SAPIENZA
UNIVERSITÀ DI ROMA

Brain networks in Glioma using Topological Data Analysis

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Fisica - curriculum Biosistemi

Candidate

Giulia Moreni

ID number 1647223

Thesis Advisors

Prof. Andrea Giansanti - Sapienza University of Rome

Prof. Linda Douw - Amsterdam UMC

Prof. Fernando Nobrega Santos - Amsterdam UMC

Academic Year 2019/2020

Brain networks in Glioma using Topological Data Analysis
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 26, 2020

Author's email: giulia.moreni1995@gmail.com

Abstract

This thesis work consists of the study of a metric that can be suitable to compute distances between brain networks. The goal was to find a good method to differentiate brain networks of patients with different diseases and reveal differences in the topological networks properties of the different groups of patients.
The first part of the work consists of the study of two metrics: the Gromov-Hausdorff and Gromov-Wasserstein distances, used to compare brain networks. In this project, we implemented various Python algorithms to reach our goals.
We first investigated the robustness of the metrics mentioned above, applying them to simulated networks. Then we focused our analysis on experimental connectivity matrices based on fMRI data, freely available in a database of the Human Connectome Project composed of ASD patients and healthy controls. Later, we performed our analysis on a database of the Amsterdam UMC, based on MEG data, composed of Glioma patients and healthy controls.
The results we obtained were compatible with the ones in the literature, but these metrics were not powerful enough to distinguish the patients of the different groups at a clinical level. Therefore we investigated one of the techniques of Topological Data Analysis (TDA), and - using Betti-0 curves - we were able to significantly differentiate the patients at a group level. The mathematical branch behind our work is topology, a tool that is proven to be useful in brain network analysis.
In the second part of the work, we continued to use TDA, and we modelled the brain in a more complex way: as a *simplicial complex*. We, therefore, analyzed the high-order topological structure in brain networks, which includes not only simple connections between nodes (*edges*) but also more complex structures like triangles, tetrahedrons, and higher-dimensional objects. This analysis revealed a significant difference between brain networks of Glioma patients and brain networks of healthy controls. Lastly, we tried to relate the analyzed topological properties of the networks to the personal clinical traits of the patients, such as the grade of the tumour and other markers.

Contents

Introduction

The human brain contains around 100 billion neurons, and it has been estimated that there are about 10^{15} synapses. It is unimaginable to study brain functioning considering the activity of single separated neurons. Therefore, to model such a complex system, network theory has a central role.

With the popularization of complex system theory and network science, graph theory has become widely used in the theoretical description of the brain. Through the use of experimental data, many recent studies have successfully applied graph theory to model and analyze the brain as a network.

In the network description of the brain, the *nodes of the network* correspond to anatomical regions and *edges* can either be of a structural origin and represent anatomical connections formed by white matter axon fibre tracks or can be of a functional origin (representing coherent physiological activity). To store the measures between nodes in a network, we can use mathematical tools such as matrices. We can construct a symmetric NxN connectivity matrix, where N is the total number of nodes. Each row/column of the connectivity matrix corresponds to a distinct node, such that the position (i, j) stores the measure of interaction between the i-th and j-th node [1].

This thesis work consists of the study of a metric that can be suitable to compute distances between brain networks. We have analyzed two distances: the Gromov-Hausdorff and the Wasserstein distances. We first tested these metrics on simulated networks generated in Python, and then we applied them on experimental fMRI[1] connectivity matrices available in the Human Connectome Project. One technical problem in brain network analysis is to properly choose a reasonable threshold to construct brain networks using the connectivity matrices. To overcome this issue we used Topological Data Analysis (TDA) to understand how, from a connectivity matrix, it is possible to look at the network changes at all different possible thresholds (also known as filtrations), without choosing just one specific threshold. From the connectivity matrices of the brain, through a Python algorithm, it is possible to examine how fast the nodes cluster and study all the brain networks obtained at all possible thresholds. Therefore, we can analyze the so-called Betti-0 curves (representing the number of connected components vs filtration values) in

[1]Functional magnetic resonance imaging or functional MRI (fMRI) is explained in Chapter 1, Section 1.2

patients with different diseases and see how they differ from healthy controls (HC). After having practised with networks from the Human Connectome database, we focused on a database of the Amsterdam UMC consisting of Glioma patients and healthy controls based on MEG[2] data. With our analysis, we were able to differentiate the two groups significantly.
In the second part of the work, we kept using Topological Data Analysis (TDA) to compute high-order Betti numbers[3] and analyze Betti-1 curves (number of 2D holes vs filtration value) and Euler characteristic[4] curves to reveal differences in the brain networks of Glioma patients and healthy controls.
Our ultimate goal was then to try to relate the topological properties of the network with behavioural data associated with the Glioma patients in our database. In short, we wanted to understand how the computed properties of the network (Betti numbers and Euler characteristics) correlate with the individual clinical traits of the patient such as the grade of the tumour and other markers. This bridge between clinical traits of patients and network neuroscience is still an open question in research, both theoretical and empirical.

In the first chapter, an overview of graph theory and brain network science is given. In the second chapter, a theoretical background of distance definitions is provided. The third chapter presents methods to study and compare brain networks using Gromov-Hausdorff and Gromov-Wasserstein distances. The fourth chapter is dedicated to the application of such metrics on simulated data and the fifth chapter to the application on experimental data. The sixth chapter describes Topological Data Analysis and explains all the definitions and mathematical background needed to compute Betti numbers, while in the seventh chapter, the proposed method is applied to data. The last chapter is dedicated to the correlation between network properties (Betti numbers and Euler characteristic) and patient clinical traits.

[2]Magnetoencephalography (MEG) is explained in Chapter 1, Section 1.2

[3]The theoretical background of Betti numbers is explained in Chapter 6, Section 6.1. The computation of Betti numbers on data is performed in Chapter 7, Sections 7.1 (random networks), 7.2 (simulated data), 7.3.1 (Betti-1 curves on HC and Glioma patients)

[4]Euler characteristic will be explained in detail in Chapter 6, Section 6.3. We will apply it to experimental data in Chapter 7, Section 7.3.2

Chapter 1

Network science

In this first chapter, we will give the bases of graph theory (Section 1.1 and Section 1.4) and explain how it can be used to model brain networks. We will give an overview of the research context where this thesis work is inserted (Section 1.2) and present our research hypothesis (Section 1.3).

1.1 Graph theory: modelling the brain

In mathematics, graph theory is the study of graphs, which are mathematical structures often used to model pairwise relations between objects.
A graph is composed of *vertices* (also called nodes or points) which are connected together by *edges*. We can make a distinction between *directed graphs*, where edges connect two vertices with an orientation, and *undirected graphs*, where edges connect two vertices without an orientation. In Fig. 1.1 we have an example of an undirected and directed graph composed of 10 nodes and 13 edges.

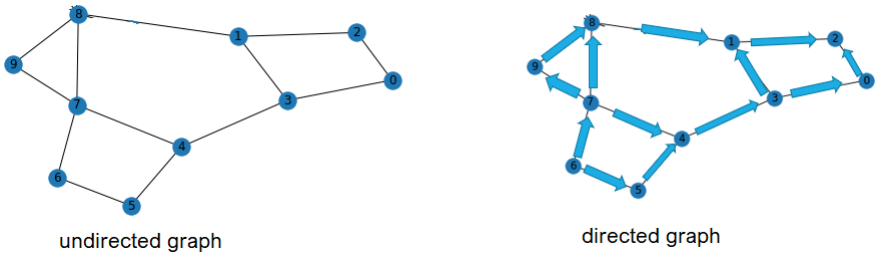


Figure 1.1. Example of undirected and directed graphs.[1]

We can formally define an undirected graph as $G = (V, E)$, consisting of the set V of vertices (or nodes) and the set E of edges $E \subseteq \{(x, y) \mid x, y \in V \text{ and } x \neq y\}$, which are unordered pairs of elements of V. In contrast we can define a directed graph as an ordered pair $G = (V, E)$, consisting of the set V of vertices and $E \subseteq \left\{(x, y) \mid (x, y) \in V^2 \text{ and } x \neq y\right\}$ which are ordered pairs of vertices.
Another distinction we can make is between *weighted* and *unweighted* graphs. A weighted graph is a graph in which each edge has a numerical weight: this can be

[1] In this work, when not otherwise stated, the figures are created by the author of the thesis.

interpreted as the strength of connections. In a weighted graph, not every edge counts the same; some edges are more important than others. In contrast, an unweighted graph is a graph where all the edges count the same, i.e. there is no distinction between the weights of the connections. It is a binary network where two nodes can be either connected (1) or disconnected (0). In Fig. 1.2, we show an example of a weighted and unweighted graph.

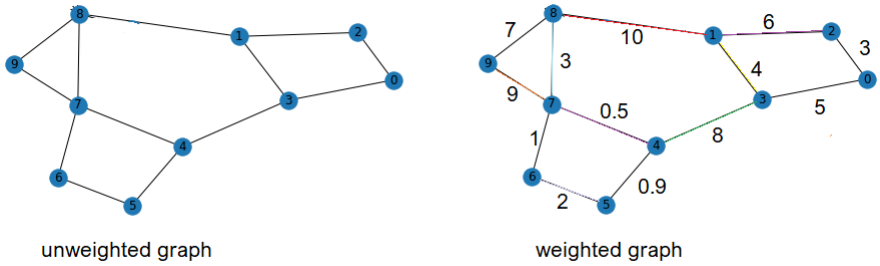


Figure 1.2. Example of weighted and unweighted graphs.

The connectivity of the human brain, also known as *human connectome*, is usually represented as a graph consisting of nodes and edges connecting the nodes. The nodes are mainly predefined anatomical regions of interest (ROIs). The edges are determined by various techniques such as correlation methods, structural equation modelling, or dynamic causal modelling [2]. The brain is, therefore modelled as a graph. Each brain network is represented by an NxN matrix C_{ij} of correlations between regions i and j. Functional brain networks are constructed by quantifying pairwise correlations between time series[2] of brain activity recorded from different regions in an atlas spanning the entire brain [3]. In Fig. 1.3 we summarize what we discussed so far.

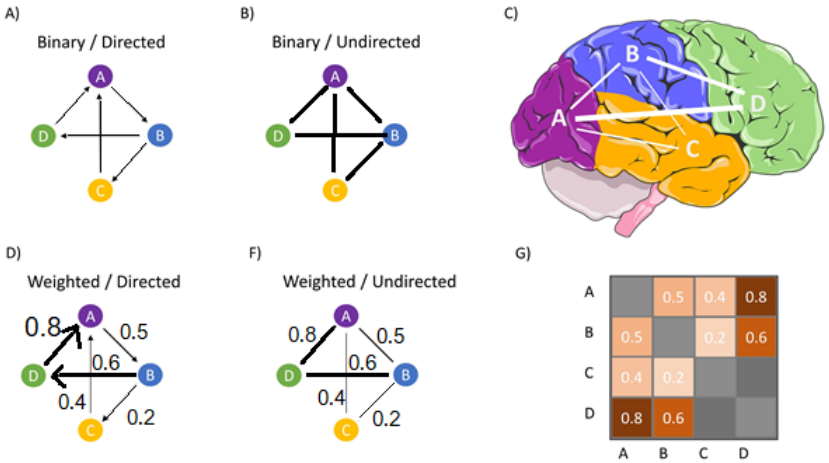


Figure 1.3. Example of directed/undirected weighted/unweighted graphs and matrix representation of the correlations between regions [4].

[2]Time series are explained in Section 3.1

After having obtained a connectivity matrix C_{ij} with the correlations measure between regions, we need to create a model of our brain network. If we do not threshold the correlation at a certain level (saying that two nodes are connected only if their correlation is greater than ϵ), we obtain a weighted network[3]. Since the weighted brain network is more difficult to be interpreted and visualized compared to the binary network, the binary brain network[4] has been more often used in the early stages of network neuroscience. In this model, researchers can establish a threshold to select only some connections, e.g., only the 20% strongest connections, and therefore a binary matrix representing that unweighted network is constructed. However, depending on where to threshold the correlation, the binary network changes, and a different system is analyzed.

One of the main problems in network neuroscience is exactly this: there are not fixed and accepted criteria to choose the threshold and choosing a threshold might cause loss of important information. In this thesis work, we will use a recent approach to overcome this problem.

1.2 Data collection techniques and research context

The brain is a complex system and its representation using graph theory is strongly related to how the data are processed. There are several ways to create a network based on data collection. In this section we will first briefly present some data collection techniques, then we will present different ways to model the brain and give an overview on the scientific context where our research question is inserted.

To build the connectivity matrix containing the correlations between the different regions of the brain we use time series obtained from experimental data collections. To obtain the time series, we can use different methods such as Electroencephalography (EEG), Magnetoencephalography (MEG), or Functional magnetic resonance imaging (fMRI). EEG is an electrophysiological monitoring method to record the electrical activity of the brain over a period of time; it measures voltage fluctuations resulting from ionic current within the neurons of the brain recorded thanks to electrodes placed on the scalp. MEG is a functional neuroimaging technique for mapping brain activity by recording magnetic fields produced by electrical currents occurring in the brain; MEG uses very sensitive magnetometers and gradiometers. fMRI measures brain activity by detecting changes associated with blood flow; this technique relies on the fact that cerebral blood level of oxygenation and neuronal activation are related: when an area of the brain is used for a task, the level of oxygenation in that region increases.

Graph theory can be used with different approaches. One way is to consider the whole brain network as composed of sub-networks (representing different 'distributed

[3]The matrix representing the brain stores the exact values of connection strength between the nodes, not only binary values.

[4]The connections in the brain are represented with a binary matrix. If there is a connection between two nodes the value in the matrix is 1 otherwise is 0.

subsystems' of the brain). We know that to complete a cognition process (or specific functions, like information processing, memory, etc.) the different regions of the brain strongly cooperate and we look at this process as cooperation and connection between the different sub-networks. In clinical brain science is crucial to understand how the different networks in the brain are connected and to associate the identified connections with a diagnostic status in case-control studies. Analyzing functional MRI data, researchers have identified a disconnected sub-network in people affected by schizophrenia [1]. Therefore, examining how sub-networks interact in a healthy or unhealthy brain, we can better understand the role of connections between networks in the brain.

Another approach in graph theory is to compare different brain networks as a whole, without dividing them into sub-networks. To compare two networks, traditionally we determine the difference between graph-theoretic measures such as assortativity, betweenness centrality, clustering coefficient, characteristic path length, small-worldness, modularity, and network homogeneity, which are central to understand the characteristics of a network. Those network characteristics can provide a useful quantitative description of networks, and we can find common traits across diseases. The analysis of the networks can reveal a malfunctioning of a cognition process which is a symptom of a particular disease. We can then compare the measures of the networks in a healthy brain to those of a brain with a disease to better understand where the differences are.

It is essential to find a method to compare the brain networks in patients with different diseases. Measuring the distance between these networks can be a way to understand how much they differ. In general, distance can be computed with different types of metrics depending on the problem we are considering. In recent research, to compute the distance between two topological spaces various metrics have been proposed. For example, Bottleneck distance [2] is often used, but it can be inadequate to compute the distance between networks of different type (as can be the case in the human brain). We need, therefore, to choose an adequate metric to compare brain networks.

1.3 Research question: abstract topology meets network neuroscience

The standard network methodologies are not sufficient to address the challenges that are currently existing in clinical network science. Therefore there is an increase in the use of an abstract methodology in network neuroscience. The goal of this thesis was to investigate the power and robustness of these techniques in clinical neuroscience. Therefore we developed two research ideas that use abstract mathematics in a practical way: we first used topological distances (such as Gromov-Hausdorff distance) in clinical neuroscience, and then we investigated TDA methods as I will explain below.

The initial goal of this project was to find an adequate metric to compute distances between brain networks and try to find suitable methods to compare brain networks of healthy controls and patients with a brain disease. The idea is that with a good metric, we could compare the measures on brain network for the same individual at different times. Moreover, we can compare measures between healthy control and patients with brain diseases and use distances to differentiate patients with different diseases. We started from the abstract notion of distance, and we analyzed two different types of distances: the Gromov-Hausdorff and the Gromov-Wasserstein, which can be used to compute networks distance. Gromov-Hausdorff is a distance used to calculate distances between sets of points, and since we can think of our network as such a set, it may be an adequate metric. The Gromov-Wasserstein distance can be useful when we need to compare networks with a different number of nodes (in this case, Gromov-Hausdorff is not adequate).

The research hypothesis was that the distances measured between brain networks of the same individual taken at different times would be smaller than the distances between brain networks of different individuals. This is because the networks of the same person should be similar. Moreover, networks of brains taken from a group of patients with the same disease will be similar among them. So the distance between them will be shorter than the distance from brain networks taken from a group of patients with a different disease. We can therefore use distances to differentiate groups of patients. To explore these distances, we can use both simulation and experimental data. We first measured distances in random networks created with computer simulation (using Python). Then we used experimental brain data obtained from fMRI measures and MEG measures.

As we will explain in the following chapters the proposed metrics (Gromov-Hausdorff and Gromov-Wasserstein), which are interesting from a theoretical perspective, were not adequate to reach our goal of differentiating the groups of patients at a clinical level. We, therefore, adopted an alternative approach using Topological Data Analysis (TDA) consisting in the analysis of the *Betti numbers* to differentiate groups of patients. With this new method, we were able to reveal differences between healthy controls and patients with a disease, and therefore we reached our initial goal. Later, we narrowed our goals to a more ambitious question: whether the topological properties of the network relate to clinical traits of the patients such as the grade of the tumour and other markers. Unfortunately, no correlations were found.

1.4 Graph theory measures

In this section, we refer to some measures which characterize a network. We will explain in detail only the *betweenness centrality* since, in our analysis, this is the only measure we take into account to perform a comparison with the new proposed Gromov-Hausdorff metric (Section 5.1.2).

In graph theory, betweenness centrality is a measure of the importance of a given

node in the graph based on shortest paths. For every pair of vertices in a connected graph, it 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[5]. The betweenness centrality for each vertex is a measure related to the number of these shortest paths that pass through the vertex. To compute the betweenness centrality of a node i, we calculate the proportion of shortest paths between nodes j and h that pass through i as

$$C_B(i) = \frac{1}{(N-1)(N-2)} \sum_{h \neq i, h \neq j, j \neq i} \frac{\rho_{hj}(i)}{\rho_{hj}}, \tag{1.1}$$

where $\rho_{hj}(i)$ is the number of shortest paths between h and j that pass through i, ρ_{hj} is the number of shortest paths between h and j, and $(N-1)(N-2)$ is the number of node pairs that does not include node i. The normalization by ρ_{hj} accounts for the possibility that multiple shortest paths may exist between any pair of nodes. The betweenness centrality measures the proportion of shortest paths between all node pairs in the network that pass through a given index node. As the name implies, we can think of this measure as indicating the extent to which a node lies "between" other pairs of nodes. Suppose that the information travels through a network along the shortest path. In this case, nodes that lie on many shortest paths will mediate a high proportion of traffic, and thus represent central elements of the network. In this sense, such a node might play a controlling role in the passage of information through the network or act as a traffic bottleneck [5]. In Fig. 1.4, we can visualize betweenness centrality in a simple network.

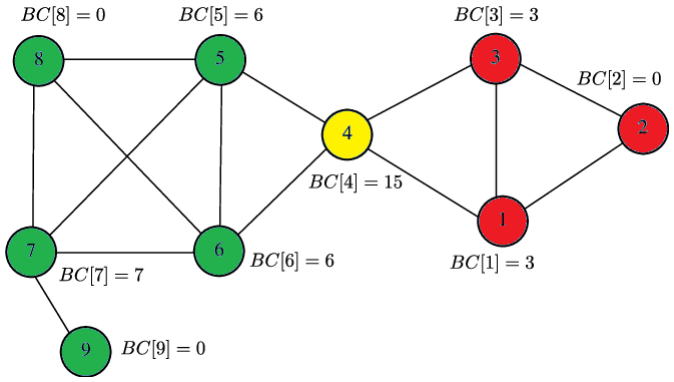


Figure 1.4. Betweenness centrality in a simple network: node 4 (yellow), has the maximum value of betweenness due to its role of linking the green and red sub-networks. Indeed, all the paths connecting both sub-networks must go through it. Figure adapted from [6].

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

There are other measures to characterize a graph such as clustering coefficient, characteristic path length, small-worldness, degree centrality etc. In Fig. 1.5 some of these quantities are summarized.

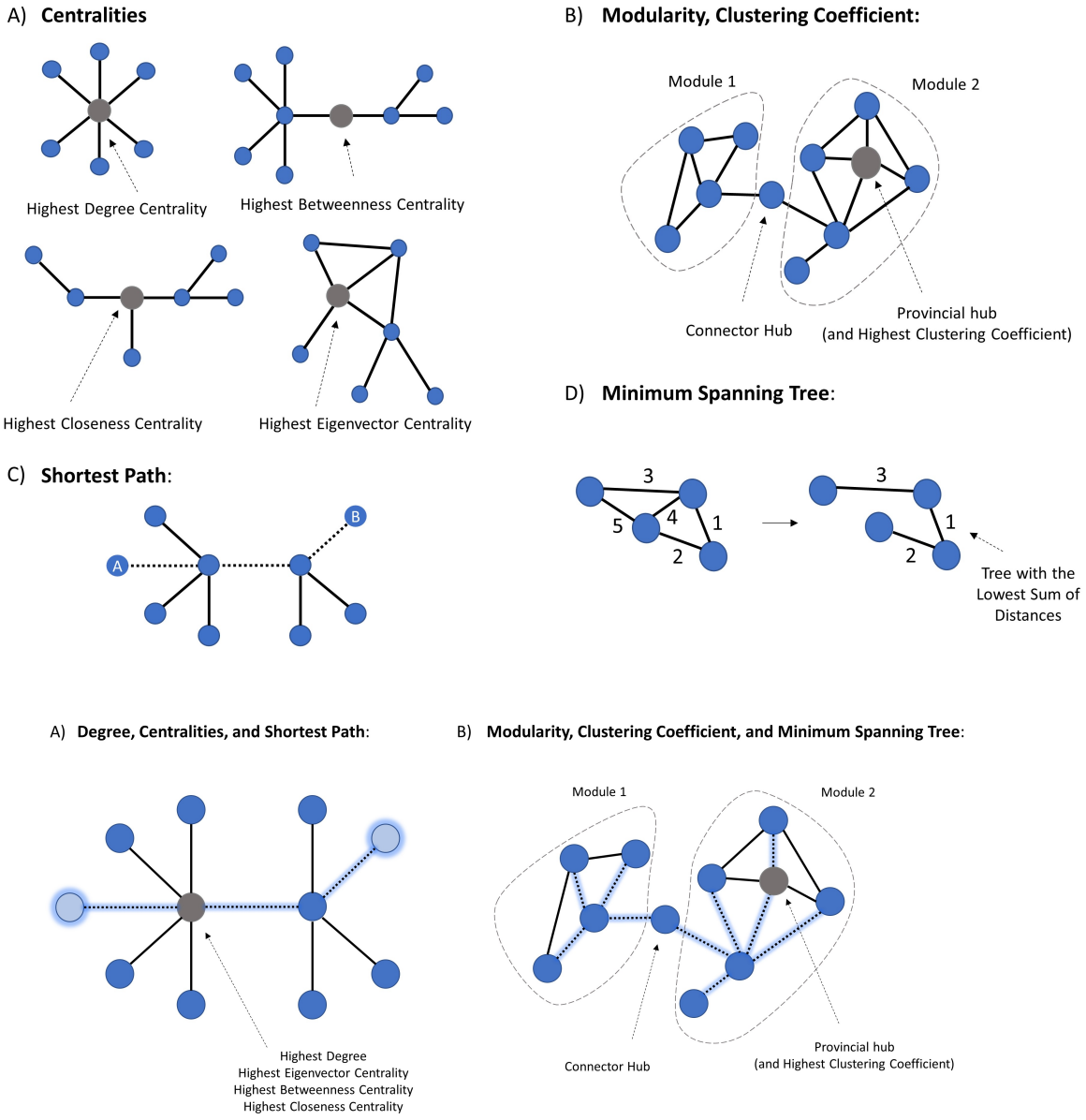


Figure 1.5. Graph theory measures [4].

In this work, we are not going to compute these properties to characterize the networks since, to study networks, we will use a different approach based on TDA.

At the local level, graph theory also enables the characterization of individual nodes and their contributions to the network's organization. For example, a special class of nodes commonly referred to as 'connector hubs' plays a particularly important role in integrating information across the system by accessing different modules and coordinating their activity in both resting and task states [7]. Such nodes are typically characterized by a combination of standard local graph metrics including high degree-centrality (i.e. the total number of connections) as well as elevated scores on the betweenness centrality and participation coefficients measures. Damage to brain regions important for communication between sub-networks (but not to those brain regions important for communication within sub-networks) leads to global changes in the modular organization of the brain, and a loss of small-world character[6]. Lesions to provincial hubs, therefore, are expected to yield specific clinical deficits, while damage to connector hubs may result in pervasive cognitive and/or behavioural impairments extending across several cognitive domains [7]. Fig. 1.6 gives an overview of graph measures and the importance of connecting hub.

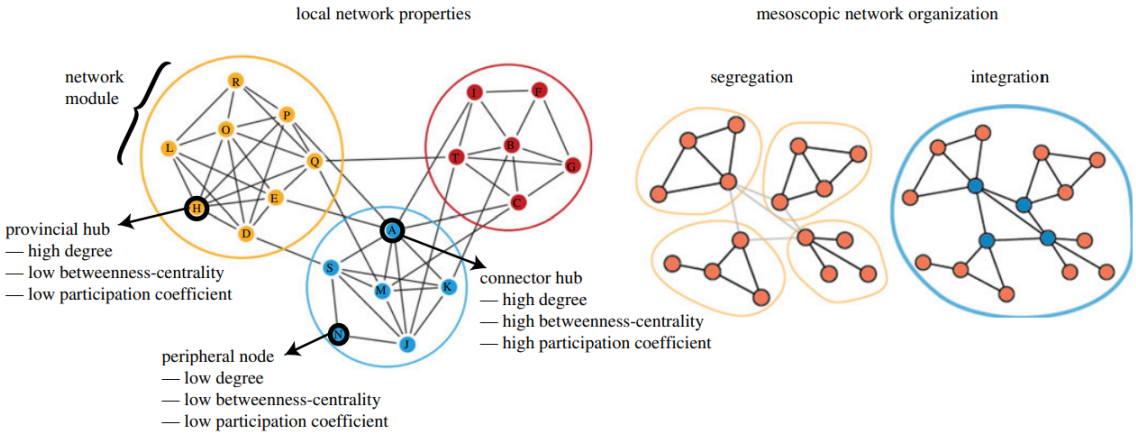


Figure 1.6. Local network properties and mesoscopic network organisation [7].

[6] A small-world network is a type of graph in which most nodes are not neighbours of one another. Still, the neighbours of any given node are likely to be neighbours of each other, and most nodes can be reached from every other node by a small number of steps.

Chapter 2

Theoretical background: the concept of distance

Since the first part of this work is based on the Gromov-Hausdorff distance and Wasserstein distance, we want to define the concept of distance with precise mathematical notation. In this chapter we will start from the general definition of *distance* and *metric* (Section 2.1) and we will then define different types of distances: *point to point* distance (Section 2.1.1), *point to set* distance (Section 2.1.2) and *set to set* distance (Section 2.1.3). The *set to set* distance is the base of our work. In fact, networks can be considered as a set of points and, in order to properly compare brain networks under this perspective, we need to define *set to set* distance.

2.1 Distance

When we consider a distance between two quantities, we can refer to different definitions of distance. The concept of distance, in fact, is not only reducible to the well-known Euclidian distance between two points. In this work, we will consider the distance between networks, and we will try to find the best way to define their distance in order to compare them.

Several types of distances can be defined, but all definitions must satisfy some distance axioms. First let's start by the definition of the terms *distance* and *metric*. This definitions are taken from *Dictionary of Distances* [8].

Definition 1 *Let X be a set. A function $d : X \times X \longrightarrow \mathbb{R}$ is called* ***distance*** *(or* ***dissimilarity****) on X if, for all $x, y \in X$, it holds:*

1. $d(x, y) \geq 0$ *(non-negativity);*
2. $d(x,y) = d(y,x)$. *(symmetry);*
3. $d(x,x) = 0$.

Definition 2 *Let X be a set. A function $d : X \times X \longrightarrow \mathbb{R}$ is called* ***metric*** *on X, if for all $x,y,z \in X$, the following conditions are satisfied:*

1. $d(x,y) \geq 0$, *and* $d(x,y) = 0$ *if and only if* $x = y$;

2. *d(x,y) = 0 if and only if x = y;*
3. *d(x,y) = d(y,x) (symmetry);*
4. *d(x,z) ≤ d(x,y) + d(y,z) (**triangle inequality**).*

As we see, Definition 2 has one extra requirement, the so-called triangle inequality and the second requirement is stricter than the combination of the first and the third requirement of Definition 1.

Example 1. *Not every distance is a metric. Let $X = \{x_1, x_2, x_3\}$ and the function $d : X \times X \longrightarrow \mathbb{R}$ such that after computing $d(x_i, x_j)$ for every i,j = 1,2,3 we get a matrix:*

$$d_X = \begin{pmatrix} 0 & 1 & 6 \\ 1 & 0 & 3 \\ 6 & 3 & 0 \end{pmatrix}.$$

It is easy to see that function d is a distance. However d is not a metric because triangle inequality is not satisfied:

$$6 = d(x_1, x_3) > d(x_1, x_2) + d(x_2, x_3) = 4.$$

Now we will look at distances/metrics defined between different data structures. In particular, we are analysing the *point to point*, *point to set* and *set to set* distances.

2.1.1 Point to point

The *point to point* distances are distances defined between two points. The most known point to point distance is the Euclidean distance. It is the "ordinary" straight-line distance between two points in Euclidean space.

Definition 3 *The **Euclidean distance** or **Euclidean metric** is the function $d : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ that assign to any two vectors in the Euclidean n-dimensional space $\boldsymbol{x} = (x_1, x_2 \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2 \ldots, y_n)$ the number:*

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

Another example of point to point distance is the *Manhattan* distance.

Definition 4 *The **Manhattan distance** between two vectors $\boldsymbol{x} = (x_1, x_2 \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2 \ldots, y_n)$ is equal to the one-norm (l_1-metric on $\mathbb{R}^n$) of the distance between the vectors.*

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |(x_i - y_i)|.$$

This metric is also called *city-block* metric.

In Fig. 2.1 we can visualize the city-block metric in two dimensions. The city-block metric in two dimension is defined as

$$\|x - y\| = |x_1 - y_1| + |x_2 - y_2|.$$

So, all the paths in Fig. 2.1 have the same Manhattan distance.

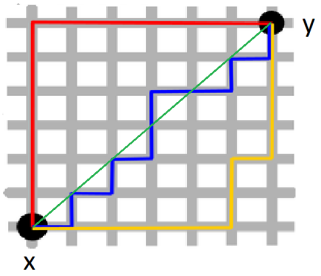


Figure 2.1. Illustration to compare different city-block metric (yellow, blue, and red paths) with the Euclidean distance (green path). In the city-block metric all three pictured paths (red, yellow and blue) have the same length.[1]

Here we introduced two examples of point to point metrics but, as long as the metric fulfils the axioms defined early, the list of metrics is endless. The choice of the metric depends on the problem we want to solve. Moreover, the point to point metric is not useful in many situations, and we need to define distances between different structure as is the case of brain networks distances.

2.1.2 Point to set

We will now define the *point to set* distance. Let's first define a structure called *metric space*.

Definition 5 *A **metric space** (X,d) is a set X equipped with a metric d.*

Example 2. *Let* $X = \{x_1, x_2\}$ *and let* $d_X = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$ *be a metric on X. Then* (X, d_X) *is a metric space.*

Definition 6 *Given a metric space* (X, d)*, the **point-set distance*** $d(x, A)$ *between a point* $x \in X$ *and a subset* A *of* X *is defined as*

$$\inf_{y \in A} d(x, y).$$

For any $x, y \in X$ *and for any non-empty subset* $A \in X$ *we have the following version of triangle inequality:* $d(x, A) \leq d(x, y) + d(y, A)$.

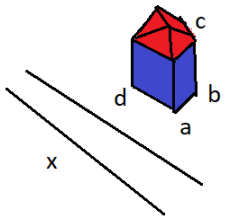
Example 3. *Let's look at an example of point to set distance where we want to know how far is a house from some point in the sidewalks. Let* $X = \{x, a, b, c, d\}$*, where x is a point* $\in X$ *(point in the sidewalk) and a,b,c,d* $\in A$ *are points of the set (defining the house). We have first to decide which metric to use. Lets use Euclidean metric such that we get*

$$d_X = \begin{array}{c} \\ x \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{ccccc} x & a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 30 & 37 & 38 & 28 \\ 30 & 0 & 10 & 14 & 10 \\ 37 & 10 & 0 & 10 & 14 \\ 38 & 14 & 10 & 0 & 10 \\ 28 & 10 & 14 & 10 & 0 \end{pmatrix} \end{array}.$$

[1] https://en.wikipedia.org/wiki/Taxicab_geometry.

Now we have the metric space (X, d_X)*. Following Definition 6 we get the point-set distance*

$$d(x, A) = min\{d(x, a), d(x, b), d(x, c), d(x, d)\} = 28.$$



We see that in this discrete case the infimum of Definition 6 was replaced by the minimum.

2.1.3 Set to set

We now define the *set to set* distance which is the distance we are most interested in since it can be used to measure the distance between brain networks.

Definition 7 *Given a metric space* (X, d)*, the* ***set to set distance*** *between two subsets* A *and* B *on* X *is defined by*

$$\inf_{x \in A, y \in B} d(x, y).$$

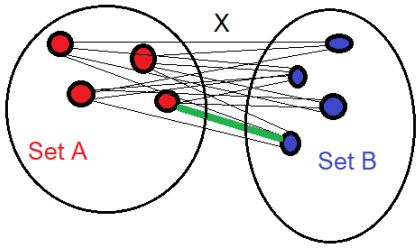


Figure 2.2. Visualisation of a *set to set* distance. The chosen distance d in this metric space (X, d) is the simple Euclidean distance. The green line corresponds to $\inf_{x \in A, y \in B} d(x, y)$.

Notice that *point to set* distance is a special case of *set to set* distance where the set A has just one point. One example of set to set distance is the *Hausdorff* distance.

