then Girvan-Newman.

5

Application to an e-mail dataset

One of the main topics of research in complex networks theory and processes is how the community structure in real-life networks influences the spread of information, fake news, diseases and how it affects the efficiency of communication. In this chapter we are going to focus on the topic of communication by e-mail: the analysis is going to be on a dataset containing e-mail exchanges between different users. The focus will be on the response times defined as the number of units of activity of the receiver pertaining to the intervals between a message and its response (see Definition 5.1.2). Thanks to the user-level response times analysis done on [38], we know that for the typical user in the *DE*1 database, the response time probability density function is close to a power law with exponent $\alpha \simeq -1.5$. Our aim is to perform a community-level analysis, which means that we are going to analyze the response times between communities of users, rather than between the single agents. The communities are going to be detected using one of the methods described in the previous chapters. The reasoning behind this analysis is to get an assessment on whether doing a community-level study instead of a user-level one causes the probability density function of the response times to deviate from a power law and whether the exponent α changes.

5.1 Preliminary concepts and dataset description

Before giving a description of our dataset, it's important that we define a truncated power law distribution.

Definition 5.1.1 *Given a positive random variable σ, we say that its probability density function follows a truncated power law distribution if $\mathbb{P}(\sigma \mid \alpha, \lambda) = \sigma^{\alpha} e^{-\frac{\sigma}{\lambda}}$, where $\alpha < 0$ is the power-law exponent and $\lambda > 0$ is the cutoff parameter.*

Remark 5.1.1 *For values of $\sigma < \alpha$ the probability density function behaves like a standard power-law, while for values larger than λ we observe an exponential decay in the distribution. The focus of this analysis is going to be on estimation of the exponent α.*

The exchanges that we're going to analyze are from the Database DE1, which is an email database concerning the long-term activity of all the accounts belonging to, and interacting with, a Department of a large EU university, extending over a period of about two years. In particular, this dataset contains 6914872 rows and three columns: "Timestamp", "Sender" and "Receiver", where timestamps are given in seconds and senders and receivers are conventionally numbered, which means that a unique code is assigned to each user.

To select the most relevant users, we have first considered the 500 agents with the largest number of outgoing messages, and having a ratio $r = \frac{\#incoming}{\#outgoing}$. From these, we have extracted the 300 agents with the largest number of question-reply pairs (this gives a set of agents with at least 390 responses each, a large percentage of which have in the order of a few thousand responses), resulting in a dataframe of 753492 rows.

	Timestamp	Sender	Receiver
1	1278829561	537	72
2	1278829561	537	365
3	1278829744	538	351
4	1278831285	559	61
5	1278831823	559	573
...	...	...	...
753488	1341701043	224	72
753489	1341701235	224	72
753490	1341701832	224	1048
753491	1341705062	1005	224
753492	1341705062	1005	13

Figure 5.1: Cleaned DE1 dataset: each row represents a message between two agents at a certain timestamp.

Another important definition is the one of response times.

Definition 5.1.2 *Given any message $\mathcal{M}$ from any agent $\mathcal{B}$ to $\mathcal{A}$, and the first ensuing message $\mathcal{M}'$ going from $\mathcal{A}$ to $\mathcal{B}$, the response times of agent $\mathcal{A}$ are defined through the activity parameter s of $\mathcal{A}$ by counting the values $\sigma = \Delta s$ pertaining to the intervals between the messages $\mathcal{M}$ and $\mathcal{M}'$, i.e. the number of outgoing messages from $\mathcal{A}$ intervening between $\mathcal{M}$ and $\mathcal{M}'$.*

