POLITECNICO
MILANO 1863

Computation of persistent homology on metric graphs

APSC's Project

MSc Mathematical Engineering

A.Y 2021-22

Author: Davide Gurrieri

Professors: Luca Formaggia, Pasquale Claudio Africa

Supervisor: Paolo Zunino

Contents

1 Introduction

The first chapter introduces the notions needed to understand the context of this project: Algebraic Topology. Only few essential topics will be covered. For further in-depth analysis, refer to [4] and [5].

1.1 Homology

Homology is a fundamental concept in algebraic topology that provides a way to study the topological properties of spaces. Often, one is interested in understanding the basic characteristics of a metric space, such as the number of connected components or the presence of holes and voids, rather than its precise geometry. Homology captures these basic characteristics associating them algebraic structures. In particular, given a space X, homology associates to it one vector space $H_i(X)$ for each $i \in \mathbb{N}$. The dimension of each $H_i(X)$ counts the number of i-dimensional holes in X (i.e. connected components for $i = 0$, holes for $i = 1$, voids for $i = 2$). These numbers are called "Betti numbers".

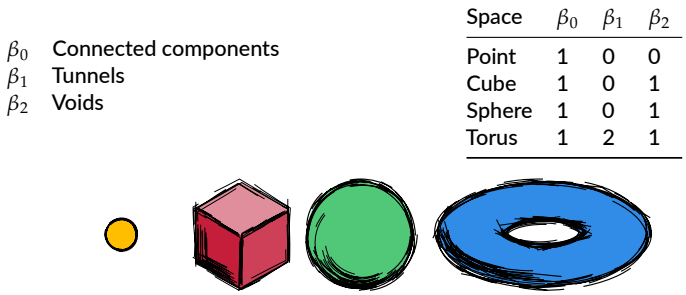

Figure 1: Betti numbers of hollow shapes.

A significant advantage of homology is that it is homotopy invariant, meaning it does not change when the space is subjected to continuous transformations like bending, stretching, or other deformations.
Computing homology for arbitrary topological spaces can be challenging, so often the spaces are approximated by combinatorial structures called *simplicial complexes*, for which homology can be easily computed algorithmically.

1.2 Simplicial complexes

In the Topological Data Analysis (TDA) framework, one of the most popular approach involves transforming the data into a *simplicial complex* and then computing its topological invariants (Betti numbers) using homology theory.

Definition 1.1. Given a set K_0, a *simplicial complex* is a collection K of non-empty subsets of K_0 such that:

- $v \in K \quad \forall\, v \in K_0$
- $\tau \subset \sigma,\ \sigma \in K \implies \tau \in K$

The elements of K_0 are called *vertices* of K, and the elements of K are called *simplices*. Additionally, a *simplex* has *dimension* p or is a p-simplex if it has cardinality $p+1$. If τ and σ are simplices such that $\tau \subset \sigma$, then τ is called a *face* of σ. The *dimension* of K is defined as the maximum of the dimensions of its simplices. K_p is used to denote the collection of p-simplices. The *k-skeleton* of K is the union of the sets K_p for $p = 0, 1, \ldots, k$.

The given definition is rather abstract but it is always possible to think of a simplicial complex as a geometric object. Each simplex of the simplicial complex corresponds to a geometric simplex of a certain dimension, which can be interpreted as a point (dimension 0), a segment (dimension 1), a triangle (dimension 2), a tetrahedron (dimension 3), or a higher-dimensional generalization of these shapes. The simplices are glued together along their faces, which represent the common boundaries between them. In this way a simplicial complex can approximate the topology of a more complex space. For example the following simplicial complexes can be geometrically interpreted as in Figure 2:

$$K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{c,d\}, \{a,b,c\}\}$$

$$K' = \{\{g\}, \{h\}, \{l\}, \{g,h\}, \{g,l\}, \{h,l\}, \{g,h,l\}\}$$

Figure 2: Geometric realizations of two simplicial complexes.

Therefore, it is always possible to consider any graph as a 1-dimensional simplicial complex or to construct a simplicial complex from a points cloud.

There are many well-known examples of simplicial complexes constructed from a set of points in $\mathbb{R}^d$. Some of these are Vietoris Rips, Čech, Witness, Delaunay, and Alpha complexes. However, the Rips and Čech complexes are often too large to handle in practice. Only the Delaunay and Alpha complexes are discussed here, since they will be used in the implementation.

Definition 1.2. Given a finite set of points $P \in \mathbb{R}^d$, a k-simplex σ is *Delaunay* if its vertices are in P and there is an open d-ball whose boundary contains its vertices and is empty (contains no point in P). Note that any number of points in P can lie on the boundary of this ball. But, for simplicity, it is assumed that only the vertices of σ are on the boundary of its empty ball. A *Delaunay complex* of P, denoted $Del(P)$, is a simplicial complex with vertices in P in which every simplex is *Delaunay* and the union of its simplices in its geometric realization coincides with the convex hull of P.

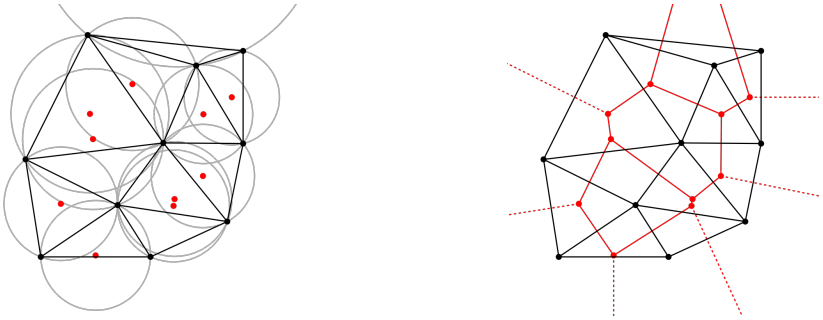

Figure 3: Delaunay triangulation with all the circumcircles and their centers (left). Connecting the centers of the circumcircles produces the Voronoi diagram (right).

Delaunay complexes are dual to the Voronoi diagrams as shown in Figure 3. This duality enables efficient conversion between the two structures, expanding their applicability in various geometric and spatial analyses. Moreover, the Delaunay complex is conceptually simple, as it's based on a local criterion (circumcircles) and doesn't involve complex global computations. This simplicity contributes to its efficiency and ease of implementation. Moreover, one of the advantages is that the number of simplices is lower compared to other simplicial complexes: given n points, Delaunay complexes in $\mathbb{R}^2$ have size that grows as $O(n)$ and in $\mathbb{R}^3$ as $O(n^2)$.

An *Alpha complexes* is a subcomplex of the Delaunay complex, parameterized by a real parameter $\alpha \geq 0$.

Definition 1.3. For a given set of points P and $\alpha \geq 0$, an *Alpha complex* consists of all simplices in $Del(P)$ that have a circumscribing ball of radius at most α.

It can also be described alternatively as follows. For each point $p \in P$, let $B(p, \alpha)$ denote a closed ball of radius α centered in p. Consider the closed set D_p^α defined as follows:

$$D_p^\alpha = \{x \in B(p, \alpha) \mid \mathrm{d}(x, p) \leq \mathrm{d}(x, q)\ \forall q \in P\}$$

The Alpha complex $Del^\alpha(P)$ is the *nerve* of the closed sets $\left\{D_p^\alpha\right\}_{p \in P}$.

Definition 1.4. Let $\mathcal{U} = \{U_i\}_{i \in I}$ be a non-empty collection of sets. The *nerve* of $\mathcal{U}$ is the simplicial complex with set of vertices given by I and k-simplices given by $\{i_0, \ldots, i_k\}$ if and only if $U_{i_0} \cap \cdots \cap U_{i_k} \neq \varnothing$.

An example of Alpha complex is provided in Figure 4.

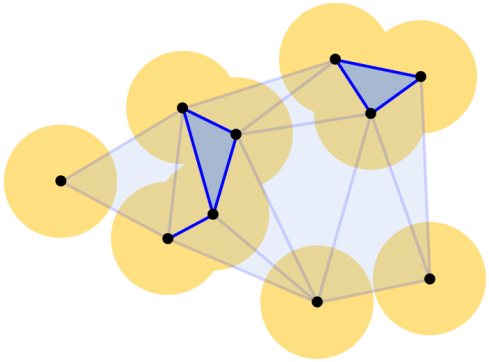

Figure 4: Geometric realization of a Delaunay complex (opaque) and of the correspondent Alpha complex for a fixed value α (vivid). Balls $B(p, \alpha)$ around each point are represented in yellow.

1.3 Homology of simplicial complexes

The next step is to characterize the shape of a simplicial complex by leveraging homology.

Definition 1.5. Given a simplicial complex K, let $C_p(K)$ denote the $\mathbb{F}_2$-vector space with basis composed by the p-simplices of K. Coefficients are in $\mathbb{F}_2$, hence all elements of $C_p(K)$ are formal sums of the form $\sum_j \sigma_j$, for $\sigma_j \in K_p$. Elements of $C_p(K)$ are called *simplicial chains*.