Definition 8 *Let* X *and* Y *be two non-empty subsets of metric space (M,d). The* ***Hausdorff distance*** $d_H(X, Y)$ *is defined as*

$$d_H(X, Y) = max\left\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\right\}.$$

Fig. 2.3 explains the concept of Hausdorff distance between two sets (the green line X and the blue line Y) visually.
Two sets are close under the Hausdorff distance if every point of either set is close to some point of the other set. The Hausdorff distance is the greatest of all distances from a point in one set to the closest point in the other set.

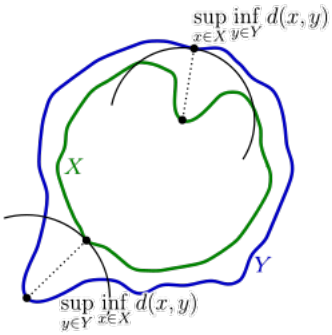


Figure 2.3. Visualisation of the Hausdorff distance.[2]

The *set to set* distance is the basis of our work and is what we will use to compare brain networks. In the next chapter, we will give the definition and explain in detail the two distances used in this work: the Gromov-Hausdorff and Gromov-Wasserstein distances.

[2] https://commons.wikimedia.org/wiki/File:Hausdorff_distance_sample.svg.

Chapter 3

Mathematical tools to investigate brain networks

In the first part of this chapter (Section 3.1), we will introduce a method to study brain networks using the idea of distance discussed in the previous chapter. This method is taken from the research *Persistent Brain Network Homology from the Perspective of Dendrogram* [2]. In the next chapter, we will reproduce the results of the referred research, which will be the starting point of our thesis work.
In Section 3.2 we will explain the algorithm used for the analysis and in Section 3.3 we will define the concept of Gromov-Hausdorff distance (GH) and Gromov-Wasserstein distance (GW).

These distances will be used in Chapter 4 to measure the difference between networks of simulated data (Section 4.1 and Section 4.4). We will then apply the studied method to a publicly available database of the Brain Connectome Project's website (Chapter 5, Section 5.1). Later We will use the learned techniques to investigate a database of data of the Amsterdam UMC (Chapter 5, Section 5.2).

3.1 The problem of thresholding a brain network

The usual approach to construct a brain network is by estimating the connectivity matrix and thresholding it at an arbitrary level. The problem with this standard method is that there are no accepted criteria to choose the threshold [2]. Moreover, the simple fact of choosing a threshold might cause to loose important information.

In the paper *Persistent Brain Network Homology from the Perspective of Dendrogram* [2], the authors proposed a multiscale framework that considers all brain networks generated over every possible threshold. We will explain in the following section how this approach is built and how it works.

In the referred paper, the authors investigated FDG-PET[1] functional brain networks of

- 24 ADHD (attention-deficit hyperactivity disorder children)
- 26 ASD (autism spectrum disorder children)
- 11 PedCon (pediatric control children).

The authors found that GH can differentiate the populations better than most available graph theoretic approaches.

The data we used to test the techniques of the paper were connectivity brain networks based on resting state (rs) fMRI measures. As we will discuss later, despite the difference from the database of the paper, we were able to reproduce the proposed techniques and find similar results.

We will now start by defining how the data is organized in connectivity matrices. We consider the measurements obtained in p selected regions of interest (ROIs) in n subjects. Each ROI represents a node in the brain network. In the paper they have the FDG-PET measurement x_i at i-th node. The measurement set is denoted as $X = \{x_1, x_2, ..., x_p\}$. We define the distance c_X between the measurement x_i and x_j through the Pearson correlation

$$c_X(x_i, x_j) = 1 - corr(x_i, x_j). \tag{3.1}$$

Functional brain network are represented using measurement set X and the distance c_X, which form the metric space (X, c_X) [2]. An example of this representation is illustrated in Fig. 3.1.
The Pearson correlation is defined in the interval $[-1, 1]$. We can therefore have a negative or positive correlation, being 0 no correlation, 1 maximum positive correlation, and -1 maximum negative correlation. This value is a measure of how much two nodes are correlated for the time slice of the data collection. To assign the correlation, we look at the time series of the two considered nodes; if the two curves are simultaneously increasing (decreasing) the two nodes are positively correlated whereas, if one curve is increasing while the other is decreasing the two nodes are negatively correlated.[2]

[1] fluorodeoxyglucose (FDG)-positron emission tomography (PET) measures glucose metabolism, which is associated with neuronal activity. The interregional metabolic correlation between brain regions was used to reflect functional connectivity during the resting state. While the resting-state fMRI records the blood oxygenation level-dependent (BOLD) signal every 2 or 3 s, FDG-PET records the FDG uptake for 30 min after 20 min from the injection. Thus, FDG-PET data at the resting state is more stationary and invariant to the noise compared to fMRI studies [2].

[2] for a more detailed explanation on time series see Ref. [11].

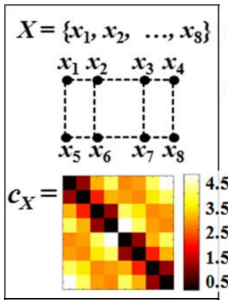


Figure 3.1. Representation of a network and correspondent connectivity matrix. We have a network composed of 8 nodes connected by edges as in the figure. The measurement set is denoted as $X = \{x_1, x_2, ..., x_p\}$ and the correlation matrix c_X stores the information of how strong is the correlation between each pair of nodes. In this example, the "strength" of connection is determined by the Euclidean distance between the nodes. Smaller values in the matrix indicate that the nodes are nearer and so more connected. Figure adapted from [2].

Previous studies on brain network modelling used a single fixed threshold ϵ.

Definition 9 *We connect the nodes i and j with an edge if the distance $c_X(x_i, x_j) \leq \epsilon$ for some threshold ϵ. The collection of all those edges is denoted as E. The binary network $B(X, \epsilon)$ at threshold ϵ is a graph consisting of the node-set V and the edge set E* [2].

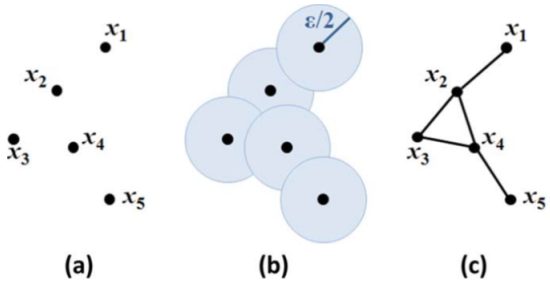


Figure 3.2. Construction of binary network $B(X, \epsilon)$. (a) Point cloud data. (b) The balls of radius $\epsilon/2$ centered at each point. (c) The binary network. Adapted from [2].

In the proposed framework [2] we use several different thresholds: when we change the threshold, we obtain a nested sequence of networks. $B(X, \epsilon_1), B(X, \epsilon_2) \ldots$. Let's see the relation between these networks: when ϵ increases, the subsequent network becomes larger (with more connected nodes) than all the previous created networks. We have the network filtration:

$$B(X, \epsilon_1) \subseteq B(X, \epsilon_2) \subseteq B(X, \epsilon_3) ..... \subseteq B(X, \epsilon_n) \quad \text{for} \quad \epsilon_1 \leq \epsilon_2 \leq \epsilon_3 .... \leq \epsilon_n.$$

Each brain network is represented by its binary network $B(X, \epsilon)$. The sequence of networks represents all the networks obtained at different thresholds in the network filtration. In Fig. 3.3 an example of these concepts is shown.

Definition 10 *A connected component in a network is a subset of the network where any two nodes are connected to each other by paths.*

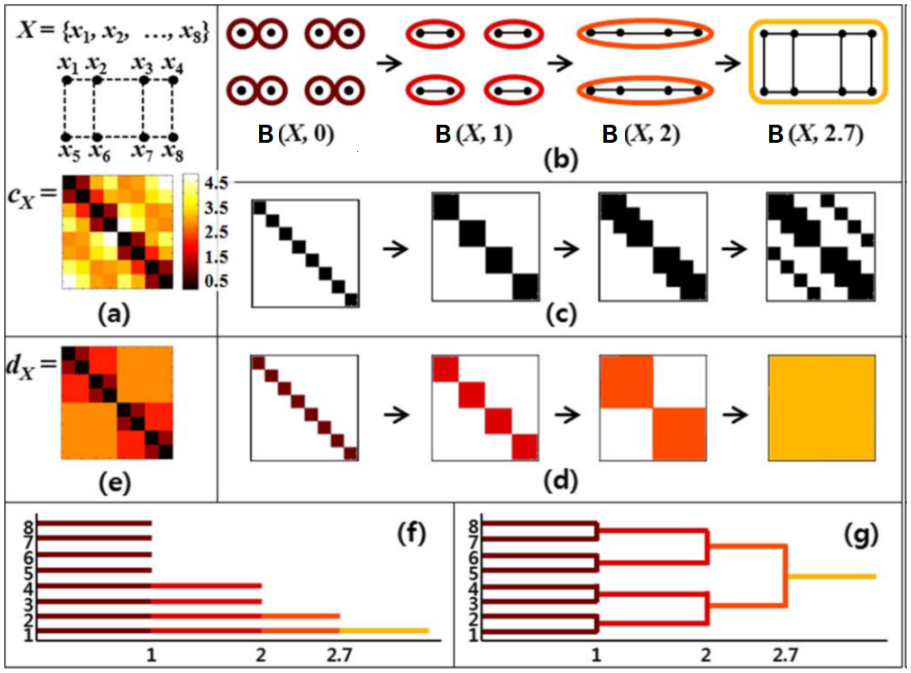


Figure 3.3. Multiscale network modelling: barcode and dendrogram. (a) Node set and metric. (b) The network filtration at the filtration values 0, 1, 2, and 2.7. (c) The corresponding adjacency matrices of the network filtration. (d) The connected component matrices representing the connected components through different colours. (e) The single linkage distance. (f) Barcode: visualization of the topological changes in the network. The number of connected components β_0 as a function of the filtration values. (g) The dendrogram (SLD). If we add the information of the node indexes and connect them following how new connections were introduced in the network filtration, the barcode (f) is transformed to SLD (g) [2].

Let's take a simple example of a network composed of 8 nodes, as shown in Fig. 3.3 (a). In Fig. 3.3 (b) we can see that, at each filtration value, the connected components are marked with a circle of a different colour. When the filtration value increases, more nodes connect together: we go from 8 separated components to one big connected component. The topological changes occurring in the network with the increment of the filtration value can be visualized using the *barcode*. There, we plot the topological changes over the different filtration values. In the y-axis there is the so-called zero*th* Betti number, β_0, which is the number of connected components in the network. The barcode is a decreasing function showing *when* (at which filtration value) the connected components are merging to form bigger connected components Fig. 3.3 (f).
If we add the information of *where* the changes occur (the node indices: which nodes are connecting at a given filtration node), we can transform the barcode into a dendrogram, which shows in a rich way how a brain network changes with the increment of the filtration value Fig. 3.3 (g).
The barcode shows the global topological characteristics of when the components are merged (at which filtration value). In contrast, the dendrogram shows the local network characteristics of which sub-networks are clustering together.

We will now explain how the components are merged together. Let's consider the network filtration $\{B(X, \epsilon_k)|k = 0, 1, \dots\}$. Let S_m^k and S_n^k be the two disconnected components of the network $B(X, \epsilon_k)$. Suppose that there are two nodes $x_i \in S_m^k$ and $x_j \in S_n^k$ such that the distance between x_i and x_j is less than the next filtration value ϵ_{k+1} : $d(x_i, x_j) < \epsilon_{k+1}$. Then S_m^k and S_n^k will be connected at ϵ_{k+1}. Summarizing it in a formula, we can say that the clusters S_m^k and S_n^k will be connected at ϵ_{k+1} if

$$d(S_m^k, S_n^k) = \min_{x_i \in S_m^k, x_j \in S_n^k} d(x_i, x_j) < \epsilon_{k+1}, \tag{3.2}$$

where $d(S_m^k, S_n^k)$ is the *single linkage distance* between the clusters S_m^k and S_n^k and is stored in the *single linkage matrix* d_X (explanation adapted from [2]). In Fig. 3.4 there is a visual example of the merging of two connected components. The sequence of how the connected components merge during the network filtration

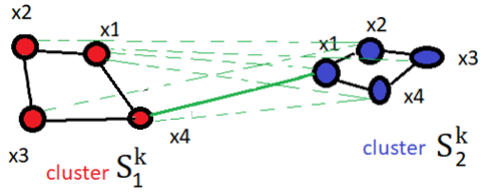


Figure 3.4. Merging of two connected components S_1 and S_2. The connection will occur if the green distance $d(x_4, x_1) = \min_{x_i \in S_1^k, x_j \in S_2^k} d(x_i, x_j)$ is smaller than ϵ_{k+1} : $d(x_4, x_1) < \epsilon_{k+1}$.

corresponds to the sequence of the clustering in the single linkage dendrogram and is visualized in the dendrogram. This sequence is determined by the matrix of the single linkage distance. The filtration value ϵ_{k+1} at which the two clusters S_m^k and S_n^k are merged is determined by $d(S_m^k, S_n^k)$.
The single linkage matrix in Fig. 3.3 (e) is basically a matrix where the distances between nodes are no longer based on the definition of the distance of the connectivity matrix c_X: we have redefined the distances between nodes using the distance between clusters given by $d(S_m^k, S_n^k)$. We can explain in more detail how this is mathematically defined. Let $x_i = w_0, ..., w_k = x_j$ be a path between x_i and x_j. The *single linkage distance* d_X between the two nodes is given as

$$d_X(x_i, x_j) = min\{\max_{l=0,\dots,k-1} c_X(w_l, w_{l+1})|x_i = w_0, ..., w_k = x_j)\}. \tag{3.3}$$

This means that, for each considered path between x_i and x_j, we look at all the distances (defined by c_X) between any two nodes of the path $c_X(w_l, w_{l+1})$. We later compare them and store the maximum value among them. Finally, we look all the paths (where for each of them we have chosen a maximum value of distance) and we take the minimum of all these maximum values. The minimum is taken over every possible path between x_i and x_j. This is now the definition of distance between node x_i and x_j. This procedure is summarized in Fig. 3.5. The founded distance $d(x_i, x_j)$ is stored in a matrix. The same procedure is followed for all pair of nodes in the network in order to build the *single linkage matrix* (SLM) d_X.

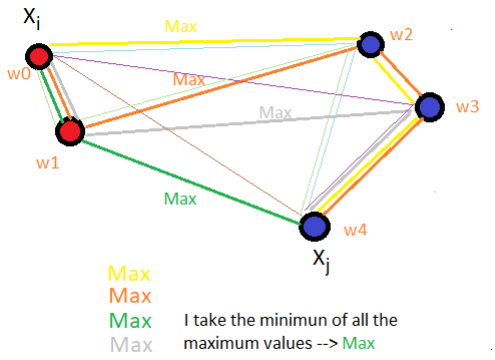


Figure 3.5. Procedure to find a definition of distance between node x_i and x_j.

Fig. 3.3 (e) shows the SLM d_X, which can be decomposed into the sequence of matrices representing the connected components. In Fig. 3.3 (d) the points of the matrix with the same colour correspond to the nodes which are members of a connected component of the same order of filtration.

3.2 Python algorithm: find d_X from c_X

In order to count the number of connected components with the increment of the filtration value, we have built a Python algorithm. We will give here a general explanation on how this algorithm is constructed, and in Appendix A.2 the complete code with detailed comments is given.

We start from the c_X matrix where the values of strength of connections between nodes are stored, and we move step by step toward the construction of the single linkage matrix d_X. As explained in Section 3.1, we are giving a new definition of distance, and for each filtration value, we look at how the connections have shaped the network and how many clusters we have. At each step of the algorithm, when we look for the next filtration value (which is a value in the matrix c_X), new nodes are added to the existing clusters, and two or more clusters can merge together. Starting from the c_X matrix, at the end of our process, we will end with the d_X matrix. Changes are done step by step by the algorithm: the nodes are added to the clusters one at a time.
At first, we initialize three different lists that at the end will contain:

1. the filtration values at which a change in the network occurs (*a_values*, line 5 in the code A.2)
2. the number of connected components at each step: from 100 different components to 1 giant component[3] (*connected_comp*, line 6 A.2)
3. the different clusters, called connected components (*list_clusters*, line 7 A.2). Each cluster is a list containing the nodes which are part of that particular cluster.

[3]the giant component is a graph where all the nodes are part of the same connected component.

In the beginning, we have N different clusters (in our case N=100) composed each by one node, and at the end of the process, we will have just one big list containing all the nodes. To do so, we start a loop (line 16 A.2), and we continue the process until all the values in the c_X matrix are substituted with the values corresponding to the orders of the different clusters (the new definition of distance between nodes).

In the loop, we start from the smallest value (order) a in the matrix, we look for the nodes in the matrix which have a connection of value a, and we save each couple in an array (*array_positions*, line 20 A.2). Then for each couple, we look in which cluster the two nodes $node_1$ and $node_2$ are contained (for loop, line 23 A.2). If the two nodes belong to two different clusters, we can connect the two clusters. We can ,therefore, connect all the points in the two clusters with the same order of connection a because they now belong to the same cluster (definition of d_X). Practically, what we do is to replace with a, in the matrix, the values regarding the connections between all those nodes (lines 37, 38 A.2). We have therefore merged the two clusters and modified the *list_clusters*, which contains all the different clusters (lines 44, 45 A.2). We do this procedure for all the couples of nodes connected with a (all the couples in *array_positions*). The number of connected components for this analyzed a is then the length of the modified list *list_clusters* (line 48 A.2). We now append to the filtration value list the filtration value a we were using (*a_values*, line 49 A.2). We repeat the same procedure for the next filtration value we have in the matrix (which is the minimum value greater than a).

At the end of the process, all the values in the matrix will be substituted (the nodes in the same cluster will have the same value of connection) and we obtain our d_X matrix. With the last filtration value of the process, all the clusters will merge together, forming one big connected component. Now in the two lists *a_values* and *connected_comp* we have all the information to plot the topological changes of the network with the increment of the filtration value. We can visualize these changes in the barcode.

In Chapter 4 we will apply the algorithm to simulated data and see the results. Here (Fig. 3.6) we report an example of a Betti-0 curve associated to the corresponding barcode. The curve shows the topological changes in the brain (merging of connected components) with the increment of the filtration value. The clusters in the brain are shown in different colours.

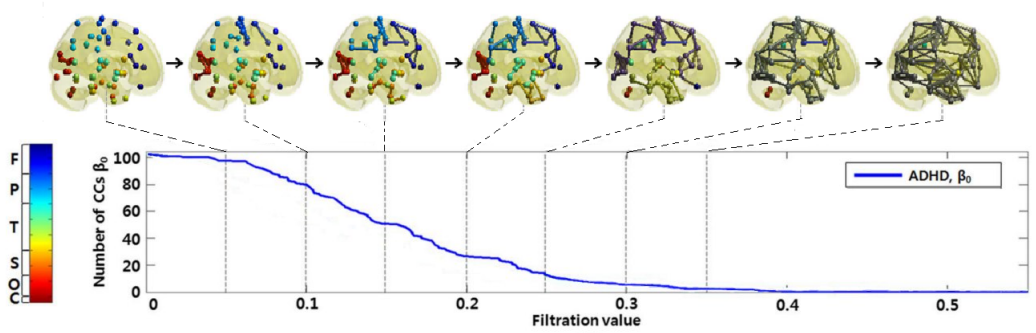


Figure 3.6. Changes in the brain network with the increment of the filtration value. Figure adapted from [2].

3.3 Distance between networks

In general, the network comparison is achieved by looking at the differences between the graph-theoretic measures such as high degree, betweenness centrality, and small worldness. Here, we propose the Gromov-Hausdorff distance to compare two dendrograms and investigate how similar two networks are.

3.3.1 Gromov-Hausdorff distance (GH)

Since each network has is own metric we must find a definition of distance that can be used to measure the distance between different metric spaces. The GH distance can be defined as follows:

$$d_{GH}(X,Y) = \frac{1}{2} \max_{x_i \in X, y_j \in Y} |d_X(x_i, x_j) - d_Y(y_i, y_j)|, \tag{3.4}$$

where X and Y are the two metric spaces and correspond to the nodes set X and Y (the two networks that we are comparing). $d_X(x_i, x_j)$ is the distance between node x_i and x_j in network X defined using the new definition of distance explained above (Eq. 3.3). Given the two single linkage matrices d_X and d_Y (obtained from the connectivity matrices c_X and c_Y with the explained algorithm 3.2), we compare each value of one matrix with the correspondent value of the other matrix, and we store all the comparisons. Then we take the maximum of these values. The so computed value is the distance GH between the two matrices. The GH distance finds where the difference is maximized between the two networks. In Section 4.1 we will apply these definitions to compute GH in simulated data.

3.3.2 Gromov-Wasserstein distance (GW)

In this section, we will study the Gromov-Wasserstein (GW) distance. We first need to give some relevant definitions, and after that, we can move step by step toward the definition of GW distance. The definitions in this section are taken from [9] and [10]. We start with the definition of a *metric measure space*.

Definition 11 ***A metric measure space*** *(or mm-space) is a triple* (X, d_X, μ_X) *where* (X, d_X) *is a metric space and* μ_X *is a Borel probability measure on* X*.*

In the finite case, μ_X reduces to a collection of non-negative weights, one for each point $x \in X$, such that the sum of all weights equals 1. The interpretation is that $\mu_X(x)$ measures the "importance" of x: points with zero weight should not matter, points with lower values of the weight should be less prominent than points with larger values of weight [9].

Definition 12 (Measure coupling) *Given two metric spaces* (X, d_X, μ_X) *and* (Y, d_Y, μ_Y) *one says that a measure* μ *on the product space* $X \times Y$ *is a* ***coupling*** *of* μ_X *and* μ_Y *if and only if*

$$\mu(A \times Y) = \mu_X(A) \quad \text{and} \quad \mu(X \times B) = \mu_Y(B)$$

for all measurable sets $A \subset X, B \subset Y$*. Denote by* $\mathcal{M}(\mu_X, \mu_Y)$ *the set of all couplings of* μ_X *and* μ_Y ([10] p. 450).

$\mu_X(x)$ is the way of weighting x. Notice that $\mu_X(X) = 1$ and $\mu_Y(X) = 1$ because X and Y represent the whole space.

Now we can define the Gromov-Wasserstein distance. We want to construct a new, tentative notion of distance between metric spaces. The idea is to use Eq. 3.4 as the starting point because the goal for the new distance is to directly compare the metrics of X and Y (in a meaningful way).

For metric spaces (X, d_X) and (Y, d_Y) let

$$\Gamma_{X,Y} : X \times Y \bigtimes X \times Y \longrightarrow \mathbb{R}^+$$

be given by

$$\Gamma_{X,Y}(x, y, x', y') := |d_X(x, x') - d_Y(y, y')|.$$

For $p \in [1, \text{inf})$ and $\mu \in \mathcal{M}(\mu_X, \mu_Y)$ let

$$\mathbf{J}_p(\mu) := \frac{1}{2}\left(\int_{X\times Y}\left[\int_{X\times Y}(\Gamma_{X,Y}(x, y, x', y'))^p \mu(dx \times dy)\right]\mu(dx' \times dy')\right)^{1/p}. \quad (3.5)$$

Each little portion of space $dx \times dy$ can have a different weight. $\mu(dx \times dy)$ is the weight of the little portion of space $(dx \times dy)$ we are considering during the computation of the integral. In practice, the integral is done on little portions of space $(dx \times dy)$ weighted applying μ to those spaces until all the spaces are covered, and the integral is completed.

We consider all the possibilities of assigning weight to the different portions of space, and for each combination, we compute an integral. Then the Gromov-Wasserstein distance is defined as the minumun of all these integrals:

Definition 13 (Gromov-Wasserstein) *For $1 \leq p \leq \infty$ we define the distance $\mathcal{D}_p$ between two mm-spaces X and Y by*

$$\mathcal{D}_p := \inf_{\mu\in\mathcal{M}(\mu_X,\mu_Y)} \boldsymbol{J}_p(\mu).$$

Let's try to explain the meaning of all these definitions intuitively and in relation to our brain matrices. We have two matrices X and Y corresponding to two different networks. In matrix X we have stored all the distances between $\boldsymbol{x}$= $\{x_1, x_2, \ldots, x_n\}$ and $\boldsymbol{x'}$= $\{x_1, x_2, \ldots, x_n\}$ whereas in Y all the distances between $\boldsymbol{y}$ and $\boldsymbol{y'}$.

$$\mathbf{X} = \begin{array}{c} \\ \mathrm{x}_1 \\ \mathrm{x}_2 \\ \vdots \\ \mathrm{x}_n \end{array}\begin{array}{c} \begin{array}{cccc} \mathrm{x}_1 & \mathrm{x}_2 & \ldots & \mathrm{x}_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \\ \mathrm{a}_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \end{array}, \quad \mathbf{Y} = \begin{array}{c} \\ \mathrm{y}_1 \\ \mathrm{y}_2 \\ \vdots \\ \mathrm{y}_n \end{array}\begin{array}{c} \begin{array}{cccc} \mathrm{y}_1 & \mathrm{y}_2 & \ldots & \mathrm{y}_n \end{array} \\ \begin{pmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ a_{21} & a_{22} & \ldots & b_{2n} \\ \vdots & & & \\ \mathrm{b}_{n1} & b_{n2} & \ldots & b_{nn} \end{pmatrix} \end{array}.$$

When we compute $\Gamma_{X,Y}$ in Eq. 3.5 we can compute the difference of any two points in the two matrices and not just the corresponding points, as it was the case of the

Gromov-Hausdorff distance previously defined (e.g. not just a_{12}-b_{12}, a_{21}-b_{21}). For instance, we can compare the value in the position a_{12} in the X matrix (distance between nodes x_1 and x_2) with the value b_{34} in the Y matrix (distance between nodes y_3 and y_4). In the integral is written that we have to multiply this difference by $\mu(dx \times dy)$. Let's try to understand what does it mean. In the case of spaces we defined above, this corresponded to multiply the difference by the weight we assigned to that particular portion of space $\mu(dx \times dy)$. Here we don't have 2D spaces: we are considering matrices, so in the integral we need to multiply by $\mu(x \times y) = \mu(a_{12} \times b_{34})$ and this value can be found in the matrix $\mathcal{M}(\mu_X, \mu_Y)$ (Definition 12). What is this matrix? To build it, we do the following: we assign values of μ to all the combinations of $\boldsymbol{x}$ and $\boldsymbol{y}$. We, therefore, generate a matrix with dimensions $len(X) \times len(Y)$ which contains all the values we assigned to μ (chosen arbitrarily). Remember that if we sum all the values in this matrix, we obtain 1 since the sum of all the weights of a given space is exactly 1. This matrix define entirely $\mathcal{M}(\mu_X, \mu_Y)$.
We can build different matrices of this type (assigning new values to μ) and each time compute the integral in Eq. 3.5. Then the GW will be the minimum of all these $J_p(\mu)$ integrals (Definition 13). In Section 4.4 we will apply these definitions to compute GW in simulated data.

Chapter 4

Simulated data: application of the studied tools

In this chapter, we will describe how we generated different types of simulated networks and how we created their connectivity matrices C_X. Then, using the Python algorithm described in Section 3.2, we will obtain the D_X matrices from the generated C_X matrices. Finally, we will show the results of the application of the Gromov-Hausdorff distance (Section 4.1) on the generated data. To test the validity of this metric, we will analyze the effects of the change of some parameters. We will change the variance of the Gaussian distribution used to distribute the nodes (Section 4.2), and the number of nodes in the network (Section 4.3). In the last part (Section 4.4), we will apply Gromov-Wasserstein distance to data.

4.1 Generate C_X, compute D_X and GH

To test the validity of the GH distance, we generated simulated data. In this section, we will explain how we implemented this procedure. The Python code used to generate the data is given in Appendix A.1.1 and A.1.2.

Our goal is to build a network with 100 nodes and obtain its connectivity matrix C_X. First, we want to distribute the 100 data points in a square of edge length L. For each point, we generate an x and y position. The points have been distributed in 10 different ways; the different probability maps are denoted with $C_{i=1}, C_{i=2}, \ldots, C_{i=10}$. In C_1 all the 100 points are distributed around the centre of the square using a Gaussian distribution with mean the centre of the square. The other ways are to distribute the 100 points around 4 different centers (C_2 distribution), or 9 (C_3), 16 (C_4), 25 (C_5),... 100 (C_{10}) centers as shown in Fig. 4.1. The variance of the Gaussian distributions is defined by $\sigma^2 = var * L/(i+1)$ with var=0.1[1]. In Appendix A.1.1 the code to generate these distributions is given.

[1]The variance of course depends on the type of probability map we are using, the Gaussian centered around the centers has a different value of variance depending on how many centers we have in the square. The formula is $\sigma^2 = var \cdot L/(i+1)$, where i is the number of the distribution and L the length of the edge of the square (see line 32 of A.1.1). In the following text we will refer to the variance σ^2 only by saying the value of var we are using, "var=0.1" (or "var=0.2").

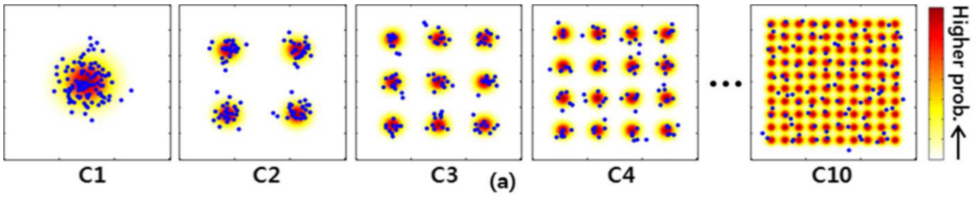


Figure 4.1. Synthetic data with 10 different types of probability maps. 100 points in each square. In C_1, all the points are distributed around the center. In C_2 they are distributed in 4 groups of 25 points etc. [2].

Each point in the square is considered as a node; we can therefore build the equivalent of a connectivity matrix using in our simulation the Euclidean distance between the points. The distance between points represents the strength of connections (correlation) between brain regions. We computed the distance between each pair of points in the square, and we saved all the distances in a C_X matrix. Since we have 100 points (representing the nodes in this simulated network) the C_X matrix is a symmetric 100x100 matrix. Each row/column corresponds to a distinct node; position (i, j) contains the value of the distance between the i-th and j-th nodes.

We have 10 different types of matrices corresponding to the different ways of distributing the points. The C_X matrices differ a lot depending on the corresponding probability map they were built from. The obtained matrices clearly show how the points were distributed in the square: in fact, in the C_X matrix correspondent to C_1 all the points are more or less at the same distances, and the values are all similar.[2] In C_2 we have 4 different groups, and so the matrix will have 4 groups of 25 points where the distance between any two points within one group is smaller than the distance of any two points belonging to different groups.
We can visualize the C_X matrix with a colour representation, using the visual representation of the Python library of *seaborn* (Fig. 4.2). The result shows the number of centres around which the points were distributed. Darker colour means a smaller distance. For example, in C_2 we can distinguish 4 darker squares, in C_3 we can distinguish 9 darker squares, in C_4 16, etc. corresponding to the points which are part of the same group. In Fig. 4.2 we show an example of the first three of the C_X matrices obtained from the first three distributions (C_1, C_2, C_3).

[2]For notation simplicity we will call C_p the matrix obtained from the C_p distribution. It will be clear from the context if we are referring to the distribution of points C_p or to the correspondent matrix C_p.

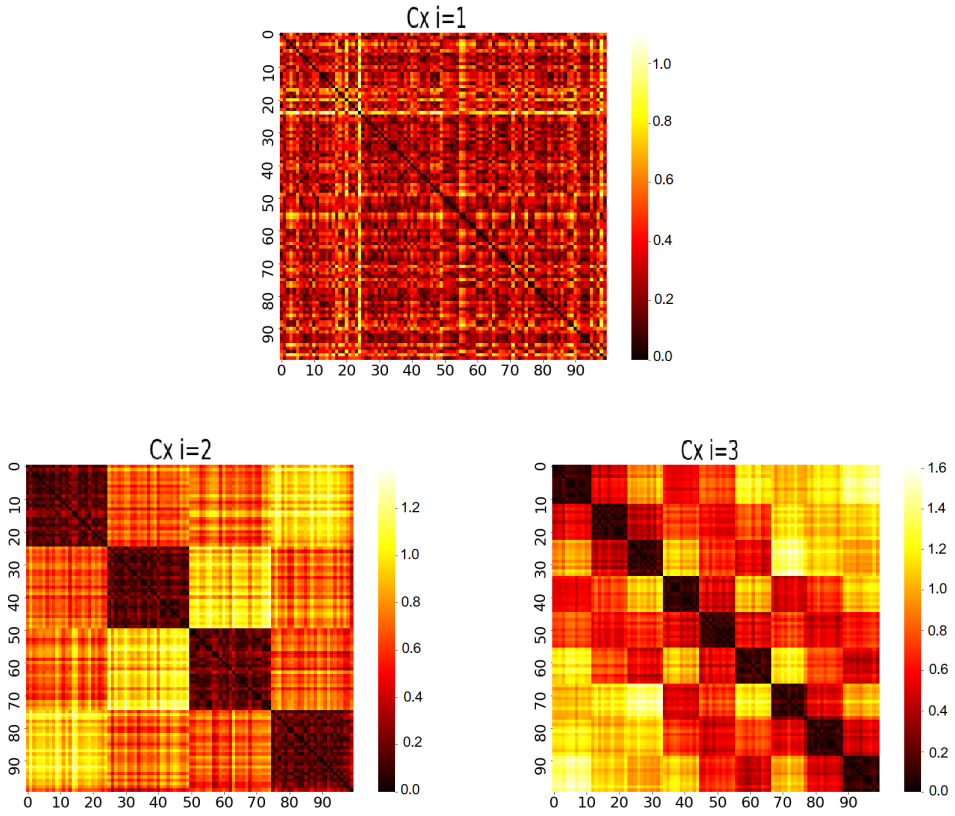


Figure 4.2. C_X matrices 100x100 obtained respectively from the three distributions of the 100 points around 1 center (C_1 distribution), 4 centers (C_2) and around 9 centers (C_3).

From the C_X matrices we can build the new D_X matrices. Each C_X will have a corresponding D_X matrix. The transformation of C_X to D_X matrices is done through the Python algorithm where we used the new definition of distance based on the cluster distances as explained above in Section 3.2.
In the D_X matrices, we can clearly distinguish all the clusters of points, as shown in Fig. 4.3.