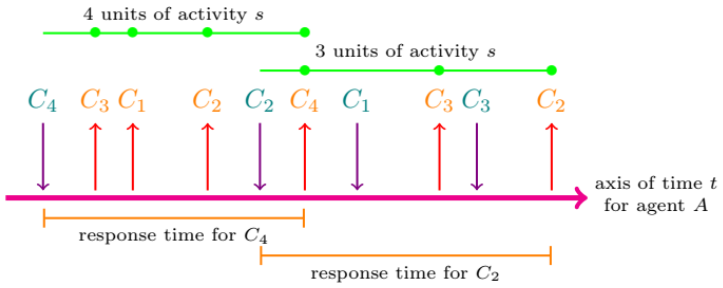

Figure 5.2: Activity clock for a node in an interaction network. Representation of the node's temporal activity, in this case written communication, along the axis of time t for an agent $\mathcal{A}$. Arrows pointing into the t axis mark incoming messages from the indicated correspondents $C_1, C_2, \ldots$ of $\mathcal{A}$. Arrows pointing out of the t axis mark response messages from $\mathcal{A}$ to the same correspondents. The intervals between the outgoing arrows define the response times of $\mathcal{A}$ pertaining to each correspondent C_i counts the number of outgoing messages from $\mathcal{A}$. Picture taken from [39].

	Timestamp	Sender	Receiver
1	1278829561	537	72
2	1278829561	537	365
3	1278829744	538	351
4	1278831285	559	61
5	1278831823	559	573
...	...	...	...
219	1278912974	61	265
220	1278912974	61	567
221	1278912976	61	8
222	1278912976	61	86
223	1278912976	61	559

Figure 5.3: First 223 rows of the DE1 dataset: each row represents a message between two agents at a certain timestamp.

Example 5.1.1 *Let's say we want to calculate the response time of the message at Row* 4 *in Figure 5.3: we have five units of activity from agent* 61 *between the message (Row* 4*) and the response (Row* 223*), these five outgoing messages are from Row* 219 *to Row* 223 *(we have consider the response itself as a unit of activity), so in this case the response time would be* 5.

Remark 5.1.2 *It is possible that some messages don't receive any response, in that case the response time won't be computed.*

The idea is that, given a fixed agent $\mathcal{A}$, we calculate the response time of every message $\mathcal{M}$ from any other user $\mathcal{B}$ to $\mathcal{A}$, the result is a response times vector. If we repeat this process for every agent we get n response times vectors, where n is the number of agents.

As stated earlier, the studies on [38] allows us to observe the behavior of the probability density function for the response times of a single user and it's close to a standard power law with exponent $\alpha \simeq -1.5$.

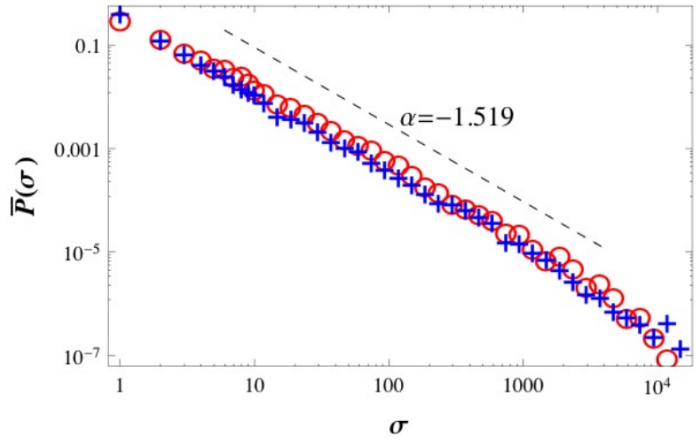

Figure 5.4: Log-log plot of the response-time probability density $\mathbb{P}(\sigma)$ of one of the users in the DE1 database. We can also observe the estimated slope $\alpha \simeq -1.519$ represented as a dotted line to guide the eye. Red circles indicate the calculated empirical data; blue crosses represent predictions using the standard power law model $\mathbb{P}(\sigma) = \sigma^{\alpha}$, where $\alpha = -1.519$. The plots were produced by using logarithmic binning. Picture taken from [38].

The first thing that is necessary in order to switch from user-level to cluster-level analysis is to transform the dataframe into a network, where each user represents a node. Given the graph we can then apply the three algorithms and select the method that yields the optimal network partitioning.

