Algorithms and data structures in computational topology

Clément Maria

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour l'obtention du grade de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention Informatique

présentée et soutenue par

Clément MARIA

Algorithms and Data Structures in Computational Topology

Thèse dirigée par Jean-Daniel BOISSONNAT

soutenue le 28 octobre 2014

Jury

Rapporteurs:	Konstantin MISCHAIKOW	Professeur à Rutgers University
	Dmitriy MOROZOV	Chargé de recherche au Lawrence Berkeley National Lab
	Gilles VILLARD	Directeur de recherche CNRS/INS2I
Examinateurs:	Luc BOUGE	Professeur à l'ENS Cachan
	Tamal DEY	Professeur à Ohio State University
	Olivier FAUGERAS	Directeur de recherche INRIA
	André LIEUTIER	Ingénieur à Dassault Systèmes
	Steve OUDOT	Chargé de recherche INRIA
Directeur:	Jean-Daniel BOISSONNAT	Directeur de recherche INRIA

Résumé

La théorie de l'homologie généralise en dimensions supérieures la notion de connectivité dans les graphes. Étant donné un domaine, décrit par un complexe simplicial, elle définit une famille de groupes qui capturent le nombre de composantes connexes, le nombre de trous, le nombre de cavités et le nombre de motifs équivalents en dimensions supérieures. En pratique, l'homologie permet d'analyser des systèmes de données complexes, interprétés comme des nuages de points dans des espaces métriques. La théorie de l'homologie persistante introduit une notion robuste d'homologie pour l'inférence topologique. Son champ d'application est vaste, et comprend notamment la description d'espaces des configurations de systèmes dynamiques complexes, la classification de formes soumises à des déformations et l'apprentissage en imagerie médicale. Dans cette thèse, nous étudions les ramifications algorithmiques de l'homologie persistante. En premier lieu, nous introduisons l'arbre des simplexes, une structure de données efficace pour construire et manipuler des complexes simpliciaux de grandes dimensions. Nous présentons ensuite une implémentation rapide de l'algorithme de cohomologie persistante à l'aide d'une matrice d'annotations compressée. Nous raffinons également l'inférence de topologie en décrivant une notion de torsion en homologie persistante, et nous introduisons la méthode de reconstruction modulaire pour son calcul. Enfin, nous présentons un algorithme de calcul de l'homologie persistante zigzag, qui est une généralisation algébrique de la persistance. Pour cet algorithme, nous introduisons de nouveaux théorèmes de transformations locales en théorie des représentations de carquois, appelés principes du diamant. Ces algorithmes sont tous implémentés dans la librairie de calcul `Gudhi`.

Summary

The theory of homology generalizes the notion of connectivity in graphs to higher dimensions. It defines a family of groups on a domain, described discretely by a simplicial complex, that captures the connected components, the holes, the cavities and higher-dimensional equivalents. In practice, the generality and flexibility of homology allows the analysis of complex data, interpreted as point clouds in metric spaces. The theory of persistent homology introduces a robust notion of homology for topology inference. Its applications are various and range from the description of high dimensional configuration spaces of complex dynamical systems, classification of shapes under deformations and learning in medical imaging. In this thesis, we explore the algorithmic ramifications of persistent homology. We first introduce the simplex tree, an efficient data structure to construct and maintain high dimensional simplicial complexes. We then present a fast implementation of persistent cohomology via the compressed annotation matrix data structure. We also refine the computation of persistence by describing ideas of homological torsion in this framework, and introduce the modular reconstruction method for computation. Finally, we present an algorithm to compute zigzag persistent homology, an algebraic generalization of persistence. To do so, we introduce new local transformation theorems in quiver representation theory, called diamond principles. All algorithms are implemented in the computational library `Gudhi`.

Contents

Introduction

Computational Topology. Topology is the area of mathematics that formalizes and studies ideas of continuity. Continuity is a local property. It is defined by adding, to set of points, a qualitative notion of "proximity", relying on the definition of neighborhoods and limit. The notion of proximity allows the drawing of a global figure, called a *topological space*. Consider the three shapes of Figure 1. They geometrically appear different, but they

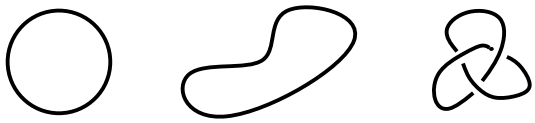


Figure 1: Three homeomorphic shapes.

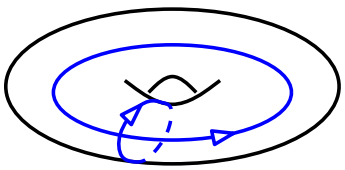
are topologically identical, in the sense that the proximity between points is the same: every small enough neighborhood around a point is a segment, and all points are connected by a path. This topological equivalence is measured by the concept of homeomorphism. A *homeomorphism* h between two topological spaces $\mathbb{X}$ and $\mathbb{Y}$ is a continuous bijective map $h : \mathbb{X} \to \mathbb{Y}$ whose inverse h^{-1} is continuous. Topology studies topological spaces up to homeomorphism, and in particular studies topological invariants of classes of homeomorphic topological spaces.

Algebraic topology is the subdomain of topology that assigns algebraic invariants to classes of homeomorphic topological spaces. The approach we are interested in is to define algebraic invariants from the incidence relations in a cellular discrete approximation of the topological space; specifically, a cell complex. We focus in this dissertion on simplicial complexes but results generalize naturally to arbitrary cell complexes. A *simplicial complex* is a combinatorial structure that consists of a collection of *simplices* – which are elementary building blocks like vertices, edges, triangles, tetrahedra and higher-dimensional equivalent – glued together along common faces. The

Figure 2: Graph whose underlying domain is homeomorphic to a shape.

graph in Figure 2 is a simplicial complex (containing only vertices and edges connected by their endpoints) that describes a domain topologically equivalent to a continuous shape. Consequently, through *adjacency and incidence relations* between simplices, simplicial complexes offer a discrete encoding of the notion of proximity in topological spaces.

Homology Theory. A natural topological invariant for spaces is path connectivity: are any two points in a topological space connected by a path? In the discrete setting, this question reduces to graph connectivity. *Homology theory* generalizes the notion of connectivity to higher dimensions. Homology attaches to a topological space a family of groups $\mathbf{H}_d$, one per dimension d, as topological invariants. Intuitively, $\mathbf{H}_0$ captures the number of connected components by measuring the void between them, $\mathbf{H}_1$ captures the number of holes by encircling them with non-contractible loops, $\mathbf{H}_2$ captures the number of cavities by wrapping them with non-contractible surfaces, etc. Consider the torus and its homology groups $\mathbf{H}_0$, $\mathbf{H}_1$ and $\mathbf{H}_2$ in Figure 3.

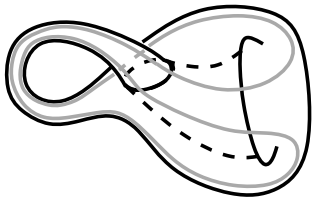


$$\begin{cases} \mathbf{H}_0 \cong \mathbb{Z} \\ \mathbf{H}_1 \cong \mathbb{Z} \oplus \mathbb{Z} \\ \mathbf{H}_2 \cong \mathbb{Z} \end{cases}$$

Figure 3: Homology groups with $\mathbb{Z}$ coefficients of the torus.

The number of $\mathbb{Z}$ summands in the primary decomposition of $\mathbf{H}_d$ corresponds to the number of homology features of dimension d: the torus has one connected component (dimension 0), two non-contractible independent loops (dimension 1, pictured in blue) and one cavity (dimension 2).

The information provided by homology groups is more precise and they also capture a notion of high-dimensional "twisting" of the shape. Consider the Klein bottle, which is a non-orientable surface whose 3-dimensional self-intersecting projection is pictured in Figure 4. The Klein bottle has one non-contractible loop, inducing the $\mathbb{Z}$ summand in the primary decomposi-



$$\begin{cases} \mathbf{H}_0 \cong \mathbb{Z} \\ \mathbf{H}_1 \cong \mathbb{Z} \oplus \mathbb{Z}_2 \\ \mathbf{H}_2 \cong 0 \end{cases}$$

Figure 4: Homology groups with $\mathbb{Z}$ coefficients of the Klein bottle.

tion of $\mathbf{H}_1$, and the "high-dimensional twisting" is represented by the cyclic summand $\mathbb{Z}_2$.

Topology in the Real World. The generality of topology has made it a great unifying theory of mathematics, where topological spaces show up naturally in almost all domains. As a consequence, lots of systems in applied sciences may be interpreted as topological spaces, and studied per se. A noticeable example is *topological inference*, where a system – a probability distribution, a dynamical system, etc – is known only by a discrete sampling $\mathcal{P}$ and a proximity relation between points, like a metric. The classic approach is to fix a scale ε at which an approximation of the underlying space is reconstructed, using a simplicial complex, from which the homology is computed; see Figure 5. There exist many simplicial complexes (like Čech

Figure 5: Homeomorphic approximation of a topological space from a point sample.

complex, Rips complex, and witness complex) defined, for a given scale parameter ε, on a point cloud. The scale ε represents, intuitively, the maximal size of a simplex.

Homology groups are meant to differenciate topological spaces that are not homeomorphic. They are consequently sensitive to discontinous transformations of a space. By nature, point clouds are discontinuous and the slightest perturbation of the points or variation of scale may affect the homology groups of the approximating complex; see Figure 6. In order to provide a robust topological invariant in such a setting, persistent homology has been introduced as a multi-scale theory of homology.

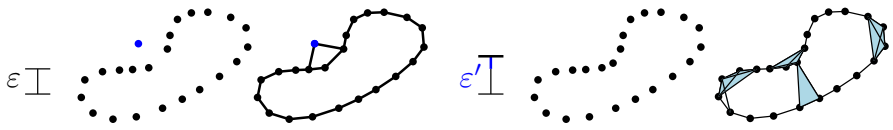


Figure 6: Approximations from a perturbed point cloud with the wrong homology.

The Theory of Persistent Homology. Consider now not only one topological space $\mathbb{X}$, but a sequence of topological spaces $\mathbb{X}_1, \cdots, \mathbb{X}_n$ connected by continuous maps $\mathbb{X}_i \to \mathbb{X}_{i+1}$ or $\mathbb{X}_i \leftarrow \mathbb{X}_{i+1}$. These continuous maps induce group homomorphisms at the homology level for any dimension d:

$$\begin{array}{ccccccccc} \mathbb{X}_1 & \longleftrightarrow & \mathbb{X}_2 & \longleftrightarrow & \cdots & \longleftrightarrow & \mathbb{X}_{n-1} & \longleftrightarrow & \mathbb{X}_n \\ \downarrow & & \downarrow & & & & \downarrow & & \downarrow \\ \mathbf{H}_d(\mathbb{X}_1) & \longleftrightarrow & \mathbf{H}_d(\mathbb{X}_2) & \longleftrightarrow & \cdots & \longleftrightarrow & \mathbf{H}_d(\mathbb{X}_{n-1}) & \longleftrightarrow & \mathbf{H}_d(\mathbb{X}_n) \end{array}$$

where an arrow $\longleftrightarrow$ is either backward or forward and each square:

$$\begin{array}{ccc} \mathbb{X}_i & \longrightarrow & \mathbb{X}_{i+1} \\ \downarrow & & \downarrow \\ \mathbf{H}_d(\mathbb{X}_i) & \longrightarrow & \mathbf{H}_d(\mathbb{X}_{i+1}) \end{array} \quad \text{or} \quad \begin{array}{ccc} \mathbb{X}_i & \longleftarrow & \mathbb{X}_{i+1} \\ \downarrow & & \downarrow \\ \mathbf{H}_d(\mathbb{X}_i) & \longleftarrow & \mathbf{H}_d(\mathbb{X}_{i+1}) \end{array}$$

commutes. The object of study of persistence is the induced sequence of homology groups. This is a module. When homology is considered with coefficients in a field $\Bbbk$ – a somehow weaker version of homology – *Gabriel's theorem* from quiver theory tells us that the sequence admits a direct sum decomposition into interval modules, which may be interpreted as persistent topological features. Specifically, define $\mathbb{I}[b;d]$ to be the *interval module*:

$$\underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[1;b-1]} \overset{0}{\longleftrightarrow} \underbrace{\Bbbk \overset{\mathbb{1}}{\longleftrightarrow} \cdots \overset{\mathbb{1}}{\longleftrightarrow} \Bbbk}_{[b;d]} \overset{0}{\longleftrightarrow} \underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[d+1;n]}$$

that we interpret as a homology feature which is born at index b and dies after index d. The sequence of homology groups admits a direct sum decomposition of the form $\bigoplus_i \mathbb{I}[b_i; d_i]$. Computing the persistent homology of the sequence of topological spaces $\mathbb{X}_1, \cdots, \mathbb{X}_n$ consists in computing this interval decomposition.

Considering homology with field coefficients is necessary for the interval decomposition to exist. However, integral homology is strictly more informative than the homology groups with coefficients in a field $\Bbbk$, denoted by

$\mathbf{H}_d(\mathbb{X}, \Bbbk)$. Indeed, $\mathbf{H}_d(\mathbb{X}, \Bbbk)$ has the structure of a $\Bbbk$-vector space and has a decomposition $\mathbf{H}_d(\mathbb{X}, \Bbbk) \cong \Bbbk^\beta$, with β the dimension of the vector space. Consequently, there is no notion of torsion. Specifically, the torsion subgroups in the decomposition of $\mathbf{H}_d(\mathbb{X}, \mathbb{Z})$ either vanish or appear as infinite in the decomposition of $\mathbf{H}_d(\mathbb{X}, \Bbbk)$. As an example, consider the homology of the torus and the Klein bottle with $\mathbb{Z}_2$ coefficients in Figure 7. Their

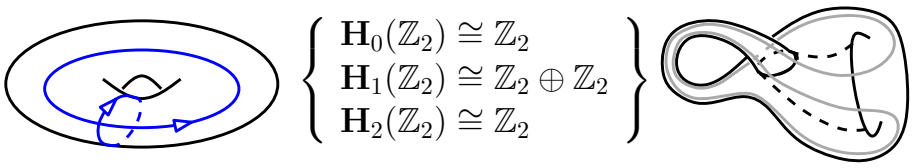


Figure 7: Homology with $\mathbb{Z}_2$ coefficients of the torus and the Klein bottle.

homology groups are identical whereas the spaces are not homeomorphic. For the torus, every $\mathbb{Z}$ summand of its integral homology turns into a $\mathbb{Z}_2$ summand in homology with $\mathbb{Z}_2$ coefficients, and one can read the same information in term of homology features. For the Klein bottle though, the $\mathbb{Z}_2$ summand capturing the torsion in integral homology turns into a $\mathbb{Z}_2$ summand in $\mathbf{H}_1(\mathbb{Z}_2)$. Note also that the 2-homology is affected. Finally, if $\Bbbk$ is a field of characteristic different from 2, the homology with $\Bbbk$ coefficients of the torus and the Klein bottle become different, and one can distinguish between the two spaces from their homology. The *universal coefficient theorem* of homology relates the integral homology groups of a topological space with its homology groups with $\Bbbk$ coefficients, depending on the characteristic of the field $\Bbbk$.

We now apply the idea of persistence to a noisy sample of one of our earlier topological spaces. We reconstruct an approximation of the topological space at increasing scales ε, which induces a sequence of growing simplicial complexes $\mathbf{K}_{\varepsilon_1} \hookrightarrow \cdots \hookrightarrow \mathbf{K}_{\varepsilon_n}$, $\varepsilon_1 \leq \cdots \leq \varepsilon_n$, where arrows are all left-to-right and the continuous maps between spaces are inclusions; see Figure 8. This case is known as *persistent homology* and is the standard approach in topological inference. We compute the direct sum decomposition into interval modules of the induced sequence of homology groups and represent each interval $\mathbb{I}[b; d]$ of the decomposition by a segment $[b; d]$, hence giving a *persistence barcode*. The long bars in the barcode give the meaningful topological features of the underlying space, while the short bars are topological noise. Additionally, unlike homology, persistent homology admits good stability properties, which make it a robust mathematical invariant for the study of topological spaces in practice.

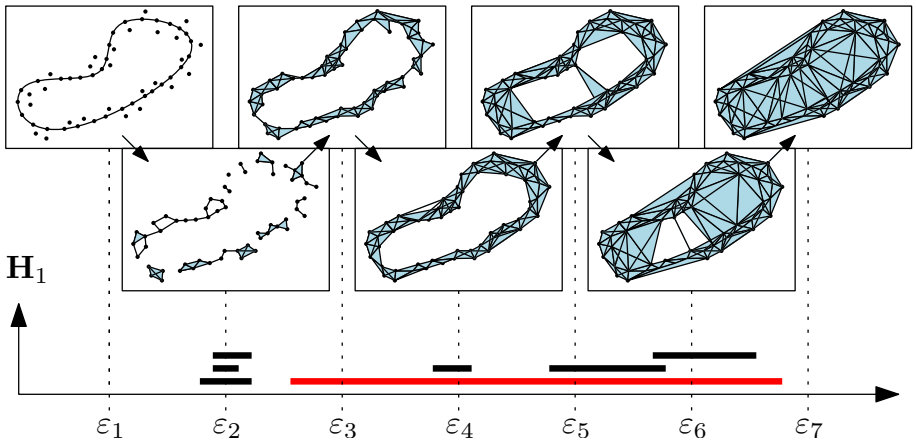


Figure 8: Persistent homology for topological inference.

The study of a persistence module with arrows in both directions is known as *zigzag persistent homology*. It is an important improvement over standard persistent homology. Indeed, persistent homology is defined for a nested family of topological spaces, which essentially reduces its area of study to the multi-scale example presented above. However, other parameters are of interest for the study of data. An important one is the number of points used as vertices for constructing the simplicial complexes. For example, one may want to subsample many times a point cloud, in order to reduce the size of the simplicial complexes constructed or based on density considerations. The simplicial complexes obtained for the different subsamples are not connected by inclusions, and we cannot study their persistent homology. However, zigzag persistence allows one to connect these simplicial complexes in a global zigzag module. Consider the example of Figure 9, where we adapt the scale ε to the number of points we sample (the more points, the smaller is ε). In this example, up-right arrows consist in adding more points, and up-left ones consist in increasing the scale; they are both inclusions but in different directions. With this approach, we are able to infer the homology of our previous example while maintaining much smaller simplicial complexes.

The Algorithmic Problem. Going back to the simple problem of path connectivity, an algebraic algorithm to determine whether a graph $\mathcal{G}(V, E)$ is connected or not could consist in representing the graph by its $|V| \times |E|$-*incidence matrix* and computing its rank r. The incidence matrix of a graph represents the incidence relations between vertices and oriented edges. Given

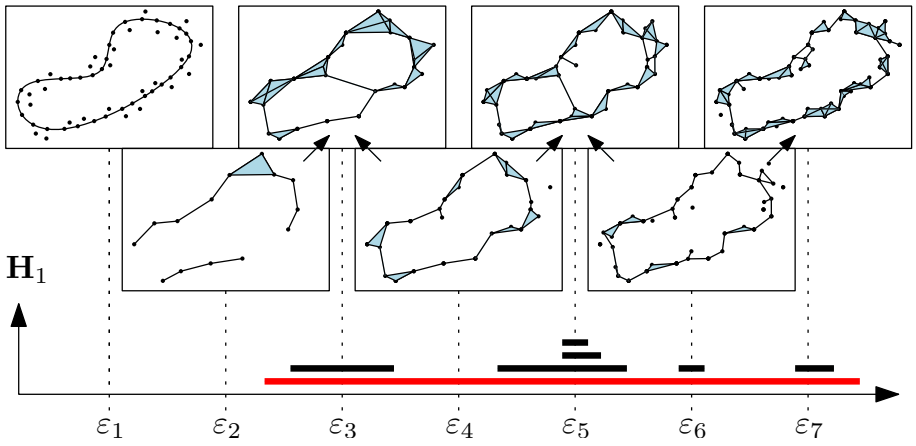


Figure 9: Zigzag persistent homology for topological inference.

the graph represented by its incidence matrix M in Figure 10, a diagonalisation of M gives us its rank. Algebraic graph theory teaches us that the graph is connected *iff* $|V| - r = 1$. However, this approach is not natural for two reasons. The first one is that the encoding of the graph with an incidence matrix is not efficient. In the context of topology, a topological space has usually a sparse connectivity; an adjacency list encoding is consequently more appropriate. The second one is that the connectivity of a graph may be computed with a simple graph traversal, or with a union-find data structure depending on the graph representation; see Figure 11.

The connectivity example for the graph seems a bit far-fetched, because historically graphs have been studied as combinatorial structures before being studied algebraically. However, despite being a higher-dimensional generalization of connectivity in graphs, the homology of simplicial complexes is a mathematical theory that relies on non-trivial algebra, and studies an intricate combinatorial structure. Consequently, existing algorithms in computational topology tend to represent all incidence relations explicitly and

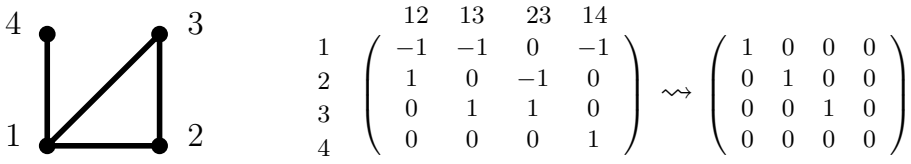


Figure 10: Graph represented by its incidence matrix. Connectivity is computed by diagonalising the incidence matrix.

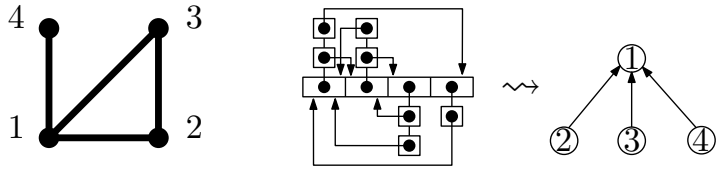


Figure 11: Graph represented by an adjacency list data structure. Connectivity is computed with union-find.

rely on black-box matrix algorithms, without taking advantage of the intrinsically combinatorial structure of the problem.

In light of former mathematical and computer algebra works in the domain, the main thread of this dissertation is to revisit computational topology towards the design of efficient algorithms, from point clouds to persistence barcodes.

State of the Art. Homology groups, with arbitrary coefficients, of a simplicial complex are known to be computable with a reduction to Smith normal form of matrices representing all the incidence relations between simplices in all dimensions [59], leading to a $O(n^3)$ algorithm for a complex with n simplices and for coefficients in a finite field $\mathbb{k}$. Efficient methods exist for the computation [41].

The first attempt to exploit the combinatorial structure of the problem was to generalize the disjoint-set implementation of graph connectivity to the computation of homology in a field $\mathbb{k}$ of complexes embedded on the 3-sphere [37]. The algorithm is simple and has a quasi-linear $O(n\alpha(n))$ time complexity – using a union-find data structure [32] and where $\alpha(\cdot)$ is the slowly growing inverse Ackermann function – which is a substancial improvement compared to the Smith normal form approach. The algorithm is dynamic and, up to a $O(n \log n)$ time for reordering the simplices, adapts to the computation of persistent homology.

The concept of persistent homology has been introduced in [43] for 3-dimensional complexes and for homology with $\mathbb{Z}_2$ coefficients, together with an efficient sparse matrix implementation. Both theory and algorithm have been generalized to arbitrary dimensions and coefficient fields in [70]. The later article analyzes the sequence of homology groups with $\mathbb{k}$ coefficients as a $\mathbb{k}[t]$-module, and shows that persistent homology may be computed by the reduction to column echelon form of a single incidence matrix.

A persistent cohomology algorithm [35] has been introduced as a new

approach – using cohomology, the dual of homology – to compute persistence. By duality, it computes the same decomposition of the persistence module, but shows better performance in practice. An interesting presentation of the later algorithm was introduced in [40], where a cohomological information is attached to each simplex of the complex as an *annotation*.

Both persistent homology and cohomology algorithms are simple algorithms reducing to the computation of the echelon form of the incidence matrix of a complex, and run in $O(n^3)$ time with $O(n^2)$ memory.

The major property of persistent homology is its stability with regard to perturbation of the input sequence of topological spaces. This has been proved in [30]. This result validates persistent homology as the right tool to study the topology of a space in practice.

Persistent homology has been generalized in [24] to the theory of zigzag persistence. The authors use quiver theory [38, 60] to give a general framework for the decomposition of persistence modules, and introduce a general algebraic algorithm for computing their decomposition. The algorithm is applied to zigzag persistent homology in [25], and admits a cubic complexity analysis. Compared to persistent homology, the description of the zigzag persistence algorithm requires extra machinery, which makes it more challenging to grasp the higher-level picture of how and why it works. The algorithm performs reasonably well in practice but is nowhere as efficient as the optimized algorithms for standard persistence described below.

The complexity of persistent homology and zigzag persistent homology algorithms has been improved in various directions. The authors of [29] present an output-sensitive algorithm in the number of long-life homology features. The complexity of zigzag persistence homology, and consequently persistent homology as well, has been improved to matrix multiplication time in [54]. These algorithms are based on advanced computer matrix algebra, and remain essentially theoretical.

Finally, computational topology knows an exciting expansion in various applied domains, requiring more efficient computational methods. This is in particular visible through the publication of [42], introducing the foundations of computational topology and making the domain accessible to a large audience.

During my Ph.D. studies, other directions have been followed in order to improve the algorithms in computational topology. One direction has been to improve the matrix algorithm for persistent homology. In [10] the authors show that most of the computation can be avoided due to the structure of the reduced matrix, leading to a significant improvement in practice. They also show that the algorithm may be parallelized, and, in a later publication,

distributed [11]. Another direction has been to simplify combinatorially the input complex using Morse simplifications [55] in order to compute persistence on smaller complexes.

Finally, various computational libraries exist in the domain; we mention three of them. The historically first library is `JPlex`, which furnishes construction of Rips and witness complexes, as well as the computation of the classic persistent homology algorithm. `Dionysus` [56] is a widely used library that implements the construction of Rips complexes, the matrix reduction algorithms for persistent homology and persistent cohomology [35, 70] with $\mathbb{Z}_p$ coefficients, as well as the $O(n^3)$ algorithm for zigzag persistence described in [25]. Fianlly, `PHAT` [9] is a library that implements the optimized computation of persistent homology of [10] and its parallelized version with $\mathbb{Z}_2$ coefficients.

Contribution. This dissertation presents a complete and coherent set of effective algorithms and data structures to solve the algorithmic problems presented above, taking advantage of the intrinsically combinatorial nature of computational homology. We present efficient algorithms and data structures to construct and represent simplicial complexes and compute their persistent homology. We present also an algorithm to compute persistent homology with various coefficients and infer the torsion subgroups of the integral homology of the approximated topological space. We then introduce an algorithm for computing zigzag persistence homology. Finally, we present the design of a computational library implementing these algorithms and data structures.

In Part I, we introduce the *simplex tree* data structure to represent general simplicial complexes. The simplex tree may be interpreted as a spanning tree of the *Hasse diagram* of the complex – representing all incidence relations between simplices – that respects a notion of lexicographic order between the ordered lists of vertices of the simplices. The data structure requires $O(1)$ memory word per simplex – which is optimal in persistence where an order of the simplices must be represented – and allows the efficient construction of simplicial complexes from point sets (Rips complexes, witness complexes), and their manipulation (retrieval of incidence relations, insertion, deletion, edge contraction, etc). The work has been published in [BM12, BM14b] and received the conference best paper award at the *European Symposium on Algorithms* 2012.

In Part II, we present the *compressed annotation matrix*, a data structure to efficiently compute persistent cohomology via the annotation algorithm

of [40]. Taking advantage of the structure of cohomology, we introduce a framework separating the represention of the simplicial complex from the representation of the cohomology groups, in order to design an efficient sequential algorithm. The algorithm may be seen as a generalization of the union-find algorithm for low dimensional persistent homology on the 3-sphere [37], where a cohomology annotation is attached to each simplex. In particular, most of the algebraic computation is replaced with combinatorial union-find operations. The work has been published in [BDM13] and a journal version has been invited to a special issue of *Algorithmica* [BDM15]. The algorithm is the most stable and one of the most efficient implementations currently available.

In Part III, we present an algorithm for computing persistent homology while infering efficiently torsion coefficients of the homology groups. As explained earlier, homology with field coefficients, as opposed to homology with integral coefficients, provides a less informative description of the topology of a space. In particular, the description of the homology groups with coefficients in a field is perturbed by the existence of torsion subgroups in integral homology. As persistent homology must be computed with field coefficients for an interval decomposition to exist, we overcome this difficulty by formulating an arithmetic framework for computing persistent homology with coefficients in a family of fields with distinct characteristics simultaneously. From this computation, one can reconstruct the integral homology groups using the *universal coefficient theorem*. The algorithm admits an output-sensitive complexity analysis, in terms of the differences in the interval decompositions of the persistence module with different coefficients. The work has been published in [BM14a]. The algorithm is the only non-trivial solution proposed to the problem of computing persistence with different coefficient fields, and shows only a small overhead in practice compared to persistence in a single field, while furnishing more topological information.

In Part IV, we introduce a simple algorithm for computing zigzag persistence, designed in the same spirit as the standard persistence algorithm. Our algorithm reduces a single matrix, maintains an explicit set of chains encoding the persistent homology of the current zigzag, and updates it under simplex insertions and removals. The total worst-case running time matches the usual cubic bound. A noticeable difference with the standard persistence algorithm is that we do not insert or remove new simplices at the end of the zigzag, but rather in the middle. To do so, we use arrow reflections and transpositions, in the same spirit as reflection functors in quiver theory. Our analysis introduces a new kind of reflection called the *weak diamond*, for which we are able to predict the changes in the interval decomposition

and associated compatible bases. Arrow transpositions have been studied previously in the context of standard persistent homology [31], and we extend the study to the context of zigzag persistence. The algorithm compares favorably to the existing one in practice. Moreover, being close in nature to the standard persistence algorithm, it is potentially amenable to the same optimizations as the ones described in Parts I, II and III of this dissertation. This opens exciting research questions towards very efficient zigzag persistence implementations. The work has been submitted [MO15].

The theoretical techniques introduced in this dissertation are of practical significance, and open new computational horizons for applications. We present in Part V the design of the `Gudhi` library (`https://project.inria.fr/gudhi/software/`), a generic `C++` computational library implementing the algorithms and data structures mentioned above. The work has been published in [MBGY14]. The goal of the `Gudhi` library is to make available sophisticated tools for complex high-dimensional geometry and topology computation. The simplex tree and persistent cohomology packages are available at [Mar].

The mathematical concepts involved in the different parts of the dissertation are diverse. Consequently, Part I to IV begin with introductory chapters. These chapters give the reader the material to understand formally the contributions presented in this work, and the context in which they have been developed. When the concepts mentioned in the introductory chapters are not new, their presentation, focused towards the algorithmic nature of the theory and its various algorithmic branches, is original.

Publications

[BDM13] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *ESA*, pages 695–706, 2013.

[BDM15] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. *Algorithmica*, 2015.

[BM12] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *ESA*, pages 731–742, 2012.

[BM14a] Jean-Daniel Boissonnat and Clément Maria. Computing persistent homology with various coefficient fields in a single pass. In *ESA*, 2014.

[BM14b] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427, 2014.

[Mar] Clément Maria. Gudhi, simplicial complexes and persistent homology packages. `https://project.inria.fr/gudhi/software/`.

[MBGY14] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The Gudhi library: Simplicial complexes and persistent homology. In *International Congress on Mathematical Software*, pages 167–174, 2014.

[MO15] Clément Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. In *SODA*, 2015.

How to Read this Dissertation

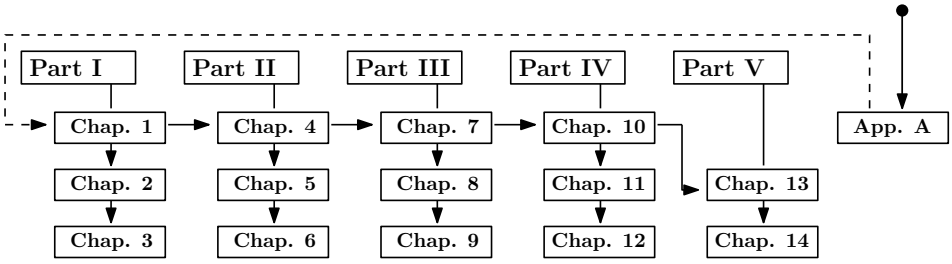


Figure 12: Chapter dependency.

Each part of this dissertation presents an algorithmic problem and a solution. They are all independent. However, as illustrated in Chapter 14, solving computationally a standard topological problem requires the use of all methods from Part I to Part IV chronologically.

Parts I to IV are subdivided into three chapters. The first chapter contains the necessary background to understand formally the problem treated, the underlying motivation and the state of the art. The second chapter introduces an algorithmic solution and the third chapter presents practical performance, except for Part IV which is of a more theoretical nature and contains proofs of validity of the algorithm.

Any path starting from the root in Figure 12 is self-contained. In particular, walking entirely the horizontal path of the dependency tree gives a complete introduction to computational topology and the topology inference problem, with a view towards algorithms. Specifically, the motivation under this introduction to computational topology is to start from the general mathematical concepts of topology to describe formally how to define topological invariants that may be computed in a discrete setting and that are of interest in practice.

The concepts of topology are not presented in their most general form, but rather on a suitable level for computation. Theorems are proved, apart from the ones requiring background out of scope of this dissertation. For

these later results, sketches of proof are provided for giving an intuition to the reader.

Applications and Experimental Protocol

The work presented in this dissertation is of theoretical nature; it consists in the design of algorithms and data structures to solve complex combinatorial and algebraic problems. However, these methods are of practical significance. An emphasis is put on the efficiency of these methods on real-world problems, as expressed by the detailed experiments provided in each part of this dissertation and culminating in the design and demonstration of use of a computational library in Part V.

The algorithmical study of high-dimensional (more than 2 or 3) shapes by a topological approach is a recent research topic. Persistent homology has become a mathematical domain per se in the years 2000. Zigzag persistent homology has been introduced even more recently and its theoretical foundation are still not fully understood (see the Concluding Remarks of this dissertation). As a consequence, despite the great success of persistence in the applied sciences, it is still quite early to establish a list of applied domains in which persistent homology is the right tool.

As already suggested in the Introduction, we have decided to focus in this work on the problems of topological reconstruction and topological inference. They are, at the same time, the more mature theoretically, the more general model of reality and they offer the possibility for an interesting range experiments. The problem is the following; we assume we are given a set of points, with a notion of distance between them, sampling an unknown topological space. The problem is to compute topological invariants of this space, by reconstructing an approximation of it and computing the topology of this approximation. Figures 5, 8 and 9 of the Introduction picture examples of this problem.

The topology inference from a discrete sample is a problem that is theoretically mature because all computations may be done with theoretical garantees. This will be detailled in the first Chapter of every Part of this dissertation. It is a good model of reality because of the generality of the concept of topological space, which is a simple set with a general notion of proximity between points (see Chapter 1. Hence, many systems may be

Data	manifold	embedded	D	d
Bud	✓	✓	3	2
Nep	✓	✓	3	2
Cy8	✗	✓	24	2
Kl	✓	✓	5	2
L35	✓	✗	–	3
L57	✓	✗	–	3
S3	✓	✓	4	3
S4	✓	✓	5	4
Bro	✗	✓	25	unknown

Figure 13: Data sets used in the experiments.

modeled as a topological space, and study as such. We now enumerate the topology inference experiments we use to validate the practical performance of our algorithmic methods.