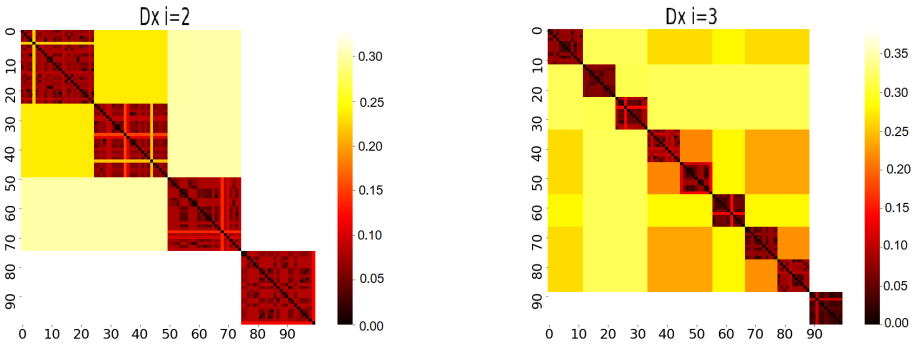


Figure 4.3. D_X matrices obtained through the Python algorithm from C_X matrices. We show the matrices obtained from C_2 and C_3.

To better visualize the merging of the nodes with the increment of the filtration value, we can build the *dendrogram*, explained in Section 3.1. In the dendrogram, we give the information of which nodes are merging at which filtration value. We observe the merging from 100 connected components (100 distinct points at the beginning) to a single giant component (where all the nodes are merged together). In Fig. 4.4 the dendrogram corresponding to C_2 is shown, as an example.

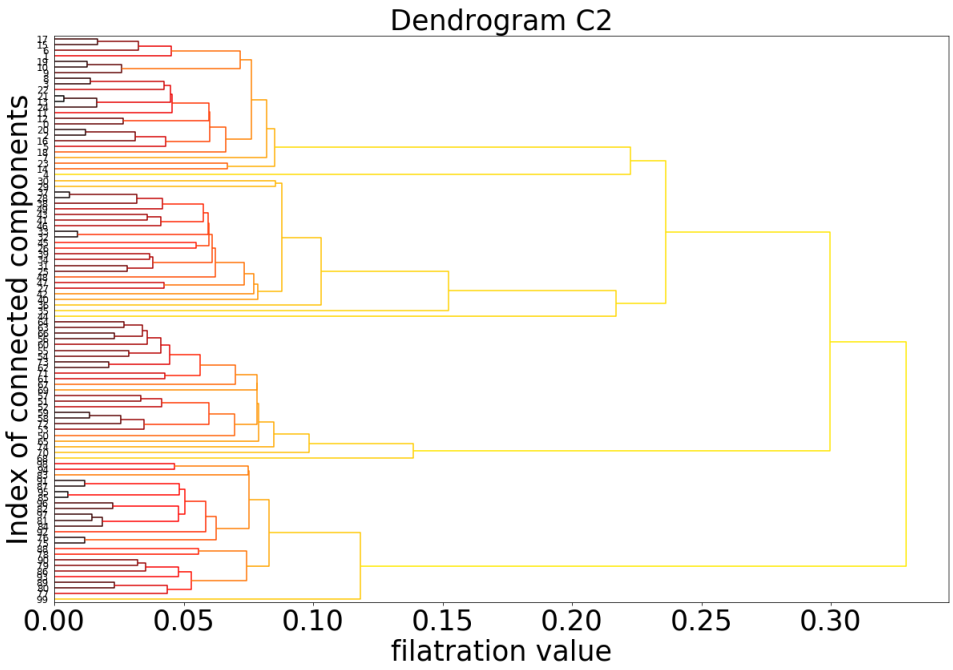


Figure 4.4. Dendrogram, visualisation of which nodes are connecting with the increment of the filtration value. First, the nodes in the same group are connecting (red and dark red lines). Then, when the filtration value has increased, the four groups are merged together (yellow lines).

Observing the figure, we note that the first nodes which connect are the ones belonging to the same group. The distances between the nodes in the same group are smaller compared to the distances from the points in the other groups, so they cluster together with a small filtration value (red and dark red lines). Then the four groups are merged together, and a final big component containing all the nodes is created. We can see that the two groups in the upper part of the figure are the first to merge (yellow line). They merge around a filtration value of 0.24; then the newly formed cluster merges with the third group around a filtration value of 0.3 and then finally the resulting cluster merges with the fourth group at a value of 0.35 forming one big connected component.

We can visualize the merging of the connected components looking at the barcode where is ignored the information of which nodes (their index number) are connecting, and we just count the number of connected components vs the filtration

value (as explained in Section 3.1). In Fig. 4.5 all the 10 types of D_X matrices are visualized using the Betti-0 curves (the curves associated to the barcodes). Each colour represents a matrix of a different type.

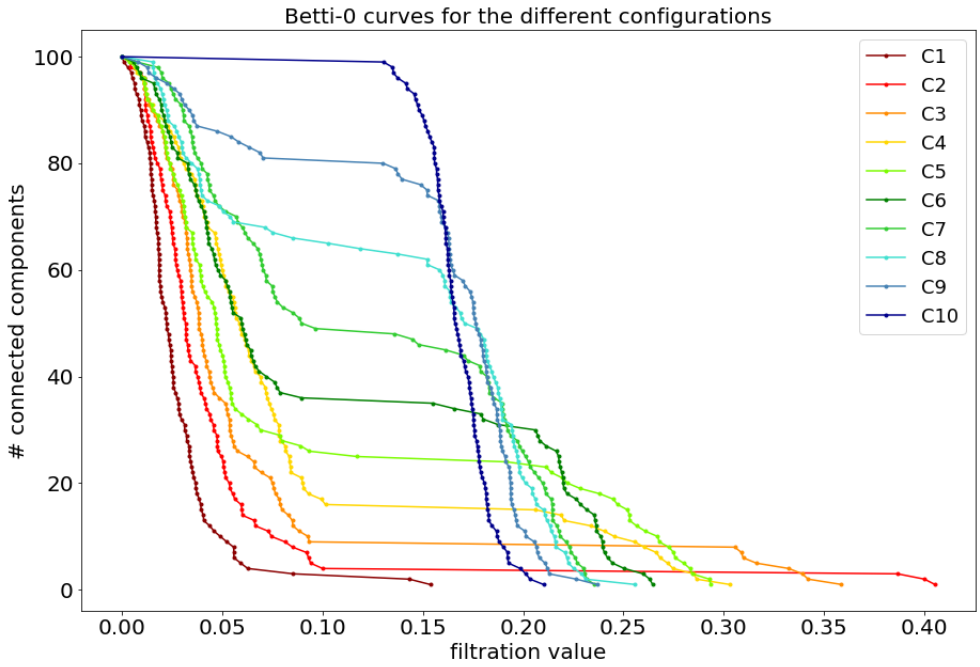


Figure 4.5. Betti-0 curves representing the number of connected components vs filtration value. Each colour represents a matrix obtained from one of the 10 different types of configuration of the nodes $C_1, C_2, \ldots, C_{10}$.

We note that in the case where all the points were distributed around the centre C_1 (dark red) the decreasing of the number of connected components β_0 proceeds with steps of the same size, the decrement is sharp because all the points were near each other. So within a small range of the filtration value, all the nodes are connected (filtration values in range [0, 0.15]). In the Betti-0 curve of C_2 (red), we can clearly see that at the beginning there is a small range of filtration value where the majority of points connect (from 0 to 0.1), and this corresponds to the connections within the 4 groups of 25 points. Then we have to connect the 4 clusters, and their distance is, of course, greater compared to the distances within the same group so, in the curve, we see a filtration value that is bigger than the previous one (≈ 0.38). We can exactly see the three last points in the red curve which correspond to the connection of the 4 groups ($\approx 0.38, 0.40, 0.41$). The same is true for the other curves: first, the connections within the group occur and then the ones between groups. In C_3 we can also clearly see when the connections within the groups are finished, and so we have to start connecting the groups (we go suddenly from a filtration value of 0.10 to 0.31). When we move to C_8, C_9 this effect tends to be less evident. In C_{10} this effect has totally disappeared because all the points are distributed uniformly in the

square, and so the distances between one point and its neighbours are more or less the same all over the points, there are no groups which first connect. *"When the dataset is changed from C_1 to C_{10}, the barcode has a flatter peak, lighter tail, and steeper slope."* [2].

For each type of distribution of points (probability maps $\{C_1, \dots, C_{10}\}$) we run the program 20 times so we generated 20 different experiments: 20 squares, each containing the positions of 100 points. So for each type of distribution of points, we create 20 C_X matrices and then 20 D_X matrices. Since we have 10 different types of configurations, we end up with 10x20=200 matrices. We now want to use the GH distance to compare them. What we expect is that the matrices from the same type (the 20 C_1 matrices, 20 C_2 matrices... 20 C_{10} matrices) will be more similar between them than with the other types. To perform the comparisons we use $d_{GH}(X,Y) = \frac{1}{2}\max_{x_i \in X, y_j \in Y} |d_X(x_i, x_j) - d_Y(y_i, y_j)|$ (Eq. 3.4). We consider two matrices at a time, and we compute the difference between one value of one matrix and the corresponding value of the other matrix. We repeat this for all the values, we store the obtained distances in a list, and we then take the maximum value of the list. Now we have a value of the distance between the two considered matrices, and we store it in the GH matrix. We repeat this procedure for all the possible couples of matrices. So in the GH matrix, there will be stored all the distances between any two matrices. The code for the GH can be found in Appendix A.3. We obtain a 200x200 dimensional distance matrix GH, that can be visualized using the *seaborn* library in Python. The result is shown in Fig. 4.6.

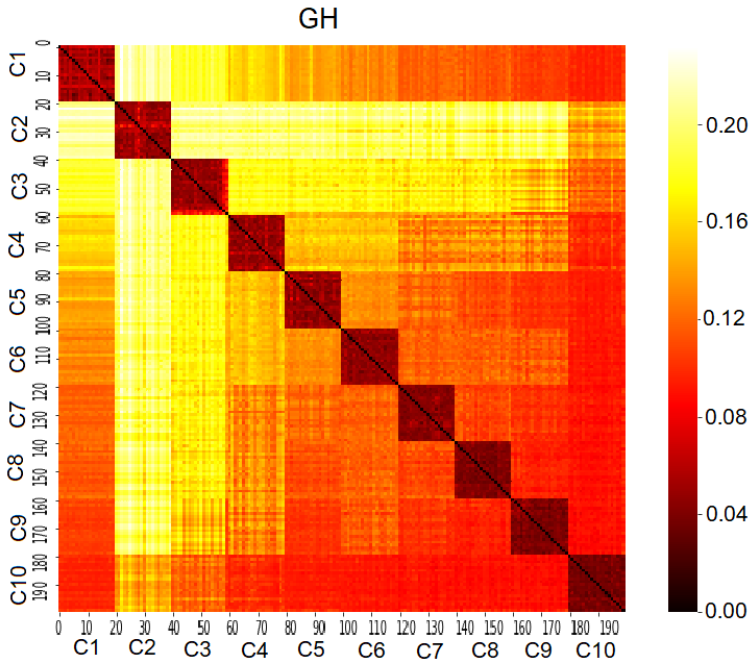


Figure 4.6. GH matrix 200x200. We can distinguish 10 squares corresponding to similar values, the matrices obtained from the same group are, in fact, similar.

In Fig. 4.6, we note that, as expected, the 200 D_X are clustered in 10 groups corresponding to the 10 types of probability maps used to generate the sample points. The 20 matrices driven from the same distribution, for each of the 10 groups, are very similar within the same group. In contrast, if we compare two matrices obtained from different distributions, for instance, a matrix from C_1 distribution to the one from C_2 the resulting value is high (≈ 0.2), meaning that the matrices are different. The GH is a powerful method to compare different matrices and to understand which matrices were driven from the same distribution. So we can use GH to understand if two matrices are similar or not.

In Fig. 4.7 we summarize the procedure explained so far.

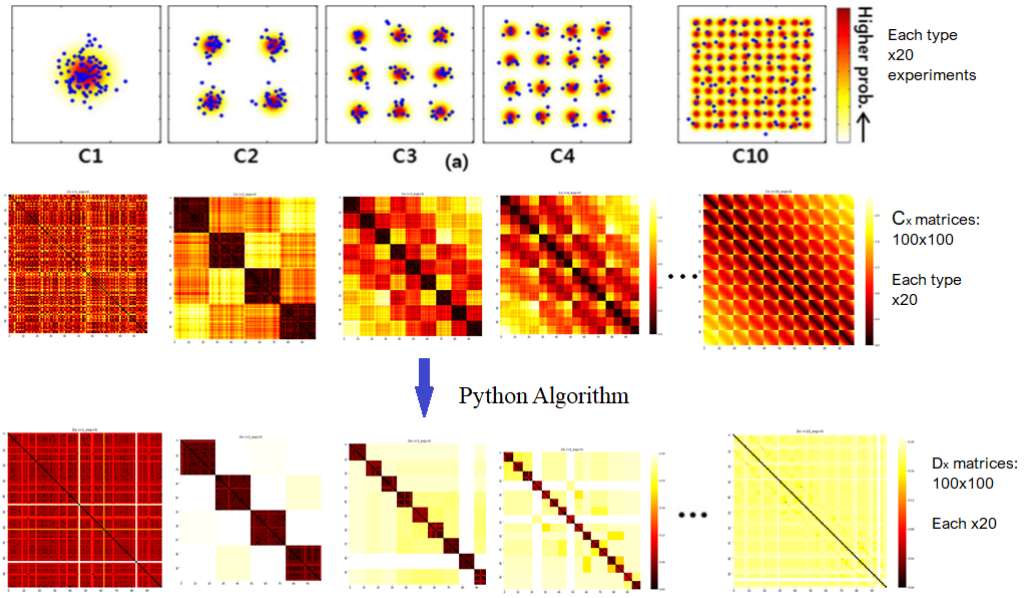


Figure 4.7. From the different type of distributions $\{C_1, C_2, \ldots, C_{10}\}$ we compute the correspondent C_X matrix where the distances between all the points are stored. The procedure is repeated 20 times for each type of distribution, so we have 10x20=200 C_X matrices. Then with the Python algorithm, we obtain the D_X matrices, and we can then compare them using the Gromov-Hausdorff distance to obtain the GH matrix.

4.2 Increasing the value of the variance of the distribution of points

In this section, we want to understand what happens to the GH if the matrices of the 10 different types are less different than in the just described case. We achieve this by increasing the variance of the Gaussian distributions when generating the 100 points around the centres.

We repeat the same procedure of generating 20 matrices for each of the 10 types but, this time, increasing the variance of the distribution of points. The Gaussians are again centred where we want to distribute the points, but there is more variability

compared to before: the points are more spread around the centres compared to the previous case. The C_X matrices still show the different groups of points, but the division between the groups is less defined compared to the previous case. For example, in the C_2 distribution, the four groups are nearer compared to the previous case because the points of each group are more spread around the four centres.

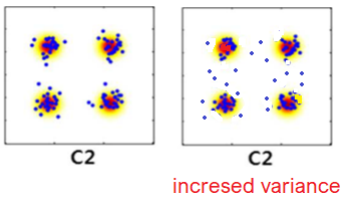


Figure 4.8. Visualisation of what happens to the C_2 distribution when we increase the value of the variance of the Gaussian distributions.

The matrix correspondent to the C_2 and the D_2 are shown in Fig. 4.9. As we can see, the four groups of points are less distinguishable compared to Fig. 4.2.

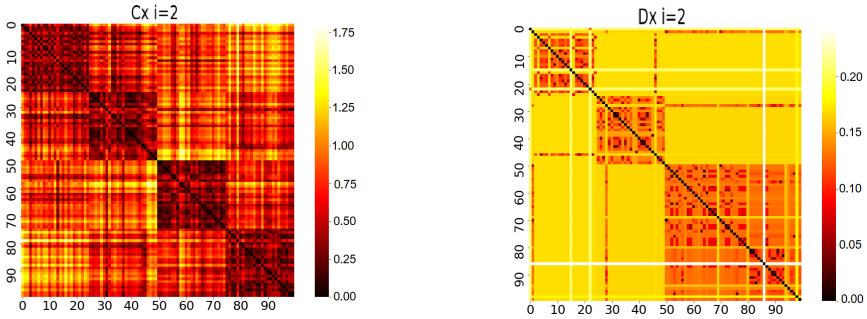


Figure 4.9. On the left: C_X matrix obtained from 100 points distributed around 4 centers (C_2 distribution). The variance of the Gaussian distribution of points around each center has been increased compared to the previous case (here var=0.2). On the right: D_X matrix obtained with the Python algorithm correspondent to that C_X matrix.

In Fig. 4.10 (see above) we show, using a dendrogram, the merging of the connected components in the case of the network obtained from C_2 with the 0.2 increased variance.
The distinction between the four groups is less evident compared to the case where the variance of distribution of point was smaller (Fig. 4.4 Dendrogram of C_2 with smaller variance). We notice that for any of the four groups the connections between the points within the same group (red and dark red lines) occur at greater filtration values compared to the connections of Fig. 4.4. This happens because the points are more spread than the previous configuration, the distance between them is bigger and so to connect them the filtration value must be greater. With the increment of the filtration value, the four groups merge together (yellow lines) as in the previous case. We note that the giant component occur at a smaller filtration value (0.25 compared to 0.35 of Fig. 4.4).

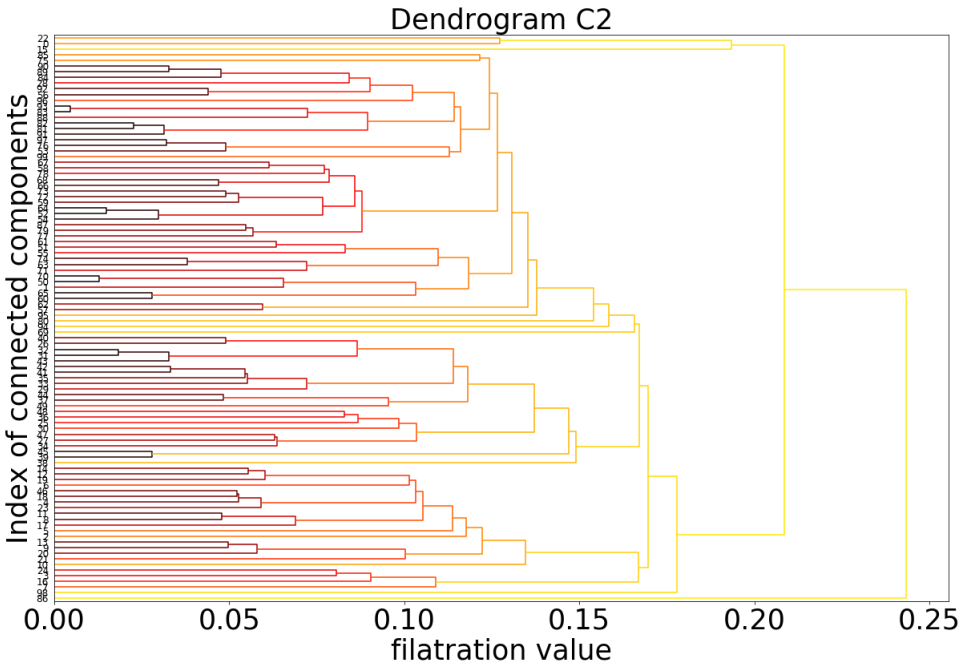


Figure 4.10. Dendrogram C_2: merging of the nodes with the increment of the filtration value. The 100 points were distributed around the four centers with a Gaussian distribution with var= 0.2.

In the following figure Fig. 4.11, we show the comparison between the Betti-0 curve of two C_2 distributions.

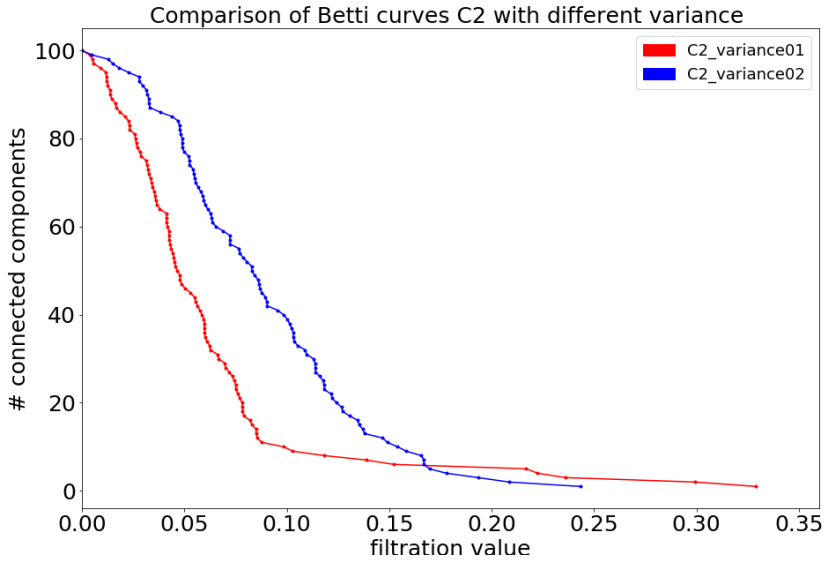


Figure 4.11. Comparison of two Betti-0 curves obtained from the same type of probability map C_2 but with different values of the variance of the Gaussian distribution.

In the first C_2 distribution, the Gaussian of the distribution of points (each Gaussian is centred around one of the four centres) had var=0.1, and in the second it had var=0.2. In the second case (blue curve) the points in the four groups cluster at greater filtration value but the giant component occurs before. This corresponds to the comparison between the two dendrograms that we have just explained.

Using the same procedure of the previous case (GH comparison Fig. 4.6), we compare the 200 Dx matrices (10 types of configuration var=0.2 x 20 experiments each); the result of this new GH is shown in the first image of Fig. 4.12.

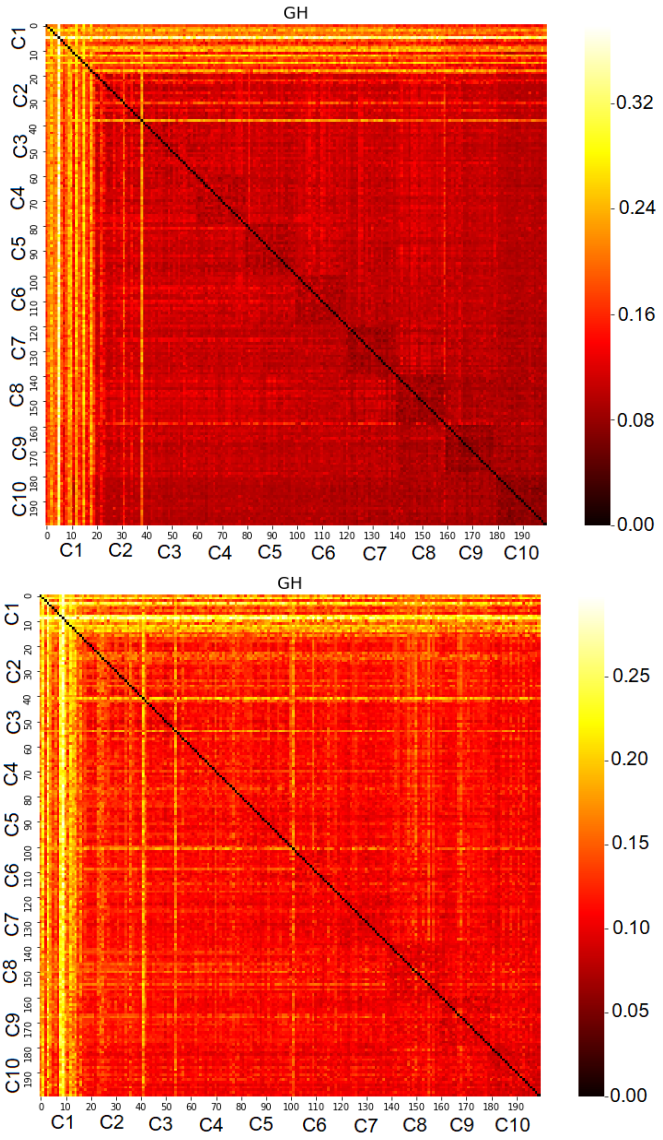


Figure 4.12. Two GH matrices 200x200. When we use var=0.2, we can still note the 10 squares corresponding to similar values. However, the distinction is less clear than the previous case. The matrices obtained from the same group are similar, but the increased variance introduces noise in the data. With var=0.3 (second figure), we can no longer distinguish the 10 squares corresponding to matrices of the same group.

In the first matrix of Fig. 4.12 we can still distinguish the 10 groups of matrices, but the distinction is worse than before (Fig. 4.6). Since the points are more spread around the centres, the difference between the different types of distributions is less evident. Increasing the variance corresponds to an introduction of noise in the data.

When we repeat the same procedure increasing further the variance (var=0.3) of the Gaussian distribution of points around the centres, we note that in the C_X matrix the clusters are less clear than before. The points are more spread and the net distinction between the groups is more difficult. So, the matrices of the different groups of distributions ($C_1, C_2 \dots C_{10}$) are now more similar among them.

Therefore, in the obtained GH matrix (second image of Fig. 4.12), we can not distinguish the 10 different types of matrices (based on the different distributions).

4.3 Increasing the number of points (density of the network)

In this section, we analyze the effect on the Betti curves when we increase the number of points in each square. The dimension of the C_X matrices is therefore increased. Increasing the number of nodes corresponds to create networks that are denser than before.

When we increase the number of points in the network, and then we compute the number of connected components vs the filtration value, we note that the tendency is different. In Fig. 4.13 the orange curves represent the Betti curves of

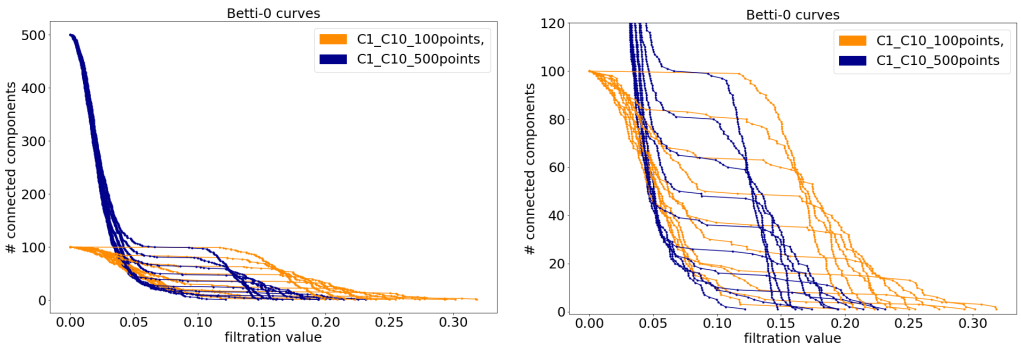


Figure 4.13. Comparison between Betti-0 curves with 100 points (orange) and 500 points (blue). The different curves of the same colour correspond to the 10 different types of initial distributions of nodes. In the left figure, a zoom in the y-range [0,120] is shown. The filtration values at which the giant components occur are smaller for the networks with 500 points.

the 10 D_X matrices obtained from the 10 configurations with 100 points in each configuration (matrix 100x100), whereas the blue curves are with 500 points (matrix 500x500). When, in our simulation, we increase the number of points that we put in the square we are increasing the density, and when we generate the C_X we are creating a bigger network where the nodes are nearer each other. When we have

more points, the filtration value where the giant component occurs is smaller. This happens because when we have more points, they are nearer to each other and with a small filtration value, many points have clustered together. In the right side of Fig. 4.13 we zoom on the Betti curves, the y-axis (number of connected components) is in the range [120,1] instead of [500,1]. We notice that the filtration values where the giant components occur are shifted to the left (smaller values) for the matrices 500x500 compared to the matrices 100x100. For example, for C_1 the giant component is shifted from ≈ 0.20 (orange line) to ≈ 0.12 (blue line).

4.4 Gromov-Wasserstein distance applied to simulated data

In this section, we will apply the Gromov-Wasserstein distance defined in Section 3.3.2 to our simulated networks and see if it can distinguish the matrices as the Gromov-Hausdorff distance. This method is useful when we want to compare networks of different sizes. In fact, if we have matrices of different size, the Gromov-Wasserstein distance is still able to compare them, whereas with the Gromov-Hausdorff it is not possible.

To compute the Wasserstein distance between networks, we used the *scipy.stats* library of Python where the function of GW is already implemented.
First, we tried to compute the Wasserstein distances on the 200 matrices of size 100x100 obtained in the way described above. The 10 types of matrices (20 of each type) can be distinguished with this method as is shown in Fig. 4.14.

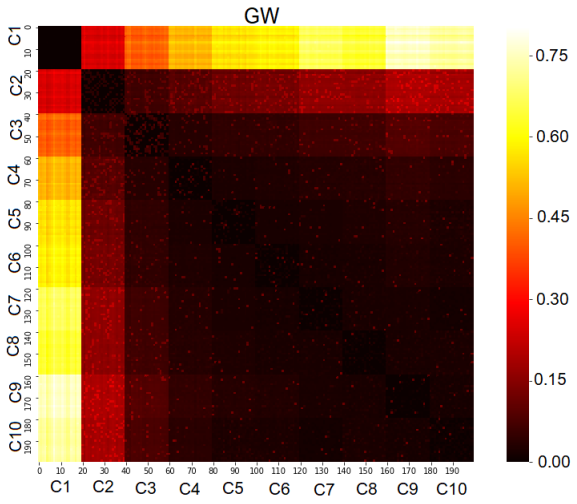


Figure 4.14. Matrix 200x200 obtained using the Gromov-Wasserstein distance. The 10 types of configurations with 20 matrices each are distinguishable. For the same type of distribution, the 20 generated matrices are similar, and so their distance is small, so a dark square for each of the ten distributions is distinguishable along the diagonal.

Then, we tried to compare networks of different sizes in the following way: for each of the 10 types of configurations, we generate 20 experiments. Each of them will have a different number of points, chosen with a Gaussian distribution with a variance of 20. So, the generated C_X matrices have different dimensions. Even if we changed the number of points, the different types of matrices are still distinguishable.

The Gromov-Wasserstein can be a good metric to compare networks that have different sizes.

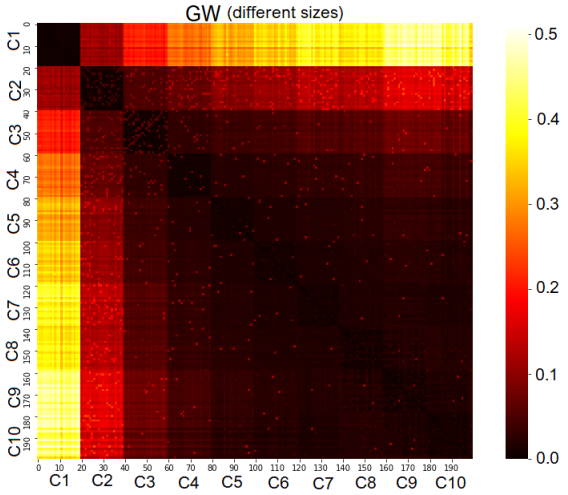


Figure 4.15. Matrix 200x200 using Gromov-Wasserstein distance. In each type of distribution, for each of the 20 experiments, we had a different number of generated points. Using the GW, we can still compare the 200 matrices even if they have different dimensions. In the figure, we notice that the 10 types of matrices are still distinguishable, even if they have a different number of points.

Chapter 5

Experimental data: application of the studied tools

In this chapter, we will apply the methods described so far, first to a database of experimental data available in the Human Connectome Project (Section 5.1) and then to a Database of the Amsterdam UMC (Section 5.2). We will analyze the GH matrices obtained for these two databases, we will introduce the Jackknife method (Section 5.1.2), and a detailed analysis of the Betti-curves will be developed in Section 5.2.2 and Section 5.2.3.

5.1 Human Connectome Database

The Human Connectome Database we used, is a database composed of a total of 66 connectivity matrices referred to 36 Autism Spectrum Disorder (ASD) children and 30 Typically Developed (TD) children. The connectivity matrices were obtained from fMRI measures. In the brain of the patients, 264 nodes were considered, and the correlation between each pair of nodes was recorded. So, each matrix had dimension 264x264. Before applying all the described methods, the matrices of the database needed to be rescaled. This way we can have connectivity matrices where the values are in the desired range, i.e. between 0 and 2, as is the case of the matrices in the paper [2] which we used as a starting point. The elements in the diagonal of the matrices in the database were set to "infinity"; this indicates that one node with itself has a high correlation. To manipulate the data, we needed finite values, so we set the diagonal of the connectivity matrix to 1. We then computed the maximum of each matrix and transformed the matrix such as each element is divided by that max and so we obtain a rescaled C_X matrix. Then we took 1-C_X so that we have a connectivity matrix where the diagonal is 0, meaning that the distance between one node and itself is 0. In the rescaled matrices, the greater values represent nodes that are less correlated (less connected).

Then the D_X matrices, the dendrograms, and Betti curves were created as described in the previous chapter. In the database, we have connectivity matrices of ASD patients and TD. What we expect is that, if we compare the matrices of the

two groups, the Gromov-Hausdorff distance (GH) will show that the matrices in the same group are more similar and so have a smaller distance compared to the ones in the other group.

5.1.1 GH analysis

When we compared the matrices of the two groups, using the GH method explained in Section 4.1, we obtained the result shown in Fig. 5.1.

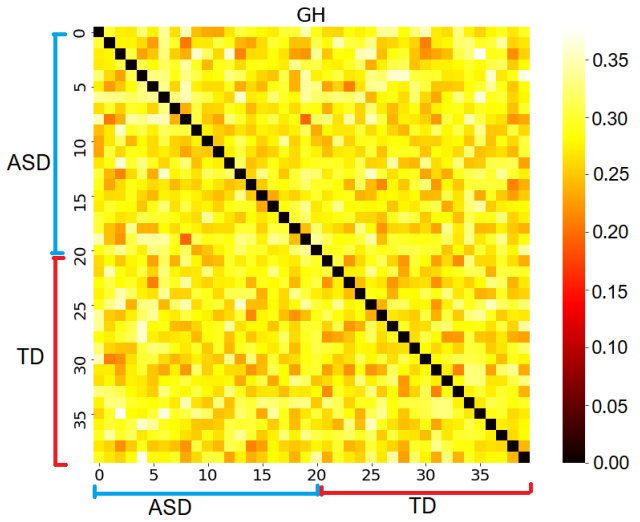


Figure 5.1. GH matrix 40x40. The comparisons were performed using the original matrices. We compared any two pairs of matrices in a group composed of 20 ASD matrices and 20 TD matrices using the GH distance.