For example, given $K = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$, two possible 1-chains are $\{a, c\}$ and $\{a, b\} + \{a, c\}$. Note that the sum of the preceding two elements is $\{a, b\}$ since the coefficients are in $\mathbb{F}_2$ and $\{a, c\} + \{a, c\} = 0$

Definition 1.6. Given a simplicial complex K and $p \in \{1, 2, \dots\}$, the p^{th} *boundary homomorphism* is a function $d_p : C_p \to C_{p-1}$ that assigns each p-simplex $\sigma = \{v_0, \dots, v_p\} \in K$ to its boundary: $d_p(\sigma) = \sum_i \{v_0, \dots, \widehat{v_i}, \dots, v_p\}$. d_0 is defined as the zero map.

In the equation above, $\widehat{v_i}$ indicates that the set does not contain the i^{th} vertex. It is possible to prove that the function d_p is actually an homomorphism.

Since the boundary of a boundary is always empty, the linear maps d_p have the property that composing any two consecutive maps yields the zero map: $d_p \circ d_{p+1} = 0 \forall p \in \{0, 1, 2, \dots\}$.

Considering the previously cited simplicial complex K, you have:

$d_2(\{a, b, c\}) = \{a, b\} + \{b, c\} + \{a, c\}$ and

$d_1(\{a, b\} + \{b, c\} + \{a, c\}) = \{a\} + \{b\} + \{b\} + \{c\} + \{a\} + \{c\} = 0$ so

$d_1(d_2(\{a, b, c\})) = 0.$

Consequently, the image of d_{p+1} is contained in the kernel of d_p, so the quotient of kernel(d_p) by image(d_{p+1}) is well defined. $Z_p :=$ kernel(d_p) is called *Cycle group* and its elements *p-cycles*. $B_p :=$ image(d_{p+1}) is called *Boundary group* and its elements *p-boundaries*.

The following definition can thus be made.

Definition 1.7. For any $p \in \{0, 1, 2, \ldots\}$, the p^{th} *homology* of a simplicial complex K is the quotient vector space

$$H_p(K) := Z_p / B_p$$

Its dimension

$$\beta_p(K) := \dim(H_p(K)) = \operatorname{rank}(Z_p) - \operatorname{rank}(B_p)$$

is called the p^{th} *Betti number* of K.

Intuitively, the p-cycles that are not boundaries represent p-dimensional holes. Therefore, the p^{th} Betti number "counts" the number of p-holes. Additionally, if K is a simplicial complex of dimension n, then for all $p > n$ $H_p(K) = 0$, as K_p is empty and hence $C_p(K) = 0$. The following sequence of vector spaces and linear maps is therefore obtained.

$$0 \xrightarrow{d_{n+1}} C_n(K) \xrightarrow{d_n} \cdots \xrightarrow{d_2} C_1(K) \xrightarrow{d_1} C_0(K) \xrightarrow{d_0} 0.$$

In order to compute homology, it is convenient to represent each non-trivial boundary homomorphism as a matrix that records the incidences between simplices. Using standard row and column operations it is possible to extract the ranks of Cycle and Boundary groups and compute Betti numbers using the definition.

Let K be a simplicial complex. Its p-th boundary matrix represents the $(p-1)$-simplices as rows and the p-simplices as columns. Assuming an ordering on the simplices of the same dimension, this matrix is $d_p = \left[a_i^j\right]$, where $a_i^j = 1$ if and only if the i-th $(p-1)$-simplex is a face of the j-th p-simplex. Given a vector c_p of dimension n_p (number of p-simplices) representing a p-chain, the boundary can be computed by matrix multiplication:

$$d_p c_p = \begin{bmatrix} a_1^1 & a_1^2 & \ldots & a_1^{n_p} \\ a_2^1 & a_2^2 & \ldots & a_2^{n_p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_{p-1}}^1 & a_{n_{p-1}}^2 & \ldots & a_{n_{p-1}}^{n_p} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n_p} \end{bmatrix}$$

The p^{th} boundary matrix can be reduced to *Smith normal form*, using row and column operations as those of Gaussian elimination for solving linear

systems. This means that in the reduced matrix there is an initial segment of the diagonal composed by 1's and everything else is 0, as in Figure 5.

Once the boundary matrices are in Smith normal form, the information needed to calculate the Betti numbers can be directly obtained:

- rank(Z_p) is the number of zero columns of the boundary matrix of d_p.
- rank(B_p) is the number of non-zero rows of the boundary matrix of d_{p+1}.

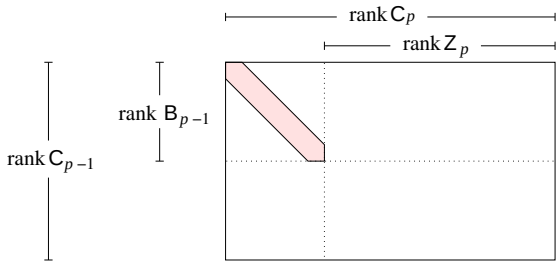

Figure 5: Smith normal form of the boundary matrix d_p.

Note that this result holds thanks to deep algebraic topology theorems that are omitted.

To fix the idea, consider the 1-skeleton of a tetrahedron, as shown in Figure 6. It is a simplicial complex of dimension 1 and the only non-trivial boundary matrix is d_1. Row and column operations are used to reduce d_1 to Smith normal form. The computation of $rank(Z_0)$ and $rank(B_1)$ is straightforward. $Z_0 = \text{rank}(d_0) = span(\{a\}, \{b\}, \{c\}, \{d\})$, since each of these simplices is mapped to 0. Therefore, $\text{rank}(Z_0) = 4$. Moreover there are no 2-simplices in K, so $B_1 = \text{image}(d_2) = \{0\}$.

The values of $rank(Z_1)$ and $rank(B_0)$ can instead be read from the Smith normal form of d_1. In conclusion:

- $\beta_0 = \text{rank}(Z_0) - \text{rank}(B_0) = 4 - 3 = 1$
- $\beta_1 = \text{rank}(Z_1) - \text{rank}(B_1) = 3 - 0 = 3$

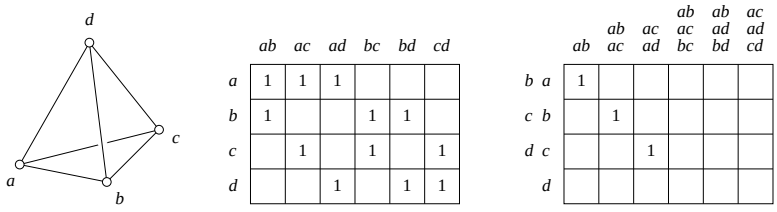

Figure 6: From left to right: the edge skeleton of the tetrahedron; its first boundary matrix; the reduced boundary matrix with $\text{rank}(B_0) = 3$ non-zero rows and $\text{rank}(Z_1) = 3$ zero columns.

1.4 Persistent Homology

While homology gives information about a single simplicial complex, Persistent Homology (PH) allows to study topological features across sequences, so-called *filtrations*, of simplicial complexes.

Definition 1.8. Given K a simplicial complex, a *filtration* of K is a sequence of embedded simplicial complexes,

$$\varnothing = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n = K$$

starting with the empty complex and ending with the entire simplicial complex K.

For example, filtrations of the Delaunay complex using Alpha complexes with increasing parameter α can be built. The first non empty simplicial complex in the filtration consists only of points. Increasing α, more and more simplices are added until the Delaunay complex is obtained. Some simplicial complexes of an alpha filtration are shown in Figure 7.

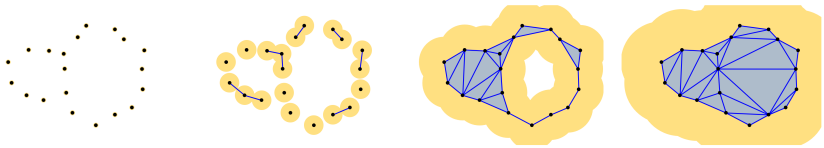

Figure 7: Four simplicial complex of an Alpha filtration.

The simplicial complexes in a filtration are connected by inclusion maps. An important property of homology, called *functoriality*, can now be applied:

any map between simplicial complexes $f_{i,j} : K_i \to K_j$ induces a map between their p-chains spaces $\tilde{f}^p_{i,j} : C_p(K_i) \to C_p(K_j)$ which induces a map between their homology $f^p_{i,j} : H_p(K_i) \to H_p(K_j)$. In particular, this means that there exist maps between the homology groups of every simplicial complex in a filtration, induced by the inclusion maps of the nested simplicial complexes.

$$0 = H_p(K_0) \to H_p(K_1) \to \ldots \to H_p(K_n) = H_p(K)$$

In other words, there are maps that relate voids, loops or connected components (depending on the dimension p) in simplicial complexes across a filtration.

