

An Introduction to Persistent Homology and Simplicial Collapses

Abhijit Bhalachandra

MS16035

*A dissertation submitted for the partial fulfilment of
BS-MS dual degree in Science*




Indian Institute of Science Education and Research, Mohali

April 2021

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Prof. Michael Kerber at the Technische Universität, Graz, Austria, and Dr. Krishnendu Gongopadhyay at the Indian Institute of Science Education and Research, Mohali, India.

This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of work done by me and all sources listed within have been detailed in the bibliography.



Abhijit Bhalachandra
(MS16035)

In my capacity as the supervisor of the candidate's project work, I certify that the above statements by the candidate are true to the best of my knowledge.



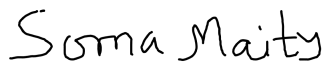
Dr. Krishnendu Gongopadhyay
(Co-supervisor)

Certificate of Examination

This is to certify that the dissertation titled “**Introduction to Persistent Homology and Simplicial Collapses**” submitted by Mr. Abhijit Bhalachandra (Reg. No. MS16035) for the partial fulfilment of BS-MS dual degree programme of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.



Dr. Abhik Ganguli



Dr. Soma Maity



Dr. Krishnendu
Gongopadhyay
(Co-supervisor)

Acknowledgements

I would like to thank my guide Prof. Michael Kerber, without whom this thesis would not have been possible. I also thank Dr. Krishnendu Gongopadhyay for his support and supervision of my progress. I am thankful to the faculty members at IISER Mohali for all the wonderful interactions, courses I've taken, and introducing me to the amazing world of science, mathematics, and philosophy. I am eternally grateful to IISER Mohali and everyone I've interacted with during my five years here, as every moment on campus has changed me in different ways, and has made me grow as a person.

Thank you Amma and Dad for your motivation, support, and unconditional love.

Special thanks to my prior summer internships guides Dr. Pierre Lairez and Dr. Amit Chattopadhyay, working with whom has made me see and appreciate mathematics through various lenses.

I've been fortunate to have made some of the closest friendships during my time on campus. I've cherished every moment with all of you. Thank you Nikhil for always being there. Anshuman, Rahul, and Rishi, I will miss our long walks and conversations. Thank you Kausthub, Prashant, and Vedang for all the memories on the tennis court and our shared experiences at tournaments.

Last but not the least, life would have been very different had I not taken to quizzing early on. I thank the IISER Mohali Quiz Club for all the brilliant fundaes, conversations, and rich experiences. My quizzing experience would have been very different had I not had excellent teammates. Thank you Ojaswi for being a superb teammate and an even better friend! Thank you Ayush, Dhruv, and Shantanu for all the memories both a part of and outside of quizzing.

Abstract

The explosion of data has brought in the fervent need to analyze large and higher-dimensional datasets accurately and fast. Conventional tools are quickly becoming redundant when the focus is on the speed of computation and the expectations of impactful insight. There is a vibrant community of researchers who are looking at topology-based tools which are able to extract shape-pertinent features of these large datasets. Looking at alternate and non-classical tools has led to the development of some of the most impactful sub-fields of mathematics, one being Topological Data Analysis, which has been widely accepted and noticed for its effectiveness on certain use cases. This thesis will focus on studying a method called Persistent Homology, which in a sense forms the vein of Topological Data Analysis. This thesis will build mathematical theory to understand Persistent Homology, and subsequently proceed to comment on contemporary challenges with regard to the method, and novel techniques to overcome them including algorithmic approaches with experimental observations.

List of Figures

2.1	Boundary matrix after performing row and column operations [EH10]	8
3.1	Representation of the height function applied on an upright torus with critical points u, v, w, z [EH10]	10
3.2	Evolution of the sublevel set [EH10]	11
3.3	The first two barycentric subdivisions of the given simplicial complex [EH10]	13
3.4	Example of a Vietoris-Rips complex [CdSEG07]	15
4.1	Barcode and persistence diagram representations of the persistence elements $[1, 2], [1, 3], [3, 3], [3, 4]$, and $[3, 4]$ [GC09]	17
4.2	Point cloud approximation of a circle	19
4.3	One persistent generator for H_1 corresponding to a loop	19
4.4	Point cloud approximation of a sphere	20
4.5	One persistent generator for H_2 corresponding to a void	20
4.6	Point cloud approximation of a torus	21
4.7	Three persistent generators, two for H_1 and one for H_2	21
4.8	Bifiltration of a triangle [GC09]	22
5.1	An elementary strong collapse. Here, v is dominated by v' . [BP19]	26
5.2	Constructing \mathbb{G}_{i+1} from \mathbb{G}_i [BP19]	30
6.1	Example 1	35
6.2	Example 2	36
6.3	FIRep file format	38
6.4	Computing the minimal presentation of a test case	39

Contents

1	Introduction	1
2	Homology Theory	3
2.1	Simplicial Homology	3
2.2	Computing Homology of a Simplicial Complex	7
3	An Introduction to Morse Theory and Filtrations	10
3.1	Morse Theory	10
3.2	Filtrations	13
4	Persistent Homology	16
4.1	Computing Persistence	16
4.1.1	Circle	19
4.1.2	Sphere	20
4.1.3	Torus	20
4.2	Multi-dimensional Persistence	21
5	Collapses	23
5.1	Prerequisites	24
5.2	Computing Persistent Homology of Flag Complexes using Strong Collapses	28
5.3	An Approach for Multi-dimensional Persistence	31
6	Computational Experiments	33
6.1	Input Bifiltration	34
6.1.1	Examples by Hand	34
6.1.2	Adding Edges Randomly	37
6.1.3	function-Rips	37
6.2	Reduction using Algorithm-R	37
6.3	Computing Minimal Presentation of Reduced Complex	37
6.4	Verification	38

6.5 Results	39
7 Concluding Remarks	40
A Collapses	41
A.1 Computing Persistent Homology of Flag Complexes using Strong Collapses	41
B Computational Experiments	44

Chapter 1

Introduction

Computational Topology is a meeting ground for mathematics and computer science. The field aims to address pertinent problems in topology by developing efficient algorithms for the same. In this process, numerous applications of these algorithms usually come to light. Some problems in topology include computing the fundamental group of a surface, computing the homology group, and determining the genus of a surface among others. Some applications which have surfaced include using these algorithms to address the protein docking problem, image classification problems, and trajectory analysis among others.

This thesis will focus singularly on Persistent Homology, a technique in Computational Topology which aims at discovering the “intrinsic shape” of a topological space. The idea is to look at the topological space through the lens of a filtration, and to track homological components of the space as they appear and disappear with respect to the filtration.

This thesis aims at introducing and building relevant mathematics to understand contemporary research in developing Persistent Homology. Much of the latter part of the thesis, which looks at methods to improve the computational performance of Persistent Homology is a survey on the research carried out by Siddharth Pritam during his PhD [Sid20], and his current idea we are working on in the field of Multi-dimensional Persistence [Pri21].

A lot of interesting developments in Persistent Homology come from the implementation side of the method, either through writing efficient code and building robust packages, or coming up with ingenious algorithms to reduce computational time and/or resources. This thesis aims to look at both these aspects and includes results from computational experiments of the same.

We start with the basics of Homology Theory in Chapter 2, and will provide an algorithmic

method to compute the rank of the homology group. Most of the explanation is inspired from my readings of the popular introductory text in Computational Topology by Herbert Edelsbrunner and John Harer [EH10]. In the subsequent chapter, we will look at grasping the basics of Morse Theory, and to understand how Filtrations of Simplicial Complexes form the soul of Persistent Homology. Contents of the same are again learnings from [EH10]. In Chapter 4, we will provide the definition of Persistent Homology, its mathematical basis (from the seminal work by Gunnar Carlsson and Afra Zomorodian [GC09]), and look at some examples where we compute the same. In the next chapter, we get familiar with the future possibilities of Persistent Homology, of how one could possibly scale and improve existing techniques of computing Persistent Homology. We will address this through the spirit of simplicial collapses, through the work [Sid20]. In the penultimate chapter, we provide evidence of some of the computational experiments we've done of trying to algorithmically reduce bifiltration data in order to speed up computation of Persistent Homology. The last chapter will comment on the span of the learnings, and a mention of future work and further improvements.

Chapter 2

Homology Theory

This chapter is an introduction to Homology Theory, specifically Simplicial Homology, and will build the pre-requisites required for understanding Persistent Homology. The chapter will start by defining what a simplicial complex is and will build upto algorithmically computing the rank of the n th-homology group.

2.1 Simplicial Homology

Definition 1 (Affine Combination of Points). Let u_0, u_1, \dots, u_k be a collection of points in \mathbb{R}^n . $x = \sum_{i=0}^k \lambda_i u_i$, $\lambda_i \in \mathbb{R}$ is an affine combination of u_i s if $\sum_{i=0}^k \lambda_i = 1$.

Definition 2 (Affine Hull). The affine hull is the set of affine combinations. The affine hull is a k -plane if the taken $k + 1$ points are affinely independent, implying that for $x = \sum \lambda_i u_i$ and $y = \sum \mu_i u_i$, $x = y$ iff. $\lambda_i = \mu_i$.

Definition 3 (Convex Combination). An affine combination is said to be a convex combination if λ_i s are non-negative. The convex hull is the set of convex combinations.

Definition 4 (Convex Hull). The convex hull is the set of convex combinations.

Definition 5 (Simplex). A k -simplex is a convex hull of $k + 1$ affinely independent points, expressed as $\sigma = \text{conv}\{u_0, \dots, u_k\}$. We can also define $|\sigma| = k + 1$.

Definition 6 (Face of a Simplex). A face of a simplex σ is the convex hull of a non-empty subset of u_i s, and it is said to be proper if the subset is not the entire set. For a face τ of σ , if it's proper, we denote it as $\tau < \sigma$, else as $\tau \leq \sigma$.

Definition 7 (Simplicial Complex). A simplicial complex K is a finite collection of simplices such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and for $\sigma' \in K$, $\sigma \cap \sigma' = \emptyset$ or is a face of both σ and σ' .

Definition 8 (Underlying Space). The underlying space of K , denoted as $|K|$, is defined as the union of all the simplices of K with the underlying topology inherited from the ambient Euclidean space the simplices live in.

Definition 9 (Simplicial Chain). A simplicial p -chain in K is a sum of p -simplices in K . For example, $c = \sum \alpha_i \sigma_i$, where the α_i s are coefficients in \mathbb{Z} , \mathbb{Q} , etc.

Proposition 2.1.1. All the p -chains form an Abelian group denoted by $(C_p, +)$.

Proof. The proof is through verification of all properties of an Abelian group -

- The 0 p -chain is the identity element.
- For any two p -chains c_1 and c_2 , $c_1 + c_2 \in C_p$, as C_p contains all combinations of p -chains.
- The inverse element of a p -chain c is simply $-c$.
- Addition of p -chains is associative in nature, i.e. for $c_1, c_2, c_3 \in C_p$, $c_1 + (c_2 + c_3) = (c_1 + c_2) + c_3$.
- The group is Abelian as addition of p -chains is commutative in nature.

□

Definition 10 (Boundary of a Simplex). For $\sigma \in C_p$, the boundary of σ , represented as ∂_p is calculated as $\partial_p \sigma = \sum_{i=0}^p [u_0, \dots, \hat{u}_i, \dots, u_p]$. In the sum, \hat{u}_i is the removed vertex.

Proposition 2.1.2. $\partial_p \circ \partial_{p+1} \sigma = 0$ where σ is a $(p+1)$ -chain.

Proof. Let $t_i(\sigma) = \sigma \setminus \{u_i\}$. The boundary operator can be redefined as $\partial_p \sigma = \sum_{i=0}^p (-1)^i t_i$. Hence -

$$\partial_{p+1} = \sum_{i=0}^{p+1} (-1)^i t_i$$

$$\begin{aligned}
\partial_p \circ \partial_{p+1} &= \sum_{j=0}^p (-1)^j t_j \left(\sum_{i=0}^{p+1} (-1)^i t_i \right) \\
&= \sum_{j < i} (-1)^j (-1)^i t_j t_i + \sum_{j > i} (-1)^{j-1} (-1)^i t_j t_i
\end{aligned}$$

The terms from the first summation will cancel out the terms from the second summation, thereby completing the proof. \square

Definition 11 (Chain Complex). A chain complex is a cascading sequence of simplicial complexes connected by the boundary map -

$$\dots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \dots$$

Definition 12 (Group of p -cycles). Having defined the boundary map, we define the group of p -cycles, represented as Z_p as the kernel of ∂_p , or $Z_p = \ker \partial_p$.