Differently from what we expected, using the GH we can not distinguish the two groups of patients. The explanation is that all the matrices are too similar, and the GH method is not powerful enough to detect the differences between the two groups of patients. Moreover, looking at the analysis we did on the variance of generated data in Section 4.2, we can understand that if the matrices are too similar the GH can not distinguish them (Fig. 4.12). In the paper of Lee [2] to test the GH method on data obtained from experimental data collection, they used the Jackknife method.

5.1.2 Jackknife method

In order to create two groups of patients that have similar matrices within the same group and different matrices compared to the other group and obtain a group distinction using GH, we used the Jackknife method. In this section, we will explain this method that the paper of Lee et al. [2] uses, and we will obtain the same distinction between groups as they did. In Section 5.1.3 however, we will conclude that this method is more of a theoretical interest than of an experimental one.

To compute a new group of 20 similar matrices starting from 20 matrices of ASD patients, we proceed in the following way: we compute the mean of the starting matrices leaving the first matrix out. The matrix obtained will be part of the new group. Then we repeat the same procedure by taking out, this time, the second matrix and so on for 20 times (each time one different matrix is taken out).

$$Cx_{new}1 = \frac{Cx_2 + Cx_3 + ... + Cx_{20}}{19} \tag{5.1}$$

$$Cx_{new}2 = \frac{Cx_1 + Cx_3 + ... + Cx_{20}}{19} \tag{5.2}$$

$$Cx_{new}k = \frac{\sum_{i=1, i\neq k}^{20} Cx_i}{19} \tag{5.3}$$

Using this method, we have created matrices that are similar in a robust way. We do the same for the TD group. We now have two groups of 20 matrices each. If we then follow the usual procedure where we compute the D_X matrices, the barcodes, and then we compare the matrices using the GH metric, we obtain the result in Fig. 5.2.

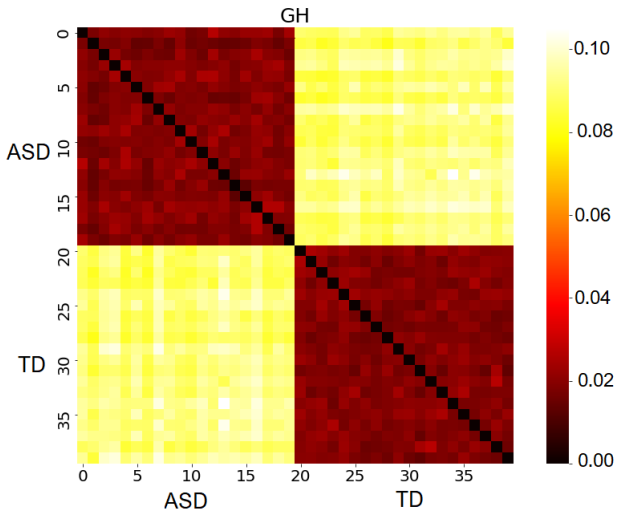


Figure 5.2. GH 40x40 matrix. Comparisons performed between 40 matrices. We have two different groups of matrices: 20 matrices were obtained with the Jackknife method from 20 ASD patients, and 20 matrices were obtained, using Jackknife, from 20 TD. The GH is able to distinguish the two groups. In this matrix 40x40, the darker squares are ASD-ASD/TD-TD the lighter are ASD-TD.

As we can notice in Fig. 5.2, the GH metric is perfectly able to distinguish the two groups. The matrices from the same group are, in fact, very similar between them (dark red) whereas the matrices from different groups are different (yellow). The GH matrix is a 40x40 matrix where all the pairs of two matrices are compared. On the x-axis, we have 20 ASD matrices (index 0-19) and then 20 TD matrices (index 20-39), on the y-axis the indexing is in the same order. The value in the matrix indicates how different two matrices are. To visualize the values, we used a colour map

(*seaborn* Python library) where darker colour means similar matrices and light means different. If for example, we compare one matrix from the ASD group (for instance index=10) to one from the TD group (for instance index=20) the two matrices will be very different, and so the corresponding value of distance will be big (light colour).

We also tried to compare the matrices using a graph measure: the betweenness centrality (Fig. 5.3). In Section 1.4 we have defined this quantity. We computed the betweenness centrality of all the nodes in a matrix and then computed the average in order to have one measure for each patient. We then compared the measures for all patients to see if there were differences between them.

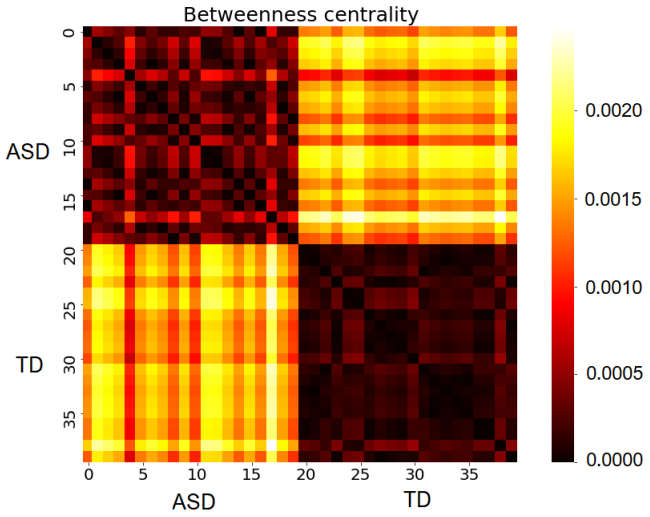


Figure 5.3. Matrix 40x40 were we store the comparisons of the values of betweenness centrality for all the pairs of matrices (20 ASD and 20 TD). The matrices in the same group of patients have similar values of betweenness centrality, so the correspondent values in the matrix will be small (dark colour). The two darker square are ASD-ASD, TD-TD whereas the lighter squares are TD-ASD and ASD-TD.

Comparing this result to the GH matrix (Fig. 5.2), we can see that GH distance differentiates better the two groups than betweenness centrality. In fact, if we take the lighter of the darker square and the darkest of the lighter square and we try to compare these two patients using GH we can distinguish them, whereas using betweenness centrality this is not possible.

To compare the matrices of the patients; we also used a different method. In the formula of GH (Eq. 3.4) instead of taking the maximum of all the differences between the two matrices, we can take the mean. Of course, in this way, the matrices will result to be more similar because when we compare them, we are not taking the maximum difference between the two but the average (Fig- 5.4). Suppose we use the original formula of GH (Eq. 3.4) and we compare two matrices which are identical except from one value. In that case, the GH will detect it and will say that the two matrices are different (because we are taking the maximum of all the difference between the two matrices) whereas, if we use the mean, the difference between the

two matrices will not appear.

The resulting matrix of this new way of comparison is the following:

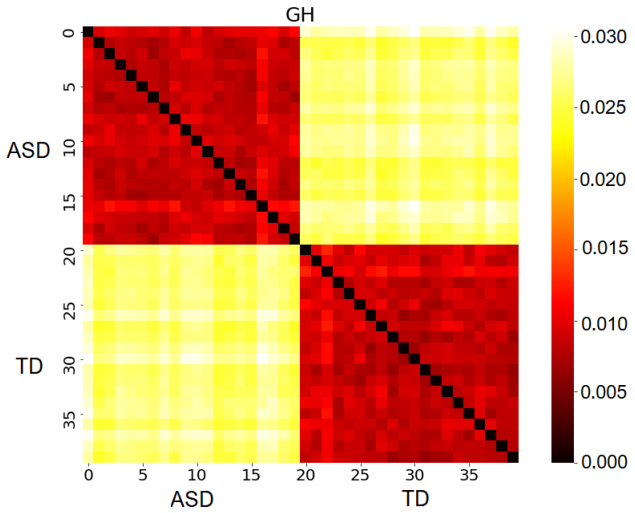


Figure 5.4. GH matrix 40x40 where we are comparing the matrices obtained using Jackknife method. Here, in the GH formula we are using "mean" instead of "max".

We notice that compared to Fig. 5.2 the values in the matrix are smaller. The use of "mean" in the GH formula cause the matrices to be similar than using "max".

5.1.3 Betti curves of ASD and TD

In this section, we will present the Betti-0 curves of the TD and ASD patients and analyze their difference.

We computed the Betti-0 curves of the networks with the algorithm described in Section 3.2, and we plot all the curves in the same figure. The results for the original data (not the groups obtained with the Jackknife method) for 20 ASD and 20 TD is shown in Fig. 5.5.

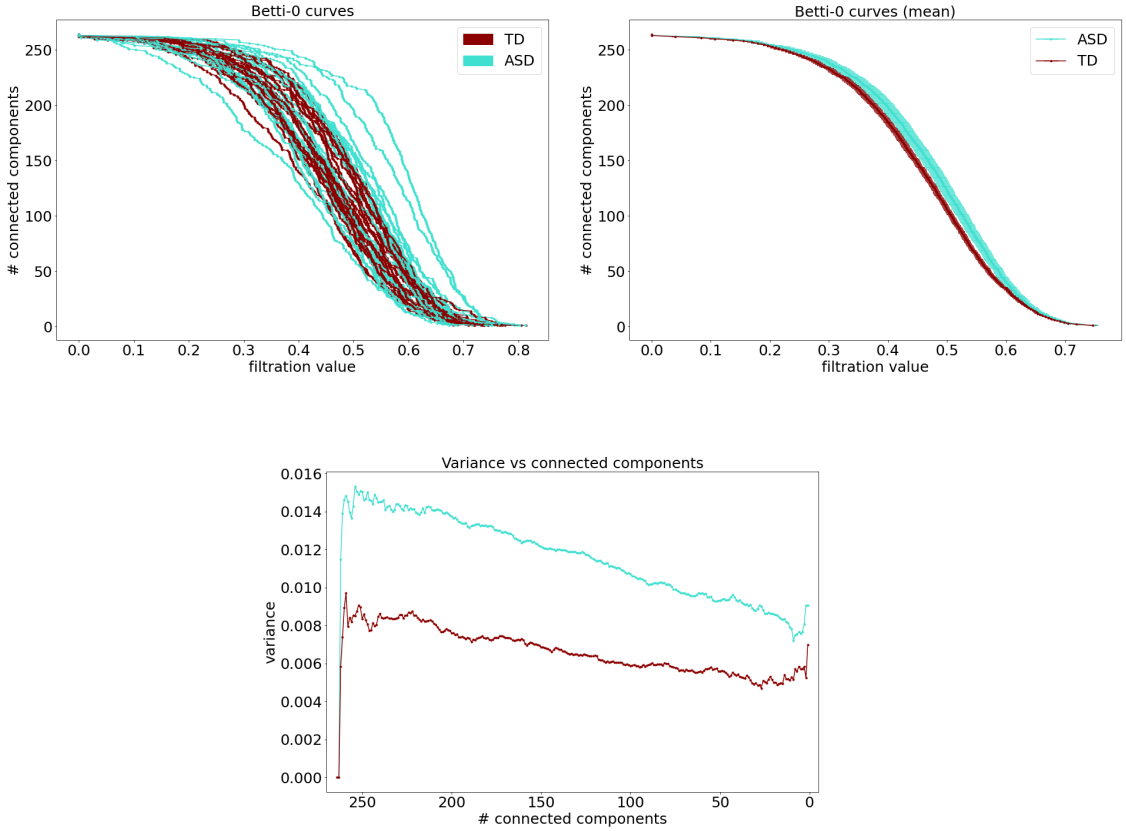


Figure 5.5. Left plot: Betti-curves of 20 TD and 20 ASD patients. The Betti-0 curve of the number of connected components vs the filtration value is shown for each patient. ASD patients show more significant variability. Right plot: mean curves for the two groups of patients. The mean curves indicate that at a group level, the nodes in brain networks of the ASD patients cluster slower compared to the TD. Bottom plot: Variance of the two groups, as we already noticed in the first plot the ASD patients are more variable. Colour legend: ASD-turquoise, TD-dark red.

We noted that patients with ASD disease were more spread compared to healthy patients. We confirmed that by looking at the variance of the two groups (Fig. 5.5 third picture). This result is in accordance with other studies: patients affected by a disease are, in general, a lot more variable compared to a group of healthy

individuals. Then we computed the mean curve (with error[1]) of all the curves of each group and we obtained the plot on the right side of Fig. 5.5. The red curve represents the TD group, whereas the light blue represents the ASD. The curves show that at a group level, the mean curve of ASD patients is above the mean curve of TD patients.

We can interpret this result thinking that, at a group level, the nodes in the brain of healthy individuals cluster faster than in ASD patients. At the given filtration value in fact, in the ASD patient curve, we have a bigger number of connected components compared to the TD group. The Betti-0 curves are a good method to differentiate patients from different groups.

If we analyze the Betti curves on the jackknife data we can fully understand what is happening when we are using this method (Fig. 5.6). In each group, all the matrices we obtained using this re-sampling technique, look very similar and differ from the matrices of the other group. This similarity/dissimilarity is evident when looking at its Betti curves. This net difference between groups is introduced by the method and does not reflect real data differences. With this method, we have created ad hoc two new datasets where of course everything works.

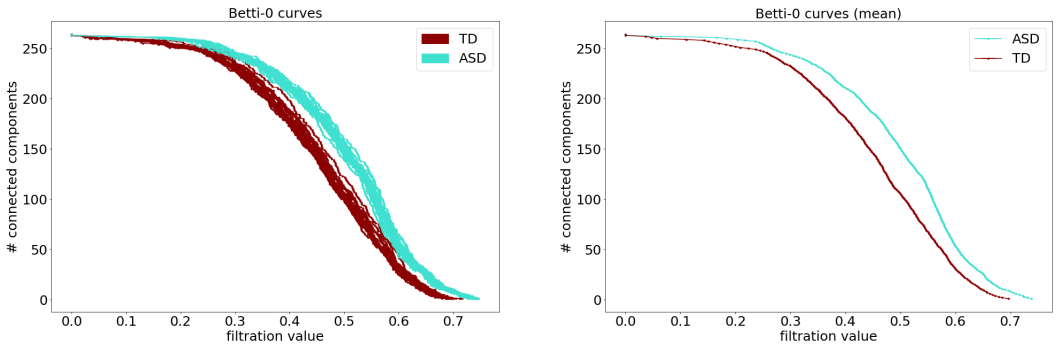


Figure 5.6. Betti-curves of 20 TD and ASD patients (obtained with the jackknife method) and correspondent mean curve (with errors).

To prove that this net difference is introduced by the Jackknife method and is not truly due to the shape of data, we also performed the following analysis. We created two groups of real matrices where each group contains 20 matrices: 10 ASD, and 10 TD (we have two mixed groups). We performed the Jackknife method (Eq. 5.1) on the two groups of 20 matrices separately, creating two groups of new matrices. The result of the GH analysis on these two groups and the Betti curves are precisely the same as the above-analyzed case (where the starting matrices of the two groups were 20 ASD and 20 TD, not mixed groups). So no matter the starting matrices we use in the group, the jackknife method gives a new group where all the matrices are

[1]The error associated with the mean value of the curves was calculated as $\frac{\sigma}{\sqrt{N}}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\overline{x})^2}{N-1}}$, N being the number of individuals in the group. For all the Betti mean curve appearing in the text the error is computed as explained and is shown in the plot with error bars.

similar and differ from another jackknife created group (Fig. 5.7).

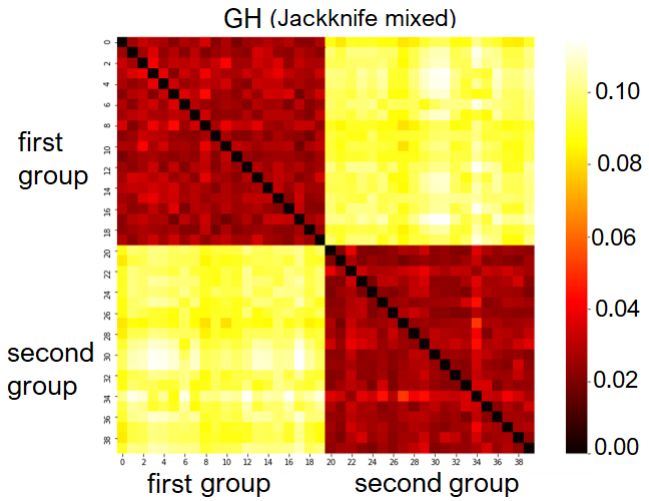


Figure 5.7. GH 40x40 matrix. Comparisons performed between 40 matrices. We have two different groups of matrices: 20 matrices were obtained with the Jackknife method from the first group, and 20 matrices were obtained, using Jackknife, from the second group. First group: 10 TD, 10 ASD resampled with the jacknife method obtaining 20 new matrices. Second group: 10 TD, 10 ASD resampled with the jacknife method obtaining 20 new matrices. The GH on the 40 matrices shows that the same group has similar matrices (but in that group there were ASD and TD mixed!).

We can conclude that the Jackknife method is not suitable if we want to understand the differences in real data. It can be a method of theoretical interest to create new matrices to run simulations on them (trying, for example, GH or other metrics) but doesn't have experimental or clinical interest.

5.2 Database of the Amsterdam UMC

In this section, we will apply the studied methods on a database of the Amsterdam UMC (University Medical Center) and present all the results. The comparison using GH methods is illustrated in Section 5.2.1, the Betti-0 curves are presented in Section 5.2.2, and a detailed analysis on the area under the Betti-0 curves is performed in Section 5.2.3.

We will now present the database of the Amsterdam UMC we used for this study. The database is composed of 71 glioma patients and 53 healthy control individuals (HC). For each brain, 78 nodes were considered, so the matrices had shape 78x78. For each Glioma patient, 60 consecutive epochs[2] of 4096 samples (3.27 seconds) were available for analyses, and for each HC, 52 consecutive epochs of identical length. So for each patient, we have 60 (or 52) different connectivity matrices [11].

The values in the matrices indicate how strongly two nodes in the brain are correlated (connected). The matrices are obtained based on MEG data. One significant

[2]time slices of data recording.

advantage of MEG data is that we can consider all the nodes, even the ones who are in a tumour brain region, whereas other data collection systems force to exclude these nodes (e.g. in fMRI data, we need to exclude these nodes because of the signal distortion caused by the tumor). To construct the matrices, the group who collected the data first built the times series obtained by recording magnetic fields produced by electrical currents occurring in the brain. Then to obtain the matrix, the idea is to consider the correlation in the time series: if two curves, correspondent to the electrical current of two nodes, are increasing (or decreasing) at the same time we say that those nodes are positively correlated. If one curve is increasing, and the other is decreasing at the same time, the value in the matrix will be negative; the two nodes are negatively correlated. (See also Section 3.1 and for more detail on this procedure see [11]). So greater value (in absolute value) correspond to more correlated nodes.

To start with the analysis of the data and carry on the method listed above we first rescaled the matrices in order to have all positive values. To do so, we took the absolute value of the matrix, we put 1 in the diagonal, and then we computed the 1-C_X matrix. So we end up with a matrix where smaller values indicate that the nodes are more correlated ("nearer" if we think of intuitive distance) and great values correspond to nodes that have small correlation ("far apart"). In the diagonal, we have 0 (one node with itself is strongly correlated). Just to give an example of how the data in the matrix are, we plot here a histogram of the values of one HC individual. We excluded the left part of the histogram where we had 78 0 values (the diagonal of the matrix) in order to have a more zoomed plot.

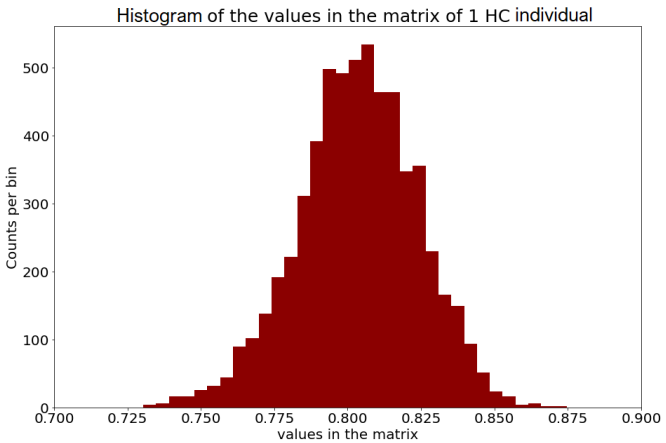


Figure 5.8. Histogram showing the distribution of values in the matrix of a healthy individual. We excluded the diagonal of the matrix, which contains 0 values.

5.2.1 GH analysis

In order to compare the networks of the patients, first we built one matrix for each patient by doing the mean of the 60 (or 52) connectivity matrices that we had for each patient. Then from the obtained matrices, we created the D_X matrices using the Python algorithm (Section 3.2), and then we compared them using the GH method.

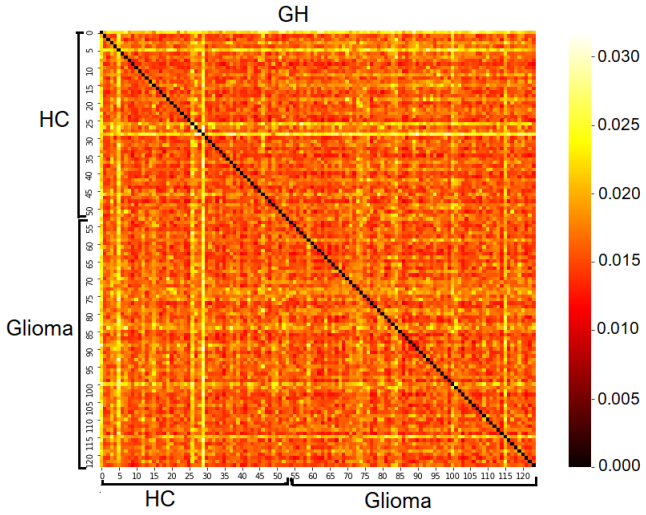


Figure 5.9. GH matrix of 53 HC and 71 Glioma patients. For each patient, we have 1 matrix, and we compare all the possible pairs of matrices using GH distance. This matrix is then 124x124. The diagonal corresponds to the comparisons of one matrix with itself; this is why the values in the diagonal are 0.

Unfortunately, as shown in Fig. 5.9 we aren't able to distinguish the two groups of patients, all the matrices are similar independently of the group they are part of. This is again because the matrices are too similar and the GH is not strong enough to distinguish them.

We also analyzed the Betti-0 curves: here, we can appreciate a difference at a group level. Again we first computed the curve for each patient and then computed the mean curve (with error) for each group. The results are shown in Fig. 5.13 and this result is explained in detail in Section 5.2.2.

Using the Jackknife method we can, of course, distinguish the two groups but basically as explained before (Section 5.1.3) we are creating two new datasets. So this doesn't tell us much about the real difference between the two original groups, and as said, this is not the right way if we want to investigate real differences among data.

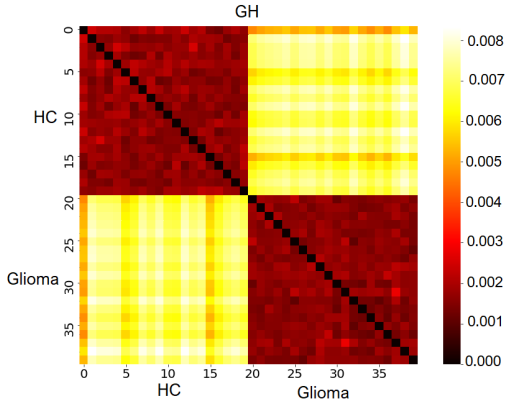


Figure 5.10. GH matrix 40x40. Comparison of each pair of matrices (20 HC and 20 Glioma patients). The matrices were obtained using Jackknife method.

Comparing two patients

In this section, we present the result of the comparison between all the matrices of two patients. In this case, we don't have just one matrix for each patient (as in the previous section): now we are taking into account for each patient all his matrices and not their mean (60 matrices for the Glioma patient and 53 for the HC).

The original hypothesis was that the matrices of the same individuals (brain representation taken at different times) should be more similar among them compared to the matrices of another patient.
In order to test this hypothesis, we consider two individuals: one Glioma patient and one healthy control. We generate all the 60 (or 52) D_X matrices for each patient, and then we compared them using the GH equation (first with max and then also with mean in Eq. 3.4). What we expected to see in the GH matrix was two distinct groups corresponding each to the 60 (or 53) matrices of each patient.

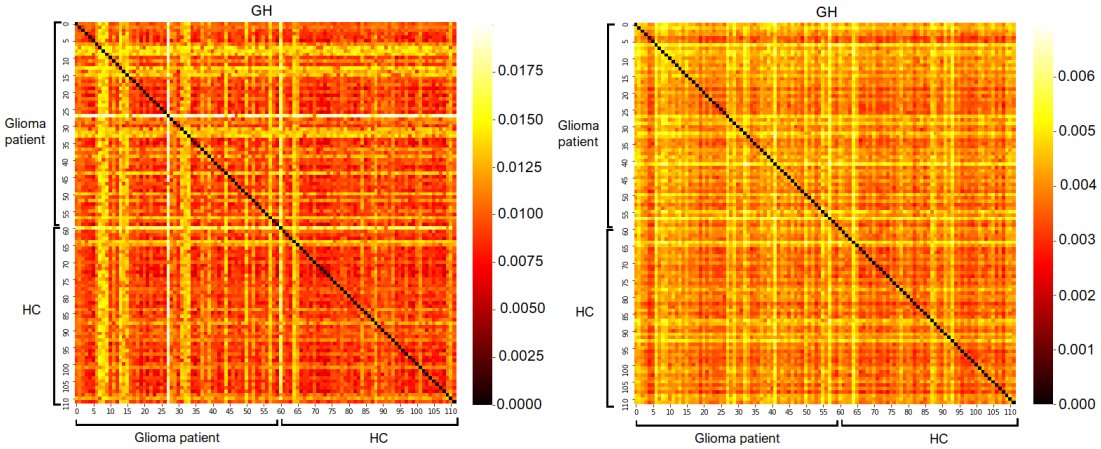


Figure 5.11. GH comparisons between all the matrices of two individuals. For the healthy individual, we have 52 matrices, for the Glioma patient we have 60 matrices so this GH matrix has dimension 112x112. The left matrix is obtained using "max" in the GH formula the right matrix is obtained using "mean".

In Fig. 5.11 we can't see what we expected, it seems that our hypothesis failed. The reason for that could be the fact that the matrices (the different epochs) are noisy, a little perturbation during the data collection, such as a blink of the eye of the patient can influence the matrix. So the matrices of the two patients are too noisy to be distinguished. The 60 matrices of one patient are not different enough from the 53 matrices of the HC to be distinguished using the GH method, which works only if the matrices show considerable differences in their values.

Comparing the matrices of the same individual

We also compared the matrices of the same individual (each matrix is a different epoch) to better analyze them and see how much noise there is in the data (Fig. 5.12).

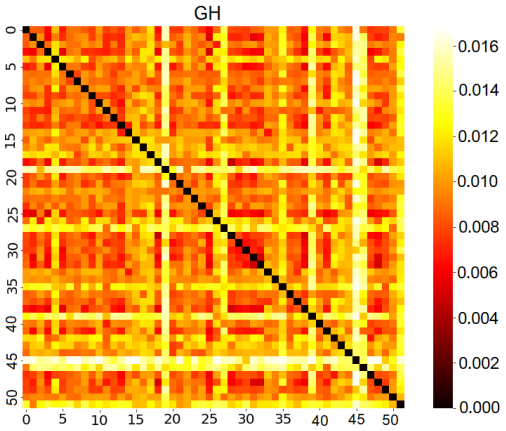


Figure 5.12. GH matrix 52x52. Comparisons of all the pairs of the 52 matrices belonging to the first HC.

We found that the data are noisy. As we saw in Section 4.2 if there is too much noise in the data, the GH is not a good method to classify the matrices.

5.2.2 Betti curves of Glioma and HC

In this section, we will analyze the Betti curves of each patient and reveal the difference between the Glioma patients and HC.

Using the Jackknife, as we did in the database of ASD from the Human connectome project, we can distinguish well the two groups. With the Jackknife, we are creating ad hoc the matrices to be similar among the same group so we will, of course, see a difference in the Betti-0 curve at a group level. We omit the figures because this analysis is not interesting to find real differences in the patients.

Using the real data and looking at the Betti curves we obtain:

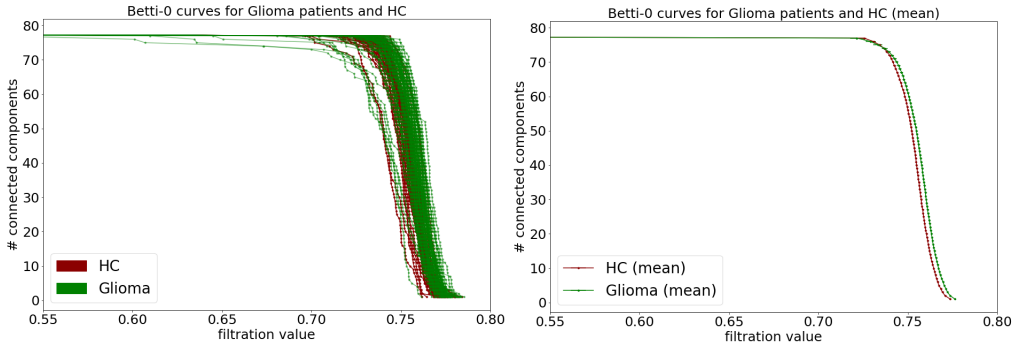


Figure 5.13. Betti-0 curves of 53 Hc and 71 Glioma patients and correspondent mean curves (with errors).

Also in this database, as it was in the ASD database from the Human Connectome Project, we can notice that the disease group, in this case, the Glioma, has a greater variability compared to the HC. However, using the Betti-0, we are able to distinguish the two groups. Overall we notice that the nodes in the brain of the Glioma patients are clustering slower than the HC, the mean curve of the Glioma is above the HC; the curve of the HC has a steeper slope.

We can interpret this result thinking that, at a group level, in the brain of Glioma patients, the clustering of the connected components with the increment of the filtration value is slower compared to HC. At the given filtration value, in fact, in the Glioma patient curve, we have a more significant number of connected components compared to the HC group.

This result tells us that the use of Betti-curves is a powerful method to compare the two groups of patients and to distinguish them and can give an insight into the disease. This method is better than the GH, in which if the matrices are too similar (as is the case of real data) we can not differentiate the patients. Therefore from now on, we will continue analyzing the Betti-0 curves and move to high order Betti numbers using TDA in the next chapter.

5.2.3 Area under the Betti curves

To carry a detailed analysis of the difference between the two groups, we can compute the area under the Betti-0 curve for each patient, so we have numerical values to perform a comparison. In this section, we will analyze the area under the curve for the two groups and use a statistical test to determine if the difference is significant.

In the left picture of Fig. 5.14 we plot for each HC and Glioma patient the value of the area under the Betti curve respectively marked in dark red and green. The two dashed lines represent the means of the areas under the curves for the two

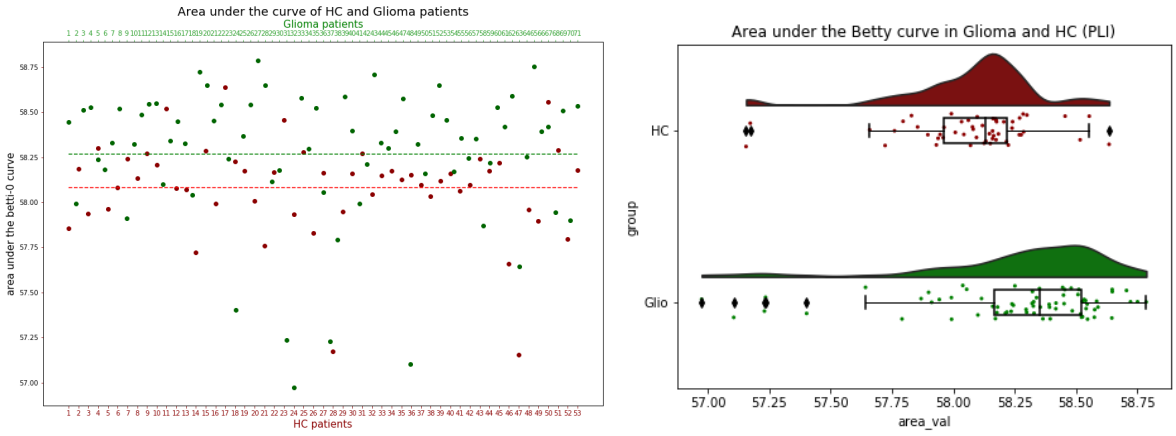


Figure 5.14. Areas under the Betti-0 curves for 53 HC and 71 Glioma patients. Left plot: scatter plot where, for each patient, we have the value of its area under the Betti-0 curve. Right plot: rain cloud plots which show the distribution of the area values for the two groups.

groups. On the right plot of Fig. 5.14, we have the *rain cloud plots* where we plot for each patient the value of its area under the curve. With this type of plot, we can better visualize the distributions of the values. The box plot under each of the two distributions indicates the mean area under the curve of that group.

The mean values of the areas under the curves for the two groups are:

	mean	error
HC	58.082	0.036
Glioma	58.266	0.045

where the error is computed as $\frac{\sigma}{\sqrt{N}}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\overline{x})^2}{N-1}}$, N being the number of individuals in that group.[3] To test if this result is statistically significant, since the data are not normally distributed, we used the Mann-Whitney test. The resulting p-value is p=0.000004, so the difference between the two groups is significant. Before going on with the analysis, we will explain the Mann-Whitney test.

Mann-Whitney test: With the Mann-Whitney test[4], we can answer the common question about two or more datasets: whether they are different or not. Specifically, whether the difference between their mean is statistically significant. In this test, the null hypothesis (H_0) is the assumption that both samples were drawn from a population with the same distribution, and therefore with the same population parameters, such as mean or median. Suppose that, after calculating the significance test on two or more samples, the null hypothesis is rejected. In that case, it indicates that there is evidence to suggest that the samples were drawn from different populations. So the difference between sample estimates of population parameters, such as means or medians, may be significant. The tests return a p-value that we use to interpret the result of the test. The p-value can be thought of as the probability that the two observed data samples were drawn from a population with the same distribution (null hypothesis H_0). The p-value can be interpreted in the context of a chosen significance level called alpha. A common value for alpha is 5% (or 0.05). Suppose the p-value is below the significance level. In that case, the test says there is enough evidence to reject the null hypothesis and that the samples were likely drawn from populations with differing distributions.
p $\leq$ alpha: reject H_0, different distribution.
p $\geq$ alpha: fail to reject H_0, same distribution.