Topological features such as loops or connected components across a filtration can be visualized in a summary diagram called *barcode*. It represents the information carried by the homology groups and the maps $f^p_{i,j} : H_p(K_i) \to H_p(K_j)$. A topological feature of dimension p in $H_p(K_b)$ is *born* in $H_p(K_b)$, if it is not in the image of $f^p_{b-1,b}$. For example, a loop is born in filtration step b, if the loop appears closed in the simplicial complex K_b for the first time. A topological feature from $H_p(K_i)$ *dies* in $H_p(K_d)$, where $i < d$, if d is the smallest index such that the feature is mapped to zero by $f^p_{i,d}$. If the topological feature is a loop, intuitively it dies in the filtration step where it is first fully covered by triangles (or other higher-dimensional simplices). Note that some topological features never die in a filtration. For example, in a non-empty simplicial complex there is always one connected component that is never mapped to zero.
Given a non decreasing function w : $\mathbb{Z} \to \mathbb{R}$ that assigns a weight to each simplicial complex of the filtration, topological features in the filtration of a simplicial complex are represented by half-open intervals $[w(b), w(d))$. A barcode is made up of these intervals. In the case of an Alpha filtration, w is the function that associates each simplicial complex to the correspondent value α. The lifetime of a topological feature, the so-called *persistence*, is defined as $w(d) - w(b)$. For topological features that persist until the last filtration step (and beyond), the persistence is said to be infinite.

In Figure 8 is reported another example of a filtration as well as the persistence barcode of its topological features.

To compute the persistent homology of a filtered simplicial complex K and obtain a barcode like the one illustrated in Figure 8, it is necessary to associate to it a *global boundary matrix* that stores information about the faces of

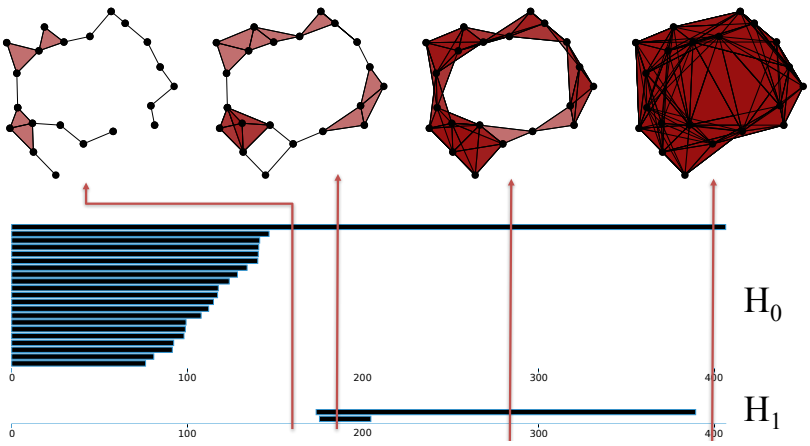

Figure 8: Some simplicial complex of a Vietoris-Rips filtration and the correspondent persistence barcode.

every simplex. To do this, a total ordering on the simplices of the complex is needed. This ordering must be compatible with the filtration in the following sense:

- A face of a simplex precedes the simplex;
- A simplex in the i^{th} complex K_i precedes simplices in K_j for $j > i$, which are not in K_i.

Let m denote the total number of simplices in the complex, and let $\sigma_1, \ldots, \sigma_m$ denote the simplices with respect to this ordering. The boundary matrix B has dimension $m \times m$ and stores a 1 in position (i, j) if the simplex σ_i is a face of simplex σ_j of codimension 1; otherwise, it stores a 0. Define $low(j)$ to be the largest index value i such that $B(i, j)$ is different from 0. If column j only contains 0 entries, then the value of $low(j)$ is undefined. Once the boundary matrix is constructed, it must be reduced using Gaussian elimination following, for example, the standard algorithm reported below. In the worst case, the complexity of the standard algorithm is cubic in the number of simplices.

for j = 1 to m **do**
  while there exists i $<$ j with low(i) = low(j) **do**
    add column i to column j
  end while
end for

The intervals of the barcode can be read from the reduced global boundary matrix in the following way:

- If $\text{low}(j) = i$, then the simplex σ_j is paired with σ_i, and the entrance of σ_i in the filtration causes the birth of a feature that dies with the entrance of σ_j.
- If $\text{low}(j)$ is undefined, then the entrance of the simplex σ_j in the filtration causes the birth of a feature. If there exists k such that $\text{low}(k) = j$, then σ_j is paired with the simplex σ_k, whose entrance in the filtration causes the death of the feature. If no such k exists, then σ_j is unpaired.

A pair (σ_i, σ_j) gives the half-open interval $[\text{dg}(\sigma_i), \text{dg}(\sigma_j))$ in the barcode, where for a simplex $\sigma \in K$ $\text{dg}(\sigma)$ is defined to be the smallest number l such that $\sigma \in K_l$. An unpaired simplex σ_k gives the infinite interval $[\text{dg}(\sigma_k), \infty)$. Finally, a weight function w can be applied to the extremes of the intervals.

Another way to visualize persistence information is through *persistence diagrams*. In this case, the pairs $(\text{birth}, \text{death})$ are represented as points on the Cartesian plane.

In Figure 9, taken from Paper [6], the entire process applied to a specific example is detailed.

(a) A filtered simplicial complex:

(b) We put a total order on the simplices that is compatible with the filtration:

where σ_i denotes the ith simplex in this order.

(c) (Left) The boundary matrix B for the filtered simplicial complex in (a) with respect to order on simplices in (b), and (right) its reduction $\overline{B}$ given by applying Algorithm 1 (one first adds column 5 to column 6, and then column 4 to column 6):

$$B = \begin{pmatrix} 0&0&0&1&1&0&0\\ 0&0&0&1&0&1&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0 \end{pmatrix} \quad \overline{B} = \begin{pmatrix} 0&0&0&1&1&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0 \end{pmatrix}.$$

(d) We read off the following intervals from the matrix $\overline{B}$ in (c):

- σ_1 is positive, unpaired; this gives the interval $[1, \infty)$ in H_0.
- σ_2 is positive, paired with σ_4; this gives no interval, because σ_2 and σ_4 enter at the same time in the filtration.
- σ_3 is positive, paired with σ_5: this gives the interval $[2, 3)$ in H_0.
- σ_6 is positive, paired with σ_7: this gives the interval $[3, 4)$ in H_1.

Figure 9: Persistent barcode computation.

2 Applications to vascular networks

This project aims to apply the techniques explained in the introductory paragraph to metric graphs, specifically focusing on vascular networks. Applying persistent homology to a vascular network can provide valuable insights into its structural and topological properties and can offer several interesting advantages.

2.1 Reasons for using Persistent Homology

Vascular networks can be highly complex and challenging to characterize using simple metrics. Persistent homology provides a quantitative measure of the network's complexity by identifying the number of persistent features (e.g., loops, voids) and their lifetimes. This can help to differentiate between healthy and pathological vascular structures because diseases like cancer, cardiovascular disorders, and neurological conditions can significantly alter vessels structures. Persistent homology can assist in detecting and characterizing these changes, potentially serving as a diagnostic tool.

Moreover, vascular data, especially those obtained from imaging techniques, can be noisy and imperfect. Persistent homology is capable of handling noisy data, providing a more robust analysis compared to traditional methods that might be sensitive to small variations.

This project particularly focuses on vascular networks analysis for tumor detection. As a tumor grows, it eventually requires its own blood supply. It achieves this by using the existing vessels within the body. These vessels are then prompted to develop new capillaries that infiltrate the tumor, forming an intricate network of blood vessels specific for the tumor. This phenomenon is known as angiogenesis and it is a significant hallmark of cancer. It plays a critical role in invasive behavior, allowing the tumor to breach neighboring tissues and to expand beyond a certain size. This step is of paramount importance, enabling the tumor to assume a malignant state. Functionally, these vessels carry out pivotal tasks for the tumor. They not only provide essential nutrients but they also offer structural support, contributing to the physical integrity of the tumor. Moreover, they participate in a range of additional functions, including immune suppression in the tumor's vicinity.

Structurally, vessels spawned from this process exhibit marked irregularities. A salient feature is the formation of voids, regions within the network where

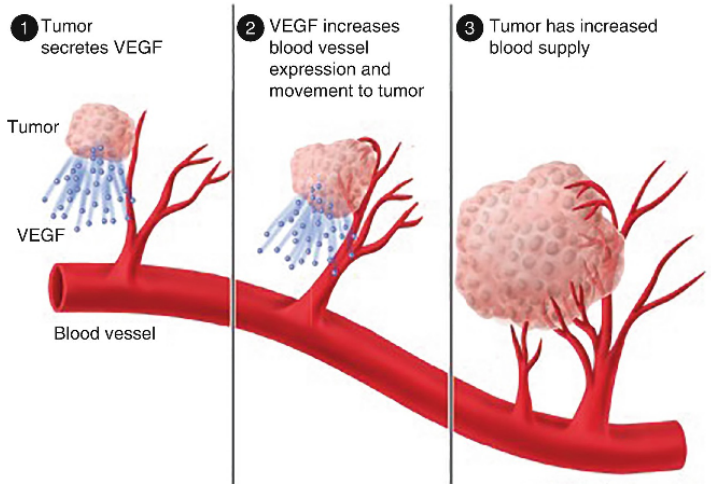

Figure 10: Process of tumor angiogenesis through the secretion of Vascular Endothelial Growth Factor (VEGF)

vessel density diminishes. This is due to the tumor's rapid growth outpacing the capacity of angiogenesis to establish a comprehensive network. Consequently, these voids can impede optimal blood supply to certain areas of the tumor. In addition to voids, the network demonstrates variances in vessel diameter, branching patterns, and the formation of loops.

The use of persistent homology is crucial for identifying these anomalies and for quantifying the irregularity of vascular networks.

