Collapses and persistent homology

Siddharth Pritam

To cite this version:

Siddharth Pritam. Collapses and persistent homology. Computational Geometry [cs.CG]. Université Côte d'Azur, 2020. English. NNT : 2020COAZ4033 . tel-02962587v2

HAL Id: tel-02962587
https://inria.hal.science/tel-02962587v2

Submitted on 8 Feb 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ÉCOLE DOCTORALE

SCIENCES ET
TECHNOLOGIES DE
L'INFORMATION ET DE
LA COMMUNICATION

THÈSE DE DOCTORAT

Effondrements et homologie persistante

Siddharth Pritam

Inria Sophia Antipolis - Méditerranée, Équipe-Projet DataShape

Présentée en vue de l'obtention
du grade de docteur en Sciences
d'Université Côte d'Azur
Mention : Informatique

Dirigée par Jean-Daniel Boissonnat

Soutenue le : 18/06/2020

Devant le jury, composé de :
Dominique ATTALI, Directrice de recherche, CNRS, France, (Rapporteuse).
Jean-Daniel BOISSONNAT Directeur de recherche INRIA, (Directeur de thèse).
Frédéric CAZALS, Directeur de recherche Inria, France, (Président).
Tamal DEY, Professeur, The Ohio State University, USA, (Rapporteur).
Marc GLISSE, Chercheur, INRIA Saclay, France (Invité).
Kathryn HESS, Professeure, EPFL Lausanne.
Michael KERBER, Professeur, TU Graz, Autriche, (Rapporteur).
André LIEUTIER, Ingénieur, Dassault Systèmes, Aix en Provence, France.

UNIVERSITÉ CÔTE D'AZUR

DOCTORAL THESIS

Collapses and Persistent Homology

Author:
Siddharth Pritam

Supervisor:
Jean-Daniel Boissonnat

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

DataShape
Inria

July 1, 2020

UNIVERSITÉ CÔTE D'AZUR

Résumé

ED-Stic
Inria

Doctor of Philosophy

Collapses and Persistent Homology

by Siddharth Pritam

Dans cette thèse, nous introduisons deux nouvelles approches pour calculer l'homologie persistante (HP) d'une séquence de complexes simpliciaux. L'idée de base est de simplifier les complexes de la séquence d'entrée en utilisant des types spéciaux de collapses (effondrement), les collapses forts et les collapses d'arêtes, et de calculer l'HP d'une séquence réduite de plus petite taille qui a la même HP que la séquence initiale.

Notre première approche utilise les collapses forts introduits par J. Barmak et E. Miniam [DCG (2012)]. Un collapse fort supprime les sommets dits dominés d'un complexe simplicial. Notre approche utilisant les collapses forts a plusieurs caractéristiques qui la distinguent des travaux antérieurs. La méthode n'est pas limitée aux filtrations (c'est-à-dire aux séquences de sous-complexes simpliciaux imbriqués) mais fonctionne pour d'autres types de séquences comme les tours et les zigzags. Par ailleurs, pour implémenter les collapses forts, il suffit de représenter les simplexes maximaux du complexe, et pas l'ensemble de tous ses simplexes, ce qui économise beaucoup d'espace et de temps. De plus, les complexes de la séquence peuvent être collapsés indépendamment et en parallèle.

Dans le cas des complexes en drapeaux (flag complexes), les collapses forts peuvent être réalisés sur le 1-squelette du complexe et le complexe résultat est également un complexe en drapeau. Nous montrons que si l'on restreint la classe des complexes simpliciaux aux complexes en drapeaux, on peut améliorer la complexité en temps et en espace de facon décisive par rapport aux travaux antérieurs. Lorsque les collapses forts sont appliqués aux complexes d'une tour de complexes en drapeau, nous obtenons une séquence réduite qui est aussi une tour de complexes en drapeau que nous appelons le coeur de la tour. Nous convertissons ensuite le coeur de la tour en une filtration équivalente pour calculer son HP. Là encore, nous n'utilisons que les 1-squelettes des complexes. La méthode résultante est simple et extrêmement efficace.

Nous étendons la notion de sommet dominé au cas de simplexes de dimension quelconque. Le concept d'arête dominée apparait très puissant et nous l'étudions dans le cas des complexes en drapeaux de facon plus détaillée. Nous montrons que les collapses d'arêtes (suppression des arêtes dominées) dans un complexe en drapeaux peut être effectué, comme précédemment, en utilisant uniquement le 1-squelette du complexe. En outre, le complexe résiduel est également un complexe de drapeaux. Ensuite, nous montrons que, comme dans le cas des collapses forts, on peut utiliser

les collapses d’arêtes pour réduire une filtration de complexes en drapeaux en une filtration de complexes en drapeaux plus petite qui a la même HP. Là encore, nous utilisons uniquement le 1-squelettes des complexes.

Comme l’ont démontré de nombreuses expériences sur des données publiques, les approches développées sont extrêmement rapides et efficaces en mémoire. En particulier, la méthode utilisant les collapses d’arêtes offre de meilleures performances que toutes les méthodes connues, y compris l’approche par collapses forts. Enfin, nous pouvons faire des compromis entre précision et temps de calcul en choisissant le nombre de complexes simpliciaux de la séquence à collapser.

UNIVERSITÉ CÔTE D'AZUR

Abstract

ED-Stic

Inria

Doctor of Philosophy

Collapses and Persistent Homology

by Siddharth Pritam

In this thesis, we introduce two new approaches to compute the Persistent Homology (PH) of a sequence of simplicial complexes. The basic idea is to simplify the complexes of the input sequence by using special types of collapses (strong and edge collapse) and to compute the PH of an induced sequence of smaller size that has the same PH as the initial one.

Our first approach uses strong collapse which is introduced by J. Barmak and E. Miniam [DCG (2012)]. Strong collapse comprises of removal of special vertices called *dominated* vertices from a simplicial complex. Our approach with strong collapse has several salient features that distinguishes it from previous work. It is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. To strong collapse a simplicial complex, we only need to store the maximal simplices of the complex, not the full set of all its simplices, which saves a lot of space and time. Moreover, the complexes in the sequence can be strong collapsed independently and in parallel.

In the case of flag complexes strong collapse can be performed over the 1-skeleton of the complex and the resulting complex is also a flag complex. We show that if we restrict the class of simplicial complexes to flag complexes, we can achieve decisive improvement in terms of time and space complexities with respect to previous work. When we strong collapse the complexes in a flag tower, we obtain a reduced sequence that is also a flag tower we call the core flag tower. We then convert the core flag tower to an equivalent filtration to compute its PH. Here again, we only use the 1-skeletons of the complexes. The resulting method is simple and extremely efficient.

We extend the notions of dominated vertex to a simplex of any dimension. Domination of edges appear to be very powerful and we study it in the case of flag complexes in more detail. We show that edge collapse (removal of dominated edges) in a flag complex can be performed using only the 1-skeleton of the complex as well. Furthermore, the residual complex is a flag complex as well. Next we show that, similar to the case of strong collapses, we can use edge collapses to reduce a flag filtration $\mathcal{F}$ to a smaller flag filtration $\mathcal{F}^c$ with the same persistence. Here again, we only use the 1-skeletons of the complexes.

As a result and as demonstrated by numerous experiments on publicly available data sets, our approaches are extremely fast and memory efficient in practice. In particular the method using edge collapse performs the best among all known methods including the strong collapse approach. Finally, we can compromize between precision

and time by choosing the number of simplicial complexes of the sequence we strong collapse.

Declaration of Authorship

I, Siddharth Pritam, declare that this thesis titled, "Collapses and Persistent Homology" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Various pictures used in this thesis specially in Chapter 1 and Chapter 2 has been retrieved with free access from internet.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Siddharth Pritam

Date: 01/07/2020

“Arise awake and stop not until the goal is achieved ”

Swami Vivekanand

Acknowledgements

This thesis is an outcome of a collaborative effort and has a unique story behind it, like almost all other theses. It is almost impossible to thank each and everyone who has been part of this journey but I'll try my best to do so and if I could not mention someone please forgive me.

First and foremost, I would like to thank Jean-Daniel for giving me the opportunity to be his PhD student. His unique mentorship has effortlessly nurtured the essential qualities in me that are required to be a researcher. His vision has helped me develop a taste for algorithmic mathematics with an emphasis on simplicity and efficiency. He also gave me a complete freedom and let me wonder scientifically and geographically. Apart from teaching me the technical skills, his encouragement and support has always helped me navigate through the tough times. Despite his busy schedule, he always had time for his students and was always available to provide his valuable inputs. I also thank him for his amiable personality and sense of humor which made the whole journey way more smoother than it should be.

I thank Michael Kerber who introduced me to Persistent Homology during my internship at MPI Saarbruecken. That internship turned out to be very useful in the end. I also thank him for writing the report of my thesis and being part of the jury.

I would not have met Jean-Daniel, if it wasn't for Arijit Ghosh, whom I met at MPI Saarbruecken. Being a senior from IIT Kharagpur, Arijit has always supported me like a big brother and I am very grateful for that.

I want to thank the members of my thesis committee: Dominique Attali, Frédéric Cazal, Tamal Dey, Marc Glisse, Kathryn Hess and André Lieutier. Their feedback, and insights have been very helpful, specially Dominique Atalli for a thorough examination of the manuscript and identifying many crucial mistakes.

It's been a privilege to do my graduate studies at DataShape, one of the world's leading research team in Computational Topology. I thank Marc Glisse for taking interest in my research and providing many valuable inputs during numerous discussions we had. I thank Vincent Rouvreau for his help with implementations of all the codes in GUDHI. I thank Mathijs Wintraecken for reading initial drafts of some of my papers and his peculiar way of helping me maintain some mental sanity during the initial years of my PhD studies. I thank Kunal Dutta for not letting me miss India too much. I would like to thank Owen Rouille for organizing the online defense during the Covid-19 pandemic. And I thank David-Cohen Steiner and Clément Maria for their many academic and non-academic discussions. I would also like to thank Florence Barbara and Sophie Honnorat for their efficient and sincere support in administrative matters.

Working on a doctoral thesis can lead to obnoxious behavior, I thank all my friends at Inria who kept me socially acceptable, specially Romain Tetley and Ivana Kojcic.

I thank my brother Shashank and my sister Shristi for their consistent support and making me feel that I am doing something worthwhile. I thank my deerest friend Kritiva for her love and encouragement and being part of the journey since the very beginning of the search for doctoral position.

I finally end with thanking my parents and my uncle for their immense contribution, support, encouragement, hardships and sacrifices they made to make me achieve whatever little I could do in this world.

Contents

Abstract **v**

Declaration of Authorship **vii**

Acknowledgements **xi**

1 Introduction **1**

1.1 Topological Data Analysis 1

1.2 Persistent Homology 3

1.3 Extensions of Persistent Homology 7

1.4 Challenges and Previous Work 8

1.5 Our Contribution 10

1.5.1 Strong Collapse Approach. 10

1.5.2 Edge Collapse Approach. 12

1.6 Outline 13

2 Basic Notions **15**

2.1 Topology 15

2.1.1 Topological Space 15

2.1.2 Topological Equivalences 16

2.2 Simplicial complex 17

2.2.1 Geometric simplicial complex 17

2.2.2 Abstract simplicial complex 18

2.2.3 Examples of simplicial complexes 18

2.2.4 Star, Link and Simplicial Cone 19

2.2.5 Flag complex and Neighborhood 20

2.2.6 Simplicial maps 20

2.3 Collapses and Edge Contraction 21

2.3.1 Simple Collapse 22

2.3.2 Edge Contraction 22

2.4 Homology Groups 23

2.4.1 Simplicial Homology 23
2.4.2 Singular Homology 25
2.5 Persistent Homology 27
2.5.1 Sequences of complexes 28
2.5.2 Persistent homology. 28
2.5.3 Computing Persistence Diagram 29
2.5.4 Equivalence and Stability of Persistence Modules 30
2.6 Conclusion 33

3 Strong Collapse And Persistence 35
3.1 Strong Collapse 35
3.2 Nerve and Strong Collapse 37
3.3 Strong collapse of a simplicial complex 38
3.4 Strong collapse of a sequence of simplicial complexes 42
3.5 Approximation of the persistence diagram of filtrations 45
3.6 Discussion 49

4 Persistence of Flag Complexes 51
4.1 Strong Collapse of a Flag complex 51
4.2 From a Flag Tower to a Flag Filtration 54
4.2.1 Previous work 54
4.2.2 A new construction 54
4.3 Computational experiments 59

5 Edge Collapse And Persistence 63
5.1 Edge Collapse 63
5.2 Simple Collapse and Persistence 65
5.3 Edge collapse of a flag filtration 67
5.4 Computational Experiments 71
5.5 Discussion 74

6 Conclusion 75

List of Figures

1.1 Point cloud sampled from a torus. 1
1.2 Left: Point set in $\mathbb{R}^2$ and Right: Its clusters represented in different colors. 2
1.3 A regression line fitting the given point set. 2
1.4 Points in $\mathbb{R}^2$ and euclidean balls around them, with increasing radii. . 3
1.5 At no scale the union of balls can capture both the loops. 3
1.6 Union of balls and the associated Čech complex. 4
1.7 Union of restricted balls and the associated alpha complex. The black edges are the 1-cells of the Voronoi cell decomposition that restrict the balls around the points. 5
1.8 A Persistence Diagram: Different colors denote different dimensions of the homology groups. 6

2.1 On the left is the Čech complex and the right is the Rips complex. . . . 19
2.2 Maps ϕ and ψ are contiguous. 21
2.3 Examples of simple collapses. 22
2.4 The edge $[u, v]$ satisfies the link condition. The complex (in the right) which is the image of an edge contraction may not be a subcomplex of the original complex K. As the triangle $[p, q, u]$ doesn't exist in K. . . . 23
2.5 At no scale the union of balls can capture both the loops. 27
2.6 A sequence of simplicial complexes connected through inclusions. . . 28

3.1 Illustration of an *elementary strong collapse*. In the complex on the left, v is dominated by v'. The link of v is highlighted in red. Removing v leads to the complex on the right. 36
3.2 Left: K (in grey), Right: $\mathcal{N}(K)$ (in grey) and $\mathcal{N}^2(K)$ (in red). $\mathcal{N}^2(K)$ is isomorphic to a full-subcomplex of K highlighted in red on the left. . . 38

4.1 Demonstration of strong expansion: On the left we have the active neighborhoods of the vertex u (in red) and of v (in blue) in the graph $\mathbb{G}_i$. The graphs $\mathbb{G}_i$ might have other vertices and edges, here we have zoomed into the local active neighborhoods of u and v. On the right, we have the local neighborhoods of u and v in $\mathbb{G}_{i+1}$ after conning v to the active neighborhood of u. 55

4.2 An example of the graphs associated to the complexes defined in the proof of Lemma 4.2.1. Vertices in red are dominated (inactive) and in green are active. 57

5.1 The above complex does not have any dominated vertex and thus cannot be 0-collapsed. However, by proceeding from the boundary edges, one can edge collapse this complex to a 1-dimensional complex. The 1-core obtained in this way is collapsible to a point using 0-collapse. 65

5.2 The complex on the left has two different 1-cores, the one in the middle is obtained after removing the inner edges $[1, 3]$ and $[4, 6]$, and the one in the right by removing the outer edges $[1, 2]$ and $[4, 5]$. Note that the one in the right can be further strong collapsed. 65

List of Tables

3.1 From left to right $M, \mathcal{N}(M)$ and $\mathcal{N}^2(M)$. 39

3.2 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum scale parameter (Threshold), dimension of the VR Complex (Dim), time taken to compute the VR complex (Rips-Comp-Time), total time (Total-Time), parameter step (Step) and total number of snapshots used (TotSnaps). All times are averaged over five trials. . 47

3.3 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum scale parameter (Threshold), input dimension for Ripser (Dim), total time taken by Ripser (Time). Most results are averaged over five trials except the longer ones. ∞ in the Time column means that the experiment ran longer than 12hrs or crashed due to memory overload. 47

4.1 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by VertexCollapser, total time (in seconds) including PD computation (Tot-Time), parameter *Step* and the number of snapshots used (Snaps). K and M in the size columns are thousands and millions respectively. 61

4.2 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), parameter (*dim*), Time is the total time (in seconds) taken by Ripser. ∞ means that the experiment ran longer than 12 hours or crashed due to memory overload. 62

5.1 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), Initial number of edges/Critical (final) number of edges *Edge(I)/Edge(C)*, number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by Edge-Collapser and total time (in seconds) including PD computation (Tot-Time). K and M in the size columns are thousands and millions respectively. 72

5.2 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by VertexCollapser, total time (in seconds) including PD computation (Tot-Time), parameter *Step* (linear approximation factor) and the number of snapshots used (Snaps). *The last experiment (torus) could not finish (>12hrs) the preprocessing due to large number of snapshots and the size of the complex. K and M in the size columns are thousands and millions respectively. 73

5.3 The first three columns are as in Table 5.2 and then, from left to right: parameter (*dim*), Time is the total time (in seconds) taken by Ripser. ∞ means that the experiment ran longer than 12 hours or crashed due to memory overload. 73

Dedicated to my village and river Surhar.

Chapter 1

Introduction

1.1 Topological Data Analysis

Empirical verification is at the heart of scientific investigation and data is the main object which facilitates the verification. There are many different ways one can possibly extract information out of data. Mathematics is the language that provides different tools through which we can interact with data. Each tool reveals a different nature of the data and often their combinations provides a more broader understanding. In this thesis we focus on one specific mathematical tool, Topology along with Geometry.

Topology is the branch of mathematics which studies the characterization of invariants of a space under 'continuous deformations'. These invariants can be the number of connected components, loops, voids, holes, tunnels and their higher-dimensional analogs. They can also be purely algebraic or combinatorial in nature, like the Euler characteristic.

In most cases data comes with a notion of similarity between its constituent elements (*points*) which leads to the notion of distance between them. The notion of distance allows us to talk about geometry and topology. Therefore data has *shape* and using geometric and topological methods one could extract the information about its shape. Metric space is the formalism of the concept of similarity and often in practice data is presented as a point set in a metric space. Given a point set P in a metric space, we assume that it is a finite sample of some underlying space X and we want to infer the topology of X using P. Below is a simple example, a point set sampled from a torus.

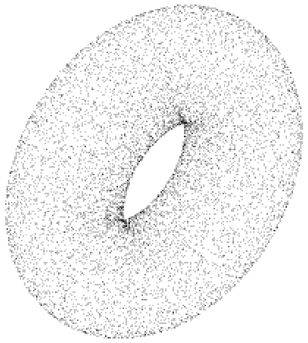


FIGURE 1.1 – Point cloud sampled from a torus.

The idea of topological inference is not entirely new and it has been done in rather simplistic forms. For example given a point set as in Figure 1.2, we would like to classify or to cluster them into similar groups. The topological interpretation of clustering would be to understand the nature and the number of connected components of the point set. Traditionally there are algorithms like K-means and PCA (Principle Component Analysis) to perform such tasks.

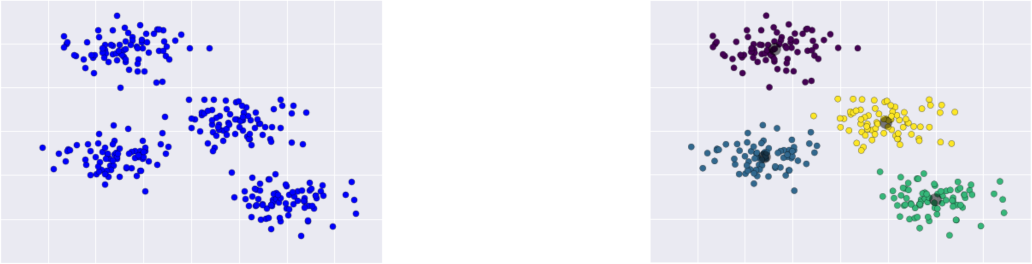


FIGURE 1.2 – Left: Point set in $\mathbb{R}^2$ and Right: Its clusters represented in different colors.

Another example is regression where one starts with certain hypothesis about the shape of the data like being linear and wants to compute the line that fits the data best as shown in Figure 1.3.

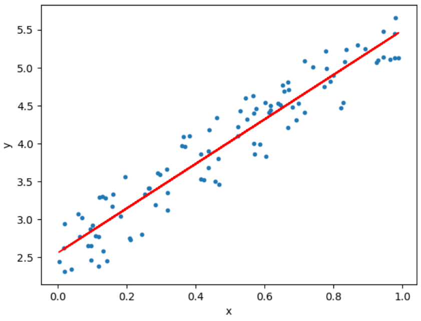


FIGURE 1.3 – A regression line fitting the given point set.

The underlying space X could be far more complex than being linear or of the shape of torus, with possibly many more *holes* of different sizes and of high dimensions. The mathematical formalism which captures the notion of connectivity, loops, voids, tunnels and their higher-dimensional analogs is called *homology*. Homology is a topological invariant, that is, it does not change under ‘continuous deformations’ and it is of algebraic nature. We associate one group per dimension to the space X that ‘measures’ the number and the nature of the ‘holes’ of X. Topological Data Analysis (abbreviated as TDA) is a new and fast emerging field, whose goal is to extract geometric and topological properties of complex data with advanced tools like homology.

1.2 Persistent Homology

Since the point set P is usually a finite sample of X therefore it can not have the exact topology of X. One possible way to infer the topology of X is to consider balls of some radius ϵ (called *scale*), around each point of P such that their union *covers* X in a 'nice' way.

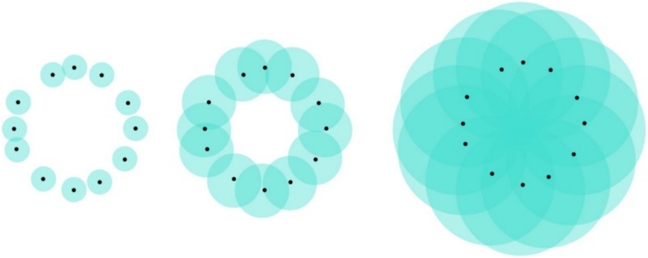


FIGURE 1.4 – Points in $\mathbb{R}^2$ and euclidean balls around them, with increasing radii.

In the above example Figure 1.4, at very small scale, the points seem to be isolated, with balls around them forming many small clusters. At medium scale, it is apparent that the points have circular shape. Finally, at large scale, the balls around them forms a single large cluster and does not seem to represent the original shape. Therefore, the radius of the balls in the middle could be thought as the 'right' scale at which the union of balls represents the circle from which the points could have been possibly sampled. This brings us the question of how to choose the 'right' scale. However, there might not be a single 'right' scale. In the example below, Figure 1.5, at no scale the union of balls could capture the two loops.

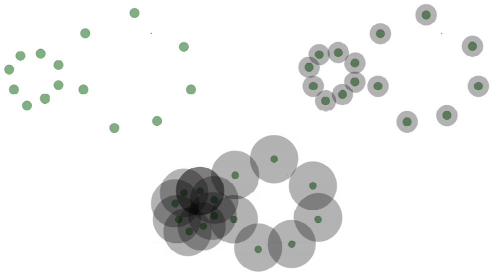


FIGURE 1.5 – At no scale the union of balls can capture both the loops.

To resolve this issue we would naturally like to look at the balls around the point set at multiple scale. If we track the evolution of the union of the balls as the scale increases, we see that a specific topological feature (hole) starts appearing at a certain scale and then dies at another scale. Therefore, every specific topological feature has a life span in this process, some *persist* longer and others die quickly. The natural conclusion of this observation would be that the features that last longer are real features of the data and the small persisting features are noise. The mathematical idea to capture this intuition is called *Persistent homology*.

Persistent homology (PH) was introduced by Edelsbrunner, Letscher and Zomorodian in their milestone paper [27]. The scale of life and death of all topological features could be summarized as points *(life, death)* in $\mathbb{R}^2$. This summary of points is called the *persistence diagram*, see Figure 1.8 for an example. In the same paper [27], Edelsbrunner et. al. provided a simple and fast algorithm to compute the *persistence diagram* as well.

The computation of persistence diagram is performed over a combinatorial or discrete structure on the given point set P. This structure is called a *Simplicial complex*. A simplicial complex is a collection of points, edges, triangles, tetrahedron and their higher dimensional analogs called *simplices*, such that they are 'nicely' glued. Formally, a *k-simplex* σ is the convex hull of $(k+1)$ affinely independent points and k is called its *dimension*. A sub-collection τ of the $(k+1)$ points is also a simplex, called a *face* of σ and σ is called a *co-face* of τ. A simplicial complex K is then a collection of simplices such that

1. A face τ of any simplex σ is in K.
2. Two simplices σ_1 and σ_2 in K intersect only at a common face τ.

The dimension of a simplicial complex K is the maximum of the dimensions of its simplices. Simplicial complexes are one of the most important combinatorial structure in computational topology and arise naturally in many different contexts. There is a simplicial complex associated to the union of balls around the points in P. Given P and scale ϵ, we add edges between the points of P whenever two balls around them of radius ϵ have a common intersection, we put a triangle between three points of P whenever the balls around them have a common intersection, likewise we add tetrahedrons and higher dimensional simplices. The simplicial complex constructed in this fashion is called the *Čech complex* denoted as $C(P, \epsilon)$. Most important, the union of balls and the Čech complex have the same topology (more precisely they are *homotopy equivalent*). See Figure 1.6 for an example of a union of balls and its associated Čech complex.

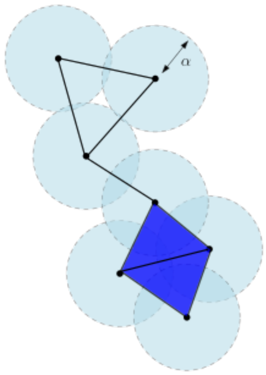


FIGURE 1.6 – Union of balls and the associated Čech complex.

As we increase the scale, the associated Čech complexes $\{C(P, \epsilon)\}_{0 \leq \epsilon}$ form a nested sequence, i.e. for $\epsilon_1 \leq \epsilon_2$, $C(P, \epsilon_1) \subseteq C(P, \epsilon_2)$. More general such a sequence of nested simplicial complexes is called a *filtration*. Traditionally persistence has been defined and computed for such sequences.

The *Alpha complex* is another simplicial complex which arises from the union of balls and has the same homology as the Čech complex. Here, we restrict the ball around a point by the corresponding *Voronoi cell* of the point and look at the intersection patterns of the restricted balls. When two restricted balls intersect we put an edge and for an intersection of three restricted balls we put a triangle and so on so forth. This means that given two points whose Voronoi cells are not intersecting the balls around them would not intersect as well. Therefore points that are far apart will not span a simplex in the alpha complex. The alpha complex is a subcomplex of the Čech complex and the *Delaunay triangulation*. Below is an example of a union of restricted balls and its associated alpha complex, Figure 1.7.

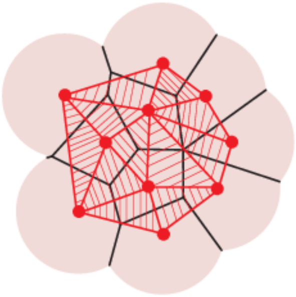
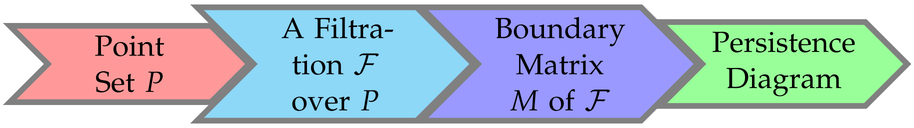


FIGURE 1.7 – Union of restricted balls and the associated alpha complex. The black edges are the 1-cells of the Voronoi cell decomposition that restrict the balls around the points.