Experimental Protocol. All along this work, we use a variety of both real and synthetic data sets, summarized in Figure 13.

- **Bud** and **Nep** are sets of points sampled from the surfaces in $\mathbb{R}^3$ of, respectively, the *Happy Buddha* from the *Stanford 3D Scanning Repository* [62] and the *Neptune statue* from the *Aim@Shape Project* [3].

- **Cy8** is a set of points in $\mathbb{R}^{24}$, sampled from the space of conformations of the cyclo-octane molecule, which is the union of two intersecting surfaces [53].

- **Kl** is a set of points sampled from the surface of the figure eight Klein Bottle embedded in $\mathbb{R}^5$, generated by embedding a regular grid of points $(u, v) \in [0; 2\pi) \times [0; 2\pi)$ into $\mathbb{R}^5$ using the formula:

$$\begin{cases} x_1 &= [1/2 + \cos(u/2) \times \sin(v) - \sin(u/2) \times \sin(2v)] \times \cos(u) \\ x_2 &= [1/2 + \cos(u/2) \times \sin(v) - \sin(u/2) \times \sin(2v)] \times \sin(u) \\ x_3 &= \sin(u/2) \times \sin(v) + \cos(u/2) \times \sin(2v) \\ x_4 &= \sin(u) \\ x_5 &= \cos(v) \end{cases}$$

- **L35** and **L57** are sets of points sampling the *lens spaces* $L(3, 5)$ and $L(5, 7)$ respectively. A lens space $L(p, q)$ is a non-embedded manifold homeomorphic to a 3-sphere quotiented by the equivalence relation

defined by an orbit on the sphere. The orbit depends on p and q and all points of an orbit are equivalent. The point clouds are generated by sampling randomly a 3-sphere, generating the orbit of every point, selecting a representative for each orbit and computing its geodesic distance to every other orbit.

- **S3** and **S4** are sets of points sampled from, respectively, the unit 3-sphere in $\mathbb{R}^4$ and the unit 4-sphere in $\mathbb{R}^5$.

- **Bro** is a set of 5×5 *high-contrast patches* derived from natural images, interpreted as vectors in $\mathbb{R}^{25}$, from the Brown database (with parameter $k = 300$ and cut 30%) [26, 48]. It is a data set of statistical nature, and reveals different shapes depending on subsampling based on density.

Based on these points, we construct a variety of simplicial complexes used in computational topology. The most popular is the *Rips complex* (defined in Chapter 1) which gives a provably correct topological reconstruction (see Chapter 4). The second construction is the *(relaxed) witness complex* (defined in Chapter 1), which is a Delaunay-like simplicial complex based only on distance computation. Both complexes are well-suited for high-dimensional geometry. Finally, we experiment on α-shapes which are filtrations of Delaunay triangulations. While they furnish good topological reconstruction, they are well-suited for low dimensional spaces, and implementations restrict to $\mathbb{R}^3$.

Our implementations are in `C++` and are integrated in the `Gudhi` library [50], developped as part of this work. All timings are measured on a Linux machine with 3.00 GHz processor and 32 GB RAM. All timings are averaged over 10 independent runs. Timings are provided by the `clock` function from the `Standard C Library`, and zero means that the measured time is below the resolution of the `clock` function.

We compare our implementations with state of the art softwares in the domain. Depending on the algorithms available, the softwares used for comparisons are `JPlex` [65], `Dionysus` [56] and/or `PHAT` [9]. `JPlex` is a `Java` library which can be used with `Matlab` and provides an implementation of the construction of Rips complexes and witness complexes. `Dionysus` is a `C++` library which provides implementations of the construction of Rips complexes and oscillating Rips zigzag, the computation of persistent homology and cohomology with $\mathbb{Z}_p$ coefficients (for an arbitrary prime p), and the computation of zigzag persistence with $\mathbb{Z}_2$ coefficients. `PHAT` is a `C++` library

that furnishes implementations of optimized matrix reduction algorithms for persistent homology and persistent cohomology with $\mathbb{Z}_2$ coefficients.

Standard Notations

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, respectively the relative integers, the rationals and the real numbers
$\mathbb{Z}_n$, the integer modulo n
$\mathbb{R}^D$, the D-dimensional Euclidean space
$\#A$, the size of a finite set A
$\cong$, isomorphism relation
$\mathbb{1}_A$, the identity function on a set A
iff, if and only if

Part I

Simplex Tree for Simplicial Complexes

This Part is based on the following publications:

- Jean-Daniel Boissonnat and Clément Maria. The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. *European Symposium on Algorithms 2012, Best Paper Award* [18]
- Jean-Daniel Boissonnat and Clément Maria. The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. *Algorithmica, 2014* [20]

Chapter 1

Topology and Simplicial Complexes

In this chapter we introduce basic notions of *topology*. The central concept of topology is the *topological space*. A topological space is essentially a set of points together with a qualitative notion of proximity, expressed through the idea of *neighborhood*. Topology classifies topological spaces up to *homeomorphisms*. Being homeomorphic is an equivalence relation between topological spaces, which preserves local relations between points in the space. The classic approach to study equivalent topological spaces is to study *topological invariants*, which are properties of topological spaces that remain unchanged under homeomorphisms.

In computational topology, one studies *simplicial complexes*. They are finite representation which are both *geometric* domains admitting a topology and *abstract* combinatorial structures. Simplicial complexes are related by *simplicial maps*, which are the equivalent of continuous maps in the discrete setting. Finally, we introduce the *Čech complex*, the *Rips complex* and the *witness complex*, that are simplicial complexes useful in topology to solve the *topological inference* problem.

1.1 General Topology

Topological Space. Let $\mathbb{X}$ be a point set. A *topology* on $\mathbb{X}$ is a collection $\mathcal{O}$ of subsets of $\mathbb{X}$, called the *open sets*, such that:

(i) $\mathbb{X}$ and the empty set $\emptyset$ are open.

(ii) every union of open sets is open,

(iii) every intersection of finitely many open sets is open,

A *topological space* is a pair $(\mathbb{X}, \mathcal{O})$ consisting of a set $\mathbb{X}$, called the *space*, and a topology $\mathcal{O}$ on $\mathbb{X}$. The elements of $\mathbb{X}$ are called *points*. If for some $x \in \mathbb{X}$ and an open set $U \subseteq \mathbb{X}$ we have $x \in U$, we say that U is a *neighborhood* of x. A set $S \subseteq \mathbb{X}$ is open *iff* for every $x \in S$ there exists a neighborhood U_x of x contained in S. Indeed, if S is an open set then we can take $U_x = S$ for every $x \in S$, and if the condition is satisfied, then $S = \cup_{x \in S} U_x$ is open by virtue of axiom (i).

Let $(\mathbb{X}, \mathcal{O})$ be a topological space. A set $F \subseteq \mathbb{X}$ is *closed iff* its complement $\mathbb{X} \setminus F$ is open. The set of closed sets $\mathcal{C}$ have properties dual to the ones of open sets:

(i') the empty set $\emptyset$ and $\mathbb{X}$ are closed.

(ii') every intersection of closed sets is closed;

(iii') every union of finitely many closed sets is closed;

The *closure* $\overline{S}$ of a set $S \subseteq \mathbb{X}$ is defined to be the intersection of all closed sets $F \subseteq \mathbb{X}$ containing S. By virtue of property (i') of $\mathcal{C}$, this is a closed set. It follows from the definition that the closure of a set is the smallest closed set with regards to inclusion that contains it. Moreover, the family of closed sets is exactly the family of subsets of $\mathbb{X}$ that are equal to their closure.

Let $\mathbb{Y}$ be a subset of a topological space $\mathbb{X}$ with topology $\mathcal{O}$. The *subspace topology* $\mathcal{O}_{\mathbb{Y}}$ of $\mathbb{Y}$ is the topology induced by the one of $\mathbb{X}$ in the following way

$$\mathcal{O}_{\mathbb{Y}} = \{U \cap \mathbb{Y} : U \in \mathcal{O}\}$$

Metric Space. Let $\mathbb{X}$ be a set. A *metric* on the set $\mathbb{X}$ is a function $\mathsf{d} : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ satisfying, for every $x, y, z \in \mathbb{X}$,

(i) $\mathsf{d}(x, y) = 0$ *iff* $x = y$;

(ii) $\mathsf{d}(x, y) = \mathsf{d}(y, x)$;

(iii) $\mathsf{d}(x, z) \leq \mathsf{d}(x, y) + \mathsf{d}(y, z)$.

In particular, these axioms implies that $\mathsf{d}(x, y) \geq 0$.

A *metric space* is a pair $(\mathbb{X}, \mathsf{d})$ consisting of a set $\mathbb{X}$, called the *space*, and a metric d on $\mathbb{X}$. The elements of $\mathbb{X}$ are called *points*, and the number $\mathsf{d}(x, y)$ is called the *distance between x and y*. An *open ball* $\mathcal{B}(x, r)$, centered at

$x \in \mathbb{X}$ and of radius $r > 0$, is defined to the be set of points of $\mathbb{X}$ at distance at most r from x, *ie*

$$\mathcal{B}(x, r) = \{y \in \mathbb{X} : \mathsf{d}(x, y) < r\}$$

One can define a topology on a metric space by considering as open set any union of open balls.

Example 1.1. *The set of all unions of open balls of the Euclidean space $\mathbb{R}^D$ is a topology.*

In a metric space $(\mathbb{X}, \mathsf{d})$, we can define the distance between two subsets $U, V \subseteq \mathbb{X}$, called the *Hausdorff distance*, to be:

$$\mathsf{d}_{\mathcal{H}}(U, V) = \max\{\sup_{u \in U} \inf_{v \in V} \mathsf{d}(u, v), \sup_{v \in V} \inf_{u \in U} \mathsf{d}(u, v)\}$$

Equivalence between Topological Spaces. Let $(\mathbb{X}, \mathcal{O})$ and $(\mathbb{Y}, \mathcal{O}')$ be two topological spaces; a mapping f from $\mathbb{X}$ to $\mathbb{Y}$ is called *continuous* if $f^{-1}(U) \in \mathcal{O}$ for any $U \in \mathcal{O}'$, *ie* if the inverse image of any open subset of $\mathbb{Y}$ is open in $\mathbb{X}$. A continuous mapping $f \colon \mathbb{X} \to \mathbb{Y}$ is called a *homeomorphism* if f maps $\mathbb{X}$ onto $\mathbb{Y}$ in a one-to-one way and the inverse mapping f^{-1} of $\mathbb{Y}$ to $\mathbb{X}$ is continuous. We say that two topological spaces are *homeomorphic* if there exists a homeomorphism between them. Being homeomorphic is an equivalence relation.

A *topological invariant* is a property of topological spaces that is preserved under homeomorphisms, *ie* if $\mathbb{X}$ has a property $\mathcal{P}$ that is a topological invariant, then so does all spaces homeomorphic to $\mathbb{X}$.

Example 1.2. *A topological space $(\mathbb{X}, \mathcal{O})$ is* connected *iff it cannot be represented as the union of two disjoint open sets of $\mathcal{O}$. Being connected is a topological invariant because homeomorphisms preserve open sets and hence the property. Note that in general this is weaker than the more intuitive notion of path-connectedness mentioned in the introduction. However, in our setting (topological spaces described by finite simplicial complexes), the two notions are equivalent.*

A weaker notion of equivalence between topological spaces is a interest. Let $\mathbb{X}$ and $\mathbb{Y}$ be two topological spaces, and $f, g : \mathbb{X} \to \mathbb{Y}$ two continuous maps. A *homotopy* between f and g is a continuous map $H : \mathbb{X} \times [0; 1] \to \mathbb{Y}$ such that

(i) $H(\cdot, 0) : \mathbb{X} \to \mathbb{Y}$ and f are equal, *ie* $H(x, 0) = f(x)$ for every $x \in \mathbb{X}$,

Figure 1.1: Three homotopy equivalent shapes. The curve in the middle may be embedded homeomorphically in both the Möbius strip (left) and the annulus (right). The embeddings are drawn in red. There are natural deformation retracts from the Möbius strip and the annulus to the embedded curve.

(ii) $H(\cdot, 1) : \mathbb{X} \to \mathbb{Y}$ and g are equal, *ie* $H(x, 1) = g(x)$ for every $x \in \mathbb{X}$.

This is an equivalence relation for continuous maps, we write $f \simeq g$ and call f and g *homotopic*. We can think of a homotopy as a continuous deformation of a continuous map into the other.

The notion can be used to relate spaces. Specifically, $\mathbb{X}$ and $\mathbb{Y}$ are *homotopy equivalent* if there are continuous maps $f : \mathbb{X} \to \mathbb{Y}$ and $g : \mathbb{Y} \to \mathbb{X}$ such that $g \circ f \simeq \mathbb{1}_{\mathbb{X}}$ and $f \circ g \simeq \mathbb{1}_{\mathbb{Y}}$. This gives an equivalence relation, and we write $\mathbb{X} \simeq \mathbb{Y}$ and say that the two spaces have same *homotopy type*.

An important type of homotopy equivalence is induced by *deformation retractions*. We call a space $\mathbb{Y} \subseteq \mathbb{X}$ a *retract* of $\mathbb{X}$ if there is a continuous map $r : \mathbb{X} \to \mathbb{Y}$ that is the indentity $\mathbb{1}_{\mathbb{Y}}$ when restricted to $\mathbb{Y}$. The map r is called a *retraction*. We call $\mathbb{Y}$ a *deformation retract* and r a *deformation retraction* if there is a homotopy between the retraction r and $\mathbb{1}_{\mathbb{X}}$. We say that $\mathbb{X}$ *deformation retracts* into $\mathbb{Y}$. This is clearly a particular case of homotopy equivalence. See Figure 1.1 for examples. Actually, two spaces $\mathbb{X}$ and $\mathbb{Y}$ are homotopy equivalent *iff* there exists a third space $\mathbb{T}$ that contains both of them and that deformation retracts to each of them.

We will see in Chapter 4 that being homotopy equivalent is enough for having isomorphic homology groups, the topological invariants we study in this dissertation.

1.2 Triangulation of Topological Spaces

Geometric Simplicial Complex. Let $x_0, \cdots, x_d$ be points in the Euclidean space $\mathbb{R}^D$. A point $x = \sum_{i=0}^{d} \lambda_i x_i$, with each $\lambda_i \in \mathbb{R}$, is an *affine combination* of the x_i if the λ_i sum to 1. The *affine hull* is the set of affine combinations. We say that the $d+1$ points $x_0, \cdots, x_d$ are *affinely independent* if any two affine combinations $x = \sum_{i=0}^{d} \lambda_i x_i$ and $y = \sum_{i=0}^{d} \mu_i x_i$ are identical iff $\lambda_i = \mu_i$ for every i. An affine combination $x = \sum_{i=0}^{d} \lambda_i x_i$ is a

convex combination if all λ_i are non-negative. The *convex hull* is the set of convex combinations

$$\text{Conv}\,\{x_0, \cdots, x_d\} = \left\{ \sum_{i=0}^{d} \lambda_i x_i : \sum_i \lambda_i = 1 \text{ and } \lambda_i \geq 0 \text{ for all } i \right\} \quad (1.1)$$

A *d-simplex* is the convex hull of $d+1$ affinely independent points. The *dimension* of a d-simplex σ is $\dim \sigma = d$. Simplices of dimension 0 are called *vertices*, 1-simplices are called *edges*, 2-simplices *triangles* and 3-simplices tetrahedra (see Figure 1.2). Every subset of affinely independent points is

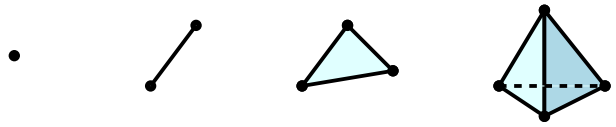


Figure 1.2: From left to right, vertex, edge, triangle and tetrahedron embedded in $\mathbb{R}^3$.

affinely independent and therefore also defines a simplex. A *face* of a simplex σ is the convex hull of a non-empty subset of the x_i and it is *proper* if the subset is not the entire set. If τ is a (proper) face of σ we say that σ is a (proper) coface of τ. The *boundary* of σ, denoted by $\text{bd}\,\sigma$, is the union of all of its proper faces and its *interior* is σ from which the boundary has been removed $\text{int}\,\sigma = \sigma \setminus \text{bd}\,\sigma$.

For the definitions above, we note that the interior of a simplex is an open set. The simplex itself is the closure of its interior and is hence closed.

A *geometric simplicial complex* $\mathbf{K}$ in $\mathbb{R}^D$ is a collection of simplices in $\mathbb{R}^D$ such that:

(i) every face of a simplex of $\mathbf{K}$ is in $\mathbf{K}$,

(ii) the intersection of any two simplices of $\mathbf{K}$ is either empty or a face of each of them.

The set of vertices of $\mathbf{K}$ is denoted by $\text{Vert}\,\mathbf{K}$. For simplicity, we identify each vertex $\{x_i\} \in \mathbf{K}$ with the corresponding point $x_i \in \mathbb{R}^D$.

The *dimension* of $\mathbf{K}$ is the maximum dimension of one of its simplices. A *subcomplex* of $\mathbf{K}$ is a simplicial complex $\mathbf{L} \subseteq \mathbf{K}$. A particular subcomplex of $\mathbf{K}$ is the *j-skeleton*, which contains all simplices of $\mathbf{K}$ of dimension less or equal to j, *ie* $\mathbf{K}^{(j)} = \{\sigma \in \mathbf{K} : \dim \sigma \leq j\}$.

A subset of a simplicial complex useful when discussing local neighborhoods is the *star* of a simplex τ consisting of all cofaces of τ, *ie* $\text{St}\,\tau = \{\sigma \in$

$\mathbf{K} : \tau \subseteq \sigma\}$. The star of a simplex is not a simplicial complex in general. We can make it into a complex by adding all missing faces. The result is the *closed star* $\overline{\text{St}}\ \tau$ which is the smallest subcomplex of $\mathbf{K}$ containing the faces of St τ. The *link* of τ consists of all the faces of its star that are disjoint from τ, *ie* Lk $\tau = \{\sigma \in \overline{\text{St}}\ \tau : \sigma \cap \tau = \emptyset\}$.

The *underlying space* of $\mathbf{K}$, denoted by $|\mathbf{K}|$, is the union of its simplices together with the subspace topology inherited from the ambient Euclidean space $\mathbb{R}^D$ in which the simplices live. It is a topological space.

We study in the following the combinatorial structure of simplicial complexes, through the concept of *abstract simplicial complex*. Note that related concepts between geometric and abstract simplicial complexes share the same name.

Abstract Simplicial Complex. All following definitions are the combinatorial equivalents of the geometric notions introduced above. Let V be an arbitrary finite set, called the set of *vertices*. A *simplex* σ is a subset of the vertices $\sigma = \{v_0, \cdots, v_d\} \subseteq V$, and we say that $v_0, \cdots, v_d$ are the *vertices* of σ. If a simplex σ contains precisely $d+1$ vertices ($d \geq 0$), it is called a d-simplex and has *dimension* $\dim \sigma = d$. A *face* of a simplex $\sigma = \{v_0, \cdots, v_d\}$ is a simplex whose vertices form a subset of $\{v_0, \cdots, v_d\}$. It is *proper* if it is different from σ. If τ is a (proper) face of σ we say that σ is a (proper) *coface* of τ. The *boundary* of σ is the set of its proper faces of maximal dimension ($\dim \sigma - 1$), *ie*

$$\partial\sigma = \partial\{v_0, \cdots, v_d\} = \{\{v_0, \cdots, \widehat{v_i}, \cdots, v_d\} : 0 \leq i \leq d\}$$

where the symbol $\widehat{v_i}$ means that v_i is removed from the set. We call the faces of the boundary of a simplex its *facets*.

An *abstract simplicial complex* on a set of vertices V is a set $\mathcal{K}$ of simplices that is required to satisfy the following two conditions:

(i) every vertex $v \in V$ is a 0-simplex $\{v\} \in \mathcal{K}$,

(ii) every face of a simplex in $\mathcal{K}$ is in $\mathcal{K}$, *ie* $\sigma \in \mathcal{K}, \tau \subseteq \sigma \Rightarrow \tau \in \mathcal{K}$.

The set of vertices of $\mathcal{K}$ is equal to V and is denoted by Vert $\mathcal{K}$.

The *dimension* of the simplicial complex $\mathcal{K}$ is the maximum dimension of its simplices. We denote by $\mathcal{K}^j$ the set of j-simplices of $\mathcal{K}$. A *subcomplex* of $\mathcal{K}$ is a abstract simplicial complex $\mathcal{L} \subseteq \mathcal{K}$. A particular subcomplex of $\mathcal{K}$ is the *j-skeleton*, denoted by $\mathcal{K}^{(j)}$, which contains all the faces of $\mathcal{K}$ of dimension at most j. In particular, the 1-skeleton of $\mathcal{K}$ contains the vertices and the

edges of $\mathcal{K}$ and has the structure of an undirected graph. We equivalently refer to the 1-skeleton of a simplicial complex or the *graph* of a simplicial complex.

The subset of simplices consisting of all the cofaces of a simplex $\tau \in \mathcal{K}$ is called the *star* of τ. The *link* of τ is defined as the set of faces

$$\mathrm{Lk}\tau = \{\sigma \in \mathcal{K} : \tau \cap \sigma = \emptyset \text{ and } \tau \cup \sigma \in \mathcal{K}\}$$

Geometric simplicial complexes and abstract simplicial complexes are concepts that are closely related, as explained in the next paragraph.

Abstraction and Geometric Realization. Let $\mathbf{K}$ be a geometric simplicial complex with $\mathrm{Vert}\,\mathbf{K} = \{x_1, \cdots, x_n\} \subseteq \mathbb{R}^D$. We also denote the set $\{x_1, \cdots, x_n\}$ by V and consider V as a set of (abstract) vertices. If, for each simplex $\sigma = \{x_{i_0}, \cdots, x_{i_d}\} \in \mathbf{K}$ we consider the (abstract) simplex $\upsigma = \{i_0, \cdots, i_d\}$, we obtain naturally an abstract simplicial complex $\mathcal{K}$ whose vertex set is V, specifically $\mathcal{K} = \{\upsigma : \sigma \in \mathbf{K}\}$. We call $\mathcal{K}$ the *abstraction* of $\mathbf{K}$.

Conversely, a *geometric realization* of an abstract simplicial complex $\mathcal{K}$ is a geometric simplicial complex $\mathbf{K}$ together with a bijective embedding $\phi : \mathrm{Vert}\,\mathcal{K} \to \mathrm{Vert}\,\mathbf{K}$ such that an abstract simplex $\upsigma = \{v_0, \cdots, v_d\}$ is in $\mathcal{K}$ *iff* the convex hull $\mathrm{Conv}\,\{\phi(v_0), \cdots, \phi(v_d)\}$ is a geometric simplex of $\mathbf{K}$. Every abstract simplicial complex admits an embedding in low dimension, as expressed by the following theorem:

Theorem 1.1 (Geometric Realization Theorem)**.** *Every abstract simplicial complex of dimension d admits a geometric realization in $\mathbb{R}^{2d+1}$.*

Consequently, geometric simplicial complexes and abstract simplicial complexes are closely related and the geometric and abstract definitions match. Note in particular that the notion of face and coface are identical, and by extension the notion of star and link. Note also that the concepts of boundary differ; the geometric boundary $\mathrm{bd}\,\sigma$ is defined to contain all proper faces of a simplex when the abstract boundary $\partial\upsigma$ contains only the maximal ones. However, the geometric realization restricted to the simplices of $\partial\upsigma$ is equal to the geometric boundary $\mathrm{bd}\,\sigma$. We use the same notations for both concepts in the following, refering to a simplicial complex $\mathbf{K}$, a simplex σ, etc.

Simplicial Maps. A *simplicial map* $f : \mathbf{K} \to \mathbf{L}$ between two simplicial complexes is a map that sends every vertex $v \in \mathrm{Vert}\,\mathbf{K}$ to a vertex $f(v) \in$

Vert$\mathbf{L}$, and every simplex $\{v_0, \cdots, v_d\} \in \mathbf{K}$ to a simplex $\{f(v_0), \cdots, f(v_n)\} \in \mathbf{L}$. Note that they may be redundancy in the set $\{f(v_0), \cdots, f(v_n)\}$, in which case the simplex image has lower dimension that its pre-image.

The simplicial map f can be extended to a continuous maps between the underlying spaces $|\mathbf{K}|$ and $|\mathbf{L}|$. Specifically, note that every point $x \in |\mathbf{K}|$ belongs to the interior of exactly one simplex $\mathbf{K}$. Let σ be that simplex, with $\sigma = \text{Conv}\,\{x_0, \cdots, x_d\}$. According to Equation 1.1, the point x may be uniquely written as $x = \sum_{i=0}^{d} \lambda_i x_i$, with $\sum_i \lambda_i = 1$ and $\lambda_i > 0$ for all i. The last inequality is strict because we have removed the boundary $\text{bd}\,\sigma$. We can consequently extend the simplicial map f to a continuous map $f : |\mathbf{K}| \to |\mathbf{L}|$ by setting $f(x) = \sum_{i=0}^{d} \lambda_i f(x_i)$ where $\sigma = \{x_0, \cdots, x_d\}$ is such that $x \in \text{int}\,\sigma$.

Consequently, simplicial complexes together with simplicial maps offer a piecewise linear counterpart to topological spaces and continuous maps. Even further, simplicial complexes can also represent, up to homeomorphism, of a much wider class of topological spaces – the *triangulable* spaces – and continuous maps between them. We elaborate on this point in the next paragraph.

Triangulation of Topological Spaces. A *triangulation* of a topological space $\mathbb{X}$ is a simplicial complex $\mathbf{K}$ together with a homeomorphism between $|\mathbf{K}|$ and $\mathbb{X}$. A topological space is *triangulable* if it has a triangulation. As a consequence, we can study the topological invariants of any triangulable topological space by studying a homeomorphic simplicial complex, that gives a discrete representation.

When simplicial maps can be extended to continuous maps, continuous maps between triangulable topological spaces can be approximated by simplicial maps. Specifically, we mention the following theorem:

Theorem 1.2 (Simplicial Approximation Theorem)**.** *Let* $\mathbf{K}$ *and* $\mathbf{L}$ *be two simplicial complexes, and* $h : |\mathbf{K}| \to |\mathbf{L}|$ *a continuous map between their underlying domains. There exist a* subdivision $\mathbf{K}'$ *of* $\mathbf{K}$ *and a simplicial map* $f : \mathbf{K}' \to \mathbf{L}$ *such that the continuous map* $|f| : |\mathbf{K}'| \to |\mathbf{L}|$ *is homotopic to* h*.*

Hence, simplicial complexes offer a complete encoding, up to homeomorphism, of triangulable topological spaces together with continuous maps between them.

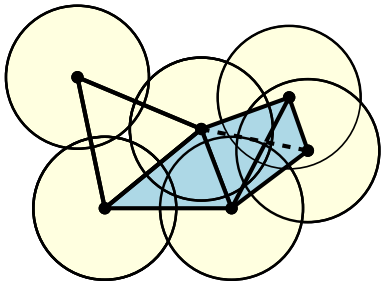


Figure 1.3: Čech complex of 6 points from the Euclidean space $\mathbb{R}^2$. The existence of a tetrahedron prevents the complex to be embeddable in $\mathbb{R}^2$.

1.3 Approximation of Topological Spaces

Topological Inference. Let $\mathcal{P}$ be a finite set of point in a metric space. Assuming $\mathcal{P}$ sample a topological space, we want to construct a simplicial complex using the points of $\mathcal{P}$ as vertices that is a faithful topological approximation of the underlying unknown topological space.

For a radius $\rho > 0$, consider the family $\mathcal{F}$ of closed balls $\overline{\mathcal{B}}(x, \rho)$ centered on the points $x \in \mathcal{P}$ and of radius ρ. The *nerve* of $\mathcal{F}$ consists in all non-empty subcollections of $\mathcal{F}$ whose balls have non-empty common intersection, *ie*

$$\mathrm{Nrv}\mathcal{F} = \{X \subseteq \mathcal{F} : \bigcap_{\overline{\mathcal{B}}(x,\rho)\in X} \overline{\mathcal{B}}(x, \rho) \neq \emptyset\}$$

The nerve of a collection of arbitrary sets is always an abstract simplicial complex. Moreover, we can relate the topology of the union of sets, if convex, with the topology of its nerve.

Theorem 1.3 (Nerve Theorem)**.** *Let $\mathcal{F}$ be a finite collection of closed convex sets in Euclidean space, and let $\mathbf{K}$ be its nerve* $\mathrm{Nrv}\mathcal{F}$*. Then the underlying space of $\mathbf{K}$ and the union of the sets in $\mathcal{F}$ have the same homotopy type.*

Čech Complex. For a radius $\rho \geq 0$, the *Čech complex* of a set of points $\mathcal{P}$ embedded in a metric space is the simplicial complex which is the nerve of the balls $\overline{\mathcal{B}}(x, \rho)$ centered on the points $x \in \mathcal{P}$ and of radius ρ

$$\mathcal{C}^\rho(\mathcal{P}) = \{\sigma \subseteq \mathcal{P} : \bigcap_{x\in\sigma} \mathcal{B}(x, \rho) \neq \emptyset\}$$

Note that the Čech complex of a set of points is not in general embedded in the metric space in which the points are, see for example Figure 1.3.

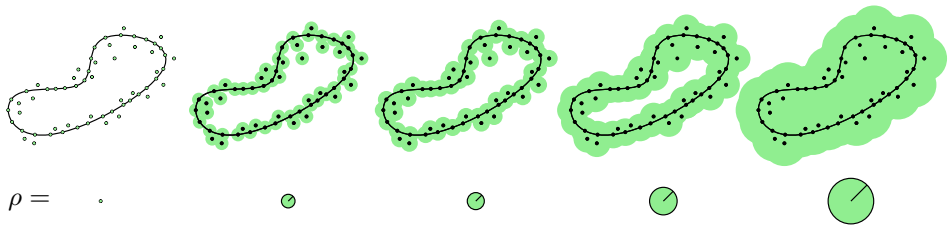


Figure 1.4: Union of balls centered on a dense sample of an unknown topological space, for different radii ρ. The first and second radii are to small to close the loop, when the last one is to big and fills up the hole. The third and fourth radii give a homotopy equivalent reconstruction.

The main property of Čech complexes is that, under mild conditions, they furnish a topologically correct reconstruction of a topological space,

Theorem 1.4 (Topological Reconstruction)**.** *Let C be a compact set of $\mathbb{R}^D$, with a strictly positive* topological feature size[1]*, and let $\mathcal{P}$ be a set of points such that $\mathsf{d}_{\mathcal{H}}(C, \mathcal{P}) < \varepsilon$. For sufficiently small ε, there exists a range $(\gamma_\varepsilon; \delta_\varepsilon)$ such that for every $\gamma_\varepsilon < \rho < \delta_\varepsilon$, C and $|\mathcal{C}^\rho(\mathcal{P})|$ are homotopy equivalent.*

We refer to Figure 1.4 for an example.

Intuitively, the small Hausdorff distance between the point cloud and the unknown domain ensures a dense sample. The proof of Theorem 1.4 uses the following arguments. One can define a retraction from the union of balls following a flow induced by the distance function to the compact domain C

$$\mathsf{d}(\cdot, C) : \mathbb{R}^D \to \mathbb{R}$$

See Figure 1.5 for an illustration. The domain C needs to be a compact for the flow to be well-defined. For a suitable radius ρ, the retract of the union of balls does not induce a change in topology. As a consequence, the retract defines an homotopy equivalence between C and the union of balls. By applying the Nerve Theorem 1.3, one concludes that the underlying space of the corresponding Čech complex has the same homotopy type as the compact C.

Despite its good topological approximation properties, the Čech complex is hard to construct algorithmically due to the algebraic complexity of the problem. Indeed, computing intersections of balls is equivalent to minimum

[1] A wide variety of topological reconstruction theorems exists, with different notion of topological feature size, which is a parameter like the *weak feature size*, the *μ-reach* or the *convexity defect*.

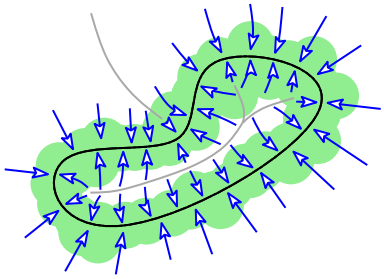


Figure 1.5: Arrows following the flow induced by the distance function to the compact domain.

enclosing ball queries. In the Euclidean space $\mathbb{R}^D$, the best known algorithm to compute the minimum enclosing ball of N points is linear in N, but depends superexponentially on the dimension D. Hence, it does not scale to high dimensions. Moreover evaluating higher dimensional geometric predicates results into numerical instability. We introduce in the following two families of simplicial complexes, the flag complexes and the witness complexes, both based on simpler geometric predicates.

Flag Complex. A *flag complex* is a simplicial complex whose combinatorial structure is entirely determined by its 1-skeleton. Specifically, a simplex is in the flag complex if and only if its vertices form a clique in the graph of the simplicial complex, or, in other terms, if and only if its vertices are pairwise linked by an edge.

The most popular flag complex for topological inference is the *Rips complex*, which may be seen as an approximation of the Čech complex. Specifically, for a threshold $\rho \geq 0$, the Rips complex $\mathcal{R}^\rho(\mathcal{P})$ on a set of points $\mathcal{P}$ in a metric space is the flag complex defined on the 1-skeleton of $\mathcal{C}^\rho(\mathcal{P})$. In other words, given a set of points $\mathcal{P}$, every two points are linked by an edge *iff* their distance is less or equal than 2ρ; the Rips complex is then the maximal simplicial complex admitting this graph as its 1-skeleton; see Figure 1.6.

As a consequence, the Čech complex $\mathcal{C}^\rho(\mathcal{P})$ is included in the Rips complex $\mathcal{R}^\rho(\mathcal{P})$. The converse is true if we augment slightly the threshold ρ,

Lemma 1.1. *For a set of points $\mathcal{P}$ in Euclidean space $\mathbb{R}^D$, and for every threshold $\rho \geq 0$,*

$$\mathcal{C}^\rho(\mathcal{P}) \subseteq \mathcal{R}^\rho(\mathcal{P}) \subseteq \mathcal{C}^{\vartheta_D \cdot \rho}(\mathcal{P}) \qquad \text{where } \vartheta_D = \sqrt{\frac{2D}{D+1}}$$

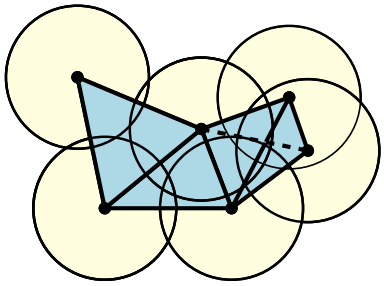


Figure 1.6: Rips complex of 6 points from the Euclidean space $\mathbb{R}^2$. Note that it contains one triangle that is not in the previous Čech complex with same threshold.