2.2 Available data

To apply persistent homology techniques, it's necessary to have a graph that describes the vascular network of interest. This graph can be obtained through various imaging procedures, as detailed in Paper [7]. In this project, it's assumed that the graphical representation of vascular networks is already available. The entire process required to obtain it is not covered.

The research group led by Professor Zunino has provided a useful MatLab tool that allows the creation of synthetic 2-dimensional vascular networks. These networks are constructed using Voronoi tessellations from a set of points in a given domain, resulting in segments that form a 2-dimensional graph. By distributing points uniformly in the domain, applying Voronoi tessellations, and subsequent transformations, the tool produces network structures resembling real vascular systems. Indeed, this construction allows

equidistant spacing between vessel vertices and generating points, a crucial feature for accurate microcirculation representation.

It is also possible to generate anomalous networks that include areas where circulation doesn't reach. This is achieved by employing a Poisson process that selects vertices to be removed. This methodology generates networks with desired level of disorder, controllable by a parameter that assumes values ranging from 0 to 1. The network characterized by the value 0 shows no anomalies, while increasing the parameter simulates increasingly severe pathology. Corrective measures are applied to balance disorder and density, aiming for accurate and controlled vascular representations. In Figure 11 some of these vascular networks are shown.

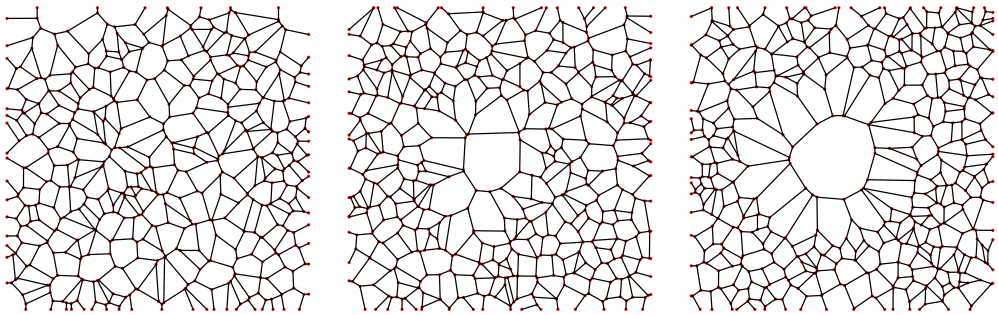

Figure 11: Synthetic vascular networks generated with pathology parameter 0, 0.1 and 0.2 (from left to right).

These artificial vascular networks are intended to test algebraic topology methods on samples that are small in size and easy to use. In this way, it is immediate to verify the effectiveness of these methods and direct the choice toward the most appropriate tools.

Then, it is possible to apply the selected strategies to real vascular networks in three dimensions, like the freely available one described in Paper [8]. The authors grew tumors on female mice that were 8/10 weeks old and they obtained vascular networks like the one shown in Figure 12.

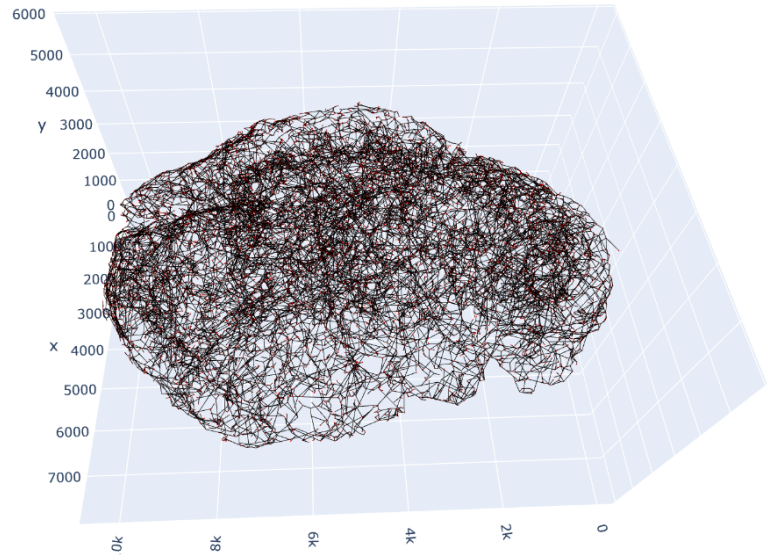

Figure 12: Vascular network of a mouse's brain where a tumor has grown.

2.3 Methods

To characterize the geometric properties of the vascular networks at disposal, it was firstly decided to consider the graph representing the networks as a simplicial complex of dimension 1. Recall that in real networks, blood vessels assume disordered behavior near tumors and tend to create loops and chaotic patterns. Starting from the network simplicial complex, it is possible to calculate Betti numbers and obtain information regarding the number of connected components and loops of the graph. Moreover, in order to discover further insights, it was decided to use the *Radial filtration* introduced in Paper [7]. In this filtration one compute the centre of mass of the nodes and grow a sphere from the centre outwards in uniform steps. In each step one determine the nodes located inside the growing sphere and connect two nodes when there is a vessel between them, resulting in a growing network. Then the persistent homology of connected components and loops can be computed. The connected components (dimension 0) of the radial filtration characterise the tortuosity: a vessel with high tortuosity will intersect the growing sphere multiple times and generate many small components which then quickly connect as the sphere radius increases and manifest in the barcode as multiple short bars. A tortuosity descriptor can be defined as the number of short bars in dimension 0 barcodes divided by the number of vessel

segments. A loops descriptor can also be defined using the number of loops (Betti number of dimension 1) divided by the number of vessel segments. Moreover, using the correspondent barcode, information about the spatial distribution of the loops with respect to the center can be obtained.

Another element indicative of the existence of a tumor is the presence of voids within the vascular system. To identify them, it was decided to use persistent homology, and in particular Alpha filtration constructed from the vertices of the graph. This allows the technique to be applied even to vascular networks of considerable size such as the one in Figure 12, since the number of simplices in the complex is kept small. In these cases, alternatives such as the Vietoris Rips filtration may become computationally infeasible. Furthermore, the Alpha filtration allows to obtain an estimate of the voids size by exploiting information contained in the correspondent barcode. Using this method, it is possible to understand whether there may be potential risk areas and also to have an estimate of the level of tumor advancement.

Figure 13 illustrates the chosen topological analysis pipeline.

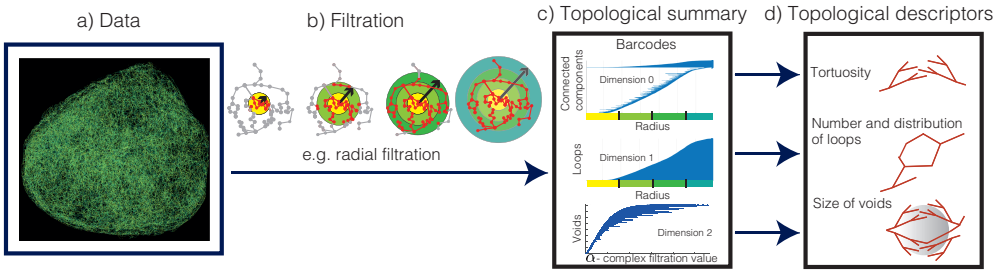

Figure 13: Illustration of topological data analysis for vascular network data.

3 Implementation

This section discusses the implementation of the methods described above.

3.1 Libraries for TDA

Several popular libraries are available for calculating homology and persistent homology, including Ripser, Dionysus, PHAT, Perseus and GUDHI. Each of them offers unique features and capabilities. As part of the project, a comprehensive evaluation of these tools was conducted to determine the most suitable one. Some libraries such as Ripser, Perseus and GUDHI were installed and tested on the data presented in the previous section. In addition, the results reported in [6] were extensively used. The authors of this Paper did a detailed comparison of many libraries, reporting their features, pros and cons.

After some analysis and comparisons, the GUDHI (Geometry Understanding in Higher Dimensions) library, written in C++, emerged as the preferred software for this project. It is supported by a Grant of the European Research Council and hosted by INRIA. GUDHI is known for its efficient and highly optimized algorithms and data structures, making it suitable for large and complex datasets. Another feature is that GUDHI has a very large range of tools. There are numerous filtered complexes, modules for calculating persistent homology, and even modules for plotting barcodes and persistence diagrams. A great advantage is that GUDHI offers well-kept documentation and helpful tutorials, creating a supportive environment for users to navigate the library and overcome challenges. Finally, the library provides interfaces to well-known programming languages like Python and R. This maintains the performance of the C++ code while adding the versatility of these languages in manipulating and visualizing data.

Cmake, Boost and CGAL (Computational Geometry Algorithms Library) are necessary prerequisites to successfully install the C++ GUDHI library. Moreover, the CGAL module requires at most GMP, MPFR and Eigen3 to be installed. Some parts of the code have been parallelized with the Intel TBB library. In order to use Python, one need to install Cython to compile the library. Numpy and Matplotlib are also needed for persistence visualization tools. An easier way may be to use the available pre-compiled Python module.

It was decided to implement the project in C++ since the GUDHI library is written in this language. The source files are in the `ph-metric-graphs/src/` folder. Subsequently, a Python version was also implemented to facilitate the visualization of networks, filtrations, and persistence (barcode and persistence diagrams). Python files are in `ph-metric-graphs/python/`.