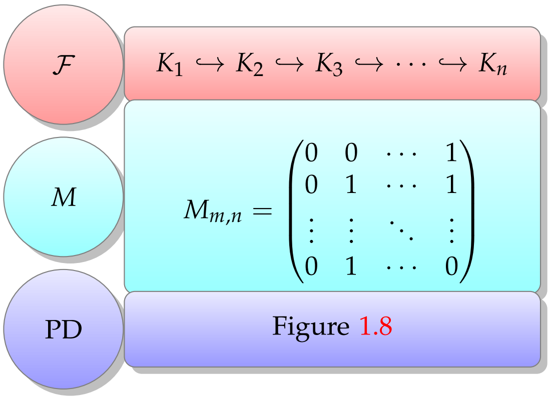
We define another simplicial complex closely related to the Čech complex. We define a subset of points of P as a simplex σ whenever pairwise distance between any two points of σ is less than 2ϵ. We call this simplicial complex the *Vietoris-Rips* complex or simply the *Rips* complex, denoted as $R(P, \epsilon)$.

The Rips complex is a special type of simplicial complex called a *Flag complex*. Flag complexes are fully characterized by their graph (or 1-skeleton), the other faces being obtained by computing the cliques of the graph. Hence, a flag complex can be represented by its 1-skeleton, which is a very compact representation. Flag complexes are very popular and, in particular, Rips complexes are widely used in Topological Data Analysis. Along with Rips complex, Čech and alpha complexes are commonly used too, specially in low dimensions. We discuss more about their advantages and disadvantages in Section 1.4.

We briefly summarize the complete pipeline of the PH computation.


Point Set P
A Filtration $\mathcal{F}$ over P
Boundary Matrix M of $\mathcal{F}$
Persistence Diagram

Given a point set P in a metric space, we build a sequence of simplicial complexes (for example a filtration $\mathcal{F}$ of Rips complex) over P. We then compute a *boundary matrix* M over a filtration, such that the columns and rows of the boundary matrix are the simplices of the filtration sorted as per their filtration value. A non-zero entry $M[\sigma_i][\sigma_j]$ implies σ_i is a face of σ_j. We then reduce this matrix to a special form, to retrieve special pairings of the simplices that corresponds to the points in the persistence diagram. We discuss the reduction algorithm in more details in Chapter 2.


$\mathcal{F}$
$K_1 \hookrightarrow K_2 \hookrightarrow K_3 \hookrightarrow \cdots \hookrightarrow K_n$
M
$M_{m,n} = \begin{pmatrix} 0 & 0 & \cdots & 1 \\ 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 0 \end{pmatrix}$
PD
Figure 1.8

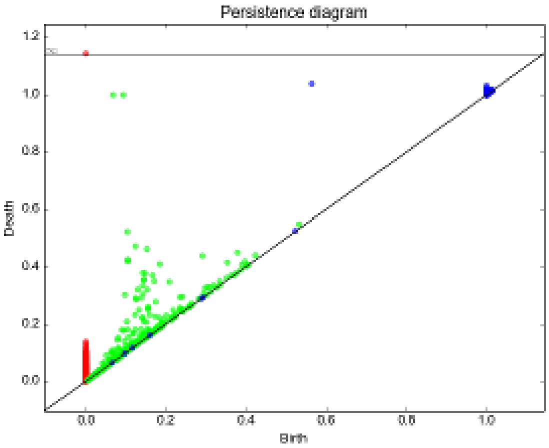


FIGURE 1.8 – A Persistence Diagram: Different colors denote different dimensions of the homology groups.

Stability: Two point sets P and P' can be compared by their corresponding persistence diagrams associated to some filtrations defined over P and P'. The *bottleneck distance* is a measure of similarity between two persistence diagrams, which we define more precisely in Chapter 2.

Stability of persistence diagrams is one of the most important result in the theory of persistence. It states that, small perturbations in the point set P leads to a small perturbation in its persistence diagram. More formally if P and P' are two point sets such that their *Hausdorff distance* is at most ϵ then the bottleneck distance between their corresponding persistence diagrams would be bounded above by ϵ. The first result about stability of persistence diagram was proved by Cohen-Steiner, Edelsbrunner and Harer in [17]. The stability in [17] was defined for functions however we used the formulation of point sets for simple exposition.

At a scale ϵ, the Čech complex and the Rips complex of a point $P \in \mathbb{R}^d$ follow the following inclusion property,

$$R(P,\epsilon) \subseteq C(P,\sqrt{2}\epsilon) \subseteq R(P,\sqrt{2}\epsilon)$$

and are called *interleaved*. As a consequence of stability theorem, the interleaving relation implies that the bottleneck distance between the persistence diagram of a

Rips filtration and the persistence diagram of the corresponding Čech filtration is bounded above by $\sqrt{2}$.

1.3 Extensions of Persistent Homology

Since its introduction, the theory of persistence has evolved very rapidly in subsequent years both on theoretical and algorithmic fronts. Carlsson and Zomorodian provided solid algebraic foundations and extended the algorithm for more general cases in [50]. Subsequently, Persistent Homology was defined for sequences more general than filtrations. A *Zigzag filtration* is a sequence of simplicial complexes connected through inclusions and removals (inclusion in the reverse direction) of simplices:

$$\text{A zigzag filtration: } K_1 \hookrightarrow K_2 \hookleftarrow \cdots \hookrightarrow K_m$$

Carlsson and Silva defined and provided an algorithm to compute zigzag persistence [12] which was further improved in [13].

The theory of persistence extends naturally to zigzag sequences due to their connection with *quiver theory* [20]. For the same reason it was natural to define persistence diagrams corresponding to sequences of simplicial complexes connected through more general simplicial maps not just inclusions. Such a sequence is called a *tower* if the maps are in the same directions and a *zigzag sequence* if the maps are in both directions. Below are examples of a tower and a zigzag sequence respectively.

$$\text{Tower: } K_1 \xrightarrow{f_1} K_2 \xrightarrow{f_2} \cdots \to K_m$$

$$\text{A zigzag sequence: } K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} \cdots \to K_m$$

Here f_i and g_i are simplicial maps. The first algorithm to compute the persistence of a tower was given by Dey, Fan and Wang in [22]. Their strategy was to transform a tower into an equivalent filtration using the *link condition* over edges between contracting vertices of the simplicial maps. Their approach was further improved by Kerber and Schreiber in [34] by using *coning* instead of the link condition. They also provided a theoretical guarantee over the size of the equivalent filtration. Using the same strategies it is possible to compute the zigzag persistence of any general zigzag sequence by converting it to an equivalent zigzag filtration.

Towers appear frequently in situations where one wants to approximate or sparsify a filtration using certain methods that involves mapping several vertices to one vertex that is either a topological or a geometric representative. The examples of such methods can be found in the work of Dey et. al. [22] and the work of Choudhary et. al. [16]. Our work reported in [10, 9] also reduces a filtration usually to a tower.

Persistence has also been extended to multiple parameters, that is, there are more variables than the distance that dictate the growth of the filtrations. It is then called *multidimensional persistence*. Multidimensional persistence has been introduced by Carlsson and Zomorodian [51]. Multidimensional persistence does not have a well behaved descriptor like persistence diagram. However, there are other invariants like the *rank invariant* [51] and *minimal presentations*. Recently an algorithm to compute minimal presentations for 2-dimensional filtrations has been introduced [36].

1.4 Challenges and Previous Work

Challenges. The size of a filtration is defined as the size (number of simplices) of the last (largest) complex in the sequence. If the size of the filtration is n then the computation of persistence takes $\mathcal{O}(n^3)$ time [1]. In practice, it has been observed that the running time of the persistence algorithms is linear in n. This makes computation of persistence reasonably fast for small filtrations. However as the dimension of the simplices grows the size of any filtration (Čech, Rips, alpha) grows exponentially with the dimension and it becomes impractical to compute persistence for higher dimensional filtrations. The major bottleneck in computing persistence diagram is the computation time of the complexes and their sizes. Improving the performance of computing Persistent Homology has been a central goal in Topological Data Analysis since the early days of the field about 20 years ago.

The time complexity of computing the Čech complex depends exponentially on the dimension d of the ambient space. Also the number of simplices in a Čech filtration could be very large, exponential in the number of points in P, $\mathcal{O}(2^{n_v})$, where $n_v = |P|$. Therefore the computation time and its size makes a Čech filtration almost unusable for PH computation.

Alpha complexes have a better size complexity, $\mathcal{O}(n_v^{\lfloor \frac{d}{2} \rfloor})$, where n_v is the number of points in P and d is the ambient dimension. That leads to a smaller size and fast computation in low ambient dimensions. However, in higher ambient dimensions the alpha complex has the same issues as the Čech complex.

The time to compute Rips complexes is much faster as we only need to compute the pairwise distances between the points of P and therefore the time complexity is independent of the ambient dimension d of the point set. However, Rips complexes has the same size complexity as the Čech complex, and therefore as the dimension of the complex grows the computation of persistence of Rips filtration is time intensive as well.

The *k-skeleton* of a simplicial complex K is the set of simplices of dimension less than or equal to k, it is a subcomplex of K. To compute Persistent Homology until dimension k, we only need simplices up to dimension $k+1$, i.e. the $(k+1)$-skeleton of the filtration. One way to restrict the size of a filtration is to restrict it to some particular skeleton. However, by doing so we loose the information about possible higher dimensional features in the persistence diagram of the filtration. Also, in practice restricting to k-skeleton is only tractable for values of k up to 4 or 5.

Therefore for higher dimensional simplicial complexes (Rips, Čech, alpha) the problem of size persists. The large size of these filtrations not only slows down the actual persistence computation, but it also induces large computation time to set-up the boundary matrix which we need to *reduce* to compute the persistence diagram. This thesis addresses this problem in a novel way and provides a solution which is a quite significant step forward.

---

[1] The theoretical time complexity to compute PH is $O(n^{\omega})$, where $\omega \leq 2.4$ is the matrix multiplication exponent [39, 29]. However, most practical algorithms has time complexity $\mathcal{O}(n^3)$.

Previous Work. Much progress has been accomplished in recent years along several directions. Regarding PH computation itself, a number of clever implementations and optimizations have led to a new generation of software [30, 5, 4, 41]. Gudhi [30] and Phat [5] compute PH for any filtrations whereas Ripser [4] specializes in Rips filtration. Dionysus [41] is designed to compute the persistence diagram of a zigzag filtration.

One of the first significant optimization technique was developed in [6]. The notion of persistent cohomology was introduced in [19]. Gudhi and Dionysus use the dual algorithm (computes *persistent cohomology*) to compute persistence diagrams which is substantially faster than the standard algorithm (persistent homology). In all the above mentioned technique the goal was to make the reduction of the boundary matrix faster.

A second way to improve the overall process is to reduce the size of the complexes in the input sequence while preserving (or approximating in a controlled way) the persistent homology of the sequence.

The first approximation scheme for filtrations was given by Hudson et. al in [32]. Sheehy gave the first approximation scheme for Rips filtrations [45]. In a metric space, for a given $\epsilon \in (0, 1)$, they construct a $(1 + \epsilon)$-approximate filtration of the Rips filtrations. The size of the k-skeleton of the approximate filtraion is $n(\frac{1}{\epsilon})\mathcal{O}(\lambda k)$, where λ is the doubling dimension of the metric. The work of Sheehy [45] has two significant new features, 1. the approximation factor ϵ is an input parameter of the scheme, 2. the size of the approximate filtration depends on the dimension of the skeleton.

Dey et. al gave a similar approximation scheme for the Rips filtrations [22]. Their method used vertex collapses based on geometric proximity and they reduce a filtration to an approximate simplicial tower. The scheme was further improved in [23], leading to a significantly better performance in practice compare to [22] and [45].

Several other algorithms with theoretical guarantees have been proposed recently using a variety of similar ideas and techniques, e.g. interleaving smaller and easily computable simplicial complexes, or sub-sampling the point sample [14, 11, 35, 15]. The work reported in [35, 15] approximates a Čech filtrations using geometric techniques. The work of Botnan and Spreemann [11] approximates a Čech filtration using collapses derived using geometric criterion. Chazal and Oudot [14] use interleaving of the Rips complex and the *Witness* complex with the Čech complex to approximate the persistence of a Čech filtration.

The above approximation methods are based on geometric techniques. There are reduction methods based on topological notions. Examples of exact reductions can be found in the work of Mischaikow and Nanda [40] who use Morse theory to reduce the size of a filtration. The work of Maria and Schreiber [38] extends the work of Mischaikow and Nanda [40] to zigzag filtrations. Similar to the work of Mischaikow and Nanda [40], the work of Dłotko and Wagner use simple collapses [25]. These methods preserve the filtration nature of the output sequence, i.e. a filtration is reduced to a filtration and a zigzag filtration is reduced to a zigzag filtration. The above schemes [40, 25, 38] do not provide theoretical guarantees over the output size of the filtration as it is NP-Complete to compute optimal Morse-matching or a smallest 'simple'-core.

Recently Dey and Slechta [24] have used the notion of *Edge contraction* (different than the Edge collapses introduced in this thesis) that involves contracting edges that satisfy so called *link condition* to simplify a filtration. Their method is approximate for general simplicial complexes and exact when the underlying space of the simplicial complex is a 2-manifold.

1.5 Our Contribution

As described in the previous section the problem of efficient computation of persistent homology is the central topic of research since its introduction. Motivated by this problem, in this thesis, we introduce two new approaches (corresponding to two types of *collapses*) to compute the PH of a given sequence of simplicial complexes (filtration, tower, zigzag sequence) in an efficient way. Both approaches are closely related and are as follows,

1. We use the notion of strong collapse introduced by J. Barmak and E. Minian [3] to simplify the complexes of the input sequence.

2. Instead of strong collapses, we use the so-called edge collapses. In fact, we more generally define k-collapses. When $k = 0$, we have strong collapses and when $k = 1$ edge collapses and we use edge collapse to simplify a filtration.

Both of our approaches are pre-processing methods and they are applied even before the *boundary matrix* is computed. In the case of general simplicial complexes our collapse strategy requires only the maximal faces of the simplicial complexes. And In the case of Flag complex both of them work on the 1-skeleton and this is the most powerful aspect of our approach. Therefore the collapses of the simplicial complexes are fast and memory efficient. Once we have a smaller sequence of simplicial complex we proceed normally by setting up the boundary matrix and then its reduction.

Numerous experiments show that both approaches outperform previous methods. Edge collapse performs better than strong (vertex) collapse. However, vertex and edge collapses can be applied together one after another. Also as both vertex and edge collapses are pre-processing methods they can be combined with any other pre-processing method and any PH computation software.

1.5.1 Strong Collapse Approach.

We use the notion of strong collapse as introduced by Barmak and Minian [3] to reduce a sequence of simplicial complexes. Specifically, our 1st approach can be summarized as follows. Given a sequence

$$\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$$

of simplicial complexes K_i connected through simplicial maps $\{\xrightarrow{f_i} \text{ or } \xleftarrow{g_j}\}$. We independently strong collapse the complexes of the sequence to reach a sequence

$$\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\},$$

with *induced simplicial maps* $\{\xrightarrow{f_i^c} \text{ or } \xleftarrow{g_j^c}\}$ (defined precisely in Chapter 3). The complex K_i^c is called the **core** of the complex K_i and we call the sequence $\mathcal{Z}^c$ the **core sequence** of $\mathcal{Z}$. We show that one can compute the PH of the sequence $\mathcal{Z}$ by computing the PH of the core sequence $\mathcal{Z}^c$, which is usually of much smaller size.

Our approach has some similarity with the work of Wilkerson et. al. [48, 49] who also use strong collapses to reduce the PH computation but it differs in three essential aspects: it is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. It also differs in the way strong collapses are computed and in the manner PH is computed.

General simplicial complexes. A first central observation is that to strong collapse a simplicial complex K, we only need to store its maximal simplices (i.e. those simplices that have no coface). The number m of maximal simplices is smaller than the total number of simplices by a factor that is exponential in the dimension d of the complex and m is linear in the number n_v of vertices for a variety of complexes [7]. Working only with maximal simplices dramatically reduces the time and space complexities compared to the algorithm of [49]. We prove that the complexity of our algorithm is $\mathcal{O}(n_v^2 d\Gamma_0^2)$. Here Γ_0 is the largest number of maximal simplices incident to a vertex. As observed in [8, 7], Γ_0 is usually small and appears to be almost constant along a filtration (see Section 3.5 for a discussion). As a consequence, the time to collapse the i-th simplicial complex K_i in the sequence is almost independent of i, which is a clear advantage over methods that use a full representation of the complexes and suffer an increasing cost as i increases.

Once a reduced sequence has been computed as described above, we cannot compute its PH directly. Indeed, all PH algorithms require as input a full representation of the complexes. We thus have to convert the representation by maximal simplices used to efficiently perform the strong collapses into a full representation of the complexes. This takes time exponential in the dimension (of the collapsed complexes). However, this exponential burden is to be expected since it is known that computing PH is NP-hard when the complexes are represented by their maximal faces [1]. Nevertheless, we demonstrate in Chapter 3 that strong collapses combined with known persistence algorithms lead to major improvements over previous methods to compute the PH of a sequence.

The major advantages of our approach are:

– The collapses of the complexes in the sequence can be performed independently and in parallel. This is due to the fact that strong collapses can be expressed as simplicial maps, unlike simple collapses [47].

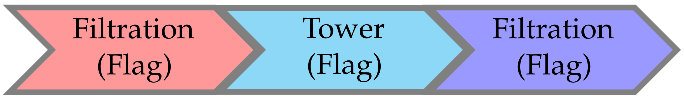
– Instead of computing the exact PH, we can compute an approximate PH which is substantially faster at a very minimal cost. The approximation scheme is detailed in Sections 3.5.

Flag complex. Most of the previous methods described in Section 1.4 are for general simplicial complexes except [4, 45] which focus on the Rips complex, an example of a flag complex. In Chapter 4, we show that further decisive progress can be obtained if one restricts the family of simplicial complexes to flag complexes.

In the case of a Rips complex, the maximal simplices are the maximal cliques of the 1-skeleton of the complex and their computational time can be very large (exponential in the dimension of the complex). As a result, a naive application of our method would devote most of the time to compute the maximal faces of the complexes prior to their strong collapse. We show that we can avoid computing maximal cliques and we can strong collapse any flag complex using only the *1-skeleton* or graph of the complex. The strong collapses of a flag complex can be computed in time $O(n_v^2 k^2)$ where n_v is the number of vertices of the complex and k the maximal degree of its graph. Another crucial observation is that the reduced complex obtained by strong collapsing a flag complex is itself a flag complex.

Furthermore, if we consider a *flag tower*, i.e. flag complexes connected by simplicial maps, the obtained core sequence is also a flag tower. In the general case where the core sequence is not a filtration (which usually happens even if the original sequence is a filtration), we need to convert the flag tower to an equivalent filtration to compute its PH using known algorithms. To do so, we build on the work of [22, 34] again by restricting ourselves to flag complexes. This allows us to convert a flag tower to an equivalent flag filtration using again only the 1-skeleton.

The figure shows the flow diagram of our method when restricted to a flag filtration.



Further advantages of our approach for flag complexes are:

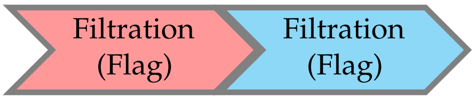
– We can compute PH for large complexes of high dimensions as we don't need to compute the maximal simplices.

– The dimension of the original sequence is in fact irrelevant and what matters is the dimension of the core sequence, which is usually quite small.

1.5.2 Edge Collapse Approach.

We extend the notion of strong collapse to *edge collapse*. Edge collapses appear to be very powerful, specially in the context of flag complexes. Keeping the same general philosophy we further improve on our approach using strong collapse by using edge collapse. We again use edge collapse to simplify a flag filtration $\mathcal{F}$ to another smaller flag filtration $\mathcal{F}^c$ with the same persistence.

The figure shows the flow diagram of our method using the edge collapse when applied to a flag filtration.



Our method with edge collapse or in fact more general simple collapse works for all kind of simplicial complexes. However we focus on the case of flag complexes as the reduction in this case can be performed using only the 1-skeletons of the complexes.

The approach with edge collapse has some new key features that makes it several orders of magnitude more efficient than all known methods including our strong collapse approach.

1. Edge collapses share some important properties with strong collapses. Most notably, we can use edge collapses to reduce flag filtrations $\mathcal{F}$ to smaller flag filtrations $\mathcal{F}^c$ with the same persistence, using only the 1-skeletons of the complexes.

2. The reduction is exact and the PH of the reduced sequence is identical to the PH of the input sequence. Our algorithm thus computes the exact PH as does [4] but differs from [45] where provably good approximations were computed. However, our algorithm using edge collapse can easily be adapted to compute approximate PH as well with larger gain in the computation time.

3. In the strong collapse approach, the reduced sequence associated to a filtration is usually a tower and part of the computing time was devoted to transforming this tower in another equivalent filtration. There is no such need in the algorithm using edge collapse, which is another main source of improvement.

4. The resulting method is simple and extremely efficient. On the theory side, we show that the edge collapse of a flag filtration can be computed in time $O(n\, n_c\, k^2)$, where n and n_c are the number of edges in the input and output 1-skeletons respectively and k is the maximal degree of a vertex in the input graph.

1.6 Outline

An outline of this thesis is as follows.

–In Chapter 2, we review the basic ideas and constructions related to topology, simplicial complexes, simple and strong collapses, homology, persistent homology. For more details on the topics of persistent homology readers can refer to [26] , for Algebraic topology to [31], for strong collapse to [3].

–In Chapter 3, we provide an algorithm to strong collapse a general simplicial complex. We then prove that strong collapse of a sequence of simplicial complexes preserves its persistence. We provide a scheme to approximate a Rips filtration and then we perform some experiments to show case the efficiency or our approach.

–In Chapter 4, we restrict our attention to flag complexes and build upon the work of Chapter 3. We show that the core of a flag complex is a flag complex and we can compute the core skeleton of a flag complex using only its 1-skeleton. The time complexity of the algorithm is $O(n_v^2 k^2)$ where n_v is the number of vertices of the complex and k the maximal degree of its graph. This is a major improvement over using maximal simplices. Then we use the same scheme as in Chapter 3 to compute the core sequence of a filtration which is usually a flag tower. We further build on the work of [22, 34] and provide an algorithm to convert a flag tower to an equivalent flag filtration using again only the 1-skeleton. We end with some experiments. The experiments show that the gain is tremendous when we work only with the 1-skeleton.

–In Chapter 5, we more generally define k-collapses that are identical to the *extended collapses* introduced in [2]. When $k = 0$, we have strong collapses and when $k = 1$ edge collapses and we use edge collapse to simplify a filtration. We show that edge collapse of a flag complex is also a flag complex. A complex with no possibilities of further edge collapse is called a 1-core. We show that, we can compute 1-*core* of a flag complex using only its skeleton. Furthermore, we show that a simplicial complex may have several 1-cores.

We then prove that simple collapse preserves the PH of a sequence of simplicial complex [2]. We then use edge collapse to simplify a flag filtration to another flag filtration again using only 1-skeleton. We provide some experiments and show that edge collapse outperforms even strong collapse and works on more general point set that strong collapse.

---

[2] This could be seen as an extension of the same result about strong collapse preserving the PH of sequence of simplicial complex.

Chapter 2

Basic Notions

In this chapter, we briefly review some of the basic concepts of Topology, notions like simplicial complex, homotopy equivalence, different notions of collapses, homology groups, persistent homology. Readers can refer to [31, 26, 42] for a comprehensive introduction of these topics.

In Section 2.1, we provide definitions of a topological space, continuous maps and define the notions of topological equivalences like homeomorphism and homotopic equivalence. In Section 2.2, notions of simplicial complex, simplicial maps and examples of commonly used simplicial complexes are discussed. In Section 2.3 we introduce the notion of simple collapses. Different notions of homology groups are defined in Section 2.4. We provide more details about Persistent Homology along with the matrix reduction algorithm in Section 2.5.

2.1 Topology

Topology is the branch of mathematics which studies the properties a space X under continuous deformation. More specifically it characterizes invariants that do not change when we deform the space continuously.

2.1.1 Topological Space

Definition 2.1.1. *A **topological space** is an ordered pair $(X, \mathcal{U})$, where X is a set and $\mathcal{U}$ is a collection of subsets of X, satisfying the following axioms:*

1. The empty set $\emptyset$ and X itself belong to $\mathcal{U}$.

2. Any arbitrary (finite or infinite) union of members of $\mathcal{U}$ is in $\mathcal{U}$ as well.

3. The intersection of any finite number of members of $\mathcal{U}$ still belongs to $\mathcal{U}$.

*The elements of $\mathcal{U}$ are called **open sets** and the collection $\mathcal{U}$ is called a **topology** on X. A subset A of X is called **closed** if $X \setminus A$ is an open set.*

Examples:

1. Given $X = \{a, b, c, d\}$, the collection $\mathcal{U} = \{\emptyset, \{a, b, c, d\}\}$ of only the two subsets of X forms a topology of X, called the trivial topology (indiscrete topology) on X.

2. Given $X = \{a, b, c, d\}$, the collection $\mathcal{U} = \mathcal{P}(X)$ (the power set of X), $(X, \mathcal{U})$ is a topological space. $\mathcal{U}$ is called the discrete topology on X.

3. Given $X = \mathbb{R}$, i.e. the set of real numbers, the set $\mathcal{U} = \{\bigcup(a,b) | \ a < b; a, b \in \mathbb{R}\}$ of union of open intervals in $\mathbb{R}$ is a topology on $\mathbb{R}$, called the *standard topology*.

A subset $A \subseteq X$ is called a **subspace** of $(X,\mathcal{U})$ if $A \cap \mathcal{U}$ is a topology on A, we call this an **induced topology** on A.

Standard topology on $\mathbb{R}^d$: The set of points $B(c,\epsilon) := \{x \in \mathbb{R}^d | \quad ||x - c|| \leq \epsilon\}$ is called a d-**ball** in $\mathbb{R}^d$, here c is the center and ϵ is the radius of the ball. If we replace the inequality $||x - c|| \leq \epsilon$ with the strict inequality $||x - c|| < \epsilon$, then we call the set an **open** d**- ball**, denoted as $B^o(c,\epsilon)$. The open ball extends the notion of the open interval (a,b) to the higher dimensional Euclidean space.

If $X = \mathbb{R}^d$ and $\mathcal{U}$ is the set of union of open balls in $\mathbb{R}^d$, then $\mathcal{U}$ defines a topology on $\mathbb{R}^d$ and is called the **standard topology** on $\mathbb{R}^d$.

For brevity we will write a topological space $(X,\mathcal{U})$, as X assuming some well defined topology $\mathcal{U}$ on X. We will also refer a topological space X by simply *space* unless it means something else.

The notion of open sets is an abstraction of the concept of closeness (distance) on a metric space and that in turn facilitates a more abstract definition of continuous maps.

Definition 2.1.2. *A map $f : X \to Y$ between two topological spaces is called **continuous** if for every open set V in Y the inverse $f^{-1}(V)$ is an open set in X.*

This definition of continuous map is then further used to define different notions of equivalence between two topological spaces.

2.1.2 Topological Equivalences

The most fundamental and strict form of equivalence between two topological spaces is called *homeomorphism*, which we define as follow.

Definition 2.1.3. *A continuous map $f : X \to Y$ between two topological spaces is a **homeomorphism** if it has the following properties:*

1 . f is a bijection (one-to-one and onto), i.e. there exist an inverse $f^{-1} : Y \to X$.

2. The inverse map f^{-1} is continuous.

If such a map exists, X and Y are called homeomorphic. Being homeomorphic is an equivalence relation on topological spaces.

Topology is precisely the study of properties of a topological space X which are preserved under homeomorphisms. Very often the topological spaces we consider are geometric objects, such as disks, sphere, $\mathbb{R}^d$ etc. Some properties that are preserved under homeomorphism are connectedness, dimension, number of holes/cavities...