Hence, the Rips complex approximate the Čech complex, but does not inherit, in general, its properties for topological reconstruction. Note that computing the Rips complex of a set of points requires only distance computations. As a consequence, the Rips complex can be constructed in an arbitrary metric space.

Witness Complexes. Let L be a finite set of points, called the *landmarks*, and W a possibly infinite set of points, called the *witnesses*, embedded in a metric space. We say that a witness $w \in W$ *witnesses*, or *is a witness* for, a simplex $\sigma \subseteq L$ if

$$\forall x \in \sigma \text{ and } \forall y \in L \setminus \sigma, \quad \text{we have } \mathsf{d}(w, x) \leq \mathsf{d}(w, y)$$

For simplicity of exposition, we suppose that no landmarks are at the exact same distance to a witness. In this case, a witness $w \in W$ *witnesses* a simplex $\sigma \subseteq L$ *iff* the vertices of σ are the $\#\sigma$ nearest neighbors of w in L. The *witness complex* $\mathrm{Wit}(W, L)$ is the maximal simplicial complex, with vertices in L, whose faces admit a witness in W.

When the Rips complex is an approximation of the Čech complex with only distance computations, the witness complex is an approximation of the Delaunay triangulation

Theorem 1.5. *In the Euclidean space $\mathbb{R}^D$, the witness complex $\mathrm{Wit}(\mathbb{R}^D, L)$, picking the whole space for witnesses, is equal to the Delaunay triangulation of L.*

For computation the number of witnesses has to be finite, which induces missing simplices. We introduce a relaxed version of the complex. Given a

threshold $\rho \geq 0$ we define the *relaxed witness complex*. We say that a witness $w \in W$ *ρ-witnesses*, or is a ρ-witness for, a simplex $\sigma \subseteq L$ if

$$\forall x \in \sigma \text{ and } \forall y \in L \setminus \sigma, \quad \text{we have } \mathsf{d}(w, x) \leq \mathsf{d}(w, y) + \rho$$

The *relaxed witness complex* $\text{Wit}^{\rho}(W, L)$ with threshold ρ is the maximal simplicial complex, with vertices in L, whose faces admit a ρ-witness in W. For $\rho = 0$, the relaxed witness complex is the standard witness complex. Note that the witness complexes rely on nearest neighbors predicates, which are solved efficiently in high-dimension.

Filtration Function. The Čech complex, the Rips complex and the relaxed witness complex have the common point of depending on a threshold ρ, which defines a *filtration* of the space. A *filtration function* on a simplicial complex is a function $f : \mathbf{K} \to \mathbb{R}$ satisfying $f(\tau) \leq f(\sigma)$ whenever $\tau \subseteq \sigma$. If $\mathbf{K}$ contains the simplices $\{\sigma_1, \cdots, \sigma_n\}$, the sequence $[\sigma_i]_{i=1,\cdots,n}$ sorted according to increasing f values, and breaking ties so as a face of a simplex appears before it, induces the sequence of inclusions $\emptyset = \mathbf{K}_0 \subsetneq \mathbf{K}_1 \subsetneq \cdots \subsetneq \mathbf{K}_{n-1} \subsetneq \mathbf{K}_n = \mathbf{K}$, $\mathbf{K}_{i+1} = \mathbf{K}_i \cup \{\sigma_{i+1}\}$. In topological inference, a filtration function allows us to consider the reconstructed topological space at different scales. Considering Figure 1.4, for example, the correct threshold ρ for reconstruction is unknown. Computing a reconstruction of a space to learn its topology is one of the motivation of persistent homology introduced in Chapter 4, which defines the equivalent of a multi-scale topological invariant and a more general notion of filtration.

The main difficulty with these simplicial complexes is that their size increases extremely fast when ρ gets bigger. The next chapter introduces a data structure that is both optimal in memory and provides efficient construction of both Rips complex and witness complex.

Bibliographic Notes.

The introduction to topological spaces and metric spaces follows [44]. The introduction to homotopy and simplicial complexes follows [42] and [59]. We refer to these references for further reading.

We refer to [59] for more details on the Simplicial Approximation Theorem 1.2. This is a foundational result for algebraic topology, as topological invariants defined on simplicial complexes are well-defined on triangulable topological spaces. In particular, the simplicial approximation theorem will appear implicitly in Chapter 4 when defining homology groups.

The Nerve Theorem 1.3 is a classic result in homotopy theory. We refer to [47] for a proof. It relies on the fact that the intersection of finitely many convex sets is contractible, and a homotopy equivalence can be explicitely constructed from this fact. The best algorithm for the minimum enclosing ball is described in [68].

The main result for topological inference is the Topological Reconstruction Theorem 1.4. It results from a long list of articles, defining sampling conditions, measures of complexity of a shape and different level of reconstruction theorems. We refer to [27] for a definition of the different sampling conditions using parameters based on the medial axis, and to [8] for a more recent approach.

The witness complex has been studied in [33]. The equality with the Delaunay triangulation has been generalized in [17] under weaker sampling conditions, in particular with a finite set of witnesses. For more details on nearest neighbors computation in a general topological setting, we refer to [4].

Chapter 2

Simplex Tree Data Structure

In this chapter, we introduce a data structure to represent arbitrary simplicial complexes. Our approach aims at combining both generality and scalability. We propose a tree representation for simplicial complexes. The nodes of the tree are in bijection with the simplices (of all dimensions) of the simplicial complex. In this way, our data structure, called a *simplex tree* [15, 20], explicitly stores all the simplices of the complex but does not represent explicitly all the adjacency relations between the simplices, two simplices being adjacent if they share a common face. Storing all the simplices provides generality, and the tree structure of our representation enables us to implement basic operations on simplicial complexes efficiently, in particular to retrieve incidence relations, *ie* to retrieve the simplices that contain a given simplex or are contained in a given simplex.

2.1 Representation of Simplicial Complexes

Simplicial complexes are a generalization of graphs in the sense that they allow higher order adjacency, *ie* more than two vertices being connected by a simplex. In computer science, they were first used to represent surfaces in $\mathbb{R}^3$ and many specific data structures have been introduced for encoding them. This section does not aim at giving an inventory of all data structures dedicated to specific cases, but rather aim at raising attention to the difficulties encountered when generalizing a data structure for simplicial complexes encountered in computational topology. Hence, we focus on one data structure, that is in particular used in `CGAL` [1, 14], and try to generalize it.

Let $\mathbf{K}$ be a simplicial complex with $\mathrm{Vert}\,\mathbf{K} = \{1, \cdots, \#\mathrm{Vert}\,\mathbf{K}\}$. Suppose that $\mathbf{K}$ triangulates a surface. As a consequence, every maximal simplex is

a triangle and every triangle shares one of its edges with exactly one other triangle. We represent only the triangles and all other simplices are deduced by enumerating faces of these triangles. Each triangle maintains its three vertices and three pointers towards its adjacent triangles; see Figure 2.1. Additionally, because the dimension of the maximal simplices is fixed, ver-

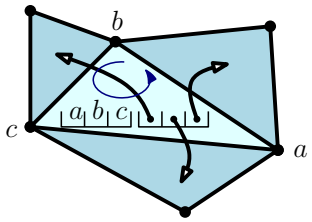


Figure 2.1: Data structure for surfaces, representing only the maximal simplices of a complex.

tices and pointers may be stored in arrays. Because each edge is shared by exactly two triangle, we can orient the triangles (counterclockwise in Figure 2.1) and order the vertices and pointers so as the i^{th} triangle pointed to is the one opposite to the i^{th} vertex stored.

This data structure naturally generalizes to d-dimensional simplicial complexes triangulating d-dimensional manifolds (the generalization of surfaces). We maintain only the d-simplices and each of them is represented by an array of $d+1$ vertices and an array of $d+1$ pointers towards the d-simplices sharing exactly one facet with the simplex. The structure also generalizes to simplicial complexes whose underlying domain has a boundary, *ie* where some simplices admit exactly one coface. This results into `NULL` pointers in the arrays. For a simplicial complex $\mathbf{K}$ of dimension k, and under the previous conditions, the memory complexity of the data structure is

$$\#\mathbf{K}^k \times 2(k+1)$$

Suppose now that the facets of a maximal simplex may be shared by strictly more than one other maximal simplex. Arrays need to be replaced by dynamic structures, like lists. Moreover, ordering according to an orientation does not allow anymore to associate vertices and there opposite faces. With a list representation, all inner data structures are twice bigger. Moreover, the full data structure has potentially a quadratic size in the number of k-dimensional simplices. Finally, consider the case where all maximal simplices do not have the same dimension. It is not clear how to link adjacent simplices of different dimension, like the triangle and the tetahedron in Figure 2.2.

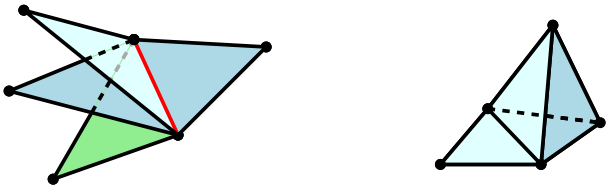


Figure 2.2: General adjacency relations in a complex.

The general configurations mentioned above and summarized in Figure 2.2 are common in computational topology and topological inference. A reason for that is the very general setting of the problem, *ie* points in a metric space allowing only distance computations and no higher dimensional geometric predicates. Moreover, the fact that Čech complex provides a topologically correct approximation of a topological domain relies on the ability of the union of balls, whose nerve is the Čech complex, to cover entirely the unknown space. As a consequence, many simplices are adjacent to the same facet.

Another information useful for computational topology is the filtration function $f\colon \mathbf{K} \to \mathbb{R}$ defined on a simplicial complex. All mentioned complexes so far are naturally filtered. Storing such information requires $\Omega(\#\mathbf{K})$ memory words. The data structure described above does not allow the representation of a filtration of a simplicial complex, as the non-maximal simplices are represented implicitly. We consequently take a different approach.

The *Hasse diagram* of a poset $(P, \preceq)$ is a graph containing one node per element in P and one directed edge $(x, y) \in P \times P$ for every elements x and y such that $x \prec y$ and there is no $z \in P$ such that $x \prec z \prec y$. A Hasse diagram is usually represented as a drawing of the graph in the plane, with bigger elements on top. For convenience, we reverse the representation and draw the smaller elements on top. Simplicial complexes are posets *wrt* the inclusion relation between simplices. Hence, they admit a Hasse diagram; see Figure 2.3.

The Hasse diagram is also a data structure. Let $\mathbf{K}$ be an arbitrary k-dimensional simplicial complex. The Hasse diagram of $\mathbf{K}$ contains $\#\mathbf{K} + 1$ nodes, and a node representing a d-simplex stores $d{+}1$ vertices and $d{+}1$ edges directed towards codimension 1 cofaces. The structure has size $O(k \times \#\mathbf{K})$ for a complex of dimension k. Note that the structure is insensitive to fact that maximal simplices may have distinct dimensions and that simplices may share facets with more than one other simplex.

Additionally, as every simplex is represented explicitely, one can represent

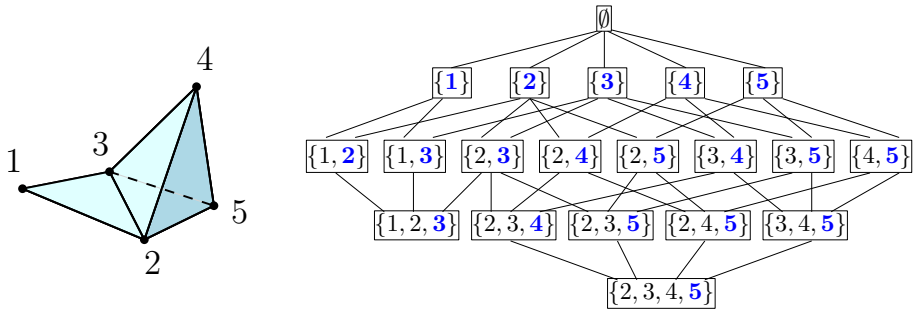


Figure 2.3: Simplicial complex and corresponding Hasse diagram.

a filtration on the complex.

The *simplex tree* data structure that we introduce in this chapter is a specific spanning tree of the Hasse diagram. Its properties allow us to store in a node only the biggest vertex of the simplex it represents and reach an optimal memory complexity of $O(1)$ word per simplex. The biggest labels are pictured in blue in Figure 2.3.

Before introducing the simplex tree, we mention other data structures used for representing simplicial complexes. Brisson [21] and Lienhardt [49] have introduced data structures to represent d-dimensional cell complexes, most notably subdivided manifolds. While those data structures have nice algebraic properties, they are very redundant and do not scale to large data sets or high dimensions. Zomorodian [69] has proposed the tidy set, a compact data structure to simplify a simplicial complex and compute its homology. The construction of the tidy set requires the computation of the maximal simplices of the simplicial complex, which is difficult in general without constructing the whole complex explicitly. In the same spirit, Attali et al. [6] have proposed the skeleton-blockers data structure. Again, the representation is general but it requires to compute blockers, the simplices which are not contained in the simplicial complex but whose proper faces are. Computing the blockers is as difficult as computing maximal simplices in general. These two last data structures are efficient in the special case of flag complexes, where the maximal simplices are the maximal cliques in the graph and the set of blockers is empty. None of these data structures are, at the same time, well-suited to general simplicial complexes and scale to both dimension and size.

2.2 The Simplex Tree

Let $\mathbf{K}$ be a simplicial complex of dimension k. The vertices are labeled from 1 to $\#\mathrm{Vert}\,\mathbf{K}$ and ordered accordingly. We can thus associate to each simplex of $\mathbf{K}$ a word on the alphabet $1\cdots\#\mathrm{Vert}\,\mathbf{K}$. Specifically, a j-simplex of $\mathbf{K}$ is uniquely represented as the word of length $j+1$ consisting of the ordered set of the labels of its $j+1$ vertices. Formally, let simplex $\sigma = \{v_{\ell_0}, \cdots, v_{\ell_j}\} \in \mathbf{K}$, where $v_{\ell_i} \in \mathrm{Vert}\,\mathbf{K}$, $\ell_i \in \{1, \cdots, \#\mathrm{Vert}\,\mathbf{K}\}$ and $\ell_0 < \cdots < \ell_j$. σ is represented by the word $[\sigma] = [\ell_0, \cdots, \ell_j]$. The last label of the word representation of a simplex σ is called the last label of σ and denoted by $last(\sigma)$.

The simplicial complex $\mathbf{K}$ can be defined as a collection of words on an alphabet of size $\#\mathrm{Vert}\,\mathbf{K}$. To compactly represent the set of simplices of $\mathbf{K}$, we store the corresponding words in a tree satisfying the following properties:

1. the nodes of the tree are in bijection with the simplices (of all dimensions) of the complex. The root is associated to the empty simplex,

2. each node of the tree, except the root, stores the label of a vertex. Specifically, a node associated to a simplex $\sigma \neq \emptyset$ stores the label $last(\sigma)$,

3. the vertices whose labels are encountered along a path from the root to a node associated to a simplex σ are the vertices of σ. Along such a path, the labels are sorted by increasing order and each label appears no more than once.

We call this data structure the *simplex tree* of $\mathbf{K}$. It is a trie [12] on the words representing the simplices of the complex; see Figure 2.4. The depth of the root is 0 and the depth of a node is equal to the dimension of the simplex it represents plus one. In addition, we augment the data structure so as to quickly locate all the instances of a given label in the tree. Specifically, all the nodes at a same depth j which contain a same label ℓ are linked in a circular list $L_j(\ell)$, as illustrated in Figure 2.4 for label $\ell = 5$.

The children of the root of the simplex tree are called the *top nodes*. The top nodes are in bijection with the elements of $\mathrm{Vert}\,\mathbf{K}$, the vertices of $\mathbf{K}$. Nodes which share the same parent (*eg* . the top nodes) are called *sibling nodes*. We also attach to each set of sibling nodes a pointer to their parent so that we can access a parent in constant time.

We give a constructive definition of the simplex tree. Starting from an empty tree, we insert the words representing the simplices of the complex in the following manner. When inserting the word $[\sigma] = [\ell_0, \cdots, \ell_j]$ we

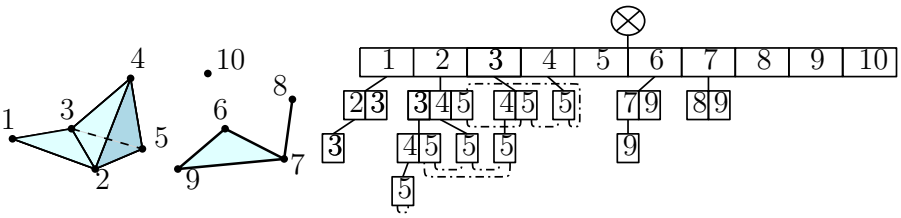


Figure 2.4: A simplicial complex on 10 vertices and its simplex tree. The deepest node represents the tetrahedron of the complex. All the positions of a given label at a given depth are linked in a list, as illustrated in the case of label 5.

start from the root, and follow the path containing successively all labels $\ell_0, \cdots, \ell_i$, where $[\ell_0, \cdots, \ell_i]$ denotes the longest prefix of $[\sigma]$ already stored in the simplex tree. We then append to the node representing $[\ell_0, \cdots, \ell_i]$ a path consisting of the nodes storing labels $\ell_{i+1}, \cdots, \ell_j$. It is easy to see that the three properties above are satisfied. Hence, if $\mathbf{K}$ consists of $\#\mathbf{K}$ simplices, the associated simplex tree contains exactly $\#\mathbf{K} + 1$ nodes, counting the empty simplex at the root.

We use dictionaries with size linear in the number of elements they store (like a red-black tree or a hash table) for searching, inserting and removing elements among a set of sibling nodes. Consequently these additional structures do not change the asymptotic memory complexity of the simplex tree. For the top nodes, we simply use an array since the set of vertices Vert is known and fixed. Let $\deg(\mathcal{T})$ denote the maximal outdegree of a node, in the simplex tree $\mathcal{T}$, distinct from the root. Remark that $\deg(\mathcal{T})$ is at most the maximal degree of a vertex in the graph of the simplicial complex. In the following, we will denote by D_{m} the maximal number of operations needed to perform a search, an insertion or a removal in a dictionary of maximal size $\deg(\mathcal{T})$ (for example, with red-black trees $D_{\mathrm{m}} = O(\log \deg(\mathcal{T}))$ worst-case, with hash-tables $D_{\mathrm{m}} = O(1)$ amortized). Some algorithms, that we describe later, require to intersect and to merge sets of sibling nodes. In order to compute fast set operations, we will prefer dictionaries which allow to traverse their elements in sorted order (*eg* red-black trees). We discuss the value of D_{m} at the end of this section in the case where the points have a geometric structure.

We introduce two new notations for the analysis of the complexity of the algorithms. Given a simplex $\sigma \in \mathbf{K}$, we define C_σ to be the number of cofaces of σ, *ie* $\#\mathrm{St}\sigma$. Note that C_σ only depends on the combinatorial structure of the simplicial complex $\mathbf{K}$. Let $\mathcal{T}$ be the simplex tree associated

to $\mathbf{K}$. Given a label ℓ and an index j, we define $\mathcal{T}_\ell^{>j}$ to be the number of nodes of $\mathcal{T}$ at depth strictly greater than j that store label ℓ. These nodes represent the simplices of dimension at least j that admit ℓ as their last label. $\mathcal{T}_\ell^{>j}$ depends on the labelling of the vertices and is bounded by $C_{\{v_\ell\}}$, the number of cofaces of the vertex with label ℓ. For example, if ℓ is the greatest label, we have $\mathcal{T}_\ell^{>0} = C_{\{v_\ell\}}$, and if ℓ is the smallest label we have $\mathcal{T}_\ell^{>0} = 1$ independently from the number of cofaces of $\{v_\ell\}$.

We provide in the following algorithms for

- `Search/Insert/Remove-simplex` to search, insert or remove a single simplex, and `Insert/Remove-full-simplex` to insert a simplex and its faces or remove a simplex and its cofaces,
- `Locate-cofaces` to locate the cofaces of a simplex,
- `Locate-boundary` to locate the boundary, *ie* the maximal proper faces, of a simplex,
- `Elementary-collapse` to proceed to an elementary collapse,
- `Edge-contraction` to proceed to the contraction of an edge.

Insertions and Removals. Using the previous top-down traversal, we can *search* and *insert* a word of length j in $O(jD_{\mathrm{m}})$ operations.

We can extend this algorithm so as to insert a simplex and all its faces in the simplex tree. Let σ be a simplex we want to insert with all its faces. Let $[\ell_0, \cdots, \ell_j]$ be its word representation. For i from 0 to j we insert, if not already present, a node N_{ℓ_i}, storing label ℓ_i, as a child of the root. We recursively call the algorithm on the subtree rooted at N_{ℓ_i} for the insertion of the suffix $[\ell_{i+1}, \cdots, \ell_j]$. Since the number of faces of a simplex of dimension j is $\sum_{i=0\cdots j+1} \binom{j+1}{i} = 2^{j+1}$, this algorithm takes time $O(2^j D_{\mathrm{m}})$.

We can also remove a simplex from the simplex tree. Note that to keep the property of being a simplicial complex, we need to remove all its cofaces as well. We locate them thanks to the algorithm described below.

Locate Cofaces. Computing the cofaces of a simplex is required to retrieve adjacency relations between simplices. In particular, it is useful when traversing the complex or when removing a simplex. We also need to compute the cofaces of a simplex when contracting an edge (described later) or during the construction of the witness complex, described later.

If τ is represented by the word $[\ell_0 \cdots \ell_j]$, the cofaces of τ are the simplices of $\mathbf{K}$ represented by words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$, where $\star$ denotes any word on the alphabet, possibly the empty word.

To locate all words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$ in the simplex tree, we first find all words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j]$. Using the lists $L_i(\ell_j)$ $(i > j)$, we find all nodes at depth at least $j + 1$ which contain label ℓ_j. For each such node N_{ℓ_j}, we traverse the tree upwards from N_{ℓ_j}, looking for a word of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j]$. If the search succeeds, the simplex represented by N_{ℓ_j} in the simplex tree is a coface of τ, as well as all the simplices represented by the nodes in the subtree rooted at N_{ℓ_j}, which have word representation of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$. Remark that the cofaces of a simplex are represented by a set of subtrees in the simplex tree. The procedure searches only for the roots of these subtrees.

The complexity of searching for the cofaces of a simplex σ of dimension j depends on the number $\mathcal{T}^{>j}_{last(\sigma)}$ of nodes with label $last(\sigma)$ and depth at least $j + 1$. If k is the dimension of the simplicial complex, traversing the tree upwards takes $O(k)$ time. The complexity of this procedure is thus $O(k\mathcal{T}^{>j}_{last(\sigma)})$.

Locate Boundary. Locating the boundary of a simplex efficiently is the key point of the incremental algorithm we use to construct witness complexes in Section 2.3, and is central when computing the homology and persistent homology of a simplicial complex in Part II.

Given a simplex σ, we access the nodes of the simplex tree representing the facets of σ. Let the word representation of σ be $[\ell_0, \cdots, \ell_j]$. Thus the word representations of the facets of σ are the words $[\ell_0, \cdots, \widehat{\ell_i}, \cdots, \ell_j]$, for all $0 \leq i \leq j$, where $\widehat{\ell_i}$ indicates that ℓ_i is omitted. As before, we denote by N_{ℓ_i}, for all $i = 0, \cdots, j$, the nodes representing the words $[\ell_0, \cdots, \ell_i]$, $i = 0, \cdots, j$ respectively. A traversal from the node representing σ up to the root will exactly pass through the nodes N_{ℓ_i}, $i = j, \cdots, 0$. When reaching the node $N_{\ell_{i-1}}$, a search from $N_{\ell_{i-1}}$ downwards for the word $[\ell_{i+1}, \cdots, \ell_j]$ locates (or proves the absence of) the facet $[\ell_0, \cdots, \widehat{\ell_i}, \cdots, \ell_j]$. See Figure 2.5 for a running example. This procedure locates the entire boundary of a j-simplex σ in $O(j^2 D_{\mathrm{m}})$ operations.

We now present the implemention of two topology preserving operations on a simplicial complex represented as a simplex tree, specifically *elementary collapses* and *edge contractions*. Both have been used in computational topology in order to reduce the size of a simplicial complex before studying its topology. These operations do not change the homotopy type of the

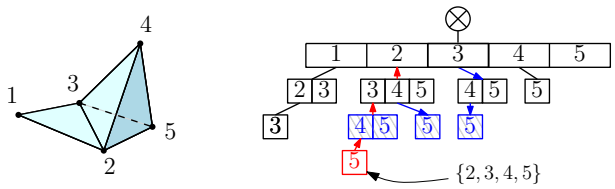


Figure 2.5: Boundary computation of the simplex $\sigma = \{2, 3, 4, 5\}$, starting from the position of σ in the simplex tree. The nodes representing the facets are colored in grey.

simplicial complex.

Elementary Collapse. We say that a simplex σ is *collapsible* through one of its simplex τ if σ is the only coface of τ. This can be verified algorithmically by computing the cofaces of τ. Such a pair of simplices (τ, σ) is called a *free pair*. Removing both simplices of a free pair is an *elementary collapse*.

Since τ has no coface other than σ, either the node representing τ in the simplex tree is a leaf (and so is the node representing σ), or it has the node representing σ as its unique child. An elementary collapse of the free pair (τ, σ) consists either in the removal of the two leaves representing τ and σ, or the removal of the subtree containing exactly two nodes: the node representing τ and the node representing σ.

Edge Contraction. Edge contractions are used in [7] as a tool for homotopy preserving simplification and in [40, 66] for designing small filtrations for topological inference.

It has been proved in [7, 39] that contracting an edge $\{v_{\ell_a}, v_{\ell_b}\}$ preserves the homotopy type of a simplicial complex whenever the *link condition* is satisfied

$$\text{Lk}\ \{v_{\ell_a}, v_{\ell_b}\} = \text{Lk}\ \{v_{\ell_a}\}\ \cap\ \text{Lk}\ \{v_{\ell_b}\}$$

The link condition can be checked algorithmically using the procedure `Locate-cofaces`.

Let $\mathbf{K}$ be a simplicial complex. Given an edge $\{v_{\ell_a}, v_{\ell_b}\}$ in the complex, the *contraction* of v_{ℓ_b} to v_{ℓ_a} is the simplicial map $f : \mathbf{K} \to \mathbf{K}$ that maps v_{ℓ_b} to v_{ℓ_a} and acts as the identity function on all other vertices

$$f(u) = \begin{cases} v_{\ell_a} & \text{if} \quad u = v_{\ell_b} \\ u & \text{otherwise} \end{cases}$$

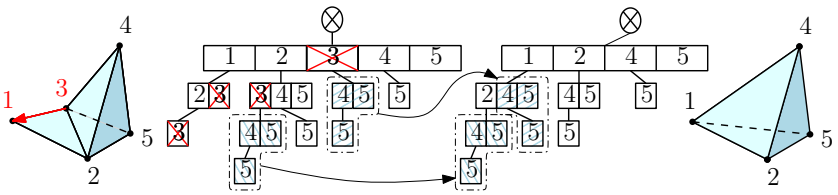


Figure 2.6: Contraction of vertex 3 to vertex 1 and the associated modifications of the simplicial complex and of the simplex tree. The nodes which are removed are marked with a red cross, the subtrees which are moved are colored in blue.

A consequence of the contraction is that the link of v_{ℓ_a} is augmented with the link of simplex $\{v_{\ell_b}\}$ and, v_{ℓ_b} and its cofaces are removed from the complex. Specifically, for every simplex $\sigma \in \mathbf{K}$, we distinguish three cases

1. σ does not contain v_{ℓ_b} and remains unchanged,
2. σ contains both v_{ℓ_a} and v_{ℓ_b}, hence $f(\sigma) = \sigma \setminus \{v_{\ell_b}\}$, $\#f(\sigma) = \#\sigma - 1$ and $f(\sigma)$ is a proper face of σ,
3. σ contains v_{ℓ_b} but not v_{ℓ_a} and $f(\sigma) = (\sigma \setminus \{v_{\ell_b}\}) \cup \{v_{\ell_a}\}$, $(\#f(\sigma) = \#\sigma)$.

We describe how to compute the contraction of v_{ℓ_b} to v_{ℓ_a} when $\mathbf{K}$ is represented as a simplex tree. We suppose that the edge $\{v_{\ell_a}, v_{\ell_b}\}$ is in the complex and, without loss of generality, $\ell_a < \ell_b$. All the simplices which do not contain v_{ℓ_b} remain unchanged (case 1.) and we do not consider them. If a simplex σ contains both v_{ℓ_a} and v_{ℓ_b} (case 2.), it becomes $\sigma \setminus \{v_{\ell_b}\}$ after edge contraction, which is a simplex already in $\mathbf{K}$. We simply remove σ from the simplex tree. Finally, if σ contains v_{ℓ_b} but not v_{ℓ_a} (case 3.), we remove σ from the simplex tree and add the new simplex $(\sigma \setminus \{v_{\ell_b}\}) \cup \{v_{\ell_a}\}$.

We consider each node N_{ℓ_b} with label ℓ_b in turn. To do so, we use the lists $L_j(\ell)$ which link all nodes cointaining the label ℓ at depth j. Let σ be the simplex represented by N_{ℓ_b}. The algorithm traverses the tree upwards from N_{ℓ_b} and collects the vertices of σ. Let $T_{N_{\ell_b}}$ be the subtree rooted at N_{ℓ_b}. As $\ell_a < \ell_b$, σ containing both v_{ℓ_a} and v_{ℓ_b} implies that every simplex represented by a node in $T_{N_{\ell_b}}$ also contains both v_{ℓ_a} and v_{ℓ_b}. Moreover, if σ contains only v_{ℓ_b}, so does every simplex representated by a node in $T_{N_{\ell_b}}$. Consequently, when σ contains both v_{ℓ_a} and v_{ℓ_b}, we remove the whole subtree $T_{N_{\ell_b}}$ from the simplex tree. When σ contains only v_{ℓ_b}, all words represented in $T_{N_{\ell_b}}$ are of the form $[\sigma'] \cdot [\sigma''] \cdot [\ell_b] \cdot [\sigma''']$ and turn into words $[\sigma'] \cdot [\ell_a] \cdot [\sigma''] \cdot [\sigma''']$ after edge contraction, where $\cdot$ denotes the concatenation

of words. We consequently move the subtree $T_{N_{\ell_b}}$ (apart from its root) from position $[\sigma'] \cdot [\sigma'']$ to position $[\sigma'] \cdot [\ell_a] \cdot [\sigma'']$ in the simplex tree. If a subtree is already rooted at this position, we have to merge $T_{N_{\ell_b}}$ with this subtree as illustrated in Figure 2.6. In order to merge the subtree $T_{N_{\ell_b}}$ with the subtree rooted at the node representing the word $[\sigma'] \cdot [\ell_a] \cdot [\sigma'']$, we successively insert every node of $T_{N_{\ell_b}}$ in the corresponding set of sibling nodes, stored in a dictionary. See Figure 2.6.

We analyze the complexity of contracting an edge $\{v_{\ell_a}, v_{\ell_b}\}$. For each node storing the label ℓ_b, we traverse the tree upwards. This takes $O(k)$ operations for a simplicial complex of dimension k. As there are $\mathcal{T}_{\ell_b}^{>0}$ such nodes, the total cost is $O(k\mathcal{T}_{\ell_b}^{>0})$. We also manipulate the subtrees rooted at the nodes storing label ℓ_b. Specifically, either we remove such a subtree or we move a subtree by changing its parent node. In the later case, we have to merge two subtrees. This is the more costly scenario and it takes, in the worst case, $O(D_{\mathrm{m}})$ operations per node in the subtrees to be merged. As any node in such a subtree represents a coface of vertex v_{ℓ_b}, the total number of nodes in all the subtrees we have to manipulate is at most $C_{\{v_{\ell_b}\}}$, and the manipulation of the subtrees takes $O(C_{\{v_{\ell_b}\}} D_{\mathrm{m}})$ operations in total. Consequently, the time needed to contract the edge $\{v_{\ell_a}, v_{\ell_b}\}$ is $O(k\mathcal{T}_{\ell_b}^{>0} + C_{\{v_{\ell_b}\}} D_{\mathrm{m}})$.

Remark 2.1. *The quantity D_m appears as a key value in the complexity analysis of the algorithms. Recall that D_m is the maximal number of operations needed to perform a search, an insertion or a removal in a dictionary of maximal size $deg(\mathcal{T})$ in the simplex tree. As mentioned earlier, $deg(\mathcal{T})$ is bounded by the maximal degree of a vertex in the graph of the simplicial complex. For an arbitrary simplicial complex, $deg(\mathcal{T})$ may be as big as the number of vertices in the complex. However, in a framework where the vertices are points in a space, $deg(\mathcal{T})$ is bounded by a small quantity depending only on the intrinsic dimension of the point cloud [20].*

2.3 Construction of Simplicial Complexes

In this section, we describe algorithm to construct flag complexes and witness complexes using a simplex tree.

Flag Complex. Given the 1-skeleton, presented as a graph, of a flag complex, we call *expansion of order k* the operation which consists in constructing the k-skeleton of the flag complex. If the 1-skeleton is stored in a simplex

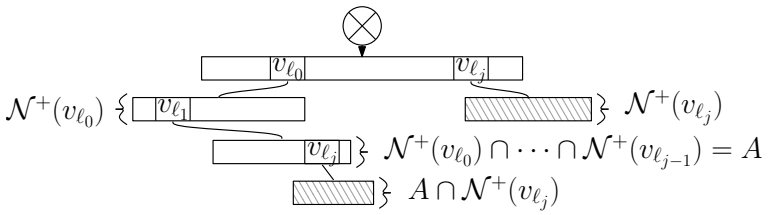


Figure 2.7: Representation of a set of sibling nodes as intersection of neighborhoods.

tree, the expansion of order k consists in successively inserting all the simplices of the k-skeleton into the simplex tree.

Let $\mathcal{G} = (V, E)$ be the graph of the simplicial complex, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. For a vertex $v_\ell \in V$, we denote by

$$\mathcal{N}^+(v_\ell) = \{\ell' \in \{1, \cdots, \#V\} \ : \ (v_\ell, v_{\ell'}) \in E \text{ and } \ell' > \ell\}$$

the set of labels of the neighbors of v_ℓ in $\mathcal{G}$ that are bigger than ℓ. Let N_{ℓ_j} be the node in the tree that stores the label ℓ_j and represents the word $[\ell_0, \cdots, \ell_j]$. The children of N_{ℓ_j} store the labels in $\mathcal{N}^+(v_{\ell_0}) \cap \cdots \cap \mathcal{N}^+(v_{\ell_j})$. Indeed, the children of N_{ℓ_j} are neighbors in $\mathcal{G}$ of the vertices v_{ℓ_i}, $0 \leq i \leq j$, (by definition of a clique) and must have a bigger label than $\ell_0, \cdots, \ell_j$ (by construction of the simplex tree). Consequently, the sibling nodes of N_{ℓ_j} are exactly the nodes that store the labels in $A = \mathcal{N}^+(v_{\ell_0}) \ \cap \ \cdots \ \cap \ \mathcal{N}^+(v_{\ell_{j-1}})$, and the children of N_{ℓ_j} are exactly the nodes that store the labels in $A \cap \mathcal{N}^+(v_{\ell_j})$; see Figure 2.7. For every vertex v_ℓ, we have an easy access to $\mathcal{N}^+(v_\ell)$ since $\mathcal{N}^+(v_\ell)$ is exactly the set of labels stored in the children of the top node storing label ℓ. We easily deduce an in-depth expansion algorithm.