3.2 C++ implementation

The algorithms and data structures needed for the topological analysis presented earlier are already implemented very efficiently in the GUDHI library. Therefore, it was not necessary to write the methods from scratch, making the code structure relatively simple.

The `NetworkData` class was implemented to represent a generic vascular network. It contains the vertices and edges of the network and methods for reading this data from files. To represent vertices, the class template `Point<T>` was defined, where `T` is the type used to store the coordinates. The dimension of the space is not specified in advance, since it is necessary to work with both 2-dimensional and 3-dimensional networks. The class includes some operators for addition and subtraction, as well as for multiplication and division by a scalar. Additionally, there is a method for calculating the Euclidean norm. These operations are useful for implementing the Radial filtration. To represent a vessel in the network, it was sufficient to use an `std::array<unsigned,2>` that stores the indices of the two connected nodes.

The main structure for topological analysis is the filtered simplicial complex, which is implemented in GUDHI as the template class `Gudhi::Simplex_tree<SimplexTreeOptions>`. All the simplices of the simplicial complex are explicitly stored in a trie whose nodes are in bijection with the simplices of the complex. In order to build a `Simplex_tree`, let K_0 a set of vertices and K the correspondent simplicial complex of dimension k. The vertices are labeled from 1 to $|K_0|$ and ordered accordingly. Each j-simplex $\sigma = \{v_{\ell_0}, \dots, v_{\ell_j}\}$ of K can be identified with a word $[\sigma] = [\ell_0, \dots, \ell_j]$ of length $j+1$, where $v_{\ell_i} \in K_0, \ell_i \in \{1, \dots, |K_0|\}$ and $\ell_0 < \cdots < \ell_j$.
The last label of the word representation of a simplex σ is denoted $\text{last}(\sigma)$. To represent the set of simplices of K, the `Simplex_tree` stores the corresponding words satisfying the following properties:

1. The nodes of the `Simplex_tree` are in bijection with the simplices of the complex. The root is associated to the empty face.
2. Each node of the tree, except the root, stores the label of a vertex. Specifically, a node associated to a simplex $\sigma \neq \emptyset$ stores $\text{last}(\sigma)$.
3. The vertices whose labels are encountered along a path from the root to a node associated to a simplex σ, are the vertices of σ. Along such a path, the labels are sorted by increasing order and each label appears no more than once.

The depth of the root is 0 and the depth of a node is equal to the dimension of the simplex it represents plus one.
All the nodes at the same depth j which contain the same label ℓ are linked in a circular list in order to quickly locate all the instances of ℓ in the tree. An example is reported in Figure 14.

Nodes which share the same parent are called *sibling nodes*. Dictionaries like red-black trees are used for searching, inserting and removing elements among a set of sibling nodes. They are preferred over hash maps to compute fast set operations traversing their elements in sorted order.

It is possible to store an appropriate filtration value for each simplex of the simplicial complex. In this way a filtration can be defined.

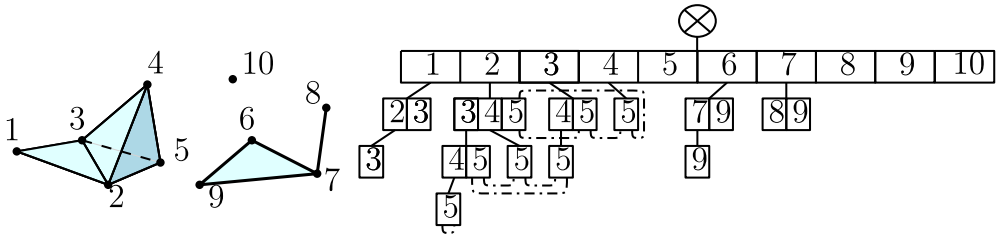

Figure 14: A simplicial complex and its simplex tree. Linked list for label 5 are shown.

`Simplex_tree` allows to efficiently implement a large range of operations on simplicial complexes. For example it is possible to search and insert a simplex of dimension j in $O(j \log(\deg(T))$ operations, where $\deg(T)$ is the maximal outdegree of a node. Many additional details are reported in Paper [2].

To make the implementation modular and easily extensible, an abstract class `Filtration` was implemented. This class encapsulates all the necessary ele-

ments for conducting an analysis of the vascular network through a specific filtration of a simplicial complex. The interface of the class is provided below:

```
class Filtration {
public:
  /// Represents a filtered simplicial complex
  using SimplexTree = Gudhi::Simplex_tree<>;
  using Filtration_value = SimplexTree::Filtration_value;
  /// Homology coefficents
  using Field_Zp = Gudhi::persistent_cohomology::Field_Zp;
  using Persistent_cohomology =
      Gudhi::persistent_cohomology::Persistent_cohomology<SimplexTree,
                                                          Field_Zp>;
protected:
  SimplexTree simplex_tree;
  std::unique_ptr<Persistent_cohomology> persistent_cohomology = nullptr;
  Chrono chrono;
public:
  void compute_persistent_cohomology(bool persistence_dim_max);
  void save_persistence(const string &filename) const;
  void print_complex_info();
  void print_range_simplices(unsigned start_index, unsigned length);
  void print_betti_numbers() const;
  void print_elapsed_time() const;
  virtual void make_analysis() = 0;
  virtual void print_analysis() = 0;
};
```

Listing 1: `Filtration` class declaration.

Some aliases for GUDHI classes are defined. `Field_Zp` represents the coefficient space of the chain vector spaces $C_p(K)$. The library is general and allows handling cases different from the one discussed in the introductory chapter, where $\mathbb{F}_2$ is used. For this application, it is sufficient to use the presented version. `Persistent_cohomology` is a template class that computes the persistent homology of a filtered simplicial complex. The template parameters are the type of data structure used to store the simplicial complex and the type of coefficients. Since the `Persistent_cohomology` class is not copy-constructible, it was decided to store a smart pointer to this class so that it can be initialized by child classes. There is also an object of the `Chrono` class to measure the time taken for the persistent homology computation. Regarding the available methods, `compute_persistent_cohomology` and `save_persistence` allow calculating and saving the persistent cohomology of `simplex_tree`. GUDHI uses an optimized version of the procedure presented in the introductory chapter to perform the computation. This version uses the algorithm detailed in Paper [3] and the Compressed Annotation Matrix implementation of Paper [1].

There are also a series of functions to print information. Some of them are not declared as `const` because they call functions from the `SimplexTree` class, which are also not declared as `const`. Finally, the last two methods are pure virtual and allow performing topological analysis through filtration and printing the results. The definition of these methods must be specified by the child classes of `Filtration`.

As mentioned earlier, two types of filtrations was identified to perform topological analysis of a vascular network: the Alpha and the Radial filtrations. These concepts are implemented as derived classes of the `Filtration` class.

The `AlphaFiltration` class declaration is reported in listing 2.

```
class AlphaFiltration : public Filtration {
public:
  using Kernel = CGAL::Epeck_d<CGAL::Dynamic_dimension_tag>;
  using AlphaPoint = Kernel::Point_d;
  using Alpha_complex = Gudhi::alpha_complex::Alpha_complex<Kernel>;
private:
  vector<double> voids_diameter;
  vector<std::pair<Filtration_value, Filtration_value>> voids_persistence;
public:
  AlphaFiltration() = default;
  AlphaFiltration(const vector<Point<CoordType>> &points);
  void compute_voids_diameter();
  void print_diameters(unsigned nmax) const;
  void make_analysis() override;
  void print_analysis() override;
};
```

Listing 2: `AlphaFiltration` class declaration.

It uses the GUDHI module `Alpha_complex<Kernel>` to construct a filtered Alpha simplicial complex. `Alpha_complex` is a template class that requires an `Epick_d` or `Epeck_d` d-D Geometry Kernel from CGAL as a template parameter. Using the default `CGAL::Epeck_d` ensures safe construction. On the other hand, using `CGAL::Epick_d` can result in slightly faster operations but the computation of filtration values can occasionally be arbitrarily bad. Therefore, the first option, `CGAL::Epeck_d`, was chosen.

`Alpha_complex` constructs a Delaunay Triangulation from CGAL and then creates a `Simplex_tree` with its cells, assigning the proper filtration value to each simplex. Refer to the GUDHI documentation for further details.

To construct the simplicial complex as described, it is necessary to provide an input vector of CGAL Point_d. Therefore, the constructor of the

`AlphaFiltration` class converts the points from `Point<T>` to `Point_d` and subsequently constructs the Alpha `Simplex_tree`.

This filtration is particularly useful for analyzing the $(d-1)$-dimensional voids or holes in the vascular network, where d is the dimension of the metric space of the network. In addition to studying the persistence of these voids, it is possible to provide an estimate of their size. This is achievable thanks to the filtration value at which the void disappears. This value represents the radius of the ball that circumscribes the simplex that kills the void. To better understand this, consider the example in Figure 15.