Definition 13 (Group of p -boundaries). Similarly, we define the group of p -boundaries, represented as B_p as the image of ∂_{p+1} , or $B_p = \text{im} \partial_{p+1}$.

Definition 14 (Homology Group). The p -th homology group, represented as H_p is the group of p -cycles quotiented out with the group of p -boundaries, or $H_p = Z_p / B_p$.

Definition 15 (Betti Number). The p -th Betti number is defined as $\beta_p = \text{rank} H_p$. Or by the definition of H_p , $\beta_p = \text{rank} Z_p - \text{rank} B_p$.

Definition 16 (Simplicial Map). Given two simplicial complexes K and L , a simplicial map f is a function $f : K \rightarrow L$ which always maps a simplex in K to a simplex in L . Simplicial maps are induced by vertex-to-vertex maps.

We can redefine the above as follows -

For two simplicial complexes K and L , let c be a p -chain in K , defined as $c = \sum a_i \sigma_i$. We obtain $f_{\#}(c) = \sum a_i \tau_i$ where $\tau_i = f(\sigma_i)$ if $f(\sigma_i)$ has dimension p , else $\tau_i = 0$ if the dimension of $f(\sigma)$ is less than p .

By representing the boundary maps in K and L as ∂_K and ∂_L respectively, we can come to this observation -

Proposition 2.1.3. $f_{\#} \circ \partial_K = \partial_L \circ f_{\#}$

Proof. The above is a natural observation, and can be noted in a straightforward manner. When $f(\sigma_i)$ has dimension p , all the $(p-1)$ -faces of σ_i will map to the $(p-1)$ -faces of τ_i . When $f(\sigma_i)$ has dimension less than p , we can observe that $(p-1)$ -faces of σ_i will map to simplices of dimension lower than $p-1$. It is possible that they also map to two $(p-1)$ -faces whose images coincide and cancel out each other. The idea here is to observe for a dimension lower than p , both operations $f_{\#}(\partial_K \sigma_i)$ and $\partial_L f_{\#}(\sigma)$ will be 0. \square

Continuing on with the above observation, we can also observe that the induced map $f_{\#}$ takes cycles to cycles, or $f_{\#}(Z_p(K)) \subseteq f_{\#}(Z_p(L))$. And it also takes boundaries to boundaries, or $f_{\#}(B_p(K)) \subseteq f_{\#}(B_p(L))$.

Aimed with the above observations, given a simplicial map $f : K \rightarrow L$, we can define the induced map on homology, represented as $f_* : H_p(K) \rightarrow H_p(L)$. [EH10] also notes that the rank of the image is bounded from above by the Betti numbers $\beta_p(K)$ and $\beta_p(L)$, viz. $\text{rank } f_*(H_p(K)) \leq \min\{\beta_p(K), \beta_p(L)\}$.

Definition 17 (Euler characteristic). The Euler characteristic of a simplicial complex is equal to the alternating sum of the number of simplices in each dimension. Let $z_p = \text{rank } Z_p$ and $b_p = \text{rank } B_p$. Therefore, the number of p -simplices in K is simply $n_p = \text{rank } C_p = z_p + b_p$. The Euler characteristic is represented as χ , and from our definition, we can see that -

$$\chi = \sum_{p \geq 0} (-1)^p (z_p + b_{p-1})$$

The above expression can be obtained by observing that n_p is the sum of the ranks of $\ker \partial_p$ and $\text{im } \partial_{p-1}$ respectively.

The expression can be rewritten as -

$$\chi = \sum_{p \geq 0} (-1)^p (z_p - b_p)$$

But $z_p - b_p$ is the Betti number, thus -

$$\chi = \sum_{p \geq 0} (-1)^p \beta_p$$

This result is popularly known as the Euler-Poincaré theorem.

Another important observation one can make is that isomorphic simplicial complexes will have the same Euler characteristic. This stems from the fact that isomorphic simplicial complexes will have the same simplicial homology, and thereby the same Betti numbers as well.

2.2 Computing Homology of a Simplicial Complex

We will look at a classical algorithm to compute homology of a simplicial complex in this section, given in [EH10].

The p -boundary map can be expressed as a matrix, viz. $\partial_p = [a_i^j]$. Hence, for any given p -chain $c = \sum_i a_i \sigma_i$, $\partial_p c$ can be computed by the matrix multiplication -

$$\begin{bmatrix} a_1^1 & a_1^2 & \cdots & a_1^{n_p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_{p-1}}^1 & a_{n_{p-1}}^2 & \cdots & a_{n_{p-1}}^{n_p} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{n_p} \end{bmatrix}$$

The matrix can be interpreted as the rows of ∂_p forming a basis of the C_{p-1} , and the columns forming the basis of C_p . The idea is to reduce the p -th boundary matrix to its Smith-Normal form, and subsequently compute β_p .

The Smith-Normal form of a matrix looks of the form -

$$\begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

In our context, the matrix will look something like the one shown below.

For reducing ∂_p , we can use Gaussian elimination. Essentially, we would like to perform two exchange operations wherein we are able to obtain a 1 in the upper left corner, and simply proceed to obtain 0s in the rest of the first row and first column.

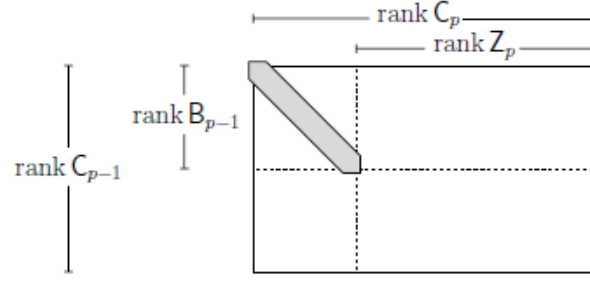


Figure 2.1: Boundary matrix after performing row and column operations [EH10]

The algorithm of Gaussian elimination performs over n_{p-1} row and n_p column operations per recursive call, and thereby performing $(n_{p-1} + n_p)\min\{n_{p-1}, n_p\}$ operations in total. Taking into consideration their lengths, we obtain a running time of a constant times $2n_{p-1}n_p\min\{n_{p-1}, n_p\}$. The memory utilized will simply be the memory required to store the matrices, which would be $(n_{p-1} + n_p)^2$.

We will look at an example where we use the above algorithm for reduction, and subsequently compute the Betti numbers of a tetrahedron.

Example 1. Computing the homology of a tetrahedron For the tetrahedron, we will be dealing with two matrices, viz. -

$$M_1 = \begin{matrix} & \begin{matrix} ab & ac & ad & bc & bd & cd \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

$$M_2 = \begin{matrix} & \begin{matrix} abc & abd & acd & bcd \end{matrix} \\ \begin{matrix} ab \\ ac \\ ad \\ bc \\ bd \\ cd \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

On reduction, we obtain the following matrices -

$$M_1 \text{ (reduced)} = \begin{matrix} & ab & ab+ac & ac+ad & ab+ac+bc & ab+ad+bd & ac+ad+cd \\ \begin{matrix} a+b \\ b+c \\ c+d \\ d \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

$$M_2 \text{ (reduced)} = \begin{matrix} & abc & abc+abd & abc+abd+acd & abc+abd+acd+bcd \\ \begin{matrix} ab+ac+bc \\ ac+ad+bc+cd \\ bc+bd+cd \\ ad \\ bd \\ cd \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

Post-reduction, we observe the following results -

$$\text{rank} B_0 = 3, \text{rank} Z_0 = 4, \beta_0 = 4 - 3 = 1$$

$$\text{rank} B_1 = 3, \text{rank} Z_1 = 3, \beta_1 = 3 - 3 = 0$$

$$\text{rank} B_2 = 0, \text{rank} Z_2 = 1, \beta_2 = 1 - 0 = 1$$

These observations coincide with those obtained when we compute the kernel and the image of the boundary map for the simplicial complex in question by hand.

Chapter 3

An Introduction to Morse Theory and Filtrations

We've had a look at simplicial homology, and are familiar with computing homology of a simplicial complex algorithmically. The idea of this chapter is to introduce the reader to Morse theory, which will be contextual when we deal with filtrations, and subsequently progress to build up enough theory so as to be able to comprehend and compute Persistent Homology.

3.1 Morse Theory

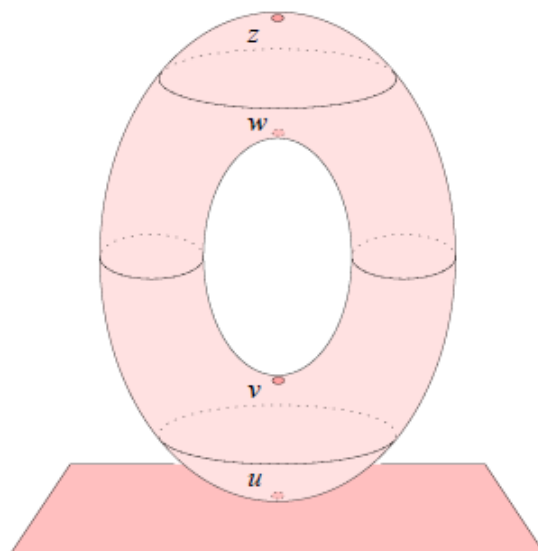


Figure 3.1: Representation of the height function applied on an upright torus with critical points u, v, w, z [EH10]

We begin with a rather lucid example from [EH10], an upright torus, say \mathbb{M} , and the vertical height function $f : \mathbb{M} \rightarrow \mathbb{R}$, wherein each real value of the height, say $h \in \mathbb{R}$ has a preimage $f^{-1}(h)$, which is called as a level set. This level set consists of all points $x \in \mathbb{M}$ at height h .

Definition 18 (Sublevel Set). A sublevel set is a set of all points in \mathbb{M} upto a certain height value. We can define it mathematically as -

$$\mathbb{M}_h = f^{-1}(-\infty, a] = \{x \in \mathbb{M} \mid f(x) \leq h\}$$

The idea is to track the evolution of the sublevel set. This can be done by increasing the value of h . Points of interest are u, v, w, z , which are precisely the critical points. For $h < f(u)$, $\mathbb{M}_h = \emptyset$. For $f(u) < h < f(v)$, $\mathbb{M}_h \cong \mathbb{D}^2$ (a disk). For $f(v) < h < f(w)$, \mathbb{M}_h will be a cylinder, and $\mathbb{M}_h \cong S^1$ (a circle). For $f(w) < h < f(z)$, \mathbb{M}_h will correspond to a capped torus. And for $f(z) < h$, we obtain the entire torus \mathbb{M} as the sublevel set \mathbb{M}_h .

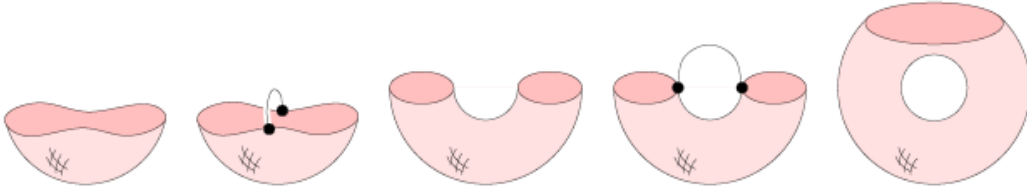


Figure 3.2: Evolution of the sublevel set [EH10]

We will move on to rigorously define a smooth function. For a d -manifold \mathbb{M} , \mathbb{M} is endowed with an atlas of coordinate charts diffeomorphic to an open ball in \mathbb{R}^d . Smooth here implies that the maps of the atlas are infinitely differentiable.

For a point $x \in \mathbb{M}$, the tangent space at x is represented as $T\mathbb{M}_x$, and is the d -dimensional vector space of all the tangent vectors of \mathbb{M} at x . For a smooth map $f : \mathbb{M} \rightarrow \mathbb{N}$, from one smooth manifold to another, f induces a linear map on the level of their respective tangent spaces, viz. the derivative - $Df_x : T\mathbb{M}_x \rightarrow T\mathbb{N}_{f(x)}$.

Definition 19 (Regular Point). Given a point $x \in \mathbb{M}$, x is a regular point of f if Df_x is surjective.

Definition 20 (Critical Point). Given a point $x \in \mathbb{M}$, x is a critical point of f if Df_x is the zero map.