We want now to study if there is a difference among the epochs. We did the following: instead of taking for each patient all the matrices and doing the mean, we took just the first half of them (corresponding to the first half of the epochs) and with them we computed the mean matrix for each patient. Then we started the procedure to calculate the Betti curve (from the C_X we computed the D_X and so the Betti curves) and finally, we could calculate the area under the curves of all the patients.

[3] All the errors appearing in the text are all computed with this formula.
[4] The following explanation is adapted from [12].

We repeated the same procedure two times. Firstly, taking for each patient the second half of its matrices and computing the mean and lastly taking half of the matrices of each patient in a random way and calculating the mean. All three results in Fig. 5.15 show that the mean of the areas under the curve for Glioma patients is greater than the HC. We can't appreciate a significant difference between the epochs, so we can conclude that the data are relatively homogeneous in the epochs.

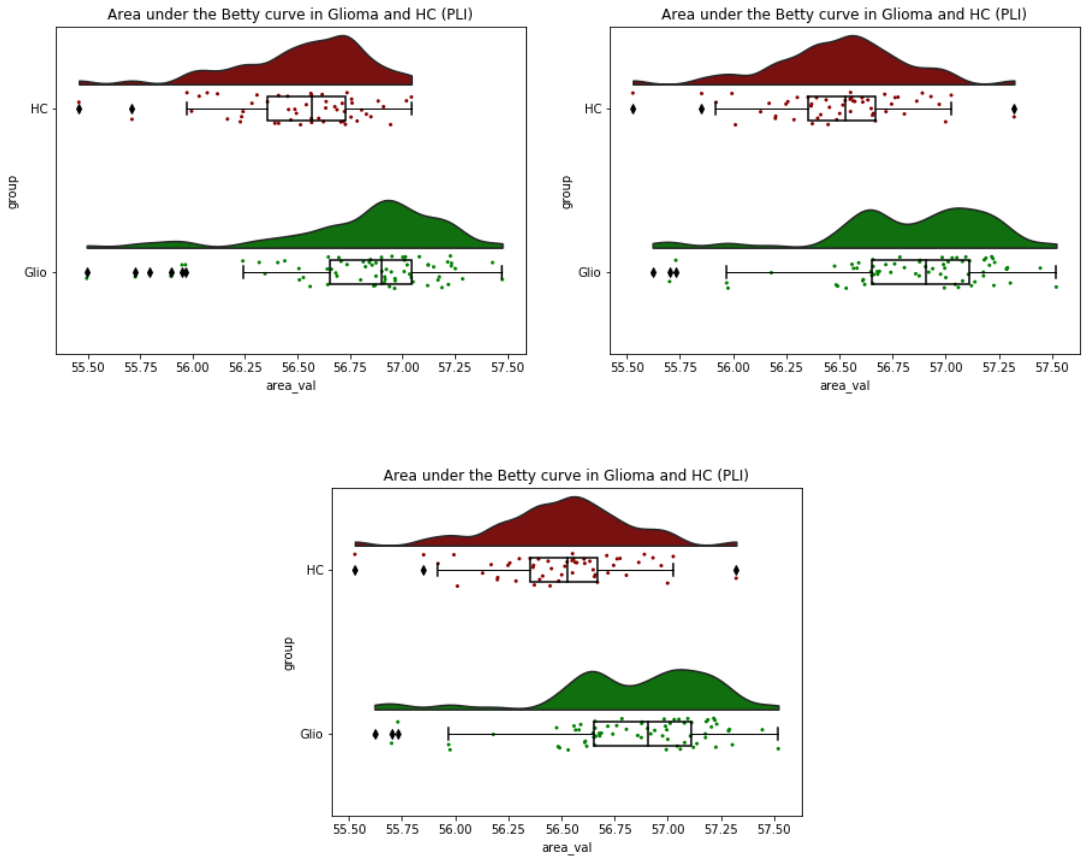


Figure 5.15. Areas under the Betti-0 curves of 53 HC and 71 Glioma patients. To do the calculation, for each patient, we took the mean of its matrices in three different ways: first half (left plot), second half (right plot), random half (bottom plot).

5.2.4 Effect of the brain tumour side on the Betti-curves

In this section, we want to understand the origin of the difference in the area under the curve between the two groups (Glioma patients and HC): we want to clarify if the difference is due to the tumour or not.

To achieve this goal, we do the following analysis: we exclude the side of the brain where the tumour is located (left or right), and we study only the other side of the brain. Then we compare its network to the brain network correspondent to the same side in the HC (we compare left sides with left sides and right sides with right

sides). For the analyzed side of the network, we look at how fast the nodes are clustering with the increment of the filtration value (Betti-0 curves), and we compare the curves of the two different groups. To compute the Betti-0 curves, we apply the usual procedure in which from the C_X we computed the D_X using the Python algorithm of Section 3.2. The only difference is that, in this case, the starting matrices have different dimensions because we need to consider just half of each matrix. The matrix representing the whole brain was 78x78, so we are now taking into account either the left side of the brain (nodes 1-39) or the right side (nodes 40-78). The new matrices have then shape 39x39. For each patient, we had 60 matrices, so we selected the side of the matrices we were interested in (obtaining 60 39x39 matrices for each patient) and then took the mean of the 60 half matrices in order to have one 39x39 matrix for each patient. Then we computed the Betti-0 curve with the known procedure.

Of the total 71 patients of the database, 25 patients have a tumour on the brain right side, 42 patients have a tumour on the brain left side and 4 patients have a tumour on both sides, these last were excluded from this analysis.

What we expect is that, if we consider the side of the brain where the tumour is absent, the network should be similar to the network of the HC: we should not see a difference between the two groups. Taking the patients who have the tumour on the right side (and so taking into account the left side of the brain), we obtain what we expected. The result is shown in Fig. 5.16.

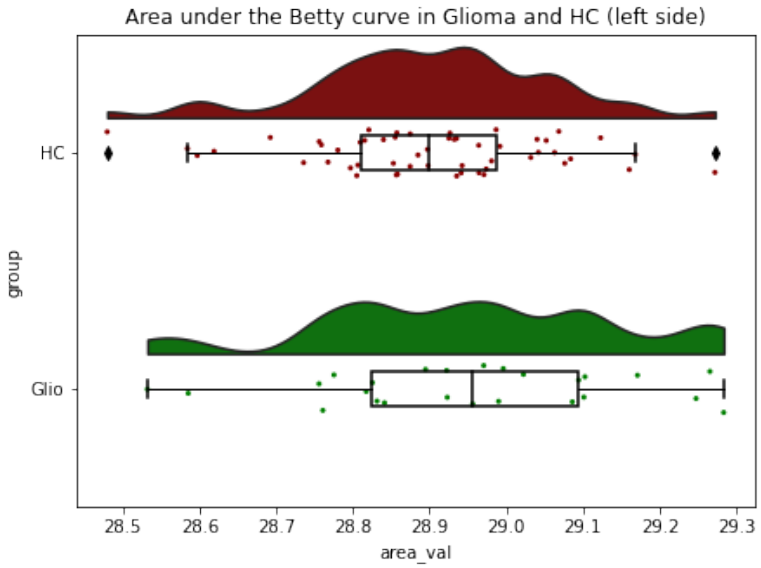


Figure 5.16. Areas under the curves for the two groups. Here, we are considering only the patients with a brain tumour on the right side. The curves for these Glioma patients were obtained taking into account only the left side of their brain (side without the tumour). Analogously, for the HC we considered only the left side of the brain.

The two mean values of the areas under the curves for the HC and Glioma are the same within the error:

	mean	error
HC	28.90	0.02
Glioma	28.95	0.04

Doing the Mann-Whitney test results in a p-value of p=0.135, so the two groups have the same distribution (fail to reject H_0). This result is in accordance with what we expected: the two groups can't be distinguished.

Taking the patients who have the tumour on the left side (and so taking into account the right side of the brain) we can not conclude the same thing. The result is shown in Fig. 5.17. The two group can be distinguished.

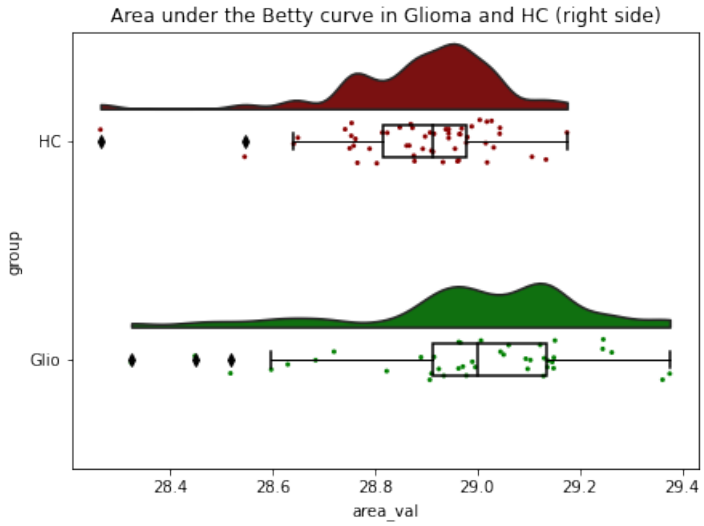


Figure 5.17. Areas under the Betti-0 curves for the two groups. Here, we are considering only the patients with brain tumour on the left side. So, the curves for these Glioma patients were obtained taking into account only the right side of their brain (side without the tumour). Analogously, for the HC we considered only the right side of the brain.

The mean and error of the two group are:

	mean	error
HC	28.89	0.02
Glioma	28.98	0.03

Doing the Mann-Whitney test results in a p-value of p=0.0009, so the two groups have different distributions (reject H_0). This result is in accordance with the known fact that the left tumour influence the network of the whole brain, so also the right side is changed by the tumour. The left tumour is more pervasive compared to the right tumour this is why it can influence the whole network. Even if we look at the brain right side where the tumour is absent, the network is different from the HC group.

We then considered the side of the brain where the tumour was located and compared it with HC. What we expected is that the difference would be significant.

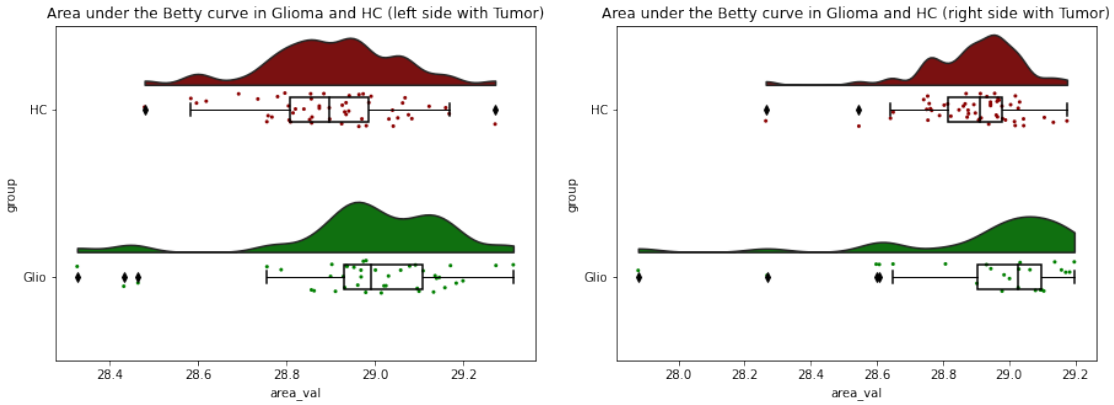


Figure 5.18. Comparison of the areas under the Betti-0 curves for the two groups. The curves for the Glioma patients were obtained taking into account the side of the brain where the tumour was located. Left plot: patients with brain tumour on the left side. Right plot: patients with brain tumour on the right side.

For the patients with the tumour in the left side (42 patients) we obtain the areas under the curves showed in the left of Fig. 5.18. The mean and error of the two groups are:

	mean	error
HC	28.90	0.02
Glioma	28.98	0.03

Doing the Mann-Whitney test results in a p-value of p=0.001, so the two groups have different distributions (reject H_0).

For the patients who have the tumour in the right side (25 patients) the results are shown in the right side of Fig. 5.18. The mean and error of the two groups are:

	mean	error
HC	28.89	0.02
Glioma	28.92	0.06

Doing the Mann-Whitney test result in a p-value of p=0.009 so the two groups have different distributions (reject H_0). Even if the Mann-Whitney test says that the two distributions are different, looking at the mean and error, we can't distinguish well the two means within the error. This is because the population for the Glioma patients is composed by just 25 elements so the error we assign to it is bigger then the other group (53 HC), the error is computed as $\frac{\sigma}{\sqrt{N}}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\overline{x})^2}{N-1}}$, N being the number of patients in that group.

To sum up we found that if in patients with right-side tumour we compare the left side of their brain (healthy brain side) with the HC group we can't see differences between the two groups, as expected. If in patients with left-side tumour, we compare the right side of their brain (which should be healthy) with the HC, we found that there is a difference. This is due to the fact that the left tumour also affects the right side of the brain. Lastly, when we compared for left tumour patients the side of their brain where the tumour is located, we could significantly differentiate them from HC. For the right tumour side, we expected the same thing, but since the sample is small (25 patients) we can't do proper statistics.

Chapter 6

Topological data analysis: a mathematical background

In the second part of the thesis, we will expand the use of the method of brain network analysis known as Topological data analysis (TDA) that we started approaching when we used Betti-0 curves. We will first explain why this tool is powerful in the study of brain networks and then give an overview of the sections of this chapter.

Functional brain networks are constructed by quantifying correlations between time series of activity of brain regions. We have seen that the brain can be modelled as a graph: a collection of nodes connected by edges.
To study brain networks, we can analyze local properties such as the degree of a vertex (number of nodes connected to one vertex) or global properties such as the average path length (average number of steps along the shortest paths for all possible pairs of network nodes). Knowing only local or only global properties doesn't tell us the structure of the graph: for example, two graphs can have the same average path length but be very different.
Topological data analysis (TDA) provides info about both the local and global properties of a graph and can help to understand in-depth the structure of a network.
To fully exploit TDA, we should model the brain with a more complex model. We consider that the complex network is related to a multidimensional structure, called the *simplicial complex*. It is constituted not only by the set of nodes (0-simplex) and edges (1-simplex) but also by triangles (2-simplex), tetrahedrons (3-simplex), and higher-dimensional structures. A simplicial complex is, therefore, a multidimensional structure that can be extracted from a functional brain network.
TDA is the emerging framework to process data sets under this perspective: it reveals structures of higher dimensions in the brain. TDA uses methods of topology and geometry to study the shape of data. In particular, a technique within TDA known as *persistent homology* provides a summary of the shape of the data at multiple scales (features such as k-dimensional holes, known as Betti numbers[1], can be revealed in the simplicial complex associated with the brain network). This perspective can help to understand the connection between the structure of neural connections and their function. A key success of persistent homology is the ability to provide robust

[1] In the following sections we will define these features in detail.

results, even if the data are noisy.

Our goal is to identify the number of k-simplex and k-holes in the brain of Glioma patients and healthy controls to enhance if there are any differences between the two groups. This analysis can help to understand the composition of the network in the brain and have some insight on Glioma. In this chapter we will first define *k-simplex* from a geometrical and algebraic point of view (Section 6.1). We will then define theoretically the *Boundary operator* and *Boundary matrix* which allows to count the numbers of holes (*Betti numbers*) in the complex structure (Section 6.2). In Section 6.3, we will define another topological invariant: the Euler characteristic and explain its importance in TDA. Lastly, in Section 6.4, we will explain the algorithm we implemented to perform these analyses of Betti numbers and Euler characteristics on the data.

6.1 Simplex complex

In this section, we will define mathematically the concept of *simplex complex* also called *simplicial complex*, which is the base of our analysis in this second part of the work.

In order to perform a topological analysis of real networks, the first step is to construct a simplicial complex starting from the brain network. A simplicial complex is a higher-order network structure, which includes interactions between two or more nodes, described by simplices. A complex simplex is not just formed by nodes (0-dimensional simplex) and edges (1-dimensional simplex) like a simple network; it is also composed by higher-dimensional simplices such as triangles (2-dimensional simplex), tetrahedrons (3-dimensional simplex), and so on.

A k-simplex μ is an object in k dimension and is formed by a subset of $k+1$ nodes of the total network. The subset of $k+1$ nodes which form the k-simplex is called *(k+1)-clique*. The clique is, therefore, a subset of the complex where all the nodes are connected.

Definition 14 *Let G be a network. A **clique** is an all-to-all connected collection of vertices in G. An k-clique is a clique consisting of k vertices. Note that if σ is a clique of G, then all subsets of σ are also cliques* [13].

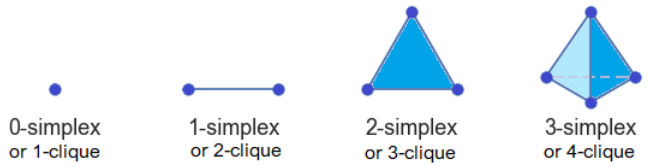


Figure 6.1. Simplices of different dimensions. From left to right: 0, 1, 2, 3 dimensional.

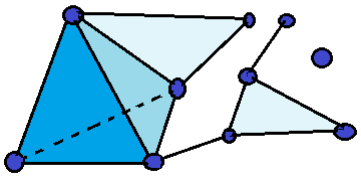


Figure 6.2. Example of a *simplicial complex* composed by: 10 nodes (0-simplex), 13 edges (1-simplex), 6 triangles (2-simplex), 1 tetrahedron (3-simplex).

To give a more precise mathematical definition, we say that a *simplicial complex* X is a topological space realized as a union of any collection of simplices (possibly of different dimensions) which has the following two properties:
(a) If a simplex σ belongs to the simplicial complex, i.e. $\sigma \in X$ then any simplex σ' formed by a subset of its nodes is also included in the simplicial complex, i.e. if $\sigma' \in \sigma$ then, $\sigma' \in X$;
(b) Given two simplices of the simplicial complex $\sigma \in X$ and $\sigma' \in X$ then either their intersection belongs to the simplicial complex, i.e. $\sigma \cap \sigma' \in X$ or their intersection is a null set, i.e. $\sigma \cap \sigma' = 0$ [14].

We can give an idea of 2-simplex in the brain with the following figure 6.3. When we increase the filtration value ϵ we obtain more connections, and so many 2-simplices arise. In Fig. 6.3, only 2-simplex are shown; however, the same can be done for higher-dimensional simplices like tetrahedron and so on.

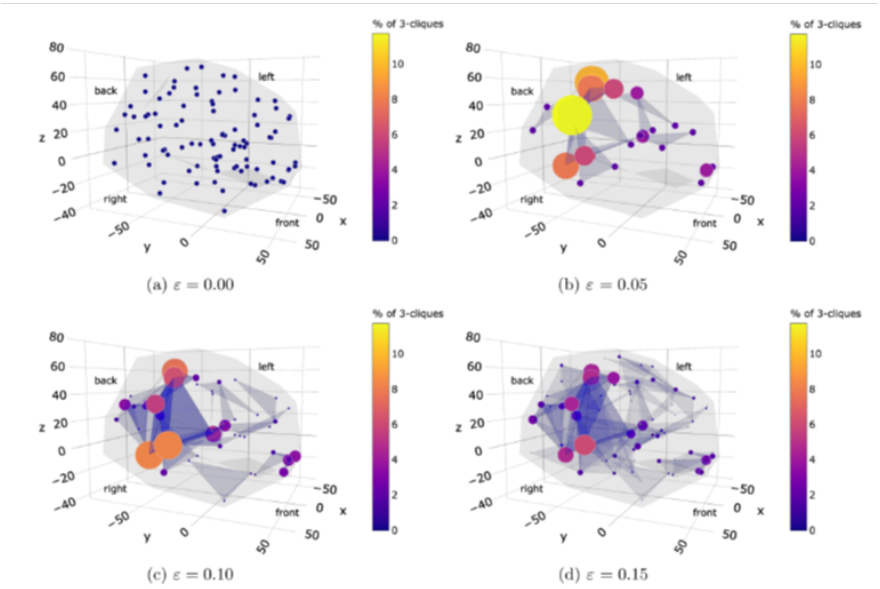


Figure 6.3. Here we can visualize the rising of the number of 2-simplex (triangles) in a brain network when increasing the filtration value ϵ. Here, for the biggest filtration value ($\epsilon = 0.15$, figure in the right bottom) we have a more significant number of 2-simplex compared to the cases where the filtration value is smaller ((a), (b), (c) in the figure). The figure is taken from [3].

6.2 Boundary matrix to compute Betti numbers

In this section we will define the *Boundary operator* and the *Boundary matrix* which allow us to count the number of k-holes in the complex structure of the network.

The topology of the brain network can be studied by computing the Betti numbers of the resulting simplicial complex. These are topological invariants derived from the simplicial complex and correspond, for each $i \geq 0$, to the number of linearly independent i-dimensional holes in the space [3].

- Betti-0 (B-0) is the number of connected components
- B-1 is the number of 2 dimensional holes (loops)
- B-2 is the number of 3-dimensional holes (voids)
- B-3 is the number of 4-dimensional holes

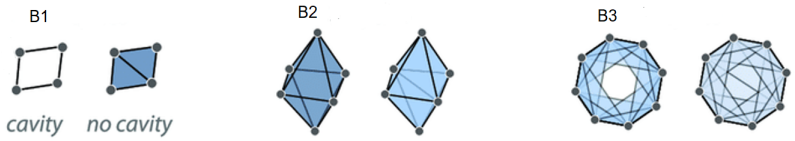


Figure 6.4. Betti numbers [13].

The simplex has an easy way to describe its boundary. From a geometrically perspective we can easily understand its definition: the boundary of the line segment is the two endpoints; the boundary of the triangle is the union of all three of its edges; the boundary of the tetrahedron is the union of its four triangular faces; etc. So the boundary of a clique $\sigma \subseteq G$ is the collection of sub cliques $\tau \subset \sigma$ which have one fewer vertex [13]. It corresponds to the set of lower-dimensional simplices that constitute the boundary of the simplex defined by σ. In Fig. 6.5 an example of boundary is given.

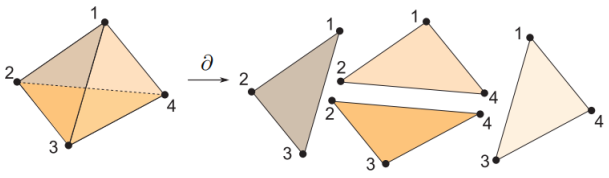


Figure 6.5. Example of the boundary of a complex [13].

Since we want to calculate Betti numbers in an algorithmic way, we must develop a formal method; the boundary matrix provides us with such an approach. Part of the following explanation is adapted from [15].
To start, we first want an algebraic way to describe holes and the boundary. In order to define an algebraic boundary, we also need to define the simplicial complex as algebraic objects.

Let X_k be the set of k-simplices in the simplicial complex X. Given a simplicial complex X, we define the k-th chain group $C_k(X)$ to be the $\mathbb{Q}$-vector space with X_k for a basis: it is the group of all possible formal sums of k-simplices in X. The elements of the k-th chain group are called k-chains on X.
If σ, σ' are two k-simplices, then we just define a collection of new "chains" as all possible "sums" and scalar multiples of the simplices. For example, sums involving two elements would look like aσ + bσ' for some a,b $\in \mathbb{Q}$ [15]. Indeed, we include any finite sum of such simplices, as is standard in taking the span of a set of basis vectors in linear algebra.

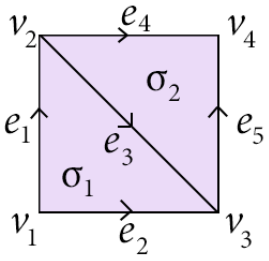


Figure 6.6. Example of a simplex complex formed by 4 vertex, 5 edges and 2 triangles [15].

Take the example of the simplicial complex in Fig. 6.6 where we can easily describe the $C_k(X)$ chains:

- $C_0(X)$ is the linear span of the set of vertices $\{v_1, v_2, v_3, v_4\}$. Geometrically we can think of the union of two points (v_1 and v_3) to be the sum $v_1 + v_3$.
- $C_1(X)$ is the linear span of the set $\{e_1, e_2, e_3, e_4, e_5\}$ with coefficient in $\mathbb{Q}$. We can take into account a possible path from v_1 to v_2 as the sum $e_1 + e_4 - e_5 - e_3$ where we use a negative coefficient when we want to indicate we are going against the orientation of the edge.
- $C_2(X)$ is the vector space $\{\sigma_1, \sigma_2\}$ formed by the combinations of the two triangles.

We can now define the *Boundary operator* on chain groups as a linear map:

$$\partial_k : C_k(X) \longrightarrow C_{k-1}(X).$$

The subscript of the ∂ is often omitted since the dimension can be deduced from the context, it is specified only when it is crucial to be clear which dimension is being talked about. To understand the boundary operator, we will first define it in a low dimension and then generalize it in high dimension.

Considering a 0-simplex, we have that a single vertex is boundariless so $\partial v = 0$ for each vertex. Extending to the entire chain, we have that ∂ on 0-chains $C_0(X)$ is identically the zero map.

For 1-simplex we have that, if the orientation of the 1-simplex (the edge) is (v_1, v_2) then the boundary is $\partial(v_1, v_2) = v_2 - v_1$: the front end minus the back end of

the edge. This defines the boundary operator on the basis elements, and we can again extend linearly to the entire group of 1-chains. So let's see how the boundary operator operates on a "path".
If we consider the path $e_1 + e_4 - e_5 - e_3$ the boundary is computed as

$$\partial(e_1 + e_4 - e_5 - e_3) = \partial e_1 + \partial e_4 - \partial e_5 - \partial e_3$$
$$= (v_2 - v_1) + (v_4 - v_2) - (v_4 - v_3) - (v_3 - v_2) = v_2 - v_1.$$

So the result is the end point of the path minus the starting point.

If the path is a *loop*, meaning that the starting and ending point are the same, then the boundary is zero. The condition for a chain to be a "loop" is if it is in the kernel[2] of the boundary operator: ∂(loop)=0. This particular type of chains are called *cycles*.

Let's now see the boundary operator for a 2-simplex. In a 2-simplex, a triangle with vertex (v_0, v_1, v_2) the boundary $\partial(v_0, v_1, v_2)$ should be $(v_1, v_2) - (v_0, v_2) + (v_0, v_1)$. Let's now check that if we apply the boundary operator to a boundary we obtain 0: a boundary is by definition boundariless.

$$\begin{aligned}\partial_1(\partial_2(v_0, v_1, v_2)) = \\ \partial_1((v_1, v_2) - (v_0, v_2) + (v_0, v_1)) = \\ \partial_1(v_1, v_2) - \partial_1(v_0, v_2) + \partial_1(v_0, v_1) = \\ (v_1 - v_2) - (v_0 - v_2) + (v_0 - v_1) = 0.\end{aligned}$$

The composition of the boundary operator in one dimension with the boundary operator in another dimension (say, $\partial_{k-1}\partial_k$), is usually written ∂^2 and we always have $\partial^2 = 0$. In the example we just saw we had $\partial_1(\partial_2(v_0, v_1, v_2)) = 0$. Every chain which is a boundary of a higher-dimensional chain is boundariless. Is easy to understand this statement in low dimension: as we saw above if we take the boundary of a 2-simplex, we get a cycle of three 1-simplices, and the boundary of this chain is zero.

We can now generalize the definition of boundary operator to high order dimension. Let $[0, 1, \ldots, k]$ be a k-simplex with k vertices. We can denote the removal of a vertex from this list by putting a hat over the removed index. So $[0, 1, \ldots, \hat{i}, \ldots, k]$ represents the simplex which has all of the vertices from 0 to k excluding the vertex v_i. The boundary operator is defined via the alternative sum

$$\partial([0, 1, \ldots, n]) = \sum_{k=0}^{n} (-1)^k [0, \ldots, \hat{k}, \ldots, n]. \tag{6.1}$$

Now that we have defined the boundary operator for the basis elements of a chain group, we automatically get a linear operator on the entire chain group by extending ∂ linearly on chains [15].

[2] Let V,W be vector spaces $f : V \longrightarrow W$, $ker(f) := \{v \in V : f(v) = 0 \in W\}$, $ker(f) \subseteq V$. The kernel of a linear map f is the group of vectors of the starting vector space V that are mapped by f to the element 0.

Definition 15 *The k-cycles on X are those chains in the kernel of ∂. We will call them k-cycles boundariless. The k-boundaries are the image*[3] *of ∂, im(∂).*

We call the k-cycles $Z_k(X) = ker(\partial_k)$ and this is a subspace of $C_k(X)$. The set $B_k(X)$ of k-boundaries, that is, the image of ∂_{k+1}, is a subgroup of $Z_k(X)$. This is because, as we just saw, every boundary is itself boundariless, so $B_k(X)$ is a subset of $Z_k(X)$. All of these data can be expressed in one big diagram: each of the chain groups are organized in order of decreasing dimension, and the boundary maps connect them.

$$\cdots \xrightarrow{0} 0 \xrightarrow{\partial_3 = 0} C_2(X) \xrightarrow{\partial_2} C_1(X) \xrightarrow{\partial_1} C_0(X) \xrightarrow{\partial_0 = 0} 0 \xrightarrow{0} \cdots$$

The condition that $im\partial_{k+1} \subset ker\partial_k$, which is equivalent to $\partial^2 = 0$, is what makes this sequence a chain complex.

Geometrically we say that the holes are all those cycles (loops) that don't arise as the boundaries of higher-dimensional objects. In algebraic terms, this would correspond to a quotient space of the k-cycles by the k-boundaries. That is, a cycle would be considered a "trivial hole" if it is a boundary, and two "different" cycles would be considered the same hole if their difference is a k-boundary. This is the spirit of homology, and formally, we define the homology group (vector space) as follows.

Definition 16 *The k-th homology group of a simplicial complex X, denoted $H_k(X)$, is the quotient vector space $Z_k(X)/B_k(X) = \ker(\partial_k)/\mathrm{im}(\partial_{k+1})$. Two elements of a homology group which are equivalent (their difference is a boundary) are called homologous* [15].

The number of k-dimensional holes in X is thus realized as the dimension of $H_k(X)$ as a vector space. The dimension of $H_0(X) = ker(\partial_0)/im(\partial_1)$ is the number of connected components of X. Therefore, computing homology generalizes the graph-theoretic methods of computing connected components. The dimension of $H_1(X)$ is the number of 2D holes, $H_2(X)$ of 3D holes etc.

Let's sum up what we did so far: we built up a topological space as a simplicial complex (triangles glued together), we defined an algebraic way to represent collections of simplices called chains as vectors in a vector space, we defined the boundary homomorphism ∂_k as a linear map on chains, and finally defined the homology groups as the quotient vector spaces.

Because the chain groups are vector spaces, and the boundary mappings are linear maps, they can be represented as matrices (the so-called *Boundary matrices*) whose dimensions depend on our simplicial complex structure. We will now define the boundary matrix.

[3] Let V,W be vector spaces and $f : V \longrightarrow W$ a linear transformation.
$im(f) := \{w \in W : \exists\ v \in V \quad \text{for which} \quad f(v) = w$, $im(f) \subseteq V$. The image of a linear map f is the subgroup of vectors of W, which are image through f, of the vectors of V.

Remember that we have $Z_k = ker(\partial_k)$ and $B_k = im(\partial_{k+1})$. By the rank-nullity theorem[4] we have

$$n_k = z_k + b_{k-1}, \tag{6.2}$$

where n_k is the number of k-simplex in X and z_k, b_{k-1} are the rank of Z_k and B_{k-1} respectively. We can compute the Betti number as

$$B_k = z_k - b_k = dim(ker\partial_k) - dim(im\partial_{k+1}). \tag{6.3}$$

We can calculate the B_k for each k as long we can decompose n_k as Eq. 6.2 [16]. Let's see how we can procede to compute Betti with the boundary matrix. Let $C_k = span\{\alpha_1, \alpha_2, \dots \alpha_{n_k}\}$ and $C_{k-1} = span\{\tau_1, \tau_2, \dots \tau_{n_{k-1}}\}$ we define the boundary matrix

$$M_k = \begin{array}{c} \\ \tau_1 \\ \tau_2 \\ \vdots \\ \tau_{n_{k-1}} \end{array} \begin{array}{c} \begin{array}{cccc} \alpha_1 & \alpha_2 & \dots & \alpha_{n_k} \end{array} \\ \begin{pmatrix} a_1^1 & a_1^2 & \dots & a_1^{n_k} \\ a_2^1 & a_2^2 & \dots & a_2^{n_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_{k-1}}^1 & a_{n_{k-1}}^2 & \dots & a_{n_{k-1}}^{n_k} \end{pmatrix} \end{array}$$

where the row labels are the basis for $C_{k-1}(X)$ and the column labels are the basis for $C_k(X)$. In the matrix, $a_i^j = 1$ if and only if the ith $(k-1)$-simplex τ_i is in the face of the jth k-simplex α_j, otherwise $a_i^j = 0$ [16]. Then, we have

$$\partial_k \alpha_j = \sum_{i=1}^{n_{k-1}} a_i^j \tau_i.$$

From an algebraic point of view, if we use $\{\tau_1, \tau_2, \dots \tau_{nk-1}\}$ as the basis of B_{k-1}, the jth column of M_k is just the coordinate of $\partial_k \alpha_j$ under that basis. Thus, the kth boundary matrix actually encodes all the possible relationship between kth simplices and their boundaries. Given a kth chain c in X, we have $c = \sum_{j=1}^{n_k} c_j \alpha_j$ and its boundary is

$$\partial_k c = \partial_k \sum_{j=1}^{n_k} c_j \alpha_j = \sum_{j=1}^{n_k} c_j \partial_k \alpha_j = \sum_{j=1}^{n_k} c_j \sum_{i=1}^{n_{k-1}} a_i^j \tau_i = \sum_{i=1}^{n_{k-1}} \sum_{j=1}^{n_k} (a_i^j c_j) \tau_i.$$

Thus, under the basis $\{\tau_1, \tau_2, \dots \tau_{n_{k-1}}\}$, the coordinate of $\partial_k c$ is

$$\begin{bmatrix} a_1^1 & a_1^2 & \dots & a_1^{n_k} & \\ a_2^1 & a_2^2 & \dots & a_2^{n_k} & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n_{k-1}}^1 & a_{n_{k-1}}^2 & \dots & a_{n_{k-1}}^{n_k} & \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n_k} \end{bmatrix}.$$

[4] Let V,W be vector spaces and $T : V \longrightarrow W$ a linear transformation. The rank-nullity theorem states that $Rank(T) + Nullity(T) = dim(V)$, where $Rank(T) = dim(ImT)$ and $Nullity(T) = dim(kerT)$ so we have $dim(ImT) + dim(kerT) = dim(V)$.