Homeomorphism is a very strong notion of equivalence and when two spaces are homeomorphic they are topologically the same. However it is not an easy task to prove that two spaces are homoemorphic. This difficulty leads to a relatively relaxed notion of equivalence between topological spaces called *homotopy equivalence*.

Definition 2.1.4. *Let f and g be two continuous maps from a space X to a space Y. A **homotopy** between f and g is defined to be a continuous function $H : X \times [0,1] \to Y$ from* *the product of the space X with the unit interval $[0,1]$ to Y such that $H(x,0) = f(x)$ and $H(x,1) = g(x)$ for all $x \in X$.*

Intuitively H describes a continuous deformation of f into g. If such an H exists then f and g are said to be **homotopic**.

Definition 2.1.5. *Two spaces X and Y are called **homotopy equivalent**, if there exist continuous maps $f : X \to Y$ and $g : Y \to X$ such that $g \circ f$ is homotopic to the identity map id_X and $f \circ g$ is homotopic to id_Y.*

We call the maps f and g **homotopy equivalences**. Every homeomorphism is a homotopy equivalence, but the converse is not true. For example, a solid disk and a single point are homotopy equivalent however, the disk is not homeomorphic to the point. Next we look into a special type of homotopy equivalence called *deformation retraction*.

Definition 2.1.6. *Let X be a topological space and A a subspace of X. Then a continuous map $r : X \to A$ is a **retraction** if the restriction of r to A is the identity map on A; that is, $r(a) = a$ for all a in A.*

Definition 2.1.7. *A continuous map $F : X \times [0,1] \to X$ is a **deformation retraction** of a space X onto a subspace A if, for every $x \in X$ and $a \in A$,*

1. $F(x,0) = x$
2. $F(x,1) \in A$
3. $F(a,1) = a$.

Equivalently, a deformation retraction is a homotopy between a retraction and the identity map on X. The subspace A is called a **deformation retract** of X. As stated before a deformation retraction is a special case of a homotopy equivalence.

If a deformation retraction does not move the points in A throughout the homotopy then F is called a **strong deformation retraction**, that is we add an additional (4th) condition of $F(a,t) = a$ for all $t \in [0,1]$ and $a \in A$.

Definition 2.1.8. *A space is said to be **contractible**, if it is homotopy equivalent to a point.*

2.2 Simplicial complex

In this section, we will introduce the concept of *simplicial complexes* which are of utmost importance in computational topology. We begin with the geometric notion of simplicial complex. However later we will primarily work with the abstract simplicial complexes.

2.2.1 Geometric simplicial complex

Definition 2.2.1. *A **geometric** k-**simplex** σ is the convex hull of any $k+1$ affinely independent points $\{v_0, v_1, ...v_k\}$ in $\mathbb{R}^d$.*

Here k is called the **dimension** of σ and v_is are its vertices. For example, 0-simplex is a point, a 1-simplex is an edge, a 2-simplex is a triangle and a 3-simplex is a tetrahedron and so on and so forth. Since $\{v_0, v_1, ...v_k\}$ are affinely independent any smaller subset of vertices are also affinely independent, the convex hull τ of a subset of the set $\{v_0, v_1, ...v_k\}$ is then called a **face** of σ and σ is called a **coface** of τ.

Definition 2.2.2. *A* ***geometric simplicial complex*** *K is a set of geometric simplices that satisfies the following conditions:*

1. *Every face τ of a simplex $\sigma \in K$ is also in K.*
2. *The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in K$, is a face of both σ_1 and σ_2.*

Informally, a geometric simplicial complex is a set of simplices that are glued nicely, i.e. they only intersect each other at common faces.

Since K is a subset of $\mathbb{R}^d$ any well defined topology on $\mathbb{R}^d$ can easily be induced on K. When we consider K as a topological space, we will assume that the induced topology is the standard topology unless otherwise stated.

2.2.2 Abstract simplicial complex

Definition 2.2.3. *An* ***abstract simplicial complex*** *K is a collection of subsets of a non-empty finite set X, such that for every subset A in K, all the subsets of A are in K.*

An element of K is called a **simplex** (abstract). An element of cardinality $k + 1$ is called a k-simplex and k is called its **dimension**. A simplex is called **maximal** if it is not a proper subset of any other simplex in K. A sub-collection L of K is called a **subcomplex**, if it is a simplicial complex itself. L is a **full subcomplex** if it contains all the simplices of K that are spanned by the vertices (0-simplices) of the subcomplex L.

The abstract simplicial complexes are purely combinatorial objects. However, any abstract simplicial complex K can be realised as a geometric simplicial complex in $\mathbb{R}^d$ if d is sufficiently large. This can be easily seen through embedding a single abstract k-simplex σ in $\mathbb{R}^k$. We can always choose $k + 1$ affinely independent points in $\mathbb{R}^k$, mapping the vertices of σ to these $k + 1$ points we embed σ geometrically in $\mathbb{R}^k$, denoted as $|\sigma|$. A consistent choice of such points for all simplices in the abstract simplicial complex K will provide a geometric realisation of K in $\mathbb{R}^d$ for some d. The geometric realisation K will be denoted as $|K|$. $|K|$ itself is a geometric simplicial complex and can have the induced standard topology.

From now on we will call an *abstract simplicial complex* simply a *simplicial complex* or just a *complex*.

2.2.3 Examples of simplicial complexes

Now we will introduce some commonly used simplicial complexes.

Čech complex : Let P be a set of finite points in $\mathbb{R}^d$ and $B(P, \epsilon) := \{B(c, \epsilon) | \quad c \in P\}$ be the set of d-balls of some non-negative radius ϵ whose centers c are in P. The set of all non-empty intersections of such balls is an abstract simplicial complex and is called **Čech complex**, denoted as $C(P, \epsilon)$.

Vietoris-Rips (VR) complex : Let $\sigma \subseteq P$ be a subset of points in P such that any two points p, q in σ is at most 2ϵ far apart, that $||p - q|| \leq 2\epsilon$. The collection of all such subsets of P defines a simplicial complex called the **Vietoris-Rips (VR) complex** of the point set P at distance (scale) 2ϵ, denoted as $R(P, \epsilon)$. Vietoris-Rips (VR) complexes are often called simply Rips complexes.

Figure 2.1 is an example of a Čech (bottom left) and a Rips(bottom right) complex of a point set.

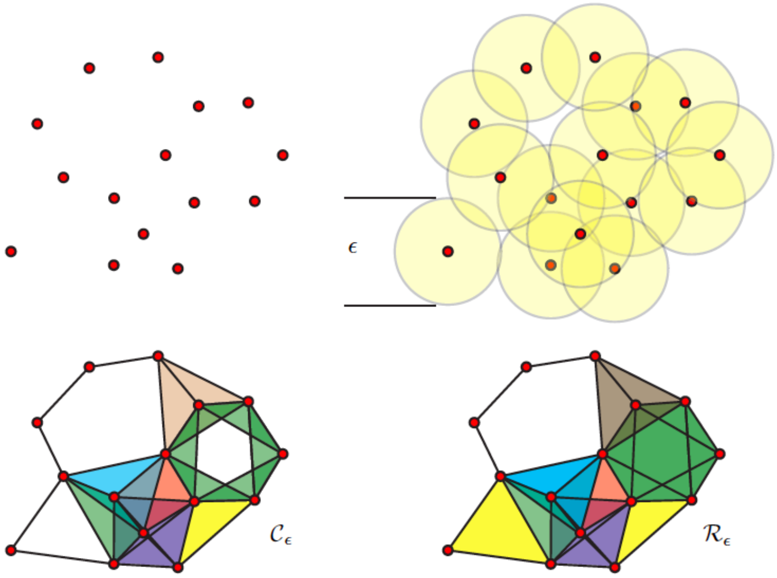


FIGURE 2.1 – On the left is the Čech complex and the right is the Rips complex.

It is not hard to see that the Čech complex $C(P, \epsilon)$ is a subcomplex of the Rips complex $R(P, \epsilon)$. In fact, they satisfy more stronger inclusion property,

$$R(P, \epsilon) \subseteq C(P, \sqrt{2}\epsilon) \subseteq R(P, \sqrt{2}\epsilon)$$

and are called **interleaved**.

When the point set P is clear from the context, for parameter(radius) ϵ, we will denote the Čech complex as C_ϵ and the Rips complex as R_ϵ.

2.2.4 Star, Link and Simplicial Cone

Definition 2.2.4. *Let σ be a simplex of a simplicial complex K, the* ***closed star*** *of σ in K, $st_K(\sigma)$ is a subcomplex of K which is defined as follows,*

$$st_K(\sigma) := \{\tau \in K | \ \tau \cup \sigma \in K\}.$$

Definition 2.2.5. *The* ***link*** *of σ in K, $lk_K(\sigma)$ is defined as the set of simplices in $st_K(\sigma)$ which do not intersect with σ,*

$$lk_K(\sigma) := \{\tau \in st_K(\sigma) | \tau \cap \sigma = \emptyset\}.$$

The **open star** of σ in K, $st^o_K(\sigma)$ is defined as the set $st_K(\sigma) \setminus lk_K(\sigma)$. It is not a subcomplex of K.

Let L be a simplicial complex and a a vertex not in L. Then the **simplicial cone** aL is defined as

$$aL := \{a, \tau \mid \tau \in L \ or \ \tau = \sigma \cup a; \text{ where } \sigma \in L\}.$$

2.2.5 Flag complex and Neighborhood

Definition 2.2.6. *A complex K is a* ***flag complex*** *if, when a subset of its vertices has pairwise edges (1-simplices) between them, they span a simplex of K.*

It follows that the full structure of K is determined by its 1-skeleton (or graph) we denote by G. The above defined Rips complex is an example of a flag complex.

For a vertex v in G, the **open neighborhood** $N_G(v)$ of v in G is defined as

$$N_G(v) := \{u \in G \mid [uv] \in E\},$$

here E is the set of edges of G.

The **closed neighborhood** $N_G[v]$ is

$$N_G[v] := N_G(v) \cup \{v\}.$$

Similarly we define the closed and open neighborhood of an edge $[xy] \in G$, $N_G[xy]$ and $N_G(xy)$ respectively as

$$N_G[xy] := N[x] \cap N[y] \text{ and}$$

$$N_G(xy) := N(x) \cap N(y).$$

The above definitions can be extended to any k-clique (k-simplex of K) $\sigma = [v_1, v_2, ..., v_k]$ of G;

$$N_G[\sigma] := \bigcap_{v_i \in \sigma} N[v_i] \text{ and}$$

$$N_G(\sigma) := \bigcap_{v_i \in \sigma} N(v_i).$$

2.2.6 Simplicial maps

Definition 2.2.7. *A vertex to vertex map $\psi : K \rightarrow L$ between two simplicial complexes is called a* ***simplicial map****, if it always maps a simplex in K to a simplex in L.*

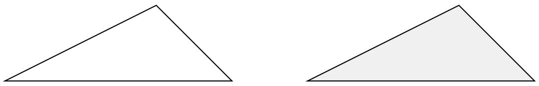
Since simplicial maps are determined by the images of the vertices they are finitely many.

A simplicial map $\psi : K \rightarrow L$ between two simplicial complexes K and L induces a continuous map $|\psi| : |K| \rightarrow |L|$ between the underlying geometric realizations.

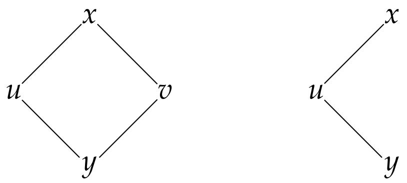
Any general simplicial map can be decomposed into more elementary simplicial maps, namely **elementary inclusions** (i.e., inclusions of a single simplex) and **elementary contractions** $\{\{u,v\} \mapsto u\}$ (where a vertex is mapped onto another vertex). The inverse operation of inclusion is called **simplicial removal** denoted as $K \hookleftarrow L$, where L is a subcomplex of K.

Here are examples of an elementary inclusion and an elementary contraction.

1. Elementary inclusion: inclusion of a single simplex (the solid triangle) $K_1 \overset{\cup\sigma}{\hookrightarrow} K_2$.



2. Elementary contraction: mapping a vertex v to another vertex u $K_1 \xrightarrow{\{u,v\}\mapsto u} K_2$.



Definition 2.2.8. *Two simplicial maps $\phi : K \to L$ and $\psi : K \to L$ are **contiguous** if, for all $\sigma \in K$, $\phi(\sigma) \cup \psi(\sigma) \in L$.*

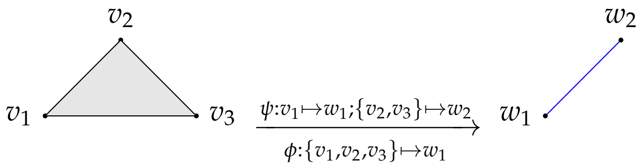

v_2
v_1
v_3
$\psi : v_1 \mapsto w_1; \{v_2, v_3\} \mapsto w_2$
$\phi : \{v_1, v_2, v_3\} \mapsto w_1$
w_2
w_1

FIGURE 2.2 – Maps ϕ and ψ are contiguous.

We denote two contiguous maps as $\phi \sim_c \psi$. Two contiguous maps are known to be homotopic [42, Theorem 12.5]. Two maps ϕ_1 and ϕ_2 are in the same **contiguity class** if there exists a sequence of simplicial maps $\phi_1 = \psi_0, \psi_1, ..., \psi_k = \phi_2$ such that the consecutive maps ψ_i and ψ_{i+1} are contiguous for all $i = 0...k$. We denote two maps in the same contiguity class as $\phi_1 \sim \phi_2$. Two maps in the same contiguity class are homotopic.

2.3 Collapses and Edge Contraction

In this section, we will review some of the basic notion of simplicial collapses. Collapses are combinatorial analog of deformation retractions defined in section 2.1.

2.3.1 Simple Collapse

Simple collapse is the most fundamental type of collapse. The notion of simple collapse was introduced by J.H.C. Whitehead in the late 1930s [47].

Definition 2.3.1. *Given a complex K, a simplex $\sigma \in K$ is called a **free simplex** if σ has a unique coface $\tau \in K$.*

The pair $\{\sigma, \tau\}$ is called a **free pair**.

Definition 2.3.2. *The action of removing a free pair: $K \to K \setminus \{\sigma, \tau\}$ is called an **elementary simple collapse.***

A series of such elementary simple collapses is called a **simple collapse**. We denote it as $K \searrow L$. This operation preserves the homotopy type of the simplicial complex K, which we write $K \sim L$. In particular, there is a retraction map $|r| : |K| \to |L|$ between the underlying geometric realization of K and L and there is a strong deformation retraction between $|r|$ and the identity over $|L|$. The inverse operation of (elementary) simple collapse is called an **(elementary) simple expansion**. Two complexes K and L are said to be **simple homotopic**, if there exists a series of elementary simple collapses and/or expansions between them. Figure 2.4 is an example of a simple collapse and expansion.

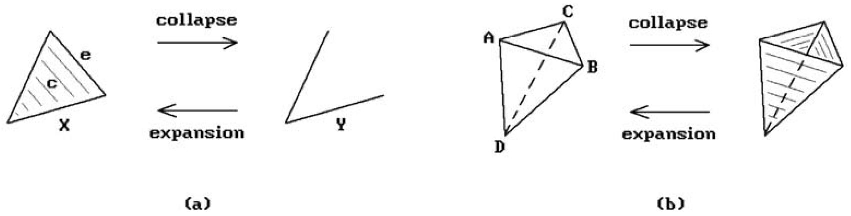


FIGURE 2.3 – Examples of simple collapses.

A complex K' will be called **simply-minimal** if there is no free pair $\{\sigma, \tau\}$ in K'. A subcomplex K^{ec} of K is called an **elementary core** of K if $K \searrow K^{ec}$ and K^{ec} is simply-minimal. The elementary cores are not unique and they depend on the order of removals of free pairs.

Definition 2.3.3. *A complex K is said to be **simply collapsible** or **collapsible**, if it simply collapses to a vertex.*

2.3.2 Edge Contraction

Related to the notion of collapses there is a notion called **Edge contraction** introduced by Dey et. al. in [21], which is a contraction of special edges that satisfies so called **link condition** defined below.

Definition 2.3.4. *An edge $[u, v]$ in a complex K is said to satisfy the link condition if $lnK([u, v]) = lnK(u) \cap lnK(v)$.*

Definition 2.3.5. *Given an edge $[u, v]$, the associated simplicial map $r_{[u,v]}$ defined by*

$$r(x) = x; \text{ if } x \notin \{u, v\},$$

$$r(x) = u; \text{ if } x \in \{u, v\}$$

is called an edge contraction.

The contraction operation merges the two vertices u, v to a single vertex either u or v. Given an edge $[u, v] \in K$ that satisfies the link condition, the associated retraction map $r_{[u,v]}$ preserves the homotopy type of the complex K.

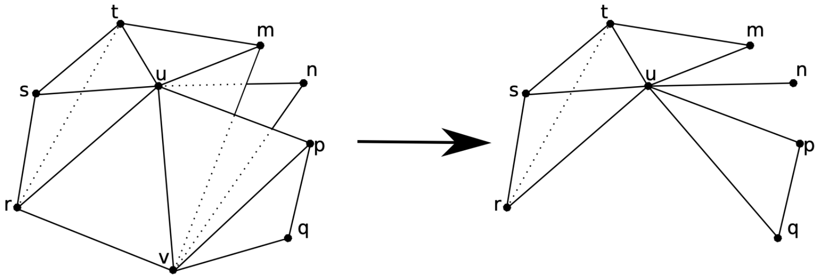


FIGURE 2.4 – The edge $[u, v]$ satisfies the link condition. The complex (in the right) which is the image of an edge contraction may not be a subcomplex of the original complex K. As the triangle $[p, q, u]$ doesn't exist in K.

2.4 Homology Groups

Intuitively *Homology groups* capture the number of connected components, number of holes, number of cavities and higher dimensional equivalents of a space. We associate a family of groups, one per dimension to a topological space. The homology groups are topological invariants in the sense, that two homotopic spaces have the isomorphic homology groups. There are many different types of homology theories, in this section we will focus on *simplicial homology* and briefly talk about *singular homology*.

2.4.1 Simplicial Homology

Let K be a simplicial complex and $\mathbb{G}$ be any abelian group. We start with defining *orientation* of the simplices in the complex K by fixing an order on the vertices of the complex K. A k-simplex σ with vertices $\{v_0, v_1, \cdots, v_k\}$ will be denoted as an ordered set $\sigma = [v_0, v_1, \cdots, v_k]$. A face τ of σ will inherit the order of the vertices from σ. A $(k-1)$ face of $\sigma = [v_0, v_1, \cdots, v_k]$ obtained by removing the vertex v_i from σ will be denoted as $[v_0, \cdots, \hat{v}_i, \cdots, v_k]$.

Definition 2.4.1. *For any non-negative integer p, the **p-th chain group** $C(K, \mathbb{G}, p)$, is the free abelian group defined using the p-simplices of K as basis and the elements of $\mathbb{G}$ as coefficients.*

Therefore, any element $c \in C(K, \mathbb{G}, p)$ can be written as the formal sum

$$\sum_{\alpha} g_\alpha \sigma_\alpha$$

Where $g_\alpha \in \mathbb{G}$ and σ_α is a p-simplex. c is called a **p-chain**. The summation $\oplus$ operation in $C(K, \mathbb{G}, p)$ is defined component-wise (at each basis element) using the summation operation of $\mathbb{G}$. For a fixed K and $\mathbb{G}$ we will write $C(K, \mathbb{G}, p)$ as C_p.

We define a **boundary homomorphism** between two consecutive chain groups $C_p \xrightarrow{\partial_p} C_{p-1}$ by specifying them on each p-simplex $\sigma_\alpha = [v_0, v_1, \cdots, v_p]$.

Definition 2.4.2 (boundary homomorphism). $\partial_p(\sigma_\alpha) = \sum\limits_i (-1)^i [v_0, \cdots, \hat{v}_i, \cdots v_p]$

Apart from being a homomorphism, ∂_p satisfies the following property (Lemma 2.4.1).

Lemma 2.4.1. $\partial_{p-1}(\partial_p(\sigma_\alpha)) = 0$.

Proof. The proof follows directly from the definition. □

We can link all the chain groups C_ps through the boundary homomorphisms ∂_p and we get the following sequence of abelian groups connected through homomorphisms,

$$\cdots \to C_{p+2} \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \to \cdots C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0.$$

with the property $\partial_{p-1}\partial_p = 0$. Such a sequence of chain groups linked by boundary homomorphisms is called a **chain complex**. Observe that the sequence has been extended by 0 at the end with a zero boundary map, $\partial_0 = 0$.

Definition 2.4.3. *A p-chain $c \in C_p$ is called a p-**boundary** if for there exists a $(p+1)$-chain $\gamma \in C_{p+1}$, the following is true,*

$$\partial_{p+1}(\gamma) = c.$$

We denote the set of p-boundaries by $B_p = Im\partial_{p+1} := \partial_{p+1}(C_{p+1})$.

Definition 2.4.4. *A p-chain $c \in C_p$ is called a p-**cycle** if $\partial_p(c) = 0$.*

The set of p-cycles will be denoted by $Z_p = Ker\partial_p$. It is not hard to see that, Z_p and B_p both are subgroups of C_p.

Due the property $\partial_{p-1}(\partial_p(\sigma_\alpha)) = 0$, it directly follows that $B_p \subseteq Z_p$.

Definition 2.4.5. *The p-th **homology group** $H_p(K)$ of a chain complex is defined as the quotient group Z_p/B_p.*

It is again an easy exercise to check that $H_p(K)$ is in fact a group. The operation of taking quotient establishes an equivalence relation between elements of Z_p, where two p-cycles $c_1, c_2 \in Z_p$ are equivalent if they differ by a p-boundary, that is, $c_1 = c_2 + b_1$; for some $b_1 \in B_p$. Therefore the elements of H_p are the equivalence classes of Z_p. Two p-cycles representing the same homology class are called **homologous**.

When we choose elements from a field $\mathbb{F}$ as coefficients instead of a group $\mathbb{G}$ to define the chain groups C_p, that is $C_p = C(K, \mathbb{F}, p)$, then the homology groups $H_p(K)$ are in fact vector spaces. This is a mere consequence of the fact that $C(K, \mathbb{F}, p)$ is itself a vector space. In computational topology, $\mathbb{Z}_2$ (Z mod 2) is the most common used set of coefficients.

Definition 2.4.6. *The **p-th Betti number** β_p is defined as the rank of the p-th homology group, $\beta_p = rank(H_p)$.*

The Betti numbers can be written as the difference between the rank of Z_p and the rank of B_p.

$$\beta_p = rank(Z_p) - rank(B_p).$$

The definition of simplicial homology is purely combinatorial and algebraic. It does not use any topological structure of the simplicial complex. However, homology groups are topological invariants. Indeed, they are isomorphic for homeomorphic topological spaces and in fact they are isomorphic for homotopic spaces. The following lemma is the first fact that establishes the invariant nature of homology groups.

Lemma 2.4.2. *Let $f : K \to L$ be a simplicial map between two simplicial complexes K and L. f induces a homomorphism between the corresponding homology groups of K and L, that is, there exists a map $f^* : H_p(K) \to H_p(L)$, induced by f for all p.*

Proof. We will provide a sketch of the proof. Using f we first define a homomorphism between the pth chain groups $f_\# : C_p(K) \to C_p(L)$. For each p-simplex $\sigma \in K$, we define $f_\#(\sigma) = f(\sigma)$ if $dim(f(\sigma)) = p$, and $f_\#(\sigma) = 0$ otherwise. Then, for

$$c \in C_p(K)$$

$$c = \sum_\alpha g_\alpha \sigma_\alpha$$

$$f_\#(c) := \sum_\alpha g_\alpha f(\sigma_\alpha)$$

$f_\#$ satisfies the following property,

$$f_\# \partial = \partial f_\#$$

Where ∂ is the respective boundary homomorphisms of $C_p(L)$ and $C_p(K)$. The property $f_\# \partial = \partial f_\#$ implies that $f_\#$ takes cycles to cycles and boundaries to boundaries. Therefore, it induces a homomorphism $f^* : H_p(K) \to H_p(L)$ on the corresponding homology groups as well. □

The above defined map $f_\#$ is called a **chain map** as it satisfies $f_\# \partial = \partial f_\#$.

2.4.2 Singular Homology

Simplicial homology is one of the most traditional definition of homology, which is defined for simplicial complexes. However using the notion of *singular simplices* (defined later) it is possible to have a more general notion of homology that extends to any general topological spaces.

Let $\Delta^p = [v_0, \cdots, v_p]$ denote a **standard** p**-simplex**, whose vertices v_i are the unit vectors along the coordinate axes of $\mathbb{R}^d$.

Definition 2.4.7. *A* ***singular*** p***-simplex*** *in a topological space X is a continuous map $\sigma^o : \Delta^p \to X$.*

Using this basic definition of a singular simplex we can extend all the notions like chain complex, boundary maps, homology groups etc.

Definition 2.4.8. *For any non-negative integer p, the* ***singular*** p***-th chain group*** $C^o(X, \mathbb{G}, p)$, *is the free abelian group defined using the singular p-simplices of X as basis and the elements of $\mathbb{G}$ as coefficients.*

We will write C^o_p for $C^o(X, \mathbb{G}, p)$ when X and $\mathbb{G}$ are clear from the context.

We define the boundary homomorphism between two consecutive singular chain groups C^o_p and C^o_{p-1} implicitly by the boundary homomorphism on the corresponding standard simplex. Below is the definition of a singular boundary homomorphism over a singular p-simplex which can be extended to any **singular p-chain** $c \in C^o_p$ componentwise.

Let $\sigma^o_\alpha|[v_0, \cdots, \hat{v}_i, \cdots v_p]$ denotes the restriction of the map $\sigma^o_\alpha : \Delta^p \to X$ to the face $[v_0, \cdots, \hat{v}_i, \cdots v_p]$ of Δ^p.

Definition 2.4.9 (singular boundary homomorphism).

$$\partial^o_p(\sigma^o_\alpha) := \sum_\alpha (-1)^i \sigma^o_\alpha|[v_0, \cdots, \hat{v}_i, \cdots v_p].$$

∂^o_p also follows the same properties of being a 'boundary' homomorphism, that is, $\partial^o_{p-1}(\partial^o_p(\sigma^o_\alpha)) = 0$. Therefore we can define the **p-th singular homology** $H^o_p(X) = Ker\partial^o_p / Im\partial^o_{p+1}$.

In the case of simplicial homology, a simplicial map between two simplicial complexes induces a homomorphism between their homology groups. we have the following analog for singular homology

Lemma 2.4.3. *If $f : X \to Y$ be a continuous map between two spaces X and Y, then f induces a homomorphism between the corresponding singular homology groups of X and Y. In other words, there exists a map $f^* : H^o_p(X) \to H^o_p(Y)$, induced by f for all p.*

Using the above lemma and some of its consequences we have the following homotopy invariant property of singular homology groups.

Theorem 2.4.1. *Let $f : X \to Y$ be a continuous map between two spaces X and Y such that f is a homotopy equivalence. Then $f^* : H^o_p(X) \to H^o_p(Y)$ is an isomorphism for all p.*

With the definition of singular homology we can have a homology theory for any topological space. Furthermore, there is a remarkable fact that singular homology and simplicial homology of a simplicial complex K are isomorphic.

Theorem 2.4.2. *$H_p(K)$ and $H^o_p(|K|)$ are isomorphic for all p.*

The implicit consequence of the above theorem is that for any *triangulated* space X its simplicial homology does not depend on that particular triangulation. As a consequence of the above theorem and lemmas, we have the following homotopy invariant property of simplicial homology groups.