In the final section we perform a community-level study of the probability distribution of the response times: this means that instead of identifying each agent by a unique code, we

identify it by the assigned cluster. Instead of having the 300 agents like in the user-level analysis, we have m, where m is the number of communities of the network partition that we're going to utilize.

The last step is to observe if the behavior of the density function actually depends by the community structure. In order to study this correlation we randomly reassign each vertex to one the clusters. The random assignment of the vertices is based on a specific probability distribution: if we have a network of N nodes (in our case $N = 300$) and the original partition is into m clusters, where the respective number of nodes in each cluster is $n_1, \ldots, n_m$, in the randomization process each vertex is reassigned to cluster i with probability $p_i = \frac{n_i}{N}$.

If there is indeed a correlation between the community structure of the network and the response times of each cluster, the probability distributions of the response times for each community should follow distribution similar to a standard or truncated power law, after the randomization process this property should be lost and the probability density function of the response times should not follow that same distribution.

5.2 Community detection on the e-mail dataset

In order to apply the algorithms, we need to create a network from our original dataset. The best way to represent our data is by building a weighted directed graph: we create an edge from User1 to User2 if at certain timestamp User1 sent a message to User2 and the value of assigned weight is the number of messages that User1 sent to User2.

The results of the Louvain method to our graph is four communities made by 119, 109, 51 and 21 nodes each.

The modularity value for this partition is 0.43131974320099575 and the map equation value is 10.866464455561054, so we can say that this community assignment is fairly good according to both metrics.

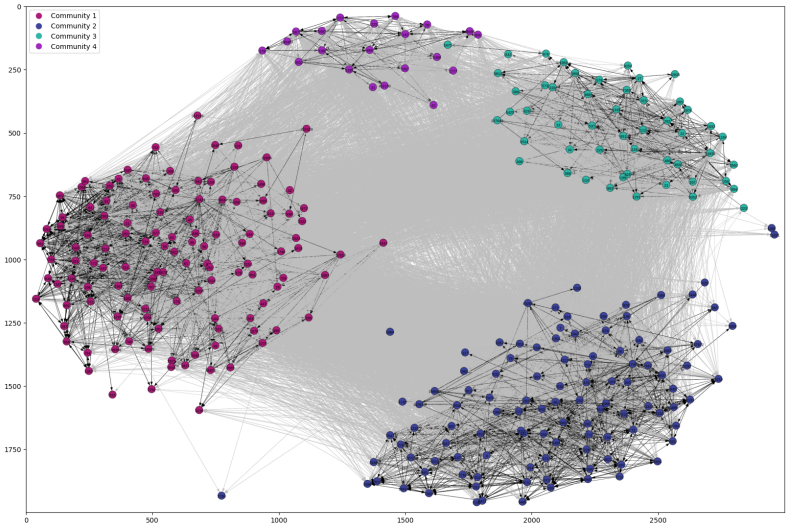

Figure 5.5: Graphical representation of the communities yielded by the Louvain algorithm on the DE1 network.

Remark 5.2.1 *Both Infomap and Girvan-Newman return a large number of communities,* 27 *and* 195 *respectively, with lots of very small or even singular-node communities. Since we already know from [38] that the behavior of the probability distribution of the response times for single users is close to a power law with* $\alpha \simeq -1.5$*, our focus is going to be on the network partition returned by Louvain.*