The time complexity for the expansion algorithm depends on our ability to fastly compute intersections of the type $A \cap \mathcal{N}^+(v_{\ell_j})$. In all of our experiments in Chapter 3 on the Rips complex we have observed that the time taken by the expansion algorithm depends linearly on the size of the output simplicial complex, for a fixed dimension.

Witness Complex. Let L and W be respectively a set of landmarks and witnesses. For the description of the algorithm, we call a simplex σ *fully witnessed* if it is witnessed and all of its facets are in the witness complex $\mathrm{Wit}(W, L)$.

We suppose the sets L and W finite and give them labels $\{1, \cdots, \#L\}$ and $\{1, \cdots, \#W\}$ respectively. We describe an algorithm to construct the k-skeleton of the witness complex, for an arbitrary dimension k. Our construction algorithm is incremental, from lower to higher dimensions. At step j we insert in the simplex tree the j-dimensional fully witnessed simplices.

During the construction of the k-skeleton of the witness complex, we need to access the nearest neighbors of the witnesses, in L. To do so, we compute the $k+1$ nearest neighbors of all the witnesses in a preprocessing phase, and store them in a $\#W \times (k+1)$ matrix. Given an index $j \in \{0, \cdots, k\}$ and a witness $w \in W$, we access in constant time the $(j+1)^{\text{th}}$ nearest neighbor of w. We denote this landmark by s_j^w. We maintain a list of *active witnesses*, initialized with W. We insert the vertices of $\mathrm{Wit}(W, L)$ in the simplex tree. For each witness $w \in W$ we insert a top node storing the label of the nearest neighbor of w in L, if no such node already exists. w is initially an active witness and we make it point to the node mentionned above, representing the 0-dimensional simplex w witnesses. We maintain the following loop invariants:

1. at the beginning of iteration j, the simplex tree contains the $(j-1)$-skeleton of the witness complex $\mathrm{Wit}(W, L)$,

2. the active witnesses are the elements of W that witness a $(j-1)$-simplex of the complex; each active witness w points to the node representing the $(j-1)$-simplex in the tree it witnesses.

At iteration $j \geq 1$, we traverse the list of active witnesses. Let w be an active witness. We first retrieve the $(j+1)^{\text{th}}$ nearest neighbor s_j^w of w from the nearest neighbors matrix (Step 1). Let σ_j be the j-simplex witnessed by w and let us decompose the word representing σ_j into $[\sigma_j] = [\sigma'] \centerdot [s_j^w] \centerdot [\sigma'']$, where $\centerdot$ denotes the concatenation of words. We then look for the location in the tree where σ_j might be inserted (Step 2). To do so, we start at the node N_w which represents the $(j-1)$-simplex witnessed by w. Observe that the word associated to the path from the root to N_w is exactly $[\sigma'] \centerdot [\sigma'']$. We walk $\#[\sigma'']$ steps up from N_w, reach the node representing $[\sigma']$ and then search downwards for the word $[s_w^j] \centerdot [\sigma'']$; see Figure 2.8, left. The cost of this operation is $O(jD_{\mathrm{m}})$.

If the node representing σ_j exists, σ_j has already been inserted; we update the pointer of the active witness w and return. If the simplex tree contains neither this node nor its father, σ_j is not fully witnessed because the facet represented by its longest prefix is missing. We consequently remove w from the set of active witnesses. Lastly, if the node is not in the tree but its

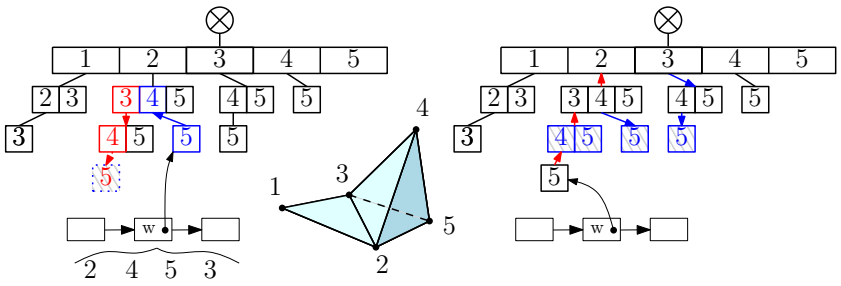


Figure 2.8: Third iteration of the witness complex construction. The active witness w witnesses the tetrahedron $\{2,3,4,5\}$ and points to the triangle $\{2,4,5\}$. Left: search for the potential position of the simplex $\{2,3,4,5\}$ in the simplex tree. Right: facets location for simplex $\{2,3,4,5\}$ and update of the pointer of the active witness w.

father is, we check whether σ_j is fully witnessed. To do so, we search for the $j+1$ facets of σ_j in the simplex tree (Step 3). The cost of this operation is $O(j^2 D_{\mathrm{m}})$ using the `Locate-boundary` algorithm described in Section 2.2. If σ_j is fully witnessed, we insert σ_j in the simplex tree and update the pointer of the active witness w. Else, we remove w from the list of active witnesses; see Figure 2.8, right. It is easily seen that the loop invariants are satisfied at the end of iteration j.

We study the complexity of the construction. The cost of accessing a neighbor of a witness using the nearest neighbors matrix is $O(1)$. We access a neighbor (Step 1) and locate a node in the simplex tree (Step 2) at most $k\#W$ times during the whole construction. In total, the cost of Steps 1 and 2 together is $O(\#W k^2 D_{\mathrm{m}})$. In Step 3, either we insert a new node in the simplex tree, which happens exactly $\#\mathbf{K}$ times, where the simplicial complex $\mathbf{K}$ is the output, or we remove an active witness, which happens at most $\#W$ times. The total cost of Step 3 is thus $O((\#\mathbf{K}+\#W)k^2 D_{\mathrm{m}})$. In conclusion, constructing the k-skeleton $\mathbf{K}$ of the witness complex takes time

$$O((\#\mathbf{K}+\#W)k^2 D_{\mathrm{m}} + k\#W) = O((\#\mathbf{K}+\#W)k^2 D_{\mathrm{m}})$$

Landmark Insertion. We present an algorithm to update the simplex tree under landmark insertions. Given the set of landmarks L, the set of witnesses W and the k-skeleton of the witness complex $\mathrm{Wit}(W,L)$ represented as a simplex tree, we take a new landmark point x and we update the simplex tree so as to construct the simplex tree associated to $\mathrm{Wit}(W, L\cup\{x\})$. We assign to x the biggest label $\#L+1$. We suppose we have at our disposal

an oracle that can compute the subset $W^x \subseteq W$ of the witnesses that admit x as one of their $k+1$ nearest neighbors. Computing W^x is known as the *reverse nearest neighbor* search problem.

Let w be a witness in W^x and suppose x is its $(i+1)^{\text{th}}$ nearest neighbor in $L \cup \{x\}$, with $0 \leq i \leq k$. Let $\sigma_j \subseteq L$ be the j-dimensional simplex witnessed by w in L and let $\widetilde{\sigma_j} \subseteq L \cup \{x\}$ be the j-dimensional simplex witnessed by w in $L \cup \{x\}$. Consequently, $\sigma_j = \widetilde{\sigma_j}$ for $j < i$ and $\sigma_j \neq \widetilde{\sigma_j}$ for $j \geq i$. We equip each node N of the simplex tree with a *counter of witnesses* which maintains the number of witnesses that witness the simplex represented by N. As for the witness complex construction, we consider all nodes representing simplices witnessed by elements of W^x, proceeding by increasing dimensions. For a witness $w \in W^x$ and a dimension $j \geq i$, we decrement the witness counter of σ_j and insert $\widetilde{\sigma_j}$ if and only if its facets are in the simplex tree. We remark that $[\widetilde{\sigma_j}] = [\sigma_{j-1}] \centerdot [x]$ because x has the biggest label of all landmarks. We can thus access in time $O(D_{\mathrm{m}})$ the position of the word $[\widetilde{\sigma_j}]$ since we have accessed the node representing $[\sigma_{j-1}]$ in the previous iteration of the algorithm.

If the witness counter of a node is turned down to 0, the simplex σ it represents is not witnessed anymore, and is consequently not part of $\mathrm{Wit}(W, L \cup \{x\})$. We remove the nodes representing σ and its cofaces from the simplex tree, using `Locate-cofaces`.

The update procedure is a "local" variant of the witness complex construction, where, by "local", we mean that we reconstruct only the star of vertex x. It admits an output-sensitive complexity analysis. Let C_x denote the number of cofaces of x in $\mathrm{Wit}(W, L \cup \{x\})$. The same analysis as above shows that updating the simplicial complex takes time $O((\#W^x + C_x)k^2 D_{\mathrm{m}})$, plus one call to the oracle to compute W^x.

Relaxed Witness Complex. Given W, L and $\rho \geq 0$, we resort to a similar incremental algorithm as above for the construction of $\mathrm{Wit}^{\rho}(W, L)$. At each step j, we insert, for each witness w, the j-dimensional simplices which are ρ-witnessed by w. Differently from the standard witness complex though, there may be more than one j-simplex that is witnessed by a given witness $w \in W$. Consequently, we do not maintain a pointer from each active witness to the last inserted simplex it witnesses. We use simple top-down insertions from the root of the simplex tree.

Given a witness w and a dimension j, we generate all the j-dimensional simplices which are ρ-witnessed by w. For the ease of exposition, we suppose we are given the sorted list of nearest neighbors of w in L, denoted by

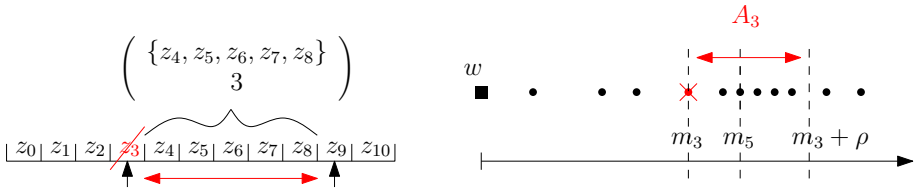


Figure 2.9: Computation of the ρ-witnessed simplices σ of dimension 5. If z_3 is the first neighbor of w not in σ, then σ contains $\{z_0, z_1, z_2\}$ and any 3-uplet of $A_3 = \{z_4, \cdots, z_8\}$.

$\{z_0 \cdots z_{\#L-1}\}$, and their distance to w, denoted by $m_i = \mathsf{d}(w, z_i)$, with $m_0 \leq \cdots \leq m_{\#L-1}$, breaking ties arbitrarily. Note that if one wants to construct only the k-skeleton of the complex, it is sufficient to know the list of neighbors of w that are at distance at most $m_k + \rho$ from w. We preprocess this list of neighbors for all witnesses. For $i \in \{0, \cdots, \#L-1\}$, we define the set A_i of landmarks z such that $m_i \leq d(w, z) \leq m_i + \rho$. For $i \leq j+1$, w ρ-witnesses all the j-simplices that contain $\{z_0, \cdots, z_{i-1}\}$ and a $(j+1-i)$-subset of A_i, provided $\#A_i \geq j+1-i$. We see that all j-simplices that are ρ-witnessed by w are obtained this way, and exactly once, when i ranges from 0 to $j+1$.

For all $i \in \{0, \cdots, j+1\}$, we compute A_i and generate all the simplices which contain $\{z_0, \cdots, z_{i-1}\}$ and a subset of A_i of size $(j+1-i)$. In order to easily update A_i when i is incremented, we maintain two pointers to the list of neighbors, one to z_i and the other to the end of A_i. We check in constant time if A_i contains more than $j+1-i$ vertices, and compute all the subsets of A_i of cardinality $j+1-i$ accordingly; see Figure 2.9.

We study the complexity of the algorithm. Let R_j be the number of j-simplices ρ-witnessed by w. Generating all those simplices takes $O(j+R_j)$ time. Indeed, for all i from 0 to $j+1$, we construct A_i and check whether A_i contains more than $j+1-i$ elements. This is done by a simple traversal of the list of neighbors of w, which takes $O(j)$ time. Then, when A_i contains more than $j+1-i$ elements, we generate all subsets of A_i of size $j+1-i$ in time $O(\binom{\#A_i}{j+1-i})$. As each such subset leads to a ρ-witnessed simplex, the total cost for generating all those simplices is $O(R_j)$.

We can deduce the complexity of the construction of the k-skeleton of the relaxed witness complex. Let $\mathcal{R} = \sum_{w \in W} \sum_{j=0 \cdots k} R_j$ be the number of ρ-witnessed simplices we try to insert. The construction of the relaxed witness complex takes $O(\mathcal{R} k^2 D_{\mathrm{m}})$ operations. This bound is quite pessimistic and,

in practice, we observed that the construction time is sensitive to the size of the output complex. Observe that the quantity analogous to $\mathcal{R}$ in the case of the standard witness complex was $k\#W$ and that the complexity was better due to our use of the notion of active witnesses.

Chapter 3

Performance of the Simplex Tree

In this section, we report on the performance of the `Locate-boundary` algorithm, `Locate-cofaces` algorithm and the construction of Rips complexes, witness complexes and relaxed witness complexes with a simplex tree. We compare the performance for the construction of simplicial complexes with `JPlex` and `Dionysus`. We also provide a study of the memory performance of the simplex tree compared to other representations. We use a variety of real and synthetic data, listed in Figure 3.3 with details on the sets of points $\mathcal{P}$ or landmarks L and witnesses W, their size $\#\mathcal{P}$, $\#L$ and $\#W$, the ambient dimension D, the intrinsic dimension d of the object the sample points belong to (if known), the threshold ρ for the complex, the dimension k up to which we construct the complexes, the time T_{g} to construct the Rips graph or the time T_{nn} to compute the lists of nearest neighbors of the witnesses, the number of edges $\#E$, the time for the construction of the Rips complex T_{Rips} or for the construction of the witness complex T_{Wit^ρ}, the size of the complex $\#\mathbf{K}$, and the total construction time T_{tot} and average construction time per simplex $T_{\mathrm{tot}}/\#\mathbf{K}$. Unless mentioned otherwise, all simplicial complexes are computed up to the embedding dimension.

We use the `ANN` library [58] to compute the 1-skeleton graph of the Rips complex, and to compute the lists of nearest neighbors of the witnesses for the witness complexes. For its efficiency and flexibility, we use the `map` container of the `Standard Template Library` for storing sets of sibling nodes, except for the top nodes which are stored in an array. Experiments are stopped after one hour of computation, and data missing on plots means that the computation ran above this time limit. For readability, we first report on

dim j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$\#\mathbf{K}^j$	300	2700	8057	15906	25271	30180	26568	17618	8900	3445	1015	217	30	2

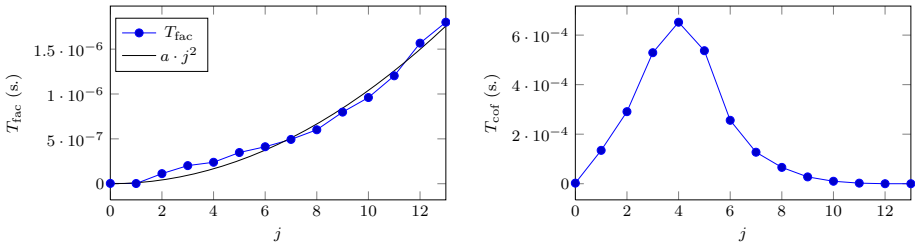


Figure 3.1: Repartition of the number of simplices per dimension (top) and average time to locate the boundary (left) and the cofaces (right) of a simplex of a given dimension.

the performance of each algorithm on a subset of the data, and furnish more timings at the end of the paragraphs.

As illustrated in Figure 3.3, we are able to construct and represent both Rips and relaxed witness complexes of up to several hundred million simplices in high dimensions, on all datasets.

Data Structures in `JPlex` and `Dionysus`. Both `JPlex` and `Dionysus` represent the combinatorial structure of a simplicial complex by its Hasse diagram. `JPlex` and `Dionysus` are libraries dedicated to topological data analysis, where only the construction of simplicial complexes and the computation of the boundary of a simplex are necessary.

For a simplicial complex $\mathbf{K}$ of dimension k and a simplex $\sigma \in \mathbf{K}$ of dimension j, the Hasse diagram has size $\Theta(k\,\#\mathbf{K})$ and allows to compute `Locate-boundary`(σ) in time $O(j)$, whereas the simplex tree has size $\Theta(\#\mathbf{K})$ and allows to compute `Locate-boundary`(σ) in time $O(j^2 D_{\text{m}})$.

3.1 Incidence Retrieval

We report on the experimental performance of the boundary and cofaces location algorithms. They are at the heart of other computations, like the edge contractions and the construction of witness complexes. Figure 3.1 represents the average time for these operations on a simplex, as a function of the dimension of the simplex. We use the dataset **Bro**, consisting of points in $\mathbb{R}^{25}$, on top of which we build a relaxed witness complex with 300 landmarks and 15,000 witnesses, and relaxation parameter $\rho = 0.15$. We

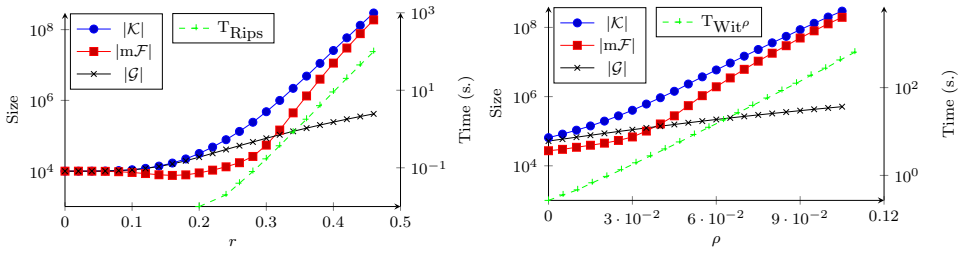


Figure 3.2: Statistics and timings for the Rips complex (Left) and the relaxed witness complex (Right) on **S4**.

obtain a 13-dimensional simplicial complex with 140,000 simplices in less than 3 seconds.

The theoretical complexity for computing the boundary of a j-simplex σ is $O(j^2 D_{\mathrm{m}})$. As reported in Figure 3.1, the average time to search all facets of a j-simplex is well approximated by a quadratic function on the dimension j (the standard error in the approximation is 2.0%).

A bound on the complexity for computing the cofaces of a j-simplex σ is $O(k\mathcal{T}^{>j}_{last(\sigma)})$, where $\mathcal{T}^{>j}_{last(\sigma)}$ stands for the number of nodes in the simplex tree that store the label $last(\sigma)$ and have depth larger than $j+1$. Figure 3.1 provides experimental results for a random labelling of the vertices. As can be seen, the time for computing the cofaces of a simplex σ is low, on average, when the dimension of σ is either small (0 to 2) or big (6 to 13), and higher for intermediate dimensions (3 to 5). The value $\mathcal{T}^{>j}_{last(\sigma)}$ in the complexity analysis depends on both the labelling of the vertices and the number of cofaces of the vertex $v_{last(\sigma)}$: these dependencies make the analysis of the algorithm quite difficult, and we let as an open problem to fully understand the experimental behavior of the algorithm as observed in Figure 3.1 (right).

3.2 Memory Performance

In order to represent the combinatorial structure of an arbitrary simplicial complex, one needs to mark all maximal simplices. Indeed, except in some special cases (like in flag complexes where all simplices are determined by the 1-skeleton of the complex), one cannot infer the existence of a simplex in a simplicial complex $\mathbf{K}$ from the existence of its faces in $\mathbf{K}$. Moreover, the number of maximal simplices of a k-dimensional simplicial complex is at least $\#\mathrm{Vert}\,\mathbf{K}/(k+1)$. In the case, considered in this paper, where the vertices are identified by their labels, a minimal representation of the maxi-

Data	$\#\mathcal{P}$	ρ	T_{g}	$\#E$	T_{Rips}	$\#\mathbf{K}$	T_{tot}	$T_{\text{tot}}/\#\mathbf{K}$
Bud	49,990	0.11	1.5	1,275,930	104.5	354,695,000	104.6	$3.0 \cdot 10^{-7}$
Bro	15,000	0.019	0.6	3083	36.5	116,743,000	37.1	$3.2 \cdot 10^{-7}$
Cy8	6,040	0.4	0.11	76,657	4.5	13,379,500	4.61	$3.4 \cdot 10^{-7}$
Kl	90,000	0.075	0.46	1,120,000	68.1	233,557,000	68.5	$2.9 \cdot 10^{-7}$
S4	50,000	0.28	2.2	1,422,490	95.1	275,126,000	97.3	$3.6 \cdot 10^{-7}$

Data	$\#L$	$\#W$	ρ	T_{nn}	T_{Wit^ρ}	$\#\mathbf{K}$	T_{tot}	$T_{\text{tot}}/\#\mathbf{K}$
Bud	10,000	49,990	0.12	1.	729.6	125,669,000	730.6	$0.58 \cdot 10^{-5}$
Bro	3,000	15,000	0.01	9.9	107.6	2,589,860	117.5	$4.5 \cdot 10^{-5}$
Cy8	800	6,040	0.23	0.38	161	997,344	161.2	$16 \cdot 10^{-5}$
Kl	10,000	90,000	0.11	2.2	572	109,094,000	574.2	$0.53 \cdot 10^{-5}$
S4	50,000	200,000	0.06	25.1	296.7	163,455,000	321.8	$0.20 \cdot 10^{-5}$

Figure 3.3: Data, timings in seconds and statistics for the construction of Rips complexes (top) and relaxed witness complexes (bottom). All complexes are constructed up to embedding dimension.

mal simplices would then require at least $\Omega(\log(\#\text{Vert}\,\mathbf{K}))$ bits per maximal simplex, for fixed k. The simplex tree uses $O(\log(\#\text{Vert}\,\mathbf{K}))$ memory bits per simplex *of any dimension*. The following experiment compares the memory performance of the simplex tree with the minimal representation described above, and with the representation of the 1-skeleton.

Figure 3.2 shows results for both Rips and relaxed witness complexes associated to $10,000$ points from **S4** and various values of the threshold ρ. The figure plots the total number of simplices $\#\mathbf{K}$, the number of maximal simplices $\#\text{m}\mathcal{F}$, the size of the 1-skeleton $\#\mathcal{G}$ and the construction times T_{Rips} and T_{Wit^ρ}.

As expected, the 1-skeleton is significantly smaller than the two other representations. However, as explained earlier, a representation of the graph of the simplicial complex is only well suited for flag complexes.

As shown on the figure, the total number of simplices and the number of maximal simplices remain close along the experiments. Moreover, we catch the topology of **S4** when $\rho \approx 0.4$ for the Rips complex and $\rho \approx 0.08$ for the relaxed witness complex. For these "good" values of the parameters, the total number of simplices is not much bigger than the number of maximal simplices. Specifically, the total number of simplices of the Rips complex is less than 2.3 times bigger than the number of maximal simplices, and the ratio is less than 2 for the relaxed witness complex.

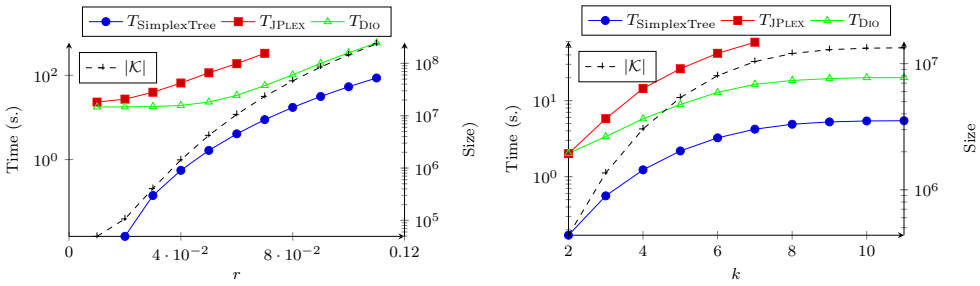


Figure 3.4: Statistics and timings for the construction of the Rips complex on (Left) **Bud** and (Right) **Cy8**.

3.3 Construction of Simplicial Complexes

Construction of Rips Complexes. We test our algorithm for the construction of Rips complexes. In Figure 3.4 we compare the performance of our algorithm with `JPlex` and with `Dionysus` along two directions.

In the first experiment, we build the Rips complex on 49,000 points from the dataset **Bud**. Our construction is at least 36 times faster than `JPlex` along the experiment, and several hundred times faster for small values of the parameter ρ. Moreover, `JPlex` is not able to handle the full dataset **Bud** nor big simplicial complexes due to memory allocation issues, whereas our method has no such problems. In our experiments, `JPlex` is not able to compute complexes of more than 23 million simplices ($r = 0.07$) while the simplex tree construction runs successfully until $r = 0.11$, resulting in a complex of 237 million simplices. Our construction is at least 7 times faster than `Dionysus` along the experiment, and several hundred times faster for small values of the parameter ρ.

In the second experiment, we construct the Rips complex on the 6040 points from **Cy8**, with threshold $r = 0.4$, for different dimensions k. Again, our method outperforms `JPlex`, by a factor 11 to 14. `JPlex` cannot compute complexes of dimension higher than 7 because it is limited by design to simplicial complexes of dimension smaller than 7. Our construction is 4 to 12 times faster than `Dionysus`.

The simplex tree and the expansion algorithm we have described are output sensitive. As shown by our experiments, the construction time using a simplex tree depends linearly on the size of the output complex. Indeed, when the Rips graphs are dense enough so that the time for the expansion dominates the full construction, we observe that the average construction time per simplex is constant and equal to 3.7×10^{-7} seconds for the first

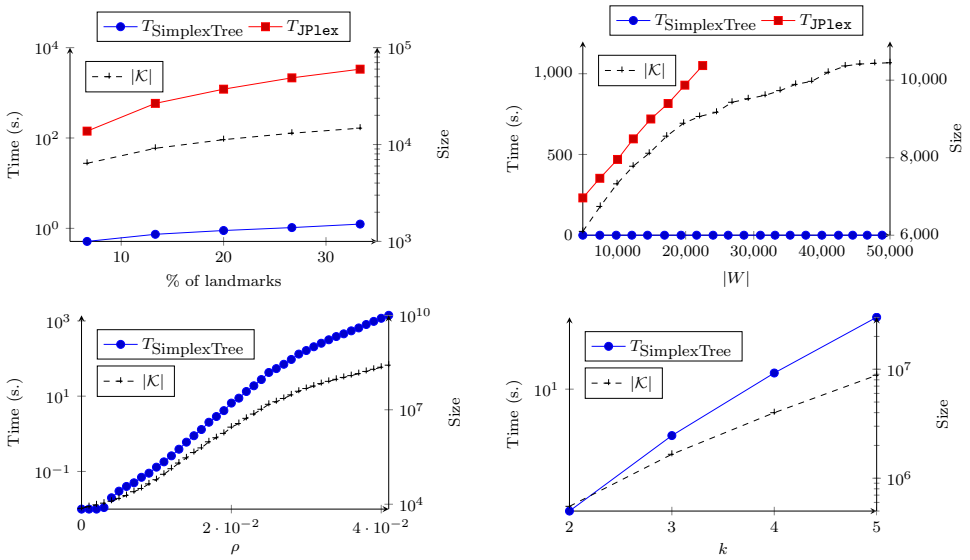


Figure 3.5: Statistics and timings for the construction of: (top) the witness complex and (bottom) the relaxed witness complex, on datasets (left) **Bro** and (right) **Kl**.

experiment, and 4.1×10^{-7} seconds for the second experiment (with standard errors 0.20% and 0.14% respectively). Additional timings are reported in Figure 3.6.

Construction of Witness Complexes. We test our algorithms for the construction of witness complexes and relaxed witness complexes. Figure 3.5 (top) shows the results of two experiments for the full construction of witness complexes. The first one compares the performance of the simplex tree algorithm and of `JPlex` on the dataset **Bro** consisting of $15,000$ points in dimension $\mathbb{R}^{25}$. Subsets of different size of landmarks are selected at random among the sample points. Our algorithm is from several hundred to several thousand times faster than `JPlex` (from small to big subsets of landmarks). Moreover, the simplex tree algorithm for the construction of the witness complex represent less than 1% of the total time spent, when more than 99% of the total time is spent computing the nearest neighbors of the witnesses.

In the second experiment, we construct the witness complex on $2,500$ landmarks from **Kl**, and sets of witnesses of different size. The simplex tree algorithm outperforms `JPlex`, being tens of thousands times faster. `JPlex` runs above the one hour time limit while the simplex tree algorithm stays

under 0.1 second all along the experiments. Moreover, the simplex tree algorithm spends only about 10% of the time constructing the witness complex, and 90% computing the nearest neighbors of the witnesses.

Finally we test the full construction of the relaxed witness complex. `JPlex` does not provide an implementation of the relaxed witness complex as defined in this paper; consequently, we were not able to compare the algorithms on the construction of the relaxed witness complex. We test our algorithms along two directions, as illustrated in Figure 3.5 (bottom). In the first experiment, we compute the 5-skeleton of the relaxed witness complex on **Bro**, with $15,000$ witnesses and $1,000$ landmarks selected randomly, for different values of the parameter ρ. In the second experiment, we construct the k-skeleton of the relaxed witness complex on **Kl** with $10,000$ landmarks, $100,000$ witnesses and fixed parameter $\rho = 0.07$, for various k. We are able to construct and store complexes of up to 260 million simplices. In both cases the construction time is linear in the size of the output complex, with a contruction time per simplex equal to 4.9×10^{-6} seconds in the first experiment, and 4.0×10^{-6} seconds in the second experiment (with standard errors 1.6% and 6.3% respectively).

Bud: ρ	0.08	0.085	0.090	0.095	0.100	0.105	0.110
T_{Rips}	19.4	26.5	35.8	46.7	60.5	77.7	98.7
$\#\mathbf{K}$	$69 \cdot 10^6$	$94 \cdot 10^6$	$127 \cdot 10^6$	$167 \cdot 10^6$	$217 \cdot 10^6$	$280 \cdot 10^6$	$355 \cdot 10^6$
Bro: ρ	0.184	0.186	0.188	0.190	0.192	0.194	0.196
T_{Rips}	15.3	18.1	28.2	34.5	40.8	56.2	81.1
$\#\mathbf{K}$	$52 \cdot 10^6$	$61 \cdot 10^6$	$95 \cdot 10^6$	$117 \cdot 10^6$	$138 \cdot 10^6$	$190 \cdot 10^6$	$275 \cdot 10^6$
Cy8: ρ	0.406	0.415	0.424	0.433	0.442	0.451	0.460
T_{Rips}	5.7	8.7	13.6	21.4	34.5	57.3	96.6
$\#\mathbf{K}$	$17 \cdot 10^6$	$27 \cdot 10^6$	$42 \cdot 10^6$	$67 \cdot 10^6$	$108 \cdot 10^6$	$180 \cdot 10^6$	$305 \cdot 10^6$
Kl: ρ	0.059	0.062	0.065	0.068	0.071	0.074	0.077
T_{Rips}	7.0	11.1	17.8	26.3	38.4	58.3	87.3
$\#\mathbf{K}$	$24 \cdot 10^6$	$38 \cdot 10^6$	$61 \cdot 10^6$	$90 \cdot 10^6$	$133 \cdot 10^6$	$204 \cdot 10^6$	$305 \cdot 10^6$
S4: ρ	0.22	0.23	0.24	0.25	0.26	0.27	0.28
T_{Rips}	2.7	4.7	8.5	15.4	28.0	50.9	93.7
$\#\mathbf{K}$	$7 \cdot 10^6$	$13 \cdot 10^6$	$23 \cdot 10^6$	$43 \cdot 10^6$	$79 \cdot 10^6$	$146 \cdot 10^6$	$271 \cdot 10^6$

Figure 3.6: Timings T_{Rips} for the construction of the Rips complex on the data sets and size of the simplicial complexes $\#\mathbf{K}$, for different values of the parameter ρ. On all these experiments, the time complexity is linear in the number of simplices. Specifically, the timing per simplex ranges between 2.79×10^{-7} and 3.47×10^{-7} seconds per simplex depending on the dataset, with standard error at most 0.40%.

Bud: ρ	0.06	0.07	0.08	0.09	0.10	0.11	0.12
T_{Wit^ρ}	18.3	36.9	71.1	135.8	249.1	440.2	758.6
$\#\mathbf{K}$	$7.8 \cdot 10^6$	$14 \cdot 10^6$	$23 \cdot 10^6$	$38 \cdot 10^6$	$58 \cdot 10^6$	$88 \cdot 10^6$	$130 \cdot 10^6$

Bro: ρ	0.0075	0.0080	0.0085	0.0090	0.0095	0.0100	0.0105
T_{Wit^ρ}	4.0	6.1	10.7	16.5	39.3	123.2	530.9
$\#\mathbf{K}$	$1.2 \cdot 10^6$	$1.5 \cdot 10^6$	$1.9 \cdot 10^6$	$2.2 \cdot 10^6$	$3.1 \cdot 10^6$	$4.6 \cdot 10^6$	$7.0 \cdot 10^6$

Cy8: ρ	0.194	0.200	0.206	0.212	0.218	0.224	0.230
T_{Wit^ρ}	18.7	33.1	130.2	273.0	512.9	37.2	1411.2
$\#\mathbf{K}$	$0.45 \cdot 10^6$	$0.66 \cdot 10^6$	$0.82 \cdot 10^6$	$1.1 \cdot 10^6$	$1.7 \cdot 10^6$	$2.3 \cdot 10^6$	$3.6 \cdot 10^6$

Kl: ρ	0.05	0.06	0.07	0.08	0.09	0.10	0.11
T_{Wit^ρ}	3.2	9.7	24.6	55.3	118.0	261.1	584.5
$\#\mathbf{K}$	$0.78 \cdot 10^6$	$2.2 \cdot 10^6$	$5.2 \cdot 10^6$	$11 \cdot 10^6$	$23 \cdot 10^6$	$49 \cdot 10^6$	$109 \cdot 10^6$

S4: ρ	0.03	0.035	0.040	0.045	0.050	0.055	0.060
T_{Wit^ρ}	7.6	14.1	26.4	48.9	89.2	164.6	297.3
$\#\mathbf{K}$	$2.8 \cdot 10^6$	$5.3 \cdot 10^6$	$11 \cdot 10^6$	$22 \cdot 10^6$	$43 \cdot 10^6$	$85 \cdot 10^6$	$161 \cdot 10^6$

Figure 3.7: Timings T_{Wit^ρ} for the construction of the relaxed witness complex on the data sets and size of the simplicial complexes $\#\mathbf{K}$, for different values of the parameter ρ. The timings per simplex vary, as the complexity of the construction algorithm depends also on the number of witnesses. It however ranges between $\approx 10^{-6}$ and $\approx 10^{-4}$ seconds per simplex depending on the number of witnesses compared to the number of simplices of the output.