Since it is true for any k chain in X, we have B_{k-1}=$im(\partial_k)$=$span\,\{col_1, col_2, ..., col_{n_k}\}$, where col_j is the jth column of boundary matrix M_k. With the knowledge of linear algebra, we have $b_{k-1} = dim(B_{k-1}) = rank(M_k)$[5]. We can see that we have just related b_{k-1} to the rank of the boundary matrix. The rank is calculated through the known linear algebra technique, the Gaussian elimination.
To compute the Betti with the boundary matrix we do the following: after the reduction of matrix ∂_k and the reduction of ∂_{k+1}, we count the number of columns/rows with all 0. If the number of pivots arising in ∂_k is y and the number of pivots arising in ∂_{k+1} is z, and the dimension of $C_k(X)$ is n, then the dimension (the number of Betti-k) is exactly

$$(n - y) - z = dim(ker\partial_k) - dim(im\partial_{k+1}). \tag{6.4}$$

6.2.1 Example of calculation of Betti number

Before we move forward, the following example will help to illustrate the concept we discussed so far. Consider the simplex complex in Fig. 6.7.

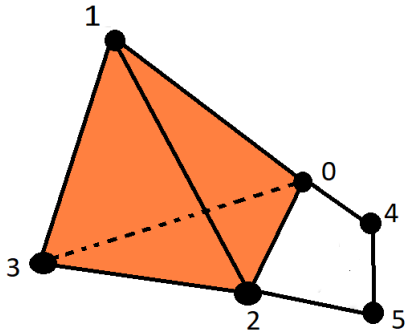


Figure 6.7. Simplex complex constituted by a total of 6 vertices, where 4 of them (vertex 0,1,2,3) are forming a tetrahedron (3-simplex).

We will compute the dimension of H_1, which corresponds to the number of Betti-1 (2D holes) we have in this simplex complex. For this complex we have

- $C_0(X) = \text{span}\,\{0, 1, 2, 3, 4\}$
- $C_1(X) = \text{span}\,\{[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [1, 3], [2, 3], [2, 4]\}$
- $C_2(X) = \text{span}\,\{[0, 1, 2], [0, 1, 3], [0, 2, 3], [1, 2, 3]\}$

Given our known definitions of ∂_k as an alternating sum (Eq. 6.1), we can give a complete specification of the boundary map as a matrix. For ∂_1, this would be

$$\partial_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccccccc} [0,1] & [0,2] & [0,3] & [0,4] & [1,2] & [1,3] & [2,3] & [2,5] & [4,5] \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array},$$

[5]The rank of a matrix is the maximum number of linear independent columns of the matrix.

where the row labels are the basis for $C_0(X)$ and the column labels are the basis for $C_1(X)$. Given a column of the matrix, notice that we have 1 or -1 if the element in the rows is part of the element of the label of that column; otherwise, we have 0. Similarly, ∂_2 is

$$\partial_2 = \begin{array}{c} \\ [0,1] \\ [0,2] \\ [0,3] \\ [0,4] \\ [1,2] \\ [1,3] \\ [2,3] \\ [2,5] \\ [4,5] \end{array} \begin{array}{c} \begin{array}{cccc} [0,1,2] & [0,1,3] & [0,2,3] & [1,2,3] \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Just to check if the matrices are correct we can see that in the figure $\partial_1[0,1] = 1 - 0$ and this corresponds, correctly, to first column of ∂_1. Similarly $\partial_2[0,1,2] = [1,2] - [0,2] + [0,1]$ and this corresponds to the first column of ∂_2. Remember the crucial property of ∂, that $\partial^2 = \partial_k \partial_{k+1} = 0$. Indeed, the composition of the two boundary maps just defined corresponds to the matrix product of the two matrices; it can be verified that the product of these two matrices is the 0 matrix.

We know from basic linear algebra that to compute the kernel of a linear map expressed as a matrix, we need to column reduce and inspect the columns of zeros. So to carry the calculation, we reduce ∂_1 and ∂_2 with the Gaussian reduction technique and at the end we count the number of columns/row with all 0. The two reduced matrices are:

$$\partial_{1_{reduced}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \partial_{2reduced} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

After the reduction technique, ∂_1 has five "pivots", and this means that the rank of the matrix is 5. Moreover, the five basis vectors representing the columns with non-zero pivots, which we'll call v_1, v_2, v_3, v_4, v_8, span a complementary subspace to the kernel of ∂_1. Hence, the remaining four vectors (which we will call v_5, v_6, v_7, v_9) span the kernel. In particular, this says that the kernel has dimension 4: $dim(ker\partial_1) = 4$.

On the other hand, we performed the same transformation of the basis of $C_1(X)$ for ∂_2. Looking at the matrix that resulted, we see that the last six rows, (representing $v_4, v_5, v_6, v_7, v_8, v_9$) are entirely zeros and the remaining three rows (representing v_1, v_2, v_3) have non-zero pivots. This tells us exactly that the image of ∂_2 is spanned

by v_1, v_2, v_3, so $dim(im\partial_2) = 3$. The quotient to get homology is simply

$$H_1(X) = \frac{ker(\partial_1)}{im(\partial_2)} = \frac{\text{span}\{v_5, v_6, v_7, v_9\}}{\text{span}\{v_1, v_2, v_3\}},$$

and the dimension of this homology group is 1. This corresponds to the number of Betti-1= $dim(ker\partial_1) - dim(im\partial_2) = 4 - 3 = 1$. In the simplex, in fact, there is just 1 2D hole (defined by the vertex 0, 2, 4, 5).

To sum everything we did, we can say that the algorithm to compute homology for simplicial complexes is based on Gaussian reduction of the matrices representing the boundary operators ∂_k and ∂_{k+1} with respect to the canonical basis. After the reduction we count the entries on the diagonals of the two matrices (the number of non-zero columns/rows) and their difference will be the dimension of H_k and so the number of k-holes (k-Betti).

6.3 Euler characteristic and transitions

In this section, we will define from a theoretical point of view the Euler characteristic and transitions in the network. We will then apply these concepts to data analysis in the next chapter.

Mathematics research in algebraic topology focuses on finding topological invariants which are numbers that we assign to shapes (or simplicial complex in our case), to learn something about their global structure. Euler characteristic is one of these topological invariants. Let's first give an intuitive explanation. In three-dimensional polyhedrons (for example a cube, a tetrahedron, etc.) the Euler characteristic is defined as numbers of vertices - edges + faces. For objects without holes, the Euler characteristic is always 2, as shown in Fig. 6.8.

Name	Image	Vertices V	Edges E	Faces F	Euler characteristic: $V-E+F$
Tetrahedron		4	6	4	**2**
Hexahedron or cube		8	12	6	**2**
Octahedron		6	12	8	**2**
Dodecahedron		20	30	12	**2**
Icosahedron		12	30	20	**2**

Figure 6.8. Euler characteristic in polyhedrons. Note that when there are no holes in the shapes, the Euler is always equal to 2.[6]

If we take the cube and make a hole, the Euler drops to 0 as it is in the torus. If we make two holes in the torus, the Euler drops to -2. We can easily understand that the Euler characteristic tells us something about the topology of a surface.

We can now generalize the definition to simplicial complex in any dimension. The Euler characteristic of the functional brain network of each individual is expressed by the alternate sum of the numbers $f_k(\epsilon)$ of k-simplex present in the simplicial complex of the network for a given value of the threshold ϵ.

$$\chi = f_0 - f_1 + f_2 - ....f_n = \sum_{k=1}^{N} (-1)^{k+1} f_k(\epsilon). \tag{6.5}$$

It is also defined as the alternative sum of the Betti numbers

$$\chi = B_0 - B_1 + B_2 \ldots. \tag{6.6}$$

The *Euler entropy* is obtained from

$$S_\chi = ln|\chi|. \tag{6.7}$$

When $\chi = 0$ for a given value of the threshold in the network, the Euler entropy is singular, $S_\chi \longrightarrow \infty$. We will show in the next chapter that a zero of χ and so a singularity of S_χ are related to a significant topological change in the network. In fact, a topological phase transition in a complex network takes place at the point where the Euler characteristic is null, and the Euler entropy is singular. This statement finds support in the behaviour of S_χ at the thermodynamic phase transitions across various physical systems [3]. Moreover, we show below that the behaviour of the Betti numbers, another set of topological invariants, also concurs to verify this statement. In fact, each time that we have a transition, a Betti-number of higher order arises in the structure of the brain complex network.

6.4 Algorithm to compute Betti numbers and Euler

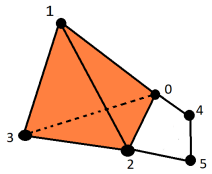
In this section, we will explain how we implemented the algorithm to compute the Betti numbers and the Euler characteristic and illustrate the most critical steps.
In Section 6.2, we explained how to compute the Betti-number of a network. The algorithm reproduces the same steps of the theoretical explanation: it counts the number of k-simplex in the graph, defines the boundary matrices needed for the computation, and after the Gaussian reduction it counts the rank of the matrices in order to compute Betti numbers.

Here we will explain the general working of the algorithm and, in Appendix B.1, the complete code with detailed comments is given. The algorithm is composed by 5 fases:

[6] https://en.wikipedia.org/wiki/Euler_characteristic.

1. Prepare maximal cliques
2. Enumerate all simplices
3. Construct the boundary operator
4. Compute rank and $dim(\ker)$ of the boundary operators
5. Compute the Betti number

1. In the first step, we find all the *maximal clique* in the graph, and we create a list containing the different maximal cliques sorted from the smallest to the largest (line 25, 29 code B.1). Let's understand the concept of the *maximal clique*. A maximal clique is a clique that includes the largest possible number of vertices and can not be included in a greater clique. For example, if we have 4 nodes all connected (4-clique), also the different subsets of 3 nodes will be all connected (3-clique), but in the maximal clique list, we will consider only the 4-clique because it is the maximum number of connected nodes. If we take the same example of Section 6.2 (Fig. 6.7),



the list of maximal cliques of that complex is $C : [(0,1,2,3),(0,4),(2,5),(4,5)]$.

2. In the second step we do the following: for every element in C, we do all the possible combinations to find all the k-sub cliques of each maximal clique. We create different lists S_k where we store these sub cliques. S_k is the list containing the cliques of dimension k (for loop, line 48, 50 B.1).
We want to enumerate all this k-simplex, so we create a dictionary where each key is an element (a clique) of S_k and the value is an index. We then put this dictionary in an S list (list of dictionaries). The length of the dictionary depends on how many dimensions of k-simplex we have (line 58 B.1).
The dictionary for our example is:
S=$[\{(0):0,(1):1,(2):2,(3):3,(4):4,(5):5\}$,
$\{(0,1):0,(0,2):1,(0,3):2,(0,4):3,(1,2):4,(1,3):5,(2,3):6,(2,5):7,(4,5):8\}$,
$\{(0,1,2):0,(0,1,3):1,(0,2,3):2,(1,2,3):3\}]$.

3. In the third step, since we want to build the boundary matrix, what we want is to find, for each k-simplex (label of the columns of the matrix) all the (k-1)-simplex contained in it. And then fill the column of the matrix with 1 or -1 (if contained) and 0 (otherwise). The (k-1)-simplex are the labels of the rows. The ∂ matrix has dimension $S_{k-1} \times S_k$. So we create a matrix $D[k] = np.zeros(lenS[k-1], lenS[k])$, (line 85 B.1).
To fill the matrix, we look at every element of the S[k] dictionary, and for each k-clique we do all the possible combinations to build the (k-1)-cliques. We then find in the dictionary S the indexes of these (k-1)-cliques, and we store them in a list (line 89, 92 B.1). Then, in correspondence of these indexes in the column of the

boundary matrix, we will fill with 1 or -1 (line 95, 96 B.1). After looking at every element of the dictionary and repeating this procedure, the boundary matrix will be entirely filled. We then do the same for S[k+1] so we have the matrix ∂_k and ∂_{k+1}.

4. In the fourth step we do the Gaussian reduction and then count the rank of the two matrices (line 111 B.1). We compute $dim(im\partial_{k+1})$ and $dim(ker\partial_k)$ and finally (line 122) we can compute the Betti number as $dim(ker\partial_k) - dim(im\partial_{k+1})$ (Eq. 6.3).

To compute the Euler characteristic, the first two steps of the algorithm are the same. Then, in the third step, we simply count the number of k-simplex in each S_k list which corresponds to compute *len(S_k)*. Finally we compute the Euler characteristic by applying the formula in Eq. 6.5 (see end of Appendix B.1). In Fig. 6.9, we can visualize the increment of connections in the brain when the filtration value increases. The computation of the Euler characteristic for each network is reported.

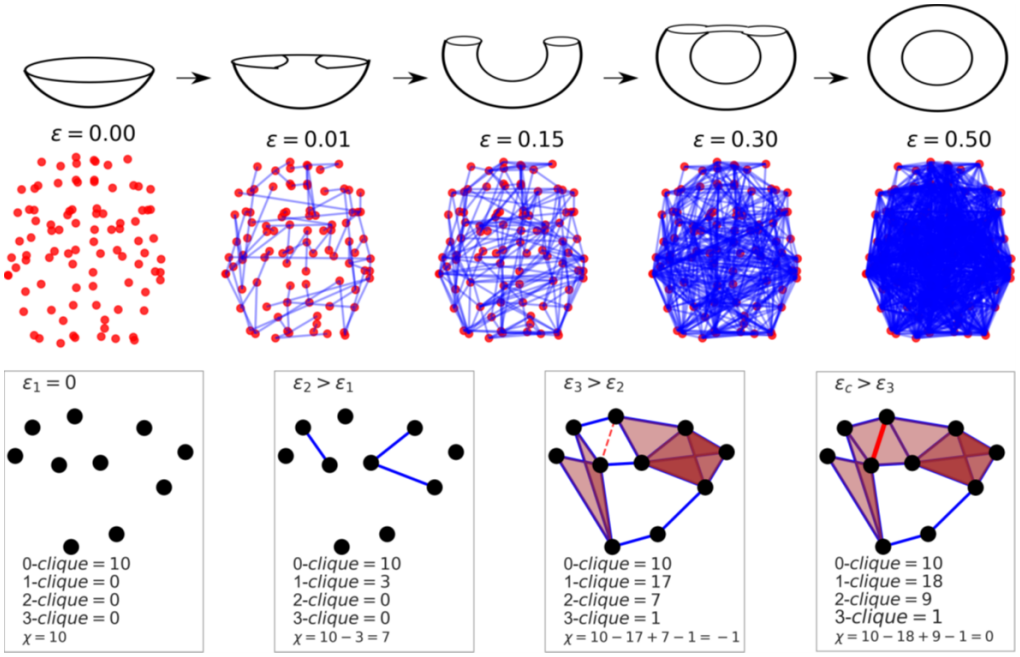


Figure 6.9. Representation of the brain network filtration and correspondent number of k-cliques. When the filtration value increase, there is a rise in the number of connections in the network. The Euler characteristic χ is computed each time. Figure taken from [3].

Chapter 7

Applying TDA to data

Our goal is to investigate the phase transitions in functional brain networks, which remarkably coincides with the emergence of multidimensional topological holes in the brain network. In this chapter, we will show that the Euler characteristic, a topological invariant, is able to characterize the sequence of phase transitions in the complex network and that the transitions occur when the Euler entropy has a singularity. We will compute the Betti-1 curves, and we will also relate the Euler characteristic to the Betti numbers: the Euler is, in fact, an approximation of the Betti curves. To show this, we will first start by analyzing these quantities in the Erdos-Rényi random networks (Section 7.1). We will then look at our generated data (Section 7.2) and finally apply this method to the Amsterdam UMC database of Glioma and HC patients (Section 7.3).

Due to the universal character of phase transitions and noise robustness of TDA, our findings open perspectives toward establishing reliable topological and geometrical markers for groups and possibly individual differences in functional brain network organization. With this method, we were able to distinguish the HC and Glioma patients at a group level. These differences regard the areas under the Euler curves, the position of the transitions, and the Betti-1 curves.

7.1 Random networks

In this section, we will analyze topological invariant as Betti numbers and Euler characteristic on random networks (Erdos–Rényi graph). We will start by defining the characteristics of an Erdos–Rényi graph.

From theoretical studies on simplex complex, we know that a given Betti number arises in a given range of numbers of connections [13]. Increasing the links will cause the appearance of Betti numbers of higher-order, and smaller Betti numbers will disappear.
To test this statement known from theory, we created a random network, the Erdos–Rényi graph. In this type of random network, the connections between nodes occur with probability p. The probability corresponds to the chance that a link is created between two nodes. We can thus investigate the evolution of complex random

networks (Erdos-Rényi graphs) for fixed N nodes as a function of the probability parameter $p \in [0,1]$. Increasing the probability we will obtain a more connected graph. If p=0, then no nodes are connected (empty graph), whereas if p=1, then all nodes are connected to each other and we obtain a so-called *complete graph*. The number of connections in a complete graph is $N(N-1)/2$.

In an Erdos-Rényi graph, it is possible to identify subgraphs containing k-cliques with k all-to-all connected nodes and then determine its Euler characteristic. The mean value of the Euler characteristic of Erdos-Rényi graphs with N nodes and linking probability p is exactly given by the alternate sum:

$$\langle \chi \rangle = \sum_{k=1}^{N} (-1)^{k+1} \binom{N}{k} p^{\binom{k}{2}}. \tag{7.1}$$

In the above expression $\binom{N}{k} p^{\binom{N}{2}}$ is the mean number of k-simplex since there are $\binom{k}{2} = k(k-1)/2$ links in a k-clique that occur with probability $p^{\binom{k}{2}}$. Also, the possible number of choices of k nodes from a total of N is $\binom{N}{k} = N!/(N-k)!k!$ [3].

To create our random graphs we choose arbitrarily to set the number of nodes to 25, and then we create a network for each given probability p in the interval [0,0.8]. For each network, we compute the Euler characteristic χ (defined as Eq. 6.5) and the Betti numbers (Betti-0, Betti-1, Betti-2, Betti-3), using the algorithm implemented in Python described in Section 6.4.
We repeated the generation of random networks ($p \in [0, 0.8]$) 10 times. In each experiment, for each network (with a given p), we computed the mentioned quantities. We obtained 4 curves for each experiment (Euler, Betti-0, Betti-1, Betti-2, Betti-3 curves). Since we have 10 experiments, for each of the 4 observed quantities, we have 10 curves; we computed the mean curve (with errors[1]) for each of them. In Fig. 7.1 we show the obtained result.

[1]The error associated with the mean curve was calculated as $\frac{\sigma}{\sqrt{N}}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}}$, N being the number of experiments.

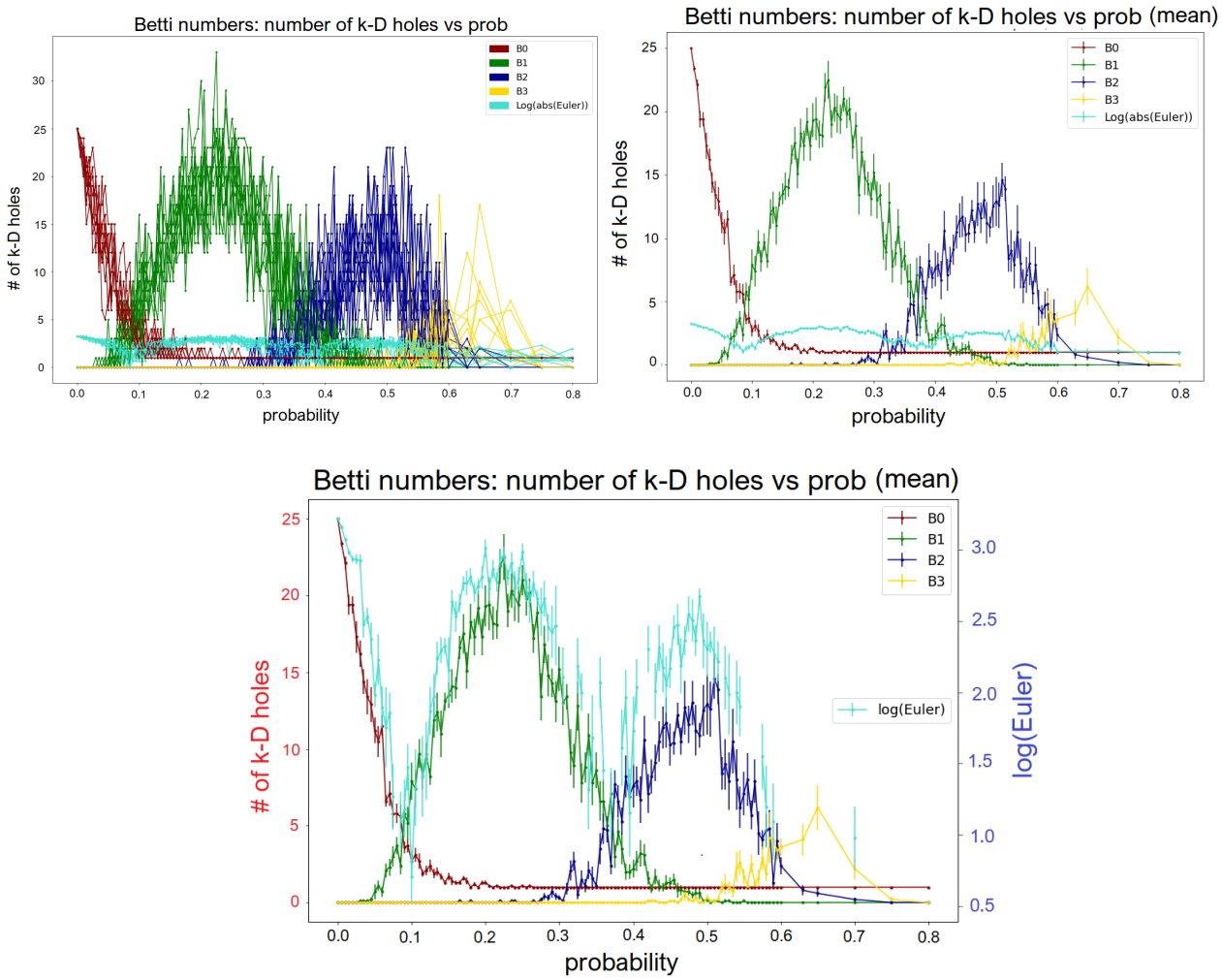


Figure 7.1. For each probability, we create a random network, and we compute the Betti numbers and Euler. We repeated the experiment 10 times (left figure) and computed the mean curves with errors (right figure). In the last plot (bottom figure), the right y-axis and left y-axis have different scales to better visualize the Euler characteristic in relation to the Betti numbers.

We first analyze the Euler curve in turquoise in Fig. 7.1. In the figures, we have plotted the logarithm of the value of Euler characteristic $S_\chi = ln(\chi)$. We notice the presence of several singularities in S_χ associated with the many zeros of the Euler characteristic. To avoid problems with Python when $\chi = 0$ and so $S_\chi = ln(0) = -\infty$ we rescaled the value and we computed and plotted *ln(abs(Euler)+1))*. So when we have 0 in the plot, it means there is a singularity in the Euler.
We notice that the topological phase of the network that sets in between the kth and $(k+1)th$ transitions (when we have a singularity in the Euler) corresponds very closely to the range in p where the Betti number B_k prevails. This means that the Euler characteristic is in the same range of the Betti numbers and when we have a transition a higher order Betti number appears.

As p increases and so the number of connections is greater, we find a sequence of dominant Betti numbers B_k, starting from k=0, that change (i.e., k is incremented by one unity) every time a topological phase transition is crossed. While the location of the transitions is determined by the singularities of the Euler entropy S_χ, the Betti numbers B_k characterize which kind of multidimensional hole prevails in each topological phase. In Fig. 7.1 for instance, we note that for N=25 and a range of p $0.09 \leq p \leq 0.36$ (between the first and second transitions) B_1 is much larger than all other B_k's. This means that 2-dimensional holes (k=1) are abundant in such random networks in this range of p. Then B_1 disappears and B_2 prevails and so on.

We also studied the absolute value of the Euler characteristic. So instead of looking at *log(abs(euler)+1))* we took *abs(euler)*. The absolute value of the Euler characteristic is a good approximation of the Betti numbers, as shown in Fig. 7.2. In the figure, the mean of 10 experiments is shown (we omit the plot of the 10 experiments).

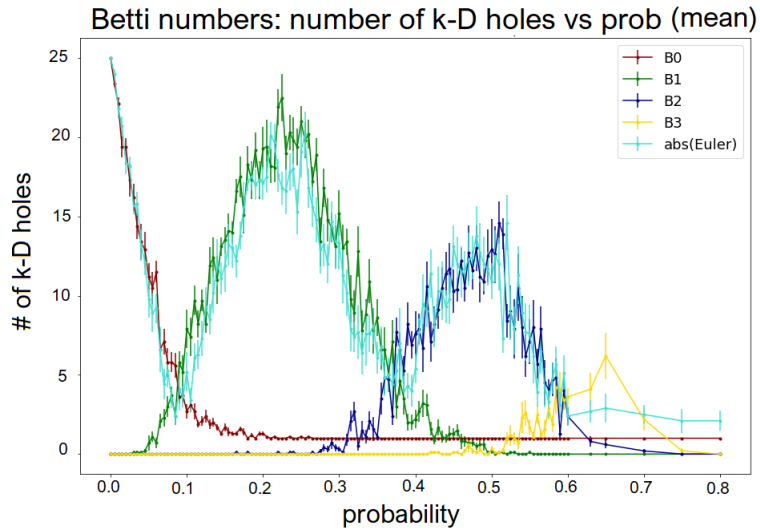


Figure 7.2. For each probability, we create a random network, and we compute the Betti numbers and the absolute value of the Euler characteristic. We repeated the experiment 10 times and computed the mean curves with errors. In this plot, we only show the mean (with errors) of the 10 experiments for the Euler and Betti curves. We notice that the absolute value of Euler is a good approximation of Betti numbers.

Being able to use this approximation is computationally convenient because the computation of the Euler characteristic is faster than the Betti numbers. To compute the Euler, we only need to compute the number of k-simplex but not the boundary matrix as is the case of the Betti numbers. In Section 6.4, where we explained the algorithms, it was immediate to note that the Euler is computationally less expensive since to be computed it only needs the preliminary part of the algorithm needed to compute the Betti numbers. The most expensive part of the algorithm, from a run time and needed memory point of view, is the computation of the boundary matrices and the Gaussian reduction of matrices. In case of many connections between nodes (very high number of simplex), the matrix can have shape 1800x2200 and even more,

and so it is computationally heavy. If we want to analyze higher order Betti numbers, we can instead compute the Euler characteristic since this approximation allows us to save computational power.

Another way to check the theory which states that when transitions in the Euler curve occur we have the rise of a new Betti number, is the following procedure. We take a network with 25 nodes and increase the density. The density is defined as the number of connections/total possible number of connections. We gradually increase the density and compute each time how many Betti-k numbers we have. Here, we don't need to repeat the experiment. In fact, if we repeated the experiment 10 times, we would always obtain the exact same networks because we are not looking at the probability of connections of a single pair of nodes as is the case of p in the Erdos-Rény. In fact, each time we increase the density, we are deciding in advance the exact number of connected nodes. In Fig. 7.3 we notice that the Euler transitions occur when a new Betti-k arise confirming again what we know from theory.

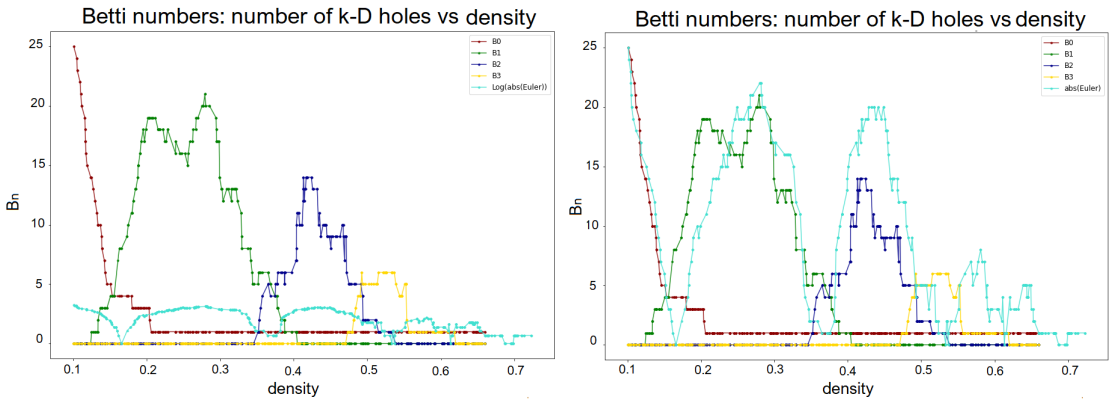


Figure 7.3. We created a network with 25 nodes. We increase the number of connections gradually and compute, each time, the Betti numbers and the Euler characteristic. In the left figure we plot *log(abs(Euler)+1)* in the right figure we plot *abs(Euler)*.

7.2 Simulated data: C_1-C_{10}

In this section we apply the algorithm to count Betti-1 and Betti-2 to the simulated networks of Chapter 4. Remember we have networks obtained from 10 different configurations of nodes $\{C_1, C_2, C_3, \dots C_{10}\}$.

In Fig. 7.4, we show the plot where we have the number of 2D holes (Betti-1) vs the increment of the filtration value for the 10 types of networks. Each different colour indicates the Betti-1 curve of a network obtained from a different initial configuration of points. The plot of Betti-1 is consistent with what we expected: when the points are more spread, for example in C_5, we need a greater filtration value to create enough connections to see the rise of Betti-1. For C_1, since the points

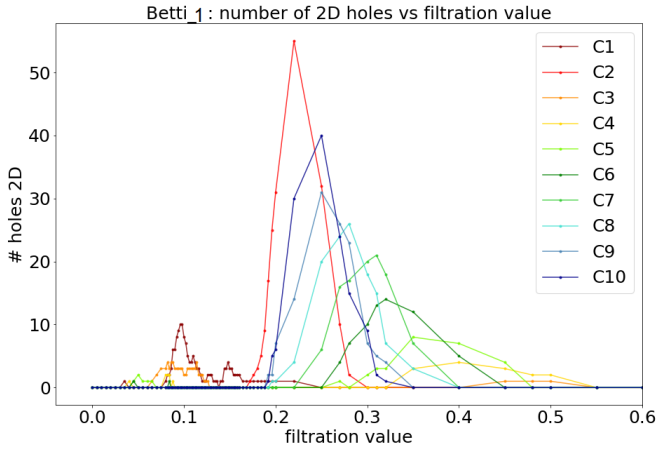


Figure 7.4. Betti-1 curves for the 10 types of configuration of simulated data.

are all spread around the centre, the distance between them is more or less always the same (small distance). With a small filtration value we already have a great number of connections, so many 2D holes can emerge (dark red Fig. 7.4). The filtration values to compute the Betti-1 were chosen to be more distanced when we increased the filtration value. We made this choice because when the number of connections is high, the computation is very slow and so we choose more distanced values of filtration to allow a fast calculation even if the plots look less precise.

In Fig. 7.5 we show the computation of Betti-2. The problem is that for most of

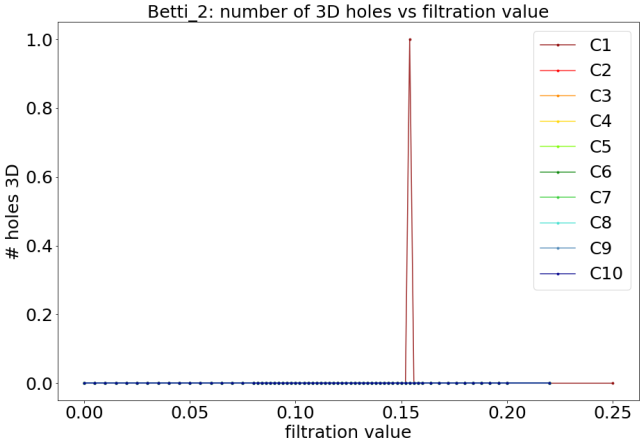


Figure 7.5. Betti-2 curves for the 10 types of configuration of simulated data.

the configurations, to see Betti-2, we have to increase the filtration value in order to take into account more connections and so see 3D holes. For not enough big filtration values, as in Fig. 7.5, no Betti-2 emerge and so the count of Betti-2 is 0. If we increase the filtration, in the attempt of seeing Betti-2, the calculation is

computationally expensive: the program needs to run for hours.
This preliminary work on simulated data was done just to test the correctness of the algorithm so we didn't proceed to further investigations and we moved to the data of the Amsterdam UMC database to reveal the difference between Glioma patients and healthy controls.

7.3 Experimental Data: Glioma patients and HC

In this section, we will apply the explained methods on the database of the Amsterdam UMC composed of 71 Glioma and 53 HC patients. First, we will compute Betti-1 (Section 7.3.1) and then Euler characteristic (Section 7.3.2) on both groups. We will try to reveal the difference between the two groups using these measures.

7.3.1 Betti-1 of Glioma patients and HC

We computed the Betti-1 for all Glioma patients and HC, then we calculated the mean of the curves for each of the two groups. The result is shown in Fig. 7.6.