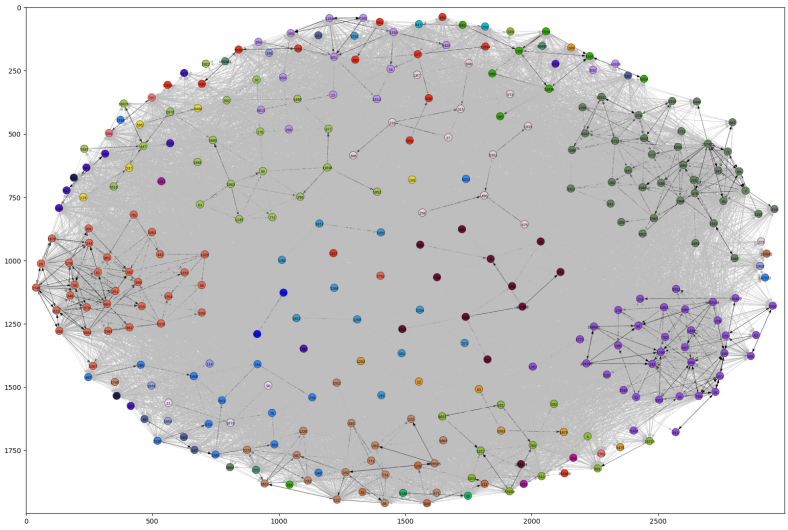

Figure 5.6: Graphical representation of the 27 communities yielded by the Infomap algorithm on the DE1 network.

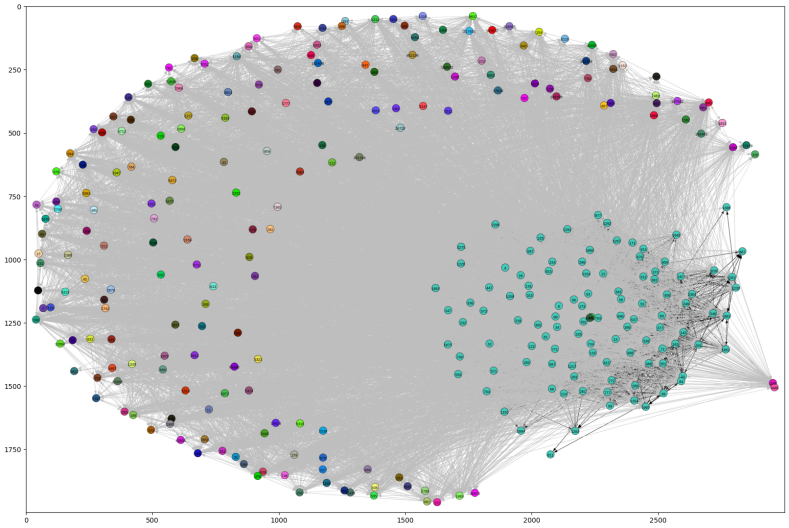

Figure 5.7: Graphical representation of the 195 communities yielded by the Girvan-Newman algorithm on the DE1 network: we have 1 large community with 106 nodes and 194 communities with one vertex each.

Now that the community assignment is selected we can proceed the community-level response times analysis: in particular we are going to compare how the response times are distributed when the partitioning is actually based on the modular structure of the network versus when the partition is created randomly. At the end of this analysis we are going to have an assessment on the influence that the community structure of the DE1 database has on efficiency of the communication between different clusters.

5.3 Community-level response times analysis

In this section we are going to analyze how the community structure of the DE1 network influences the response times between users belonging to different modules. The first thing we need to do is create a new version of the DE1 dataset where both the sender and receiver are mapped to their corresponding community, resulting in two new columns "Community1" and "Community2", we are going to denote this dataframe as DE1 community dataset.

	Timestamp	Community1	Community2
1	1278829561	1	1
2	1278829561	1	2
3	1278829744	2	2
4	1278831285	4	3
5	1278831823	4	4
...	...	...	...
753488	1341701043	1	1
753489	1341701235	1	1
753490	1341701832	1	1
753491	1341705062	1	1
753492	1341705062	1	1

Figure 5.8: *DE*1 community dataset, where each is user is mapped to its corresponding community: Community1 is the community of the sender, while Community2 is the community of the receiver. For this example, we mapped the nodes based on the partition in Figure 5.5.