Part II

Compressed Annotation Matrix for Persistent Cohomology

This Part is based on the following publications:

- Jean-Daniel Boissonnat, Tamal K. Dey and Clément Maria. The Compressed Annotation Matrix: An Efficient Data Structure for Computing Persistent Cohomology. *European Symposium on Algorithms 2013* [15]
- Jean-Daniel Boissonnat, Tamal K. Dey and Clément Maria. The Compressed Annotation Matrix: An Efficient Data Structure for Computing Persistent Cohomology. *Algorithmica 2015* [16]

Chapter 4

Homology Theory and Persistence

Homology theory generalizes the notion of connectivity to higher dimensions. A family of groups – one per dimension – is attached to a topological space. These groups are topological invariants, and intuitively capture the number of connected components of the space, its number of holes, its number of cavities and higher dimensional equivalents. Similarly to the fact that simplicial complexes are a natural generalization of graphs allowing more general incidence relations, homology groups are a natural generalization of graph connectivity to higher dimensional connectivity for triangulated shapes.

The homology groups are topological invariants in the sense that two homeomorphic shapes have isomorphic homology groups. However, for practical purpose like in topological inference, we do not deal with exact representations of spaces and comparing homology groups give an excessively rigid classification. Persistent homology has been introduced as a dynamic version of homology, where one studies the evolutation of the homology groups under a sequence of modifications of the topological space, called a *filtration*. A filtration induces a persistence module which we use as a topological invariant. We study the algebraic decompositions of the persistence modules and define a distance to compare them.

Finally, we present algorithms for computing persistent homology and persistent cohomology, and detail their implementation.

We assume in the following knowledge about general algebra. All necessary results are summarized in Appendix A.

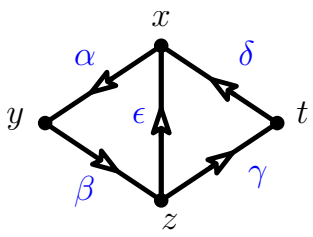


Figure 4.1: 1-dimension simplicial complex with vertices $\{x, y, z, t\}$ and edges $\{\alpha, \beta, \gamma, \delta\}$ with an arbitrary orientation.

Intuition. A hole is, by essence, something that is not material; it exists only because a domain encloses it. Mathematically, it is the same: nothing exists outside the topological space, which is only a set with a notion of proximity between its points. However, this is enough to draw closed loops and hence to enclose holes. Specifically, we count the number of holes in Figure 4.1 by enumerating the closed loops in the simplicial complex. With an additive notation and keeping track of the orientation of edges, the left loop is $\alpha + \beta + \epsilon$, the right loop is $\delta - \epsilon + \gamma$ and finally the external loop is $\alpha + \beta + \gamma + \delta$. We call them cycles in the following. Additionally, there is a linear dependence between these three cycles. Indeed, if we suppose we can commute the terms in each sum, we get

$$(\alpha + \beta + \epsilon) + (\delta - \epsilon + \gamma) = (\alpha + \beta + \gamma + \delta)$$

What we are constructing is the structure of an abelian group. We formalize this idea. Consider $\mathbf{C}_0$ to be the free abelian group generated by the vertices, *ie* $\mathbf{C}_0 = \langle x, y, z, t \rangle$ and $\mathbf{C}_1$ to be the free abelian group generated by the edges, *ie* $\mathbf{C}_1 = \langle \alpha, \beta, \gamma, \delta, \epsilon \rangle$. The cycles presented above belong to $\mathbf{C}_1$, together with other elements like $\alpha - 3\gamma$.

We characterize cycles algebraically by defining the following group homomorphism, called *boundary operator*

$$\partial_1 : \mathbf{C}_1 \to \mathbf{C}_0$$

which assigns to an oriented edge $[u, v]$ the oriented sum of its endpoints, *ie* $\partial_1([u, v]) = v - u$. For example, the boundary of α is $y - x$. The cycles above have a 0 boundary, because vertices cancel pairwise in the closed loop. For example

$$\partial_1(\alpha + \beta + \gamma + \delta) = (-x + y) + (-y + z) + (-z + t) + (-t + x) = 0$$

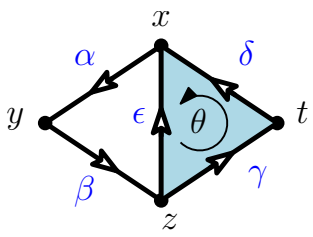


Figure 4.2: 1-dimension simplicial complex with vertices $\{x, y, z, t\}$ and edges $\{\alpha, \beta, \gamma, \delta\}$ with an arbitrary orientation.

The kernel of ∂_1 is called the group of 1-cycles and denoted $\mathbf{Z}_1$. In Figure 4.1, $\mathbf{Z}_1$ is a free abelian group generated by, for example, $\{\alpha + \beta + \epsilon, \gamma + \delta - \epsilon\}$. Note that the basis of $\mathbf{Z}_1$ contains as many elements as holes in the simplicial complex.

Consider now the simplicial complex of Figure 4.2, with an additional (oriented) triangle θ. We define similarly the free abelian group $\mathbf{C}_2$ generated by θ. The groups $\mathbf{C}_1$ and $\mathbf{Z}_1$ remain unchanged, but now the rank of $\mathbf{Z}_1 = \ker \partial_1$ is not anymore equal to the number of holes. We refine the notion by defining equivalence classes of cycles, where a class contains essentially all cycles that encircle the same hole. The boundary operator can be generalized naturally to dimension 2

$$\partial_2 : \mathbf{C}_2 \to \mathbf{C}_1$$

where the boundary of a triangle is the oriented sum of the edges of its boundary, for example $\partial_2 \theta = \gamma + \delta - \epsilon$.

Note now that the cycles $\alpha + \beta + \epsilon$ and $\alpha + \beta + \gamma + \delta$, which encircle the same hole after the insertion of the triangle θ, may be deformed one into another by crossing the triangle θ. Algebraically, their difference is equal to the boundary of one (or more) triangles

$$\partial_2 \theta = \gamma + \delta - \epsilon = (\alpha + \beta + \gamma + \delta) - (\alpha + \beta + \epsilon)$$

Denote by $\mathbf{B}_1$ the image of ∂_2, which we call the group of *boundaries*. We study the equivalence classes of the quotient $\mathbf{Z}_1/\mathbf{B}_1$ in order to capture the number of holes of a shape. Note that in this quotient, $\alpha + \beta + \epsilon$ and $\alpha + \beta + \gamma + \delta$ belongs to the same non-trivial equivalence class, and $\gamma + \delta - \epsilon$ belongs to the trivial class $[0]$. Now, the quotient $\mathbf{Z}_1/\mathbf{B}_1$ is generated by one element $\{[\alpha + \beta + \epsilon]\}$ and we count exactly one hole.

4.1 Simplicial Homology and Cohomology Theories

Let $\mathbf{K} = (V, S)$ be a simplicial complex and G be an abelian group, either equal to $\mathbb{Z}$ or to a field $\Bbbk$.

Homology Groups. Let $\sigma = \{v_0, \cdots, v_d\}$ be a simplex of $\mathbf{K}$. Define two orderings of its vertex set to be equivalent if they differ from one another by an even permutation. If $d > 1$, these orderings fall into two equivalence classes. Each of these classes are called an *orientation* of σ. If σ has dimension 0, there is only one orientation of σ. An *oriented* simplex $[v_0, \cdots, v_d]$ is a simplex $\{v_0, \cdots, v_d\}$ equipped with an orientation (for which the ordering $v_0, \cdots, v_d$ is a representative).

A *d-chain* on $\mathbf{K}$ is a function c from the set of oriented d-simplices of $\mathbf{K}$ to G, such that:

(i) $c(\sigma) = -c(\sigma')$ if σ and σ' are opposite orientations of the same simplex,

(ii) $c(\sigma) = 0$ for all but finitely many oriented d-simplices.

In our case, we consider only simplicial complexes with finitely many simplices, so condition (ii) is automatically satisfied.

We add d-chains by adding there values; the set of d-chains with $+$ forms a free abelian group $\mathbf{C}_d(\mathbf{K}, \mathrm{G})$ generated by the d-simplices of $\mathbf{K}$, $\mathbf{C}_d(\mathbf{K}, \mathrm{G}) = \langle\sigma\rangle_{\sigma\in\mathbf{K}^d}$. If $d < 0$ or $d > \dim \mathbf{K}$, we let $\mathbf{C}_d(\mathbf{K})$ denote the trivial group.

If σ is an oriented simplex, the *elementary chain* c corresponding to σ is the function defined as follows:

1. $c(\sigma) = 1$,
2. $c(\sigma') = -1$ if σ' is the opposite orientation of σ,
3. $c(\tau) = 0$ for all other oriented simplices.

By abuse of notation, we denote by σ the elementary chain associated to the oriented simplex σ and by $-\sigma$ the elementary chain associated to σ', the opposite orientation of σ. If we fix an orientation on the simplices of $\mathbf{K}$, we can consequently talk about *formal sum* of d-simplices to describe a d-chain, as any d-chain c can be written:

$$c = \sum_{\alpha} n_\alpha \sigma_\alpha$$

where the sum is over all oriented d-simplices of $\mathbf{K}$, and verifies $c(\sigma_\alpha) = n_\alpha$ and $c(\sigma'_\alpha) = -n_\alpha$, where σ_α belongs to the oriented d-simplices of $\mathbf{K}$ and σ'_α

is the opposite orientation. We call $c(\sigma_\alpha)$ the *coefficient* of simplex σ_α in the chain c, and we say that c *contains* σ_α if $c(\sigma_\alpha) \neq 0$.

We now define the homomorphism:

$$\partial_d : \mathbf{C}_d(\mathbf{K}, \mathrm{G}) \to \mathbf{C}_{d-1}(\mathbf{K}, \mathrm{G})$$

called the *boundary operator*. If $\sigma = [v_0, \cdots, v_d]$ is an oriented simplex with $d > 0$, we define:

$$\partial_d \sigma = \partial_d [v_0, \cdots, v_d] = \sum_{i=0}^{d} (-1)^i [v_0, \cdots, \widehat{v_i}, \cdots, v_d]$$

where $\widehat{v_i}$ means that v_i is deleted from the list. It is easy to verify that:

Lemma 4.1 (Fundamental lemma of homology)**.** $\partial_d \circ \partial_{d+1} = 0$.

Proof. It is enough to prove that $\partial_d \circ \partial_{d+1}$ is null on a simplex $\sigma = [v_0, \cdots, v_{d+1}]$:

$$\begin{aligned}
\partial_d \circ \partial_{d+1} [v_0, \cdots, v_{d+1}] &= \sum_{i=0}^{p+1} (-1)^i \partial_d [v_0, \cdots, \widehat{v_i}, \cdots, v_{d+1}] \\
&= \sum_{j<i} (-1)^i (-1)^j [v_0, \cdots, \widehat{v_j}, \cdots, \widehat{v_i}, \cdots, v_{d+1}] \\
&\quad + \sum_{j>i} (-1)^i (-1)^{j-1} [v_0, \cdots, \widehat{v_i}, \cdots, \widehat{v_j}, \cdots, v_{d+1}] \\
&= 0
\end{aligned}$$

because every term of the first sum appears in the second sum with opposite sign. □

The kernel of $\partial_d : \mathbf{C}_d(\mathbf{K}, \mathrm{G}) \to \mathbf{C}_{d-1}(\mathbf{K}, \mathrm{G})$ is called the group of *d-cycles* and denoted $\mathbf{Z}_d(\mathbf{K}, \mathrm{G})$. The image of $\partial_{d+1} : \mathbf{C}_{d+1}(\mathbf{K}, \mathrm{G}) \to \mathbf{C}_d(\mathbf{K}, \mathrm{G})$ is called the group of *d-boundaries* and denoted $\mathbf{B}_d(\mathbf{K}, \mathrm{G})$. By the preceding fundamental lemma, each boundary of a $(d+1)$-chain is a d-cycle, and $\mathbf{B}_d(\mathbf{K}, \mathrm{G}) \subseteq \mathbf{Z}_d(\mathbf{K}, \mathrm{G})$. We define

$$\mathbf{H}_d(\mathbf{K}, \mathrm{G}) = \mathbf{Z}_d(\mathbf{K}, \mathrm{G}) / \mathbf{B}_d(\mathbf{K}, \mathrm{G})$$

to be the d^{th} *homology group* of $\mathbf{K}$ with G coefficients.

Homology Coefficients and Decomposition. We have considered homology with coefficients in G, where G is either $\mathbb{Z}$ or a field $\Bbbk$. With integral coefficients, the abelian group of d-chains $\mathbf{C}_d(\mathbf{K}, \mathbb{Z})$ is, by construction, free and admits as a basis the set of d-simplices of $\mathbf{K}$. Being subgroups of a free group, the group of d-cycles $\mathbf{Z}(\mathbf{K}, \mathbb{Z})$ and the group of d-boundaries $\mathbf{B}(\mathbf{K}, \mathbb{Z})$ are also free abelian groups. Consequently, the homology groups are all finitely generated, as quotients of these groups. By virtue of the *structure theorem of finitely generated abelian groups* (Theorem A.1), every homology group with integer coefficients admits a primary decomposition:

$$\mathbf{H}_d(\mathbf{K}, \mathbb{Z}) \cong \mathbb{Z}^{\beta_d(\mathbb{Z})} \bigoplus_{q \text{ prime}} \left(\mathbb{Z}_{q^{k_1}} \oplus \cdots \oplus \mathbb{Z}_{q^{k_{t(d,q)}}} \right)$$

for uniquely defined integers $\beta_d(\mathbb{Z})$, and $k_i > 0$ and $t(d, q) \geq 0$ over all prime numbers q. The summand $\mathbb{Z}^{\beta_d(\mathbb{Z})}$ of the decomposition describes the free part of $\mathbf{H}_d(\mathbf{K}, \mathbb{Z})$ and the integer $\beta_d(\mathbb{Z})$ is the d^{th} *Betti number* of $\mathbf{K}$ in integral homology. The summand $\bigoplus_{q \text{ prime}} \left(\mathbb{Z}_{q^{k_1}} \oplus \cdots \oplus \mathbb{Z}_{q^{k_{t(d,q)}}} \right)$ of the decomposition, made of finite cyclic groups, is the *torsion* part of $\mathbf{H}_d(\mathbf{K}, \mathbb{Z})$ and, if $t(d, q) > 0$, the integers $q^{k_1}, \cdots, q^{k_{t(d,q)}}$ of the primary decomposition are the *torsion coefficients* of $\mathbf{H}_d(\mathbf{K}, \mathbb{Z})$.

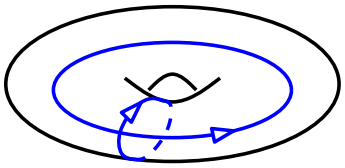
When homology is defined with coefficients in a field $\Bbbk$, the groups $\mathbf{C}_d(\mathbf{K}, \Bbbk)$, $\mathbf{Z}_d(\mathbf{K}, \Bbbk)$, $\mathbf{B}_d(\mathbf{K}, \Bbbk)$ and $\mathbf{H}_d(\mathbf{K}, \Bbbk)$ become $\Bbbk$-vector spaces. As a consequence, the homology group $\mathbf{H}_d(\mathbf{K}, \Bbbk)$ admits a decomposition:

$$\mathbf{H}_d(\mathbf{K}, \Bbbk) \cong \Bbbk^{\beta_d(\Bbbk)}$$

for a unique integer $\beta_d(\Bbbk)$ – the dimension of the vector space – also called the d^{th} *Betti number* of $\mathbf{K}$ in homology with $\Bbbk$ coefficients. Note that there is no notion of torsion anymore.

Integral homology is strictly stronger than homology with coefficients in a field, in the sense that one can deduce the decomposition of the vector space $\mathbf{H}_d(\mathbf{K}, \Bbbk)$ for every d, knowing the primary decomposition if the homology groups $\mathbf{H}_d(\mathbf{K}, \mathbb{Z})$, for every d. This is formalized by the *universal coefficient theorem* that we describe in Chapter 7. However, being vector spaces, homology groups with $\Bbbk$ coefficients offer better computational properties as well as stronger algebraic decomposition theorems when defining persistence in Section 4.2. When there is no ambiguity on the coefficients we use or when coefficients do not play a role, we omit the coefficient field from the notations.

To come back to the intuitive behavior of homology groups, the Betti numbers β_d may often be related as the number of d-dimensional holes. For



$$\begin{cases} \mathbf{H}_0 \cong \mathbb{Z} \\ \mathbf{H}_1 \cong \mathbb{Z} \oplus \mathbb{Z} \\ \mathbf{H}_2 \cong \mathbb{Z} \end{cases}$$

Figure 4.3: Homology groups of the torus.

example, it is always true that $\mathbf{H}_0$ is free, even with $\mathbb{Z}$ coefficients, and that β_0 is equal to the number of connected components. In Figure 4.1, $\beta_0 = 1$, $\beta_1 = 2$ and there is no torsion, as expected. A more interesting example, the torus in Figure 4.3, has 1 hole that induces 2 independent non-contractible loops. When homology groups have torsion subgroups, the Betti numbers differ depending on the coefficients. We discuss the question of torsion and its influence on homology groups depending on the coefficients used in Chapter 7.

Cohomology Groups. Cohomology is the theory dual to the theory of homology, in a sense that we make precise below. Its definition is highly algebraic in nature. In our setting, cohomology groups capture a topological information equivalent to the one of homology groups but offer new algorithmic possibilities as explained in Chapter 5.

For simplicity, we consider in the following homology and cohomology with coefficients in a field $\Bbbk$. Under this assumption, cohomology groups are dual to homology groups in the sense of vector spaces and are consequently isomorphic. For a $\Bbbk$-vector space U, we define $\mathrm{Hom}(U)$ to be the vector space of all homomorphisms of U into $\Bbbk$. The group of *d-cochains*, with $\Bbbk$ coefficients, of a simplicial complex $\mathbf{K}$ is the group

$$\mathbf{C}^d(\mathbf{K}, \Bbbk) = \mathrm{Hom}(\mathbf{C}_d(\mathbf{K}), \Bbbk)$$

This is the $\Bbbk$-vector space dual to $\mathbf{C}_d(\mathbf{K})$. For a simplex $\sigma \in \mathbf{K}$, we define the *elementary cochain* σ^* that is the dual to the chain σ, *ie* the homomorphism of $\mathrm{Hom}(\mathbf{C}_d(\mathbf{K}), \Bbbk)$:

$$\begin{cases} \sigma^*(\sigma) &= 1 \\ \sigma^*(\tau) &= 0 \quad \text{for } \tau \neq \sigma \end{cases}$$

The *coboundary operator* $\delta_d : \mathrm{Hom}(\mathbf{C}_d(\mathbf{K}), \Bbbk) \to \mathrm{Hom}(\mathbf{C}_{d+1}(\mathbf{K}), \Bbbk)$ is defined to be the dual of the boundary operator ∂_d, *ie* for any d-cochain $\varphi \in \mathbf{C}^d(\mathbf{K})$ we have $\delta_d(\varphi) = \varphi \circ \partial_{d+1} \in \mathrm{Hom}(\mathbf{C}_{d+1}(\mathbf{K}), \Bbbk)$. Note that δ raises

dimension by 1. We define $\mathbf{Z}^d(\mathbf{K}, \mathrm{G})$ to be the kernel of δ_d, $\mathbf{B}^{d+1}(\mathbf{K}, \mathrm{G})$ to be the image of δ_d and, noting that $\delta_{d+1} \circ \delta_d = 0$,

$$\mathbf{H}^d(\mathbf{K}, \mathrm{G}) = \mathbf{Z}^d(\mathbf{K}, \mathrm{G}) / \mathbf{B}^d(\mathbf{K}, \mathrm{G})$$

These groups are called respectively the group of *d-cocycles*, the group of *d-coboundaries* and the d^{th} *cohomology group* of $\mathbf{K}$ with $\Bbbk$ coefficients.

Invariance of Homology Groups. Let $f : \mathbf{K} \to \mathbf{L}$ be a simplicial map. It induces a homomorphism between chain groups:

$$f_\# : \mathbf{C}_d(\mathbf{K}) \to \mathbf{C}_d(\mathbf{L})$$

defined as follows. For every oriented simplex $\sigma = [v_0, \cdots, v_d]$ of $\mathbf{K}$

$$f_\#([v_0, \cdots, v_d]) = \begin{cases} [f(v_0), \cdots, f(v_d)] & \text{if all } f(v_0), \cdots, f(v_d) \text{ are distinct,} \\ 0 & \text{otherwise.} \end{cases}$$

This map is well-defined. Exchanging two vertices in the expression of $[v_0, \cdots, v_d]$ changes the sign of the right side of the equation.

A simple computation shows that the induced map $f_\# : \mathbf{C}_d(\mathbf{K}) \to \mathbf{C}_d(\mathbf{L})$ commutes with the boundary operator, such that $\partial_d^{\mathbf{L}} \circ f_\# = f_\# \circ \partial_{d+1}^{\mathbf{K}}$. As a consequence, the homomorphism $f_\#$ carries cycles to cycles, since the equation $\partial c = 0$ for $c \in \mathbf{Z}(\mathbf{K})$ implies $\partial f_\#(c) = f_\#(\partial c) = 0$. Moreover $f_\#$ carries boundaries to boundaries, since the equation $c = \partial d$ implies that $f_\#(c) = f_\#(\partial d) = \partial f_\#(d)$. Thus, $f_\#$ induces a homomorphism $f_* : \mathbf{H}_d(\mathbf{K}) \to \mathbf{H}_d(\mathbf{L})$ at the homology level.

The following theorem shows that continuous maps between underlying domains also induce homomorphisms at the homology level:

Theorem 4.1 (Induced Homomorphism)**.** *Let* $\mathbf{K}$ *and* $\mathbf{L}$ *be two simplicial complexes and* $h : |\mathbf{K}| \to |\mathbf{L}|$ *be a continuous function between their respective underlying domains. There exists an homomorphism induced by* h *at the homology level,*

$$h_* : \mathbf{H}_d(\mathbf{K}) \to \mathbf{H}_d(\mathbf{L})$$

This is the homology analog of the simplicial approximation Theorem 1.2. Finally, the following theorem sets the homology groups as topological invariants.

Theorem 4.2 (Invariance of Homology Groups)**.** *If* $h : |\mathbf{K}| \to |\mathbf{L}|$ *is a homeomorphism, then* $h_* : \mathbf{H}_d(\mathbf{K}) \to \mathbf{H}_d(\mathbf{L})$ *is an isomorphism.*

As a direct consequence, by composing homeomorphisms, homology groups are well-defined on triangulable topological spaces. Moreover, two triangulable topological spaces $\mathbb{X}$ and $\mathbb{Y}$ that are homeomorphic have isomorphic homology groups. Homology groups are thus topological invariants.

4.2 Persistent Homology

As shown in the previous section, homology groups are invariants for a topological space. They are meant to differenciate topological spaces that are not homeomorphic. They are consequently sensitive to discontinous transformations of a space. When dealing with practical problems in computational topology, we are unlikely to study a homeomorphic representation of a topological space, but rather a coarse approximation. In topological data analysis for example, we reconstruct a topological space from a point sample. To do so, we use the types of simplicial complexes – Rips complex, witness complex – introduced in Chapter 1. By nature, point clouds are discontinuous and the slightest perturbation of the points or variation of scale may affect the homology groups of the approximating complex; see Figure 4.4. In order to

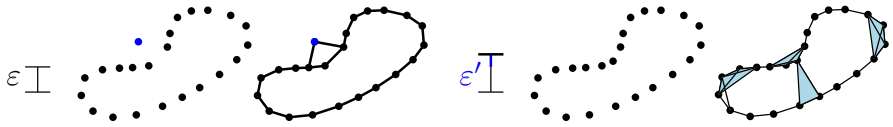


Figure 4.4: Approximations from a perturbed point cloud with the wrong homology.

provide a robust topological invariant in such a setting, persistent homology has been introduced as a multi-scale theory of homology.

Specifically, persistent homology formalizes the idea of topological invariant for a monotonic sequence of topological spaces connected by continuous maps. In the discrete setting, consider the sequence of simplicial complexes connected by simplicial maps:

$$\mathbb{K} := \quad \mathbf{K}_0 \xrightarrow{f_1} \mathbf{K}_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} \mathbf{K}_{n-1} \xrightarrow{f_n} \mathbf{K}_n$$

We call such sequence a *filtration*. We see that a filtration function, defined in Chapter 1, on a simplicial complex induces such sequence $\mathbb{K}$ of simplicial complexes, where all maps are inclusions. For every dimension d, the sequence of simplicial complexes induces a sequence a homology groups con-

nected by induced homomorphisms:

$$\mathbb{H}\text{om} := \quad \mathbf{H}_d(\mathbf{K}_0) \xrightarrow{f_{1*}} \mathbf{H}_d(\mathbf{K}_1) \xrightarrow{f_{2*}} \cdots \xrightarrow{f_{n-1*}} \mathbf{H}_d(\mathbf{K}_{n-1}) \xrightarrow{f_{n*}} \mathbf{H}_d(\mathbf{K}_n)$$

This sequence of homology groups connected by homomorphisms is called a *persistence module*. It is the object of study of persistent homology.

Persistence and Living Homological Features. Unless mentioned otherwise, all homology groups in the following have coefficients in a field $\Bbbk$. We focus on the case where the simplicial maps f_i in the filtration $\mathbb{K}$ are all elementary inclusion, *ie* $\mathbf{K}_{i+1} = \mathbf{K}_i \cup \{\tau_{i+1}\}$ where τ_{i+1} is a simplex not in $\mathbf{K}_i$. For simplicity, we denote such a map by τ_{i+1} and the induced map at homology level by τ_{i+1*}.

First, we study the effect of an elementary inclusion $\mathbf{K} \xrightarrow{\tau} \mathbf{L}$ on the homology groups. Suppose τ is a d-simplex. We prove,

Lemma 4.2. *If there exists a cycle $c \in \mathbf{C}_d(\mathbf{L})$ with $c(\tau) \neq 0$, then* $\dim \mathbf{H}_d(\mathbf{L}) = \dim \mathbf{H}_d(\mathbf{K}) + 1$ *and the other homology groups do not change. We say that τ is a* creator. *Otherwise,* $\dim \mathbf{H}_{d-1}(\mathbf{L}) = \dim \mathbf{H}_{d-1}(\mathbf{K}) - 1$ *and other homology groups do not change. We say that τ is a* destructor.

Proof. We use the same notation for a chain in $\mathbf{K}$ and its image in $\mathbf{L}$, as they are equal to the same formal sum of simplices.
Case 1: If τ is contained in a cycle c, we prove that c cannot be homologous to any cycle in $\mathbf{C}_d(\mathbf{K})$. Suppose instead that there is such a cycle $c' \in \mathbf{C}_d(\mathbf{K})$ such that $c - c' = \partial a$, for a chain $a \in \mathbf{C}_{d+1}(\mathbf{L})$. Denote by $\alpha \neq 0$ the coefficient of τ in c. Because c' is a cycle in $\mathbf{K}$, it does not contain the simplex τ, *ie* $c'(\tau) = 0$ in $\mathbf{L}$. Consequently, $(\partial a)(\tau) = \alpha \neq 0$. This is a contradiction because $\mathbf{L}$ does not contain any proper coface of τ. Hence, $[c]$ is independent from other homology classes in $\mathbf{H}_d(\mathbf{L})$ and $\dim \mathbf{H}_d(\mathbf{L}) \geq \dim \mathbf{H}_d(\mathbf{K}) + 1$. The inequality $\dim \mathbf{H}_d(\mathbf{L}) \leq \dim \mathbf{H}_d(\mathbf{K}) + 1$ is a direct consequence from the fact that $\mathbf{H}_d$ is the quotient of $\mathbf{Z}_d$ over $\mathbf{B}_d$, and the addition of a simplex increases the dimension of $\mathbf{Z}_d$ by at most 1. Note that in this case $[\partial\tau] = 0$ in $\mathbf{H}_{d-1}(\mathbf{K})$. Indeed, $\partial c = 0 = \partial\tau + \partial a$ for some chain $a \in \mathbf{C}_d(\mathbf{K})$. Consequently $\partial\tau$ is a boundary in $\mathbf{K}$.
Case 2: Suppose τ is contained in no cycle. Then the chain $\partial\tau$ is a non-trivial cycle in $\mathbf{K}$ that become a boundary in $\mathbf{L}$. It is a cycle because $\partial \circ \partial = 0$. It is obviously a boundary in $\mathbf{L}$ because $\tau \in \mathbf{C}_d(\mathbf{L})$. We prove it is not a boundary in $\mathbf{K}$. Suppose instead that there is a chain $a \in \mathbf{C}_d(\mathbf{K})$ such that $\partial\tau = \partial a$. Thus, $\tau - a$ is a cycle of $\mathbf{C}_d(\mathbf{L})$ containing τ which is a

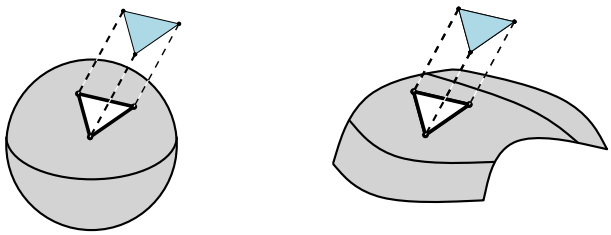


Figure 4.5: Elementary inclusion with addition of a triangle. Left: creation of a homological feature in dimension 2. Right: destruction of a homological feature in dimension 1.

contradiction. Using a similar argument as in Case 1, we conclude that $\dim \mathbf{H}_{d-1}(\mathbf{L}) = \dim \mathbf{H}_{d-1}(\mathbf{K}) - 1$.

The fact that the homology groups in other dimensions do not change is direct. □

We refer to Figure 4.5 for an illustration of this lemma. Assuming we can decide if a simplex is a creator or a destructor, we can deduce from this lemma an incremental algorithm to compute the Betti numbers of all complexes $\mathbf{K}_i$ of the filtration $\mathbb{K}$, by adding the simplices τ_i in turn and update the Betti numbers under elementary inclusions.

Persistent homology furnishes a stronger information on the evolution of the topology in the filtration. It also gives a description of which homological feature is created or destructed, and furnishes a pairing of the birth and death of each feature. From this pairing we can construct a topological invariant for filtrations.

In order to define an invariant of algebraic nature for the persistence module, we need to classify persistence modules up to isomorphism. Every sequence of homology groups, with field coefficients, $\mathbb{H}\mathrm{om}$ decomposes into interval submodules $\mathbb{I}[b; d]$:

$$\mathbb{H}\mathrm{om} \cong \bigoplus_i \mathbb{I}[b_i; d_i] \tag{4.1}$$

where an interval submodule $\mathbb{I}[b_i; d_i]$ is a persistence module equal to:

$$\underbrace{0 \xrightarrow{0} \cdots \xrightarrow{0} 0}_{[1;b_i-1]} \xrightarrow{0} \underbrace{\Bbbk \xrightarrow{\mathbb{1}} \cdots \xrightarrow{\mathbb{1}} \Bbbk}_{[b_i;d_i]} \xrightarrow{0} \underbrace{0 \xrightarrow{0} \cdots \xrightarrow{0} 0}_{[d_i+1;n]}$$

and we call b_i and d_i respectively the *birth* and the *death* of $\mathbb{I}[b_i; d_i]$. This decomposition is unique up to reordering of the terms and allows a full

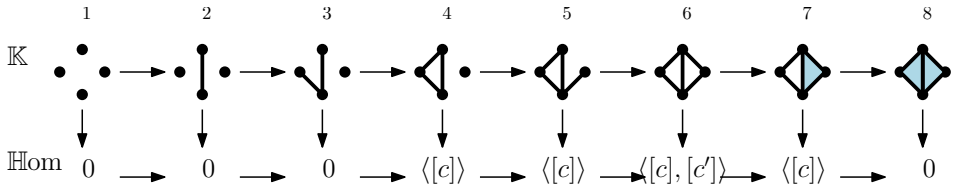


Figure 4.6: Filtration $\mathbb{K}$ and corresponding persistence module $\mathbb{H}\text{om}$. The basis with homology classes $[c]$ and $[c']$ is compatible with the interval decomposition of $\mathbb{H}\text{om} \cong \mathbb{I}[4;7] \oplus \mathbb{I}[6;6]$, in the sense that interval $[4;7]$ corresponds to the lifespan of $[c]$, catching the left hole, and interval $[6;6]$ corresponds to the lifespan of $[c']$, catching the right hole.

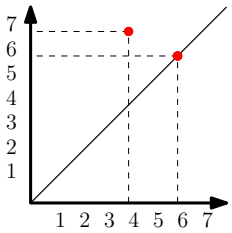
classification of persistence modules up to isomorphism. Note that such decomposition does not exist in general when homology is defined with integer coefficients. We partly overcome this difficulty in Part III where we extract some information about torsion in the context of persistence.

The success of persistent homology relies somehow on the intuition one can have of this decomposition. Consider the example in Figure 4.6. The persistence module decomposes into two interval modules that represent the lifespan of the classes $[c]$ and $[c']$, which in turns are linked to the *birth* and *death* of 1-dimensional holes. The interval modules spanned by $[c]$ and $[c']$ also agree with Lemma 4.2, in the sense that c and c' are the cycles (the ones encircling the holes as in Figure 4.1) containing the new simplex when they are created.

This intuition generalizes to the case of persistent homology, as we see in algorithmic Section 4.3 through the concept of *encoding*. We prove that the approach is correct in Chapter 10.

Persistence Diagram and Stability. In standard topology, two topological spaces are equivalent if they are homeomorphic, which implies that they have isomorphic homology groups. The isomorphism of invariants is too rigid in persistent homology, where one does not usually deal with an exact representation of a space. Persistence modules are instead compared using a more flexible notion of closeness expressed by a distance between *persistence diagrams*.

The direct sum decomposition of the persistence module is visualized using a *persistence diagram*. For a persistence module $\mathbb{H}\text{om}$ of length n with decomposition $\mathbb{H}\text{om} \cong \bigoplus_i \mathbb{I}[b_i; d_i]$, $b_i, d_i \in \{1, \cdots, n\}$ for all i, the *index persistence diagram* is a plot in the plane with a point (b_i, d_i) for every interval module $\mathbb{I}[b_i; d_i]$ of the decomposition. For convenience, the diagonal



$\mathbb{H}\mathrm{om} \cong \mathbb{I}[4;7] \oplus \mathbb{I}[6;6]$

Figure 4.7: Index persistence diagram.

$x = y$, with infinitely many points, is added; see Figure 4.7.

When a filtration function $f : \mathbf{K} \to \mathbb{R}$ is defined on the simplicial complex $\mathbf{K}$, every index $j \in \{1, \cdots, n\}$ corresponds to the insertion of a simplex τ_j. In this context, we call *persistence diagram* the plot in the plane of the points $(f(\tau_{b_i}), f(\tau_{d_i}))$. Another convenient representation is the *(index) persistence barcode*, where every interval module $\mathbb{I}[b_i; d_i]$ is represented by an horizontal bar with endpoints at abscissa $f(\tau_{b_i})$ and $f(\tau_{d_i})$ (or b_i and d_i). See Figure 4.8 for a practical exemple on a topological inference problem. The long bar in the barcode corresponds to a meaningful homological feature – *ie* one appearing in the topological space we approximate – when the short bars are *topological noise*. In the persistence diagram, the topological noise is represented by the points close to the diagonal.