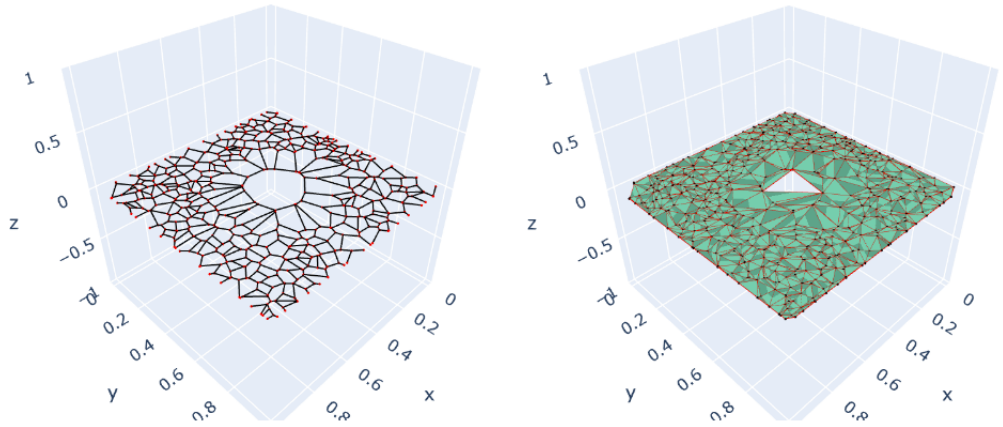

Figure 15: Patological bidimensional vascular network (left) and its Alpha complex for $\alpha = 0.13$ (right).

The patological void disappears when the last simplex (the empty triangle in the image on the right) appears in the filtration. This happens at $\alpha = 0.13081$, and this value corresponds to the radius of the circle that circumscribes the triangle, which is a good approximation of the void's radius.

The `compute_voids_diameter` function calculates the diameters of the voids as described earlier. Finally, `make_analysis` computes the persistent homology and calls the previous function to obtain the diameters of the voids.

The `RadialFiltration` class declaration is reported in listing 3.

```
class RadialFiltration : public Filtration {
public:
  unsigned n_edges;
  double max_radius;
  double tortuosity_descriptor;
```

```
  double loops_descriptor;

  RadialFiltration() = default;
  RadialFiltration(const vector<Point<CoordType>> &points,
                   const vector<Edge> &edges);
  void compute_descriptors();
  void make_analysis() override;
  void print_analysis() override;
};
```

Listing 3: `RadialFiltration` class declaration.

The Radial filtration is not a well-known filtration in the literature. It was introduced in Paper [7] and its implementation is not publicly available. It is only mentioned that the code was written in MATLAB, discretizing the radius values into a certain number of steps. In GUDHI, this filtration is not present, but it can be easily implemented thanks to the `Simplex_tree` data structure. The constructor of the `RadialFiltration` class handles this.

Initially, the centroid of the network's vertices is computed. Then, the vertices and edges of the network are added to the simplicial complex with the appropriate filtration values. For vertices, the value is calculated as the Euclidean distance between the vertex and the centroid. For edges, it is calculated as the maximum of the filtration values of the two connected vertices. During the filtration creation, the maximum distance of a point from the center and the number of edges inserted into the simplicial complex are also calculated. This latter number may differ from the one read from the data because the data can be noisy and may contain the same edge repeated. Consequently, a simplex already present in the simplicial complex is not added.

Once the persistent homology is computed using the inherited function `compute_persistent_cohomology`, the `compute_descriptors` method can be used to calculate the topological descriptors mentioned earlier. The tortuosity descriptor is defined as the number of connected components in dimension 0 barcodes with persistence less than or equal to 10% of the maximum distance between a vertex and the center. The loop descriptor is simply calculated as the number of loops. Both descriptors are then divided by the number of vessel segments to ensure that the contributions are topological.

The `make_analysis` method computes the persistent homology and calls the `compute_descriptors` function.

The complete analysis is performed by the `NetworkAnalysis` class. The interface is provided in Listing 4.

```
class NetworkAnalysis {
private:
  const NetworkData *data;
  std::vector<std::unique_ptr<Filtration>> filtrations;

public:
  NetworkAnalysis() = default;
  NetworkAnalysis(const NetworkData *data_);
  void analyse() const;
  void print_global_analysis() const;
  void save_persistence(const std::string &name) const;
};
```

Listing 4: `NetworkAnalysis` class declaration.

A pointer to the vascular network data to be analyzed and a vector of pointers to the base class `Filtration` are stored in this class. In the constructor, two objects of type `RadialFiltration` and `AlphaFiltration` are instantiated and `unique_ptr` to these objects are inserted into the filtrations vector. The functions `analyse` and `print_global_analysis` simply iterate through the filtrations vector and for each filtration call the overridden functions `make_analysis` and `print_analysis`, respectively. This way, if you wanted to define a new filtration for a different topological analysis in the future, you would only need to insert a pointer to that filtration into the vector. Finally, the `save_persistence` method saves the persistent homology of the various filtrations to files.

The `main` program is quite straightforward: it uses `GetPot` to select the network to be analyzed, parses the files of the vascular network, and then calls methods of the `NetworkAnalysis` class.

3.3 Python implementation

As previously mentioned, the GUDHI library provides a precompiled Python interface. It was decided to implement topological analysis also in this language, adding useful functions for visualizing vascular networks, filtrations, and the results of persistent homology computation through persistence barcodes and persistence diagrams. To achieve this, the Python library Plotly was used to generate dynamic 3-dimensional visualizations.

Similar to the C++ case, a class called `Network` was defined to store the nodes and vessels of a vascular network by reading them from files. Additionally, the class provides a `plot` method that generates a dynamic visualization of the vascular network. This plot is customizable with various graphical parameters, and the result can be saved to an `.html` file. The `Network` class also offers a `plot_static` method, similar to the previous one, except that it produces a static visualization. It is possible to save this plot to a `.pdf` file.

Regarding topological analysis, the same structure as the C++ implementation was followed, defining a base class called `Filtration` and two derived classes: `RadialFiltration` and `AlphaFiltration`. The implementation follows the same path as before, making use of the Python version of the `Simplex_tree` data structure. In this case, its usage is even more flexible, as the module for persistent homology calculation is implemented within the `Simplex_tree` class.

The added value of the Python version is given by the methods `plot_persistence` and `plot_simplicial_complex` of the base class `Filtration`. The first method allows for the visualization and saving of persistence barcodes and persistence diagrams using the Persistence graphical tools module from GUDHI.

The second method uses the Plotly library and generates a dynamic 3-dimensional plot of the simplicial complex for a given filtration value. It displays all the simplices of the simplicial complex that have a filtration value lower than the one provided with the `filtration_value` parameter. The `max_skeleton_order` parameter can be set to visualize only simplices of dimension less than or equal to its value. In this case as well, the result can be saved to an `.html` file.

Two Jupyter notebooks were written to perform topological analysis on artificially generated vascular networks and on the real one, respectively. They use the methods described earlier, and the dynamic plots can be viewed directly in the notebooks.

Lastly, a Python script was implemented to save the barcodes and persistence diagrams from the files produced by the C++ code. This way, persistence visualizations can be obtained even without using the Python `NetworkAnalysis` module.

4 Results

In this section, the results of the topological analyses conducted on the vascular networks presented earlier are summarized.

4.1 Synthetic networks

The goal of artificially generated vascular networks is to simulate the presence of a tumor by creating a void within the network. Therefore, for such networks, the most effective filtration was the Alpha one, capable of highlighting persistent voids.

The Alpha filtration implemented in GUDHI uses the square of the radius that circumscribes each simplex as filtration value. Therefore, to interpret the persistence information correctly, it is necessary to take the square root of the filtration values. Table 1 presents the results of the Alpha filtrations for the three networks in Figure 11.

The filtration is capable of identifying the abnormal void in pathological networks, which is much more persistent and larger in size compared to the others. The persistence intervals $(\text{birth}, \text{death})$ are also depicted in the barcodes in Figure 19, which better highlight the presence of the dominant void. Additionally, Figure 20 displays some Alpha simplicial complexes of the pathological network characterized by the parameter 0.1. It can be observed that the void is correctly identified by the filtration and it is the last to die. The last simplicial complex corresponds to the Delaunay complex constructed with the vertices of the vascular network.

The Radial filtration correctly identifies all the loops in the vascular network, and their number can be obtained from the 1-dimensional Betti number of the complete simplicial complex. Additionally, all the connected components that arise during the filtration are identified. The artificial generation of networks is not sophisticated enough to simulate greater tortuosity and loops localized around the tumor. Consequently, the Radial filtration applied to these networks struggles to distinguish pathological ones from healthy ones.

Table 2 shows the results obtained from the Radial filtration applied to the networks in Figure 11.

The pathological networks have a higher tortuosity index compared to the healthy one. However, the descriptor for loops doesn't prove to be useful

(birth, death)	$\sqrt{p}$	diameter
(0.00104, 0.00436)	0.05762	0.13208
(0.00173, 0.00456)	0.05319	0.13503
(0.00125, 0.00360)	0.04852	0.12004
(0.00067, 0.00287)	0.04689	0.10711
(0.00108, 0.00299)	0.04372	0.10934

(a) Synthetic 0.0

(birth, death)	$\sqrt{p}$	diameter
(0.00103, 0.01132)	0.10143	0.21275
(0.00111, 0.00417)	0.05533	0.12913
(0.00106, 0.00402)	0.05441	0.12681
(0.00282, 0.00563)	0.05295	0.15004
(0.00146, 0.00423)	0.05260	0.13002

(b) Synthetic 0.1

(birth, death)	$\sqrt{p}$	diameter
(0.00084, 0.01711)	0.12755	0.26161
(0.00257, 0.00735)	0.06913	0.17146
(0.00212, 0.00682)	0.06856	0.16512
(0.00206, 0.00671)	0.06818	0.16382
(0.00156, 0.00552)	0.06294	0.14862

(c) Synthetic 0.2

Table 1: Results of the Alpha filtration applied to the three vascular networks in Figure 11

since the networks are generated to have approximately the same number of windings.