Theorem 2.4.3. *Let $f : K \to L$ be a simplicial map between two simplicial complexes K and L such that $|f| : |K| \to |L|$ is a homotopy equivalence. Then $f^* : H_p(K) \to H_p(L)$ is an isomorphism for all p.*

The simplicial chain group of a finite simplicial complex is finitely generated since it has finitely many simplices and therefore a finite basis. However even for a *compact* ('finite') space X there is an infinite number of continuous maps from Δ^p to X and therefore an infinite number of singular p-simplices. This implies that a *singular p-th chain group* is infinitely generated. The definition of singular homology is more widely applicable and mathematically more powerful, however due to the infinite number of singular simplices, it is computationally intractable. Therefore for all practical purposes we use simplicial homology as our main tool.

2.5 Persistent Homology

Persistent Homology (PH) is a new child of the traditional homology theories, a dynamic variant. Imagine a topological space, for example, a simplicial complex which is evolving with time (e.g. growing with inclusion of more simplices) and you want to keep track of the evolution using homological measurements. Persistent homology is precisely a tool for such a measurement. The advent of Persitent homology was to understand the topology of data, usually represented as a point cloud P in some euclidean space $\mathbb{R}^d$. The underlying assumption is that the point cloud P is a finite sample of some topological space X. The goal of persistent homology is then to infer the topological (more specifically homological) information of X through P.

Now since P is just a finite sample of X, we can not get the true homological information of X by computing the homology groups of P. One way to solve this problem and approximate the space X would be to inflate the points in P as solid d-balls of some radius ϵ. And if we choose the right parameter ϵ the union of balls $P^\epsilon := \bigcup_{p \in P} B(p, \epsilon)$ will be a 'good' cover of X and then we compute the homology group of P^ϵ. However there are two problems associated with this approach, the first problem is how to choose the right parameter and the second and more fundamental problem is that there might not be a single parameter providing the right homology. Figure 2.5 is an illustration of this problem.

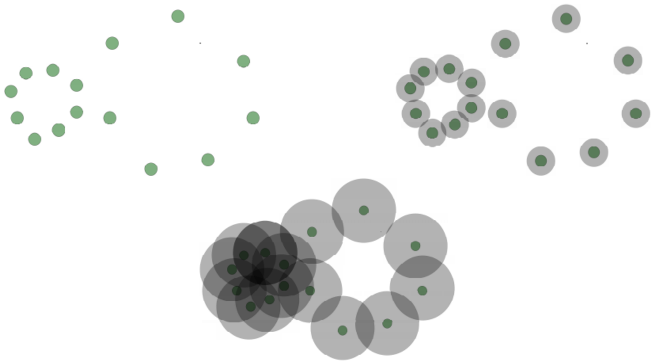


FIGURE 2.5 – At no scale the union of balls can capture both the loops.

Persistent homology solves both of these problems, by computing homology at multiple scales and provides information about the many 'right' scales that correspond to some topological feature of the data. Mathematically, there are several different ways to characterize the evolution of a space. We will start with the simplest version of it, which is called a *filtration*.

2.5.1 Sequences of complexes

Definition 2.5.1. *A **filtration** $\mathcal{F}$ is a sequence of nested simplicial complexes*

$$\mathcal{F} : K_1 \subseteq K_2 \subseteq \cdots \subseteq K_m.$$

A filtration could also be defined as a function $f : K \to \mathbb{R}$ over the final simplicial complex $K = K_m$. Where f assigns a real value to each simplex σ in K such that if $\tau \subseteq \sigma$ is a face of σ then $f(\tau) \leq f(\sigma)$.

The most natural example of a filtration is a filtration coming out of the union of balls P^ϵ and the Čech complex C_ϵ is precisely the desired simplicial complex. For increasing values of $\epsilon > 0$, the Čech complexes $\{C_\epsilon\}_{0 \leq \epsilon}$ forms a filtration. Also, the Rips complexes $\{R_\epsilon\}_{0 \leq \epsilon}$ with increasing scale parameter forms a filtration. Illustration of a filtration, Figure 2.6.

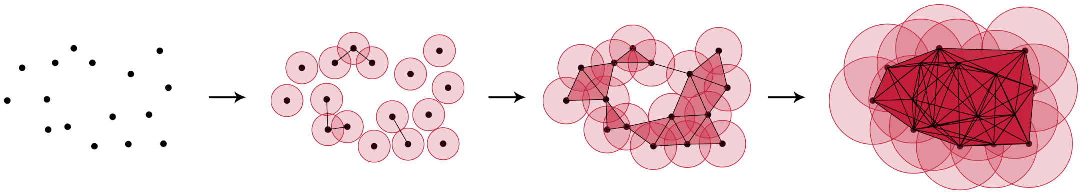


FIGURE 2.6 – A sequence of simplicial complexes connected through inclusions.

Filtrations can be further generalized to a *tower*.

Definition 2.5.2. *A sequence of simplicial complexes*

$$\mathcal{T} : K_1 \xrightarrow{f_1} K_2 \xrightarrow{f_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(m-1)}} K_m.$$

*connected through simplicial maps f_i is called a **simplicial tower** or simply a **tower**.*

This could be even further generalized to a *zigzag sequence*.

Definition 2.5.3. *A sequence of simplicial complexes*

$$\mathcal{Z} : K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(m-1)}} K_m.$$

*connected through simplicial maps f_i and g_is in both directions is called a **zigzag sequence**.*

When the maps f_i and g_i are inclusions, a zigzag sequence is called **zigzag filtration**.

A tower is then a special case of zigzag sequence, when the simplicial maps are only in one direction and a filtration is a special case of tower where the simplicial maps are only inclusions.

2.5.2 Persistent homology.

Although Persistent homology could be defined for zigzag sequences, we will restrict our exposition mostly to towers and filtrations.

If we compute the homology classes of all the K_i with coefficients from a field $\mathbb{F}$, we get the sequence

$$\mathcal{P}(\mathcal{T}) : H_p(K_1) \xrightarrow{f_1^*} H_p(K_2) \xrightarrow{f_2^*} H_p(K_3) \xrightarrow{f_3^*} \cdots \xrightarrow{f_{(m-1)}^*} H_p(K_m).$$

Here f_i^* is the homomorphism induced from f_i. $\mathcal{P}(\mathcal{T})$ is a sequence of vector spaces connected through the homomorphisms f_i^* and is called a **persistence module**. More formally, a *persistence module* $\mathbb{V}$ is a sequence of vector spaces $\{V_1 \to V_2 \to V_3 \to \cdots \to V_m\}$ connected through homomorphisms $\{\to\}$ between them. A persistence module arising from a sequence of simplicial complexes captures the evolution of the topology of the sequence.

For two integers b and d, $1 \leq b \leq d \leq n$, we can define an **interval module** $\mathbb{I}[b, d]$ by assigning V_i to $\mathbb{F}$ when $i \in [b, d]$, and to the null space otherwise. The maps between any two $\mathbb{F}$ vector spaces are identity and are zero otherwise. For example $\mathbb{I}[2, 4] : \{0 \xrightarrow{0} \mathbb{F} \xrightarrow{I} \mathbb{F} \xrightarrow{I} \mathbb{F} \xrightarrow{0} 0 \xrightarrow{0} 0\}$, here $n = 6$.

The following theorem states that any persistence module can be *decomposed* into a collection of intervals of the form $[i, j]$ [12, 50].

Theorem 2.5.1. *Any persistence module* $\mathbb{V}$ *is isomorphic to a direct sum of interval modules,*

$$\mathcal{P}(\mathcal{T}) \cong \bigoplus \mathbb{I}[i, j]$$

The multiset of all the intervals $[i, j]$ in this decomposition is called the **persistence diagram** of the persistence module. An interval of the form $[i, j]$ in the persistence diagram of $\mathcal{P}(\mathcal{T})$ corresponds to a homological feature (a 'cycle' of dimension p) which is born at i and died at j.

2.5.3 Computing Persistence Diagram

In this subsection, we will discuss an algorithm to compute the persistence diagram of a persistence module induced by a filtration function f on a complex K with coefficients Z_2.

Let $\{\sigma_1, \sigma_2, \cdots, \sigma_m\} = K$ be the set of ordered (by filtration value) simplices of K.

Notice that as we move in increasing order of simplices, each simplex either creates a homological feature ('cycle') or destroys one. The simplex which creates a cycle is called a **positive** simplex and the one which destroys a cycle is called **negative** simplex. Therefore, an algorithm which computes the persistence diagram of a filtered simplicial complex K should essentially compute the pairings [positive, negative] of simplices corresponding to each homological feature.

We set up the following boundary matrix, ∂,

$$\partial[i, j] = 1 \text{ if } \sigma_i \subset \sigma_j \text{ and } \dim(\sigma_i) = \dim(\sigma_j) \text{ -1;}$$

$$\partial[i, j] = 0 \text{ otherwise.}$$

Therefore, the rows and columns of ∂ have the same order as the simplices and its columns stores the boundary of a simplex. Also ∂ is an upper-triangular matrix since a face simplex appears before its cofaces. We reduce ∂ to another matrix R

using column operations so that we get the desired pairings of positive and negative simplices.

Let $low(j)$ be the maximum value of the non-zero row index in column j (i.e. the index of the lowest non-zero row) and let it undefined for a zero column. The algorithm reduces ∂ by adding columns from left to right and it is reduced if $low(j) \neq low(j')$ whenever $j \neq j'$ for any two non-zero columns.

Algorithm 1 Reduction Algorithm

```
procedure REDUCE($\partial$)                          ▷ $\partial$ is the boundary matrix.
    $R = \partial$;
    for j = 1 to m do
        while there exists $j' < j$ with $low(j') = low(j)$ do
            add column $j'$ to column $j$                 ▷ mod 2 operation.
        end while
    end for
end procedure                                     ▷ Return $R$
```

We discuss a few properties of the matrix R.

Each zero column of R corresponds to a positive simplex, and the number of zero columns corresponding to the p-simplices of K is the rank of Z_p. Each non-zero column corresponds to a negative simplex, and the number of non-zero columns corresponding to the p-simplices of K is the rank of B_p. This gives us the rank of the homology classes of K as $\beta_p = rank(Z_p) - rank(B_p)$. However to compute the persistent homology we need to find the appropriate pairings of positive and negative simplices. Interestingly the matrix R does contain this information.

To compute the pairings we consider a zero column i, as mentioned before σ_i is a positive simplex and therefore we look for its negative partner. We look at the corresponding row, i for the negative partner simplex. There are two possibilities:

1. There exists a column j, such that $low(j) = i$. Then σ_j is said to be paired with σ_i and therefore $[i, j]$ or with the filtration values $[f(\sigma_i), f(\sigma_j)]$ is a point in the persistence diagram of $H_p(K)$, where $dim(\sigma_i) = p$.

2. There exists no column j, such that $low(j) = i$. Then σ_i remains un-paired, so σ_i corresponds to a class that was created but was never destroyed and this gives a point $[i, \infty)$ or $[f(\sigma_i), \infty)$ in the persistence diagram of $H_p(K)$, where $dim(\sigma_i) = p$.

Although the reduced matrix R is not unique, the above properties are true for any matrix reduced from δ. The runtime complexity of the algorithm is $\mathcal{O}(m^3)$. However, in most practical cases, the runtime is usually near-linear in m as ∂ is usually sparse.

2.5.4 Equivalence and Stability of Persistence Modules

The following theorem states a condition under which two persistence module will be equivalent [12, 20].

Theorem 2.5.2. *Two different persistence modules* $\mathbb{V} : \{V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_m\}$ *and* $\mathbb{W} : \{W_1 \rightarrow W_2 \rightarrow \cdots \rightarrow W_m\}$*, connected through a set of homomorphisms* $\phi_i : V_i \rightarrow W_i$ *are* ***equivalent*** *if the* ϕ_i *are isomorphisms and the following diagram commutes.*

$$\begin{tikzcd}
V_1 \arrow[r] \arrow[d, "\phi_1"] & V_2 \arrow[d, "\phi_2"] & \cdots \arrow[r] & V_{m-1} \arrow[r] \arrow[d, "\phi_{m-1}"] & V_m \arrow[d, "\phi_m"] \\
W_1 \arrow[r] & W_2 & \cdots \arrow[r] & W_{m-1} \arrow[r] & W_m
\end{tikzcd}$$

Equivalent persistence modules have the same interval decomposition, hence the same diagram.

For simplicity we have chosen the set of natural numbers as our index set, that is the vector spaces V_i in $\mathbb{V}$ are indexed over $\mathbf{N}$ and therefore the intervals are also natural numbers. However, the definition works well for any index set $J \subseteq \mathbb{R}$. There exists more approximate notions of similarity between two persistence modules.

Bottleneck distance. If $J \subseteq \mathbb{R}$ is the chosen index set then a persistence diagram is a multiset of points in the extended plane, $\bar{\mathbb{R}}^2$. Clearly, the persistence diagram consists of points above the diagonal (the line $x = y$). To have a well defined notion of distance between two persistence diagrams, we add the points of the diagonal to the diagram, each with infinite multiplicity. We use the L_∞ norm to define distance between two points $v = (v_1, v_2)$ and $w = (w_1, w_2)$ in $\bar{\mathbb{R}}^2$, that is,

$$\|v - w\|_\infty = \max\{|v_1 - w_1|, |v_2 - w_2|\}.$$

Let $dgm(\mathbb{V})$ and $dgm(\mathbb{W})$ be two persistence diagrams. Let us consider bijections $\eta : dgm(\mathbb{V}) \to dgm(\mathbb{W})$ then the **bottleneck distance** $d_B(dgm(\mathbb{V}), dgm(\mathbb{W}))$ between the diagrams is defined as follows,

Definition 2.5.4 (bottleneck distance).

$$d_B(dgm(\mathbb{V}), dgm(\mathbb{W})) = \inf_{\eta} \sup_{v \in dgm(\mathbb{V})} \|v - \eta(v)\|_\infty.$$

Therefore, it is an infimum over all possible η.

The problem of computing the bottleneck distance between two persistence diagrams can be reduced to a matching problem of a bipartite graph. Assuming that the persistence diagrams consist of finitely many off-diagonal points with finite multiplicity and all the diagonal points with infinite multiplicity. In this case, computing $d_B(dgm(\mathbb{V}), dgm(\mathbb{W}))$ reduces to a bipartite graph matching problem.

Let $X = dgm(\mathbb{V})$ and $Y = dgm(\mathbb{W})$ then X_0 and Y_0 respectively denote the off-diagonal points of X and Y. If $a = (x, y)$ is an off-diagonal point, then the closest point to a on the diagonal is its orthogonal projection on the diagonal, denoted as $a' = (\frac{(x+y)}{2}, \frac{(x+y)}{2})$. Let X'_0 denote the set of all projections of X_0 and similarly Y'_0 denote the set of all projections of Y_0.

We define $M = X_0 \cup Y'_0$ and $N = X'_0 \cup Y_0$; they both have the same cardinality. Then computing bottleneck distance is computing a matching in the weighted complete bipartite graph, $G = (M \cup N, M \times N, c)$ and the weights are given by the following weight function

$$c(a, b) = \|a - b\|_\infty; \text{ if either } a \text{ or } b \text{ or both are not on the diagonal,}$$

$$c(a, b) = 0; \text{ Otherwise}$$

The points in the $dgm(\mathbb{V})$ can be transformed to log scale by mapping a point $(v_1, v_2) \in dgm(\mathbb{V}) \in \mathbb{R}^2$ to $(\log(v_1), \log(v_2)) \in \mathbb{R}^2$ and we denote this log transform on $dgm(\mathbb{V})$ as $\log(dgm(\mathbb{V}))$. The bottleneck distance on log scale is then defined as $d_B(\log(dgm(\mathbb{V})), \log(dgm(\mathbb{W})))$.

Stability. Stability of persistence diagrams is the hallmark result in the theory of persistence, it provides a context to the persistence theory. There are several different variants of the stability theorem. The following Theorem 2.5.3 states that if there are two close filtrations on a simplicial complex K then their persistence diagram would be close as well.

Theorem 2.5.3. *Let K be a simplicial complex and $f, g : K \to R$ two filtration functions on K. Let $\mathbb{V}_f$ and $\mathbb{V}_g$ be two persistence modules associated with the filtrations f and g respectively, then the bottleneck distance between $dgm(\mathbb{V}_f)$ and $dgm(\mathbb{V}_g)$ is bounded from above by the L_∞-distance between f and g, that is,*

$$d_B(dgm(\mathbb{V}_f), dgm(\mathbb{V}_g)) \leq \|f - g\|_\infty$$

where $\|f - g\|_\infty = \sup\limits_{\sigma \in K} |f(\sigma) - g(\sigma)|$

One of the application of the above stability theorem is that given a point set P, the bottleneck distance on log scale between the persistence diagrams of the Čech filtration and the Rips filtration is bounded. We make this more precise as follows.

Theorem 2.5.4. *Let $P \subset \mathbb{R}^d$ be a point set and $\{C_\epsilon\}$ and $\{R_\epsilon\}$ be the filtrations of Čech and Rips complexes for increasing values of $0 \leq \epsilon \in \mathbb{R}$, then*

$$d_B(\log(dgm(C_\epsilon)), \log(dgm(R_\epsilon))) \leq \sqrt{2}$$

Proof. The Čech and the Rips filtrations individually define two filtration functions f and g on the final simplicial complex K which is a $|P|$-simplex. As mentioned before since these filtrations are interleaved, with the following inclusions,

$$R(P, \epsilon) \subseteq C(P, \sqrt{2}\epsilon) \subseteq R(P, \sqrt{2}\epsilon)$$

the corresponding filtration functions satisfy,

$$\|\log(f) - \log(g)\|_\infty \leq \sqrt{2}$$

and therefore Theorem 2.5.3 implies,

$$d_B(\log(dgm(C_\epsilon)), \log(dgm(R_\epsilon))) \leq \sqrt{2}.$$

□

Another implicit consequence of the stability theorem is that if the point set P is transformed to P' with small perturbations then the persistence of diagram of the perturbed point changes with small amount, bounded above by the amount of perturbation.

2.6 Conclusion

In this chapter, we reviewed some essential notions of topology and in particular of persistent homology and briefly explained a simple algorithm to compute the persistence diagram of a filtration.

As discussed in Section 1.4 of Chapter 1, recently there are many improved algorithms to compute the persistence diagram of filtrations by exploiting the sparsity and the structure of the boundary matrix. Another variant is to compute persistence cohomology, which is dual to homology.

All these algorithm focuses on the faster reduction of the boundary matrix. The second approach for faster computation of the persistence diagram is to simplify the input filtration to a smaller filtration by preserving the persistence or approximating it by bounded bottleneck distance. Our work in this thesis is along the second approach we uses strong collapses and edge collapses to simplify the input filtration.

In Section 1.3 of Chapter 1, we have mentioned the extensions of persistent homology to more general sequences like towers and zigzag sequences. We also mention new techniques to compute persistence diagrams of towers as well, usually by converting the tower to an equivalent filtration by simplicial expansion (conning) and then compute the persistence of the equivalent filtration [22, 34]. We have made further advancements in the special case of the flag tower (tower of flag complexes) based on the previous works [22, 34], which we will explain in more details in Chapter 4.

There are algorithms to compute zigzag persistence of zigzag filtration [12, 13]. The usual technique to convert a tower to an equivalent filtration works for the conversion of a general zigzag sequence to an equivalent zigzag filtration. And this facilitates the computation of the zigzag persistence of any general zigzag sequence.

Chapter 3

Strong Collapse And Persistence

In this chapter, we introduce our first approach to simplify the complexes of the input sequence using the notion of strong collapse. As described before in Chapter 1, Given a zigzag sequence

$$\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$$

of simplicial complexes K_i connected through simplicial maps $\{\xrightarrow{f_i} \text{ or } \xleftarrow{g_j}\}$, we independently strong collapse the complexes of the sequence to reach a sequence

$$\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\},$$

with *induced simplicial maps* $\{\xrightarrow{f_i^c} \text{ or } \xleftarrow{g_j^c}\}$ (defined in Section 3.4).

K_i^c is the **core** of the complex K_i and the sequence $\mathcal{Z}^c$ is called the **core sequence** of $\mathcal{Z}$. In Section 3.4, we prove that $\mathcal{Z}$ and its core sequence $\mathcal{Z}^c$ are equivalent.

We introduce the notion of strong collapse in Section 3.1. In Section 3.2 we define the notion of nerve of a simplicial complex and show its relationsip with the strong collapse. We describe an algorithm to compute the core K^c of a general simplicial complex K using strong collapses. The algorithm uses a representation of K that consists of only the maximal simplices of K. It is based on the nerve-square $\mathcal{N}^2(K)$ construction defined in Section 3.2.

We then use our algorithm to compute the core sequence by independently strong collapsing the constituent complexes of the sequence. We explain an approximation scheme that compromizes between precision and time by choosing the number of simplicial complexes of the sequence we strong collapse. In Section 5.4, we perform some experiments on publicly available data sets, which shows that our approach is extremely fast and memory efficient in practice.

3.1 Strong Collapse

The notion of *strong collapse* is relatively new and it was introduced by Barmak and Minian [3]. It is a special type of simple collapse and is a stronger notion than simple collapse.

Definition 3.1.1. *A vertex v in K is called a* ***dominated vertex*** *if the link of v in K, $lk_K(v)$ is a simplicial cone, that is, there exists a vertex $v' \neq v$ and a subcomplex L in K, such that $lk_K(v) = v'L$.*

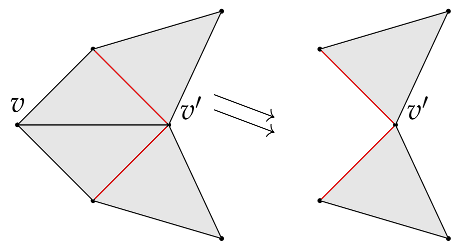


FIGURE 3.1 – Illustration of an *elementary strong collapse*. In the complex on the left, v is dominated by v'. The link of v is highlighted in red. Removing v leads to the complex on the right.

We say that the vertex v' is *dominating* v and v is *dominated* by v'. The symbol $K \setminus v$ (deletion of v from K) refers to the subcomplex of K which has all the simplices of K except the ones containing v.

Removal of a simplex. We denote by $K \setminus \sigma$ the subcomplex of K obtained by removing σ, i.e. the complex that has all the simplices of K except the faces and the cofaces of σ.

Below is an important remark from [3, Remark 2.2], which proposes an alternative definition of dominated vertices.

Remark 3.1.1. *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' [3].*

Definition 3.1.2. *An* ***elementary strong collapse*** *is the deletion of a dominated vertex v from K, which we denote with $K \searrow\searrow^{e} K \setminus v$.*

Figure 3.1 illustrates an easy case of an elementary strong collapse.

Definition 3.1.3. *There is a* ***strong collapse*** *from a simplicial complex K to its subcomplex L, if there exists a series of elementary strong collapses from K to L, denoted as $K \searrow\searrow L$.*

The inverse of a strong collapse is called a **strong expansion**. If there exists a combination of strong collapses and/or strong expansion from K to L then K and L are said to have the same **strong homotopy type**.

The notion of strong homotopy type is stronger than the notion of simple homotopy type in the sense that if K and L have the same strong homotopy type, then they have the same simple homotopy type (simple homotopic), and therefore the same homotopy type [3]. The converse is not necessarily true, there are examples of contractible or simply collapsible simplicial complexes that are not strong collapsible.

A complex without any dominated vertex will be called a **minimal complex**. A **core** of a complex K is a minimal subcomplex $K^c \subseteq K$, such that $K \searrow\searrow K^c$. *Every simplicial complex has a* ***unique core*** *up to isomorphism. The core decides the strong homotopy type of the complex*, and two simplicial complexes have the same strong homotopy type *if and only if* they have isomorphic cores [3, Theorem 2.11].

Retraction map: If a vertex $v \in K$ is dominated by another vertex $v' \in K$, the vertex map $r : K \to K \setminus v$ defined as: $r(w) = w$ if $w \neq v$ and $r(v) = v'$, induces a simplical map that is a **retraction** map. The homotopy between r and the identity $i_{K\setminus v}$ over $K \setminus v$ is in fact a strong deformation retraction. Furthermore, the composition $(i_{K\setminus v})r$ is contiguous to the identity i_K over K [3, Proposition 2.9].

Existence of a simplicial map (the retraction map r) corresponding to a strong collapse is a special property of strong collapse, which is not true in the case of simple collapse. For example in Figure 2.4 (a) there are no simplicial map corresponding to the removal of the free pair (the triangle and the edge).

3.2 Nerve and Strong Collapse

One of the most remarkable property of strong collapses is its association with the operation of *nerve* of a simplicial complex. We will first begin with the definition of a *cover* and the general definition of a nerve.

Definition 3.2.1. *A closed* ***cover*** $\mathcal{U}$ *of a topological space* $\mathcal{X}$ *is a set of closed sets of* $\mathcal{X}$ *such that* $\mathcal{X}$ *is equal to their union.*

Here we choose closed sets to define a cover, if we choose open sets we define an open cover where $\mathcal{X}$ could be a subset of the union of the open sets in $\mathcal{U}$.

Definition 3.2.2. *A cover* $\mathcal{U}$ *is called a* ***good cover*** *if for any subset* $\{U_{i_1}, \cdots, U_{i_k}\} \subseteq \mathcal{U}$ *the intersection* $\bigcap\limits_{i_j} U_{i_j}$ *is either empty or contractible.*

Definition 3.2.3. *The* ***nerve*** *of a cover* $\mathcal{U}$ *is an abstract simplicial complex, defined as the set of all non-empty intersections of the elements of* $\mathcal{U}$.

The following theorem is one of the foundational theorem of computational topology.

Theorem 3.2.1. *The nerve of a finite good closed cover* $\mathcal{U}$ *is homotopy equivalent to the space* $X := \bigcup\limits_{U_\alpha \in \mathcal{U}} U_\alpha$.

The nerve is a well known construction that transforms a continuous space to a combinatorial space preserving its homotopy type. The Čech complex defined in subsection 2.2.3 is the nerve of the cover (the set of d-balls) of the space defined by the union of the balls. Since a d-ball is convex, the intersection of any finitely many d-ball is convex and hence contractible, therefore the Čech complex is homotopic to the space defined by the union of the balls.

Nerve of a simplicial complex: The nerve $\mathcal{N}(K)$ of a simplicial complex K is defined as the nerve of the set of maximal simplices of the complex K (considered as a cover of the complex). Hence all the maximal simplices of K will be the vertices of $\mathcal{N}(K)$ and their non-empty intersection will form the simplices of $\mathcal{N}(K)$. For $j \geq 2$ the iterative construction is defined as $\mathcal{N}^j(K) = \mathcal{N}(\mathcal{N}^{j-1}(K))$. This definition of nerve preserves the homotopy type, $K \simeq \mathcal{N}(K)$[3]. A remarkable property of this nerve construction is its connection with strong collapses.

Taking the nerve of any simplicial complex K twice corresponds to a strong collapse.

Theorem 3.2.2. *[3, Proposition 3.4] For a simplicial complex* K*, there exists a subcomplex* L *isomorphic to* $\mathcal{N}^2(K)$*, such that* $K \searrow\searrow L$.

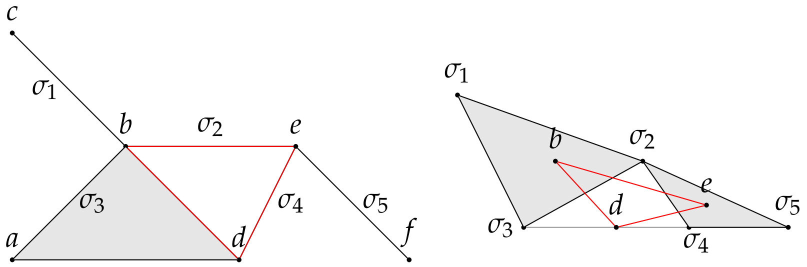


FIGURE 3.2 – Left: K (in grey), Right: $\mathcal{N}(K)$ (in grey) and $\mathcal{N}^2(K)$ (in red). $\mathcal{N}^2(K)$ is isomorphic to a full-subcomplex of K highlighted in red on the left.