The similarity between persistence modules is measured by the *bottleneck distance* between their persistence diagrams. Let X and Y be the sets of points, including the diagonal, of two persistence diagrams. We consider bijections $\mu : X \to Y$ between these sets of points. A bijection $X \to Y$ always exists thanks to the addition of infinitely many points on the diagonals. For two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ in the plane, let $\|x-y\|_\infty$ denote the L_∞-distance between them, *ie* $\|x-y\|_\infty = \max\{|x_1-y_1|, |x_2-y_2|\}$. We attach a value to every bijection $\mu : X \to Y$ which is the supremum over all pairs of points $(x, \mu(x))$, with $x \in X$, of the distance $\|x - \mu(x)\|_\infty$. The *bottleneck distance* $\mathsf{d}_\mathcal{B}(X, Y)$ between two persistence diagrams is the infimum over all bijection $\mu(x)$ of the value attached to μ, *ie*

$$\mathsf{d}_\mathcal{B}(X, Y) = \inf_{\mu : X \to Y} \sup_{x \in X} \|x - \mu(x)\|_\infty$$

One of the main result of the theory of persistent homology is the stability of the bottleneck distance. Specifically,

Theorem 4.3 (Stability Theorem)**.** *Let* $\mathbf{K}$ *be a simplicial complex and* $f, g : \mathbf{K} \to \mathbb{R}$ *two filtration functions on* $\mathbf{K}$*. For a dimension* d*, let* X *and* Y

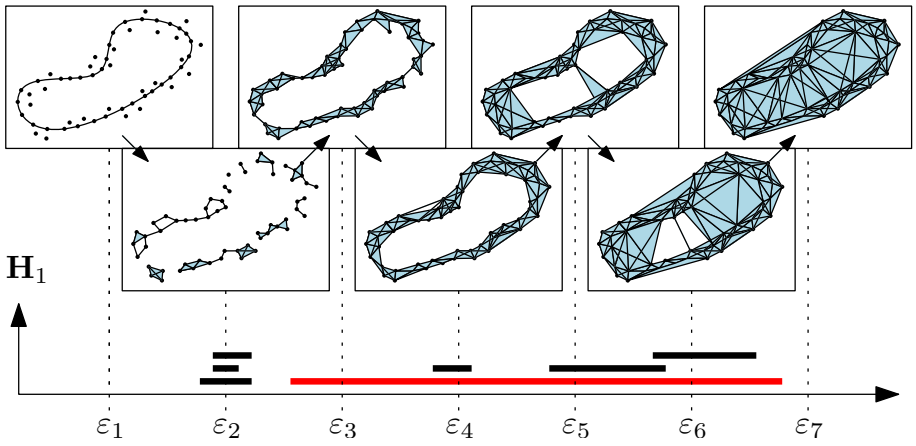


Figure 4.8: Persistent homology for topological inference.

be the points of the persistence diagrams of the d-homology of f and of g respectively. The bottleneck distance between X and Y is bounded from above by the L_∞-distance between f and g, ie

$$\mathsf{d}_{\mathcal{B}}(X,Y) \leq \|f-g\|_\infty \ , \quad \textit{with} \quad \|f-g\|_\infty = \sup_{\sigma \in \mathbf{K}} |f(\sigma)-g(\sigma)|$$

As a consequence, having close filtrations, in the sense of the Stability Theorem 4.3, implies having close persistence diagrams *wrt* bottleneck distance. Hence, we have defined a notion of topological similarity for filtrations.

Proximity of Diagrams for Čech and Rips Complexes. As an application of the Stability Theorem 4.3, we prove that in topological inference the long bars in the persistence barcodes of the Rips complex give the homology of the unknown space.

We call *quadrant* (b,d) the open area of the plane containing all points $(x,y) \in \mathbb{R}^2$ with $x < b$ and $y > d$. In a persistence diagram, the quadrant (b,d) contains all points corresponding to homological features which are born before time b and die after time d. Let $\mathcal{P}$ be a point cloud in $\mathbb{R}^D$ sampling an unknown topological space $\mathbb{X}$ such that the sampling satisfies the conditions of the Topological Reconstruction Theorem 1.4. The Topological Reconstruction Theorem implies that the Čech complex $\mathcal{C}^\rho(\mathcal{P})$ has homology groups isomorphic to the ones of $\mathbb{X}$ for every ρ in some interval $(\gamma_\varepsilon; \delta_\varepsilon)$.

Consequently, all points in the quadrant $(\gamma_\varepsilon, \delta_\varepsilon)$ of its persistence diagram correspond to homological features of $\mathbb{X}$.

Recall now Lemma 1.1: for every ρ, we have the inclusions $\mathcal{C}^\rho(\mathcal{P}) \subseteq \mathcal{R}^\rho(\mathcal{P}) \subseteq \mathcal{C}^{\vartheta_D \cdot \rho}(\mathcal{P})$, for a constant ϑ_D depending only on the dimension D. Suppose *wlog* that $\gamma_\varepsilon = \rho_0$ and $\delta_\varepsilon = (\vartheta_D)^k \rho_0$. We extend the sequence of inclusions:

$$\mathcal{C}^{\rho_0}(\mathcal{P}) \subseteq \mathcal{R}^{\rho_0}(\mathcal{P}) \subseteq \cdots \subseteq \mathcal{R}^{(\vartheta_D)^{k-1} \times \rho_0}(\mathcal{P}) \subseteq \mathcal{C}^{(\vartheta_D)^k \times \rho_0}(\mathcal{P})$$

The parameter ρ defines two filtration functions f and g, one for the Čech and one for the Rips complexes, on the final simplicial complex $\mathcal{C}^{(\vartheta_D)^k \times \rho_0}(\mathcal{P})$. In order to make the progression of the parameter ρ arithmetic instead of geometric, we take the log of f and g, which gives again two filtration functions $\log f$ and $\log g$ by monotonicity of log. The L_∞-distance between these two functions is at most $\| \log f - \log g \|_\infty = \log \vartheta_D$. As a consequence, the bottleneck distance of the persistence diagrams of the Čech and Rips log-filtrations is bounded by $\log \vartheta_D$. Consequently, if $\mathcal{P}$ is dense enough in $\mathbb{X}$ so that the quadrant $(\log \gamma_\varepsilon, \log \delta_\varepsilon)$ is far enough from the diagonal and the topological noise, then the quadrant $(\log \gamma_\varepsilon - \log \vartheta_D, \log \delta_\varepsilon - \log \vartheta_D)$ of the persistence diagram of the log-filtration of the Rips complex contains the points corresponding to the homological features of $\mathbb{X}$.

Note that there are better and more direct proofs of the topological inference properties of the Rips complex, mentioned at the end of this chapter.

4.3 Computational Aspects of Persistence

We now present algorithms to compute persistent homology and persistent cohomology. Because homology and cohomology are considered with coefficients in a field $\Bbbk$, they are dual vector spaces and the interval decomposition of the two corresponding persistence modules are isomorphic. The algorithm returns the same result.

For simplicity, we denote by $\mathbf{C}_*(\mathbf{K})$ the external direct sum of the chains groups for all dimensions, *ie* $\mathbf{C}_*(\mathbf{K}) = \bigoplus_d \mathbf{C}_d(\mathbf{K})$. We define similarly $\mathbf{Z}_*(\mathbf{K})$, $\mathbf{B}_*(\mathbf{K})$ and $\mathbf{H}_*(\mathbf{K})$. This way, we do not have to precise the dimension when computing the full persistent homology of a filtration. Similarly, we use the notations $\mathbf{C}^*(\mathbf{K})$, $\mathbf{Z}^*(\mathbf{K})$, $\mathbf{B}^*(\mathbf{K})$ and $\mathbf{H}^*(\mathbf{K})$ for the direct sums in all dimensions of the cochain, cocycle, coboundary and cohomology groups, respectively. We prove the validity of the persistent homology algorithm in Chapter 10, after introducing a uniform presentation of persistence and zigzag persistence in the framework of quiver theory.

Persistent Homology Algorithm. In order to represent the vector space $\mathbf{H}(\mathbf{K}_i)$, we represent a basis that is *compatible* with the decomposition of the persistence module. We formalize the notion of *compatible basis* in Chapter 10 where we prove the validity of the algorithm. The elements of $\mathbf{H}(\mathbf{K}_i)$ are equivalence classes of cycles that we cannot represent directly. Instead, we maintain explicitly three families of chains.

For a filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \mathbf{K}_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \mathbf{K}_{n-1} \xrightarrow{\tau_n} \mathbf{K}_n$, there exist chains $\widehat{\tau}_1, \cdots, \widehat{\tau}_n \in \mathbf{C}_*(\mathbf{K}_n)$, a partition of the indices $\{1, \cdots, n\} = F \sqcup G \sqcup H$, and a bijective pairing $G \leftrightarrow H$, denoted by $\mathcal{P} \subseteq G \times H$, satisfying the following conditions:

1. for all i, the leading term of $\widehat{\tau}_i$ is τ_i, meaning that $\widehat{\tau}_i = \alpha_1\tau_1 + \cdots + \alpha_i\tau_i$, with $\alpha_i \neq 0$,
2. for all $f \in F$, $\partial\widehat{\tau}_f = 0$,
3. for all pairs $(g, h) \in \mathcal{P}$, $\partial\widehat{\tau}_h = \widehat{\tau}_g$ and hence $\partial\widehat{\tau}_g = 0$.

We call such a partitioned set of chains an *encoding of the persistence module* . Condition 1. is equivalent to the fact that $\mathbf{C}_*(\mathbf{K}_i) = \langle \widehat{\tau}_1, \cdots, \widehat{\tau}_i \rangle$. For $f \in F$, $\widehat{\tau}_f$ is a cycle created at index f whose homology class is non-zero in $\mathbf{K}_n$. For $g \in G$, paired with $h \in H$, $\widehat{\tau}_g$ is a cycle created at index g and whose homology class is non-zero from index g up to index $h-1$, after which it becomes the boundary of the chain $\widehat{\tau}_h$. We prove in Chapter 10 that these cycles span submodules of $\mathbb{H}\mathrm{om}$ that are in direct sum and are isomorphic to interval modules. Hence, one can read directly the persistent pairs from this encoding. In other words, an encoding of a persistence module encodes completely the interval decomposition defined in Equation 4.1.

Computing the persistent homology of a filtration consists in maintaining an encoding of the persistence module under maps induced by elementary inclusions. Suppose we have an encoding $\{\widehat{\tau}_j\}$, with indices j in $F \sqcup G \sqcup H = \{1, \cdots, i\}$ and a paring $\mathcal{P}$, of the persistence module corresponding to the filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_i} \mathbf{K}_i$ and consider the map $\mathbf{K}_i \xrightarrow{\tau_{i+1}} \mathbf{K}_{i+1}$.

Note first that $\mathbf{B}_*(\mathbf{K}_i)$ is generated by the cochains $\{\widehat{\tau}_g\}_{g \in G}$ and $\mathbf{Z}_*(\mathbf{K}_i)$ is generated by the cochains $\{\tau_j\}_{j \in F \sqcup G}$. We insert τ_{i+1} and check whether it is a creator or a destructor.

Case 1: if $[\partial\tau_{i+1}] = 0$ in $\mathbf{K}_i$, τ_{i+1} is a creator (see proof of Lemma 4.2). It is equivalent to say that:

$$\partial\tau_{i+1} = \sum_{g \in G} \alpha_g \cdot \widehat{\tau}_g$$

The new cycle containing τ_{i+1} is:

$$c = \tau_{i+1} - \sum_{(g,h)\in\mathcal{P}} \alpha_g \cdot \widehat{\tau}_h$$

whose boundary is 0 by virtue of the equality above. We set $\widehat{\tau}_{i+1} \leftarrow c$ for the new chain of the encoding, and set $F \leftarrow F \cup \{i+1\}$. Note that the leading term of $\widehat{\tau}_{i+1}$ is τ_{i+1}.

Case 2: if $[\partial\tau_{i+1}] \neq 0$ in $\mathbf{K}_i$, τ_{i+1} is a destructor (see proof of Lemma 4.2). Because $\partial\tau_{i+1}$ is a cycle, it is equivalent to say that:

$$\partial\tau_{i+1} = \sum_{f\in F} \alpha_f \cdot \widehat{\tau}_f + \sum_{g\in G} \alpha_g \cdot \widehat{\tau}_g$$

where the sum over the indices in F is non zero. Let f_0 be the biggest index in F for which $\alpha_{f_0} \neq 0$ in the expression of $\partial\tau_{i+1}$; we pair f_0 and $i+1$. We set:

$$\left\{ \begin{array}{rcl} \widehat{\tau}_{i+1} & \leftarrow & \tau_{i+1} - \sum_{(g,h)\in\mathcal{P}} \ \alpha_g \cdot \widehat{\tau}_h \\ \widehat{\tau}_{f_0} & \leftarrow & \sum_{f\in F} \alpha_f \end{array} \right.$$

and update the indices:

$$F \leftarrow F \setminus \{f_0\} \qquad G \leftarrow G \cup \{f_0\} \qquad H \leftarrow H \cup \{i+1\}$$

Note that the leading term of $\widehat{\tau}_{f_0}$ is still τ_{f_0} because f_0 has been taken maximal and that the leading term of $\widehat{\tau}_{i+1}$ is indeed τ_{i+1}. Also, by construction, $\partial\widehat{\tau}_{i+1} = \widehat{\tau}_{f_0}$.

In both cases, we obtain valid encodings for the persistence module of the filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_i} \mathbf{K}_i \xrightarrow{\tau_{i+1}} \mathbf{K}_{i+1}$. Note that picking f_0 to be maximal in Case 2 implies that the simplex destroys the youngest homological feature to which its boundary is related. This is known as the *elder rule* of persistence, which preserves the older homological features.

Persistent Cohomology Algorithm. An inclusion $\mathbf{K} \to \mathbf{L}$ induces a homomorphism at cohomology level that goes in reverse direction $\mathbf{H}^*(\mathbf{K}) \leftarrow \mathbf{H}^*(\mathbf{L})$. This is due to the fact that $\mathbf{K} \to \mathbf{L}$ induces a reverse map between cochain groups $\mathrm{Hom}(\mathbf{C}_*(\mathbf{K}), \Bbbk) \leftarrow \mathrm{Hom}(\mathbf{C}_*(\mathbf{L}), \Bbbk)$ where the image of a cochain $c^* \in \mathrm{Hom}(\mathbf{C}_*(\mathbf{L}), \Bbbk)$ is its restriction to the domain $\mathbf{C}_*(\mathbf{K}) \subseteq \mathbf{C}_*(\mathbf{L})$. Note that this last map is surjective, when the map between chain groups $\mathbf{C}_*(\mathbf{K}) \to \mathbf{C}_*(\mathbf{L})$ is injective. Consequently, the filtration:

$$\mathbb{K} := \quad \mathbf{K}_0 \xrightarrow{f_1} \mathbf{K}_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} \mathbf{K}_{n-1} \xrightarrow{f_n} \mathbf{K}_n$$

induces the persistence module in cohomology:

$$\mathbb{H}\text{om} := \quad \mathbf{H}^*(\mathbf{K}_0) \xleftarrow{f_1^*} \mathbf{H}^*(\mathbf{K}_1) \xleftarrow{f_2^*} \cdots \xleftarrow{f_{n-1}^*} \mathbf{H}^*(\mathbf{K}_{n-1}) \xleftarrow{f_n^*} \mathbf{H}^*(\mathbf{K}_n)$$

We have a dual definition of an *encoding* of a persistence module in cohomology. For a filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \mathbf{K}_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \mathbf{K}_{n-1} \xrightarrow{\tau_n} \mathbf{K}_n$, there exist cochains $\widetilde{\tau}_1, \cdots, \widetilde{\tau}_n \in \mathbf{C}^*(\mathbf{K}_n)$, a partition of the indices $\{1, \cdots, n\}$ $= F \sqcup G \sqcup H$, and a bijective pairing $G \leftrightarrow H$, denoted by $\mathcal{P} \subseteq G \times H$, satisfying the following conditions:

1'. for all i, the leading term of $\widetilde{\tau}_i$ is τ_i^*, meaning that $\widetilde{\tau}_i = \alpha_i \tau_i^* + \cdots + \alpha_n \tau_n^*$, with $\alpha_i \neq 0$,

2'. for all $f \in F$, $\delta\widetilde{\tau}_f = 0$,

3'. for all pairs $(g, h) \in \mathcal{P}$, $\delta\widetilde{\tau}_h = \widetilde{\tau}_g$ and hence $\delta\widetilde{\tau}_g = 0$.

Note that because the arrows are reversed, the leading term has now smallest index and every pair $(g, h) \in \mathcal{P}$ satisfies $h < g$.

A cochain $\widetilde{\tau}_f$, with $f \in F$, is a non trivial cocycle over the range of indices $\{f, \cdots, n\}$. Each chain $\widetilde{\tau}_h$, for $h \in H$, restricts to a non trivial cocycle over the range $\{h, \cdots, g-1\}$. For indices bigger or equal to g, $\widetilde{\tau}_h$ is not a cocycle anymore as its coboundary is equal to $\delta\widetilde{\tau}_h = \widetilde{\tau}_g$ which is a non zero chain in the range $\{g, \cdots, n\}$. Note that this is different from the encoding in homology, where a homology class is represented by a cycle $\widehat{\tau}_g$ with index in G, until this cycle becomes trivial (but is still a cycle). As before, such an encoding of the persistence module encodes completely its interval decomposition.

We present an algorithm to maintain an encoding in cohomology when going backward on a homomorphism $\mathbf{H}^*(\mathbf{K}_i) \xleftarrow{f_{i+1}^*} \mathbf{H}^*(\mathbf{K}_{i+1})$. Given an encoding, with indices in $F \sqcup G \sqcup H = \{1, \cdots, i\}$, of the persistence module in cohomology of the filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_i} \mathbf{K}_i$, we consider the map $\mathbf{K}_i \xrightarrow{\tau_{i+1}} \mathbf{K}_{i+1}$. The impact of adding simplex τ_i to the complex is that the cochains have an extra coefficient, *ie*

$$\begin{cases} \delta\widetilde{\tau}_f &= \alpha_f \cdot \tau_{i+1}^* & \text{for } f \in F \\ \delta\widetilde{\tau}_h &= \widetilde{\tau}_g + \alpha_h \cdot \tau_{i+1}^* & \text{for } h \in H \text{ and } (g,h) \in \mathcal{P} \\ \delta\widetilde{\tau}_g &= \alpha_g \cdot \tau_{i+1}^* & \text{for } g \in G \end{cases}$$

We modify the encoding so as to satisfy Conditions 1'., 2'. and 3'. in $\mathbf{K}_{i+1}$. First we set:

$$\begin{cases} \widetilde{\tau}_h &\leftarrow \widetilde{\tau}_h & \text{for } h \in H \\ \widetilde{\tau}_g &\leftarrow \delta\widetilde{\tau}_h = \widetilde{\tau}_g + \alpha_h \cdot \tau_{i+1}^* & \text{for } g \in G \text{ and } (g,h) \in \mathcal{P} \end{cases}$$

Hence Condition 3'. is satisfied for the pairs in $\mathcal{P}$. Now, we consider two cases:

Case 1': All the coefficients α_f, for $f \in F$, are 0. Then we set:

$$\left\{ \begin{array}{rcl} \widetilde{\tau}_{i+1} & \leftarrow & \tau^*_{i+1} \\ \widetilde{\tau}_f & \leftarrow & \widetilde{\tau}_f \end{array} \right.$$

and set $F \leftarrow F \cup \{i+1\}$. This is equivalent to creating a cycle in homology.

Case 2': Some α_f are non zero. Let f_0 be the largest index for which $\alpha_{f_0} \neq 0$. We set:

$$\left\{ \begin{array}{rcl} \widetilde{\tau}_{f_0} & \leftarrow & \widetilde{\tau}_{f_0} \\ \widetilde{\tau}_f & \leftarrow & \widetilde{\tau}_f - (\alpha_f/\alpha_{f_0}) \cdot \widetilde{\tau}_{f_0} \quad \text{for } f \in F \setminus \{f_0\} \\ \widetilde{\tau}_{i+1} & = & \delta\widetilde{\tau}_{f_0} = \alpha_{f_0}\tau^*_{i+1} \end{array} \right.$$

As f_0 is chosen largest, Condition 1'. still holds. These modifications pair f_0 with $i+1$, we consequently set:

$$F \leftarrow F \setminus \{f_0\} \qquad H \leftarrow H \cup \{f_0\} \qquad G \leftarrow G \cup \{i+1\}$$

This is equivalent to destroying a cycle in homology.

In both cases, Conditions 1'., 2'. and 3'. are satisfied and we get, after update, a valid encoding of persistence module in cohomology of the filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_i} \mathbf{K}_i \xrightarrow{\tau_{i+1}} \mathbf{K}_{i+1}$.

These algorithms give a high level presentation of the persistence homology and cohomology algorithms. We give in the next paragraph a matrix implementation of these algorithms.

Implementation with Matrix Reduction. In homology we represent an encoding by an $(n \times n)$-matrix $\mathbf{M}$ with $\Bbbk$ coefficients, where each column j, denoted by col_j, represents the chain $\widehat{\tau}_j$ in the basis $\{\tau_1, \cdots, \tau_n\}$ of $\mathbf{C}_*(\mathbf{K}_n)$. We denote the i^{th} coefficient of a column by $\mathsf{col}[i]$. Due to Condition 1., $\mathbf{M}$ is upper-triangular, with non-zero elements on the diagonal. For a column col_j, we denote by $\mathrm{low}(j)$ the row index of its lowest non-zero element.

For any chain $c \in \mathbf{C}_*(\mathbf{K}_n)$, represented as a column col_{n+1} with index $n+1$ and length n, and for any set of indices $I \subseteq \{1, \cdots, n\}$, we can express c as a linear combination of the chains $\{\widehat{\tau}_i\}_{i \in I}$ (whenever possible) using the reduction of Algorithm 4.1. If the output value of col_{n+1} is 0, then we have computed an expression $c + \sum_{i \in I'} \alpha_i \widehat{\tau}_i = 0$, where $I' \subseteq I$ is the set of indices i_0 picked in the **while** loop, and α_i is the coefficient $-\gamma^{-1}\delta$. Otherwise, if $\mathsf{col}_{n+1} \neq 0$, then $c \notin \langle \widehat{\tau}_i \rangle_{i \in I}$. The algorithm is valid because

Algorithm 4.1: `Reduction(`col_{n+1}`,`I`)`

while *there exists* $i_0 \in I$ *with* $\mathrm{low}(i_0) = \mathrm{low}(n+1)$ **do**
  $\gamma \leftarrow \mathsf{col}_{i_0}[\mathrm{low}(i_0)]$; $\quad \delta \leftarrow \mathsf{col}_{n+1}[\mathrm{low}(n+1)]$;
  $\mathsf{col}_{n+1} \leftarrow \mathsf{col}_{n+1} - \gamma^{-1}\delta \times \mathsf{col}_{i_0}$;
end

$\mathrm{low} : \{1, \cdots, n\} \to \{1, \cdots, n\}$ is injective in $\mathbf{M}$ (actually, the identity) and every column addition strictly reduces $\mathrm{low}(n+1)$. Consequently, we can express $\partial\tau_{i+1}$ as a sum of $\widehat{\tau}_g$ and $\widehat{\tau}_f$ by creating a column col for the chain and running `Reduction(`col_{n+1}`,`G`)` and `Reduction(`col_{n+1}`,`$F \sqcup G$`)`. The modifications of the chains of the encoding are computed via elementary column operations.

Remark 4.1. *Note that if we are just interested in computing the persistence diagram, we do not need to represent the chains with indices in F and H, as the chains $\{\tau_g\}_{g\in G}$ are enough to decide whether a simplex is a creator or a destructor.*

In cohomology we represent an encoding by an $(n \times n)$-matrix $\mathbf{M}$ with $\Bbbk$ coefficients, where each row j, denoted by row_j, represents the cochain $\widetilde{\tau}_j$ in the basis $\{\tau_1^*, \cdots, \tau_n^*\}$ of $\mathbf{C}^*(\mathbf{K}_n)$. The column j, denoted col_j or a_{τ_j}, contains all values of the cochains $\widetilde{\tau}_i$ evaluated on the chain τ_j. We call the columns *annotations* in the next chapter. All the coefficients α_j of the new term $\alpha_j \cdot \tau_{j+1}^*$ of the coboundaries $\delta\widetilde{\tau}_j = \alpha_j \cdot \tau_{j+1}$ after insertion of τ_{i+1} can be computed by evaluating the sum of columns:

$$\mathsf{a}_{\partial\tau_{i+1}} = \mathsf{col}_{i_0} - \mathsf{col}_{i_1} + \cdots + (-1)^d \mathsf{col}_{i_d} \ , \qquad \text{where } \ \partial\tau_{i+1} = \sum_{j=0}^{d} (-1)^j \tau_{i_j}$$

Indeed, the j^{th} coefficient of $\mathsf{a}_{\partial\tau_{i+1}}$ is:

$$\begin{aligned} \mathsf{a}_{\partial\tau_{i+1}}[j] &= \widetilde{\tau}_j(\tau_{i_0}) - \widetilde{\tau}_j(\tau_{i_1}) + \cdots + (-1)^d \widetilde{\tau}_j(\tau_{i_d}) \\ &= \widetilde{\tau}_j(\partial\tau_{i+1}) \\ &= \delta\widetilde{\tau}_j(\tau_{i+1}) \\ &= \alpha_j \end{aligned}$$

Updating the encoding consists then only in elementary row operations.

Remark 4.2. *Note that if we are just interested in computing the persistence diagram, we do not need to represent the chains with indices in G and H, as*

the chains $\{\tau_f\}_{f\in F}$ are enough to decide whether a simplex is a creator or a destructor. In teh next chapter, we describe an efficient implementation of this algorithm using, in particular, this more compact representation of the encoding.

Bibliographical Notes.

The paragraph giving an intuition about homology reproduces the introduction to computational homology the author taught at the winter school on algorithmic geometry of triangulations, in January 2014 at INRIA Sophia Antipolis.

The formal introduction to homology and cohomology theory is mostly inspired from [59] and [42], to which we refer for further reading. In particular, we refer to [59] for an explicit construction of the induced homomorphisms of Theorem 4.1 and the proof of the invariance of homology groups of Theorem 4.2.

The homology groups with $\mathbb{Z}$ and $\Bbbk$ coefficients are computable using a reduction to Smith normal form of the matrices of the boundary operators ∂_d for all dimensions d (see [59] and [41] for an efficient implementation).

The algebraic foundation of persistent homology has first appeared in [43] for 3-dimensional simplicial complexes and $\mathbb{Z}_2$ coefficients. The theory has been generalized to its general form (arbitrary dimensions and coefficients in a field $\Bbbk$) in [70]. The authors of [70] proved that persistence modules could decompose into intervals by modelizing them with $\Bbbk[t]$-modules and using a decomposition theorem for graded modules over the PID $\Bbbk[t]$ [67]. They also provide an algorithm for computing persistence, presented as a matrix algorithm.

The matrix reduction algorithm for persistent homology has running time $O(n^3)$ where n is the number of simplices of the simplicial complex and, despite good performance in practice, this bound is tight [57].

Recent optimizations taking advantage of the special structure of the matrix to be reduced have led to significant progress in the theoretical analysis [29, 54], where the time complexity has been reduced to matrix multiplication time, as well as in practice [10, 29].

The persistent cohomology algorithm has been introduced in [35]. In order to uniformize the perspective about persistent (co)homology algorithms, the presentation in terms of a partition $F \sqcup G \sqcup H$ detailled in this chapter has been suggested in [34]. The cohomology algorithm has been reported to work better in practice than the standard homology algorithm [34] but this advantage seems to fade away when optimizations are employed to the ho-

mology algorithms [10]. A simple description of the cohomology algorithm, using the notion of annotations, has been introduced in [40] and used to design more general algorithms for maintaining cohomology groups under simplicial maps. This algorithm admits an efficient implementation with a *compressed annotation matrix* [15], that we detail in the next chapters.

The Stability Theorem 4.3 for persistence diagram has been proved in [30]. It generalizes to more general functions – with a notion of tameness – defined on triangulable topological spaces.

Chapter 5

Compressed Annotation Matrix

Along this chapter, the cohomology groups are defined with coefficients in a field $\Bbbk$. The algorithmic complexity of addition, substraction, multiplication and inversion is considered constant. The approach is independent from the type of complex used to represent the topological space. In particular, the approach is valid for any cell complex.

We introduce an implementation of a matrix algorithm for computing persistent cohomology presented in Chapter 4. We separate the representation of the simplicial complex from the representation of the cohomology groups, and introduce a new data structure for maintaining the annotation matrix, called the *compressed annotation matrix*. The implementation may be seen as a generalization of the union-find algorithm to compute the connected components of a graph. We extend this algorithm to the cohomology of a simplicial complex by attaching topological information, called *annotations*, to the simplices. In addition, we propose a heuristic to simplify further the representation of the cohomology groups and improve both time and space complexities.

5.1 Annotations and Persistent Cohomology

Disjoint Sets for Connected Components. Maintaining dynamically the number of connected-components, or in other words the dimension of the 0-dimensional homology group, of a simplicial complex can be implemented easily with a *disjoint sets data structure*. A *disjoint sets data structure* [32] maintains a collection $\mathcal{F} = \{s_1, \cdots, s_k\}$ of disjoint sets, where all s_i are a

subset of a finite universe U. Each set is identified by a unique representative element, which is a member of the set. The representative element is always the same until the data structure is modified. The data structure supports the following operations: for two elements $x, y \in U$ and a collection $\{s_1, \cdots, s_k\}$ of disjoint sets,

- `Make-sets`(x): for an element x such that $x \notin s_i$ for any i, create a new set s_{k+1} containing only element x.
- `Union-sets`(x,y): for two elements x and y belonging to some sets, respectively s_i and s_j, of the collection. If $s_i \neq s_j$, the procedure computes the collection $\{s_1, \cdots, \widehat{s_i}, \cdots, \widehat{s_j}, \cdots, s_k, s_i \cup s_j\}$. It does not modify the collection otherwise.
- `Find-set`(x): for an element x belonging to a set s_i, returns the representative of the set s_i. While `Union-sets` is not called, the representative of every set remains unchanged. In particular, if the structure is not modified between two calls to `Find-set`, the property `Find-set`(x) $\neq$ `Find-set`(y) is true *iff* x and y are in different sets of the collection.

Given a filtration of a 1-dimensional complex:

$$\mathbb{K} = \quad \emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \mathbf{K}_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \mathbf{K}_{n-1} \xrightarrow{\tau_n} \mathbf{K}_n$$

we can maintain its 0-dimensional persistent homology with the procedure `0-persistent-homology` described in Algorithm 5.1. The procedure maintains the connected components using disjoint sets of vertices. When a new vertex is added to the complex, we create a new set for a new connected

Algorithm 5.1: `0-persistent-homology`($\mathbb{K}$)

```
for i = 1 ··· n do
    if dim τ_i > 1 then do nothing;
    if τ_i is a vertex u then Make-set(u);
    if τ_i is an edge (u, v) where u = τ_{j1} and v = τ_{j2} then
        if Find-set(u) ≠ Find-set(v) then
            add a pair (max{j1, j2}, i) to the persistence diagram;
            Union-set(u,v);
        end
    end
end
```

component, using `Make-set`. When a new edge (u, v) connects two distinct components – which happens when the sets of u and v are distinct – we merge them using `Union-set` and we add a pair creator - destructor to the persistence diagram. Among the two merged components, we destroy the youngest one, as an expression of the elder rule. Note that when u and v are in the same component, the edge (u, v) actually creates a 1-cycle class. This disjoint sets approach has also been used to compute the (persistent) homology, in all dimensions, of any simplicial complex embedded in the 3-sphere [36].

This algorithm works because the 0-dimensional cohomology group of the complex $\mathbf{K}_i$ keeps a simple expression along the experiment. Suppose $\mathbf{K}_i$ has β connected components. Let s_j be the set of vertices of the j^{th} connected component of $\mathbf{K}_i$, define the chain $\widetilde{\tau}_j \colon \mathbf{C}_0(\mathbf{K}_i) \to \Bbbk$ that evaluates to 1 on every vertex of s and 0 otherwise. It is a cocycle. Indeed, on every oriented edge $e = [u, v]$, $\delta\widetilde{\tau}_j e = \widetilde{\tau} v - \widetilde{\tau} u$ is always 0, as the edge e either connects two vertices of the component s, or two vertices outside s. Finally, the set of cohomology classes $\{[\widetilde{\tau}_1], \cdots, [\widetilde{\tau}_\beta]\}$ is a basis for $\mathbf{H}^0(\mathbf{K}_i)$. Note in particular that, because we are in dimension 0, $\mathbf{B}^0 = 0$, and every cocycle must always evaluate the same for all vertices of the same component.

From a different perspective, every vertex evaluates to 1 on exactly one representative cocycle of the aforementioned cohomology basis, the one corresponding ot the connected component containing the vertex. We generalize the disjoint set approach in the rest of the chapter, where we maintain an additionnal *annotation vector* for each simplex – its values on a cohomology basis – to keep track of topological information.

Annotation Algorithm. The persistence cohomology algorithm has first appeared in [35]. We have followed the same presentation in Chapter 4In the following, we use the vocabulary of *annotations* from [40] and give an implementation based on elementary column operations instead. Given as input a filtration:

$$\mathbb{K} := \quad \emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \mathbf{K}_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \mathbf{K}_{n-1} \xrightarrow{\tau_n} \mathbf{K}_n = \mathbf{K}$$

where all maps are elementary inclusions, we compute the persistence diagram of $\mathbb{K}$. Along the computation, we maintain only the cochains of an encoding with indices in F, as defined in the previous chapter. These cochains are cocycles that represent cohomology classes forming a basis of the cohomology groups (see Remark 4.1). Given a complex $\mathbf{K}$ from the filtration, let $\widetilde{\tau}_1, \cdots, \widetilde{\tau}_\beta$ be the cocycles mentioned above and belonging to an encoding

of the persistence module of the filtration $\mathbb{K}$. By definition of an encoding, the cohomology classes of the cocycles $\{\widetilde{\tau}_f\}_{f\in F}$ form a basis of $\mathbf{H}^*(\mathbf{K})$. In particular, note that β is equal to $\dim \mathbf{H}^*(\mathbf{K})$.

The *annotation vector* of a simplex $\sigma \in \mathbf{K}$ is the vector a_σ of $\Bbbk^\beta$ such that $\mathsf{a}_\sigma[j] = \widetilde{\tau}_j(\sigma)$. We extend the definition by linearity; the annotation vector of a chain $c = \sum \alpha_j \tau_j$ is equal to $\mathsf{a}_c = \sum_j \alpha_j \mathsf{a}_{\tau_j}$. When $\mathbf{K}$ is the last simplicial complex $\mathbf{K}_n$ of a filtration $\mathbb{K}$, we say that the annotation is *valid* if the set of cochains:

$$\{\widetilde{\tau}_j \ : \ \widetilde{\tau}_j[\sigma] = \mathsf{a}_\sigma[j] \ \text{ for every } \sigma \in \mathbf{K}\}_j$$

is equal to the set of cocycles with indices in F of an encoding of $\mathbb{K}$. In particular, the annotation vector of a simplex σ, in a valid annotation, is the vector of values of the cocycles, representing a basis of $\mathbf{H}^*(\mathbf{K})$, evaluated on σ.