Considering we are working with a local coordinate system represented as (x_1, x_2, \dots, x_d) , we can explicitly define x to be a critical point when -

$$\frac{\partial f}{\partial x_1}(x) = \frac{\partial f}{\partial x_2}(x) = \dots = \frac{\partial f}{\partial x_d}(x) = 0$$

Definition 21 (Critical Value). The image of a critical point $x \in \mathbb{M}$, $f(x)$, is called the critical value of f .

Definition 22 (Regular Value). The image of a regular point $x \in \mathbb{M}$, $f(x)$, is called the regular value of f . All values which are not critical values are regular values of f .

Definition 23 (Hessian Matrix). The Hessian of f at a point $x \in \mathbb{M}$, is defined as -

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_d}(x) \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1}(x) & \frac{\partial^2 f}{\partial x_d \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_d \partial x_d}(x) \end{bmatrix}$$

Definition 24 (Non-degeneracy). A critical point $x \in \mathbb{M}$ is said to be non-degenerate if the Hessian $H(x)$ is non-singular, or $\det H(x) \neq 0$.

Lemma 3.1.1 (Morse Lemma). For a non-degenerate critical point $c \in \mathbb{M}$ of $f : \mathbb{M} \rightarrow \mathbb{R}$, there are local coordinates $c = (0, 0, \dots, 0)$ such that $f(x) = f(c) - x_1^2 - \dots - x_k^2 + x_{k+1}^2 + \dots + x_d^2$, for all $x = (x_1, \dots, x_d)$ in a small neighborhood of c .

Definition 25 (Index). The index of a critical point $x \in \mathbb{M}$ is the number of minus signs we obtain from the Morse lemma. In the above case $\text{index}(x) = k$. For a d -dimensional manifold, we can observe $d + 1$ types of non-degenerate critical points. For example, a 2-manifold will have three types of non-degenerate critical points: minimas with index 0, saddle points with index 1, and maximas with index 2.

Aimed with the above definitions, we are ready to define a Morse function.

Definition 26 (Morse Function). A Morse function is a smooth function defined on a manifold, $f : \mathbb{M} \rightarrow \mathbb{R}$, such that it satisfies the following conditions

1. Critical points are non-degenerate
2. No two critical points have the same critical value, i.e. they have distinct function values

A brief digression with an introduction to Morse theory provides us perspective on how one can obtain relevant features by traversing a smooth topological space. We will try to look at simplicial complexes with the same lens, viz. traversing a simplicial complex so as to obtain important topological features of it by “traversing” it.

3.2 Filtrations

Definition 27 (Subdivision). Given a simplicial complex K , a simplicial complex L , is called a subdivision of K if $|L| = |K|$. Every simplex in L will be contained in some simplex in K .

Definition 28 (Barycentric Subdivision). Given a simplex $\sigma \in K$, the barycenter of σ would be the average of its vertices. A barycentric subdivision would mean that we obtain all the barycenters of all such simplices in K . Here’s a nice depiction of the same from [EH10].

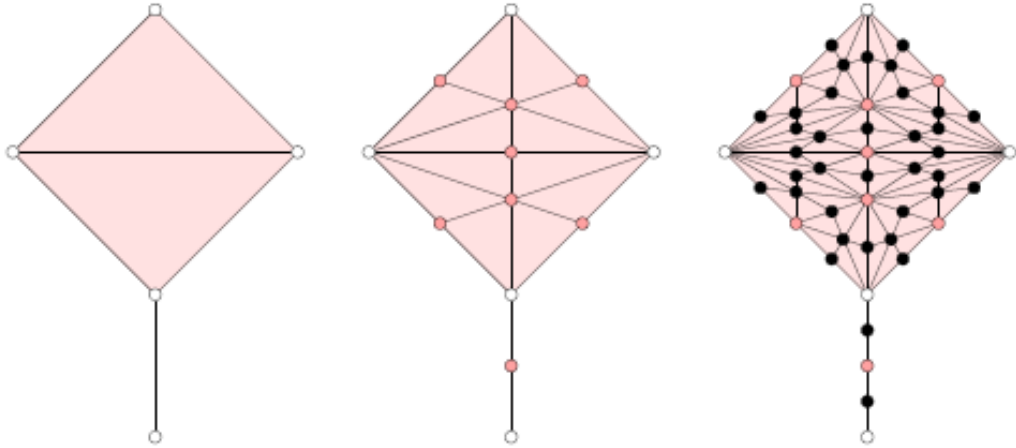


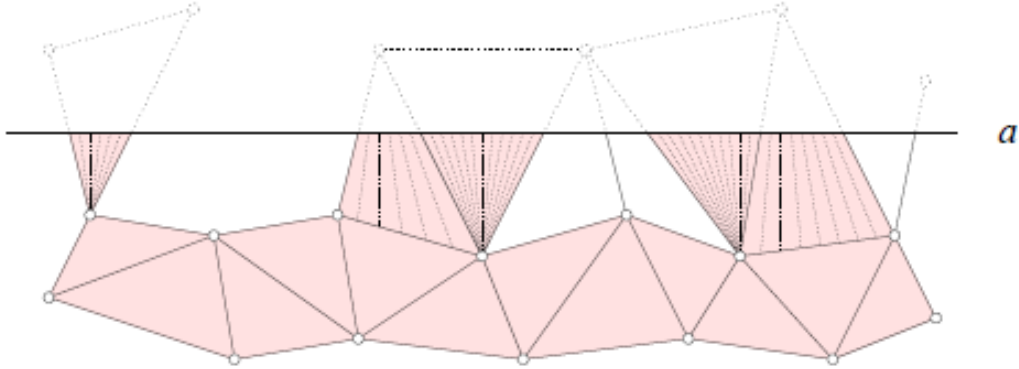
Figure 3.3: The first two barycentric subdivisions of the given simplicial complex [EH10]

Given a simplicial complex K , we can define a piecewise linear function as $f : |K| \rightarrow \mathbb{R}$, $f(x) = \sum_i b_i(x)f(v_i)$, where v_i s are the vertices of K , and $b_i(x)$ s are the barycentric coordinates of x .

The idea is to approximate smooth functions using piecewise linear functions. We assume that these piecewise linear functions have distinct function values at the vertices of the simplicial complex. This would allow us to order vertices by increasing function values in the manner: $f(v_1) < f(v_2) < \dots < f(v_n)$. Doing this allows us to define a subcomplex K_i defined by the first i vertices. This would mean that for $\sigma \in K$ belongs to the subcomplex K_i if and only if every vertex v_j of σ satisfies $j \leq i$.

Definition 29 (Lower Star). The lower star Stv_i is defined as the set of all simplices such that v_i is the vertex of each of those simplices having maximum value. It can be formally defined as: $\text{Stv}_i = \{\sigma \in \text{Stv}_i | x \in \sigma \Rightarrow f(x) \leq f(v_i)\}$.

Definition 30 (Lower Star Filtration). From the above definition, we can see that the superset of lower stars will make up the entire simplicial complex. From this, we can also observe that K_i is the union of the first i lower stars. The lower star filtration is defined as the nested sequence of subcomplexes $\phi = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K$. A visual example, given a real value a , and of the form $f(v_i) < a < f(v_{i+1})$ from [EH10] is given below -



Definition 31 (Lower Link). The lower link Lkv_i is defined as the set of all simplices such that all the vertices have a smaller functional value than v_i . It can be formally defined as: $\text{Lkv}_i = \{\sigma \in \text{Lkv}_i | x \in \sigma \Rightarrow f(x) < f(v_i)\}$.

Definition 32 (Monotonicity). A function $f : K \rightarrow \mathbb{R}$ is said to be monotonic if it is non-decreasing along increasing chains of faces, i.e. $f(\sigma) \leq f(\tau)$, where $\sigma, \tau \in K$, and σ is a face of τ .

Similar to how we obtained the subcomplexes from piecewise linear functions, we can obtain a sublevel set from a monotonic function, wherein the sublevel set $K(c)$, for a real valued c , is defined as $K(c) = f^{-1}(-\infty, c]$. And thus, we obtain a nested sequence of subcomplexes $\phi = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K$. Given an ordered set of real functional values $c_1 < c_2 < \dots < c_n$, $K_i = K(c_i) \forall i$, and also $c_0 = -\infty$. The obtained nested sequence is a filtration.

Building the above definitions provides us intuition of how one can go about partitioning a simplicial complex by constructing a filtration. This intuition is very important while defining and understanding persistent homology.

Definition 33 (Čech Complex). Given a set of points, say X in a metric space, and a real value $\varepsilon > 0$, the Čech complex, denoted as C_ε is a simplicial complex constructed in the following manner - For every subset $S \subset X$, form a $(\varepsilon/2)$ -ball around every element of S , and include S as a simplex with dimension $|S|$ if there exists a common point contained in each of the constructed balls.

Definition 34 (Vietoris-Rips Complex). The definition of the Vietoris-Rips complex builds on the Čech complex, wherein the Vietoris-Rips complex is a Čech complex, however instead of adding a simplex of dimension $|S|$ when there is a common point of intersection of all the $(\varepsilon/2)$ -balls, we add the simplex when all the balls have a pairwise intersection.

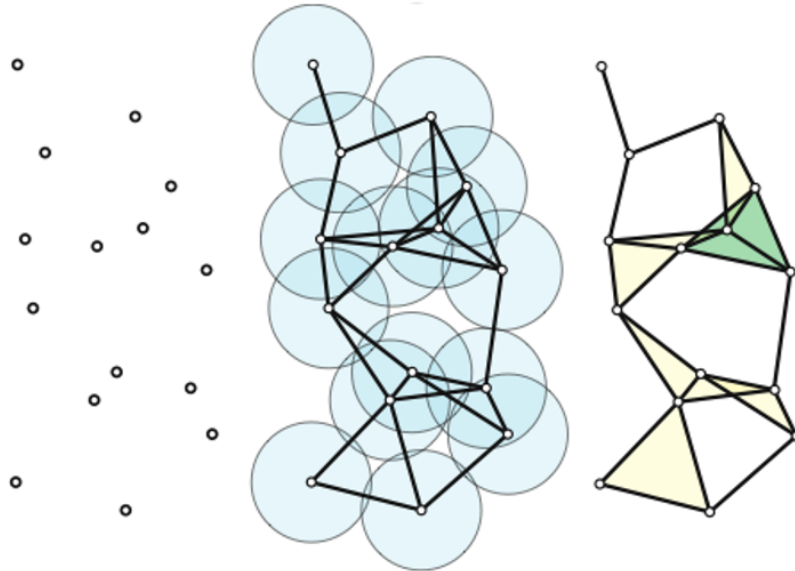


Figure 3.4: Example of a Vietoris-Rips complex [CdSEG07]

We will be working with the Vietoris-Rips complex for implementation purposes.

Chapter 4

Persistent Homology

We have seen how insightful obtaining topological characteristics of data (in our case simplicial complexes) can be while trying to study the “shape” of our dataset. Scaling and formalizing this process so as to obtain topological “hotspots” of our data is a fruitful exercise. One such method which has revolutionized this process is Persistent Homology.

4.1 Computing Persistence

For a monotonic function $f : K \rightarrow \mathbb{R}$, we obtain a filtration like shown in the previous chapter of the form $\emptyset = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K$. For any two $i \leq n, j \leq n$, and $i \leq j$, there is an inclusion map $K_i \hookrightarrow K_j$, and thereby an induced homomorphism on the level of homology groups $f_p^{i,j} : H_p(K_i) \rightarrow H_p(K_j)$.

We obtain an associated sequence of homology groups from the above filtration -

$$0 = H_p(K_0) \rightarrow H_p(K_1) \rightarrow \dots \rightarrow H_p(K_n) = H_p(K)$$

Definition 35 (Persistent Homology Group). The p -th persistent homology group is represented as $H_p^{i,j}$, defined for $0 \leq i \leq j \leq n$, and defined as $H_p^{i,j} = \text{im} f_p^{i,j}$.

Definition 36 (Persistent Betti Number). Like we defined Betti numbers for associated homology groups, we can define the p -th persistent Betti number, represented as $\beta_p^{i,j}$ as the rank of the p -th persistent homology group, or $\beta_p^{i,j} = \text{rank} H_p^{i,j}$.

The persistent homology groups can be interpreted as being made up of the homology classes of K_i which are still alive or “persist” at K_j . Formally, we can the p -th persistent

homology group as $H_p^{i,j} = Z_p(K_i)/(B_p(K_j) \cap Z_p(K_i))$.