Remark 5.3.1 *As we can see from Figure 5.8, in many rows of the DE*1 *community dataset Community*1 *and Community*2 *are equal, in fact it is very likely that two users that often communicate are assigned to the same community. If we have a message $\mathcal{M}$ going from agent $\mathcal{A}$ to $\mathcal{A}$, that row is going to be removed from the dataset.*

	Timestamp	Community1	Community2
1	1278829561	1	1
2	1278829561	1	2
3	1278829744	2	2
4	1278831285	4	3
5	1278831823	4	4
6	1278831982	1	3
7	1278832583	1	1
8	1278832583	1	3
9	1278833339	4	1
10	1278833778	1	4
11	1278833816	2	2
12	1278833816	2	1

	Timestamp	Community1	Community2
1	1278829561	1	2
2	1278831285	4	3
3	1278831982	1	3
4	1278832583	1	3
5	1278833339	4	1
6	1278833778	1	4
7	1278833816	2	1

Figure 5.9: On the left we have the first 12 rows of the DE1 community dataset and on the right DE1 cleaned community dataset, which is obtained after removing all the intra-community messages from the DE1 community dataset, so by removing rows 1, 3, 5, 7 and 11 from the original dataframe. The nodes are assigned once again based on the partition in Figure 5.5.

The focus of our analysis is on the exchanges between members of the same partition, so discarding intra-community interaction allows us to get a better assessments on the communication patterns between the different communities.

Definition 5.3.1 *Assume that we have a message $\mathcal{M}_0$ from agent $\mathcal{A}$ to agent $\mathcal{B}$, a series of messages $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_n$ from agent $\mathcal{B}$ to agent $\mathcal{A}$ with timestamps $t_0 \leq t_1 \leq \cdots \leq t_n$ and the first ensuing message $\mathcal{M}'$ from agent $\mathcal{A}$ to agent $\mathcal{B}$ with timestamp $t_{n+1} \geq t_n$.*
The messages $\mathcal{M}_2$, ..., $\mathcal{M}_n$ are considered reminders and no response times will be calculated for them, only for the first message $\mathcal{M}_1$.

Let's look at an example.

	Timestamp	Community1	Community2
20	1278838719	2	1
21	1278844047	1	2
22	1278844711	1	2
23	1278848931	4	2
24	1278851192	1	3
25	1278852816	1	2
26	1278852816	1	2
27	1278852976	1	4
28	1278853115	1	2
29	1278853384	1	2
30	1278856019	2	1

Figure 5.10: Rows 20 to 30 of the DE1 cleaned community dataset. The nodes are assigned once again based on the partition in Figure 5.5.

Example 5.3.1 *According to Definition 5.3.1, the email at Row* 20 *would be* $\mathcal{M}_0$ *and the messages at Rows* 22, 25, 26, 28 *and* 29 *are considered reminders, so for these messages the response time will not be calculated. In this case we only compute the response time of the email on Row* 21, *which would be* 1, *in fact the only unit of activity from agent* 2 *is at Row* 30.

Now we proceed with the calculation of the response times before and after the randomization process. The original sizes of four communities are 119, 109, 51 and 21 nodes, which means that in the randomization process each node is reassigned to Community 1 with probability $\frac{119}{300}$, to Community 2 with probability $\frac{109}{300}$, to Community 3 with probability $\frac{51}{300}$ and to Community 4 with probability $\frac{21}{300}$; the result is four communities with 123, 124, 35 and 18 vertices each. By reassigning the nodes according to this probability distribution the number of nodes in each cluster does not remain exactly the same, but it still allows us to maintain the communities' sizes on a larger scale and introduce a component of "noise" in the process at the same time.

In order to see how close the probability distribution is to a power-law we produced a log-log plot.