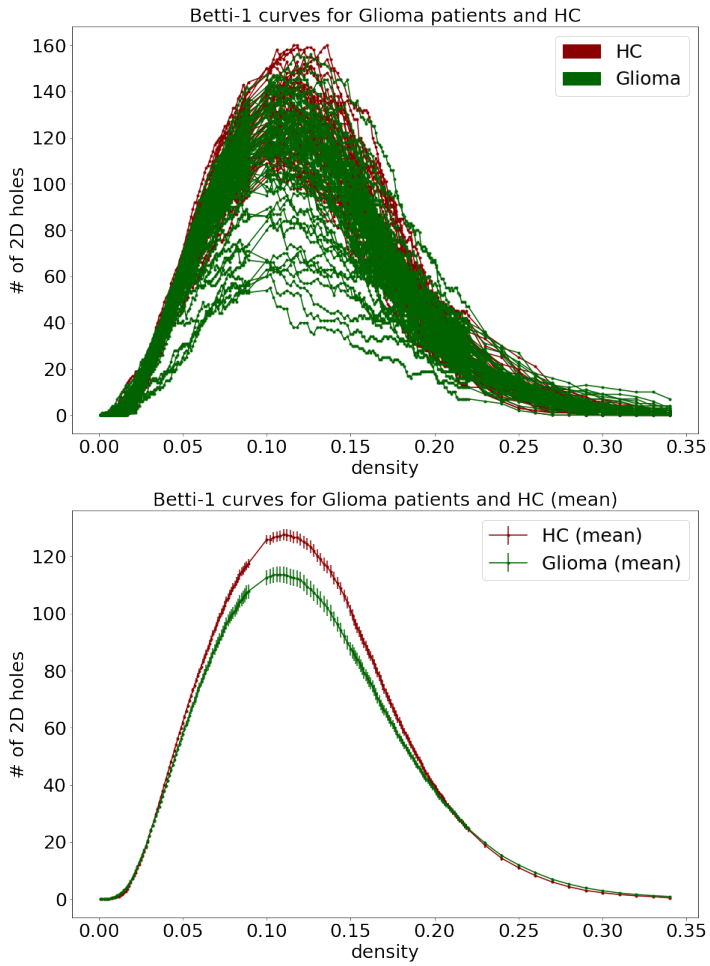


Figure 7.6. Betti-1 curves for 71 HC and 53 Glioma patients and correspondent mean curves (with errors).

We note that the Glioma patients have, as expected, a more significant variability, but overall the maximum number of Betti-1 for the Glioma patients is smaller compared to HC. In the second figure of Fig. 7.6, this is clearly visible: the mean curve of the Glioma patient is lower than the HC.

In x-axis of Fig. 7.6 we have, instead of filtration value (as it was the case of the Betti-0 curves in Section 5.2.2), the density of connections in the network. So, for each chosen density of connections, we compute the number of Betti-1 for that network.

Now we will explain what we do for each chosen density; the Python code is given in Appendix B.2. For example, if we want a density of 10% (10% of the maximum number of connections $N(N-1)/2$) what we do is the following. We start from the connectivity matrix C_X 78x78 containing the values of strength of connection for each pair of nodes. With the rescaling explained in Section 5.2, we obtain a matrix where smaller values mean stronger connections. We create a list with all these values in increasing order (line 12 code B.2), and then we take the first 10% of the list. The 10% of $(78 \times 77)/2$=3003 connections are 300 connections, so the new list contains 300 values. We save the last value of the shorted list (list of the 10% of strongest connections): this will be our threshold (line 19, 20, 26 B.2). We now create a network (represented by a new matrix 78x78) based on that threshold. When the value in the starting matrix is greater than that filtration value (meaning that it is not in the 10% of strongest connections), we put 0 (the two nodes are not connected) when the value is smaller, we put 1 (the two nodes are connected); (line 30, 32 B.2). So now we have obtained a network where we connected only the nodes that had a connection which was in the 10% of strongest connections (line 35 B.2).

On the so obtained network we can compute the desired quantities (in this case, Betti-1). Then we choose a greater value of density, and we create a new network using the same procedure and compute Betti-1 again. We repeat the procedure for all the chosen values of density in the desired range. Of course, from a run time point of view when the value of density is greater, the number of connections is higher, so the computation is slower because the Boundary matrix can be huge. In Fig. 7.6, we showed the obtained Betti-1 curves for Glioma patients and HC. The values of density were chosen to be in the range [0,0.34].

To better analyze the difference between the two groups, we can compute the area under the curve for each patient. We obtain the following result:

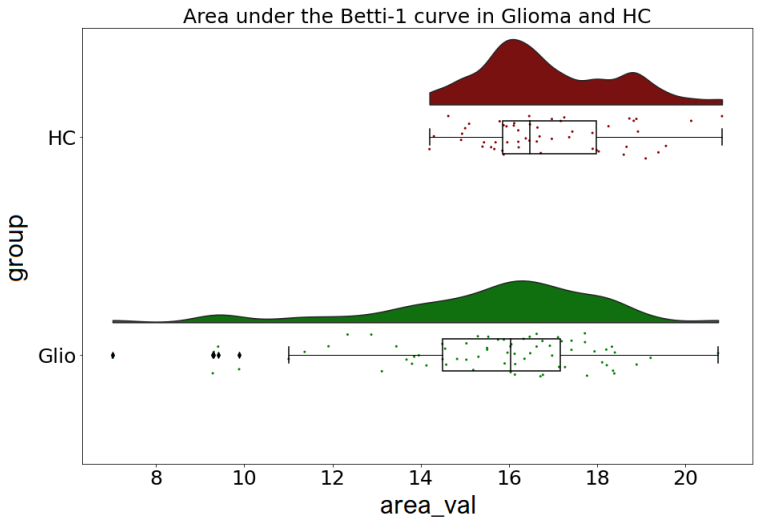


Figure 7.7. Areas under the Betti-1 curves for the two groups (HC and Glioma patients).

Doing the Mann-Whitney test we obtain a p-value=0.0038 so the two distributions are different. The mean and error of the two groups are:

	mean	error
HC	16.87	0.21
Glioma	15.50	0.31

We conclude that the two groups have a different topological structure of the brain regarding the number of 2D holes. The Glioma patients have fewer holes compared to the HC.

7.3.2 Euler characteristic of Glioma patients and HC

In the study *Topological phase transitions in functional brain networks* [3], phase transition of brain networks using the Euler characteristic was explored; the research gave really promising results. We wanted to apply those methods on our Glioma database to verify if we can obtain the same promising results. The data used in their study were fMRI data. In contrast, our database is composed of MEG data. This work is the first to perform this type of investigation on MEG data.
In this section, we will explore the Euler characteristic in the Glioma patients and healthy control.

We computed the Euler characteristic curve for each patient and then calculated the mean curve (with the error). We choose density values up to 60%, and not very close to each other because the computational time otherwise would have been very long. The resultant plot is not very precise but can give a first exploratory insight into the data. The result is shown in Fig. 7.8.

We can notice that there is a difference between the Glioma and the HC at a group level. The area under the HC is greater than the area under the Glioma curve.

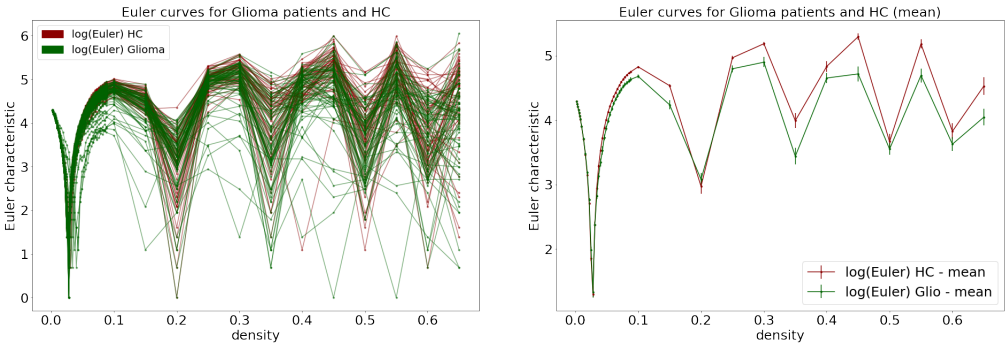


Figure 7.8. Euler characteristic curves for the 53 HC and 71 Glioma patients and correspondent mean curves (with errors). The density is chosen to be in the range [0,0.6].

To better investigate this difference, we chose density values close to each other. We reduce the range of the investigated densities in order to take into account just the first three transitions; the density was chosen to be in the range [0,0.42]. The result is shown in Fig. 7.9.

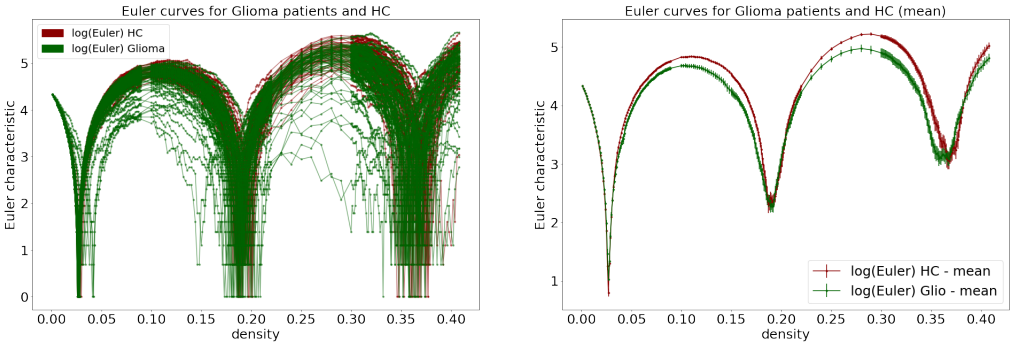


Figure 7.9. Euler characteristic curves for the 53 HC and 71 Glioma patients and correspondent mean curves (with errors). The density is in the range [0,0.42], so only the first three transitions are visible.

Here, as well, we can notice the differences between the two groups of curves. The areas under the HC curves are greater than the areas under the Glioma curves. In the mean curve (right in Fig. 7.9), this can be clearly seen.
We can compute the value of the area under the Euler curve for each Glioma patient and each HC and then compute the mean area under the curve for the two groups. The values of the areas under the Euler curves are shown in Fig. 7.10.

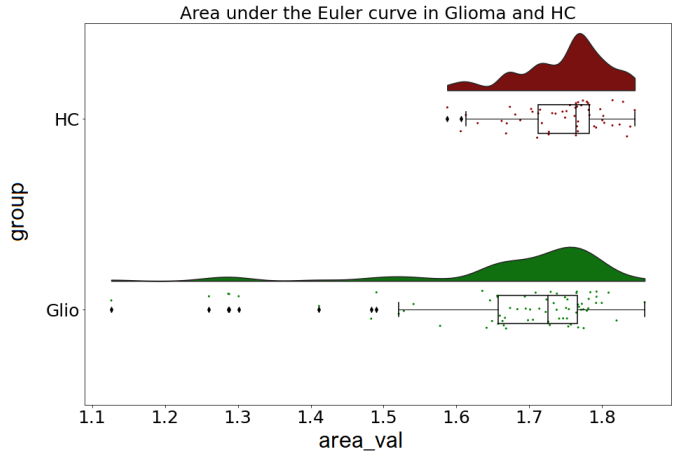


Figure 7.10. Areas under the Euler curves for the two groups (HC and Glioma patients).

The Mann-Whitney test gave a p=0.001, so the two distributions are confirmed to be different also from a statistical point of view. The mean area under the Euler curve (and error[2]) for the two groups are:

	mean area under the Euler curve	error
HC	1.746	0.008
Glio	1.676	0.017

In Fig. 7.9 we can notice that the positions of the first and third transitions are different for HC and Glioma, whereas the position of the second one seems not to differ between the two groups. We computed the position of the three transitions for every patient and used the Mann-Whitney test to see if the distributions of the positions of the transitions are different for the two groups.

To compute the position of the transitions, we did the following: we divided the list with the range of the density into 3 sub-lists, and then we looked in each sublist the correspondent value of the Euler curve for each density value. For each range of density, we computed the minimum value of the Euler. In this way, we obtained the 3 minimum values of the Euler curve corresponding to the three transitions. We repeated this procedure for each patient, and then we computed the mean of the first, second, and third transition for the two groups (HC and Glioma). The mean and the error of the three transitions for the two groups are reported in the following table.

	mean 1st tr.	error	mean 2st tr.	error	mean 3rd tr.	error
HC	0.0271	0.0001	0.0911	0.0007	0.036	0.001
Glio	0.0282	0.0004	0.188	0.001	0.361	0.001

[2]The error associated with the mean value of the area under the curves was calculated as $\frac{\sigma}{\sqrt{N}}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\overline{x})^2}{N-1}}$, N being the number of individuals in the group.

In the following table we report the results for the Mann-Whitney test between HC and Glioma for the three transitions. The Mann-Whitney test says that the first

	1st transition	2st transition	3rd transition
M.-W. test	p=0.013 (reject H_0)	p=0.059 (fail to reject H_0)	p=0.02 (reject H_0)

and third transitions are different for the two groups, whereas the second is not distinguishable for the two groups (as we predicted just looking at the plot).

We want now to compare the Euler curve of the patients with the Euler curve of a random network. We want to explore more in detail if the Euler characteristic curve of the patients is useful to distinguish networks and depends on the network of groups, or if it is just similar to the Euler characteristic of a random network. To do so, we checked the difference of the Euler curve obtained from the original matrix of a Glioma patient and the Euler curve obtained from the shuffled matrix[3] of the same Glioma patient. We report here one graph as an example.

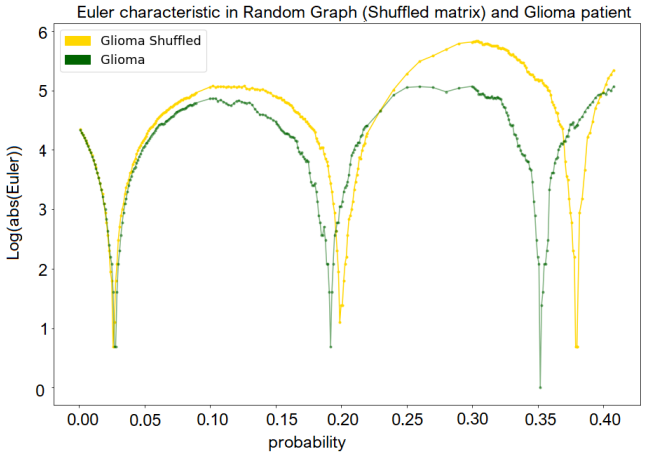


Figure 7.11. Comparison between the Euler curve of the matrix of a Glioma patient and the Euler curve of the shuffled matrix of that same Glioma patient.

In Fig. 7.11 we notice that Euler curve obtained from the original matrix of the Glioma patient is very different from the one obtained from the shuffled matrix, the second and third transitions are shifted to the right, and the maximum of the Euler (and so the area under the curve) is greater.

To analyze the difference of the Euler curve of the patient from the curve obtained on a purely random network (not just the shuffled matrix), we used the Erdos-Renyi graph. We created a random network Erdos-Renyi with 78 nodes with probabilities in a range from 0 to 0.4. We repeated the experiments 60 times and then for each network (and each probability) we computed the Euler characteristic. Then we computed the mean of all the 60 curves to obtain a mean curve (with error).

[3]The shuffled matrix is a new matrix, obtained from the original one, where we changed the order of the values in the matrix in a random way.

We compared these random networks to the Euler characteristic of the Glioma and HC. The result is shown in Fig. 7.12.

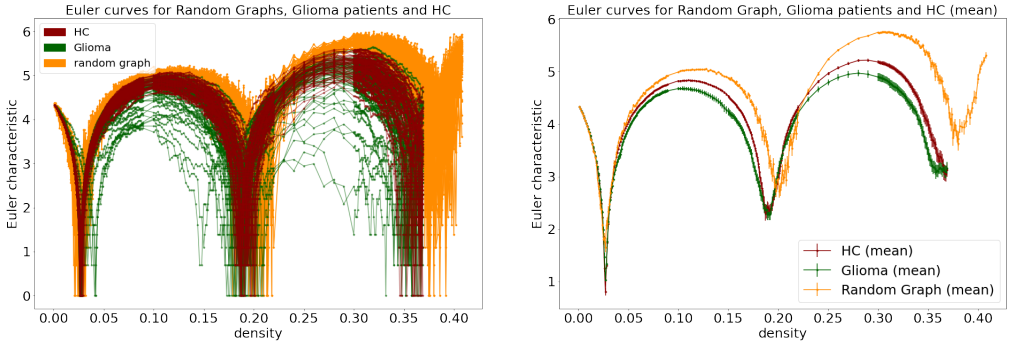


Figure 7.12. Left plot: Euler characteristic curves of 60 random networks (orange), 53 HC (red) and 71 Glioma patients (green). Right plot: mean curve (with errors) for each of the three groups. There is a significant difference between the three mean curves.

The mean random graph curve (in orange in Fig. 7.12) is very different from the other two mean curves (HC and Glioma). The second and third transitions are shifted to the right, and the two maximum of the curves are greater compared to Glioma and HC. This result demonstrates that the brain is not a random network; its characteristics are different. We can use Euler characteristic to investigate the networks and distinguish different groups of patients.

Our results, which show differences in real networks compared to random networks, are coherent with previous researches. In fact, some studies have found that taking a random graph with the same number of nodes of a real network and computing the number of simplicial we obtain that the number of simplicial in the real network is greater than in the random graph. This result means that all these connections are a crucial characteristic of the structure of the brain and not just noise [17].

7.4 Euler characteristic: an approximation of Betti numbers

In this section, we will prove that the Euler curve is an approximation of Betti-curves also for our experimental data.

Since the Euler characteristic is computationally faster to calculate compared to the Betti algorithm we can use it to have information about the Betti numbers. In fact, from the theory, we know that the Euler is an approximation of the Betti curves [13]. We have already proved in Section 7.1 that this is true for random networks. We want to check if this is also the case for our experimental data. We first compared Betti-0, Betti-1, and Euler for one HC patient, and the result is shown in Fig. 7.13.

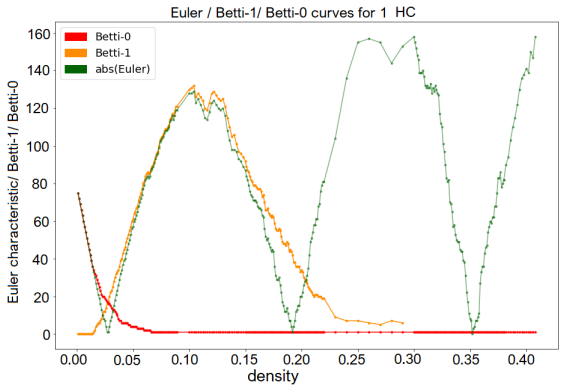


Figure 7.13. Betti-0, Betti-1 and Euler curves for one healthy individual. The Euler curves is a good approximation of the Betti curves.

We notice that the absolute value of the Euler (green in Fig. 7.13) approximate the Betti numbers (red and orange curves). Then we checked if it was the case for all the patients.

We computed the Euler curve, the Betti-0 and Betti-1 curve for each individual. Then for each group we computed the mean of all curves and the associated error, calculated as usual. We did this procedure for Glioma patients and HC, and we compared the Euler to the Betti. We can sum everything in the same graph in Fig. 7.14.

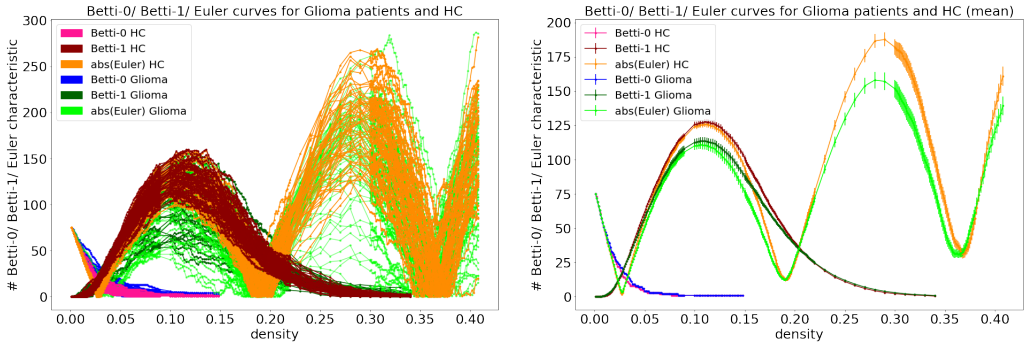


Figure 7.14. Betti-0, Betti-1 and Euler curves for all HC and Glioma patients and correspondent mean curves (with errors).

This result is extremely important because we can understand more about higher order Betti numbers (Betti-3, Betti-4, etc.) using the Euler curve, which is faster to calculate as explained in Section 6.4. We conclude that the Euler characteristic is a good tool to have insight into the brain networks and differentiate patients.

Chapter 8

Trying to relate Betti numbers and Euler characteristic to individual clinical traits

The clinical network neuroscience is the field of research which tries to find the correlation between networks and clinics. This relation has not been found yet in a solid theoretical basis and is still an open question in research. In this last chapter, we will try to relate individual clinical traits of the patients to the studied properties of its network (such as the area under the Betti-0, Betti-1, and Euler curves). Our goal is to find a correlation between individual clinical traits and the structure of the brain network.

The clinical traits that we are looking at are: the age of the patients, the biomarkers (in particular the gene IDH), the group of the tumour (II, III, IV), and the KPS. We will now explain each characteristic and try to correlate it with the networks properties.

The distributions of the age of the individuals of the two groups is shown in Fig. 8.1.

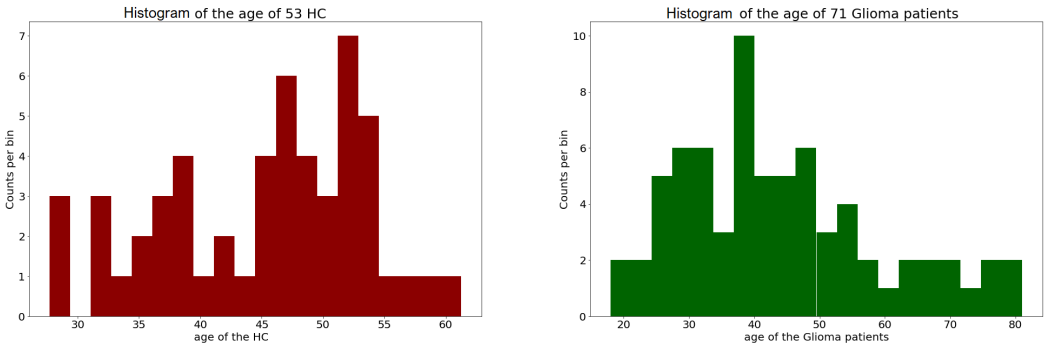


Figure 8.1. Histograms showing the distributions of the age for the two groups.

We now try to relate the age to the area under the Betti-0 curve. In Fig. 8.2, we report a scatter plot where for each patient, we relate his age to his area under the Betti-0 curve. We can't see a correlation between the areas under the curves and

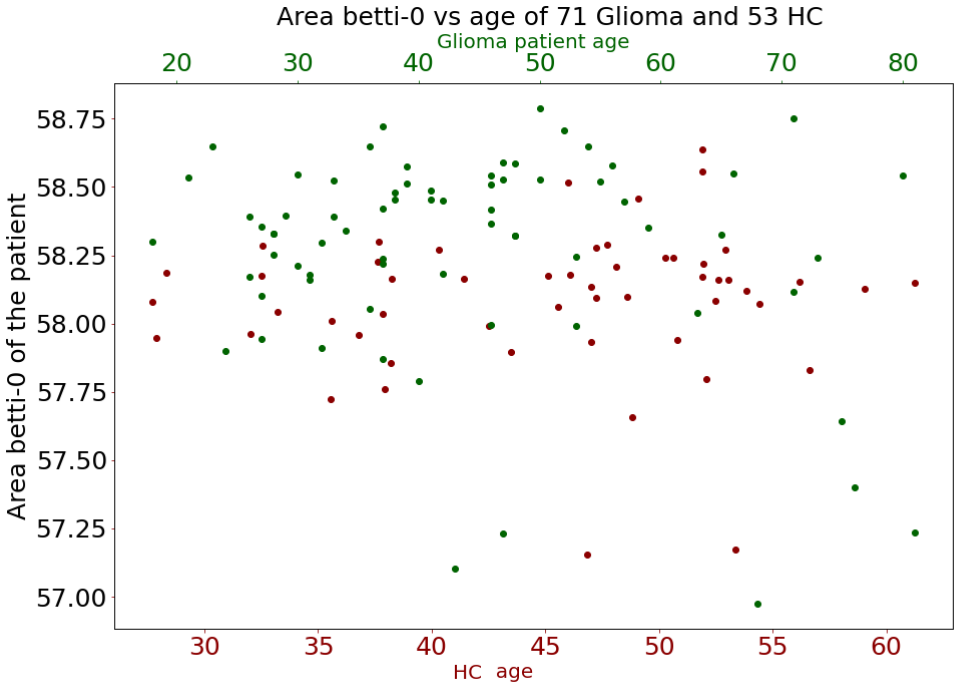


Figure 8.2. Scatter plot between the areas under the Betti-0 curves and the ages of the Glioma patients and HC.

the ages of the patients. This is a good result because we didn't want to find a correlation between the area under the Betti curve and the age. Our goal is to relate the structure of the networks to characteristics of the patients regarding the tumour.

Then we tried to relate the Betti-curves to the grade of the tumour. The Glioma patients in our database were divided into 3 groups based on their grade of the tumour (II, III, IV). We have 34 patients with a tumour of grade II, 15 patients with a tumour of grade III, and 22 patients with a tumour of grade IV. If the grade is bigger, it means that the tumour is more aggressive. We hoped to find some kind of correlations between the area under the Betti-0 curve and the grade of the tumour. For example, something that would be intuitive is that a greater grade of the tumour relates to a bigger area under the curve. Since the area under the curve for the HC is smaller compared to Glioma, a patient with an aggressive tumour should differ even more from HC. The results of the correlation are shown in Fig. 8.3.

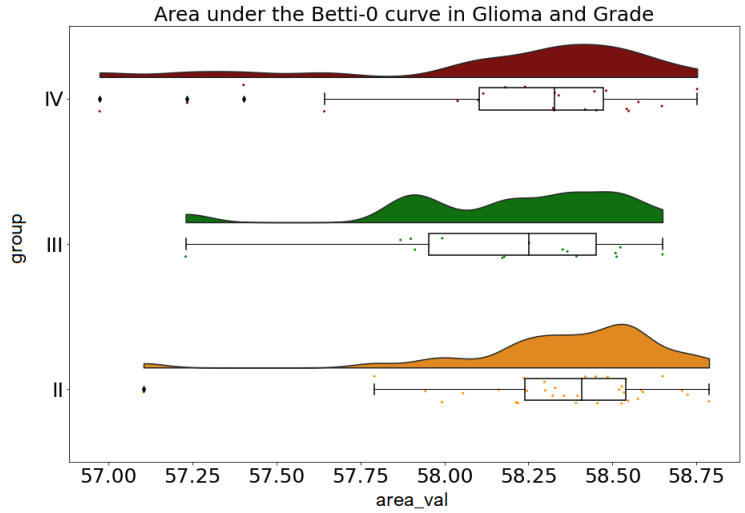


Figure 8.3. Correlation between the grade of the tumour and the area under the Betti-0 curve.

Unfortunately, this is not what we found. Looking at the plot in Fig. 8.3, we can see that the distribution of group II differs slightly from the other two. The Mann-Whitney test confirms that the mean area under the Betti-0 curve of group II differs from group III and IV. The problem is that we have many outlayers, and

Distribution checked	result of Mann-Withney test
II and III:	p=0.038 (reject H_0)
II and IV:	p=0.038 (reject H_0)
III and IV:	p=0.33 (fail to reject H_0)

we don't have enough patients to do a proper statistical analysis. The points are spread in a broad range so we can't interpret this result deducing that patients with a given tumour group have a bigger or smaller area under the curve. To investigate this, we would need a bigger database.

Then we analyzed the KPS. This value usually is in a range between 0 and 100. It is a measure of how much the tumour affects the daily life of the patient. If the value is 100 this means that the patient is able to carry a normal life. If the value is 90 they have minor symptoms, but their daily life is not much affected. 80 means that their quality of life is affected, but they are still quite good. 70 means that they have lost part of their cognition function, and 60 or below means that they need full-time assistance.
In our database, we have 1 patient with KPS=40, 3 patients with KPS=70, 10 patients with KPS=80, 10 patients with KPS=90, and 41 patients with KPS=100. The first two groups have too few patients, and so we can't do statistical analysis on them. Here we have the same problem as the previous correlation: the points are spread, and we can't find a real trend in the data (Fig. 8.4).

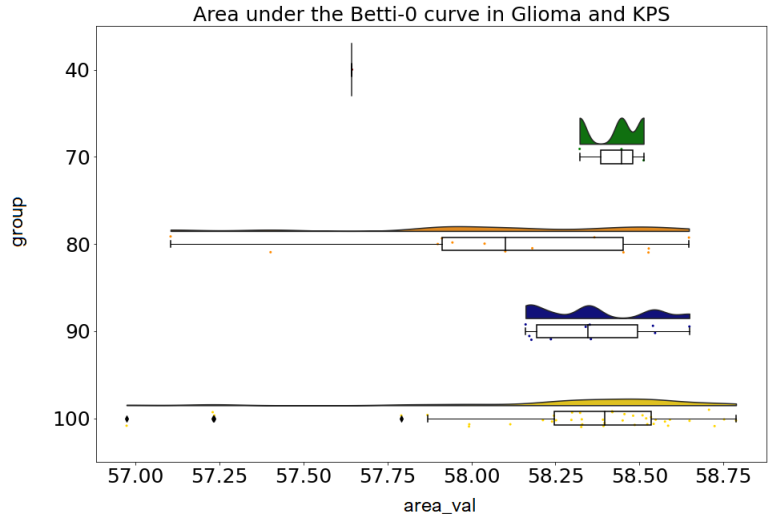


Figure 8.4. Correlation between the KPS and the area under the Betti-0 curve.

Even if the populations are not very big we carried the Mann-Whitney test and we obtained:

Distribution checked	result of Mann-Withney test
KPS 100 and KPS 90:	p=0.33 (fail to reject H_0)
KPS 100 and KPS 80:	p=0.04 (reject H_0)
KPS 90 and KPS 80:	p=0.06 (fail to reject H_0)

It seems that the Mann-Withney test differentiate the patients in group KPS=100 from the patients in group KPS=80, the test has a p value < 0.05 but this is not enough to correlate the data with the area under the Betti-0 curve and say something about their distribution.

Then we tried to correlate the area under the Betti-0 curves to the type of Glioma of the patients. To differentiate the type of Glioma researchers looks at biomarkers, in this case, they distinguished the type of tumour based on the mutation of the IDH gene. This gene can be either not mutated (wild type IDH WT) or mutated. The mutation can be of two types: it can be 1p19q codeleted (meaning that we have a deletion of both the short arm of chromosome 1p and the long arm of chromosome 19q), or it can be non-codeleted. The worst type of Glioma is IDH WT, the second is the IDH mutant+non-codeleted 1p19q, and lastly, IDH mutant+1p19q codeleted, which is the less aggressive type of tumour.
The Glioma patients of our database are divided into these three groups. We have 10 patients in the group IDH mutant+1p19q codeleted, 29 patients in the group IDH mutant+1p19q non-codeleted, 22 patients in the group IDH WT and 10 patients with no available information.
In Fig. 8.5, we show the area under the Betti-0 curves in relation to the type of tumour.

It is reasonable to expect that the worse type of tumour (IDH WT) will result in having a greater area under the curve compared to the other groups because it should differentiate more from the HC (which have a smaller area under the Betti-0 curve).

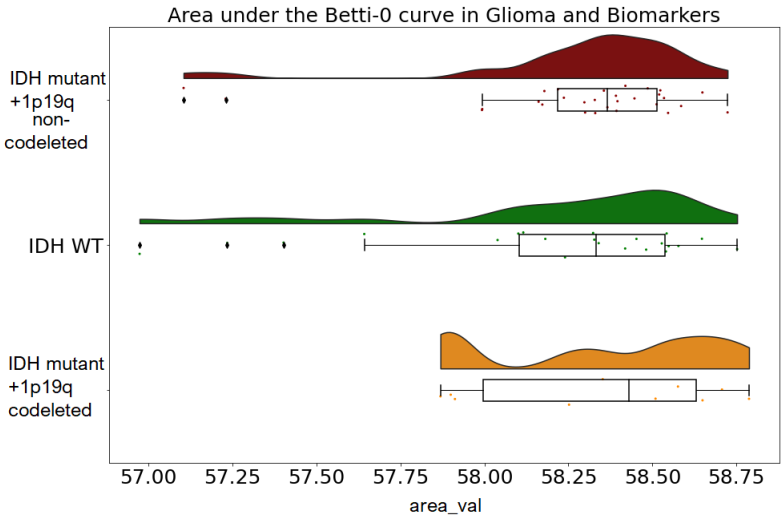


Figure 8.5. Correlation between the mutation of the IDH gene and the area under the Betti-0 curve.

Unfortunately, this is not what we found; the patients of the IDH WT group have an area under the curve spread all over the possible values of areas and don't differentiate from the other groups. As we can see from the plots, the distributions of areas are not significantly different between any of the three groups: we can't see a direct correlation between the group in which the patient is and its area under the Betti-0 curve. As usual, we checked the difference between distributions with the Mann-Whitney test.

Distribution checked	result of Mann-Withney test
cod and non-cod:	p=0.32 (fail to reject H_0)
cod and IDH WT:	p=0.20 (fail to reject H_0)
non-cod and IDH WT:	p=0.43 (fail to reject H_0)

The Mann-Whitney test confirms that the three distributions of the area don't differ in a significant way.

We then tried to relate the area under the Betti-1 curves, and the area under Euler curves to the patients individual clinical traits just described (grade of the tumour, KPS, type of tumour based on biomarkers). We repeated the same analysis described above, and we plot the data using rain cloud plots. We will not report the plots here, but they can be found in the Appendix C.
Unfortunately, also in these cases we weren't able to relate the structure of the network (Betti numbers and Euler characteristic) to the personal data of the patients:

we couldn't find any correlation. One of the reasons may be that the population is too small to do this kind of analysis.

One of the hypotheses of the correlation between network properties and clinical traits was the relation between area under the curve and strength of the tumour. The difference between the mean area under the curve of HC and the area under the curve of a patient should be more significant for patients with a worse type of tumour than for patients with less aggressive tumour. This hypothesis was not confirmed.

Maybe the performed analysis is not powerful enough to detect such differences within the disease group. Our method was not able to correlate the network proprieties to the individual traits of the patients as hoped. The differences we found were only at a group level: it was only possible to distinguish the HC from the Glioma patients. The area under the Betti-0 curves is bigger for Glioma patients compared to HC. The area under the Betti-1 curves and Euler curves was bigger for HC compared to Glioma patients.

We hope that in further studies, the correlation of network properties with clinical traits may be detected. One of the keys may be to use a huge database where statistical analysis can be performed better than what we did here.