Definition 37 (Persistence). $\gamma \in H_p(K_i)$ is said to be born in K_i if $\gamma \notin H_p^{i-1,i}$. If γ is born at K_i and dies at K_j , the persistence, $\text{pers}(\gamma)$, is defined as $\text{pers}(\gamma) = a_j - a_i$.

Definition 38 (Persistence Module). We obtain $\mathcal{P}(\mathcal{T}) : \{H_p(K_1) \xrightarrow{f_1^*} H_p(K_2) \xrightarrow{f_2^*} \dots \xrightarrow{f_{m-1}^*} H_p(K_m)\}$, which is a sequence of vector spaces connected through induced homomorphisms f_i^* s, known as the persistence module. The persistence module captures the evolution of the topology of the filtration.

Theorem 4.1.1 (Decomposition of Persistence Module, G. Carlsson & V. de Silva, 2009). *Any persistence module can be decomposed into a collection of intervals of the form $[i, j)$. And the multiset of all the intervals in this decomposition is called the persistence diagram. An interval of the form $[i, j)$ in the persistence diagram corresponds to a homological feature which appeared at i and disappeared at j .*

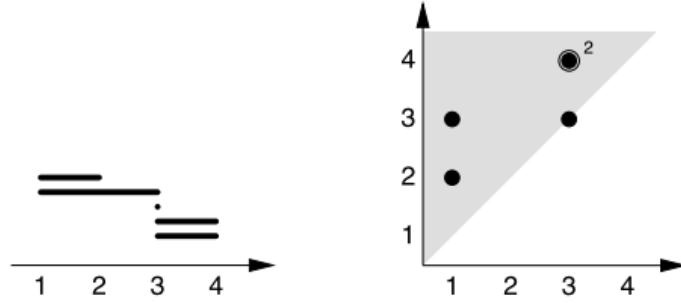


Figure 4.1: Barcode and persistence diagram representations of the persistence elements $[1, 2]$, $[1, 3]$, $[3, 3]$, $[3, 4]$, and $[3, 4]$ [GC09]

This important result from [GC09] gives us a very practical tool to track persistence, and is called a barcode, as depicted below. The barcode is precisely the multiset of intervals as mentioned in the above result.

There are multiple softwares which one can use to compute persistence: Giotto-tda (L2F, Switzerland) [TLT⁺20], Gudhi (Inria, France) [The21], PHAT (IST, Austria) [BKRW17], Ripser [Bau21], and Dionysus [Mor17] among others.

Below are some examples of computing the persistence of three objects: a circle, a sphere, and a torus.

The examples were implemented using the Giotto-tda package [TLT⁺20] on a Jupyter notebook. The package is an end-to-end tool for computing the persistence diagram, from the sampling of points to applying a filter function and so on. The idea behind this experimentation is to see if we can sample points for the required object, obtain the Vietoris-Rips complex, compute the persistence diagram, and observe the homological features, and verify if they are the same if we would compute them by hand.

4.1.1 Circle

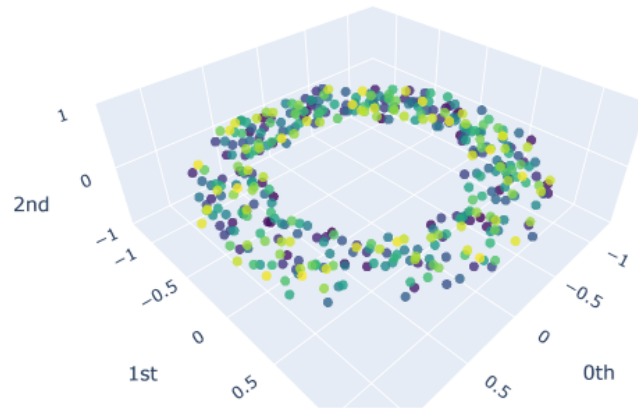


Figure 4.2: Point cloud approximation of a circle

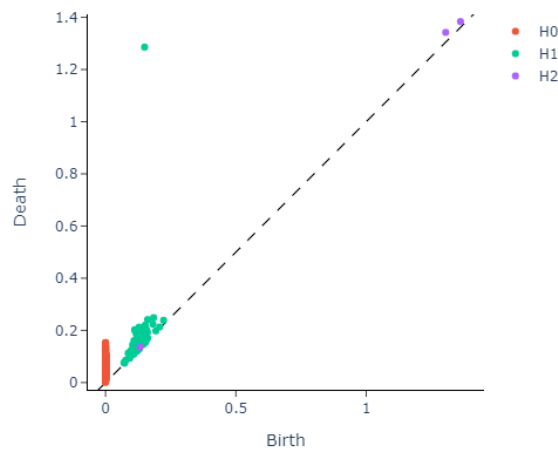


Figure 4.3: One persistent generator for H_1 corresponding to a loop

It is an easy yet important observation to make that the points lying on the dotted line of the persistence diagram are elements of the persistent homology group which are born at one instance and die at the same instance. These points need to be ignored while taking note of the homological features. Also, most points hovering above the dotted line are noise, and can also be ignored.

Taking into consideration the couple of points mentioned above, we can see that we have only one non-trivial or unignorable point, which is the one marked in green, belonging to H_1 . This is a persistent generator which corresponds to the loop of the circle.

4.1.2 Sphere

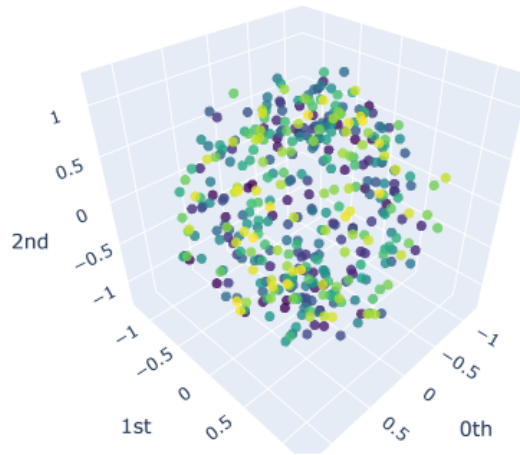


Figure 4.4: Point cloud approximation of a sphere

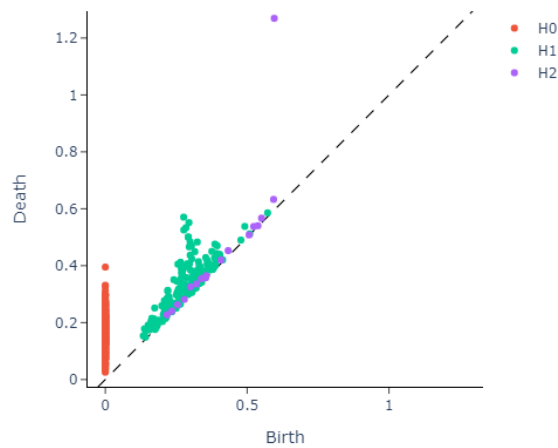


Figure 4.5: One persistent generator for H_2 corresponding to a void

In this example as well, we obtain one non-trivial or unignorable point, which is the one marked in purple, belonging to H_2 . This is a persistent generator which corresponds to the void in the sphere.

4.1.3 Torus

In this example, we obtain three non-trivial or unignorable points, two of which are marked in green, belonging to H_1 , and one in purple, belonging to H_2 . There are three persistent generators, two for H_1 and one for H_2 , as one would expect.

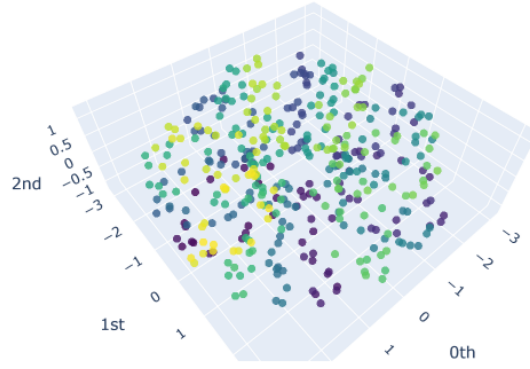


Figure 4.6: Point cloud approximation of a torus

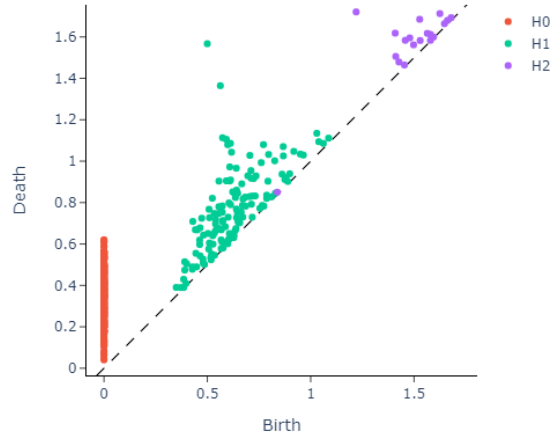


Figure 4.7: Three persistent generators, two for H_1 and one for H_2 .

4.2 Multi-dimensional Persistence

We have come across one-dimensional persistence, namely when we have a monotonic function $f : K \rightarrow \mathbb{R}$. The logical next step would be try and generalize this to multiple dimensions. However, this isn't simple at all, and multiple groups are working on laying the theoretical foundations for this generalization.

As part of this thesis, we will explore the basics of multi-dimensional persistence, and see what it really means to go from single to “multi”. For the same, let us consider a monotonic function $f : K \rightarrow \mathbb{R}^2$. Take two points $m = (m_x, m_y), n = (n_x, n_y) \in \mathbb{R}^2$. We can come up with a partial order $m \leq n$ iff. $m_x \leq n_x$ and $m_y \leq n_y$.

Definition 39 (Join). For two points $m, n \in \mathbb{R}^2$, the join, represented as $m \vee n$, is defined as $m \vee n = (\max\{m_x, n_x\}, \max\{m_y, n_y\})$.

Definition 40 (Critical Values of a Bifiltration). As we are working in the realm of finite simplicial complexes, the bifiltration f changes its value at finitely many points, and these values, represented as $Q = \{q_i \in \mathbb{R}^2\}$, are called the critical values of the bifiltration.

For a chain complex $C^* = (C_p, \partial_p)$. Given a partial order $m \leq n$, the inclusion map $C_m^* \hookrightarrow C_n^*$ induces a linear map between the corresponding homology spaces $H_p(C_m^*)$ and $H_p(C_n^*)$.

The multi-parameter persistence p -th module $H_p(C)$ of a bifiltered chain complex C is made up of all the homology spaces $H_p(C_m^*)$, where $m \in \mathbb{R}^2$.

Analogous to a filtration in the case of one-dimensional persistence, for two-dimensional persistence, we deal with bifiltrations. Below is an example of a bifiltration given in [GC09].

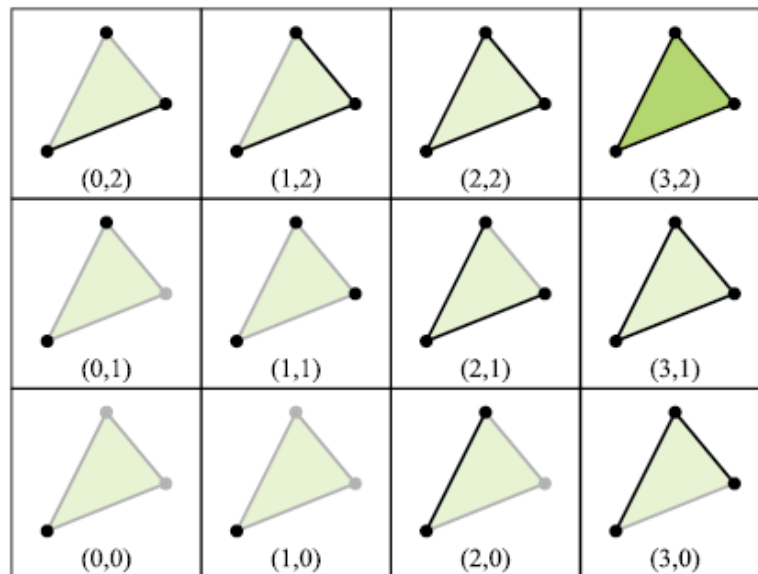


Figure 4.8: Bifiltration of a triangle [GC09]

In the next chapter, we will look at methods to speed up the computation of Persistent Homology, and also look at an ongoing work of doing the same in the case of Multi-dimensional Persistence.

Chapter 5

Collapses

As seen in the previous chapter, Persistent Homology is an important tool to analyze the shape of data, specifically while working with simplicial complexes. It is important to realize with the explosion of data, faster methods to analyze these datasets will be required. As a consequence, vital techniques like Persistent Homology will and have seen improvements in their implementation efficacy. In this chapter, we will look at two such cutting-edge methods.