An easy consequence of this theorem is that a complex K is *minimal* if and only if it is isomorphic to $\mathcal{N}^2(K)$ [3, Lemma 3.6]. This means that we can keep collapsing our complex K by applying iteratively $\mathcal{N}^2(.)$ until we reach the core of the complex K. The sequence $K, \mathcal{N}^2(K), ..., \mathcal{N}^{2p}(K)$ is a decreasing sequence in terms of number of simplices.

3.3 Strong collapse of a simplicial complex

In this section, we describe an algorithm to strong collapse a simplicial complex K, provide the details of the implementation and analyze its complexity.

Data structure. Basically, we represent K as the adjacency matrix M between the vertices and the maximal simplices of K. We will simply call M the adjacency matrix of K. The rows of M represent the vertices and the columns represent the maximal simplices of K. For convenience, we will identify a row (resp. column) and the vertex (resp. maximal simplex) it represents. An entry $M[v_i][\sigma_j]$ associated with a vertex v_i and a maximal simplex σ_j is set to 1 if $v_i \in \sigma_j$, and to 0 otherwise. For example, the matrix M in the left of the Table 3.1 corresponds to the leftmost simplicial complex K in Figure 3.2.

Usually, M is very sparse. Indeed, each column contains at most $d+1$ non-zero elements since the simplices of a d-dimensional complex have at most $d+1$ vertices, and each line contains at most Γ_0 non-zero elements where Γ_0 is the maximum number of maximal simplices that are incident to a given vertex. As already mentionned, in many practical situations, Γ_0 is a small fraction of the number of maximal simplices. It is therefore beneficial to store M as a list of vertices and a list of maximal simplices. Each vertex v in the list of vertices points to the maximal simplices that contain v, and each simplex in the list of maximal simplices points to its vertices. This data structure is similar to the SAL data structure of [8].

Core algorithm. Given the adjacency matrix M of K, we compute the adjacency matrix C of the *core* K^c. It turns out that using basic row and column removal operations, we can easily compute C from M. Loosely speaking our algorithm recursively computes $\mathcal{N}^2(K)$ until it reaches K^c.

	σ_1	σ_2	σ_3	σ_4	σ_5
a	0	0	1	0	0
b	1	1	1	0	0
c	1	0	0	0	0
d	0	0	1	1	0
e	0	1	0	1	1
f	0	0	0	0	1

	b	d	e
σ_1	1	0	0
σ_2	1	0	1
σ_3	1	1	0
σ_4	0	1	1
σ_5	0	0	1

	σ_2	σ_3	σ_4
b	1	1	0
d	0	1	1
e	1	0	1

TABLE 3.1 – From left to right M, $\mathcal{N}(M)$ and $\mathcal{N}^2(M)$.

The columns of M (which represent the maximal simplices of K) correspond to the vertices of $\mathcal{N}(K)$. Also, the columns of M that have a non-zero value in a particular row v correspond to the maximal simplices of K that share the vertex associated with row v. Therefore, each row of M represents a simplex of the nerve $\mathcal{N}(K)$. Not all simplices of $\mathcal{N}(K)$ are associated with rows of M but all maximal simplices are since they correspond to subsets of maximal simplices with a common vertex. To remedy this situation, we *remove* all the rows of M that correspond to non-maximal simplices of $\mathcal{N}(K)$. This results in a new smaller matrix M whose transpose, noted $\mathcal{N}(M)$, is the adjacency matrix of the nerve $\mathcal{N}(K)$. We then exchange the roles of rows and columns (which is the same as taking the transpose) and run the very same procedure as before so as to obtain the adjacency matrix $\mathcal{N}^2(M)$ of $\mathcal{N}^2(K)$.

The process is iterated as long as the matrix can be reduced. Upon termination, we output the reduced matrix $C := \mathcal{N}^{2p}(M)$, for some $p \geq 1$, which is the adjacency matrix of the core K^c of K. Removing a row or column is the most basic operation of our algorithm. We will discuss it in more detail later in the paragraph *Domination test*.

Example. As mentioned before, the matrix M in the left of the Table 3.1 represents the simplicial complex K in the left of Figure 3.2. We go through the rows first, rows a and c are subsets of row b and row f is a subset of e. Removing rows a, c and f and transposing M yields the adjacency matrix $\mathcal{N}(M)$ of $\mathcal{N}(K)$ in the middle. Now, row σ_1 is a subset of σ_2 and of σ_3, and σ_5 is a subset of σ_2 and of σ_4. We remove these two rows of $\mathcal{N}(M)$ and transpose $\mathcal{N}(M)$ so as to get $\mathcal{N}^2(M)$ (the rightmost matrix of Table 3.1), which corresponds to the core drawn in red in Figure 3.2.

Domination test. Now we explain in more detail how to detect the rows that need to be removed. Let v be a row of M and σ_v be the associated simplex in $\mathcal{N}(K)$. If σ_v is not a maximal simplex of $\mathcal{N}(K)$, it is a proper face of some maximal simplex $\sigma_{v'}$ of $\mathcal{N}(K)$. Equivalently, the row v' of M that is associated with $\sigma_{v'}$ contains row v in the sense that the non zero elements of v appear in the same columns as the non zero elements of v'. We will say that row v is dominated by row v' and determining if a row is dominated by another one will be called the row domination test. Notice that when a row v is dominated by a row v', the same is true for the associated vertices since all the maximal simplices that contain vertex v also contain vertex v', which is the criterion to determine if v is dominated by v' (See Remark 1 in Section 3.1.1). The algorithm removes all dominated rows and therefore all dominated vertices of K.

After removing rows, the algorithm removes the columns that are no longer maximal in K, which might happen since we removed some rows. Removing a column may lead in turn to new dominated vertices and therefore new rows to be removed. When the algorithm stops, there are no rows to be removed and we have obtained the core K^c of the complex K. Note that the algorithm provides a constructive proof of 3.2.2.

Removing columns is done in very much the same way: we just exchange the roles of rows and columns.

Computing the retraction map r. The algorithm also provides a direct way to compute the retraction map r defined in Section 3.1. The retraction map corresponding to the strong collapses executed by the algorithm can be constructed as follows. A row r being removed in M corresponds to a dominated vertex in K and the row which *contains* r corresponds to a dominating vertex. Therefore we map the dominated vertex to the dominating vertex and compose all such maps to get the final retraction map from K to its core K^c. The final map is simplicial as well, as it is a composition of simplicial maps.

Reducing the number of domination tests. We first observe that, when one wants to determine if a row v is dominated by some other row, we don't need to test v with all other rows but with at most d of them. Indeed, at most $d+1$ rows can intersect a given column since a simplex can have at most $d+1$ vertices. For example, in Table 3.1 (Left), to check if row e (highlighted in brown) is dominated by another row, we pick the first non-zero column σ_2 (highlighted in Gray) and compare e with the non-zero entries {b} of σ_2.

A second observation is that we don't need to test all rows and columns for domination, but only the so-called candidate rows and columns. We define a row r to be a **candidate row** for the next iteration if at least one column containing one of the non-zero elements of r has been removed in the previous column removal iteration. Similarly, by exchanging the roles of rows and columns, we define the **candidate columns**. Candidate rows and columns are the only rows or columns that need to be considered in the *domination* tests of the algorithm. Indeed, a column τ of M whose non-zero elements all belong to rows that are present from the previous *iteration* cannot be dominated by another column τ' of M, since τ was not dominated at the previous iteration and no new non-zero elements have ever been added by the algorithm. The same argument follows for the candidate rows.

We maintain two *queues*, one for the candidate columns (colQueue) and one for the candidate rows (rowQueue). These queues are implemented as First in First out (FIFO) queues. At each iteration, we *pop out* a candidate row or column from its respective queue and test whether it is dominated or not. After each successful domination test, we *push* the candidate columns or rows in their appropriate queue in preparation for the subsequent iteration. In the first iteration, we *push* all the rows in rowQueue and then alternatively use colQueue and rowQueue. Algorithm 2 gives the pseudo code of our algorithm.

Time Complexity. The most basic operation in our algorithm is to determine if a row is dominated by another given row, and similarly for columns. In our implementation, the rows (columns) of the matrix that are considered by the algorithm are stored as sorted lists. Checking if one sorted list is a subset of another sorted list can be done in

Algorithm 2 Core algorithm

```
 1: procedure CORE($M$)                                  ▷ $M$ is the adjacency matrix of $K$.
 2:     $rowQueue \leftarrow$ push all rows of $M$ (all vertices of K)
 3:     $colQueue \leftarrow$ empty
 4:     while rowQueue is not empty do
 5:         $v \leftarrow pop(rowQueue)$
 6:         $\sigma \leftarrow$ the first non-zero column of $v$
 7:         for non-zero rows $w$ in $\sigma$ do
 8:             if $v$ is a subset of $w$ then
 9:                 Remove $v$ from $M$
10:                 push all non-zero columns $\tau$ of $v$ to $colQueue$ if not pushed before
11:                 break
12:             end if
13:         end for
14:     end while
15:     while colQueue is not empty do
16:         $\tau \leftarrow pop(colQueue)$
17:         $v \leftarrow$ the first non-zero row of $\tau$
18:         for non-zero columns $\sigma$ in $v$ do
19:             if $\tau$ is subset of $\sigma$ then
20:                 Remove $\tau$ from $M$
21:                 push all non-zero rows $w$ of $\tau$ to $rowQueue$ if not pushed before
22:                 break
23:             end if
24:         end for
25:     end while
26:     if rowQueue is not empty then
27:         GOTO 4
28:     end if
29:     return $M$                     ▷ $M$ is now the adjacency matrix of the core of $K$.
30: end procedure
```

time $\mathcal{O}(l)$, where l is the size of the longer list. Note that the length of a row list is at most Γ_0 where Γ_0 denotes the largest number of maximal simplices incident to a vertex. The length of a column list is at most $d+1$ where d is the dimension of the complex. Hence checking if a row is dominated by another row (Line 8) takes $\mathcal{O}(\Gamma_0)$ time and checking if a column is dominated by another column (Line 19) takes $\mathcal{O}(d)$ time.

Next consider one execution of the while loop on rows (Lines 4-14). We will loop at most n_v times since *rowQueue* contains at most n_v elements, where n_v is the number of vertices of the complex K. Since simplex σ has at most $d+1$ vertices w and since testing if v is a subset of w (Line 8) takes at most Γ_0 time as shown above, a row domination test (Lines 7-13) takes time $O(d\Gamma_0)$. Hence, executing one while loop on rows takes $n_v d\Gamma_0$ time. If, during the execution of the while loop, we never execute Lines 9-10, there is no dominated vertex, *colQueue* remains empty and the algorithm stops. Otherwise, we enter the while loop on columns (Lines 15-25).

We will loop at most m times since *colQueue* contains at most m elements where m is the number of maximal simplices in K. Vertex v is incident to at most Γ_0 maximal simplices and, as shown above, testing if τ is a subset of σ (Line 19) takes $O(d)$ time. It follows that executing one while loop on columns takes $\mathcal{O}(md\Gamma_0)$ time.

It remains to bound the number of times we execute the while loops or, equivalently, the number of times we execute Line 27. This number is at most n_v. Indeed, each time we execute Line 27, the algorithm has removed at least one row from M since otherwise it stops. It follows that the total complexity of the algorithm is $\mathcal{O}(n_v d\Gamma_0\,(n_v+m))$. Noticing that $n_v\Gamma_0 \geq m$, the complexity can be simply written as $\mathcal{O}(n_v^2 d\Gamma_0^2)$.

In practice, m is much smaller than n, the total number of simplices, and Γ_0 is much smaller than Γ, the maximum number of simplices incident on a vertex. Typically Γ grows exponentially with d while Γ_0 remains almost constant as d increases. See Table 5 in [8], related results in [7], and the plots in Section 3.5.

3.4 Strong collapse of a sequence of simplicial complexes

In this section, we will present our main result that the persistence homology of a sequence of simplicial complexes is preserved under strong collapse. We begin with some brief background on zigzag persistence all the notions recalled here are simply extensions of the notions about towers and filtration recalled in Section 2.5, we mention them here for completeness. Readers interested in more details can refer to [12, 13, 20].

Given a *zigzag sequence* of simplicial complexes

$$\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}.$$

If we compute the homology classes of all K_is, we get the sequence

$$\mathcal{P}(\mathcal{Z}) : \{H_p(K_1) \xrightarrow{f_1^*} H_p(K_2) \xleftarrow{g_2^*} H_p(K_3) \xrightarrow{f_3^*} \cdots \xrightarrow{f_{(n-1)}^*} H_p(K_n)\}.$$

Here $H_p(-)$ denotes the homology class of dimension p with coefficients from a field $\mathbb{F}$ and $*$ denotes an induced homomorphism. $\mathcal{P}(\mathcal{Z})$ is a sequence of vector spaces connected through homomorphisms, called a **zigzag module**. More formally, a *zigzag module* $\mathbb{V}$ is a sequence of vector spaces

$$\{V_1 \rightarrow V_2 \leftarrow V_3 \rightarrow \cdots \leftrightarrow V_n\}$$

connected with homomorphisms $\{\rightarrow, \leftarrow\}$ between them. A zigzag module arising from a sequence of simplicial complexes captures the evolution of the topology of the sequence.

For two integers b and d, $1 \leq b \leq d \leq n$; we can define an **interval module** $\mathbb{I}[b,d]$ by assigning V_i to $\mathbb{F}$ when $i \in [b,d]$, and null spaces otherwise, the maps between any two $\mathbb{F}$ vector spaces is identity and is zero otherwise. For example

$$\mathbb{I}[2,4] : \{0 \xrightarrow{0} \mathbb{F} \xleftarrow{I} \mathbb{F} \xrightarrow{I} \mathbb{F} \xleftarrow{0} 0 \xrightarrow{0} 0\},$$

here $n = 6$. Any zigzag module can be *decomposed* as the direct sum of *finitely* many interval modules, which is unique upto the permutations of the interval modules [12].

The multiset of all the intervals $[b_j, d_j]$ corresponding to the interval module decomposition of any zigzag module is called a **zigzag (persistence) diagram**. The zigzag diagram completely characterizes the zigzag module, that is, there is bijective correspondence between them [12, 50].

Two different zigzag modules

$$\mathbb{V} : \{V_1 \rightarrow V_2 \leftarrow V_3 \rightarrow \cdots \leftrightarrow V_n\} \text{ and}$$

$$\mathbb{W} : \{W_1 \rightarrow W_2 \leftarrow W_3 \rightarrow \cdots \leftrightarrow W_n\},$$

connected through a set of homomorphisms $\phi_i : V_i \rightarrow W_i$ are **equivalent** if the ϕ_is are isomorphisms and the following diagram commutes [12, 20].

$$\begin{tikzcd}
V_1 \arrow[r] \arrow[d, "\phi_1"] & V_2 \arrow[d, "\phi_2"] & V_3 \arrow[l] \arrow[d, "\phi_3"] & \cdots & {} \arrow[r] & V_{n-1} \arrow[r] \arrow[d, "\phi_{n-1}"] & V_n \arrow[d, "\phi_n"] \\
W_1 \arrow[r] & W_2 & W_3 \arrow[l] & \cdots & {} \arrow[r] & W_{n-1} \arrow[r] & W_n
\end{tikzcd}$$

Note that the *length* of the modules and the directions of the arrows in them should be consistent. Two equivalent zigzag modules will have the same interval decomposition, therefore the same zigzag diagram.

Strong collapse of the zigzag module: Given a zigzag sequence

$$\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}.$$

We define the **core sequence** $\mathcal{Z}^c$ of $\mathcal{Z}$ as

$$\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\}.$$

Where K_i^c is the core of K_i. The forward maps are defined as, $f_j^c := r_{j+1} f_j i_j$; and the backward maps are defined as $g_j^c := r_j g_j i_{j+1}$. The maps $i_j : K_j^c \hookrightarrow K_j$ and

$r_j : K_j \to K_j^c$ are the composed inclusions and the retractions maps defined in Section 3.1 respectively. We call the procedure of forming the core sequence using the cores and the induced simplicial maps as **core-assembly**.

Theorem 3.4.1. *Zigzag modules $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are equivalent.*

Proof. Consider the following diagram

$$\begin{tikzcd}
K_1 \arrow[r, "f_1"] \arrow[d, "r_1"] & K_2 \arrow[d, "r_2"] & K_3 \arrow[l, "g_2"'] \arrow[d, "r_3"] & \cdots & {} \arrow[r] & K_{n-1} \arrow[r, "f_{n-1}"] \arrow[d, "r_{n-1}"] & K_n \arrow[d, "r_n"] \\
K_1^c \arrow[r, "f_1^c"] & K_2^c & K_3^c \arrow[l, "g_2^c"'] & \cdots & {} \arrow[r] & K_{n-1}^c \arrow[r, "f_{n-1}^c"] & K_n^c
\end{tikzcd}$$

and the associated diagram after computing the p-th homology groups

$$\begin{tikzcd}
H_p(K_1) \arrow[r, "f_1^*"] \arrow[d, "r_1^*"] & H_p(K_2) \arrow[d, "r_2^*"] & H_p(K_3) \arrow[l, "g_2^*"'] \arrow[d, "r_3^*"] & \cdots & {} \arrow[r] & H_p(K_{n-1}) \arrow[r, "f_{n-1}^*"] \arrow[d, "r_{n-1}^*"] & H_p(K_n) \arrow[d, "r_n^*"] \\
H_p(K_1^c) \arrow[r, "(f_1^c)^*"] & H_p(K_2^c) & H_p(K_3^c) \arrow[l, "(g_2^c)^*"'] & \cdots & {} \arrow[r] & H_p(K_{n-1}^c) \arrow[r, "(f_{n-1}^c)^*"] & H_p(K_n^c)
\end{tikzcd}$$

Since there exists a strong deformation retraction between r_j and i_j, the induced homomorphisms r_j^* and i_j^* are isomorphisms [31, Corollary 2.11]. We claim that the composed map $i_j r_j$ is contiguous to the identity on K_j. Indeed, i_j can be decomposed into a series of inclusions $i_j^1 i_j^2 ... i_j^s$ that correspond to elementary strong collapses and similarly r_j can be decomposed into $r_j^s r_j^{s-1} ... r_j^1$ for some s. Using [3, Proposition 2.9] and the fact that contiguity is preserved under composition, it follows that $i_j r_j$ is contiguous to the identity on K_j since

$$i_j r_j = i_j^1 i_j^2 ... i_j^s \circ r_j^s r_j^{s-1} ... r_j^1 \;\sim\; i_j^1 i_j^2 ... i_j^{s-1} \circ r_j^{s-1} ... r_j^1 \;\sim\; ... \;\sim\; i_j^1 \circ r_j^1 \;\sim \mathrm{id}$$

It then follows that $f_j^c r_j = r_{j+1} f_j i_j r_j$ is contiguous to $r_{j+1} f_j$, and similarly that $g_j^c r_{j+1}$ is contiguous to $r_j g_j$. Now, since contiguous maps are homotopic at the level of geometric realizations and since homotopic maps induce the same homomorphism, we have $(f_j^c r_j)^* = (r_{j+1} f_j)^*$ and thus $(f_j^c)^* r_j^* = r_{j+1}^* f_j^*$ and similarly $(g_j^c)^* r_{j+1}^* = r_j^* g_j^*$, see [31, Proposition (1) page 111]. Therefore all the squares in the lower diagram commute and the set of maps r_j^*s are isomorphisms, therefore $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are *equivalent* and hence their zigzag diagrams are identical. □

Remark. The above result can be extended to multidimensional persistence using the more general notion of quiver representation [20].

The algorithms to compute persistence are for sequences of simplicial complexes connected via inclusions only : inclusions can be in both directions, that is either zigzag filtrations or filtrations. To compute the persistent homology of more general sequences, one usually converts them to an equivalent inclusion-only sequence. Zigzag sequences will thus be transformed in zigzag filtrations and towers will be transformed in filtrations respectively.

The works of Dey et. al. [22] and Kerber and Schrieber [34] describe methods to convert a tower to a filtration general towers. The same techniques could be applied to convert a zigazag sequence to a zigzag filtration. Given a zigzag sequence of general simplicial complexes, one can individually strongly collapse each of the complexes using the algorithm described in Section 3.3 and then assemble the cores using the maps prescribed in this section and compute the persistence of the core sequence. The core sequence of a zigzag filtration is usually a general zigzag sequence rather than a zigzag filtration. To compute the persistent homology of such a core sequence, one can expand it so as to obtain a zigzag *filtration* using the approach described in [22] and [34].

In the next section we perform some experiments using the methods described in this section. Our experiments are performed on filtrations of Rips complex and after strong collapsing the filtrations we get towers. We then use the algorithm described in [34] to compute the persistence of the core tower.

In the special case of a flag filtration or of a flag tower, the core sequence is a flag tower. Again, to compute the persistence of a flag tower, one needs to expand it to get a filtration. We describe a method to transform a flag tower to a flag filtration using only the 1-skeleton of the tower in Chapter 4.

3.5 Approximation of the persistence diagram of filtrations

In this section, we restrict our attention to the case of filtrations, i.e. nested sequences of simplicial complexes connected by inclusions. Filtrations, and in particular Rips filtrations, are easy to construct and commonly used in Topological Data Analysis. However, because of their inclusive nature, the number of simplices grows exponentially in the number of sample points which often limits PH computation to small examples.

As we have seen, our method tackles this issue by using only maximal simplices and dramatically reducing the size of the input complexes using strong collapses. Further dramatic reduction can be obtained if we allow to approximate the PH of the input filtration as shown now.

An approximation scheme: Instead of strong collapsing all the complexes in the filtration, we can strong collapse the complexes less often, i.e. after several inclusions rather than just one. This will result in a faster algorithm but comes with a cost: the computed PD will only be approximate.

The complexes in a filtration $\mathcal{F}$ are usually associated to different real values of a scale parameter. For example, in a VR filtration, the filtration value of a simplex is the length of the longest edge of the simplex and in general the filtration consists of a sequence of **elementary** inclusions (i.e. inclusion of a single simplex) with non decreasing *filtration values*. A persistent pair then associates the filtration value of a simplex that creates a homology cycle with the filtration value of the simplex that kills the cycle. The *length* of the persistence pair is the difference between the two filtration values.

We call **snapshots** the values of the scale parameter at which we choose to strong collapse the complex. The difference between two consecutive snapshots is called a **step**. We approximate the **filtration value** of a simplex as the value of the snapshot at which it first appears. It is not hard to see that our algorithm will report all persistence pairs that are separated by at least one snapshot. Hence if all steps are at most some $\epsilon > 0$, we will compute all the persistence pairs whose lengths are at least ϵ. It follows that the bottleneck distance between the computed PD and the exact one is at most ϵ.

Computational experiments: Now, we present some computational experiments to showcase the efficiency achieved by our approach. For our experiments, we choose three datasets coming from applications, for more detail about the datasets please refer to [43]. The algorithms to strong collapse a simplicial complex (Algorithm 2) and to form the core sequence (core-assembly) have been coded in C++. We name our package to preprocess the initial sequence and construct the collapsed sequence *PH-Collapser*. The code has been compiled using the compiler 'clang-900.0.38' and all computations were performed on a '2.8 GHz Intel Core i5' machine with 16 GB of available RAM.

For each data set, we select a number of snapshots and *independently* strong collapse all the complexes associated to these snapshots. We then assemble the resulting individual *cores* using the induced simplicial maps introduced in Section 3.4. The resulting core sequence with induced simplicial maps between the collapsed complexes is in general a simplicial tower we call the *core tower*. We then convert the core tower into an equivalent filtration using the Sophia software [44], which implements the algorithm described by Kerber and Schreiber [34]. Finally, we run the persistence algorithm of the Gudhi library [30] to obtain the persistence diagram (PD) of the equivalent filtration. For the core tower, we use Gudhi through Sophia using the command <./sophia -cgudhi inputTowerFile outputPDFile> [1].

Table 3.2 summarises the results of the experiments. The total time includes 1. the time taken to compute the entire VR filtration at snapshot values, 2. the time taken to collapse all the subcomplexes, assemble their cores and transform them into an equivalent filtration, and 3. the time to compute the PD of the equivalent filtration. At each snapshot, we compute the full associated VR complex with no restriction on its dimension and compute the PD in all dimensions upto the dimension of the complex. The experiments are done on the three datasets **netw-sc**, **senate** and **eleg** from [18]

In all cases, the total time stays within a small factor (less than 2) from the time to compute the VR complex. The only exception is the dataset *senate*, which has comparatively fewer points and for which the time to compute the VR complex is relatively fast. However, here too the ratio is less than eight, which is much less than 403, the number of snapshots. This clearly indicates that in our approach the time to perform the strong collapses and the core assemblies together with the time to compute the PD is much smaller than the time to compute the VR complex. This implies that one can refine the PD (through smaller steps) at a very low cost.

Our choice of *snapshots* is arbitrary and could be done in a non uniform way after analyzing the distribution of the length of the edges in the Rips-complex at a relatively small increase in the total computing time.

---

[1] When we use the -cgudhi option, Sophia reports two computation times. The first one is the total time taken by Sophia which includes (1) reading the tower, (2) transforming it to a filtration and (3) computing PD using Gudhi. The second reported time is just the time taken by Gudhi to compute the exact PD of the input filtration. In our comparisons, we report the total time taken by Sophia.

Data	Pnt	Threshold	Strong Collapse + PH computation				
			Dim	Rips-Comp-Time	Total-Time	Step	TotSnaps
netw-sc	379	4.5	41	13s	21s	0.02	213
″	″	5.5	57	117s	144s	0.02	263
senate	103	0.415	54	1.7s	13.1s	0.001	403
eleg	297	0.3	105	443s	578.3s	0.001	284

TABLE 3.2 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum scale parameter (Threshold), dimension of the VR Complex (Dim), time taken to compute the VR complex (Rips-Comp-Time), total time (Total-Time), parameter step (Step) and total number of snapshots used (TotSnaps). All times are averaged over five trials.

Comparison with Ripser. Ripser [4] is a state-of-the-art software to compute persistence of Rips complex. To compare, we perform the same experiments using Ripser. Command <./ripser inputData –format distances –threshold inputTh –dim inputDim > was used to run Ripser and we used the distance matrix format for all the datasets. Differently from PH-Collapser, Ripser needs a parameter –dim until which it computes the PD. For a given *threshold* (maximum scale parameter) and *dim*, Ripser basically computes the *dim*-skeleton of the VR complex and then computes the PD of the skeleton.

Table 3.3 contains the results of Ripser. By comparing the tables, PH-Collapser clearly outperforms Ripser by a huge margin considering that we compute the persistence diagram until the full dimension. Ripser performs quite well for computing PD in low-dimensions, however as we move to intermediate dimensions it slows down quite considerably and in some cases (dimension above 7) the size of the complex is so huge that Ripser crashed due to memory overload. In Table 3.3, we provide the running time of Ripser with increasing dimension of the PD computed.

	Val		Val		Val			
			Dim	Time	Dim	Time	Dim	Time
netw-sc	379	4.5	4	3.8s	5	21.5s	7	357s
″	″	5.5	4	25.3s	5	231.2s	6	∞
senate	103	0.415	3	0.52s	4	5.9s	5	52.3s
″	″	″	6	406.8s	7	∞		
eleg	297	0.3	3	8.9s	4	217s	5	∞

TABLE 3.3 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum scale parameter (Threshold), input dimension for Ripser (Dim), total time taken by Ripser (Time). Most results are averaged over five trials except the longer ones. ∞ in the Time column means that the experiment ran longer than 12hrs or crashed due to memory overload.