4.2 Real network

After testing the methods on artificially generated vascular networks, topological analysis was performed on the vascular network in Figure 12. As mentioned earlier, it is the brain network of a mouse in which a tumor has been implanted.

Table 3 presents the persistence information for the first 10 voids identified by the Alpha filtration. The data is presented in descending order of persistence, similar to Table 1.

	Network 0	Network 1	Network 2
Loops number	210	223	229
Loops descriptor	0.2811	0.2802	0.2799
Tortuosity descriptor	0	0.0013	0.0012

Table 2: Results of the Radial filtration applied to the vascular networks in Figure 11.

(birth, death)	$\sqrt{\text{p}}$	diameter
(195758, 466836)	520.652	1366.51
(104306, 271075)	408.374	1041.3
(111517, 261906)	387.801	1023.54
(89133.9, 231368)	377.139	962.014
(311026, 439792)	358.84	1326.34
(145373, 256354)	333.138	1012.63
(100880, 210850)	331.617	918.369
(95344.2, 204643)	330.604	904.75
(106776, 200246)	305.729	894.976
(111536, 201720)	300.307	898.266

Table 3: Results of the Alpha filtration applied to the vascular network in Figure 12

In Figure 16, the barcodes of the 100 most persistent voids are displayed. Figure 17 represents the persistence diagram, which, in addition to the same information about the voids, also includes the persistence of loops and connected components of the Alpha filtration.

The Alpha filtration has highlighted a more persistent and larger void compared to the others (the first one in Table 3). Additionally, it has identified another void, less persistent but with a high estimated diameter (the fifth one in Table 3). These voids may correspond to areas of tumor necrosis due to the tumor's rapid growth outpacing its ability to produce VEGF and stimulate angiogenesis for its sustenance.

The results of the radial filtration are presented in Table 4.

It's noticeable that the tortuosity descriptor is much higher compared to the case of artificially generated networks. In Figure 18, the persistence barcode of the connected components of the Radial filtration is shown. There are

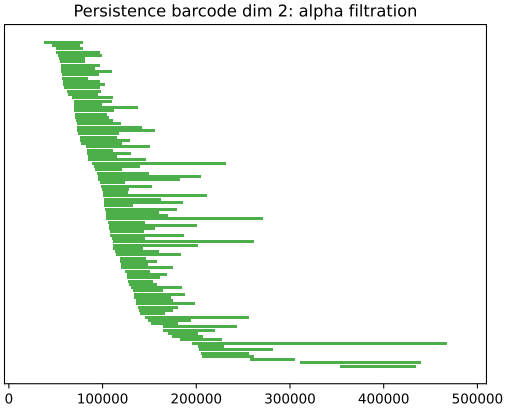

Figure 16: Voids persistence barcode obtained trough the Alpha filtration of the vascular network in Figure 12.

Descriptor	Value
Loops number	5857
Loops descriptor	0.2446
Tortuosity descriptor	0.1106

Table 4: Results of the radial filtration applied to the vascular network in Figure 12.

numerous small intervals, corresponding to connected components that die shortly after birth. This can be interpreted as an indicator of high tortuosity. In order to verify the effectiveness of filtrations in identifying pathology, it would be valuable to have additional data from the same network at different stages of tumor development. These data are not available for the analysed network but the validity of this approach is supported by the results presented in Paper [7].

Finally, in Figure 21, some simplicial complexes from the radial filtration are shown.

Regarding the performance of the C++ implementation, the computation of persistent homology is the most computationally expensive algorithm. The standard version presented in the introductory chapter has a complexity of $O(m)$, where m is the number of simplices. GUDHI implements an extremely efficient version that achieve better results. Additionally, choosing filtrations

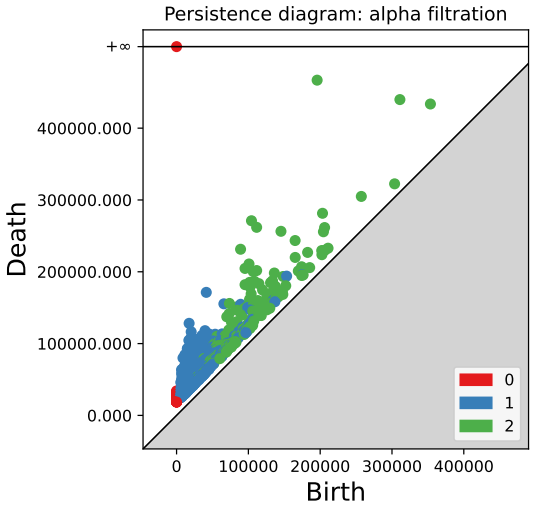

Figure 17: Persistence diagram obtained trough the Alpha filtration of the vascular network in Figure 12.

with a small number of simplices allows for topological analysis even on very large vascular networks.

In the most complex case available, the 3-dimensional real network, the Alpha filtration consists of 517655 simplices and persistent homology is calculated on average in 240 milliseconds. On the other hand, the Radial filtration is much smaller, composed only of elements from the vascular networks. In the case of the 3-dimensional network, it consists of 42042 simplices, and the calculation is performed on average in 4.8 milliseconds.

The Python version also achieves similar performance because it is just an interface, and the source code is the same as the C++ version.

5 Conclusions

This project successfully used persistent homology to characterize pathological vascular networks associated with tumors. In particular Alpha and Radial filtrations were employed to identify voids, loops, and tortuosity within these networks.

The primary contribution of this work lies in the unified, structured and efficient implementation of a framework for topological analysis of vascular

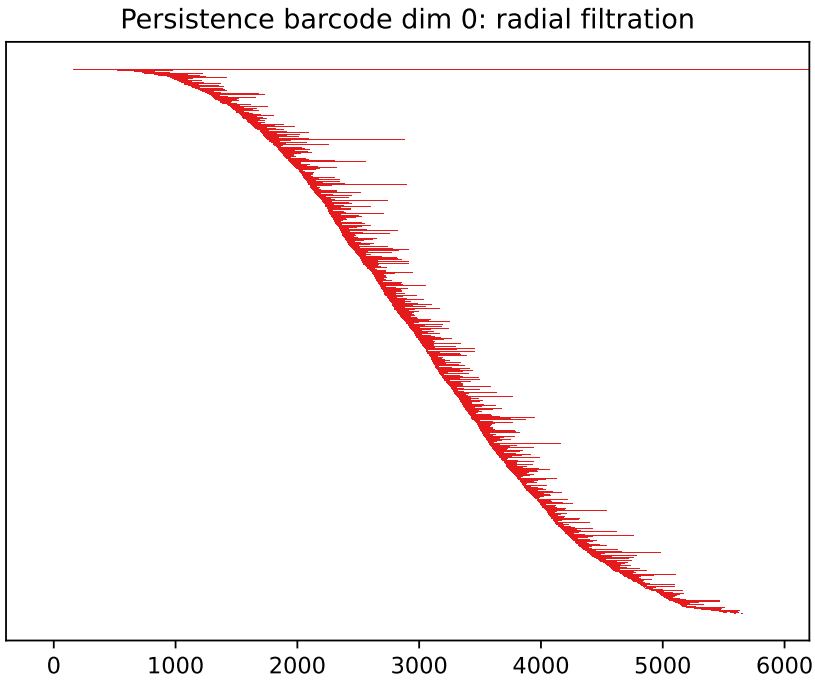

Figure 18: Connected components persistence barcode obtained trough the Radial filtration of the network in Figure 12.

networks. This framework is adaptable to various domains and provides a foundation for future researches.

Moving forward, one possible direction could involve leveraging topological information to train neural networks that takes input from the analysis data and classifies vascular networks as pathological or non-pathological. This approach promises more accurate diagnosis, ultimately contributing to improved healthcare outcomes in the field of vascular pathology.

References

[1] Jean-Daniel Boissonnat, Tamal K Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *European Symposium on Algorithms*, pages 695–706. Springer, 2013.

[2] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70:406–427, 2014.

[3] Tamal K Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 345–354, 2014.

[4] Tamal Krishna Dey and Yusu Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.

[5] Herbert Edelsbrunner and John L Harer. *Computational topology: an introduction*. American Mathematical Society, 2022.

[6] Nina Otter, Mason A Porter, Ulrike Tillmann, Peter Grindrod, and Heather A Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6:1–38, 2017.

[7] BJ Stolz, J Kaeppler, B Markelc, F Mech, F Lipsmeier, RJ Muschel, HM Byrne, and HA Harrington. Multiscale topology characterises dynamic tumour vascular networks. arxiv. *preprint*, 2020.

[8] Paul W Sweeney, Angela d'Esposito, Simon Walker-Samuel, and Rebecca J Shipley. Modelling the transport of fluid through heterogeneous, whole tumours in silico. *PLoS computational biology*, 15(6):e1006751, 2019.