There are two schools of thoughts when it comes to improving and speeding up computation of persistence, one is writing efficient software, some examples of which were provided in the previous chapter, and the other is to take an apriori approach to persistence, i.e. look at methods to simplify your dataset (simplicial complex), as a smaller or a reduced dataset will immediately imply lesser computation to be performed in comparison with the non-reduced case.

This part of the thesis will focus on the latter, the apriori approach, wherein we try to optimize persistent homology calculation by simplifying our simplicial complex. This chapter will focus on some of the novel ideas documented by Siddharth Pritam as part of his research and PhD study. His PhD focused on collapses in context with persistence theory, and algorithms which collapse various simplicial complexes efficiently enough so as to substantially speedup the computation of Persistent Homology.

The chapter will also build theory so as to give the reader a fair understanding of the work currently being done in the realm of Multi-dimensional Persistence, guided by Michael Kerber and in collaboration with Siddharth Pritam [Pri21]. The progress includes prototyping and implementation on minimalistic examples, some of the results of which are provided in the following chapter.

5.1 Prerequisites

Definition 41 (Flag Complex). A complex K is a flag complex if, when a subset of its vertices have pairwise edges between them they span a simplex. The 1-skeleton of K is denoted as G .

Definition 42 (Neighborhood of a vertex in G). For a vertex $v \in G$, the open neighborhood $N_G(v)$ is defined as $N_G(v) := \{u \in G \mid [uv] \in E\}$. The closed neighborhood $N_G[v]$ is defined as $N_G[v] := N_G(v) \cup \{v\}$.

We will be using the star and link of a simplex extensively, and hence will reiterate what they stand for at this stage of the document again for the benefit of the reader -

Definition 43 (Closed Star of a Simplex). For a simplex $\sigma \in K$, the closed star of σ , $St_K(\sigma) := \{\tau \in K \mid \tau \cup \sigma \in K\}$.

Definition 44 (Link of a Simplex). For a simplex $\sigma \in K$, the link of σ , $Lk_K(\sigma) := \{\tau \in St_K(\sigma) \mid \tau \cap \sigma = \emptyset\}$.

Definition 45 (Free Simplex). A simplex $\sigma \in K$ is called a free simplex if σ has a unique coface $\tau \in K$.

Definition 46 (Free Pair). The pair $\{\sigma, \tau\}$ obtained from the above definition of a free simplex is called a free pair.

Definition 47 (Elementary Simple Collapse). The removal of a free pair, i.e. $K \rightarrow K \setminus \{\sigma, \tau\}$ is called as an elementary simple collapse.

Definition 48 (Simple Collapse). There is said to be a simple collapse from K to its subcomplex L if there exists a series of elementary simple collapses from K to L . This is denoted as $K \searrow L$. The simple collapse preserves homotopy type of K , i.e. $K \sim L$.

Definition 49 (Simple-collapse Minimal). A complex K' is called simple-collapse minimal if there is no free pair $\{\sigma, \tau\}$ in K' .

Definition 50 (Elementary Core). For a subcomplex K^{ec} of K , K^{ec} is called as an elementary core if $K \searrow K^{ec}$, and that K^{ec} is simple-collapse minimal.

The paper by J.A. Barmak and E.G. Minian [JAB12], has proved seminal in Siddharth's work, and the following definitions are observations are from the same -

Definition 51 (Dominated Vertex). For a complex L and vertex a , the simplicial cone $aL := \{a, \tau \mid \tau \in L \text{ or } \tau = \sigma \cup a, \sigma \in L\}$. A vertex $v \in K$ is said to be dominated if $Lk_K(v) = v'L, v' \neq v, L \subseteq K$. In this case, v' is dominating v , and v is dominated by v' .

Definition 52 (Dominated vertex, J.A. Barmak & E.G. Minian). A vertex $v \in K$ is dominated by another vertex $v' \in K$, if and only if all the maximal simplices of K that contain v also contain v' . This is just a variation of the above definition, given in [JAB12]. However, we will use the one above for our purpose.

Definition 53 (Dominated Simplex). A simplex $\sigma \in K$ is known to be a dominated simplex if the link of the simplex, $Lk_K(\sigma)$ is a simplicial cone. This means that there exists a vertex $v' \notin \sigma$, and a subcomplex L of K , such that $Lk_K(\sigma) = v'L$. In this context, we claim that σ is dominated by v' , and that v' is dominating σ . This is denoted as $\sigma \prec v'$.

Definition 54 (Elementary Strong Collapse). An elementary strong collapse is defined as the deletion of a dominated vertex $v \in K$, denoted as $K \searrow \searrow K \setminus v$.

Definition 55 (Strong Collapse). There is said to be a strong collapse from K to its subcomplex L if there exists a series of elementary strong collapses from K to L . This is denoted as $K \searrow \searrow L$. If there is such a combination of strong collapses, K and L are said to have the same strong homotopy type.

We can extend the nomenclature to an edge collapse, which is essentially a 1-collapse. And similarly, a k -simplex collapse would be a k -collapse.

The below illustration of an elementary strong collapse is from [BP19].

Definition 56 (Core). The core K^c of K is a minimal subcomplex (a complex without any dominated vertices), such that $K \searrow \searrow K^c$.

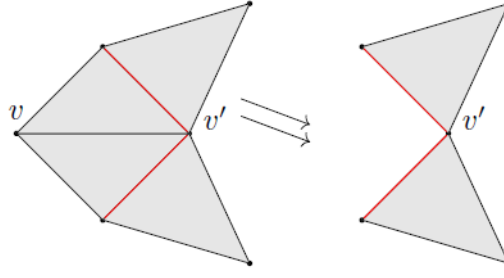


Figure 5.1: An elementary strong collapse. Here, v is dominated by v' . [BP19]

Definition 57 (Retraction Map). If $v \in K$ is dominated by $v' \in K$, the vertex map $r : K \rightarrow K \setminus v$, $r(w) = w$, $w \neq v$ and $r(v) = v'$, induces a simplicial map which is a retraction map.

Definition 58 (Contiguous Maps). Two simplicial maps $\phi : K \rightarrow L$ and $\psi : K \rightarrow L$ are said to be contiguous if for all $\sigma \in K$, $\phi(\sigma) \cup \psi(\sigma) \in L$.

Proposition 5.1.1 (J. A. Barmak & E. G. Minian, 2011). The homotopy between r and $i_{K \setminus v}$ over $K \setminus v$ is a strong deformation retract. Furthermore, $i_{K \setminus v} \circ r$ is contiguous to i_K over K .

Definition 59 (Flag tower). A sequence of flag complexes $\mathcal{T} : \{K_1 \xrightarrow{f_1} K_2 \xrightarrow{f_2} \dots \xrightarrow{f_{m-1}} K_m\}$, connected through simplicial maps is called a flag tower. If all the simplicial maps are inclusions, then the tower is called a flag filtration.

Definition 60 (Flag Filtration). If all the simplicial maps in the flag tower are inclusions, then the tower is called a flag filtration.

From [JAB12], we note the following remark -

Remark 1. For two vertices in simplicial complex K , $v, v' \in K$, v is dominated by v' if and only if all the maximal simplices of K that contain v also contain v' .

The subsequent two results are from [BP19] and [Pri21] respectively.

Lemma 5.1.2. For a simplex σ and vertex v' in simplicial complex K , $\sigma, v' \in K$, σ is dominated by v' if and only if all the maximal simplices of K that contain σ also contain v' .

Proof. Let us assume $\sigma \prec v'$. This means that $Lk_K(\sigma) = v'L$, implying that for any maximal simplex $\tau \in St_K(\sigma)$, $v' \in \tau$.

Now, let us assume all the maximal simplices of K that contain σ also contain v' . Now for any maximal simplex $\tau \in St_K(\sigma)$, $v' \in \tau$. But from $v' \notin \sigma$, $v' \in \tau \setminus \sigma$, thereby giving us $Lk_K(\sigma) = v'L$, wherein $L = (\tau \setminus \sigma) \setminus v'$. \square

Lemma 5.1.3. *For a simplex $\sigma \in K$, where K is a flag complex, σ will be dominated by a vertex $v' \in K$ if and only if $N_G[\sigma] \subseteq N_G[v']$.*

Proof. Let us assume that $\sigma \prec v'$. From Lemma 5.1.2, we can straightaway come to the fact that maximal simplices that contain σ also contain v' , leading us to $N_G[\sigma] \subseteq N_G[v']$.

Now, let us assume $N_G[\sigma] \subseteq N_G[v']$, and assume $\tau \in K$ to be a maximal simplex containing σ . For two vertices, $p \in \tau$ and $q \in \sigma$ respectively, edge $[p, q] \in \tau$. And by extension of this, we can observe that $p \in N_G[\sigma] \subseteq N_G[v']$. All vertices in τ are linked to v' by an edge, and as K is a flag complex and τ is maximal, $v' \in \tau$. This would mean that all maximal simplices containing σ will also contain v' , giving us the fact that $\sigma \prec v'$. \square

Remark 2. For a flag complex K and a subcomplex L of K obtained through an edge collapse, L will also be a flag complex.

Given two persistence modules, we have a condition for them to be equivalent.

Definition 61 (Equivalence of persistence modules). Two persistence modules $\mathbb{V} : \{V_1 \rightarrow \dots \rightarrow V_m\}$ and $\mathbb{W} : \{W_1 \rightarrow \dots \rightarrow W_m\}$ connected through homomorphisms $\phi_i : V_i \rightarrow W_i$ are said to be equivalent if the ϕ_i s are isomorphisms and the following diagram commutes -

$$\begin{array}{ccccccc} V_1 & \longrightarrow & V_2 & \longrightarrow & \dots & \longrightarrow & V_m \\ \downarrow \phi_1 & & \downarrow \phi_2 & & & & \downarrow \phi_n \\ W_1 & \longrightarrow & W_2 & \longrightarrow & \dots & \longrightarrow & W_m \end{array}$$

5.2 Computing Persistent Homology of Flag Complexes using Strong Collapses

We have looked at adequate machinery to be able to look at the algorithm in [BP19], which aims at reducing the input simplicial complex, particularly a flag complex, thereby being more efficient at computing persistent homology. However, this isn't the only task to perform. An important part of the work is to verify that the persistence module of the reduced complex is in fact equivalent to the persistence module of the original complex. We will take this up after the discussion on the algorithm.

A point to note is that the algorithm deals only with vertices and edges, which is evident from the check (*if*) condition in the algorithm. This means that for implementation purposes, we can simply work with the 1-skeleton (G) of our flag complex K . The input G is represented as an adjacency matrix M , which is a $v \times v$ matrix (where v is the total number of vertices) of 0s and 1s, encapsulating information about the edges of G . M can be formally defined as $M_{i,j} = 1$ if (i, j) is an edge in G , and $M_{i,j} = 0$ if (i, j) is not an edge in G .

Algorithm 1: Core graph algorithm [BP19]

input: the adjacency matrix M of G
output: Core graph of K
 $rowQueue \leftarrow \text{push all rows of } M \text{ (all vertices of } K)$
while $rowQueue$ is not empty **do**
 $v \leftarrow \text{pop}(rowQueue)$
 $N_G[v] \leftarrow \text{the non-zero columns of } v$
 for w in $N_G[v]$ **do**
 if $N_G[v] \subseteq N_G[w]$ **then**
 Remove from M the column and the row associated to v
 push all the entries of $N_G[v]$ to $rowQueue$ if not pushed before
 break
 end
 end
end

The algorithm can be interpreted as being made up of two important operations -

- Operation 1: Checking for domination where each row is checked against at most k other rows, where k is the maximal degree of any vertex. At most k^2 times.

- Operation 2: While loop execution. At most v^2 times, where v is the total number of vertices of the complex K .

With both of these operations put together, we can evaluate the worst-case time complexity as $O(v^2 k^2)$.

The idea of the algorithm is to simplify the simplicial complexes of the input sequence by using strong collapses, and to compute the persistent homology of an induced sequence of reduced simplicial complexes that has the same persistent homology as the initial one.

[BP19] comments how the idea of transforming a tower into an equivalent filtration can be traced back to [DFW14], and further refined in [KS17]. [BP19] notes that [KS17] provided theoretical bounds on size and time to construct equivalent filtration. Wherein they proved the size of the equivalent filtration is $O(d * n * \log n_0)$, where d is the maximal dimension of the complexes in the input tower, and n and n_0 are the total number of elementary inclusions and vertex inclusions respectively.