In the study *Topological Data Analysis of Functional MRI Connectivity in Time and Space Domains* [18] TDA was used to find a correlation between the cognition of the patients and barcodes. Their database was composed of 1003 patients (almost 10 times bigger than our database). In this research, the authors were able to relate the personal characteristics of the patients (such as cognition scores) to topological changes in the network. TDA was confirmed to be a promising method, and its use in brain networks research can bring in the future to new interesting results.

Conclusions

The goal of this thesis work was to find a metric that could be suitable to compute distances between brain networks and reveal differences in the structure of brain networks of different groups of patients. We first analyzed two distances: the Gromov-Hausdorff and the Gromov-Wasserstein (Chapter 3). To test the robustness of these metrics, we first applied them on simulated networks generated in Python (Chapter 4). The result was promising: the Gromov-Hausdorff distance was able to distinguish different types of matrices. We then proceeded to apply these methods on experimental fMRI connectivity matrices available in the Human Connectome database composed by ADS patients and healthy controls (Chapter 5). The comparison between brain networks of the two groups using the Gromov-Hausdorff distance was not satisfactory from a clinical perspective. We, therefore, analyzed a Topological Data Analysis (TDA) technique using the Betti curves. This analysis revealed to be more powerful, more robust to noise in the data, and so more adequate for our goal of differentiating brain networks of the disease group from brain networks of healthy controls.
From the connectivity matrices of the brain, through the Python algorithm we implemented, it was possible to examine how fast the nodes cluster and study all the brain networks obtained at all possible thresholds. Therefore, we analyzed the Betti-0 curves (representing the number of connected components vs filtration values) in patients with ASD to see if they differed from healthy controls. The results showed that the nodes in the brain of ASD patients cluster slower than the nodes in the brain of the healthy controls. Using Betti-0 curves, we were able to significantly differentiate the individuals of the two groups. After the analysis of the database of the Human Connectome Project (Chapter 5 Section 5.1), we focused on a database of the Amsterdam UMC consisting of Glioma patients and healthy controls based on MEG data (Chapter 5 Section 5.2). With our analysis, we were able to significantly differentiate the two groups: again, the nodes of the brain networks of the disease group cluster slower than in the HC.

TDA (Chapter 6) resulted in being a potent analysis tool, and so in the second part of the work, we continued to use it. We modelled the brain in a more complex way taking into account the simplicial complex underlying the brain structure. Using TDA, we computed high order Betti numbers in the brain networks (Chapter 7). We analyzed Betti-1 curves (number of 2D holes vs filtration value) and Euler characteristic curves to reveal differences in the brain networks of Glioma patients and healthy controls. Also, Betti-1 and Euler were significantly different for the two groups; the area under the Betti-1 curves of the HC was greater than the Glioma

group. Moreover, we showed that the phase transition of the Euler curve was related to the Betti numbers. This method was able to reveal a difference in the brain network structure of the two groups.

Our ultimate goal was then to try to relate the structure of the network to the clinics. We wanted to understand how the computed properties of the network (Betti numbers and Euler characteristic) correlate with the individual clinical traits of the patient, such as the grade of the tumour and other markers (Chapter 8). Unfortunately, we couldn't find any correlation in the data. Nevertheless, TDA is a robust method: it was able to distinguish the group of Glioma patients from the HC group and reveal differences in their brain network properties. We are confident that using TDA in further studies could also enhance differences within the disease group and find relationships between network properties and individual clinical traits.

Appendices

Appendix A

Python code 1

The Python codes used for the first part of the work are given in Appendix A. In Section A.1, is reported the code to generate the simulated data. In Section A.2, we illustrate the code to create D_X matrices from C_X matrices and the computation of distances using GH.

A.1 Generation of simulated networks

In this section, the code to generate random networks is given.

A.1.1 Distribution of points in 10 configurations

This first code is the code to distribute 100 points in the square with 10 different types of configurations.

```
1
2 #Code that generate the position of the points grouped in different
      ways (fig.)
3 npoints=100 # number of points I generate in each square (grouped in
      1,4,9,16,25...100 groups depending on the type of C)
4 positions= [[0 for x_y_coord in range(0,2)] for p in range(npoints)]
      #initialize positions x y of the points
5 L=2 # the points are distributed between 0 and 2
6 n_experiments=20 # for each configuration of sampled points I repeat
      the experiments 20 times
7
8 for exp in range(0,n_experiments):
9
10     for i in range(0,10):
11
12         grouped_points=int(npoints/(i+1)**2) #number of points in
      each group (100/1=100,100/4=25...)
13         remainder = npoints%(i+1)**2 #number of points I have then to
       distribute (ex: 100/16= 6 with reminder: 4)
14
15         f=open('position_xy_var01/position_i=%s_n=%s.txt'%(str(i+1).
      zfill(3),str(exp).zfill(3)),"w+")# I create a file to write the
      positions, 1 file for each configuration (i) and experiment(n_exp)
16         a=0 #Initialise the index of the position matrix
17         count=0 #Initialise the the count of the position matrix
```

```
18
19          for x in range(0,i+1): #2 for loops to find the center of the
        gaussian where we distribute the points (see figure 3)
20              for y in range(0,i+1):
21                  mean_x= (x+1)*L/(i+2) #mean for the x coordinate of
    the gaussian
22                  mean_y= (y+1)*L/(i+2) #mean for the y coordinate of
    the gaussian
23                  b=0 # when the remainder is 0 we place n=
    grouped_points around the founded means (see for loop below)
24                  #print(grouped_points)
25
26                  if count < remainder: # Allows to place the n points
    until you place all the points of the remainder
27                      b=1
28                      count=count+1
29                  for k in range(0,grouped_points+b): # generate the
    points in the founded position of mean_x-mean_y (ex C2: 100 points
    ,25 in each group)
30                                                      # b can be 0 or 1
        depending on if we need to put or not the points of the remainder
31                      #print(grouped_points+b)
32                      positions[a][0]=gauss (mean_x,0.1*L/(i+1)) #
    position x
33                      positions[a][1]=gauss (mean_y,0.1*L/(i+1)) #
    position y
34                      f.write('%f %f \n'%(positions[a][0],positions[a
    ][1])) #Write the positions in the file
35                      a+=1 # Increment a to then write the following
    position when the loop starts again
36          f.close()
37 #print(positions)
```

Listing A.1. Code to distribute 100 points in 10 different configurations.

A.1.2 Generate the C_X matrix

The following code generates the random network starting from the distribution of points and use Euclidian distance to build the connectivity matrix C_X.

```
1  #Code that generate the $C_X$ matrix (euclidean distance between all
       the generated points)
2  for exp in range(0,n_experiments): #I generate a $C_X$ matrix for
       each experiment and each i(different configuration of points)
3       for i in range(0,10):
4
5           positions_data  = pd.read_csv('position_xy_var01/position_i=%
       s_n=%s.txt'%(str(i+1).zfill(3),str(exp).zfill(3)),
       delim_whitespace=True, names= 'x''y')#import the data from the
       files positions
6           x= np.array(positions_data['x']) # Create an array containing
        the values of x and y stored in the file
7           y= np.array(positions_data['y'])
8           cx= [[0 for cx_x in range(0,len(x))] for cx_y in range(0,len(
       x))] #Create a matrix npoints x npoints to store the cx values
9           f=open('Cx_matrix_var01/cx_i=%s_n=%s.txt'%(str(i+1).zfill(3),
       str(exp).zfill(3)),"w+") # Open a file to store the values of the
```

```
    matrix (one file for each type of configuration i and experiment n
    )
10
11          for i in range(0,len(cx)): # Compute the distance between all
     the points
12              for j in range(i,len(cx)):
13                  cx[i][j]= m.sqrt((x[i]-x[j])**2+(y[i]-y[j])**2) #
    Compute the distance between two points
14                  cx[j][i]=cx[i][j] #the matrix is simmetric
15
16          for i in range(0,len(x)): # write the values of the Cx matrix
     in the correspondant file
17              for j in range(0,len(x)):
18                  f.write('%f '%(cx[i][j]))
19              f.write('\n')
20          f.close()
21 #Plot all the Cx matrix for the different configuration of points (
    just for the first experiment of each configuration n=000)
22 for i in range(0,10):
23     cx=np.array(np.loadtxt('Cx_matrix_var01/cx_i=%s_n=000.txt'%(str(i
    +1).zfill(3))) ) # upload the Cx file in an array
24
25     fig = plt.figure(figsize=(15, 10)) #create the figure
26     Cx_plot = sns.heatmap(cx,square=True,cmap='hot',xticklabels=10,
    yticklabels=10) #plot cx using heatmap
27     Cx_plot.set(xlim=(0,100),ylim=(100,0)) #set the x,y range
28     plt.title('Cx i=%s_exp=0'%(str(i+1)),fontsize=18) #set title
29     fig.savefig('Cx_figures_var01/Cx i=%s_exp=0.png'%(str(i+1))) #
    save the figures in the folder
```

Listing A.2. Code to generate the C_X matrices.

A.2 Algorithm from C_X to D_X matrix

The following code is the core of the first part of the thesis and allows us to build the D_X matrices starting from the connectivity matrix C_X. The important steps of this algorithm is explained in Chapter 3, Section 3.2, here more detailed comments are given.

```
1
2 #Function to convert a matrix Cx to a Dx matrix (new definition of
    distance between points, not euclidean)
3 def from_cx_to_dx(matrix):
4
5     a_values=[] # initialize the array of the filtration values
6     connected_comp=[] #initialise the array that will contain the
    number of connected components (from 100 different components to 1
     giant component)
7     list_clusters = [[i] for i in range(0,len(matrix)) ] #initialize
    the array which contains the lists of clusters(connected
    components).
8     #In each list I have the points which are part of the same
    cluster.
9     #At first I have 100 connected components (100 separetad lists),
    at the end I will have 1 big list containg all the points (1 giant
     component)
```

```
10
11      connected_comp.append(len(list_clusters)) # I append the number
        of different cluster I have at the beginnig
12      #print(list_clusters)
13      a=0 #order of the cluster
14      a_values.append(a) # I append the first filtration value to the
        list of filtration values
15
16      while (find_max(matrix) >= a ): #until all the numbers in the
        matrix are substituted with the order of the
17
18          #print("a:",a)
19          #print("I look for the "position" of a")
20          array_positions = find_pos_a(a,matrix) # I look for the
        positions where I have a connection of order a(see function below)
21          #print("array_position:",array_positions)
22
23          for position in array_positions :
24
25              index_x= find_index(list_clusters,position[0])# In the
        list list_clusters(which contain the lists of clusters) I look for
         the index of the array which contain that point(position[0])
26              #print("index_x:",index_x)
27
28              index_y=find_index(list_clusters,position[1]) # In the
        list list_clusters(which contain the lists of clusters) I look for
         the index of the array which contain that point(position[1])
29              #print("index_y:",index_y)
30
31              if index_y != index_x: #Beacause if so they already are
        in the same cluster, so I don't need to merge the two clusters
32
33              #Since position[0] and position[1] are connected with
        connection of order a (I looked for it with find_pos) I then
        looked in which cluster they belong to and now I can connect all
        the points in the two clusters with the same order of connection (
        definition of dx)
34                  for row in list_clusters[index_x]: # I go in the
        cluster where the point of position[0] is contained and for each
        point there I create a connection with the points in the cluster
        where position[1] is contained (I set matrix[row][colum]=a)
35                      for column in list_clusters[index_y]:
36
37                          matrix[row][column]=a
38                          matrix[column][row]=a
39                          #print("matrix[%d,%d]=%d"%(row,column,a))
40                          #print("matrix[%d,%d]=%d"%(column,row,a))
41
42                  #print(matrix)
43                  #print("update array cluster")
44                  list_clusters[index_x].extend(list_clusters[index_y])
        #Now I am merging the two clusters
45                  list_clusters.remove(list_clusters[index_y]) # I can
        remove the old cluster
46                  #print("list_clusters updated",list_clusters)
47
```

```
                connected_comp.append(len(list_clusters)) # I append
    the number of connected components I now have
                a_values.append(a) #I append the filtration value I
    was using
            a= find_min(a,matrix) # I update the filtration value and I
    start again the loop


    #print(a_values)
    #print(connected_comp)
    #print(matrix)
    return matrix,a_values,connected_comp

#Function to find the maximum value of the matrix
def find_max(matrix):
    c=0
    for i in range(len(matrix)):
        for j in range(i,len(matrix)):
            if matrix[i][j] > c:
                c= matrix[i][j]
    return c


#Function which save the position where I have a connenction ofthe
    order of a
def find_pos_a(a,matrix):
    positions=[] #Initialize the list of positions

    for i in range(len(matrix)): #I look for the connection of order
    a (I can check just half of the matrix since it is simmetric)
        for j in range(i+1,len(matrix)):

            if matrix[i][j]==a: # if the connection betwwen i and j
    is equal to a
                pos = [i,j] #I save the two points ("position in the
    matrix")
                positions.append(pos) # I append the position to the
    array positions
    return positions


# I look for the index of the list (in the lists of arrays) which
    contain a particular value
def find_index(list_arrays,value):
    for i in range(len(list_arrays)):
        for j in range(len(list_arrays[i])):
            if list_arrays[i][j]==value:
                return i

# I find the new filtration value, I look for the minimum value in
    the matrix but greater than the previous a
def find_min(a,matrix):
    b=100
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if matrix[i][j] < b and matrix[i][j] > a :
                b= matrix[i][j]
```

```
95
96      return b
```

Listing A.3. Algorithm to obtain D_X matrices starting from C_x matrices.

A.2.1 Create the D_X matrix

This code recalls the function above and creates the D_X matrices.

```
1  #Code to create a big object with all the Cx matices
2  txt_files_cx = glob.glob('Cx_matrix_var01/cx*.txt') #opening all the
       Cx of the 10 types (first experiment n=000 for each) which are in
       the folder "Cx_matrix"
3  print(txt_files_cx)
4  Cx_list = [] #Object that will contain all the Cx matrices
5
6
7  for i in range(0,len(txt_files_cx)): #Reading the data in the files
       and create an object which contains every matrix
8      file_i= txt_files_cx[i]
9      Cx_list.append(np.array(np.loadtxt(file_i)))   #Add the matrix of
        a file to the object Cx_list
10
11 Cx=np.array(Cx_list)  #Converting the list with all the matrix in a
       big numpy array
12 #print(type(Cx))
13
14
15 #Code to create the Dx matrices from the Cx matrices and the lists of
        filtration values and connected components for each matrix
16 for k in range(0,len(Cx)):
17
18     dx,a,conn=from_cx_to_dx(Cx[k]) # I call the function to create
       the dx matrix from the cx matrix
19     #print(a)
20     #print(conn)
21     print("Matrix dx %d created"%k)
22
23     g=txt_files_cx[k] # g is the file cx
24     g=g.replace('Cx_matrix_var01\\cx','Dx_matrix_var01\\dx')  #I
       replace the name cx with dx in the file name (and I change folder)
25     f=open(g,'w+') #I open the file in the new folder and I save the
       dx matrix
26
27     for i in range(0,len(dx)):
28         for j in range(0,len(dx)):
29             f.write('%f ' %dx[i][j]) # I write the matrix dx on the
       file
30         f.write('\n')
31     f.close()
32
33     # I can comment this part if I don't want to create the barcode
34     #TO SAVE BETTI
35     l=txt_files_cx[k]
36     l=l.replace('cx','betty')
37     l=l.replace('Cx_matrix_var01\\','Beta_var01\\')
38     f=open(l,'w+')
```

```
39
40
41      #h='Beta_var03_10/beta_zero_%d.txt'%(k+1) # In the folder "Beta"
        I create the file to save for each matrix Dx the list of
        filtration values  and correspondant number of connected
        components
42      #f=open(h,'w+')
43
44      for i in range(len(a)):
45          f.write('%f %f' %(a[i],conn[i])) # I write in the file the
        two lists
46          f.write('\n')
47      f.close()
```

Listing A.4. Code to recall the previous function and obtain the D_X matrices.

A.3 Create the GH matrix

This code compares all the D_x matrices using the Gromov-Hausdorff distance and stores the values in the GH matrix, which therefore contains all the comparisons between any two matrices. The Gromov-Hausdorff distance is explained in Chapter 3, Section 3.3.1 and applied in Section 4.1.

```
1  #Code to create a big object with all the Dx matices to then create
       GH
2  txt_files_dx = glob.glob('Dx_matrix_var01/dx*') #opening all the
       files I in the folder Dx
3  print(txt_files_dx)
4
5  dx_matrices_list = [] #dx_matrices_list is the list that will contain
        all matrix dx uploaded
6
7  for i in range(0,len(txt_files_dx)): #Reading the data in the file
       and create an object which contains all the matrices Dx
8      dx_matrices_list.append(np.array(np.loadtxt(txt_files_dx[i])))
9
10 Dx=np.array(dx_matrices_list) #Converting the list with all the
       matrix in a big numpy array
11
12
13 #Code to create the GH matrix where all the comparison between the Dx
        matrices are stored
14 Tot_GH= [[0 for GH_x in range(0,len(Dx))] for GH_y in range(0,len(Dx)
       )] #create the GH matrix filled with 0
15
16 for i in range(0,len(Dx)): #For the matrices in Dx I am comparing all
        the pairs
17     for j in range(i,len(Dx)):
18         difference=[] # I store here all the differences between two
       points of the two matrices
19
20         for k in range(0,len(Dx[i])): # for all the pair of values of
        the two matrices I compute the difference and store the value in
       list "difference"
21             for h in range(k,len(Dx[i])):
22                 difference.append(abs(Dx[i][k,h]-Dx[j][k,h]))
```

```
23
24          #max_diff= max(difference)        # find the maximum between
    the differences of all the pairs of the two matrixes
25          max_diff=np.mean(difference) #I can try with this!
26          Tot_GH[i][j]=max_diff       #The value 1/2max_diff is stored
    in the GH matrix
27          Tot_GH[j][i]= Tot_GH[i][j]        #the matrix is symmetric
28
29
30 f=open('GH_var01_mean','w+') #create the file to write the values of
    th GH matrix
31 for i in range(0,len(Dx)):
32     for j in range(0,len(Dx)):
33          f.write('%f ' %Tot_GH[i][j]) # write the value in the i,j
    position of the matrix
34     f.write('\n')
35 f.close()
36 #Plot of the GH matrix
37 GH=np.array(np.loadtxt('GH_var01_mean') ) # load the GH matrix in an
    array to then plot it
38 fig = plt.figure(figsize=(15, 10)) #create the figure
39 GH_plot = sns.heatmap(GH,square=True,cmap='hot',xticklabels=10,
    yticklabels=10) # plot GH with heatmap
```

Listing A.5. Code to compare the D_X matrices using the GH distance.

Appendix B

Python code 2

In this second part of the Appendices (Appendix B), we give the main code which uses TDA tools to compute Betti numbers and Euler characteristic (Section B.1). In Section B.2, we report the code used to create a network where we choose only a given percentage of the total connections.

B.1 Betti numbers and Euler algorithms

The following code illustrates the function used to compute Betti numbers for a given network. The general structure of the algorithm is explained in Chapter 6, Section 6.4 . Here more detailed comments are given.

```
#Function to compute the required Betti number
#To call the function insert the Network G and the required Betti K_input
#B_0 is the number of connected components
#B_1 is the number of holes
#B_2 is the number of holes 2 tetraedrs one under the other
#B_3 is the complex structure (see image)
def rango(d):
    return np.linalg.matrix_rank(d)

def Betti_k(G,K_input,verbose=True):
    # G is a network graph
    # C is networkx.find_cliques(G)

    def DIAGNOSTIC(*params): # If verbose is True it will print all the DIAGNOSTIC
        if verbose:
           print(*params)

    DIAGNOSTIC("Nodes in G: ", G.nodes())
    DIAGNOSTIC("Edges in G: ", G.edges())
    print("Number of nodes: {}, edges: {}".format(G.number_of_nodes(), G.number_of_edges()))

    # 1. Prepare maximal cliques

    #compute maximal cliques
```

```
25      C =nx.find_cliques(G) # C now is the operator "find clique" (to
        do the list I should do list(nx.find_cliques(G)) )
26
27      #Create list C with all the cliques
28      #Sort each clique, convert it from list to tuple
29      C = [tuple(sorted(c)) for c in C]
30      DIAGNOSTIC("List with all maximal simplex C:",C)
31      DIAGNOSTIC("Number of maximal cliques: %i"%(len(C)))
32
33      # 2. Enumerate all simplices
34
35      S = [] #List of dictionaries
36      # S[k] is the dictionary which contain all k-simplices
37      #S[k].keys() are simplex s (s is one of the k-simplex of the
        dictionary S[k])
38      # S[k].values() are the ID of simplex s
39      print("I start the loop where I create the required Sk to then
        compute betti. Sk is a list with the k-simplex")
40      #I set the range for the following loop
41      if K_input==0:
42          ini=0
43          fin=2
44      else:
45          ini=K_input-1
46          fin=K_input+2
47
48      for k in range(ini,fin) : # k has 2 values for betti_0 and 3
        values for betti1_2_3
49
50          Sk = sorted(set(c for mc in C for c in itertools.combinations
        (mc, k+1)))#This is the same as:
51          #Sk=[]
52          #for mc in C:
53              #for c in itertools.combinations(mc, k+1): #Sk.append(c)
54
55          DIAGNOSTIC("list of %i-simplex S%i:"%(k,k), Sk)
56
57          # Assign an ID to each simplex, in order
58          S.append(dict(zip(Sk, range(0, len(Sk))))) # zip(Sk,range())
        is an object (composed by tuples) where each element of Sk is
        associated to a number.
59          # I then from the zip object create the dictionary where the
        key is the Sk element and the value the number
60           #I put this dictionary in the S list (list of dictionary)
61          print("Number of %i-simplices: "%(k),len(Sk))
62      DIAGNOSTIC("S dictionary",S)
63      #The cliques are redundant now
64      del C
65
66      # 3. Construct the boundary operator
67
68      #Boundary Matrix
69      D =[None, None] #List with the two different k-boundary operators
70
71      if K_input==0:
72          # D[0] is the zero matrix
```

```
73          D[0]=(np.zeros((1, G.number_of_nodes())))#I create a matrix
    of size (1,#nodes)
74
75      for k in range(1, len(S)):
76
77          #I set the index of D[] and the number of nodes in each group
     for the combinatory part
78          if K_input==0:
79              index=k
80              b=k
81          else:
82              index=k-1
83              b=k+(K_input-1)
84
85          D[index] = np.zeros( (len(S[k-1]), len(S[k])) ) #I create a
    matrix of size (len(S[k-1]), len(S[k])
86
87          for (ks, j) in S[k].items() :
88
89              a=sorted(itertools.combinations(ks, b))
90              #DIAGNOSTIC("a",a)
91              # Indices of all (k-1)-subsimplices s of the k-simplex ks
92              I = [S[k-1][s] for s in sorted(itertools.combinations(ks,
     b))] #S is a list of dictionary with k different size with the di
93              #DIAGNOSTIC("I",I)
94
95              for i in range(0,len(I)):
96                  D[index][I[i]][j] = (-1)**i
97
98          if D[index].shape[1]==0:
99              DIAGNOSTIC("I can't create matrix D_ because I don't have
     the needed k-simplex")
100
101         #DIAGNOSTIC("D",D[index])
102
103     print("D_{} has shape {}".format(K_input, D[0].shape))
104     print("D_{} has shape {}".format(K_input+1, D[1].shape))
105     # The simplices are redundant now
106     del S
107
108     # 4. Compute rank and dimker of the boundary operators
109
110     # dim(Im)=Rank and dim(ker)=V-rank
111     rank = [0 if d.shape[1]==0 else rango(d) for d in D] #dim(Im)
112     ker = [(d.shape[1] - rank[n]) for (n, d) in enumerate(D)] #V -
    rank = dim(ker) ,rank=dim(Im)
113
114     #The boundary operators are redundant now
115     del D
116     DIAGNOSTIC("ker:", ker)
117     DIAGNOSTIC("rank:", rank)
118
119     # 5. Compute the Betti number
120
121     # Betti number
122     B=ker[0]-rank[1]
123     print("End of computation\nBetti %i is:"%K_input,B)
```

```
124     return B
```

Listing B.1. Code of the function to compute Betti numbers.

The following code is the code of the function of Euler to compute the Euler characteristic of a given network. The first part of the code is identical to the Betti function code, so we omit it.

```
#Function to compute the Euler Caracteristic
def euler(G, verbose = True):
    #SAME CODE AS BEFORE
    # 1. Prepare maximal cliques : Same code as Betti function
    # 2. Enumerate all simplices : Same code as Betti function

    # 3.Euler characteristic
    ec = sum(((-1)**k * len(S[k])) for k in range(0, len(S))) #
    Alternate sum of all the quantity of simplex of different
    dimension we have
    #len(S[k]) is how many k-simplex we have
     # I sum for every k we have, len(S) is the maximum k we can find
     (dimension of the simplex)
    DIAGNOSTIC("Euler characteristic:", ec)
    return ec
```

Listing B.2. Code with the function to compute the Euler characteristic of a given network.

B.2 Density

The following code selects the connections which are part of a given density value and creates a new network based on the chosen threshold. The general idea of the algorithm is explained in Chapter 7.3, Section 7.3.1, here we give more detailed comments.

```

#DEFINE THE DENSITY THRESHOLD
def dens_thresh(Set,den,i,verbose=False):
    #set contains the networks of different individuals Cx_list, d is
     the density, i individual

    def DIAGNOSTIC(*params): # If verbose is True it will print all
    the DIAGNOSTIC
        if verbose:
            print(*params)

    DIAGNOSTIC("Density choosen: %s. I will create a network where I
    keep just %s perc of the connections"%(str(den).zfill(3),str(den
    *100).zfill(3)))

    all_links_list=sorted(list((Set[i]).ravel()),reverse=True) #
    create a list with all the strenght of connection in decreasing
    order

    DIAGNOSTIC("all_links_list:",all_links_list)
    DIAGNOSTIC("All_link_lists has len:",len(all_links_list))

    #size=len(Set[0,:,:]) # I can put here 78
```

```
    size=25
    cutoff=int(np.ceil(den*size*(size-1))) #np.ceil return float type
     upper which is the closest, I convert in integer
    DIAGNOSTIC("I want a percentage of %s of total links so cutoff d*
    N(N-1)/2:"%str(den).zfill(3),cutoff)

    threshold=all_links_list[-(cutoff+78)] # I look for the value
    correspondent to the index "cutoff" (I count in the list from the
    end, and I ignore the first 78 values because they are all 0)

    DIAGNOSTIC("The threshold is:",threshold)
    if verbose:
        newlist_links = [link for link in all_links_list if link >
    threshold] #List of the wanted percentage of total connections
        #print("newlist_links:",newlist_links)
        print("len(newlist_links):",len(newlist_links))

    C_thresh=np.matrix(1*(np.copy(Set[i])<threshold)) #I create a
    matrix where 1 if value>thresh otherwise 0
    #DIAGNOSTIC(C_thresh)
    G = nx.from_numpy_matrix(np.matrix(C_thresh)) #I create the
    network from the matrix
    #print("G Number of nodes: {}, edges: {}".format(G.
    number_of_nodes(), G.number_of_edges()))

    return G, threshold
```

Listing B.3. Code to create a network with a given percentage of the total connections.

```
#Example of calling the function for the HC patients.

txt_files_cx = sorted(glob.glob('mean_matrix_HC/*')) #opening all the
     Cx of matric of the HC.
print(txt_files_cx)
Cx_list = [] #Object that will contain all the Cx matrices

for i in range(0,len(txt_files_cx)): #Reading the data in the files
    and create an object which contains every matrix
    file_i= txt_files_cx[i]
    Cx_list.append(np.array(np.loadtxt(file_i)))   #Add the matrix of
     a file to the object Cx_list
Cx=np.array(Cx_list) #len 53 in the case of HC

for i in range(0,53):

    f=open('greater_betti/betti0/betti_0densityHC_C%i.txt'%(i+1),"w+"
    )#I create one file for each betti_0 (one for each Cx)

    density00=list(np.arange(0.001, 0.09, 0.001))
    density1=list(np.arange(0.1, 0.15, 0.002))
    density=density00+density1  # I choose the densities I want

    for d in density: #For each density I create a network calling
    the function dens_thresh

        print("Creation new network based on the density %s"%str(d).
    zfill(3))
        G1,filtr=dens_thresh(Cx,d,i,verbose=False) #I call the
```

```
    function to create the different networks
24          k_in=0 #I want betti_0
25          B=Betti_k(G1,k_in, verbose=False) #I call the betti function
    to compute Betti_0 for that matrix
26          f.write('%f %f \n'%(d,B)) #I write betti and filtration value
     in a file
27
28      f.close()
```

Listing B.4. Code where we recall the previous explained function to create networks with given density.

Appendix C

Additional figures

Here we report the additional figures of Chapter 8. We tried to relate the individual clinical traits of the patients to the Euler curves and Betti-1 curves. No correations were found. The Figures are stored in this link: https://www.dropbox.com/sh/u3vzstq7xh422t0/AAD9CVapr7y0IrHCW8qUUeAKa?dl=0.

Bibliography

[1] Andrew Zalesky, Alex Fornito, and Edward T. Bullmore, *Network-based statistic: Identifying differences in brain networks*, Elsevier, 2010. https://www.sciencedirect.com/science/article/abs/pii/S1053811910008852.

[2] Hyekyoung Lee, Hyejin Kang, Moo K. Chung, Bung-Nyun Kim, and Dong Soo Lee, *Persistent Brain Network Homology From the Perspective of Dendrogram*, IEEE, 2012. https://www.researchgate.net/publication/231176286_Persistent_Brain_Network_Homology_From_the_Perspective_of_Dendrogram.

[3] Fernando A. N. Santos, Ernesto P. Raposo, Maurício D. Coutinho-Filho, Mauro Copelli, Cornelis J. Stam, and Linda Douw, *Topological phase transitions in functional brain netwo.rks*, Physical Review, 2019. https://journals.aps.org/pre/abstract/10.1103/PhysRevE.100.032414.

[4] Eduarda Centeno, Amsterdam UMC, *Literature Survey*, Master in Neuroscience, 2020, not yet published.

[5] A. Fornito, A. Zalesky, and E. T. Bullmore, *Fundamentals of Brain Network Analysis*, Academic Press, London, 2016. https://www.sciencedirect.com/book/9780124079083/fundamentals-of-brain-network-analysis.

[6] Adam McLaughlin and David A. Bader, *Scalable and High Performance Betweenness Centrality on the GPU*, International Conference for High Performance Computing, Networking, Storage and Analysis, 2015. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.728.2926&rep=rep1&type=pdf.

[7] Louis-David Lord, Angus B. Stevner, Gustavo Deco, and Morten L. Kringelbach, *Understanding principles of integration and segregation using whole-brain computational connectomics: implications for neuropsychiatric disorders*, The Royal Society, 2017. https://royalsocietypublishing.org/doi/10.1098/rsta.2016.0283.

[8] M.-M. Deza and E. Deza, *Dictionary of distances*, Elsevier, 2006. https://www.sciencedirect.com/book/9780444520876/dictionary-of-distances.

[9] F. Memoli, *Metric structures on datasets: stability and classification of algorithms. In Computer Analysis of Images and Patterns*, pages 1-33, Springer, 2011. https://link.springer.com/chapter/10.1007/978-3-642-23678-5_1.

[10] F. Memoli, *Gromov–Wasserstein Distances and the Metric Approach to Object Matching*, Foundations of Computational Mathematics, 2011. https://link.springer.com/article/10.1007/s10208-011-9093-5.

[11] Jolanda Derksa, Shanna D. Kulika, Tianne Numana, Philip C.de Witt Hamer, David P. Noskeb, Martin Kleinb, Jeroen J.G. Geurtsa, Jaap C. Reijneveldb, Cornelis J. Stamc, Menno M. Schoonheima, Arjan Hillebrandc, Linda Douw, *Understanding global brain network alterations in glioma patients*, manuscript 2020.

[12] Jason Brownlee, *How to Calculate Nonparametric Statistical Hypothesis Tests in Python*, 2018. https://machinelearningmastery.com/nonparametric-statistical-significance-tests-in-python/.

[13] Chad Giusti, Eva Pastalkova, Carina Curto, and Vladimir Itskov, *Clique topology reveals intrinsic geometric structure in neural correlations*, Princeton University, Princeton, NJ, 2015. www.pnas.org/cgi/doi/10.1073/pnas.1506407112.

[14] Alexander P. Kartun-Giles, Ginestra Bianconi, *Beyond the clustering coefficient: A topological analysis of node neighbourhoods in complex networks*, Elsevier, 2019. https://arxiv.org/abs/1901.10978.

[15] Jeremy Kun, *Homology theory*, 2013. https://jeremykun.com/2013/04/03/homology-theory-a-primer/.

[16] Qichao Que, *Lecture 6: Computation of Simplicial Homology: Matrix view Topics in Computational Topology: An Algorithmic View Scribed*. http://web.cse.ohio-state.edu/~wang.1016/courses/788/Lecs/lec7-qichao.pdf.

[17] Kelsey Houston-Edwards, *Simplicial Complexes - Your Brain as Math*, 2017. https://www.youtube.com/watch?v=akgU8nRNIp0&ab_channel=PBSInfiniteSeries.

[18] Keri L. Anderson, Jeffrey S. Anderson, Sourabh Palande, and Bei Wang, *Topological Data Analysis of Functional MRI Connectivity in Time and Space Domains*, Springer Nature Switzerland, 2018. http://www.sci.utah.edu/~beiwang/publications/CNI_TimeDomain_BeiWang_2018.pdf.

[19] Kanad Mandke, Jil Meier, Matthew J. Brookes, Reuben D. O'Dea, Piet Van Mieghem, Cornelis J. Stam, Arjan Hillebrand, Prejaas Tewarie, *Comparing multilayer brain networks between groups: Introducing graph metrics and recommendations*, Elsevier, 2018. https://www.sciencedirect.com/science/article/abs/pii/S1053811917309230.

[20] Zeus Gracia-Tabuenca, Juan Carlos Díaz-Patiño, Isaac Arelio, and Sarael Alcauter, *Topological Data Analysis reveals robust alterations in the whole-brain and frontal lobe functional connectomes in Attention-Deficit/Hyperactivity Disorder*, eNeuro, 2020.https://www.eneuro.org/content/7/3/ENEURO.0543-19.2020.

[21] Albert-László Barabási, *Network science*, Cambridge University Press, 2016. http://networksciencebook.com/.