We represent the annotation vectors of a valid annotation in a $\beta \times n$ matrix with $\Bbbk$ coefficients, whose j^{th} column is the annotation a_{τ_j} of τ_j. This matrix is equal to the matrix $\mathbf{M}$ of the matrix implementation of the persistent cohomology algorithm of Chapter 4 from which we remove the rows corresponding to the cochains $\widetilde{\tau}_j$ with index $j \in G \sqcup H$.

Suppose we are given a valid annotation for $\mathbf{K}_i$ represented as a matrix $\mathbf{M}_i$, and corresponding to the cocycles $\{\widetilde{\tau}_1, \cdots, \widetilde{\tau}_\beta\}$ with indices in F in an encoding of the filtration up to $\mathbf{K}_i$. We update the matrix $\mathbf{M}_i$ so as to get a valid annotation matrix $\mathbf{M}_{i+1}$ for $\mathbf{K}_{i+1}$, where $\mathbf{K}_i \xrightarrow{\tau_{i+1}} \mathbf{K}_{i+1}$. We compute $\mathsf{a}_{\partial\tau_{i+1}}$ and proceed as follows:

Case 1: $\mathsf{a}_{\partial\tau_{i+1}} = 0$. We are in Case 1'. of the persistent cohomology algorithm. Hence we create the new cocycle $\widetilde{\tau}_{\beta+1} = \tau^*_{i+1}$ to the cocycles $\{\widetilde{\tau}_1, \cdots, \widetilde{\tau}_\beta\}$. Specifically, we add a new row and a new column in $\mathbf{M}_i$, with value 0 everywhere except at the bottom right corner of the martrix, where the value is 1.

Case 2: $\mathsf{a}_{\partial\tau_{i+1}} \neq 0$. let f_0 be the bigger index for which $\mathsf{a}_{\partial\tau_{i+1}} \neq 0$. Denote by γ_{f_0} this value. To proceed to the row operations of the cohomology algorithm, we set:

$$\mathsf{a}_{\tau_i} \leftarrow \mathsf{a}_{\tau_i} - \mathsf{a}_{\tau_j}[f_0]\gamma^{-1} \times \mathsf{a}_{\partial\tau_{i+1}}$$

Note that this modifies only the annotation vectors a_τ for which $\mathsf{a}_\tau[f_0] \neq 0$. This operation implements the operation $\widetilde{\tau}_f \leftarrow \widetilde{\tau}_f - (\alpha_f/\alpha_{f_0}) \cdot \widetilde{\tau}_{f_0}$ of Case 2'. of the cohomology algorithm. We keep track of the pair $(f_0, i+1)$ as a point in the output index persistence diagram.

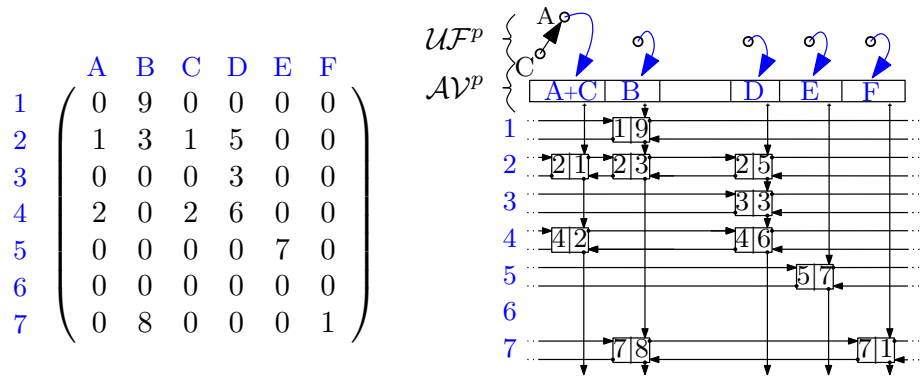


Figure 5.1: Compressed annotation matrix of a matrix with integer coefficients.

Since we process the filtration in a direction opposite to the cohomology sequence, we discover the death points of cohomology classes earlier than their birth points. To avoid confusion, we still say that a new cocycle (or its class) is born when we discover it for the first time and an existing cocycle (or its class) dies when we see it no more.

The annotation approach adapts [40] to a simple and natural extension of the persistence algorithm for filtrations connected with general simplicial maps (and not simply inclusion).

5.2 The Compressed Annotation Matrix

In this section, we present our implementation of the annotation-based persistent cohomology algorithm. We separate the representation of the complex from the representation of the cohomology groups.

Representation of the Complex. We represent the simplicial complex $\mathbf{K}$ in a data structure $\mathcal{SC}$ equipped with the operation `Compute-boundary`(σ) that computes the boundary of a simplex σ together with the coefficients of the corresponding chain. We denote by $\mathcal{C}_\partial^d$ the complexity of this operation where d is the dimension of σ. Additionally, the simplices are ordered according to the filtration.

Two data structures, presented in Chapter 2, to represent (simplicial) complexes are of particular interest here; we recall some complexity results. The first is the *Hasse diagram* which is a graph of size $O(k \cdot \#\mathbf{K})$ for a complex of dimension $\mathbf{K}$. It allows the representation of general cell complexes and gives an access to the boundary of a cell σ in $O(\#\partial\sigma)$ operations. The second

data structure is the *simplex tree*, which is a specific spanning tree of the Hasse diagram. It allows the representation of simplicial complexes. For a simplicial complex of size $\#\mathbf{K}$, the size of the simplex tree is $O(\#\mathbf{K})$ and it gives an access to the boundary of a d-simplex σ in $O(d^2 D_{\mathrm{m}})$, for a quantity D_{m} that depends on the incidence relations in the complex and remains small in practice.

Both structures can be used in our setting. For readability, we will use a Hasse diagram in the following.

Representation of the Annotation Matrix. We introduce the *compressed annotation matrix*, which is an efficient representation of the matrix $\mathbf{M}_i$ that allows to implement the persistent cohomology algorithm of Section 5.1.

Sparse encoding: in most applications, the annotation matrix is sparse and we store it as illustrated in Figure 5.1. A column is represented as the singly-linked list of its non-zero elements, where the list contains a pair (i, γ) if the i^{th} element of the column is $\gamma \neq 0$. The pairs in the list are ordered according to row index i. All pairs (i, γ) with same row index i are linked in a doubly-linked list.

Compression: to avoid storing duplicate columns, we use two data structures. The first one, $\mathcal{AV}$, stores the annotation vectors and allows fast search, insertion and deletion. $\mathcal{AV}$ can be implemented as a red-black tree or a hash table. We denote by $\mathcal{C}_{\mathcal{AV}}$ the complexity of an operation in $\mathcal{AV}$. For example, if $\mathcal{AV}$ contains m elements and $c_{\max}$ is the length of the longest sparse column, we have $\mathcal{C}_{\mathcal{AV}} = O(c_{\max} \log m)$ for a red-black tree implementation and $\mathcal{C}_{\mathcal{AV}} = O(c_{\max})$ amortized for a hash-table.

The simplices that have the same annotation vector are now stored in a same set and the collection of disjoint sets are stored in a disjoint sets data structure. For its efficiency, the disjoint set data structure is implemented with a *union-find*, and is denoted by $\mathcal{UF}$. $\mathcal{UF}$ is encoded as a forest where each tree contains the elements of a set, the root being the representative of the set. The trees of $\mathcal{UF}$ are in bijection with the different annotation vectors stored in $\mathcal{AV}$ and the root of each tree maintains a pointer to the corresponding annotation vector in $\mathcal{AV}$. Each simplex σ in the simplicial complex $\mathcal{SC}$ stores a pointer to an element of the tree of $\mathcal{UF}$ associated to the annotation vector a_σ. See Figure 5.1.

As a consequence, finding the annotation vector of σ consists in getting the element it points to in a tree of $\mathcal{UF}$ and then finding the root of the tree which points to a_σ in $\mathcal{AV}^p$. We avail the following operations on $\mathcal{UF}$.

The union-find $\mathcal{UF}$ avails the operations `Make-set` (creates a new tree with one element), `Union-sets` (merges two trees) and `Find-set` (finds the root of a tree, given an element in the tree). The number of elements maintained in $\mathcal{UF}$ is at most the number of simplices. The operations `Find-root` and `Union-sets` on $\mathcal{UF}$ can be computed in amortized time $O(\alpha(\#\mathbf{K}^p))$, where $\alpha(\cdot)$ is the very slowly growing inverse Ackermann function (constant less than 4 in practice), and `Create-set` is performed in constant time worst case.

We call this data structure the *compressed annotation matrix*. Note that it generalizes the disjoint set algorithm for connected components described in Section 5.1. Indeed, it maintains classes of simplices that are equivalent if they have the same annotation vector. As mentioned before, in dimension 0, having the same annotation vector is equivalent to being in the same connected component.

Operations on the Compressed Annotation Matrix. The compressed annotation matrix described above supports the following operations. We define $c_{\max}$ to be the maximal number of non-zero elements in a column of the compressed annotation matrix (or equivalently in an annotation vector) and $r_{\max}$ to be the maximal number of non-zero elements in a row of the compressed annotation matrix, during the computation. We express our complexities using $c_{\max}$ and $r_{\max}$.

- `Sum-ann`$(\mathtt{a}_1, \mathtt{a}_2)$: computes the sum of two annotation vectors $\mathtt{a}_1$ and $\mathtt{a}_2$, and returns the lowest non-zero coefficient if it exists. The column elements are sorted by increasing row index, so the sum is performed in $O(c_{\max})$ time.

- `Search-ann`,`Insert-ann`,`Remove-ann`$(\mathtt{a})$: searches, inserts or removes an annotation vector $\mathtt{a}$ from $\mathcal{AV}$ in $O(\mathcal{C}_{\mathcal{AV}})$ time.

- `Create-cocycle`(): implements **Case 1** of the annotation algorithm described in Section 5.1. It inserts a new column in $\mathcal{AV}$ containing one element $(\beta+1, 1)$, where $\beta+1$ is the index of the created cocycle. This is performed in $O(\mathcal{C}_{\mathcal{AV}})$ time. It also creates a new disjoint set in $\mathcal{UF}$ for the new column. This is done in $O(1)$ time using CREATE-SET. `Create-cocycle`() takes time $O(\mathcal{C}^p_{\mathcal{AV}})$ in total.

- `Kill-cocycle`$(\mathtt{a}_{\partial\sigma}, \gamma, f_0)$: implements **Case 2** of the algorithm. It finds all columns with a non-zero element at index f_0 and, for each such column $\mathtt{a}$, it adds to $\mathtt{a}$ the column $-\mathtt{a}[j]\gamma_j^{-1} \times \mathtt{a}_{\partial\sigma}$ To find the columns

with a non-zero element at index f_0, we use the doubly-linked list of row f_0. We call `Sum-ann` to compute the sums. The overall time needed for all columns is $O(c_{\max} r_{\max})$ in the worst-case. Finally, we remove duplicate columns using operations on $\mathcal{AV}$ (in $O(r_{\max} \mathcal{C}_{\mathcal{AV}}^{p-1})$ time in the worst-case) and call `Union-sets` on $\mathcal{UF}$ if two sets of simplices, which had different annotation vectors before calling KILL-COCYCLE, are assigned the same annotation vector. This is performed in at most $O(r_{\max} \alpha(\#\mathbf{K}))$ time. The total cost of `Kill-cocycle` is $O(r_{\max}(c_{\max} + \mathcal{C}_{\mathcal{AV}} + \alpha(\#\mathbf{K})))$.

Persistent Cohomology via the Compressed Annotation Matrix. Given as input a filtered simplicial complex represented in a data structure $\mathcal{SC}$, we compute its persistence diagram.

We insert the simplices in the filtration order and update the data structures during the successive insertions. The simplicial complex $\mathbf{K}$ is stored in a simplicial complex data structure $\mathcal{SC}$ and we maintain a compressed annotation matrix representing a valid annotation. The matrix is empty at the beginning of the computation. For readability, we add the following operation on the set of data structures:

- `Compute-a`$_{\partial\sigma}(\sigma)$: given a d-simplex σ in $\mathbf{K}$, computes its boundary in $\mathcal{SC}$ using COMPUTE-BOUNDARY (in $O(\mathcal{C}_\partial^d)$ time). For each of the $d+1$ simplices in $\partial\sigma$, it then finds their annotation vector using `Find-set` in $\mathcal{UF}$ (in $O(d\alpha(\#\mathbf{K}))$ time). Finally, it sums all these annotation vectors together (with the appropriate $+/-$ signs) using at most $d+1$ calls to `Sum-ann` (in $O(d\, c_{\max})$ time). Note that, with the compression method, two simplices in $\partial\sigma$ may point to the same annotation vector; the computation is fasten by adding such annotation vector only once, with the appropriate multiplicative coefficient. The total worst case complexity of this operation is $O(\mathcal{C}_\partial^d + d\,\alpha(\#\mathbf{K}) + d\, c_{\max})$.

Let σ be a d-simplex to be inserted. We compute the annotation vector of $\partial\sigma$ using `Compute-a`$_{\partial\sigma}$. Depending on the value of $\mathtt{a}_{\partial\sigma}$, we call either `Create-cocycle` or `Kill-cocycle`. The algorithm computes the pairing of simplices from which one can deduce the persistence diagram. By reversing the pointers from the $\mathcal{UF}$ to the simplices in $\mathcal{SC}$, one can compute explicitly the representative cocycles of the basis classes and have an explicit representation of the cohomology groups along the computation.

We study the complexity of the computation of persistent cohomology using the compressed annotation matrix. Let k be the dimension and n the

number of simplices of $\mathbf{K}$. Recall that $c_{\max}$ and $r_{\max}$ represent respectively the maximal number of non-zero elements in an annotation vector and in a row of the compressed annotation matrix, along the computation. Recall that, in dimension d, $\mathcal{C}_\partial^d$ is the complexity of `Compute-boundary` in $\mathcal{SC}$ and $\mathcal{C}_{\mathcal{AV}}$ the complexity of an operation in $\mathcal{AV}$. $\alpha(\cdot)$ is the inverse Ackermann function. The complexity for inserting σ with dimension d is:

$$O\left(\mathcal{C}_\partial^d + d(\alpha(\#\mathbf{K}) + c_{\max}) + \mathcal{C}_{\mathcal{AV}} + r_{\max}(c_{\max} + \mathcal{C}_{\mathcal{AV}} + \alpha(\#\mathbf{K}))\right)$$

Consequently, the total cost for computing persistent cohomology is:

$$O\left(n \times \left[\mathcal{C}_\partial^k + k(\alpha(n) + c_{\max}) + r_{\max}(c_{\max} + \mathcal{C}_{\mathcal{AV}} + \alpha(n))\right]\right)$$

Specifically, if we implement $\mathcal{SC}$ as a Hasse diagram and $\mathcal{AV}$ with a hash-table, we get $\mathcal{C}_\partial^k = O(k)$ and $\mathcal{C}_{\mathcal{AV}} = O(c_{\max})$. If we consider $\alpha(n)$ as a small constant and remove it for readability, we get that the total cost for computing persistent cohomology is:

$$O(n\, c_{\max}(k + r_{\max}))$$

We show in the next chapter that $c_{\max}$ and $r_{\max}$ remain small in practice. Hence, the practical complexity of the algorithm is linear in n for a fixed dimension.

5.3 Reordering Iso-simplices

Many simplices, called iso-simplices, may have the same filtration value. This situation is common when the filtration is induced by a geometric scaling parameter, like in the Čech, Rips and relaxed witness complexes. Assume that we want to compute the cohomology groups $H^p(\mathbf{L})$ from $H^p(\mathbf{K})$ where $\mathbf{K} \subseteq \mathbf{L}$ and all simplices in $\mathbf{L}\backslash\mathbf{K}$ have the same filtration value. Depending on the insertion order of the simplices of $\mathbf{L}\backslash\mathbf{K}$, the dimension of the cohomology groups to be maintained along the computation may vary a lot as well as the computing time. This may lead to a computational bottleneck. We propose a heuristic to reorder iso-simplices and show its practical efficiency in the next Chapter.

Intuitively, we want to avoid the creation of many short life holes of dimension d and want to fill them up as soon as possible with simplices of dimension $d+1$. For example, in Figure 5.2, we want to avoid inserting all edges first, which would create two holes that would get filled up when

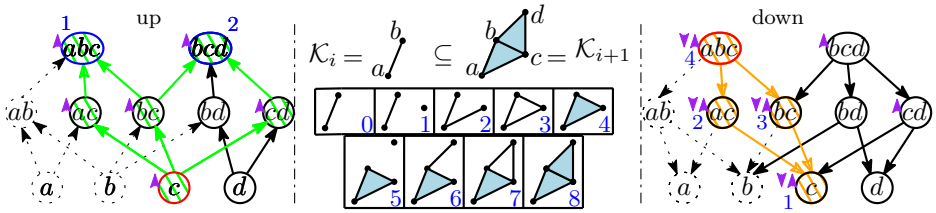


Figure 5.2: Inclusion $\mathbf{K} \subseteq \mathbf{L}$. Left: upward traversal (in green) from simplex $\{c\}$. The ordering of the maximal cofaces appears in blue. Right: downward traversal (in orange) from simplex $\{a, b, c\}$. The ordering of the faces appears in blue.

inserting the triangles. To do so, we look for the maximal simplices to be inserted and recursively insert their faces. We conduct the recursion so as to minimize the maximum number of holes. In addition, to avoid the creation of holes due to maximal simplices that are incident, maximal simplices sharing faces are inserted next to each other. We can describe the reordering algorithm in terms of a graph traversal. The graph considered is the graph of the Hasse diagram of $\mathbf{L} \setminus \mathbf{K}$; see Figure 5.2.

Let $\sigma_1 \cdots \sigma_\ell$ be the iso-simplices of $\mathbf{L} \setminus \mathbf{K}$, sorted so as to respect the inclusion order. We attach to each simplex two flags, a flag F_{up} and a flag F_{down}, set to `0` originally. When inserting a simplex σ_j, we proceed as follows. We traverse the Hasse diagram upward in a depth-first fashion and list the inclusion-maximal cofaces of σ_j in $\mathbf{L} \setminus \mathbf{K}$. The flags F_{up} of all traversed nodes are set to `1` and the maximal cofaces are ordered according to the traversal. From each maximal coface in this order, we then traverse the graph downward and order the faces in a depth-first fashion: this last order is the order of insertion of the simplices. The flags F_{down} of all traversed nodes are set to `1`. We stop the upward (resp. downward) traversal when we encounter a node whose flag F_{up} (resp. F_{down}) is set to `1`. We do not insert either simplices that have been inserted previously.

By proceeding as above on all simplices of the sequence $\sigma_1 \cdots \sigma_\ell$, we define a new ordering which respects the inclusion order between the simplices. Indeed, as the downward traversal starts from a maximal simplex and is depth first, a simplex is always inserted after its faces. Every edge in the graph is traversed twice, once when going upward and the other when going downward. Indeed, during the upward traversal, at each node N associated to a simplex σ_N, we visit only the edges between N and the nodes associated to the cofaces of σ_N and, during the downward traversal, we visit only the edges between N and the nodes associated to the faces of σ_N. If $\mathbf{L} \setminus \mathbf{K}$ contains ℓ simplices, the reordering takes in total $O(\ell \times (\mathcal{C}_\partial + \mathcal{C}_{\text{co}\partial}))$ time,

where $\mathcal{C}_\partial$ (resp. $\mathcal{C}_{\text{co}\partial}$) refers to the complexity of computing the codimension 1 faces (resp. cofaces) of a simplex in the simplicial complex data structure $\mathcal{SC}$.

The reordering of the filtration can either be done as a preprocessing step if the whole filtration is known, or on-the-fly as only the neighboring simplices of a simplex need to be known at a time. The reordering of a set of iso-simplices respects the inclusion order of the simplices and the filtration, and therefore does not change the persistence diagram of the filtered simplicial complex. This is a direct consequence of the Stability Theorem 4.3 of persistence diagrams. However, a homology feature may be created and destroyed by different simplices.

Chapter 6

Performance of the Compressed Annotation Matrix

In this section, we report on the experimental performance of the compressed annotation matrix implementation of persistent cohomology. Given a filtered simplicial complex as input, we compute the persistence diagram of the corresponding filtration. We measure timings and provide various statistics. We compare the timings with `Dionysus` and `PHAT`. `Dionysus` provides implementation for persistent homology [43, 70] and the persistent cohomology algorithm [35] (denoted `DioCoH`) described in Chapter 4, with coefficients in field $\mathbb{Z}_p$, for any prime p. `PHAT` provides an implementation of the optimized matrix algorithm for persistent homology [28, 10] (using the `-twist` option) as well as an implementation of persistent cohomology [10] (using the `-dualize` option), with coefficients in $\mathbb{Z}_2$ only. Note that the persistent cohomology algorithm of `PHAT` is different from the one we have described. It is very similar to the persistent homology algorithm, but proceeds to operations in a different order.

`DioCoH`, `PHAT`$^\perp$ and `PHAT` have been reported to be the most efficient implementation in practice [10, 35]. The symbols T_∞ means that the computation lasted more than 12 hours.

We construct three families of simplicial complexes [42] which are of particular interest in topological data analysis: the Rips complexes (denoted `Rips`), the relaxed witness complexes (denoted `Wit`) and the α-shapes (denoted α`Sh`). These complexes depend on a relaxation parameter denoted by ρ. When the data points are embedded, the complexes are constructed up to

						DioCoH		PHAT$^\perp$PHAT		CAM	
Data	Cpx	$\mathcal{P}$	$\rho_{\max}$	k	$\#\mathbf{K}$	$\mathbb{Z}_2$	$\mathbb{Z}_{11}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{11}$
Cy8	Rips	6040	0.41	16	21×10^6	420	4822	44	5.3	6.4	6.5
S4	Rips	507	0.715	5	72×10^6	943	1026	95	3591	22.5	23.2
L57	Rips	4769	0.02	3	34×10^6	239	240	35.2	972	9.3	9.5
Bro	Wit	500	0.06	18	3.2×10^6	807	T_∞	6.3	0.88	2.7	2.9
Kl	Wit	10000	0.105	5	74×10^6	569	662	101	1785	19.7	19.9
L35	Wit	700	0.06	3	18×10^6	109	110	17.5	869	5.1	5.1
Bud	αSh	49990	∞	3	1.4×10^6	30.0	30.9	2.6	0.32	0.7	0.7
Nep	αSh	2×10^6	∞	3	57×10^6	T_∞	T_∞	163	33	39.5	40.2

Figure 6.1: Data, timings (in seconds) and statistics.

embedding dimension, with Euclidean metric. They are constructed up to the intrinsic dimension of the space with intrinsic metric otherwise. We use a variety of both real and synthetic datasets, listed in Figure 6.1 with details on the sets of points $\mathcal{P}$, their size $\#\mathcal{P}$, the threshold $\rho_{\max}$, the dimension k of the simplicial complexes and the size $\#\mathbf{K}$ of the simplicial complexes.

6.1 Time Performance

As `Dionysus` and `PHAT` encode explicitely the boundaries of the simplices, we use a Hasse diagram for implementing $\mathcal{SC}$. We thus have the same time complexity for accessing the boundaries of simplices. We use the persistent homology algorithm of `PHAT` with options `-twist -sparse-pivot` and the persistent cohomology algorithm (noted `PHAT`$^\perp$) with option `-twist -sparse-pivot -dualize` as the `-sparse-pivot` representation of columns has been observed to be the most efficient in practice. As illustrated in Figure 6.1, the persistent cohomology algorithm of `Dionysus` is always two orders of magnitude slower than our implementation. Moreover, `DioCoH` is sensitive to the field used, as illustrated in the case of **Cy8** and **Bro**. On the contrary, `CAM` shows almost identical performance for $\mathbb{Z}_2$ and $\mathbb{Z}_{11}$ coefficients on all examples. The persistent cohomology algorithm `PHAT`$^\perp$ performs better than `DioCoH`. However, `CAM` is still between 2.3 and 6.9 times faster.

The persistent homology algorithm of `PHAT` shows good performance in the case of the alpha shapes and on **Cy8** and **Bro**: `CAM` and `PHAT` have close timings. However, `PHAT` provides computation with $\mathbb{Z}_2$ coefficients only, whereas `CAM` computes persistence for general field coefficients and integrates no specific optimization for $\mathbb{Z}_2$. Moreover, `CAM` scales better to more complex

Nep	$\#M$	$\#\mathbb{k}$op.
Compression	126057	84×10^6
¬Compression	574426	3860×10^6

Nep	average	maximum
$c_{\text{av}}, c_{\max}$	0.79	18
$r_{\text{av}}, r_{\max}$	1.02	18

Bro	time
Reordering	2.9 s.
¬Reordering	14.2 s.

Bro	$\mathbb{Z}_{11}$			$\mathbb{Q}$		
	M_{DS}	$\mathtt{a}_{\partial\sigma}$	M_{op}	M_{DS}	$\mathtt{a}_{\partial\sigma}$	M_{op}
	71%	19%	10%	67%	21%	12%

Figure 6.2: Statistics on the effect of the optimizations.

examples (such as **S4**, **L57**, **Kl** and **L35**, which have higher intrinsic dimension and more complex topology). Indeed, the running time per simplex of `CAM` remains stable on all examples and for all field coefficients (between 2.7×10^{-7} and 9.1×10^{-7} seconds per simplex).

6.2 Statistics and Optimization

Figure 6.2 presents statistics about the computation. The top table presents, on the left, the effect of the compression (removal of duplicate columns) of the annotation matrix on the number of non-zero coefficients $\#M$ stored in the sparse representation and the number of changes $\#\mathbb{k}$op. operated in the matrix during the computation of the persistence diagram of **Nep**. We note a reduction factor of 4.5 for the size of the matrix, and we proceed to 46 times less field operations with the compression. Considering **Nep** is 57 million simplices, we proceed to less than 1.5 field operations per simplex on average. The right part of the table shows the average and maximum number of non-zero elements in a column when proceeding to a sum of annotation vectors (`Sum-ann`) and the average and maximum number of non-zero elements in a row when proceeding to its reduction (`Kill-cocycle`). These values are key variables ($c_{\max}$ and $r_{\max}$ respectively) in the complexity analysis of the algorithm. We note that these values remain really small. The bottom table presents the effect of the reordering strategy on the example **Bro**. Reordering iso-simplices makes the computation 4.9 faster. Finally, the right side of the table presents how the computing time is divided into maintaining the compressed annotation matrix (denoted by M_{DS}), computing the annotation vector $\mathtt{a}_{\partial\sigma}$ and modifying the values of the elements in the compressed annotation matrix (denoted by M_{op}). The percentage are given when computing persistent cohomology with $\mathbb{Z}_{11}$ and $\mathbb{Q}$ coefficients. The computational complexity of field operations $\langle \mathbb{k}, +, \cdot, -, /, 0, 1 \rangle$ depends on the field we use. For $\mathbb{Z}_{11}$, or any field of small cardinal, the operations

can be precomputed and accessed in constant time. The field operations in $\mathbb{Q}$ are more costly. Specifically, an element q in $\mathbb{Q}$ is represented as a pair of coprime integers (r, s) such that $q = r/s$, and field operations may require greatest common divisor computations to ensure that nominator and denominator remain coprime. However, the computational time of `CAM` is quite insensitive to the field we use. Specifically, as it minimizes the number of matrix changes using the compression method, the computational time is only increased by 8% when computing persistence with $\mathbb{Q}$ coefficients instead of $\mathbb{Z}_{11}$, whereas the computation involving field operations takes 34% more time.

In all our experiments, the size of the compressed annotation matrix is negligible compared to the size of the simplicial complex. Consequently, combined with the simplex tree data structure [18] for representing the simplicial complex, we have been able to compute the persistent cohomology of simplicial complexes of several hundred million simplices in high dimension.

Part III

Modular Reconstruction for Multi-Field Persistence

This Part is based on the following publications:

- Jean-Daniel Boissonnat and Clément Maria. Computing Persistent Homology with Various Coefficient Fields in a Single Pass. *European Symposium on Algorithms 2014* [19]

Chapter 7

Multi-Field Persistence

We study the possibility of defining a stronger persistence-based topological invariant. Persistent homology is restricted to field coefficients in order for an interval decomposition of the persistence module to exist. However, the homology groups with coefficients in a field furnish a weaker topological invariant, even on simple examples. We study, through the *universal coefficient theorem*, the relation between homology groups with integer and field coefficients and show that a partial reconstruction of the homology groups in $\mathbb{Z}$ may be infered from the computation of homology groups in various fields $\mathbb{Z}_p$ with p prime. This approach adapts to persistent homology. We finally introduce the problem of *multi-field persistent homology*, which consists in computing persistence with various coefficient fields. We present an efficient algorithm for this problem in Chapter 8.

For simplicity, we focus in the following on simplicial homology. However, the approach applies to any type of cell complex.

7.1 Coefficients in Homology and Universal Coefficient Theorem

Torus vs Klein Bottle. We have introduced in Chapter 4 homology groups with coefficients either in $\mathbb{Z}$ or in a field $\Bbbk$. The integral homology groups provide a stronger topological information on the topological space under study, in the sense that one can compute the decompositions of the homology groups with $\Bbbk$ coefficients of a space $\mathbb{X}$ from the knowledge of the decompositions of the integral homology groups of $\mathbb{Z}$. In particular, the integral homology groups convey information about torsion. Torsion can be pictured geometrically as a twisting of the shape and happens frequently as

global topological feature in topological data analysis where, for example, Klein bottles appear naturally [26, 53]. Algebraically, torsion is characterized by cyclic subgroups of the integral homology groups. When computed with field coefficients, these subgroups may either vanish or appear as infinite, and consequently obfuscate the study of the topology of data. Consider the torus in Figure 7.1. The homology groups of the torus, with respectively

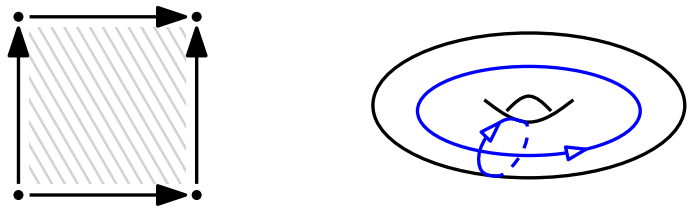


Figure 7.1: The torus is a surface embeddable in $\mathbb{R}^3$ that can be constructed by identifying the opposite oriented edges of the square on the left. Its 3-dimensional representation is pictured on the right.

$\mathbb{Z}$, $\mathbb{Z}_2$ and $\mathbb{Z}_p$ (for $p > 2$ prime) coefficients are:

$$\begin{cases} \mathbf{H}_0(\mathbb{Z}) \cong \mathbb{Z} & \mathbf{H}_0(\mathbb{Z}_2) \cong \mathbb{Z}_2 & \mathbf{H}_0(\mathbb{Z}_p) \cong \mathbb{Z}_p \\ \mathbf{H}_1(\mathbb{Z}) \cong \mathbb{Z} \oplus \mathbb{Z} & \mathbf{H}_1(\mathbb{Z}_2) \cong \mathbb{Z}_2 \oplus \mathbb{Z}_2 & \mathbf{H}_1(\mathbb{Z}_p) \cong \mathbb{Z}_p \oplus \mathbb{Z}_p \\ \mathbf{H}_2(\mathbb{Z}) \cong \mathbb{Z} & \mathbf{H}_2(\mathbb{Z}_2) \cong \mathbb{Z}_2 & \mathbf{H}_2(\mathbb{Z}_p) \cong \mathbb{Z}_p \end{cases}$$

Their is no torsion subgroups in the integral homology and the Betti numbers are identical for all coefficients considered. They describe, respectively, one connected component ($\beta_1 = 1$), two non-contractible independent loops (in blue, $\beta_1 = 2$) and one void ($\beta_2 = 1$).

Consider now the Klein bottle in Figure 7.2. The homology groups of the

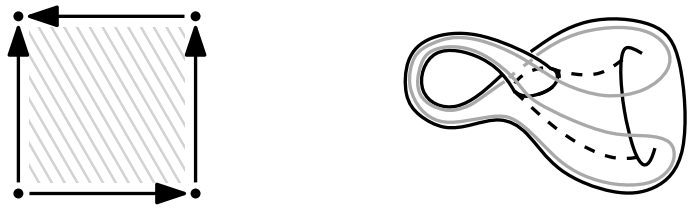


Figure 7.2: The Klein bottle is a surface embeddable in $\mathbb{R}^4$ that can be constructed by identifying the opposite oriented edges of the square on the left. One of its 3-dimensional projection is pictured on the right.

Klein bottle, with respectively $\mathbb{Z}$, $\mathbb{Z}_2$ and $\mathbb{Z}_p$ (for $p > 2$ prime) coefficients

are:

$$\left\{\begin{array}{lcl} \mathbf{H}_0(\mathbb{Z}) & \cong & \mathbb{Z} \\ \mathbf{H}_1(\mathbb{Z}) & \cong & \mathbb{Z} \oplus \mathbb{Z}_2 \\ \mathbf{H}_2(\mathbb{Z}) & \cong & 0 \end{array}\right. \quad \left\{\begin{array}{lcl} \mathbf{H}_0(\mathbb{Z}_2) & \cong & \mathbb{Z}_2 \\ \mathbf{H}_1(\mathbb{Z}_2) & \cong & \mathbb{Z}_2 \oplus \mathbb{Z}_2 \\ \mathbf{H}_2(\mathbb{Z}_2) & \cong & \mathbb{Z}_2 \end{array}\right. \quad \left\{\begin{array}{lcl} \mathbf{H}_0(\mathbb{Z}_p) & \cong & \mathbb{Z}_p \\ \mathbf{H}_1(\mathbb{Z}_p) & \cong & \mathbb{Z}_p \\ \mathbf{H}_2(\mathbb{Z}_p) & \cong & 0 \end{array}\right.$$

As expected, the Betti number β_0 remains unchanged and represents the number of connected components (see Chapter 4). More surprisingly, the Betti numbers differ in dimension 1 and 2 depending on the coefficients. When considering $\mathbb{Z}_p$ homology with $p > 2$, the torsion summand of $\mathbf{H}_1(\mathbb{Z})$ vanishes, and the Betti numbers are the one of integral homology. When considering $\mathbb{Z}_2$ coefficients, the order 2 element in dimension 1, characterized by the torsion summand in integral homology, appears as infinite which augments the free part of the decomposition of $\mathbf{H}_1(\mathbb{Z}_2)$ with one summand. This also induces the creation of a non-trivial homology feature in dimension 2.

As a consequence, the homology groups with $\mathbb{Z}_2$ coefficients are topological invariants too weak to distinguish a torus from a Klein bottle, which are not homeomorphic. We study in the next paragraph the relations between integral homology and homology with coefficients in a field.