[BP19] builds on [DFW14] and [KS17], in the sense that they look at turning a flag tower into a flag filtration with the same Persistent Homology, however by using strong expansion, which is doing the exact opposite of a strong collapse. We will now move to the construction bit.

Definition 62 (Active Vertex). For flag complex K_i and its 1-skeleton G_i , $k_i \subseteq \mathbb{K}_i$, where \mathbb{K}_i is an augmented complex, with 1-skeleton \mathbb{G}_i . $v \in \mathbb{K}_i$ is active if it is not currently dominated.

Definition 63. $ActN_{G_i}[v]$ is the set of all active vertices in $N_{G_i}[v]$.

Definition 64. $ActN_{G_i}[v \setminus u]$ is the set of active vertices in $N_{G_i}[v]$ that are not in $N_{G_i}[u]$.

Definition 65. $\{[u, N_{G_i}[v \setminus u]]\}$ is the set of edges between u and $ActN_{G_i}[v \setminus u]$.

The goal is to construct \mathbb{G}_i , which is done in the following manner [BP19] -

1. Set $\mathbb{G}_0 = \emptyset$.
2. If $G_i \xrightarrow{\cup \sigma} G_{i+1}$ is an elementary inclusion, then $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \sigma$.

3. If $G_i \xrightarrow{\{u,v\} \mapsto u} G_{i+1}$ is an elementary contraction, then -

- (a) If $|ActN_{G_i}[v \setminus u]| \leq |ActN_{G_i}[u \setminus v]|$, then $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[u, ActN_{G_i}[v \setminus u]]\}$, and v as contracted.
- (b) Else $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[v, ActN_{G_i}[u \setminus v]]\}$, and u as contracted.

Note: $\mathbb{G}_i \subseteq \mathbb{G}_{i+1} \implies \mathbb{K}_i \subseteq \mathbb{K}_{i+1}$

Below is a beautiful depiction of the construction from [BP19].

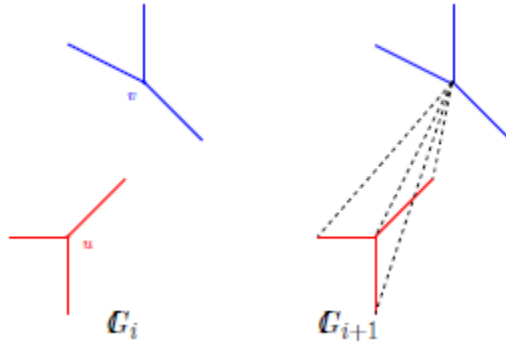


Figure 5.2: Constructing \mathbb{G}_{i+1} from \mathbb{G}_i [BP19]

There is one more important step still left: verification. We need to verify our construction, and that the Persistent Homology is the same.

Lemma 5.2.1 (J.-D. Boissonnat & S. Pritam, 2019). *Let $f_i : K_i \xrightarrow{\{u,v\} \mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathcal{T} : K_0 \xrightarrow{f_0} \dots \xrightarrow{f_{m-1}} K_m$. Then the complex K_{i+1} is a subcomplex of \mathbb{K}_{i+1} and $\mathbb{K}_{i+1} \searrow \searrow K_{i+1}$.*

Lemma 5.2.2 (J.-D. Boissonnat & S. Pritam, 2019). *Let $f_i : K_i \xrightarrow{\{u,v\} \mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathcal{T} : K_0 \xrightarrow{f_0} \dots \xrightarrow{f_{m-1}} K_m$. Then the following diagram commutes -*

$$\begin{array}{ccc} H_p(K_i) & \xrightarrow{f_i^*} & H_p(K_{i+1}) \\ & \searrow i^* & \downarrow (i')^* \\ & & H_p(\mathbb{K}_{i+1}) \end{array}$$

where $i' : K_{i+1} \hookrightarrow \mathbb{K}_{i+1}$ is the inclusion induced by the strong collapse, and i^* and $(i')^*$ are homomorphisms induced by the inclusion maps.

The proofs to the above two lemmas can be looked at here - [A.1](#). Both are from [\[BP19\]](#), and the next two results are a consequence of the same.

Lemma 5.2.3 (J.-D. Boissonnat & S. Pritam, 2019). *Given a tower $\mathcal{T} : K_0 \xrightarrow{f_0} \dots \xrightarrow{f_{m-1}} K_m$, for each $0 \leq i \leq m$, $\mathbb{K}_i \searrow \searrow K_i$.*

Theorem 5.2.4 (J.-D. Boissonnat & S. Pritam, 2019). *For the given tower \mathcal{T} and constructed filtration $\mathcal{F} : \mathbb{K}_0 \hookrightarrow \dots \hookrightarrow \mathbb{K}_m$, the following diagram commutes, and the vertical maps ϕ_i^* s are isomorphisms. Hence, by the theorem provided by Carlsson & Zomorodian, we can ascertain that both have the same persistence diagram.*

$$\begin{array}{ccccccc} H_p(\mathbb{K}_1) & \xleftarrow{*} & H_p(\mathbb{K}_2) & \xleftarrow{*} & \dots & \xleftarrow{*} & H_p(\mathbb{K}_m) \\ \downarrow \phi_1^* & & \downarrow \phi_2^* & & & & \downarrow \phi_m^* \\ H_p(K_1) & \xrightarrow{f_1^*} & H_p(K_2) & \xrightarrow{f_2^*} & \dots & \xrightarrow{f_{m-1}^*} & H_p(K_m) \end{array}$$

where ϕ_i is a strong collapse, and i^* and $*$ indicates the induced homomorphisms.

5.3 An Approach for Multi-dimensional Persistence

We will now look at an approach which will build on the foundations of [\[BP19\]](#) for the bifiltration case. We will proceed to state some definitions and make a couple of observations before providing the algorithm.

By projecting the critical values of the bifiltration onto the x and y axes, and taking the product of the projected critical values, one can obtain a grid in \mathbb{R}^2 . The lines in this grid are indexed with $0 < i \leq n$, and $0 < j \leq m$. Let E be the edge set of G , which is the 1-skeleton of K . With this, we can assign a bi-index to each edge $e \in E$. This is represented as $e_{(i,j)}$, and tells us where the edge sits in the bifiltration. We can define the following graph as a result of defining our bi-indices.

Definition 66. $G_{(i,j)} := \{e_{(i',j')} \in E \mid i' \leq i, j' \leq j\}$

Both $G_{(i,j)} \hookrightarrow G_{(i+1,j)}$ and $G_{(i,j)} \hookrightarrow G_{(i,j+1)}$ are assumed to be an elementary inclusion of a single edge, as per [\[Pri21\]](#).

The reduction step in the algorithm pertains to if an edge is removable or not. The edges which are non-removable are stored and are given as output, and are called as critical edges,

represented by E^c . An edge is removable if it is dominated. An easy observation, but of note is to realize that removability is a “status” of an edge, i.e. there is a chance that a removable edge becomes non-removable on further traversal of the grid. This is why the algorithm has a component which moves in the reverse bifiltration order.

Algorithm 2: Algorithm-R: Core flag bifiltration algorithm [Pri21]

input: set of edges E sorted by bi-filtration value

output: critical edges E^c

set $E^c = \emptyset$

set $c_i = 1, c_j = 1$

while $c_i \leq n, c_j \leq m$ **do**

for $i = c_i, i \leq n, i++$ **do**

if $e_{(c_i, j)}$ is non-dominated in $G_{(i, c_j)}$ **then**
 | set-critical-recur((i, c_j))

end

end

for $j = c_j, j \leq m, j++$ **do**

if $e_{(i, c_j)}$ is non-dominated in $G_{(c_i, j)}$ **then**
 | set-critical-recur((c_i, j))

end

end

end

Algorithm 3: Set-critical-recur (Algorithm-R) [Pri21]

input: index (i, j)

set $G = G_{(i, j)}$

$E^c = E^c \cup e_{(i, j)}$

$E^{nbd} = EN_G(e_{(i, j)})$

while $c_i \leq n, c_j \leq m$ **do**

for (x, y) in E^{nbd} **do**

if $e_{(x, y)} \notin E^c$ and non-dominated in G **then**
 | set-critical-recur((x, y))

end

else if $e_{(x, y)} \notin E^c$ and dominated in G **then**
 | $G = G \setminus e_{(x, y)}$

end

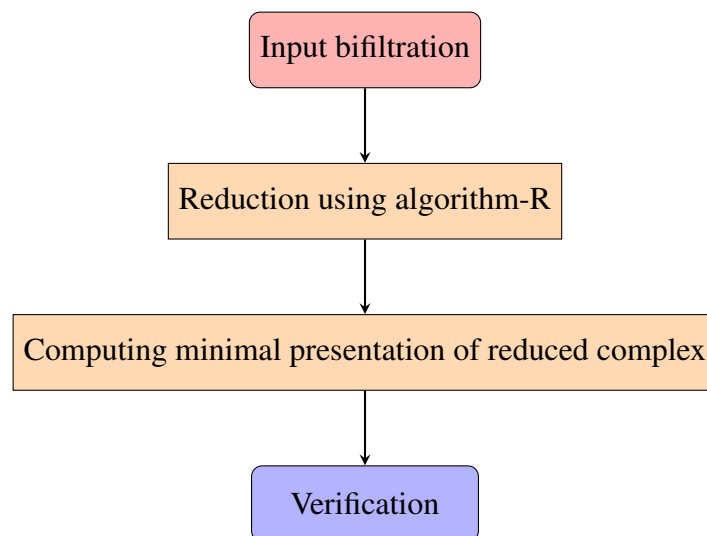
end

end

Chapter 6

Computational Experiments

This chapter will expand on the implementation aspect of the second approach explored in the previous chapter, viz. the approach for Multi-dimensional Persistence. The algorithm has been prototyped in Python, and tested on some examples. The flow of implementation work and a brief about each component of the process will be explained subsequently.



6.1 Input Bifiltration

We took three approaches to come up with test examples -

1. Examples by hand: This was the most minimalistic input dataset wherein we designed flag complexes with anywhere from 4 to 15 vertices by hand, and came up with appropriate bifiltrations.
2. Adding edges randomly: More vertices and edges were incorporated by writing a Python script which would give us a bifiltration on inputting the required number of vertices, number of edges, and the bi-indices. The script would proceed to add the required number of edges in a random manner with a pre-set probability. This was not the ideal condition as diversity couldn't be incorporated unless a more rigorous probabilistic sampling of edges was done.
3. function-Rips: This was the most realistic example, generated from a Python script written by Alexander Rolle [Rol20a].

6.1.1 Examples by Hand

Visual representations of some of the smaller examples are shown below -

Note: The bi-indices in the below example are $(0,0)$ (bottom left), $(0,1)$ (top left), $(1,0)$ (bottom right), and $(1,1)$ (top right).

For implementation purposes, we represent elements of the bifiltration as *edge ; bi-index*. For example, $1\ 2 ; 0\ 1$. The interpretation of which is that the edge $(1,2)$ comes in at the bi-index $(0,1)$.