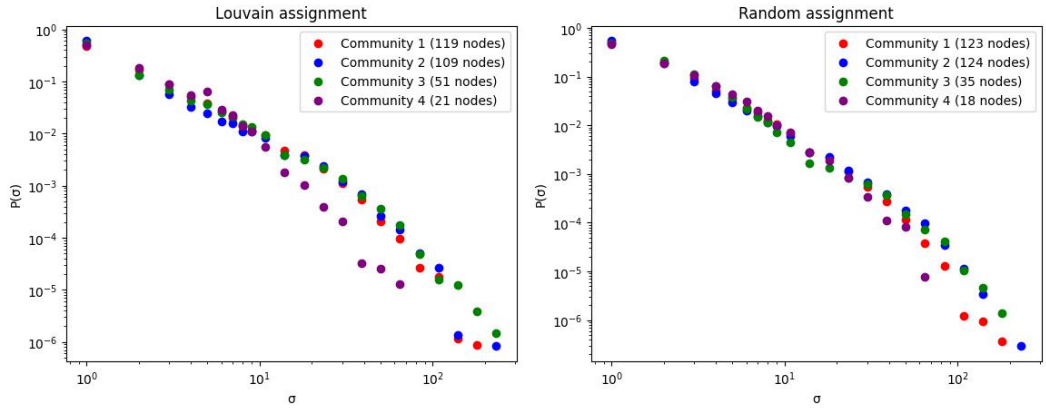

Figure 5.11: Log-log plots of the response-time probability densities $\mathbb{P}(\sigma)$ where each color represents a different agent. On the picture on the right the response times were calculated based on the partition yielded by Louvain, on the picture on the left they were calculated based on the partition yielded by the randomization process. The plots were produced by using logarithmic binning.

On the left picture the four agents seem to overlap each other for smaller values, while they deviate from each other for larger values, it seems that the cluster that deviates the most is the one with the smallest number of vertices (Community 4). On the right we observe a similar behavior, although the deviation between the agents is not as strong. In order to get a unique probability function and to get a better assessment on its behavior, for each value of σ in Figure 5.11 we computed the mean of the values of $\mathbb{P}(\sigma)$ of all the agents.

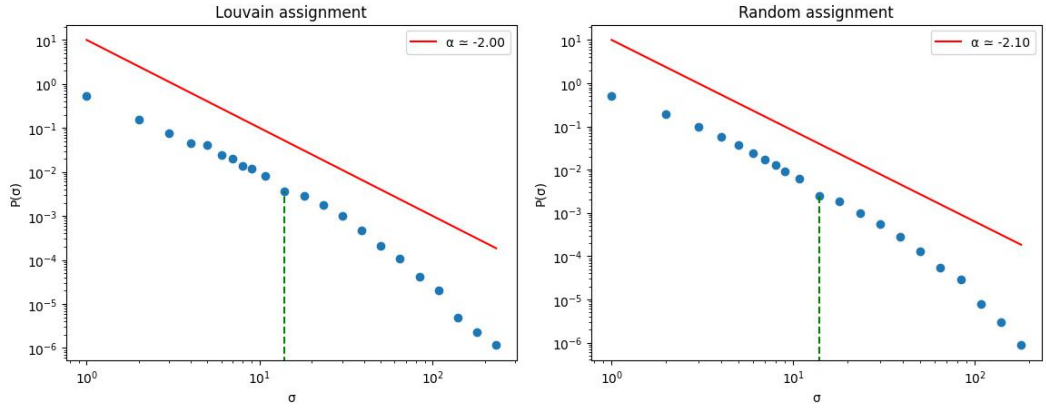

Figure 5.12: Log-log plots of the mean of the response-time probability densities $\mathbb{P}(\sigma)$ of all the agents. On the picture on the right the response times were calculated based on the partition yielded by Louvain, on the picture on the left they were calculated based on the partition yielded by the randomization process. We can also observe the estimated values of α as two red lines to guide the eye: one with slope -2 for the standard assignment and one with slope -2.1 for the random one. For values larger than $\lambda \simeq 14$ (straight green line), we can see that in both assignments the distribution starts to decay exponentially. The plots were produced by using logarithmic binning.

From these plots we can observe that, before and after the randomization process, the probability density function of the response times follow a truncated power law, in fact the points are arranged in a straight line for lower values of σ, while they decay exponentially for values larger than $\lambda \simeq 14$, also the estimated slope α remains almost the same in both plots.

These results may suggest that the correlation between the community structure and the response times that we were looking for is not present, in fact the distribution of the response times for the random assignment is very similar to the distribution of the original partition yielded by Louvain, which is an unexpected result.