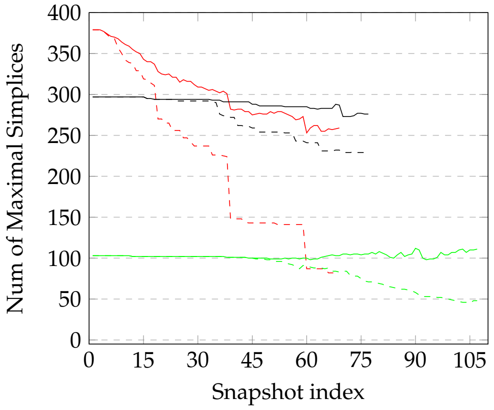
The datasets we considered here didn’t have any non-trivial persistence pairs beyond dimension 4-5, however with our approach we were able to say it and until the maximum possible dimension. Whereas in most of the cases, Ripser couldn’t compute even the Rips skeleton beyond dimension 7.

As mentioned before in this section, we define the filtration value of a simplex as the value of the snapshot parameter at which it appears for the first time. Therefore, the computed PD by PH-Collapser is not exact. However, in the above experiments, we choose steps that are very small so that the bottleneck distance between the two PD returned by Ripser and PH-collapser for a given data set is also very small.

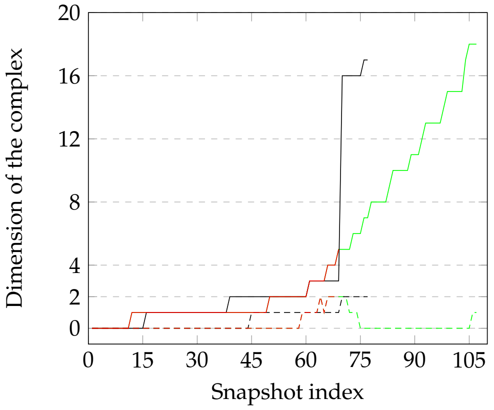
The above experiments have been performed considering Rips complexes as general simplicial complexes. This was to illustrate the general performances of our algorithm that works for any type of simplicial complexes. However, in the case of Rips complexes and, more generally, in the case of flag complexes, we can further improve the performances by representing the complexes by their 1-skeletons instead of their maximal faces, an even more compact representation. In Chapter 4, we adapt the methodology introduced in this Chapter to the case of flag complexes and show that we can compute the core filtration (which is itself a flag filtration) using only 1-skeletons.

The plots below count the maximal simplices and the dimensions of the complexes across the initial filtration (in solid) and the collapsed tower (as dashed). We show in solid red and dashed red respectively the filtration and the collapsed tower of the data **netw-sc**. Similarly we show in solid green and dashed green respectively the filtration and the collapsed tower of the data **senate**. Finally, we show in solid **black** and dashed **black** respectively the filtration and the collapsed tower of the data **eleg**. We can observe that in all cases the number of maximal simplices never increases. Also they are far fewer (by a factor exponential in the dimension) in number compared to the total number of simplices. Observe that for the uncollapsed filtrations solid red, green and **black**, the dimension of the complexes increases quite rapidly with the snapshot index. Another key fact to observe is that the dimension of the complexes in the corresponding core tower are much smaller than their counterparts in the original filtration. This has a huge effect on the performances since the total number of simplices depends exponentially on the dimension.

Count of maximal simplices across filtrations



Dimension of the complex across filtrations



Observations from the plots combined with the experimental results of Table 3.2 clearly indicate that performances are much improved when using strong collapses.

3.6 Discussion

In this chapter, we presented a novel approach to compute the persistence homology of a sequence of simplicial complexes. Our approach is based on strong collapses that has been introduced by Barmak and Minian [3]. Our method works very well in pratice and, as shown using publicly available data, it is much faster and memory efficient with respect to not using it. We believe that the solid mathematical foundations presented in [3], its applicability to all kinds of sequences of simplicial complexes, and the availability of the simple and efficient algorithms developed in this chapter, strong collapses will be immensely useful to reduce the complexity of many problems in computational topology.

On the theoretical side, this work raises several questions. In particular, it would be nice to have theoretical guarantees on the amount of reduction the algorithm can achieve. We intend to explore this and related issues in future work.

This chapter was based on our work reported in [10].

Chapter 4

Persistence of Flag Complexes

In this chapter, we restrict the class of simplicial complexes to flag complexes and we focus on the problem of computing Persistent Homology of a flag tower, i.e. a sequence of flag complexes connected by simplicial maps. The case of flag filtration is then a special case of flag tower. Flag complexes are fully characterized by their graph (or 1-skeleton), the other faces being obtained by computing the cliques of the graph. By restricting to the case of flag complexes, we achieve further decisive improvement in terms of time and space complexities with respect to our approach in the previous chapter.

We first show that strong collapses of flag complexes can be computed in time $O(k^2 n_v^2)$ where n_v is the number of vertices of the complex and k is the maximal degree of its graph. Moreover we can strong collapse a flag complex knowing only its 1-skeleton and the resulting complex is also a flag complex.

In the previous chapter, we showed that the persistent homology of Rips filtrations can be computed very efficiently using strong collapses. However, most of the time was devoted to computing the maximal cliques of the complex prior to their strong collapse. Using the results developed in this chapter we avoid computing the maximal simplices.

After strong collapsing a flag tower as prescribed in the previous chapter, Section 3.4, the reduced sequence is also a flag tower. As previously described in Chapter 1 to compute persistence of a tower we need to transform it to an equivalent filtration. In Section 4.2, we show how to transform a flag tower to an equivalent flag filtration using again only 1-skeletons. This implies that the complete pre-processing described in the previous chapter can be performed only on the 1-skeleton, when the complex is a flag complex. This leads to a very simple and extremely efficient method.

In Section 4.1 we prove that strong collapse of a flag complex can be computed using the 1-skeleton of the complex and provide a simple algorithm to compute the 1-skeleton of the core. The algorithm to transform a flag tower to a flag filtration is discussed in Section 4.2. In Section 4.3 we provide the computational experiments.

4.1 Strong Collapse of a Flag complex

In this section, we show that the core of a flag complex K is itself a flag complex whose graph is called the *core graph* of K. The core graph of K can be computed from the 1-skeleton G of K in time $\mathcal{O}(n_v^2 k^2)$.

Although this change with respect to the Algorithm 2 in previous chapter might look minor, it is crucial in practice as the time to compute all the maximal simplices of a flag complex from its graph is exponential in the number of its vertices. We thus reduce immensely the time and space complexity of the general algorithm of previous chapter whose complexity is $\mathcal{O}(v^2\Gamma_0 d + m^2\Gamma_0 d)$, where Γ_0 is an upper bound on the number of *maximal simplices* incident to a vertex.

In the following lemma, we describe a condition in terms of the closed neighborhood $N_G[v]$ of a vertex v of a flag complex K under which v will be dominated by another vertex v' of K. This result has been studied in another context in [28, Lemma 4.1].

Lemma 4.1.1. *Let K be a flag complex. A vertex $v \in K$ is dominated by v' iff $N_G[v] \subseteq N_G[v']$.*

Proof. If v is dominated by v', then, according to Remark 1, the set of maximal simplices that contain v is a subset of the set of maximal simplices that contain v'. It follows that $N_G[v] \subseteq N_G[v']$.

Now we prove the other direction. Let σ be a maximal simplex of K containing v. Any other vertex x of σ is joined to v by an edge $[x, v] \in \sigma$. Moreover, since $N_G[v] \subseteq N_G[v']$, $[v, v']$ and $[x, v']$ are in K. It follows that every vertex in σ has an edge with both v and v' and, since K is a flag complex and σ is maximal, v' must be in σ. This implies that all the maximal simplices that contain v also contain v'. Hence v is dominated by v'. □

As mentioned before in Chapter 2, an elementary strong collapse consists in removing a dominated vertex, and it can be easily observed that removing a vertex does not affect the 'flagness' of the residual complex $K \setminus v$. In other words, if σ is a maximal clique with vertex v, the resultant clique $\sigma \setminus v$ is still a maximal clique in $K \setminus v$. Moreover, all the other cliques that do not contain v still span the complete simplices. This implies that the core K^c of a flag complex K with graph G is a flag complex of a sub-graph G^c of G.

In what follows next, we describe an algorithm to compute the core graph $G^c \subseteq G$ whose flag complex is the core K^c of K.

Data structure: We represent G with its adjacency matrix M, where the rows and the columns of M represent the vertices of G. An entry $M[v_i][v_j]$ associated with vertices v_i and v_j is set to 1 if either the edge $[v_i, v_j] \in G$ or $i = j$, and to 0 otherwise. Note that we set $M[v_i][v_j] = 1$ for $i = j$ to be able to consider closed neighborhood. We will say that a row v is contained in another row v' if the set of column indices of the non-zero entries of v is a subset of the indices of the non-zero entries of v'. It is clear that if a row v is contained in another row v', we have $N_G[v] \subseteq N_G[v']$ and therefore the vertex v is dominated by the vertex v'

Core graph algorithm: Given the adjacency matrix M of G, we compute the adjacency matrix C of the core graph G^c. In view of Lemma 4.1.1, we can easily compute C from M using basic row removal operations. Loosely speaking, we remove the rows of M that are contained in another row. After removing the row associated to v, we simultaneously update the matrix by removing the column associated to v. The process is iterated as long as the matrix can be reduced. Upon termination, we output the reduced matrix C, which is the adjacency matrix of the core graph G^c of K. Since the core of a complex is always unique, the order in which vertices are removed does not matter [3].

Retraction map computation: We can easily compute the retraction map r defined in Section 3.1 using the above core graph algorithm. A row v being removed in M corresponds to a dominated vertex in K and the row which contains v corresponds to a dominating vertex. Therefore we map the dominated vertex to the dominating vertex.

Domination tests optimization: Let us observe that, to check if a row v is dominated by some other row v', it is sufficient to compare v with its neighbors, which are at most k in number, if k denotes the maximum degree of the vertices in G.

We define a row v to be a **candidate row** for the next iteration if at least one of its neighbors has been removed in a previous row removal iteration. We observe that the candidate rows are the only rows that need to be considered in the domination tests of the algorithm. Indeed, a row w of M whose set of neighbors has not been modified at the previous *iteration* cannot be dominated by another row v' of M, as w was not dominated in the previous iteration and all other vertices can only loose neighbors. This ensures that w will still remain un-dominated.

We maintain a *queue*, for the candidate rows (rowQueue) which is implemented as a First in First out (FIFO) queue. At each iteration, we *pop out* a candidate row from rowQueue for domination test. After each successful domination test, we push the new candidate rows in the queue in preparation for the subsequent iteration. In the first iteration, we push all the rows in rowQueue. Algorithm 3 gives the pseudo code of our algorithm.

Algorithm 3 Core graph algorithm

```
procedure CORE($M$)
    input : the adjacency matrix $M$ of the graph of a flag complex $K$
    $rowQueue \leftarrow$ push all rows of $M$ (all vertices of $K$)
    while rowQueue is not empty do
        $v \leftarrow pop(rowQueue)$
        $N_G[v] \leftarrow$ 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 if
        end for
    end while
    return $M$                          ▷ $M$ is now the adjacency matrix of the core of $K$
end procedure
```

Time Complexity: Let us start by analyzing the most basic operation in our algorithm which is to determine if a row is dominated by another row. We store the rows of the matrix as sorted lists. Deciding if a sorted list is included in another sorted list

(Line 8) can be done in time $\mathcal{O}(l)$, where l is the size of the longer list. In our case, the length of a row list is at most $k+1$ where k denotes as before the maximal degree of any vertex. Hence lines 8-12 takes $\mathcal{O}(k)$ time.

As explained in the paragraph *Domination tests optimization*, each row is checked against at most k other rows. Hence the for loop (Lines 7-13) is executed at most k times. Moreover, since at each iteration we ought to remove at least one row, the total number of iterations on the rows, i.e. the number of times the while loop is executed, is at most $O(n_v^2)$, where n_v is the total number of vertices of the complex K. It follows that the worst-case time complexity of our algorithm is $\mathcal{O}(n_v^2 k^2)$.

4.2 From a Flag Tower to a Flag Filtration

In this section, we show that, thanks to the notion of strong collapses, we can efficiently turn a flag *tower* into a flag *filtration* using only edge inclusions over the 1-skeletons of the complexes.

4.2.1 Previous work

It is known that any general simplicial map can be decomposed into elementary inclusions and elementary contractions. Hence if we can replace an elementary contraction $\{\{u, v\} \mapsto u\}$ by an equivalent (not necessarily elementary) inclusion, we transform a tower into an equivalent filtration. This was the philosophy introduced by Dey et al. [22]. This idea has been further refined by Kerber and Schreiber [34] who introduced a slightly different approach based on coning. They provided theoretical bounds on the size and time to construct the final equivalent filtration. Specifically, they proved that the size of the equivalent filtration is $\mathcal{O}(d * n * \log n_0)$, where d is the maximal dimension of the complexes in the input tower and n (resp. n_0) the total number of elementary inclusions (resp. vertex inclusions) in the input tower [34, Theorem 2].

4.2.2 A new construction

We now present an algorithm that turns a flag tower into a flag filtration with the same PH. Our work builds upon the above mentioned previous works [22, 34]. The difference is that we use strong expansion which is the inverse operation of a strong collapse. The main advantage of strong expansions is that, when the input is a flag tower, we can use the domination criterion of Lemma 4.1.1. This leads to a simple algorithm that only deals with edges. The output filtration is a flag filtration, which can be represented very compactly. Moreover, since a strong expansion is a coning, we will be able to use the theoretical results of [34]. Now we describe our construction.

Let K_i be a flag complex and G_i be its 1-skeleton. We associate to K_i an augmented complex $\mathbb{K}_i \supseteq K_i$. As will be seen below, $\mathbb{K}_i$ is also a flag complex whose 1-skeleton will be denoted by $\mathbb{G}_i$. Following the terminology of [34], we call a vertex $v \in \mathbb{K}_i$ to be **active** if it is currently *not dominated*. The active closed neighborhood $ActN_{G_i}[v]$ is then defined as the set of all active vertices in $N_{G_i}[v]$. Similarly, $ActN_{G_i}[v \setminus u]$ denotes the set of active vertices in the closed neighborhood $N_{G_i}[v]$ of v that are not in $N_{G_i}[u]$. Finally, let $\{[u, ActN_{G_i}[v \setminus u]]\}$ denote the set of edges between u and $ActN_{G_i}[v \setminus u]$.

Using the notions defined above, we now explain how to inductively construct a filtration associated to a given flag tower. The construction operates in a streaming fashion on the 1-skeleton. We distinguish two kinds of inputs: elementary inclusions (of a vertex or an edge) and elementary contractions. For $i = 0$, we set $\mathbb{G}_0 = \emptyset$. We then define $\mathbb{G}_i$ as follows.

1. if $G_i \xrightarrow{\cup\sigma} G_{i+1}$ is an elementary *inclusion* where σ is either a vertex or an edge, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \sigma$.

2. if $G_i \xrightarrow{\{u,v\}\mapsto u} G_{i+1}$ is an elementary *contraction*

   2.1. if $|ActN_{\mathbb{G}_i}[v \setminus u]| \leq |ActN_{\mathbb{G}_i}[u \setminus v]|$, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[u, ActN_{\mathbb{G}_i}[v \setminus u]\}$ and v as contracted

   2.2. otherwise, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[v, ActN_{\mathbb{G}_i}[u \setminus v]\}$ and u as contracted.

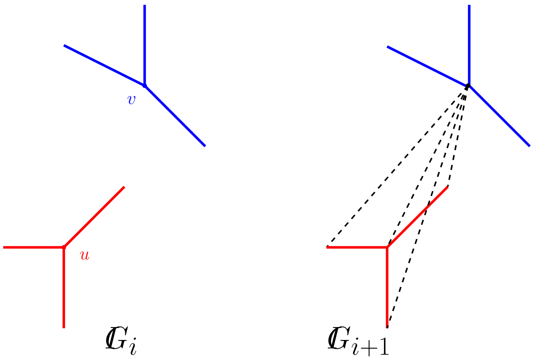


FIGURE 4.1 – Demonstration of strong expansion: On the left we have the active neighborhoods of the vertex u (in red) and of v (in blue) in the graph $\mathbb{G}_i$. The graphs $\mathbb{G}_i$ might have other vertices and edges, here we have zoomed into the local active neighborhoods of u and v. On the right, we have the local neighborhoods of u and v in $\mathbb{G}_{i+1}$ after conning v to the active neighborhood of u.

Note that $\mathbb{G}_i \subseteq \mathbb{G}_{i+1}$ and thus $\mathbb{K}_i \subseteq \mathbb{K}_{i+1}$. We continue the construction until the end of our input tower.

Complexity Analysis: The vertices marked *contracted* during our construction are exactly the same as the inactive vertices defined in [34]. By construction, any contracted vertex will be dominated permanently in the filtration. Since such a vertex stops existing in the tower later on, its neighborhood stays the same and the vertex remains dominated. Therefore, at any point in our construction, the number of active vertices is less than the number of active vertices that are used in [34]. Moreover, since a strong expansion is a coning, the size of the final filtration in our construction is at most that obtained by the construction prescribed in [34]. Moreover, since we are working with 1-skeletons only, the space and time complexity of our method is much lower than that of [34].

Let us now analyze the time complexity of the algorithm. Notice that there are two basic operations on the 1-skeleton, elementary inclusions in Line 1, and the computation and comparison of $ActN_{\mathbb{G}_i}[v \setminus u]$ and $ActN_{\mathbb{G}_i}[u \setminus v]$ in Lines 2.1 and

2.2. Elementary inclusions can be performed in constant time $\mathcal{O}(1)$. To compute $ActN_{\mathbb{G}_i}[v \setminus u]$ we need to compute $N_{\mathbb{G}_i}[v]$, $N_{\mathbb{G}_i}[u]$ and the subset of vertices that are dominated (inactive) in $N_{\mathbb{G}_i}[v \setminus u]$. We can access $N_{\mathbb{G}_i}[v]$ and $N_{\mathbb{G}_i}[u]$ in constant time $\mathcal{O}(1)$ and compute the set-difference $N_{\mathbb{G}_i}[v \setminus u]$ in $\mathcal{O}(k \log k)$ time, where k is the maximum degree of a vertex. Finally computing the dominated vertices in $N_{\mathbb{G}_i}[v \setminus u]$ can be done in $\mathcal{O}(k^3)$ time as $|N_{\mathbb{G}_i}[v \setminus u]| \leq k$. Therefore computing $ActN_{\mathbb{G}_i}[v \setminus u]$ and $ActN_{\mathbb{G}_i}[u \setminus v]$ takes $\mathcal{O}(k^3)$ time. Since the size of active relative closed neighborhoods are bounded by k, for each elementary contraction, we include at most k edges in $\mathcal{O}(k)$ time. It follows that the worst-case time complexity for each elementary contraction is $\mathcal{O}(k^3)$.

We conclude that the time complexity of the algorithm is $\mathcal{O}(|\mathbb{G}_m| + n_c * k^3)$ where n_c is the number of elementary contractions in the input tower and $|\mathbb{G}_m|$ is the size of the 1-skeleton of the output flag filtration. The space complexity of our construction is $\mathcal{O}(n_0 * k)$ which is the size of a sparse adjacency matrix of a flag complex with n_0 vertices.

Correctness: Finally we prove few lemmas and state our main result Theorem 4.2.2 to certify the correctness of the output of our construction.

Lemma 4.2.1. *Let* $f_i : K_i \xrightarrow{\{u,v\}\mapsto u} K_{i+1}$ *be the first elementary contraction in the tower* $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \dots \xrightarrow{f_{m-1}} K_m$. *Then there exists a complex* $K'_{i+1} \subseteq K_{i+1}$ *such that* $K_{i+1} \searrow\searrow K'_{i+1}$ *and* $\mathbb{K}_{i+1} \searrow\searrow K'_{i+1}$.

Proof. Since f_i is the first contraction $\mathbb{K}_i = K_i$ and $\mathbb{G}_i = G_i$. Let $\mathbb{G}_{i+1} := G_i \cup \{[u, ActN_{G_i}[v \setminus u]\}$ be the graph defined in the construction Line 2.1. By construction, contracting v to u in both graphs G_i and $\mathbb{G}_{i+1}$ yields the same graph G_{i+1}.

Let $x' \in ActN_{G_i}[v \setminus u]$. We observe that adding the edge $[ux']$ to G_i does not change the fact that all $x \in \{N_{G_i}[v \setminus u] \setminus ActN_{G_i}[v \setminus u]\}$ are still dominated in $\mathbb{G}_{i+1}$ since the addition of $[ux']$ only adds neighbors to $N_{G_i}[x']$ and $N_{G_i}[u]$. Also, x is still dominated in G_{i+1} as well. Since the only vertex whose domination can change is the one that was dominated by v and the contraction $\{u, v\} \mapsto u$ transfers the neighborhood of v to u and therefore x would now be dominated by u.

Removing all the dominated vertices in $N_{G_i}[v \setminus u]$ thus provides a sequence of elementary strong collapses both in K_{i+1} and $\mathbb{K}_{i+1}$. By performing all such elementary strong collapses, $\mathbb{K}_{i+1}$ is eventually transformed into a complex K^0_{i+1}, $\mathbb{K}_{i+1} \searrow\searrow K^0_{i+1}$ and K_{i+1} to K'_{i+1}, $K_{i+1} \searrow\searrow K'_{i+1}$. See Figure 4.2 for an example of all the graphs associated to the complexes defined here.

By doing so, in $\mathbb{G}_{i+1}$ we have removed all the dominated vertices from $N_{G_i}[v \setminus u]$ and added edges between u and the non dominated vertices that are in $N_{G_i}[v \setminus u]$. This implies that v is dominated by u in K^0_{i+1}. The elementary strong collapse of v onto u, implies $K^0_{i+1} \searrow\searrow K'_{i+1}$ and therefore $\mathbb{K}_{i+1} \searrow\searrow K'_{i+1}$. ☐

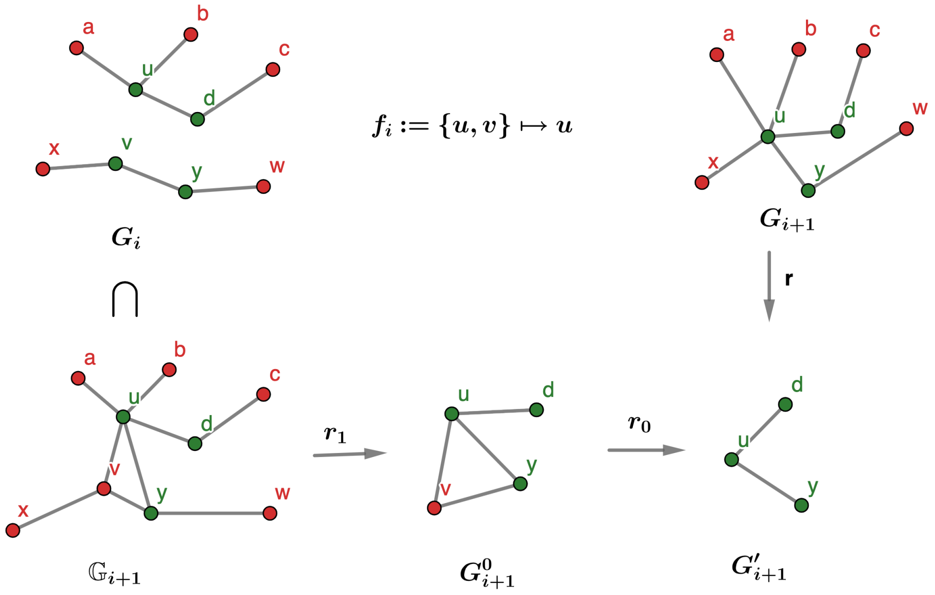


FIGURE 4.2 – An example of the graphs associated to the complexes defined in the proof of Lemma 4.2.1. Vertices in red are dominated (inactive) and in green are active.

Lemma 4.2.2. *Let* $f_i : K_i \xrightarrow{\{u,v\} \mapsto u} K_{i+1}$ *be the first elementary contraction in the tower* $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \dots \xrightarrow{f_{m-1}} K_m$*. Then the following diagram commutes*

$$\begin{tikzcd} H_p(K_i) \arrow[r, "f_i^*"] \arrow[dr, hook, "i_k^*"] & H_p(K_{i+1}) \arrow[d, hook, "\phi^*_{i+1}"] \\ & H_p(\mathbb{K}_{i+1}) \end{tikzcd}$$

and ϕ^*_{i+1} *is an isomorphism.*

Here $\phi_{i+1} := i' \circ r : K_{i+1} \xrightarrow{r} K'_{i+1} \overset{i'}{\hookrightarrow} \mathbb{K}_{i+1}$ *is the composition of the retraction* r *and the inclusion* i' *associated to corresponding strong collapses as defined in Lemma 4.2.1.* i_k^*, f_i^* *and* ϕ^*_{i+1} *are homomorphisms induced by the corresponding simplicial maps.*

Proof. Using the fact that f_i is the first contraction, we have the inclusion $\mathbb{K}_i = K_i \subseteq \mathbb{K}_{i+1}$. Let K^0_{i+1} and K'_{i+1} be the complexes as defined in the proof of Lemma 4.2.1. Consider the following diagram of simplicial complexes, and note that $i' = i_1 \circ i_0$ where i_0 and i_1 are both inclusions induced by the respective strong collapses and $r \circ f_i := K_i \xrightarrow{f_i} K_{i+1} \xrightarrow{r} K'_{i+1}$.

$$\begin{tikzcd}
K_i \arrow[r, "r\circ f_i"] \arrow[d, hook, "i_k"] & K'_{i+1} \arrow[d, hook, "i_0"] \\
\mathbb{K}_{i+1} & K^0_{i+1} \arrow[l, hook', "i_1"]
\end{tikzcd}$$

We claim that the maps $\phi_{i+1} \circ f_i$ and i_k are in the same contiguity class, which we denote $\phi_{i+1} \circ f_i \sim i_k$.

Let r_0 and r_1 be the retraction maps associated with the strong collapses $K^0_{i+1} \searrow\searrow K'_{i+1}$ and $\mathbb{K}_{i+1} \searrow\searrow K^0_{i+1}$ respectively. We have $i_1 \circ i_0 \circ r_0 \circ r_1 \sim 1_{\mathbb{K}_{i+1}}$ [3], where $1_{\mathbb{K}_{i+1}}$ is the identity over $\mathbb{K}_{i+1}$.

Now we observe that the retraction map r_0 is the same as f_i as it is the elementary contraction $\{u, v\} \mapsto u$. And the retraction map $r_1 : \mathbb{K}_{i+1} \to K^0_{i+1}$ is the same as the retraction map $r : K_{i+1} \to K'_{i+1}$ expect for any vertex x that is dominated by v in $\mathbb{K}_{i+1}$. For such x, $r_1(x) = v$ and $r(x) = u$ as x is dominated by u in K_{i+1} and is mapped to u through the map r [see proof of Lemma 4.2.1]. This implies that, the composition $r_0 \circ r_1 = r \circ f_i$ for all vertices in K_i or $\mathbb{K}_{i+1}$ (K_i and $\mathbb{K}_{i+1}$ have the same vertex set), see Figure 4.2 for an illustration.

By replacing $i_1 \circ i_0$ by i' and $r_0 \circ r_1$ by $r \circ f_i$, we have $i_1 \circ i_0 \circ r_0 \circ r_1 = i' \circ r \circ f_i \sim 1_{\mathbb{K}_{i+1}}$. Now since $K_i \subseteq \mathbb{K}_{i+1}$ and K_i and $\mathbb{K}_{i+1}$ have the same vertex set $i_k = 1_{\mathbb{K}_{i+1}}$ also by definition $\phi_{i+1} = i' \circ r$, therefore $\phi_{i+1} \circ f_i \sim i_k$. Since maps in the same contiguity class are homotopic at the level of geometric realizations, the diagram in the lemma commutes.