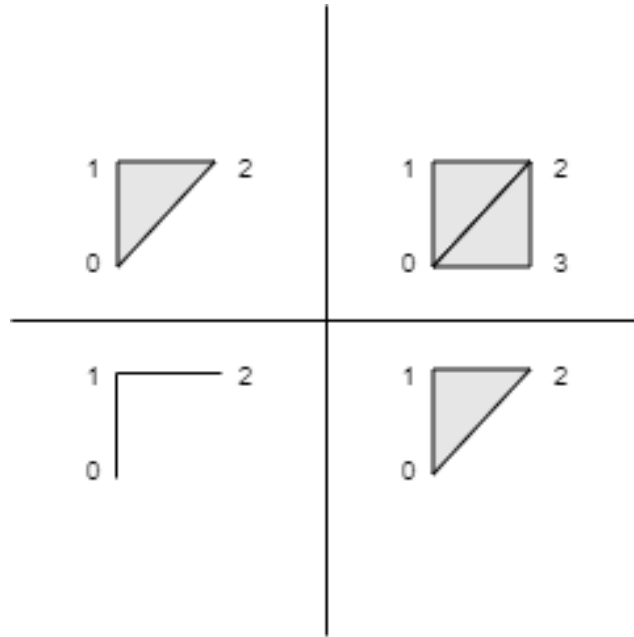


Figure 6.1: Example 1

Example 1 before reduction

Edge	Bi-index
0 1	0 0
1 2	0 0
0 2	0 1
0 3	1 1
2 3	1 1

Example 1 after reduction

Edge	Bi-index
0 1	0 0
1 2	0 0

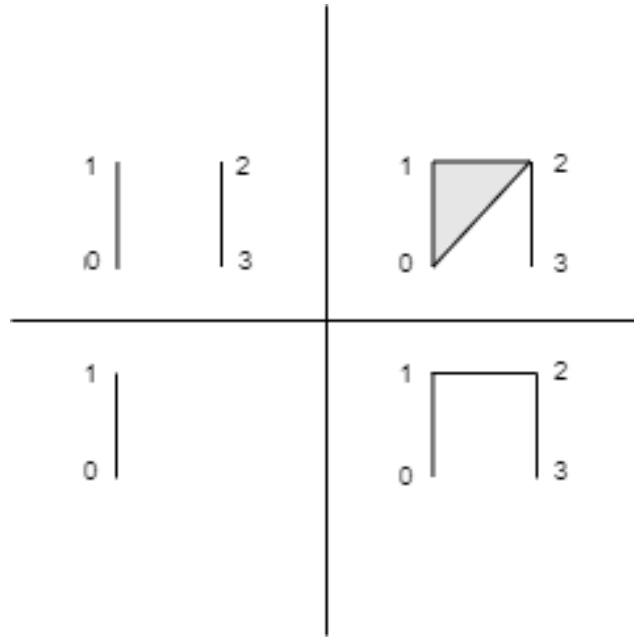


Figure 6.2: Example 2

Example 2 before reduction

Edge	Bi-index
0 1	0 0
2 3	0 1
1 2	1 0
0 2	1 1

Example 2 after reduction

Edge	Bi-index
0 1	0 0
2 3	0 1
1 2	1 0

6.1.2 Adding Edges Randomly

The program takes as input the number of vertices, number of edges, and (m, n) (to generate grid of bi-indices). We start with the first vertex, and keep adding edges randomly (with a pre-set probability) at random bi-indices until we get the required number of edges. This however, isn't realistic enough as it relies heavily on the probabilistic sampling.

6.1.3 function-Rips

function-Rips is a program [Rol20a] which generates a Vietoris-Rips complex with the required number of vertices (taken as input). Other parameters which can be tweaked in the program are the ambient dimension of the generated point cloud, the homology dimension, and bandwidth for kernel density estimate. Data generated from this program is definitely the closest to that of any real-world example, and was used in most of the testing runs.

6.2 Reduction using Algorithm-R

Data generated using methods from the previous section were used as input for testing the prototype (of algorithm-R). The implementation (code) can be observed here - [B](#).

6.3 Computing Minimal Presentation of Reduced Complex

An important component of the process is the verification task. Once we have reduced our input complex, how do we know that the Persistent Homology of the input complex is the same as that of the one obtained post-reduction?

For this reason, we used the mpfree (Minimal presentation of a free implicit representation) software [Rol20b], which is based on the paper by Michael Kerber and Alexander Rolle [KR20].

There is one additional step which needs to be followed before calling the mpfree program. The input format for the mpfree program is the *.firep* format, which was instituted to work with the RIVET software [The20]. This input format requires the user to list all the vertices, the edges, and triangles. As a result, a Python script was written to convert our complex to the required format.

firep	
x-coordinate	Number of vertices
y-coordinate	Number of edges
9880 780 40	Number of triangles
0 2 ; 0 1 2	
0 4 ; 0 3 4	
0 5 ; 1 4 5	
0 5 ; 2 3 5	
0 7 ; 3 6 7	
0 8 ; 2 7 8	
0 8 ; 5 6 8	
0 9 ; 1 8 9	
0 9 ; 4 6 9	
0 9 ; 0 7 9	
0 11 ; 6 10 11	
0 12 ; 8 10 12	

Figure 6.3: FIRep file format

On calling the `mpfree` program with the input, we obtain an output which provides the required homological information. This information can be assessed row by row, or simply noting the row and the column values will provide us a quantitative idea about the homology.

6.4 Verification

We continue immediately from where we left off in the previous subsection, namely when we obtain the row and column information of the minimal presentation. The idea here is to compare the minimal presentation of the input complex with respect to that of the reduced complex. A sanity check would be to first compare the number of rows and number of columns in the two minimal presentations. If they are not equal, they do not match. And if they do, one should ideally check for the homological components in each file. If they come from the same edge/triangle in both the cases, then the reduction is a valid one.

```

abhiitbhal@DESKTOP-L0H3RV8:/mnt/c/users/gl502vt/desktop/mpfree$ ./mpfree -v 40.firep
Execution parralized, max Number of threads: 8
Loading data into string...Read 193181 characters...done
t,s,r=9880 780 40
Found 10 different x-values and 81 different y-values
N is 10660
Chunk preprocessing...Num entries at start chunk: 29640
Local reduction
Num entries after local reduce: 34145
Sparsification
Build up smaller matrices
After chunk reduction, matrix has 8626 columns and 171 rows
N' is 8797
Num entries after chunk: 74052
done
Min Gens...done, size is 171x183
Ker basis...done, size is 171x132
Reparameterize...done
Resulting semi-minimal presentation has 183 columns and 132 rows
Minimize...Found 9 different x-values and 41 different y-values
N is 144
done
Resulting minimal presentation has 144 columns and 93 rows
Overall timer: 0.0004022
IO timer: 0.0213357 ( 31.0337% )
Chunk timer: 0.0201963 ( 29.0725% )
Mingens timer: 0.0127521 ( 18.1955% )
Kerbasis timer: 0.0011259 ( 1.59283% )
Reparam timer: 0.0052134 ( 7.31245% )
Minimize timer: 0.004086 ( 5.68347% )

```

Figure 6.4: Computing the minimal presentation of a test case

6.5 Results

All the test cases generated from the three methods were reduced using the prototype, and minimal presentation for each was checked, and they were subsequently verified. We noted stability of the implementation up until the ballpark of 80 vertices. Below are some of the results obtained -

File name	Number of vertices	Number of edges	Number of triangles	Input file size	Number of vertices post-collapse	Number of edges post-collapse	Number of triangles post-collapse	Output file size
15.firep	15	105	455	9KB	15	39	36	1KB
20.firep	20	190	1140	20KB	20	140	464	9KB
25.firep	25	300	2300	44KB	25	65	61	2KB
40.firep	40	780	9880	189KB	40	365	1484	31KB
75.firep	75	2775	67525	1464KB	75	631	1102	29KB

The above results have been verified using the mpfree software. As we can observe, there is a sizeable reduction in most cases, with a substantial reduction in the data required to be computed to process Persistent Homology. Here lies the effectiveness of Algorithm-R, and it's universality in the approach, which works well with in a diverse set of situations.

Chapter 7

Concluding Remarks

We started with the basics of Homology Theory, and built our way to understanding and computing Persistent Homology. The examples showed us the highly intuitive aspect of Persistent Homology, which has to do with how the method is efficient at coming up with shape (homological) descriptions. We subsequently looked at methods inspired by collapses to better the computational performance of Persistent Homology.

There is quite a bit of work still required to be done to foolproof the approach to multi-dimensional persistence (Algorithm-R). Firstly, the prototype needs to be more stable, implying that it should be able to work with more extensive datasets. Secondly, the correctness of the reduction needs to be proved like in the one-dimensional case. This would require the exploration of more mathematical techniques, especially at the end of proving the result.

Through all of this, one can learn to appreciate the intertwined-ness of mathematics and computation. Persistent Homology is the perfect example of how rigorous mathematics aided with effective computational techniques can accentuate progress and give rise to newer realms of performance and insight. It is exactly this interdisciplinary symbiosis which makes us strive to connect the dots and contribute to a rich academic ecosystem.

Appendix A

Collapses

A.1 Computing Persistent Homology of Flag Complexes using Strong Collapses

The following proofs are from [BP19].

Lemma A.1.1 (J.-D. Boissonnat & S. Pritam, 2019). *Let $f_i : K_i \xrightarrow{\{u,v\} \mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathcal{T} : K_0 \xrightarrow{f_0} \dots \xrightarrow{f_{m-1}} K_m$. Then the complex K_{i+1} is a subcomplex of \mathbb{K}_{i+1} and $\mathbb{K}_{i+1} \searrow \swarrow K_{i+1}$.*

Proof. Proving $\mathbb{K}_{i+1} \searrow \swarrow K_{i+1}$ would imply that $K_{i+1} \subseteq \mathbb{K}_{i+1} 1$. Since f_i is the first contraction, $\mathbb{K}_i = K_i$ and $\mathbb{G}_i = G_i$.

Let $\mathbb{G}_{i+1} := G_i \cup \{[u, \text{Act}N_{G_i}[v \setminus u]]\}$ be the graph defined in the construction. By construction, contracting v to u in both graphs yields the same graph G_{i+1} .

Let $x' \in \text{Act}N_{G_i}[v \setminus u]$. Adding the edge $[ux']$ to G_i does not change the fact that all $x \in \{N_{G_i}[v \setminus u] \setminus \text{Act}N_{G_i}[v \setminus u]\}$ are dominated since the addition of $[ux']$ only adds neighbors to $N_{G_i}[x']$ and $N_{G_i}[u]$.

Removing all the dominated vertices in $N_{G_i}[v \setminus u]$ will give us a sequence of elementary strong collapses. This will add edges between u and the non-dominated vertices that are in $N_{G_i}[v \setminus u]$, implying that v is dominated by u in K_{i+1}^0 . This implies $K_{i+1}^0 \searrow \swarrow K_{i+1}$, and therefore $\mathbb{K}_{i+1} \searrow \swarrow K_{i+1}$. \square

Lemma A.1.2 (J.-D. Boissonnat & S. Pritam, 2019). *Let $f_i : K_i \xrightarrow{\{u,v\} \mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathcal{T} : K_0 \xrightarrow{f_0} \dots \xrightarrow{f_{m-1}} K_m$. Then the following diagram commutes -*

$$\begin{array}{ccc} H_p(K_i) & \xrightarrow{f_i^*} & H_p(K_{i+1}) \\ & \searrow i^* & \downarrow (i')^* \\ & & H_p(\mathbb{K}_{i+1}) \end{array}$$

where $i' : K_{i+1} \hookrightarrow \mathbb{K}_{i+1}$ is the inclusion induced by the strong collapse, and i^* and $(i')^*$ are homomorphisms induced by the inclusion maps.

Proof. We can rightaway observe the inclusion $\mathbb{K}_i = K_i \subseteq \mathbb{K}_{i+1}$ from f_i being the first contraction.

The authors provide the following diagram, with $i' = i_1 \circ i_0$, where i_0 and i_1 are inclusions induced by the respective strong collapses shown in the diagram -

$$\begin{array}{ccc} K_i & \xrightarrow{f_i} & K_{i+1} \\ \downarrow i & & \downarrow i_0 \\ \mathbb{K}_{i+1} & \xleftarrow{i_1} & K_{i+1}^0 \end{array}$$

Now the claim made from the above is that $i' \circ f_i$ and i are contiguous. This is looked using two cases. Given $\sigma \in K_i$, $i(\sigma) = \sigma$ by definition. Hence -

Case 1: If $v \notin \sigma$, then $i' \circ f_i(\sigma) = \sigma = i(\sigma)$. Hence, $i' \circ f_i$ and i are contiguous.

Case 2: If $v \in \sigma$, $f_i(\sigma)$ is some simplex, say $\gamma \in K_{i+1}$ containing u . Therefore, $i_0 \circ f_i(\sigma) = \gamma$. When looking at the retraction $\mathbb{K}_{i+1} \searrow \searrow K_{i+1}^0$, v isn't contracted, and hence $r_1 \circ i(\sigma)$ is some simplex, say $\gamma' \in K_{i+1}^0$ containing v .

From the previous Lemma, we realize that u dominates v in K_{i+1}^0 . This means that all maximal simplices in K_{i+1}^0 which contain v will also contain u . And thereby γ' is a face of some maximal simplex $\tau \in K_{i+1}^0$ which contains u .

But we've obtained γ from contracting v to u , and thus γ should be a face of τ containing u and v . Therefore, $\gamma' \cup \gamma \subseteq \tau$, and thus we can see that $r_1 \circ i(\sigma)$ and $i_0 \circ f_i(\sigma)$ are contiguous. As i_1 is an inclusion, $i_1 \circ r_1 \circ i(\sigma)$ and $i_1 \circ i_0 \circ f_i(\sigma)$ are also contiguous. And now that $\mathbb{K}_{i+1} \searrow \searrow K_{i+1}^0$, we can see that $i_1 \circ r_1$ and $1_{\mathbb{K}_{i+1}}$ (identity map) are contiguous. And from

the diagram, $i_1 \circ i_0 = i'$, and thereby, we can say that $i' \circ f_i(\sigma)$ and $i(\sigma)$ are contiguous. And because we can observe this for any simplex $\sigma \in K_i$, we can say that $i' \circ f_i \sim i$. Now that contiguous maps are homotopic (at the level of geometric realization), the given diagram in the Lemma commutes.