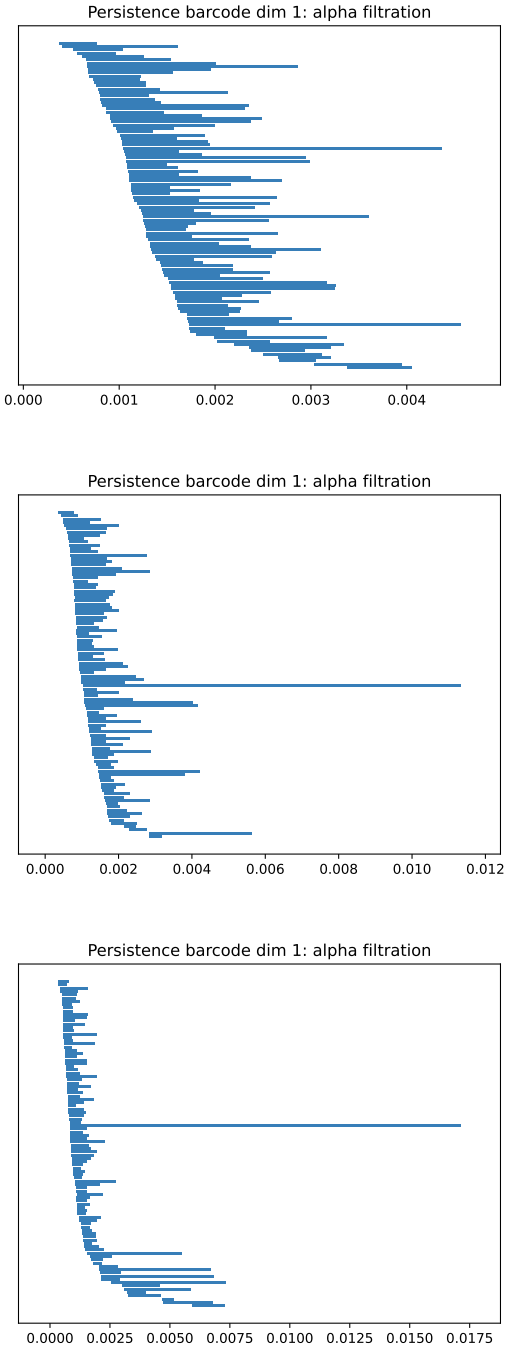

Figure 19: Voids persistence barcodes obtained trough the Alpha filtration of the three networks in Figure 11.

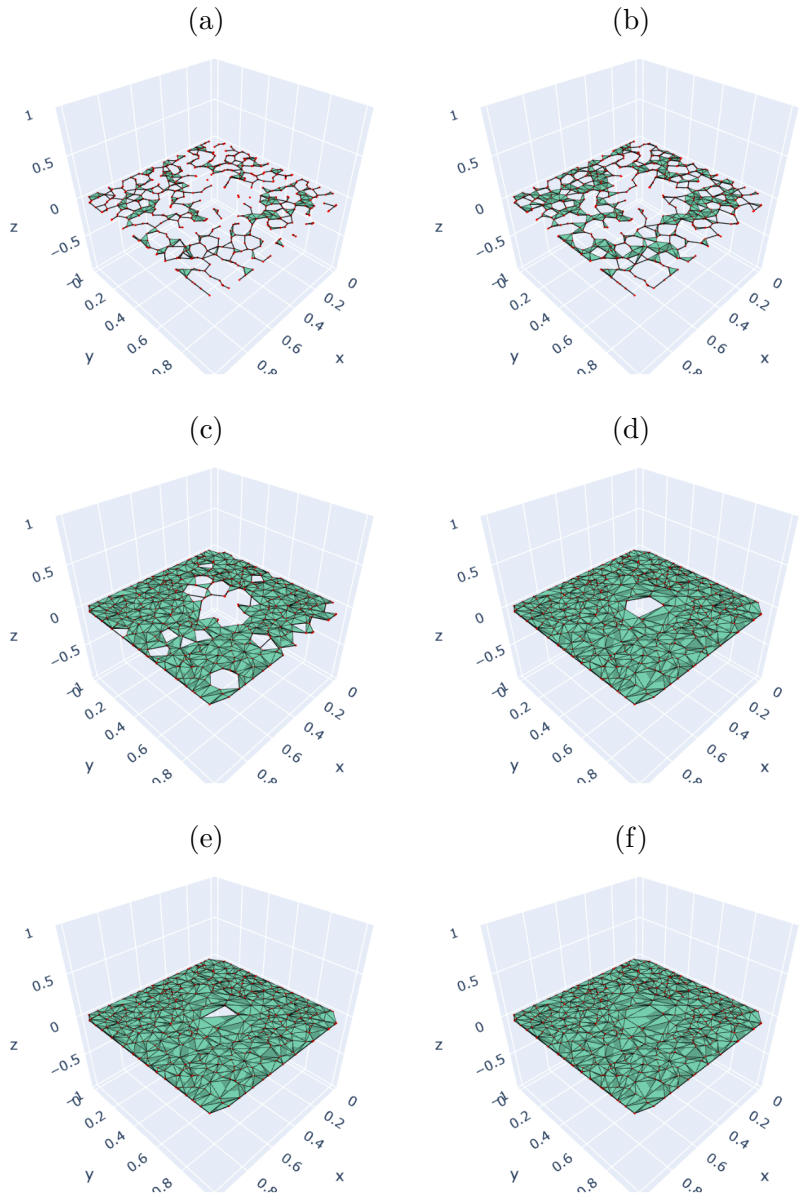

Figure 20: Alpha filtration: some simplicial complex of the synthetic vascular network with parameter 0.1 in Figure 11.

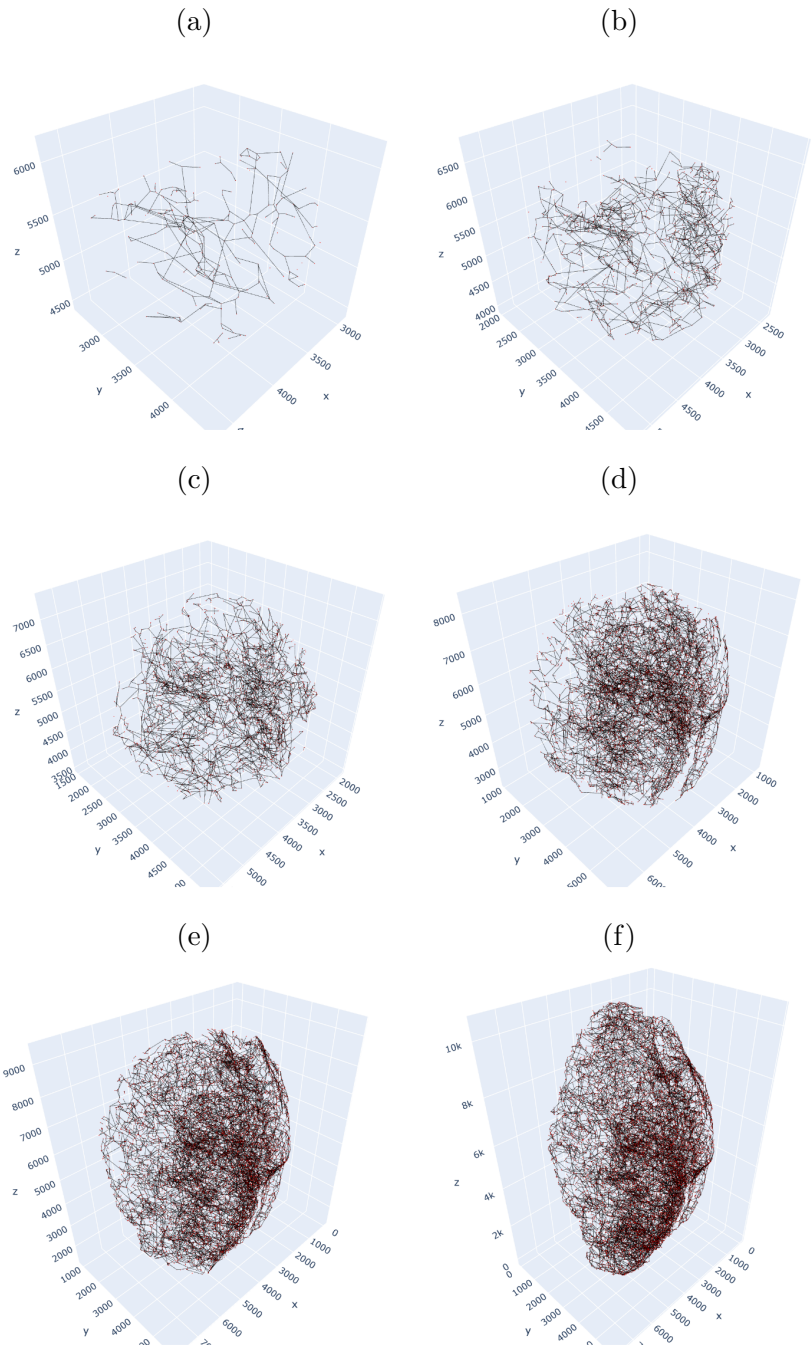

Figure 21: Radial filtration: some simplicial complex of the vascular network in Figure 12.