It follows directly from the definition that ϕ^*_{i+1} is an isomorphism as $\phi^*_{i+1} = (i')^* \circ r^*$ and $(i')^*$ and r^* both are isomorphism since (i') and r are inclusion and retraction maps associated to strong collapse.

□

Proceeding by induction, Lemma 4.2.1 then immediately implies the following result.

Lemma 4.2.3. *Given a tower* $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \dots \xrightarrow{f_{m-1}} K_m$. *For each* $0 \le i \le m$ *, there exists a map* $\phi_i : K_i \to \mathbb{K}_i$ *that is an isomorphism, where* ϕ_i *is a composition of retraction and inclusion associated to strong collapse.*

Again, using an inductive argument along with Lemmas 4.2.2 and 4.2.3, we can deduce the following result.

Theorem 4.2.1. *The following diagram commutes and all the vertical maps* ϕ^*_i *are isomorphisms.*

$$\begin{tikzcd}
H_p(K_1) \arrow[r, "f_1^*"] \arrow[d, "\phi_1^*"] & H_p(K_2) \arrow[r, "f_2^*"] \arrow[d, "\phi_2^*"] & \cdots & {} \arrow[r] & H_p(K_{m-1}) \arrow[r, "f_{m-1}^*"] \arrow[d, "\phi_{m-1}^*"] & H_p(K_m) \arrow[d, "\phi_m^*"] \\
H_p(\mathbb{K}_1) \arrow[r, hook, "*"] & H_p(\mathbb{K}_2) \arrow[r, hook, "*"] & \cdots & {} \arrow[r] & H_p(\mathbb{K}_{m-1}) \arrow[r, hook, "*"] & H_p(\mathbb{K}_m)
\end{tikzcd}$$

As a consequence, the tower $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \dots \xrightarrow{f_{m-1}} K_m$ *and the constructed filtration* $\mathcal{F} : \mathbb{K}_0 \hookrightarrow \mathbb{K}_1 \hookrightarrow \dots \hookrightarrow \mathbb{K}_m$ *have the same persistence diagram.*

Here ϕ_i is a composition of retraction map and inclusion map associated to strong collapse (as defined in Lemma 4.2.1) and therefore an isomorphism for each $i \in \{0, \cdots, m\}$ and $*$ indicates the induced homomorphisms.

We summarize our result in the following theorem. We write $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} ... \xrightarrow{f_{m-1}} K_m$ for the given flag tower where, w.l.o.g., $K_0 = \emptyset$. and each f_i is either an inclusion or an elementary contraction. The inclusions are not necessarily elementary but corresponds to an elementary inclusion on the graphs G_i. We denote by d the maximal dimension of the K_is in $\mathcal{T}$, and by n the total number of elementary inclusions of simplices in $\mathcal{T}$, by n_c the total number of elementary contractions and by n_0 the number of vertex inclusions in $\mathcal{T}$.

Theorem 4.2.2. *There exists a filtration $\mathcal{F} : \mathbb{K}_0 \hookrightarrow \mathbb{K}_1 \hookrightarrow ... \hookrightarrow \mathbb{K}_m$, where the inclusions are not necessarily elementary, such that $\mathcal{T}$ and $\mathcal{F}$ have the same persistence diagram and the size of the filtration $|\mathbb{K}_m|$ is at most $\mathcal{O}(d * n * \log n_0)$. Moreover, $\mathcal{F}$ is a flag filtration which can be computed from $\mathcal{T}$ using only the 1-skeletons G_is of the K_is. The time complexity of the algorithm to compute the 1-skeleton of $\mathcal{F}$ is $O(|\mathbb{G}_m| + n_c * k^3)$ time and its space complexity $\mathcal{O}(n_0 * k)$, where $\mathbb{G}_m$ denotes the 1-skeleton of $\mathbb{K}_m$ and k is an upper bound on the degree of the vertices in $\mathbb{G}_m$.*

4.3 Computational experiments

Next we describe the complete pre-processing pipeline for a tower and a general approximation scheme for any tower and then perform experiments on Rips filtrations, similar to what we did in the previous chapter. However, we use the approach developed in this chapter for our experiments.

The pre-processing pipeline. Let $\mathcal{T} : \{K_1 \xrightarrow{f_1} K_2 \xrightarrow{f_2} \cdots \xrightarrow{f_{(m-1)}} K_m\}$ be a flag tower of which we want to compute the persistence diagram. We first strong collapse the various complexes K_i, $i = 1, ..., m$ as suggested in Section 3.4. As each K_i is a flag complex, the computation of its core is computed much more efficiently using only its 1-skeleton.

We then compute a *core tower* that connects the K_i^cs through induced simplicial maps. This is again an adaptation of what has been done in Chapter 3 for general simplicial complexes to the case of flag complexes. Lastly, we compute a flag *filtration* with the same PH as the core tower, using our approach detailed in Section 4.2. This filtration can then be sent to any algorithm that computes the persistence homology of a flag filtration [41, 4, 5, 30].

An approximation scheme. Using the above approach we compute the exact PH of the input flag tower $\mathcal{T}$. However, instead of considering all the K_is and the associated simplicial maps, we can select some of the K_is we rename $K'_1, ..., K'_q$, and compose the original maps $f_1, ..., f_{m-1}$ to obtain simplicial maps $f'_1, ..., f'_{q-1}$ connecting the selected complexes. We thus obtain a sub-tower $\mathcal{T}'$ and the algorithm will then compute the exact PH of $\mathcal{T}'$. An application of this idea has been presented for Rips filtrations in Section 3.5 of the previous chapter, which we describe again below.

Recall that, for Rips filtrations, the filtration value of a simplex is the length of the longest edge of the simplex. Given a Rips filtration, one can choose to collapse the original complexes after each edge inclusion. However, we can also choose to strong collapse the complexes less often, i.e. after several edge inclusions rather than just one. This will result in a faster algorithm but comes with a cost: the computed PD is then only approximate. We call **snapshots** the values of the scale parameter at which we choose to strong collapse the complex. The difference between two consecutive snapshots is called a **step**. We approximate the filtration value of a simplex as the value of the snapshot at which it first appears. We can observe that our algorithm will report all persistence pairs that are separated by at least one snapshot. Hence if all steps are equal to some $\epsilon > 0$, we will compute all the persistence pairs whose lengths are at least ϵ. It follows that the l_∞-*bottleneck distance* between the computed PD and the exact one is at most ϵ. If instead the ratio between any two consecutive steps is taken to a constant $\rho > 1$, the l_∞-bottleneck distance will be at most $\log \rho$ after reparameterizing the filtrations on a log-log-scale [45].

Experimental setup : Our algorithm has been implemented for Rips filtrations as a C++ module named VertexCollapser. VertexCollapser takes as input a Rips filtration and returns the reduced flag filtration. VertexCollapser can be used in two modes: in the exact mode, the output filtration has the same PD as the input filtration while, in the approximate mode to be described above, a certified approximation is returned. The output filtration can then be sent to any software that computes the PD of a Rips filtration such as the Gudhi library (the one we chose for our experiments) [30] or Ripser [4]. Therefore VertexCollapser is to be considered as an ad-on to software computing PD. VertexCollapser will be available as an open-source package of a next release of the Gudhi library.

We present results on five datasets **netw-sc**, **senate**, **eleg**, **HIV** and **drag 1** that are publicly available [18]. Each dataset is given as the interpoint distance matrix. The reported time includes the time of VertexCollapser and the time to compute the persistent diagram (PD). The time of VertexCollapser includes: 1. The time taken to compute the largest 1-skeleton associated to the maximum threshold value, 2. The time taken to collapse all the sub-skeletons and assemble their cores. 3. To transform them into an equivalent flag-filtration.

The code has been compiled using the compiler 'clang-900.0.38' and all computations were performed on a '2.8 GHz Intel Core i5' machine with 16 GB of available RAM. We took all steps to be equal. Parameter *Step* controls the quality of the approximation, if $Step = 0$, we obtain the exact PD, otherwise *Step* is an upper bound on the l_∞-*bottleneck distance* between the output diagram and the exact one. VertexCollapser works irrespective of the dimension of the input complexes. However, the size of the complexes in the reduced filtration, even if much smaller than in the original filtration, might exceed the capacities of the PD computation algorithm. For this reason, we introduced a parameter *dim* and restricts PD computation to dimension at most *dim*.

Some experimental results are reported in Table 4.1, some of them are the same as done in Table 3.2 of Chapter 3. As previously observed in Chapter 3 the reduction done by VertexCollapser is enormous as well, same as before. However, the main point to observe is that VertexCollapser is extremely fast due to its ability to work with only the 1-skeleton. The reduced complexes are small and of low dimension (column Size/Dim) compared to the input Rips complexes which are of dimensions respectively 57, 54 and 105 for the first three datasets **netw-sc**, **senate** and **eleg**. We also observe that, while the time taken by VertexCollapser is large for exact persistence computation, very good approximations can be obtained fast. Moreover the computing time mildly increases with the number of snapshots. This suggests that implementing the collapses in parallel would lead to further substantial improvement.

Data	Pnt	Thrsld	VertexCollapser +PD					
			Size/Dim	*dim*	Pre-Time	Tot-Time	*Step*	Snaps
netw-sc	379	5.5	155/3	∞	7.28	7.38	0.02	263
″	″	″	155/3	∞	13.93	14.03	0.01	531
″	″	″	175/3	∞	366.46	366.56	0	8420
senate	103	0.415	405/4	∞	2.53	2.54	0.001	403
″	″	″	417/4	∞	15.96	15.98	0	2728
eleg	297	0.3	577K/15	∞	11.65	26.02	0.001	284
″	″	″	835K/16	∞	518.36	540.40	0	9850
HIV	1088	1050	127.3M/?	4	660	3,955	4	184
drag1	1000	0.05	478.3M/?	4	687	14,170	0.0002	249

TABLE 4.1 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by VertexCollapser, total time (in seconds) including PD computation (Tot-Time), parameter *Step* and the number of snapshots used (Snaps). K and M in the size columns are thousands and millions respectively.

Comparison with Ripser: We again compare our results with Ripser. As mentioned before in Chapter 3, Ripser [4] is the state of the art software to compute persistent diagrams of Rips filtrations. It computes the *exact* PD associated to the input filtration up to dimension *dim*. Although VertexCollapser is more complementary to Ripser than a competitor, we run Ripser[1] on the same datasets as in Table 4.1 to demonstrate the benefit of using VertexCollapser.

Results are presented in Table 4.2. The main observation is that Ripser performs quite well in low dimensions but its ability to handle higher dimensions is limited.

Data	Pnt	Threshold	Val		Val		Val	
			dim	Time	*dim*	Time	*dim*	Time
netw-sc	379	5.5	4	25.3	5	231.2	6	∞
senate	103	0.415	3	0.52	4	5.9	5	52.3
″	″	″	6	406.8	7	∞		
eleg	297	0.3	3	8.9	4	217	5	∞
HIV	1088	1050	2	31.35	3	∞		
drag1	1000	0.05	3	249	4	∞		

TABLE 4.2 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), parameter (*dim*), Time is the total time (in seconds) taken by Ripser. ∞ means that the experiment ran longer than 12 hours or crashed due to memory overload.

This chapter was based on our work reported in [9].

---

[1] We used the command <./ripser inputData –format distances –threshold inputTh –dim inputDim >.

Chapter 5

Edge Collapse And Persistence

In this chapter, we extend the notions of dominated vertex and strong collapse of a simplicial complex discussed and used for PH computation in the previous chapters. We say that a simplex (of any dimension) is dominated if its link is a simplicial cone. Domination of edges appear to be very powerful and we study it in the case of flag complexes in more detail. We show that edge collapse (removal of dominated edges) in a flag complex can be performed using only the 1-skeleton of the complex. Furthermore, the residual complex is a flag complex as well.

Edge collapses do not have an associate simplicial maps like strong collapses. However, we show that similar to the case of strong collapses, we can use edge collapse to reduce a flag filtration $\mathcal{F}$ to a smaller flag filtration $\mathcal{F}^c$ with the same persistence. Here again, we only use the 1-skeletons of the complexes. To be able to use edge collapse to simplify a filtration, we prove a much more general result and show that simple collapses preserves persistence, a result that could be seen as an extension of Theorem 3.4.1 of Chapter 3.

The resulting method to compute $\mathcal{F}^c$ is simple and extremely efficient and, when used as a preprocessing for persistence computation, leads to gains of several orders of magnitude with respect to the state-of-the-art methods (including our previous approach using strong collapse). The method is exact, irrespective of dimension, and improves performance of persistence computation even in low dimensions. Again this is demonstrated by numerous experiments on publicly available data.

In Section 5.1 we define Edge collapse and discuss some of its theoretical and algorithmic properties. In Section 5.2 we prove that simple collapse preserves the persistence of a sequence of simplicial complex. The algorithm to simplify a flag filtration to a smaller flag filtration using edge collapse is described in Section 5.3. In Section 5.4 we provide some computational experiments.

5.1 Edge Collapse

In this section, we first extend the definition of a dominated vertex introduced in [3] to simplices of any dimension. Given a simplex $\sigma \in K$, we denote by Σ_σ the set of maximal simplices of K that contain σ. The intersection of all the maximal simplices in Σ_σ will be denoted as $\bigcap \Sigma_\sigma := \bigcap_{\tau \in \Sigma_\sigma} \tau$.

Dominated simplex. A simplex σ in K is called a **dominated simplex** if the link $lk_K(\sigma)$ of σ in K is a simplicial cone, i.e. if there exists a vertex $v' \notin \sigma$ and a subcomplex

L of K, such that $lk_K(\sigma) = v'L$. We say that the vertex v' is *dominating* σ and that σ is *dominated* by v', which we denote as $\sigma \prec v'$.

***k*-collapse.** Given a complex K, the action of removing a dominated k-simplex σ from K is called an **elementary *k*-collapse**, denoted as $K \searrow\searrow^k \{K \setminus \sigma\}$. A series of elementary k-collapses is called a ***k*-collapse**, denoted as $K \searrow\searrow^k L$. We further call a complex K ***k*-collapse minimal** if it does not have any dominated k simplices. A subcomplex K^k of K is called a ***k*-core** if $K \searrow\searrow^k K^k$ and K^k is k-collapse minimal.

The notion of k-collapse is the same as the notion of *extended collapse* introduced in [2]. We give it a different name to indicate the dependency on the dimension. A 0-collapse is a strong collapse as introduced in [3]. A 1-collapse will be called an **edge collapse**. It is not hard to see that an elementary simple collapse of a k-simplex σ is a k-collapse, as it is dominated by the vertex $v = \tau \setminus \sigma$, where τ is the unique coface containing σ. Each k-collapse can be decomposed into a sequence of elementary simple collapses and therefore k-collapses preserve the simple homotopy type [46, Lemma 2.7] and [2, Lemma 8]. Therefore, like simple collapses, k-collapses induce a strong deformation retract as well on the geometric realization.

The following lemma extends a result in [3] to general k-collapse. It shows that the domination of a simplex can be characterized in terms of maximal simplices.

Lemma 5.1.1. *A simplex $\sigma \in K$ is dominated by a vertex $v' \in K$, $v' \notin \sigma$, if and only if all the maximal simplices of K that contain σ also contain v', i.e. $v' \in \bigcap \Sigma_\sigma$.*

Proof. If $\sigma \prec v'$ then $lk_K(\sigma) = v'L$ by definition. This implies that for any maximal simplex τ in $st_K(\sigma)$, $v' \in \tau$. Therefore, $v' \in \bigcap \Sigma_\sigma$. For the reverse direction, let $v' \in \bigcap \Sigma_\sigma$. Hence, for any maximal simplex τ in $st_K(\sigma)$, we have $v' \in \tau$. Now as $v' \notin \sigma$, v' belong to all the simplices $\tau \setminus \sigma$, and thus $lk_K(\sigma) = v'L$ where $L = (\tau \setminus \sigma) \setminus v'$. Hence $\sigma \prec v'$ iff $v' \in \bigcap \Sigma_\sigma$. □

Lemma 5.1.1 has important algorithmic consequences. To perform a k-collapse, one simply needs to store the adjacency matrix between the k-simplices and the maximal simplices of K.

Next we study the special case of a flag complex K and characterize the domination of a simplex σ of a flag complex K in terms of its neighborhood.

Lemma 5.1.2. *Let σ be a simplex of a flag complex K. Then σ will be dominated by a vertex v' if and only if $N_G[\sigma] \subseteq N_G[v']$.*

Proof. Assume that $N_G[\sigma] \subseteq N_G[v']$ and let τ be a maximal simplex of K that contains σ. For a vertex $x \in \tau$ and for any vertex $v \in \sigma$, the edge $[x, v] \in \tau$. Therefore $x \in N_G[\sigma] \subseteq N_G[v']$. Every vertex in τ is thus linked by an edge to v' and, since K is a flag complex and τ is maximal, v' must be in τ. This implies that all the maximal simplices that contains σ also contain v'. Hence σ is dominated by v'.

Consider the other direction. If $\sigma \prec v'$, by Lemma 5.1.1, all the maximal simplices that contain σ also contain v'. This implies $N_G[\sigma] \subseteq N_G[v']$. □

Lemma 5.1.2 is a generalisation of Lemma 1 in [9]. The next lemma, though elementary, is of crucial significance. Both lemmas show that edge collapses are well-suited to flag complexes.

Lemma 5.1.3. *Let K be a flag complex and let L be any subcomplex of K obtained by edge collapse. Then L is also a flag complex.*

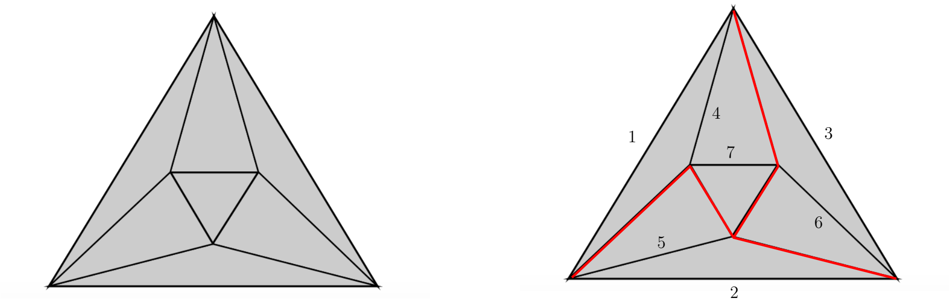


FIGURE 5.1 – The above complex does not have any dominated vertex and thus cannot be 0-collapsed. However, by proceeding from the boundary edges, one can edge collapse this complex to a 1-dimensional complex. The 1-core obtained in this way is collapsible to a point using 0-collapse.

Efficiency of reduction. As will be demonstrated in Section 5.4, edge collapse appears to be a very efficient tool to reduce the size of a complex while preserving its homotopy type. A simple example will help giving some intuition why edge collapse can be superior to vertex collapse. See Figure 5.1.

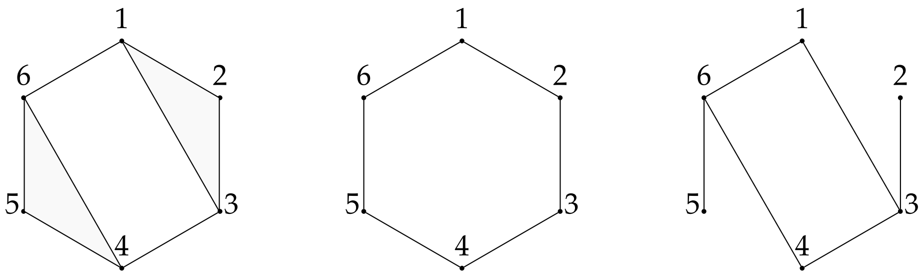


FIGURE 5.2 – The complex on the left has two different 1-cores, the one in the middle is obtained after removing the inner edges $[1,3]$ and $[4,6]$, and the one in the right by removing the outer edges $[1,2]$ and $[4,5]$. Note that the one in the right can be further strong collapsed.

5.2 Simple Collapse and Persistence

In this section, we turn our attention to the general case of simple collapses (of which k-collapses are a special case) and provide one of the main result of this chapter. This can be seen as a generalization of Theorem 2 of [10].

Theorem 5.2.1. *Let $f : K \rightarrow L$ be a simplicial map between two complexes K and L and let $K' \subset K$ and $L' \subset L$ be subcomplexes of K and L such that $K \searrow K'$ and $L \searrow L'$. Then there exists a map $f' : K' \rightarrow L'$, induced by f, such that the persistence of $f^* : H_p(K) \rightarrow H_p(L)$ and $f'^* : H_p(K') \rightarrow H_p(L')$ are the same for any integer $p \geq 0$. The induced map f' may not be simplicial. Nevertheless, it can be expressed as a combination of inclusions, contractions and removals of simplices.*

Proof. Let us consider the following diagram between the geometric realizations of the complex $|K|$, $|L|$, $|K'|$ and $|L'|$.

$$\begin{tikzcd}
|K| \arrow[r, "|f|"] \arrow[d, shift left, "|r_k|"] & |L| \arrow[d, shift left, "|r_l|"] \\
|K'| \arrow[u, shift left, "|i_k|"] \arrow[r, dotted, "|f'|"] & |L'| \arrow[u, shift left, "|i_l|"]
\end{tikzcd}$$

and the associated diagram after computing the p-th singular homology groups

$$\begin{tikzcd}
H_p^o(|K|) \arrow[r, "|f|^*"] \arrow[d, shift left, "|r_k|^*"] & H_p^o(|L|) \arrow[d, shift left, "|r_l|^*"] \\
H_p^o(|K'|) \arrow[u, shift left, "|i_k|^*"] \arrow[r, dotted, "|f'|^*"] & H_p^o(|L'|) \arrow[u, shift left, "|i_l|^*"]
\end{tikzcd}$$

Here $|r_k|$ and $|r_l|$ are the deformation retractions on the geometric realizations associated with the simple collapse and $|i_k|$ and $|i_l|$ are the inclusion maps. $H_p^o()$ denotes the singular homology and * is the induced homomorphisms by the corresponding continuous maps. The map $|f'|$ is defined as $|f'| := |r_l||f||i_k|$. Hence $|f'||r_k| = |r_l||f||i_k||r_k|$. Now observe that, since $|r_k|$ is a deformation retraction, $|i_k||r_k|$ is homotopic to the identity over $|K|$. It follows that $|r_l||f||i_k||r_k|$ is homotopic to $|r_l||f|$. Since homotopic maps induce identical homomorphisms on the corresponding homology groups [31, Proposition 2.19], we deduce that $|f'|^*|r_k|^* = |r_l|^*|f|^*$ (commutativity). Also, since $|r_k|^*$ and $|r_l|^*$ are induced by deformation retractions, they are isomorphisms on their respective singular homology groups. We have thus proved that the above diagram commutes and that the vertical maps $|r_k|$ and $|r_l|$ are isomorphisms. This implies that the two maps $|f| : |K| \to |L|$ and $|f'| : |K'| \to |L'|$ have the same singular persistent homology.

The map $|f'|$ induces a map $f' := r_l \circ f \circ i_k$ between the simplicial complexes K' and L'. Note that f' can be expressed as a composition of inclusions, contractions and removals of simplices, as i_k is an inclusion, f is simplicial and r_l is a simple collapse. Also, for simplicial complexes, singular homology is isomorphic to simplicial homology [31, Theorem 2.27]. This implies that the persistent singular homology $|f'|^* : H_p^o(|K'|) \to H_p^o(|L'|)$ and the persistent simplicial homology $f'^* : H_p(K') \to H_p(L')$ are equivalent. Therefore, the persistent simplicial homologies $f^* : H_p(K) \to H_p(L)$ and $f'^* : H_p(K') \to H_p(L')$ are equivalent. □

The use of singular homology in the proof is due to the lack of a simplicial map associated with the retraction ($|r|$) of a simple collapse. Due to the same reason, the induced map $f' : K' \to L'$ may not be necessarily simplicial. However, as mentioned in the above proof the map f' can be expressed as a combination of inclusions, contractions and removals of simplices. When a sequence of simplicial complexes contains removals of simplices, it is called a zigzag sequence. There are algorithms [41, 37] to compute zigzag persistence but they are not as efficient as the usual algorithms for filtrations and towers.

In the next section, we consider the case of flag filtrations and show that we can restrict the way the edge collapses are performed so that the reduced filtration is also a flag filtration.

5.3 Edge collapse of a flag filtration

In Section 5.1, we have introduced edge collapse for general simplicial complexes and provided an easy criterion for edge-domination in a flag complex using only the 1-skeleton of the complex. In this section, we provide an algorithm to simplify a flag filtration using edge collapse and again using only the 1-skeleton of the complex.

We define a notion of **removable edge** to help explain how our algorithm works (Algorithm 4) and to prove its correctness. Let G be a graph and K be the associated flag complex. We say that an edge e in a graph G is removable either if it is dominated in K or if there exists a sequence of edge collapses $K \searrow\searrow^1 K^c$ such that e is dominated in the reduced complex K^c. Our algorithm is based on the fact that the flag complexes K and K^c are homotopy equivalent [46, Lemma 2.7] and [2, Lemma 8]. If $e = [u, v]$, we define the **edge-neighborhood** of an edge $e \in G$ as the set $EN_G(e) := \{[x, y], x \in \{u, v\}, y \in N_G([uv])\}$.

Algorithm. Let $\mathcal{F} : K_1 \hookrightarrow K_2 \hookrightarrow \cdots \hookrightarrow K_n$ be a flag filtration and $\mathcal{G}_{\mathcal{F}} : G_1 \hookrightarrow G_2 \hookrightarrow \cdots \hookrightarrow G_n$ be the associated sequence of 1-skeletons. We further assume that $G_i \hookrightarrow G_{i+1}$ is an elementary inclusion, namely the inclusion of a single edge we name e_{i+1}. The edges in $E := \{e_1, ..., e_n\}$ are thus indexed by their order in the filtration and we denote by G_i the subset $\{e_1, ..., e_i\}$. Our algorithm computes a subset of edges $E^c \subseteq E$ and attach to each edge in E^c a new index. We thus obtain a new sequence of flag complexes $\mathcal{F}^c$ corresponding to E^c, we call the *core sequence*. The construction of E^c and of the new indices is done so that $\mathcal{F}^c$ has the same persistence diagram as $\mathcal{F}$.

Let's give an intuitive presentation of the algorithm first. The central idea is to identify edges that appear to be non-removable at some point in the algorithm. We store such edges in a set E^c. To be more specific, consider the case of the inclusion of an edge $G_{i-1} \overset{e_i}{\hookrightarrow} G_i$ such that e_i is dominated in G_i : e_i is thus removable in G_i and is not included in E^c. Suppose first that all further edges e_s are dominated in G_s, $i < s \leq n$. Then e_i remains removable and will never be put in E^c. This is consistent with the fact that e_i does not change the topology of the complexes K_s and is therefore not required when computing persistence.

Assume now that some edge e_p, $i < p \leq n$, is non-dominated in G_p. The status of e_i, that was removable in all G_s for $s < p$, may change to non-removable in G_p. Therefore, we check whether e_i is non-removable in G_p (by proceeding in the reverse filtration order) and, in the affirmative, include e_i in E^c. In turn, the fact that e_i changed from removable to non removable may change the status of the edges with smaller indices which could become non-removable after the inclusion of e_i. If such edges are found, they are also included in E^c.

Before describing the algorithm in detail, two remarks are in order. First, we do not change the status of an edge from non-removable to removable even if it has become removable: this will enforce the output sequence to be a filtration. Second, we change the filtration values of some edges: the new filtration value of an edge is the first index at which it is found to be non-removable. The second point leads to faster computation of E^c, otherwise one has to proceed backward recursively to search for new non-removable edges.