□

Appendix B

Computational Experiments

Below is the Python implementation of Algorithm-R.

```
1 import networkx as nx # Package used to help us deal with large
2                       # graphs efficiently
3
4 critical_edges = [] # What we require, outputted as a list
5                   # which is converted into a filtration and
6                   # saved in a .txt file at the end
7
8 with open('input.txt', 'r') as file: # Input filtration
9 # inputted as a .txt file | Please change the input file name
10 # accordingly
11     data = file.read().splitlines()
12
13 bf_data = [] # List initialized to convert the string elements
14 # in the input file to integer values
15
16 for elem in data:
17     bf_data.append(elem.split(' ; '))
18
19 last_entry = list(map(int, bf_data[len(bf_data)-1][1].split()))
20 # A two-element list which contains the last bigrade
21 m = last_entry[0]
22 n = last_entry[1]
23
24 """
25 core():
26     - Or the main() function of the program
27     - Is a replica of the core() as given in algorithm-R
28 """
29
30 def core():
31     c_i, c_j = 0, 0 # Same as the c_i and c_j
32     # given in algorithm-R
33     edge_set = []
```

```

34     while c_i <= n and c_j <= m:
35         for i in range(c_i, m+1, 1):
36             check = []
37             edge_set = []
38             for elem in bf_data:
39                 elem_int_grade = list(map(int, elem[1].split()))
40                 if elem_int_grade[0] <= i and
41                     elem_int_grade[1] <= c_j:
42                     edge_set.append(elem)
43             for elem in edge_set:
44                 if elem in check:
45                     continue
46                 elif elem not in critical_edges:
47                     check.append(elem)
48                     if check_non_dominated(elem, edge_set):
49                         set_critical_recur(elem, edge_set, 0)
50         for j in range(c_j, n+1, 1):
51             check = []
52             edge_set = []
53             for elem in bf_data:
54                 elem_int_grade = list(map(int, elem[1].split()))
55                 if elem_int_grade[0] <= c_i and
56                     elem_int_grade[1] <= j:
57                     edge_set.append(elem)
58             for elem in edge_set:
59                 if elem in check:
60                     continue
61                 elif elem not in critical_edges:
62                     check.append(elem)
63                     if check_non_dominated(elem, edge_set):
64                         set_critical_recur(elem, edge_set, 0)
65         c_i += 1
66         c_j += 1
67
68
69 """
70 check_non_dominated(edge, check_dom_set):
71     - Function for checking whether an edge is dominated or not
72       in a specific set of edges
73     - Takes the edge in question and the set of edges it needs to
74       check whether it is dominated in as input
75     - Note: the set of edges (check_dom_set) is constructed
76       with respect to the current
77       bifiltration value - (i, c_j) or (c_i, j)
78 """
79
80 def check_non_dominated(edge, check_dom_set):
81     # To check whether an edge is dominated or not
82     print("Checking non-dominated for -", edge)
83     print("Check_dom_set =", check_dom_set)
84     edge_set = []

```

```

85     edge_elem = list(map(int, edge[0].split()))
86     for elem in check_dom_set:
87         elem_edge = list(map(int, elem[0].split()))
88         edge_set.append(elem_edge)
89     G = nx.Graph(edge_set) # networkx graph is
90     # initialized to compute neighbors efficiently
91     neighbors_x = list(G.neighbors(edge_elem[0]))
92     neighbors_x.append(edge_elem[0])
93     neighbors_y = list(G.neighbors(edge_elem[1]))
94     neighbors_y.append(edge_elem[1])
95     neighborhood_edge = [elem for elem in
96                          neighbors_x if elem in neighbors_y]
97     # Closed neighborhood computation
98     neighborhood_edge.sort()
99     for node in list(G.nodes()):
100         if node == edge_elem[0] or node == edge_elem[1]:
101             continue
102         else:
103             neighborhood_node = list(G.neighbors(node))
104             neighborhood_node.append(node)
105             neighborhood_node.sort()
106             if set(neighborhood_edge).issubset(neighborhood_node):
107                 return False
108     return True
109
110 """
111 set_critical_recur(edge, check_dom_set, flag):
112     - Function is called once an edge is observed
113       to not be dominated in a set of edges
114     - Takes the edge in question, the set of edges
115       which were used to check for domination, and
116       a flag as input
117     - The flag is for distinguishing if the function
118       is called after the domination check or
119       whether it is a recursive call
120     - Flag as '0' implies that the function is called
121       after the edge is observed to not be dominated
122     - Flag as '1' implies that the function is called
123       within itself, viz. a recursive call
124 """
125
126 def set_critical_recur(edge, check_dom_set, flag):
127     print("Set critical recur for -", edge)
128
129     if edge not in critical_edges:
130         critical_edges.append(edge)
131     print("Critical edges -", critical_edges)
132
133     edge_elem = list(map(int, edge[0].split()))
134     filt = list(map(int, edge[1].split()))
135     # The bifiltration value of the

```

```

136 # edge in question stored as a list
137
138 edge_set = []
139 G_with_filt = [] # This is G_(i, j) in the document
140
141 """
142 For a recursive call, we construct the new graph
143 G_i_j as required (below)
144 """
145 if flag == 1: # Function call originates from a
146 # recursive call
147     for i in range(filt[0]+1):
148         for j in range(filt[1]+1):
149             for element in bf_data:
150                 element_grade = list(map(int, element[1].split()))
151                 if element_grade[0] <= i and
152                 element_grade[1] <= j:
153                     to_add = []
154                     to_add = list(map(int, element[0].split()))
155                     edge_set.append(to_add)
156                     G_with_filt.append(element)
157
158 """
159 If the function call is not recursive, we can simply
160 use the set of edges under consideration for the
161 domination check to construct the graph (below)
162 """
163 if flag == 0: # Function call originates post-domination check
164     G_with_filt = check_dom_set
165     for elem in check_dom_set:
166         elem_edge = list(map(int, elem[0].split()))
167         edge_set.append(elem_edge)
168
169 G = nx.Graph(edge_set) # networkx graph is initialized
170 # to compute neighbors efficiently
171 neighbors_x = list(G.neighbors(edge_elem[0]))
172 neighbors_x.append(edge_elem[0])
173 neighbors_y = list(G.neighbors(edge_elem[1]))
174 neighbors_y.append(edge_elem[1])
175 neighborhood_edge = [elem for elem in neighbors_x if
176                      elem in neighbors_y]
177 # Closed neighborhood computation
178 neighborhood_edge.sort()
179
180 edge_neighborhood = []
181 for elem in G_with_filt:
182     #print(elem, edge_elem)
183     elem_check = list(map(int, elem[0].split()))
184     if (elem_check[0] == edge_elem[0] and
185         elem_check[1] in neighborhood_edge)
186     or (elem_check[0] == edge_elem[1] and

```

```

187         elem_check[1] in neighborhood_edge)
188     or (elem_check[1] == edge_elem[0] and
189         elem_check[0] in neighborhood_edge)
190     or (elem_check[1] == edge_elem[1] and
191         elem_check[0] in neighborhood_edge):
192         # Condition for element being in edge neighborhood
193         if elem not in edge_neighborhood:
194             edge_neighborhood.append(elem)
195
196     temp_nbd = [] # A temp variable which will be used for
197     # sorting purposes
198
199     for a in range(len(edge_neighborhood)): # Sorting the edge
200     # neighborhood with respect to bifiltration value
201         grade_a = list(map(int, edge_neighborhood[a][1].split()))
202         for b in range(a+1, len(edge_neighborhood)):
203             grade_b = list(map(int, edge_neighborhood[b][1].split()))
204             if grade_b < grade_a:
205                 tmp_nbd = edge_neighborhood[a]
206                 edge_neighborhood[a] = edge_neighborhood[b]
207                 edge_neighborhood[b] = tmp_nbd
208
209     print("Edge_nbd -", edge_neighborhood)
210
211     for i in range(len(edge_neighborhood)-1, -1, -1): # Traversing
212     # from the last to the first element of the edge neighborhood
213     # with respect to partial order of bifiltration value
214         if edge_neighborhood[i] not in critical_edges and
215         check_non_dominated(edge_neighborhood[i], G_with_filt)
216         == True:
217             set_critical_recur(edge_neighborhood[i], G_with_filt, 1)
218         elif edge_neighborhood[i] not in critical_edges and
219         check_non_dominated(edge_neighborhood[i], G_with_filt)
220         == False:
221             print("This is being removed -", edge_neighborhood[i])
222             if edge_neighborhood[i] not in G_with_filt:
223                 continue
224             else:
225                 G_with_filt.remove(edge_neighborhood[i])
226
227
228     core() # Calling core() function which starts the required
229     # computation
230
231     #print(critical_edges)
232     """
233     The next block (for loop) is used for sorting the output in
234     increasing (partial order) with respect to bifiltration value
235     """
236     for a in range(len(critical_edges)): # Sorting the edge
237     # neighborhood with respect to bifiltration value

```



```

238     grade_a = list(map(int, critical_edges[a][1].split()))
239     for b in range(a+1, len(critical_edges)):
240         grade_b = list(map(int, critical_edges[b][1].split()))
241         if grade_b < grade_a:
242             tmp_nbd = critical_edges[a]
243             critical_edges[a] = critical_edges[b]
244             critical_edges[b] = tmp_nbd
245
246 output_file = open('output.txt', 'w') # Output filtration
247 # outputted as a .txt file | Please change the output file
248 # name accordingly
249 for edge in critical_edges:
250     to_write = edge[0] + " ; " + edge[1]
251     output_file.write(to_write)
252     output_file.write("\n")
253 output_file.close()

```

Bibliography

- [Bau21] Ulrich Bauer, *Ripser: efficient computation of vietoris-rips persistence barcodes*, February 2021, Preprint.
- [BKRW17] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner, *Phat – persistent homology algorithms toolbox*, Journal of symbolic computation **78** (2017), 76 (English), Special Issue on: Algorithms and Software for Computational Topology.
- [BP19] Jean-Daniel Boissonnat and Siddharth Pritam, *Computing Persistent Homology of Flag Complexes via Strong Collapses*, SoCG 2019 - International Symposium on Computational geometry (Portland, United States), April 2019.
- [CdSEG07] Erin W. Chambers, Vin de Silva, Jeff Erickson, and Robert Ghrist, *Rips complexes of planar point sets*, 2007.
- [DFW14] Tamal K. Dey, Fengtao Fan, and Yusu Wang, *Computing topological persistence for simplicial maps*, 2014.
- [EH10] H. Edelsbrunner and J. Harer, *Computational topology: An introduction*, Applied Mathematics, American Mathematical Society, 2010.
- [GC09] Afra Zomorodian Gunnar Carlsson, *The theory of multidimensional persistence*, Discrete & Computational Geometry (2009).
- [JAB12] Elias Gabriel Minian Jonathan Ariel Barmak, *Strong homotopy types, nerves and collapses*, Discrete & Computational Geometry (2012).
- [KR20] Michael Kerber and Alexander Rolle, *Fast minimal presentations of bi-graded persistence modules*, 2020.
- [KS17] Michael Kerber and Hannah Schreiber, *Barcodes of towers and a streaming algorithm for persistent homology*, 2017.
- [Mor17] Dimitry Morozov, *Dionysus*, 2017, Preprint.

- [Pri21] Siddharth Pritam, "*multi-parameter persistence and edge collapse*", 2021.
- [Rol20a] Alexander Rolle, "*function-rips*", June 2020.
- [Rol20b] Michael Kerber & Alexander Rolle, "*mpfree*", 2020.
- [Sid20] Siddharth Pritam, *Collapses and persistent homology. (effondrements et homologie persistante)*., Ph.D. thesis, 2020.
- [The20] The RIVET Developers, *Rivet*, 2020.
- [The21] The GUDHI Project, *GUDHI user and reference manual*, 3.4.1 ed., GUDHI Editorial Board, 2021.
- [TLT⁺20] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess, *giotto-tda: A topological data analysis toolkit for machine learning and data exploration*, 2020.