What we were hoping to obtain is a power law-like distribution when using the standard assignment and a different distribution when the assignment is random. We can conclude is that the closeness to a truncated power law of the probability distribution is more related to properties of the network like the degree distribution and connectivity, rather than the specific community assignment.

It is also important to note that the type of analysis performed is still preliminary and during the procedure some choices were made arbitrarily, for example in the selection of the algorithm or with the removal of some of the rows in the dataset, like explained in Remark 5.3.1 and in Definition 5.3.1; these are all factors that might have influenced the results.

Even if we were not able to observe the expected results, we were still able to find an example of a real-life network where the efficiency of the communication seems to not be influenced by the community structure.

6
Conclusions

In this thesis, we have studied three of the more heavily used algorithms for the detection of communities in networks: Infomap, Louvain and Girvan-Newman.

Louvain and Infomap follow a similar approach, as they are both optimization algorithms: the first operates by maximizing modularity, which is metric of evaluation for networks' partitions; Infomap is a minimization algorithm and the measure of interest is the map equation, which leans on information theory to give another quality measure of a network's partition.

Girvan-Newman, on the other hand, is hierarchical clustering method, meaning that it operates by grouping similar objects into clusters based on their distances or similarities, in our case the grouping is based on a measure called edge betweenness centrality.

The differences between the approaches are highlighted by the application of the algorithms on four real-life networks: in most of the cases, the number of communities detected by the first two algorithms are fairly close, while for Girvan-Newman they tend to be much higher.

In the final chapter we studied the effect that the community structure of a complex network has on communication, which was done by analyzing a dataset containing e-mail exchanges between different users. The focus of our studies was the response time defined as the number of units of activity of the receiver pertaining to the intervals between a message and its response.

On [38] we saw that the probability density function of the response times of single users typically follow a distribution similar to a power law with exponent $\alpha \simeq -1.5$.

We carried out a community-level analysis in order to see if the behavior of the distribution of the response times between groups of users is actually different from the one of the response

times between single users.

In order to perform this type of analysis we had to build a graph of all the users based on their interactions in the original dataset, then we applied the Louvain method to detect the community structure of such network.

After calculating the response times between the four communities detected by Louvain, we observed that the results are similar to the ones obtained in the user-level analysis on [38]: the distribution of the average response times of the four agents is very close to a truncated power law with estimated exponent $\alpha \simeq -2$.

The final step of our analysis was to randomly reassigning each node to one of the four clusters and compute the response times between the four random communities. The goal of this randomization process was to see if there is indeed a correlation between the community structure and the probability density function of the response times or if the two are unrelated. The results that we obtained suggest the latter hypothesis; in fact the probability density function obtained with the random assignment is almost the same as the one obtained with network partitioning yielded by Louvain. If there was a correlation between the communities and the distribution, reassigning the nodes randomly should "break" the power law, which did not happen.

This analysis suggests that the similarity to a power law of the probability distribution of the response times is more related to other properties of the network, like the degree distribution and connectivity, rather than the specific community assignment and that in this complex network the communication appears to be unaffected by the community structure.

References

[1] https://en.wikipedia.org/wiki/Graph_(discrete_mathematics).

[2] A. Ravikiran, "What is graph in data structure types of graph?" https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure.

[3] S. K. Gupta and D. P. Singh, "Cbla: A clique based louvain algorithm for detecting overlapping community," *Procedia Computer Science*, vol. 218, pp. 2201–2209, 2023.

[4] R. Vershynin, *High-dimensional probability: An introduction with applications in data science*. Cambridge university press, 2018, vol. 47.

[5] P. Brémaud, *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. Springer Science & Business Media, 2013, vol. 31.

[6] https://cs.stackexchange.com/questions/140279/probability-of-reaching-a-state-in-asymmetric-random-walk.

[7] https://en.wikipedia.org/wiki/Configuration_mode.

[8] https://en.wikipedia.org/wiki/Modularity_(networks).

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[10] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[11] E. A. Leicht and M. E. Newman, "Community structure in directed networks," *Physical review letters*, vol. 100, no. 11, p. 118703, 2008.