We now explain how to compute E^c. See [Algorithm 4] for the pseudo-code. The main **for** loop on line 6 (called the forward loop) iterates over the edges in the filtration $\mathcal{F}$

by increasing filtration values, i.e. in the *forward direction*, and check whether or not the current edge e_i is dominated in the graph G_i. If *not*, we insert e_i in E^c and keep its original index i.

After the insertion of an edge e_i in E^c, we proceed to the so-called backward loop ([Lines 9-26]) and look for new non-dominated edges in G_i, considering the edges by decreasing filtration values. We assign G_i to a temporary graph G, and we assign the edge-neighborhood of e_i in the graph G_i to E^{nbd} [Line 9-10]. As established in Lemma 5.3.1, the search for new non-dominated edges can be restricted to E^{nbd}. If an edge e_j is not in E^c and not in E^{nbd} [Line 13-14], e_j is still dominated : we then remove it from G [Line 22]. If $e_j \notin E^c$ and $e_j \in E^{nbd}$, then we check whether it is dominated or not. If e_j is dominated, we remove it from G [Line 19]. Otherwise, we insert e_j in E^c and assign to it the *new index* i, i.e. the index of the edge e_i that has triggered the backward search in G_i. Next we enlarge the edge-neighborhood E^{nbd} by inserting the edge-neighbors of e_j in G. We repeat this process until the last index $j = 1$. Upon termination of the forward loop [Line 6-30], we output E^c as the final set.

Algorithm 4 Core flag filtration algorithm

```
 1: procedure CORE-FLAG-FILTRATION($E$)
 2:     input : set of edges $E$ of $\mathcal{G}_{\mathcal{F}}$ sorted by filtration value.
 3:     $E^c \leftarrow \emptyset$; $i \leftarrow 1$;
 4:     $E^{nbd} \leftarrow \emptyset$
 5:     $G \leftarrow \emptyset$
 6:     for $e_i \in E$ do                          ▷ For $i = 1, ..., n$ in increasing order
 7:         if $e_i$ is non-dominated in $G_i$ then
 8:             Insert $\{e_i, i\}$ in $E^c$.
 9:             $G \leftarrow G_i$
10:             $E^{nbd} \leftarrow EN_{G_i}(e_i)$
11:             $j \leftarrow i - 1$
12:             for $e_j$ in $G_i$ do               ▷ For $j = (i-1), ..., 1$ in decreasing order
13:                 if $e_j \notin E^c$ then
14:                     if $e_j \in E^{nbd}$ then
15:                         if $e_j$ is non-dominated in $G$ then
16:                             Insert $\{e_j, i\}$ in $E^c$.
17:                             $E^{nbd} \leftarrow E^{nbd} \cup EN_G(e_j)$
18:                         else
19:                             $G \leftarrow G \setminus e_j$
20:                         end if
21:                     else
22:                         $G \leftarrow G \setminus e_j$
23:                     end if
24:                 end if
25:                 $j \leftarrow j - 1$
26:             end for
27:         end if
28:         $G \leftarrow \emptyset$
29:         $i \leftarrow i + 1$
30:     end for
31:     return $E^c$                  ▷ $E^c$ is the 1-skeleton of the core flag filtration.
32: end procedure
```

The computation of non-removable edges (the set E^c) is dependent on the order in which we do the backward search (the backward loop). In Algorithm 4 we chose to proceed in the reverse order of the filtration. A different choice of order might result in a different set of non-removable edges since edge collapses are order dependent as mentioned in Section 5.1.

We now prove the correctness of the above algorithm after some more definitions.

Critical Edges: Edges in E^c are called **critical** while edges in $E \setminus E^c$ are called **non-critical**. All edges have an original index i given by the insertion order in the input filtration $\mathcal{F}$. The critical edges received a second index j, called their **critical index**, when they are inserted in E^c. By convention, if an edge is not critical and thus has never been inserted in E^c, we will set its critical index to be ∞. Hence, at the end of Algorithm 4, each edge $e \in E$ has two indices, an original and a critical index. To make this explicit, we denote e as e_i^j. Clearly $i \leq j$. We further distinguish the cases $i = j$ and $i < j$. If $i = j$, e_i has been put in E^c during the forward loop and we call e_i a **primary critical edge**. If $i < j$, e_i has been put in E^c during the backward loop and we call it a **secondary critical edge**.

For $i = 1, ..., n$, we define the **critical graph** at index i, denoted G_i^c, as the graph whose edges are the edges in E^c with a critical index at most i. We denote the associated flag complex as K_i^c.

Correctness. We now prove some lemmas to certify the correctness of our algorithm.

The following lemma justifies the fact that the search for new critical edges during the backward loop of Algorithm 4 is restricted to the neighborhood of already found critical edges.

Lemma 5.3.1. *Let e be an edge in a graph G and let e' be a new edge and $G' := G \cup e'$. If e is dominated in G and $e \notin EN_{G'}(e')$, then e is dominated in G'.*

Proof. Let $e \prec v'$ in G, then $N_G[e] \subseteq N_G[v']$. Plainly, $N_G[v'] \subseteq N_{G'}[v']$ and, since $e \notin EN_{G'}(e')$, $N_{G'}[e] = N_G[e]$. Therefore, $N_{G'}[e] = N_G[e] \subseteq N_{G'}[v']$ implies $e \prec v'$ in G'. □

The following lemma says that a non-critical edge is always removable and that a critical edge is removable until it becomes critical.

Lemma 5.3.2. *Let e_i^j be an edge with $i < j$, then it is removable in all G_t, $i \leq t < \min(n+1, j)$.*

Proof. According to the algorithm, if $i < j$, e_i^j is dominated in G_i (j being finite or not).

1. Let us first consider the case $j = \infty$. Note that e_i^∞ is non-critical and let j_i be the smallest primary critical index greater than i. If no such index exists, set $j_i = n+1$. We show by induction that e_i^∞ remains removable in all G_t, $i \leq t < n+1$. As shown above, it is true for $t = i$ since e_i^j is dominated in G_i. So assume that e_i^j is removable in G_{t-1} and consider the insertion of e_t in G_t, for some $t < j_i$. By definition of j_i, e_t is dominated in G_t, which implies that e_i^j is removable in G_t (in the backward sequence $e_t, e_{t-1}, \ldots, e_i$).

Consider now $t = j_i$. Since e_{j_i} is a primary critical edge, it is non-dominated in G_{j_i}. According to the algorithm, a backward loop has been triggered at j_i. During this backward loop, e_i^∞ has *not* been inserted in E^c since its second critical index is ∞. This is only possible because e_i^∞ has been found to be dominated in G. Since G is initialized as G_{j_i}, it follows that e_i^∞ is removable in G_{j_i}. We can now proceed in a similar way for all $t, j_i < t < n+1$.

2. The proof is very similar for the case $i < j \leq n$. As e_i^j has not been inserted in E^c until the backward loop triggered at index j, e_i^j remains removable in all G_t, $i \leq t < j$. ☐

Note that our statement does not imply that a critical edge e_i^j, $i < j \leq n$, can never be removable in G_t, $t \geq j$. It just means that we are sure that it will remain removable until the point it becomes critical.

Lemma 5.3.3. *For each i, Algorithm 4 produces a sequence of elementary edge collapses such that $K_i \searrow\searrow^1 K_i^c$.*

Proof. By definition, $G_i \setminus G_i^c = \{e_t^m \mid t \leq i, m > i\}$ is the set of edges of G_i whose critical index m is greater than i, which includes the non-critical edges ($m = \infty$). Any edge $e_t^m \in G_i \setminus G_i^c$ is removable in all $K_j, j < m$ by Lemma 5.3.2. ☐

The proof of the following theorem certifying the correctness of our algorithm follows directly through the application of Lemma 5.3.3 and Theorem 5.2.1.

Theorem 5.3.1. *Let $\mathcal{F} : K_1 \hookrightarrow K_2 \hookrightarrow \cdots \hookrightarrow K_n$ be a flag filtration and $\mathcal{G}_\mathcal{F} : G_1 \hookrightarrow G_2 \hookrightarrow \cdots \hookrightarrow G_n$ be the associated sequence of 1-skeletons, such that $G_i \hookrightarrow G_{i+1}$ is an elementary inclusion of an edge e_{i+1}. Let G_i^c be the critical graph and K_i^c be its flag complex as defined before. Then the associated flag filtration of the critical edges, $\mathcal{F}^c : K_1^c \hookrightarrow K_2^c \hookrightarrow \cdots \hookrightarrow K_n^c$ has the same persistence diagram as $\mathcal{F}$.*

Proof. Let us consider the following diagram of the geometric realizations of the flag complexes for any $i \in \{1, ..., n\}$, where K_i^c is the flag complex of the critical graph G_i^c.

$$\begin{tikzcd}
{|K_i|} \arrow[r, hook] \arrow[d, shift left, "{|r_i|}"] & {|K_{i+1}|} \arrow[d, shift left, "{|r_{i+1}|}"] \\
{|K_i^c|} \arrow[r, hook] \arrow[u, hook, shift left] & {|K_{i+1}^c|} \arrow[u, hook, shift left]
\end{tikzcd}$$

Using Lemma 5.3.3, there is an edge collapse and therefore a simple collapse from K_i to K_i^c and from K_{i+1} to K_{i+1}^c. And $|r_i|$ and $|r_{i+1}|$ are the deformation retractions induced by the corresponding edge collapses. The equivalence of the persistence modules then follows directly from the application of Theorem 5.2.1.

☐

Complexity: Write n_v for the total number of vertices, n for the total number of edges and k for the maximum degree of a vertex in G_n. We represent each graph G_i as an adjacency list, where every vertex stores a *sorted* list of at most k adjacent vertices. Additionally, we store the set of edges (E and E^c) as a separate data structure.

The cost of inserting and removing an edge from such an adjacency list is $\mathcal{O}(k)$. Since the size of $N_G[v]$ is at most k for any vertex v, the cost of computing $N_G[e]$ for an edge e is $\mathcal{O}(k)$. Checking if an edge e is dominated by a vertex $v \in N_G[e]$ reduces to checking whether $N_G[e] \subseteq N_G[v]$, see Lemma 5.1.2. Since all the lists are sorted, this operation takes $\mathcal{O}(k)$ time per vertex v, hence $\mathcal{O}(k^2)$ time in total.

Let us now analyze the worst-case time complexity of Algorithm 4. At each step i of the forward loop [Line 6], either e_i is dominated (which can be checked in $O(k^2)$ time) or a backward loop is triggered [Line 12]. The backward loop will consider all edges with (original) index at most i and check whether they are dominated or not. Writing n_c for the number of primary critical edges, the worst-case time complexity is $nk^2 + \sum_{i=1}^{n_c} n\, k^2 = \mathcal{O}(nn_ck^2)$. The space complexity is $\mathcal{O}(n)$. In practice, n_c is a small fraction of n (see Table 5.1).

5.4 Computational Experiments

Our algorithm [Algorithm 4] has been implemented for Rips filtrations as a C++ module named EdgeCollapser. Our previous preprocessing method described in Chapter 4 to simplify Rips filtrations using strong collapse is called the VertexCollapser. Both EdgeCollapser and VertexCollapser take as input a Rips filtration and return the reduced flag filtration according to their respective algorithms.

We present results on five datasets **netw-sc**, **senate**, **eleg**, **HIV** and **torus**. The first four datasets are publicly available [18] and are given as the interpoint distance matrix of the points. The last dataset **torus** has 2000 points sampled in a spiraled fashion on a torus embedded in a 3-sphere of $\mathbf{R}^4$ [32]. The reported time includes the time of EdgeCollapser/VertexCollapser and the time to compute the persistence diagram (PD) using the Gudhi library [30].

The code has been compiled using the compiler 'clang-900.0.38' and all computations were performed on a '2.8 GHz Intel Core i5' machine with 16 GB of available RAM. Both EdgeCollapser and VertexCollapser work irrespective of the dimension of the complexes associated to the input datasets. However, the size of the complexes in the reduced filtration, even if much smaller than in the original filtration, might exceed the capacities of the PD computation algorithm. For this reason, we introduced, as in Ripser, a parameter *dim* and restricts the expansion of the flag complexes to a maximal dimension *dim*.

The experimental results using EdgeCollapser are summarized in Table 5.1. Observe that the reduction in the number of edges done by EdgeCollapser is quite significant. The ratio between the number of initial edges and the number of critical edges is approximately 20. If the number of edges in a graph is $|E|$ then the size of the $(k+1)$-cliques $\mathcal{O}(|E|^k)$. Therefore the reduction in the size of k-simplices can be as large as $\mathcal{O}(20^k)$. This is verified experimentally too, as the reduced complexes are small and of low dimension (column Size/Dim) compared to the input Rips-complexes which are of dimensions respectively 57, 54 and 105 for the first three datasets **netw-sc**, **senate** and **eleg**. [1]

Comparison with VertexCollapser. The same set of experimental results using VertexCollapser are summarized in Table 5.2, which is a subset of the experiments reported in the previous chapter in Table 4.1. As mentioned before VertexCollapser can be used in two modes: in the exact mode (step=0), the output filtration has the same PD as the input filtration while, in the approximate mode (step>0), a certified approximation is returned. For comparison, we use the results in Table 5.2 are of exact mode, for experiments in approximate mode please refer to Table 4.1.

---

[1] The sizes of the complexes are so big that we could not compute the exact number of simplices.

It can be seen that EdgeCollapser is faster than VertexCollapser by approximately two orders of magnitude. The main reason for this is the efficient preprocessing algorithm behind EdgeCollapser. As it can be noticed in some cases, the reduction obtained using by VertexCollapser is better than using EdgeCollapser, but even in those cases EdgeCollapser is faster than VertexCollapser.

In terms of size reduction, EdgeCollapser either outperforms VertexCollapser by a big amount or is comparable. Some intuition can be gained from the case of *torus*. This is a well distributed point sets sampled from a manifold without boundary. The fact that there is no boundary implies that there are only few dominated vertices, which dramatically reduces the capacity of VertexCollapser to collapse. At appropriate scale the flag complex of such point sets does not contain many dominated vertices. It is due to the fact that usually the vertices that are first dominated, lie towards a boundary. So, intuitively a strong collapse begins from a boundary of the manifold and retracts homotopically towards 'center'. To better grasp this fact, one can play with examples of well distributed points on a circle or a sphere (without boundary) and on a disk (with boundary). Remarkably, EdgeCollapser does not face this problem. The experimental finding confirms our intuition and EdgeCollapser performs much better than VertexCollapser in this case.

EdgeCollapser computes the exact PD of the input filtration while VertexCollapser has an exact and an approximate modes, Results in Table 5.2 are obtained using the exact mode of VertexCollapser, while results in Table 4.1 of Chapter 4 are obtained using the approximate mode. In both cases, EdgeCollapser performs much better than VertexCollapser. It would be easy to implement an approximate version of EdgeCollapser similarly to what has been done for VertexCollapser. Instead of triggering the backward loop of the algorithm [Line12-26] at each primary critical edge we find, we can trigger the backward loop at certain snapshot values only. See Section 4.3 of Chapter 4 for more details on the approximate methodology and description of snapshot.

Data	Pnt	Thrsld	EdgeCollapser +PD				
			Edge(I)/Edge(C)	Size/Dim	*dim*	Pre-Time	Tot-Time
netw-sc	379	5.5	8.4K/417	1K/6	∞	0.62	0.73
senate	103	0.415	2.7K/234	663/4	∞	0.21	0.24
eleg	297	0.3	9.8K/562	1.8K/6	∞	1.6	1.7
HIV	1088	1050	182K/6.9K	86.9M/?	6	491	2789
torus	2000	1.5	428K/14K	44K/3	∞	288	289

TABLE 5.1 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), Initial number of edges/Critical (final) number of edges *Edge(I)/Edge(C)*, number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by Edge-Collapser and total time (in seconds) including PD computation (Tot-Time). K and M in the size columns are thousands and millions respectively.

Data	Pnt	Thrsld	VertexCollapser +PD					
			Size/Dim	*dim*	Pre-Time	Tot-Time	*Step*	Snaps
netw-sc	379	5.5	175/3	∞	366.46	366.56	0	8420
senate	103	0.415	417/4	∞	15.96	15.98	0	2728
eleg	297	0.3	835K/16	∞	518.36	540.40	0	9850
HIV	1088	1050	127.3M/?	4	660	3,955	4	184
torus	2000	1.5		4	∞*	∞	0	428K

TABLE 5.2 – The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by VertexCollapser, total time (in seconds) including PD computation (Tot-Time), parameter *Step* (linear approximation factor) and the number of snapshots used (Snaps). *The last experiment (torus) could not finish (>12hrs) the preprocessing due to large number of snapshots and the size of the complex. K and M in the size columns are thousands and millions respectively.

Comparison with Ripser. As in both previous chapters (Chapter 3 and Chapter 4 we provide the same experimental results using Ripser [4]. Ripser computes the *exact* PD associated to the input filtration up to dimension *dim*. Again EdgeCollapser (as well as VertexCollapser) are not really competitors of Ripser since they act more as a preprocessing of the input filtration and do not compute Persistence Homology. Hence they can be associated to any software computing flag filtrations. Nevertheless, we again mention the results using Ripser[2] to demonstrate the benefit of using EdgeCollapser.

Results are presented in Table 5.3. The main observation is that, in most of the cases, EdgeCollapser computes PD in all dimensions and outperforms Ripser, even when we restrict the dimension of the input filtration given to Ripser.

Data	Pnt	Threshold	Val		Val		Val	
			dim	Time	*dim*	Time	*dim*	Time
netw-sc	379	5.5	4	25.3	5	231.2	6	∞
senate	103	0.415	3	0.52	4	5.9	5	52.3
"	"	"	6	406.8	7	∞		
eleg	297	0.3	3	8.9	4	217	5	∞
HIV	1088	1050	2	31.35	3	∞		
torus	2000	1.5	2	193	3	∞		

TABLE 5.3 – The first three columns are as in Table 5.2 and then, from left to right: parameter (*dim*), Time is the total time (in seconds) taken by Ripser. ∞ means that the experiment ran longer than 12 hours or crashed due to memory overload.

---

[2] We used the command <./ripser inputData –format distances –threshold inputTh –dim inputDim >.

5.5 Discussion

In this chapter, we introduced the notion of k-collapse which is an extension of the notion of strong collapse. We studied the special case of edge collapses in more detail and provided a novel approach to simplify a flag filtration to a smaller flag filtration using edge collapse. Through experiments we show that our approach outperforms all other approaches including the strong collapse approach described in previous chapters.

The algorithm (Algorithm 4) described in this chapter performs exact reduction of a flag filtration to a flag filtration. However, as we have mentioned it can be easily adapted to perform an approximate reduction. Such an adaptation should lead to even more efficient computation of PH.

Similarly as in the case of strong collapses in general it seems difficult to provide a theoretical bound on the amount of reduction done by edge collapse. And we would like to explore this question specially in the setting of random simplicial complexes as a future research topic.

This chapter is based on an article accepted in the proceedings of "36th International Symposium on Computational Geometry (SoCG), 2020".

Chapter 6

Conclusion

We briefly summarize the approaches developed in this thesis and discuss some possible future work.

We introduced two new approaches to improve the computation of the PH of a given sequence of simplicial complexes (either a filtration, a tower, or a zigzag sequence). Both approaches are closely related,

1. We used the notion of strong collapse introduced by J. Barmak and E. Minian [3] to simplify the complexes of the input sequence.

2. We extended the notion of domination of a vertex to any k-simplex and then used edge collapses to simplify flag filtrations.

Numerous experiments performed on publicly available data showed that both methods beat the state of the art methods to compute persistent homology. Our experiments show that our approach using edge collapses performs the best among all approaches. However, both methods can be used in combination as they are complementary. In case of general simplicial complexes our methods require a representation of the complexes consisting only of the maximal simplices. And in the case of flag complexes they can be performed over the 1-skeleton. The gain in time by applying our method is enormous especially in high dimensions.

Futute Work. Both strong collapse and edge collapse have shown promises to solve computational problems in Topology. Again their strength comes from their characterization in terms of maximal simplices or the neighborhoods of a graph (in case of flag complex). Below we mention a few problems which we think should be considered as future work.

–As mentioned in the discussion section of Chapter 3 and Chapter 5, we would like to have a theoretical bound on the reduction size of both strong collapse and edge collapse. Obtaining such bounds in general seems difficult. A possibly easier goal would be to estimate the size reduction in the case of random simplicial complexes [33]. There are many different models of random simplicial complexes, each model should provide a different insight about reduction capabilities of these collapses. Many variants of this problem have been studied in the case of simple collapse [33].

–As shown in Section 5.1 of Chapter 5, the 1-core corresponding to edge collapse is not unique. We conjecture that it is NP-complete to compute an optimal 1-core. And therefore the problem of computing an approximate 1-core by some constant factor of the optimal 1-core could be well defined and appropriate for providing theoretical guarantees about the reduction achieved by edge collapse.

Bibliography

[1] M. Adamaszek and J. Stacho. Complexity of simplicial homology and independence complexes of chordal graphs. *Computational Geometry: Theory and Applications*, 57:8–18, 2016.

[2] Dominique Attali, André Lieutier, and David Salinas. Vietoris-rips complexes also provide topologically correct reconstructions of sampled shapes. *Computational Geometry*, 46(4):448–465, 2013.

[3] J. A. Barmak and E. G. Minian. Strong homotopy types, nerves and collapses. *Discrete and Computational Geometry*, 47:301–328, 2012.

[4] U. Bauer. Ripser. URL: `https://github.com/Ripser/ripser`.

[5] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. PHAT – persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78, 2017.

[6] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. pages 103–117, 2014.

[7] J-D. Boissonnat and C. S. Karthik. An efficient representation for filtrations of simplicial complexes. In *ACM Transactions on Algorithms*, 2018.

[8] J-D. Boissonnat, C. S. Karthik, and S. Tavenas. Building efficient and compact data structures for simplicial complexes. *Algorithmica*, 79:530–567, 2017.

[9] J-D. Boissonnat and S. Pritam. Computing persistent homology of flag complexes via strong collapses. *International Symposium on Computational Geometry (SoCG)*, 2019.

[10] J-D. Boissonnat, S.Pritam, and D. Pareek. Strong Collapse for Persistence. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112, 2018.

[11] M. Botnan and G. Spreemann. Approximating persistent homology in euclidean space through collapses. *In: Applicable Algebra in Engineering, Communication and Computing*, 26:73–101.

[12] G. Carlsson and V. de Silva. Zigzag persistence. *Found Comput Math*, 10, 2010.

[13] G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. *International Symposium on Computational Geometry (SoCG)*, pages 247–256, 2009.

[14] F. Chazal and S. Oudot. Towards persistence-based reconstruction in Euclidean spaces. *International Symposium on Computational Geometry (SoCG)*, 2008.

[15] A. Choudhary, M. Kerber, and S. Raghvendra. In *Polynomial-Sized Topological Approximations Using The Permutahedron*. International Symposium on Computational Geometry (SoCG), 2016.

[16] A. Choudhary, M. Kerber, and S. Raghvendra. Improved topological approximations by digitization. Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, 2019.

[17] H. Edelsbrunner D. Cohen-Steiner and J. Harer. Stability of persistence diagrams. *Discrete and Compututaional Geometry*, 37:103–120, 2007.

[18] Datasets. URL: `https://github.com/n-otter/PH-roadmap/''`.

[19] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. pages 737–759, 2011.

[20] H. Derksen and J. Weyman. Quiver representations. *Notices of the American Mathematical Society*, 52(2):200–206, February 2005.

[21] T. K. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut Mathematique (Beograd)*, 60:23–45, 1999.

[22] T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *International Symposium on Computational Geometry (SoCG)*, pages 345–354, 2014.

[23] T. K. Dey, D. Shi, and Y. Wang. *SimBa: An efficient tool for approximating Rips-filtration persistence via Simplicial Batch-collapse*. In European Symp. on Algorithms (ESA), pages 35:1–35:16, 2016.

[24] T. K. Dey and R. Slechta. Filtration simplification for persistent homology via edge contraction. *International Conference on Discrete Geometry for Computer Imagery*, 2019.

[25] P. Dłotko and H. Wagner. Simplification of complexes for persistent homology computations,. *Homology, Homotopy and Applications*, 16:49–63, 2014.

[26] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[27] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Compututational Geometry*, 28:511–533, 2002.

[28] E. Fieux and J. Lacaze. Foldings in graphs and relations with simplicial complexes and posets. *Discrete Mathematics*, 312(17):2639 – 2651, 2012.

[29] F. Le Gall. Powers of tensors and fast matrix multiplication. *ISSAC '*, 14:296–303, 2014.

[30] Gudhi: Geometry understanding in higher dimensions. URL: `http://gudhi.gforge.inria.fr/`.

[31] A. Hatcher. *Algebraic Topology*. Univ. Press Cambridge, 2001.

[32] Benoît Hudson, Gary L. Miller, Steve Oudot, and Donald R. Sheehy. Topological inference via meshing. *International Symposium on Computational Geometry (SoCG)*, 2010.

[33] Matthew Kahle. Random simplicial complexes, 2016. `arXiv:1607.07069`.

[34] M. Kerber and H. Schreiber:. Barcodes of towers and a streaming algorithm for persistent homology. *International Symposium on Computational Geometry (SoCG)*, 2017. `arXiv:1701.02208`.

[35] M. Kerber and R. Sharathkumar. Approximate Čech complex in low and high dimensions. In *Algorithms and Computation*, pages 666–676. by Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam. Vol. 8283. Lecture Notes in Computer Science, 2013.

[36] Michael Lesnick and Matthew Wright. Computing minimal presentations and betti numbers of 2-parameter persistent homology. *CoRR*, 2019. URL: `http://arxiv.org/abs/1902.05708`.

[37] C. Maria and S. Oudot. Zigzag persistence via reflections and transpositions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA) pp*. 181–199, January 2015.

[38] Clément Maria and Hannah Schreiber. Algorithms and data structures - 16th international symposium, WADS, 2019,. pages 538–552, 2019.

[39] N. Milosavljevic, D. Morozov, and P. Skraba. Zigzag persistent homology in matrix multiplication time. In *International Symposium on Computational Geometry (SoCG)*, 2011.

[40] K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete and Computational Geometry*, 50:330–353, September 2013.

[41] D. Mozozov. Dionysus. URL: `http://www.mrzv.org/software/dionysus/`.

[42] J. Munkres. *Elements of Algebraic Topology*. Perseus Publishing, 1984.

[43] N. Otter, M. Porter, U. Tillmann, P. Grindrod, and H. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science, Springer Nature*, page 6:17, 2017.

[44] H. Schreiber. Sophia. URL: `https://bitbucket.org/schreiberh/sophia/`.

[45] D. Sheehy. Linear-size approximations to the Vietoris–Rips filtration. *Discrete and Computational Geometry*, 49:778–796, 2013.

[46] Volkmar Welker. Constructions preserving evasiveness and collapsibility. *Discrete Mathematics*, 207(1):243 – 255, 1999.

[47] J. H. C Whitehead. Simplicial spaces nuclei and m-groups. *Proc. London Math. Soc*, 45:243–327, 1939.

[48] A. C. Wilkerson, H. Chintakunta, and H. Krim. Computing persistent features in big data: A distributed dimension reduction approach. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 11–15, 2014.

[49] A. C. Wilkerson, T. J. Moore, A. Swami, and A. H. Krim. Simplifying the homology of networks via strong collapses. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 11–15, 2013.

[50] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33:249–274, 2005.

[51] A. Zomorodian and G. Carlsson. The theory of multidimensional persistence. *Discrete and Computational Geometry*, 42:71–93, 2009.