[12] https://en.wikipedia.org/wiki/Louvain_method.

[13] A. Lancichinetti and S. Fortunato, "Consensus clustering in complex networks," *Scientific reports*, vol. 2, no. 1, p. 336, 2012.

[14] A. Mishra and K. Patra, "Domination and independence parameters in the total graph of zn with respect to nil ideal," *IAENG International Journal of Applied Mathematics*, vol. 50, no. 3, pp. 1–6, 2020.

[15] P. M. Gleiser and L. Danon, "Community structure in jazz," *Advances in complex systems*, vol. 6, no. 04, pp. 565–573, 2003.

[16] L. C. Freeman, S. C. Freeman, and A. G. Michaelson, "On human social intelligence," *Journal of Social and Biological Structures*, vol. 11, no. 4, pp. 415–425, 1988.

[17] K. S. Ambrosen, S. F. Eskildsen, M. Hinne, K. Krug, H. Lundell, M. N. Schmidt, M. A. van Gerven, M. Mørup, and T. B. Dyrby, "Validation of structural brain connectivity networks: The impact of scanning parameters," *Neuroimage*, vol. 204, p. 116207, 2020.

[18] C. G. Fink, K. Fullin, G. Gutierrez, N. Omodt, S. Zinnecker, G. Sprint, and S. McCulloch, "A centrality measure for quantifying spread on weighted, directed networks," *Physica A*, 2023.

[19] C. G. Fink, N. Omodt, S. Zinnecker, and G. Sprint, "A congressional twitter network dataset quantifying pairwise probability of influence," *Data in Brief*, 2023.

[20] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[21] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the national academy of sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[22] M. Rosvall, D. Axelsson, and C. T. Bergstrom, "The map equation," *The European Physical Journal Special Topics*, vol. 178, no. 1, pp. 13–23, 2009.

[23] R. Lambiotte and M. Rosvall, "Ranking and clustering of nodes in networks with smart teleportation," *Physical Review E*, vol. 85, no. 5, p. 056107, 2012.

[24] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[25] M. T. Schaub, R. Lambiotte, and M. Barahona, "Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation," *Physical Review E*, vol. 86, no. 2, p. 026112, 2012.

[26] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Resonance*, vol. 11, no. 2, pp. 91–99, 2006.

[27] S.-H. Bae, D. Halperin, J. D. West, M. Rosvall, and B. Howe, "Scalable and efficient flow-based community detection for large-scale graph analysis," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 3, pp. 1–30, 2017.

[28] https://www.mapequation.org/infomap/.

[29] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[30] https://it.mathworks.com/help/stats/dendrogram.html.

[31] https://en.wikipedia.org/wiki/Betweenness_centrality.

[32] https://en.wikipedia.org/wiki/Girvan%E2%80%93Newman_algorithm.

[33] https://www.cl.cam.ac.uk/teaching/1617/MLRD/slides/slides13.pdf.

[34] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[35] https://toreopsahl.com/tnet/weighted-networks/node-centrality/.

[36] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[37] https://medium.com/analytics-vidhya/girvan-newman-the-clustering-technique-in-network-analysis-27fe6d665c92.

[38] M. Formentin, A. Lovison, A. Maritan, and G. Zanzotto, "Hidden scaling patterns and universality in written communication," *Physical Review E*, vol. 90, no. 1, p. 012817, 2014.

[39] ——, "New activity pattern in human interactive dynamics," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2015, no. 9, p. P09006, 2015.

Acknowledgments

First of all, I would like to thank my supervisor Professor Formentin, who provided a very interesting topic and was always available to offer advice or answer any questions I had. I would also like to thank Professor Lovison, who also assisted me during the operational phase of the dissertation.

I want to give a special thanks to my parents and sister for their constant support throughout my academic career.

Lastly, a big thank you to all my long-time friends who were there for me in various ways throughout these past two years, especially Luca, Lorenzo, Edoardo, and Marisol.