Universal Coefficient Theorem. The *universal coefficient theorem* is a theorem of homological algebra that explains the relation between the integral homology groups and the homology groups with coefficients in a field. We present a corollary that gives a formula for computing the field Betti numbers from the Betti numbers and the torsion coefficients of the integral homology groups. Let $\mathbb{X}$ be a triangulable topological space whose homology groups are written:

$$\mathbf{H}_p(\mathbb{X},\mathbb{Z}) \cong \mathbb{Z}^{\beta_p(\mathbb{Z})} \bigoplus_{q \text{ prime}} \left(\mathbb{Z}_{q^{k_1}} \oplus \cdots \oplus \mathbb{Z}_{q^{k_{t(p,q)}}}\right) \text{ and } \mathbf{H}_p(\mathbb{X},\Bbbk) \cong \Bbbk^{\beta_p(\Bbbk)} \tag{7.1}$$

Let $\Bbbk$ be a field of characteristic q, like $\mathbb{Z}_q$. We have the following corollary [47] to the universal coefficient theorem:

Corollary 7.1 (to the Universal Coefficient Theorem)**.** *For $\beta_d(\mathbb{Z})$ and $\beta_d(\Bbbk)$ the Betti numbers of $\mathbf{H}_d(\mathbb{X},\mathbb{Z})$ and $\mathbf{H}_d(\mathbb{X},\Bbbk)$ respectively, and $t(j,q)$ the number of $\mathbb{Z}_{q^{k_i}}$ summands in the primary decomposition of $\mathbf{H}_j(\mathbb{X},\mathbb{Z})$, we have:*

$$\beta_d(\Bbbk) = \beta_d(\mathbb{Z}) + t(d,q) + t(d-1,q)$$

Going back to the torus, the Betti numbers do not depend on the coefficients because there is no torsion so all $t(d,q)$ are 0. For the Klein bottle, $t(1,2) = 1$ so we find that $\beta_1(\mathbb{Z}_2) = \beta_1(\mathbb{Z}) + t(1,2) = 2$ and $\beta_2(\mathbb{Z}_2) = \beta_2(\mathbb{Z}) + t(1,2) = 1$. In other words, a $\mathbb{Z}_{q^k}$ summand in the decomposition of $\mathbf{H}_d(\mathbb{Z})$ creates a $\mathbb{Z}_q$ summand in the decomposition of $\mathbf{H}_d(\Bbbk)$ and $\mathbf{H}_{d+1}(\Bbbk)$ if $\Bbbk$ has characteristic q, and nothing otherwise.

The homology groups with field coefficients are consequently entirely characterized by the integral homology groups. We reverse partially this property. Let $\{q_1, \cdots, q_r\}$ be the first r prime numbers and suppose that q_r is a strict upper bound on the prime divisors of the torsion coefficients of $\mathbb{X}$. According to Corollary 7.1, $\beta_d(\mathbb{Z}_{q_r}) = \beta_d(\mathbb{Z})$ for all dimensions d. Moreover, we have seen in Chapter 4 that there is no torsion in 0-homology because the homology group $\mathbf{H}_0(\mathbb{X}, \mathbb{Z})$ is always isomorphic to $\mathbb{Z}^{\beta_0}$, where β_0 is the number of connected components of the space $\mathbb{X}$. Hence, $t(0,q) = 0$ for every prime q.

Given the Betti numbers of $\mathbb{X}$ in all fields $\mathbb{Z}_{q_s}$, for all $1 \leq s \leq r$, we deduce from Corollary 7.1 the recurrence formula:

$$t(d, q_s) = \beta_d(\mathbb{Z}_{q_s}) - \beta_d(\mathbb{Z}_{q_r}) - t(d-1, q_s)$$

from which we compute the value of $t(d,q)$ for every dimension d and prime q. From the knowledge of the homology groups with coefficients in $\mathbb{Z}_{q_s}$, for all $1 \leq s \leq r$, we consequently infer, for any dimension d, the integral Betti numbers and the number $t(d,q)$ of $\mathbb{Z}_{q^{k_i}}$ summands in the primary decomposition of $\mathbf{H}_d(\mathbb{X}, \mathbb{Z})$. Note however that the powers k_i from the decomposition remain unknown.

7.2 Multi-Field Persistent Homology

We have seen in Chapter 4 that an algebraic limitation of persistent homology was that it had to be defined with coefficients in a field to admit an interval decomposition. The theory defines ideas of an invariant for filtrations and distances between persistence diagrams from this interval decomposition. Such decomposition is consequently necessary.

In order to define a stronger topological invariant for filtrations, a simple solution consists in computing persistent homology with different coefficient fields and infering the prime divisors of the torsion coefficient as presented in Section 7.1. We call the algorithmic problem of computing persistent homology with various coefficient fields *multi-field persistent homology*.

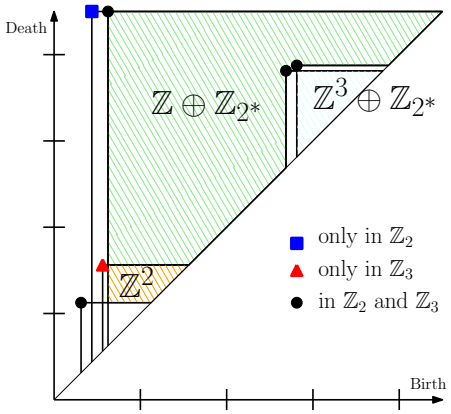


Figure 7.3: Multi-field persistence diagram of the most persistent features of $\mathbf{H}_1$ for a Rips complex reconstructing a Klein bottle. We obtain one feature that is very persistent in $\mathbb{Z}_2$ only and one very persistent in both $\mathbb{Z}_2$ and $\mathbb{Z}_3$. We consequently find the expression $\mathbb{Z} \oplus \mathbb{Z}_{2^*}$ for $\mathbf{H}_1$, which is as expected because the homology group of the Klein bottle in dimension 1 is $\mathbb{Z} \oplus \mathbb{Z}_2$. The remaining features are topological noise.

Topological Inference. We know from Chapter 1 and 4 that, under certain conditions where a sequence of complexes of a filtration $\mathbb{K}$ approximates an underlying topological space $\mathbb{X}$ at various scales, one can infer the homology of $\mathbb{X}$ from the persistent homology of $\mathbb{K}$. Specifically, for persistent homology with coefficients in any field $\Bbbk$, the number of pairs (i, j) with long persistence $|j - i|$ in dimension d corresponds to the Betti number of $\mathbf{H}_d(\mathbb{X}, \Bbbk)$.

As a consequence, given the persistent pairs of the persistent homology of $\mathbb{K}$ with various coefficient fields – more particularly $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ as defined earlier – one can infer the Betti numbers and prime divisors of the torsion coefficients of the integral homology of $\mathbb{X}$.

We proceed to this construction at all scales, and introduce a new representation of the persistence diagram, which is essentially a superimposition of the persistence diagrams of $\mathbb{K}$ for all coefficient fields.

Representation of Multi-Field Persistence Diagrams. Note first that, as the powers k_i of the decompositon in Equation (7.1) remain unknown, we simplify the notation and denote $\mathbb{Z}_{q^{\alpha_1}} \oplus \cdots \oplus \mathbb{Z}_{q^{\alpha_t}}$ by $\mathbb{Z}_{q^*}^{(t)}$, for any family of integers $\alpha_i \geq 1$. Hence, infering Betti numbers and prime divisors of torsion

coefficients consists in computing an expression:

$$\mathbf{H}_d(\mathbb{X}, \mathbb{Z}) \cong \mathbb{Z}^{\beta_d(\mathbb{Z})} \bigoplus_{q \text{ prime}} \mathbb{Z}_{q^*}^{(t(d,q))} \tag{7.2}$$

of the integral homology groups.

Consider a persistence diagram of homology with coefficients in $\Bbbk$. For any point (x, y) in the plane, the number of homology features that are created before time x and destroyed after time y is equal to the number of point in the quadrant (x, y). Hence, to every point (x, y) in the plane is attached a *persistent Betti number* $\beta_d^{x,y}$ for every dimension d.

Knowing the persistence diagram for all coefficient fields $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_s}$, we can consequently obtain, for every point (x, y), an expression in the form of Equation (7.2) from the persistent Betti numbers $\beta_d^{x,y}$. Constant persistent Betti numbers define polygonal areas pictured with color in Figure 7.3. We call the resulting diagram a *multi-field persistence diagram*. It consists essentially of the *superimposition* of the persistence diagrams in each coefficient field, but provides a readable and more informative description of the topology of a filtration. In particular, in topological ineference, one can read directly the Betti numbers and prime divisors of torsions coefficients of the integral homology groups of an approximated space.

We refer to Figure 7.3 for an example. It pictures the multi-field persistence diagram of the 1-homology of a simplicial complex $\mathbf{K}$ approximating a Klein bottle (for field coefficients $\mathbb{Z}_2$ and $\mathbb{Z}_3$). In particular, the confusion with a torus is resolved.

Remark 7.1. *This multi-field framework seems to give an interval-like decomposition of the persistence module with more general coefficients than the ones in a field. This is not in contradiction with the non-existence, in general, of an interval decomposition of the persistence module in $\mathbb{Z}$. This approach provides an inference of the integral homology groups for each complex $\mathbf{K}_i$ of a filtration and relies on the fact that all results in the theory of persistence are independent from the coefficients as long as they are from a field.*

This is not a universal coefficient theorem for persistent modules, with an algebraic relation between the decompositions of the whole persistence modules for homology with different coefficients. One the other side, the simplicity of the idea allows us to define a very efficient algorithm for multi-field persistence in the next Chapter.

Infering Torsion Coefficients with Garantees. We have previously assumed that we had a bound q_s on the prime divisors of the torsion coefficients of the complexes of the filtration. It is possible to compute such bound. Indeed, for a fixed simplicial complex $\mathbf{K}$, the torsion coefficients of $\mathbf{H}_d(\mathbf{K}, \mathbb{Z})$ are the eigenvalues of the matrix of the boundary operator $\partial_{d+1} : \mathbf{C}_{d+1}(\mathbf{K}, \mathbb{Z}) \to \mathbf{C}_d(\mathbf{K}, \mathbb{Z})$ (see [59]). Consequently, in order to bound the torsion coefficients of one single complex, one can use the Hadamard bound or finer estimates like the ovals of Cassini [41]. These bounds increase when we add simplices to a complex. Hence, for a filtration $\mathbf{K}_1 \longrightarrow \cdots \longrightarrow \mathbf{K}_n$ where each map is an inclusion, the Hadamard bound or the ovals of Cassini bound evaluated for the last simplicial complex $\mathbf{K}_n$ furnish a bound on the torsion coefficients of every simplex $\mathbf{K}_i$ of the filtration.

Bibliographic Notes.

We refer to [59] for more details on the universal coefficient theorem.

Computing persistent homology with different coefficients has been mentioned in the literature [70] in order to verify if a persisting feature was due to an actual hole (or high-dimensional equivalent) or to torsion (and consequently existed only for a certain coefficient field).

However, the work presented in this Chapter and the next one, based on [19], is, to the best of our knowledge, the first attempt to formalize the inference of torsion coefficients in the framework of persistent homology and describe an efficient algorithm to compute persistence with various coefficient fields.

Chapter 8

Modular Reconstruction Algorithm

In this chapter we introduce an algorithm to compute the persistent homology of a filtered complex with various coefficient fields in a single matrix reduction. The algorithm is output-sensitive in the total number of *distinct* persistent homological features in the diagrams for the different coefficient fields. The output of the algorithm is the multi-field persistence diagram.

For a complex of size n, we know that the persistence diagram for any coefficient field contains at most n pairs. When computing multi-field persistent homology for r coefficient fields, denote by n' the total number of *distinct* pairs in all persistence diagrams. In practice, the fields are $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ for the r first prime numbers $q_1, \cdots, q_r$, where q_r is an upper bound on the prime divisors of the torsion coefficients of the integral homology of the space, which are usually small. The quantity n' satisfies $n' \leq r \times n$ but in practice we observe that $n' \approx n$. We design in the following an algorithm for the multi-field persistence problem whose complexity depends mostly on n'. It is however an interesting open problem to exhibit a "natural" example where n' is must larger than n and/or the prime divisors of the torsion coefficients are big (for a fix n).

We build on this idea and describe an algorithm to compute persistent homology with various coefficient fields $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ in a single pass of the matrix reduction algorithm. To do so, we introduce a method we call *modular reconstruction* consisting in using the *Chinese Remainder Isomorphism* to encode an element of $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r}$ with an element of $\mathbb{Z}_{q_1 \cdots q_r}$. We describe algorithms to perform elementary row/column operations in a matrix with $\mathbb{Z}_{q_1 \cdots q_r}$ coefficients, corresponding to simultaneous elementary row/col-

umn operations in matrices with coefficients in the fields $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$. The method results in an algorithm with an output-sensitive complexity in the total number of *distinct* pairs in the echelon forms of the matrices with $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ coefficients, plus an overhead due to arithmetic operations on big numbers in $\mathbb{Z}_{q_1 \cdots q_r}$. The method is generic and applies to every algorithm for persistent homology. Finally, we describe how to infer the torsion coefficients of the integral homology using the *Universal Coefficient Theorem for Homology*.

We provide detailed experimental analysis of the algorithm and show, in particular, that on practical examples our method is substancially faster than the brute-force approach consisting in reducing separately r matrices with coefficients in $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$. It is important to note that the method does not pretend to scale to very large r, as the arithmetic complexity of operations in $\mathbb{Z}_{q_1 \cdots q_r}$ becomes problematic. Experiments show, however, that for very large r (up to 100000) our approach is still substancially faster than brute-force.

In computer algebra, working modulo small prime numbers is usually desirable in order to reduce coefficient growth. Our work goes the otherway around: we introduce tools to reduce a family of r matrices with coefficients in the fields $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ respectively, by means of a single reduction of a matrix with coefficients in $\mathbb{Z}_{q_1 \cdots q_r}$. We give theoretical and experimental evidence that, for reasonable values of r, our algorithm is significantly more efficient than the brute-force approach consisting in reducing the r matrices separately.

In the following, $\mathbb{Z}_n$ denotes the ring $(\mathbb{Z}_n, +, \times)$ for any integer $n \geq 1$. and $\mathbb{Z}_n^\times$ the subset of invertible elements for $\times$. If it exists, we denote the inverse of $x \in \mathbb{Z}_n$ by x^{-1}.

8.1 Matrix Reduction for Persistent Homology

We have presented in Chapter 4 the persistent homology and persistent cohomology algorithms at the algebraic level, which are both based on the maintainance of an encoding of the persistence modules. Working at the algebraic level is the approach chosen in this dissertation to give a uniform presentation of persistent homology (Chapter 4) and zigzag persistent homology (Chapter 11) algorithms. The validity of algorithms is proved by relating the computation to the interval decomposition of the (zigzag) persistence module.

The *modular reconstruction* method introduced in this chapter is of an arithmetic nature and works at the level of coefficients. As a consequence,

we have chosen to present the persistent homology algorithm as a matrix reduction. The atomic operation is the *elementary row/column operation*, and we can work directly with the coefficient of homology by working with the coefficients of the matrix. Note that the matrix reduction presentation is the original form of the persistent homology algorithm [43, 70], and the most popular because of its very simple description [42].

Additionally, we present the algorithms for persistent homology and multi field persistent homology with pseudo-code notations, so as to emphasize the low level operations and offer an easy comparison between the two algorithms.

Persistent Homology Algorithm. For clarity, we focus in this section on the persistent homology algorithm. The persistent cohomology algorithm is based on a similar reduction, and our approach adapts directly.

For a matrix $\mathbf{M}$, denote by col_j the j^{th} column of $\mathbf{M}$ and $\mathsf{col}_j[k]$ its k^{th} coefficient. Let $\text{low}(j)$ denote the row index of the lowest non-zero coefficient of col_j. If the column j is entirely zero, $\text{low}(j)$ is undefined. We say that $\mathbf{M}$ is in *reduced column echelon form* if $\text{low}(j) \neq \text{low}(j')$ for every non-zero columns col_j and $\mathsf{col}_{j'}$ with $j \neq j'$.

An *elementary column operation* on a matrix $\mathbf{M}$ with coefficients in a ring is one of the following three operation:

(i) exchange col_k and col_ℓ,

(ii) multiply col_k by -1,

(iii) replace col_k by $\mathsf{col}_k + x \times \mathsf{col}_\ell$, for an element x in the ring.

Elementary column operations are the atomic operation to reduce a matrix to column echelon form.

Let $\mathbf{K} = [\sigma_i]_{i=1\cdots n}$ be a filtered complex. Its boundary matrix $\mathbf{M}_\partial$ is the $n \times n$ matrix, with $\Bbbk$ coefficients, of the endomorphism ∂_* in the basis $\{\sigma_1, \cdots, \sigma_n\}$ of $\mathbf{C}_*(\mathbf{K}, \Bbbk) = \bigoplus_d \mathbf{C}_d(\mathbf{K}, \Bbbk)$. The basis is ordered according to the filtration. It is a matrix with $\{-1, 0, 1\}$ coefficients, where 0 and 1 are the identities for $+$ and $\times$ in $\Bbbk$ respectively, and -1 is the inverse of 1 in $\Bbbk$. The persistent homology algorithm consists in a left-to-right reduction to column echelon form of $\mathbf{M}_\partial$: we denote by $\mathbf{R}$ the matrix we reduce, with columns col_j, which is initially equal to $\mathbf{M}_\partial$. The algorithm returns the *(indexed) persistence diagram*, which is the set of pairs $\{(\text{low}(j), j)\}$ in the reduced column echelon form of the matrix. The reduced form of the matrix is not unique, but the pairs (i, j) such that $i = \text{low}(j)$ in the column echelon

Algorithm 8.1: `Persistent-homology`

Data: Boundary matrix $\mathbf{R} \leftarrow \mathbf{M}_\partial$, persistence diagram $\mathcal{P} \leftarrow \emptyset$
Output: Persistence diagram $\mathcal{P} = \{(i,j)\}$
for $j = 1, \cdots, n$ **do**
  while *there exists* $j' < j$ *with* $\mathrm{low}(j') = \mathrm{low}(j)$ **do**
    $k \leftarrow \mathrm{low}(j)$;
    $\mathsf{col}_j \leftarrow \mathsf{col}_j - \left(\mathsf{col}_j[k] \times \mathsf{col}_{j'}[k]^{-1}\right) \cdot \mathsf{col}_{j'}$;
  end
  if $\mathsf{col}_j \neq 0$ **then** $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\mathrm{low}(j), j)\}$;
end

form are [42]. The algorithm requires $O(n^3)$ arithmetic operations in $\Bbbk$. See Algorithm 8.1.

This reduction is correct because it implements the persistent homology algorithm of Chapter 4. Indeed, after iteration i of the `for` loop, the i first columns of the matrix maintain a chain related to an encoding $\{\widehat{\tau}_1, \cdots, \widehat{\tau}_i\}$, $F \sqcup G \sqcup H$ and $\mathcal{P}$ of the filtration $\emptyset = \mathbf{K}_0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_i} \mathbf{K}_i$. Specifically, let j be the index of a column, with $1 \leq j \leq i$:

1. if $j \in F$ or $j \in G$, then $\mathsf{col}_j = 0$,
2. if $j \in H$, then let $(g, j) \in \mathcal{P}$. The column col_j represents the chain $\partial\widehat{\tau}_j = \widehat{\tau}_g$, and its lowest non-zero element is at index g, because τ_g is the leading term of $\widehat{\tau}_g$ by definition of an encoding.

Entering the $(i+1)^{\text{st}}$ iteration of the procedure, the `while` loop corresponds to Algorithm 4.1 called with input `Reduction(col,`G`)`, where `col` is a column corresponding to chain $\partial\tau_{i+1}$. Indeed, the non-zero columns of the matrix among the i first columns are the ones representing the chains $\{\widehat{\tau}_g\}_{g \in G}$.

8.2 Modular Reconstruction for Matrix Reduction

The *modular reconstruction* method consists in representing a set of coefficients in distinct fields by their product. We use this method to define arithmetic operations in the different fields simultaneously, then to define elementary column operations and ultimately an algorithm for multi-field persistence. We first present a particular case of the *chinese remainder theorem* [46]:

Theorem 8.1 (Chinese Remainder Theorem)**.** *For a family $\{q_1, \cdots, q_r\}$ of r distinct prime numbers, there exists a ring isomorphism $\psi : \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r} \to \mathbb{Z}_{q_1 \cdots q_r}$. The isomorphisms ψ and ψ^{-1} can be computed in $O(r)$ arithmetic operations in $\mathbb{Z}_{q_1 \cdots q_r}$.*

Proof. For all $1 \leq s \leq r$, there exists U_s such that $U_s \bmod q_t = 1$ if $s = t$ and 0 otherwise. One can easily verify that ψ and ψ^{-1} realize the isomorphism:

$$\begin{array}{rccc} \psi : & \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r} & \to & \mathbb{Z}_Q \\ & (u_1, \cdots, u_r) & \mapsto & (u_1 U_1 + \cdots + u_r U_r) \bmod Q \\ \psi^{-1} : & (x \bmod q_1, \cdots, x \bmod q_r) & \leftarrow\!\shortmid & x \end{array}$$

□

Let $[r]$ refer to the set $\{1, \cdots, r\}$. For a family of r distinct prime numbers $\{q_1, \cdots, q_r\}$, and a subset of indices $S \subseteq [r]$, Q_S refers to $\prod_{s \in S} q_s$, and we write simply $Q = Q_{[r]}$. We define the function $\psi_S : \prod_{s \in S} \mathbb{Z}_{q_s} \to \mathbb{Z}_{Q_S}$ realizing the isomorphism of the Chinese Remainder Theorem for the subset $\{q_s\}_{s \in S}$ of primes, and we write simply ψ for $\psi_{[r]}$. For a family of elements $u_s \in \mathbb{Z}_{q_s}, s \in S$, we denote the corresponding $|S|$-uplet $(u_s)_{s \in S} \in \prod_{s \in S} \mathbb{Z}_{q_s}$.

Finally, we recall *Bezout's lemma* [46]:

Lemma 8.1 (Bezout's Lemma)**.** *For two integers a and b, not both 0, there exist integers v and w such that:*

$$va + wb = \gcd(a, b)$$

where $\gcd(a, b)$ is the greatest common divisor of a and b, and $|v| < |b/\gcd(a, b)|$ and $|w| < |a/\gcd(a, b)|$.

The Bezout's coefficients (v, w) can be computed with the extended Euclidean algorithm [46].

Elementary Column Operations. We are given a family of distinct prime numbers $\{q_1, \cdots, q_r\}$, and their product $Q = q_1 \cdots q_r$. Let $\mathbf{M}_Q$ be a matrix with coefficients in the ring $\mathbb{Z}_Q$. Denoting $\psi^{-1} : \mathbb{Z}_Q \to \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r}$ the isomorphism of the chinese remainder theorem, and $\pi_s : \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r} \to \mathbb{Z}_{q_s}$ the projection on the s^{th} coordinate, we call *projection of $\mathbf{M}_Q$ onto $\mathbb{Z}_{q_s}$*, denoted $\mathbf{M}_Q(\mathbb{Z}_{q_s})$, the matrix $\mathbf{M}_{q_s}$ with $\mathbb{Z}_{q_s}$ coefficients, obtained by applying $\pi_s \circ \psi^{-1}$ to each coefficient of $\mathbf{M}_Q$. Conversely, given r $(m \times m)$-matrices $\mathbf{M}_{q_1}, \cdots, \mathbf{M}_{q_r}$ with coefficients in $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ respectively, there exists a

unique matrix $\mathbf{M}_Q$ with $\mathbb{Z}_Q$ coefficients such that for every s the projection of $\mathbf{M}_Q$ onto $\mathbb{Z}_{q_s}$ is $\mathbf{M}_{q_s}$. This is simply a matrix version of the chinese remainder theorem.

We have introduce elementary column operations of type (i), (ii) and (iii) in Section 8.1. For an elementary column operation of type $(*)$ (*ie* of type (i), (ii) or (iii)) applied to columns k (and ℓ)), we denote by $(*)\circ\mathbf{M}_q$ the result of applying $(*)$ to $\mathbf{M}_q$. In this section, we introduce algorithms to run elementary column operations simultaneously on the matrices $(\mathbf{M}_{q_s})_{s=1,\cdots,r}$ by performing "partial column operations" on $\mathbf{M}_Q$. Specifically, for an elementary column operation $(*)$ and a subset of indices $S \subseteq [r]$, we call *partial column operation* on $\mathbf{M}_Q$ the operation transforming $\mathbf{M}_Q$ into $\mathbf{M}'_Q$ such that: for every $s \notin S$, the projection onto $\mathbb{Z}_{q_s}$ satisfies $\mathbf{M}_Q(\mathbb{Z}_{q_s}) = \mathbf{M}'_Q(\mathbb{Z}_{q_s}) = \mathbf{M}_{q_s}$ and for every $s \in S$, the projection onto $\mathbb{Z}_{q_s}$ satisfies $\mathbf{M}'_Q(\mathbb{Z}_{q_s}) = (*) \circ \mathbf{M}_{q_s}$.

As the correspondence $\psi : \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r} \to \mathbb{Z}_Q$ is a ring homomorphism, it satisfies the properties: $\psi(u_1, \cdots, u_r) + \psi(v_1, \cdots, v_r) \times \psi(w_1, \cdots, w_r) = \psi(u_1 + v_1 \times w_1, \cdots, u_r + v_r \times w_r)$ and we can compute addition and multiplication componentwise in $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_r}$ using addition and multiplication in $\mathbb{Z}_Q$. In order to compute partial column operations, we first introduce the set of *partial identities*, which are coefficients that allow us to proceed to the partial column operations of type (i) and (ii). Secondly, as the rings $\mathbb{Z}_{q_s}$ are fields, we need to compute the multiplicative inverse of an element, that is used as multiplicative coefficient x in elementary column operation (iii). As $\mathbb{Z}_Q$ is not a field, inversion is not possible, and we introduce the concept of *partial inverse* to overcome this difficulty. In the following, the term *arithmetic operation* refers to any operation $\{+, -, \times, \gcd(\cdot,\cdot), \cdot \bmod Q_S, \text{Extended Euclidean algorithm}\}$ on integer smaller than Q. Note they do not have constant time complexity for large Q.

Partial Identity and Partial Inverse. Given a subset of indices $S \subseteq [r]$, we define the *partial identities* wrt S, denoted L_S, by $L_S = \psi(\delta_{1,S}, \cdots, \delta_{r,S})$ where the symbol $\delta_{t,S} \in \mathbb{Z}_{q_t}$ is equal to 1 if $t \in S$ and to 0 otherwise. For any $S \subseteq [r]$, the partial identity L_S can be constructed in $O(r)$ arithmetic operations in $\mathbb{Z}_Q$ by evaluating ψ on $(\delta_{1,S}, \cdots, \delta_{r,S})$. However, it is important to notice that if $S = [r]$, $L_{[r]} = \psi(1, \cdots, 1) = 1$, because ψ is a ring isomorphism, and $L_{[r]}$ is computed in time $O(1)$.

Knowing the partial identities, we can implement the partial column operations (i) and (ii) for a set of indices S. Partial column operation (i) is implemented by replacing column k by $(\mathsf{col}_k \times L_{[r]\setminus S} + \mathsf{col}_\ell \times L_S)$ and column ℓ by $(\mathsf{col}_\ell \times L_{[r]\setminus S} + \mathsf{col}_k \times L_S)$. Partial column operation (ii) is implemented

by multiplying column k by $L_{[r]} - 2 \times L_S$.

We define now the *partial inverse* of an element in the ring $\mathbb{Z}_Q$:

Definition 8.1 (Partial Inverse)**.** *Given a set $S \subseteq [r]$ of indices, the* partial inverse *of $x = \psi(u_1, \cdots, u_r)$ with regard to S is the element $\overline{x}^S \in \mathbb{Z}_Q$:*

$$\overline{x}^S = \psi(\overline{u_1}^S, \cdots, \overline{u_r}^S), \text{ with } \overline{u_s}^S = \begin{cases} u_s^{-1} & \text{if } s \in S \text{ and } u_s \in \mathbb{Z}_{q_s}^\times \\ 0 & \text{otherwise} \end{cases}$$

Using elementary algebra, we prove:

Proposition 8.1 (Partial Inverse Construction)**.** *For $x = \psi(u_1, \cdots, u_r) \in \mathbb{Z}_Q$ and $S \subseteq [r]$,*

(1) $\gcd(x, Q_S) = Q_R$ *for some $R \subseteq S$ and for all $s \in S$, u_s is invertible in $\mathbb{Z}_{q_s}$ iff $s \notin R$; we denote $T = S \setminus R$.*

(2) The Bezout's identity for x and Q_T gives $vx + wQ_T = 1$, where v satisfies $v \bmod Q_T = \psi_T((u_s^{-1})_{s \in T})$

(3) $\overline{x}^S = \left[\psi_T((u_s^{-1})_{s \in T}) \times L_T \bmod Q\right] \in \mathbb{Z}_Q$*, where L_T is the partial identity* wrt T.

Proof. (1): The gcd of x and Q_S divides Q_S so $\gcd(x, Q_S) = Q_R$ for some $R \subseteq S$, and for every index $s \in S$, q_s divides x (denoted $q_s \mid x$) iff $s \in R$. According to the Chinese remainder theorem, for any $s \in T = S \setminus R$, $u_s = x \bmod q_s \neq 0$ as $q_s \nmid x$. As $\mathbb{Z}_{q_s}$ is a field, its unique non invertible element is 0 and consequently u_s is invertible. Conversely, as $q_t \mid x$ for $t \in R$, $x \bmod q_t = u_t = 0$ is non invertible.

(2): First note that $x \bmod Q_T = \psi_T((u_t)_{t \in T}) \in \mathbb{Z}_{Q_T}$. Indeed, as $q_t \mid Q_T, \forall t \in T$, we have $(x \bmod Q_T) \bmod q_t = x \bmod q_t = u_t$. By definition of T, $\gcd(x, Q_T) = 1$ and so the Bezout's lemma gives $vx + wQ_T = 1$. Applying $(\cdot \bmod Q_T)$ to both sides of the equality gives $(v \bmod Q_T)\psi_T((u_t)_{t \in T}) = 1$ and consequently, for every q_t, $t \in T$, we have $((v \bmod Q_T) \bmod q_t)u_t = 1$ and the result follows.

(3): Let L_T be the partial identity *wrt* T. We form the product $\widetilde{x} = \left[\psi_T((u_t^{-1})_{t \in T}) \times L_T \bmod Q\right]$ and evaluate it modulo q_s. For any index $s \in [r]$, $\widetilde{x} \bmod q_s = \left[(\psi_T((u_t^{-1})_{t \in T}) \bmod q_s) \times (L_T \bmod q_s)\right] \bmod q_s$. If $s \notin T$, then $L_T \bmod q_s = 0$ and $\widetilde{x} \bmod q_s = 0$. If $s \in T$, then $L_T \bmod q_s = 1$, $\psi_T((u_t^{-1})_{t \in T}) \bmod q_s = u_s^{-1}$ and consequently $\widetilde{x} \bmod q_s = u_s^{-1}$. Thus, $\widetilde{x}$ satisfies the definition of $\overline{x}^S$, the partial inverse of x *wrt* S. □

We directly deduce an algorithm to compute the partial inverse of x *wrt* S if Q_S is given: compute $Q_R = \gcd(x, Q_S)$ and $Q_T = Q_S/Q_R$, then v using the extended Euclidean algorithm and finally $\overline{x}^S = (v \bmod Q_T) \times L_T \bmod Q$. Computing the partial identity L_T requires $O(r)$ arithmetic operations in $\mathbb{Z}_Q$, but is constant if $T = [r]$, which happens iff $S = [r]$ and x is invertible in $\mathbb{Z}_Q$. Consequently, computing $\overline{x}^S$ requires $O(r)$ arithmetic operations in general, but only $O(1)$ arithmetic operations in the later case.

Modular Reconstruction for Multi-Field Persistent Homology Let $\mathbf{K}$ be a filtered complex with n simplices. Define $\mathbf{M}_\partial(\mathbb{Z}_{q_s})$ to be the $n \times n$ boundary matrix of $\mathbf{K}$ with $\mathbb{Z}_{q_s}$ coefficients. Define $\mathbf{M}$ to be the $n \times n$ matrix with $\mathbb{Z}_Q$ coefficients such that the projection of $\mathbf{M}$ onto $\mathbb{Z}_{q_s}$ is equal to $\mathbf{M}_\partial(\mathbb{Z}_{q_s})$, for all $s \in [r]$. Note that the matrices $\mathbf{M}$ and $\mathbf{M}_\partial(\mathbb{Z}_{q_s})$, for any s, are identical matrices in the sense that they contain 0, 1 and -1 coefficients at the same positions, where 0, 1 and -1 refer respectively to elements of $\mathbb{Z}_Q$ and $\mathbb{Z}_{q_s}$.

We reduce a matrix $\mathbf{R}$ which is initially equal to $\mathbf{M}$. Denote by col_j the j^{th} column of $\mathbf{R}$. Define $\text{low}(j, Q_S)$ to be the index of the lowest element of col_j such that $\mathsf{col}_j[\text{low}(j, Q_S)] \mod Q_S \neq 0$. In particular, $\text{low}(j, q_s)$ is equal to the index of the lowest non-zero element of column j in the projection $\mathbf{R}(\mathbb{Z}_{q_s})$. After iteration j, we say that the columns $\mathsf{col}_1, \cdots, \mathsf{col}_j$ are *reduced*. We maintain, for every reduced column col_j, the collection of "lowest indices" i as a set $\mathcal{L}(j) = \{(i, Q_S)\}$ satisfying:

- For every $(i, Q_S) \in \mathcal{L}(j)$, $i = \text{low}(j, Q_S)$
- For every $(i, Q_S), (i', Q_{S'}) \in \mathcal{L}(j)$, either $i = i'$ and $S = S'$, or $i \neq i'$ and $S \cap S' = \emptyset$
- $\cup_{(i,Q_S)\in\mathcal{L}(j)} S = [r]$

The algorithm returns the set of triplets $\mathcal{P} = \{(i, j, Q_S)\}$ such that $i = \text{low}(j)$ in the column echelon form of the matrix $\mathbf{M}_\partial(\mathbb{Z}_{q_s})$ iff $s \in S$, or, equivalently, $(i, Q_S) \in \mathcal{L}(j)$ once col_j has been reduced. This is a compact encoding of the multi-field persistence diagram of the filtered complex.

The modular reconstruction algorithm for persistent homology is described in Algorithm 8.2. The $\{\mathcal{L}(j)\}_j$ form an index table that we maintain implicitely. At iteration j of the `for` loop, we use Q_S for the product of all prime numbers $\prod_{s\in S} q_s$ for which the column j in $\mathbf{R}(\mathbb{Z}_{q_s})$ has not yet been reduced.

Algorithm 8.2: `Modular-reconstruction`

Data: Matrix $\mathbf{R} = \mathbf{M}$
Output: Multi-field persistence diagram $\mathcal{P} = \{(i, j, Q_S)\}$
for $j = 1, \cdots, n$ **do**
  $Q_S \leftarrow Q_{[r]}$;
  while $\mathrm{low}(j, Q_S)$ *is defined* **do**
    $k \leftarrow \mathrm{low}(j, Q_S)$; $Q_T \leftarrow Q_S / \gcd(\mathsf{col}_j[k], Q_S)$;
    while *there exists* $j' < j$ *with* $(i, Q_{T'}) \in \mathcal{L}(j')$ *satisfying* $[i = \mathrm{low}(j, Q_S)$ *and* $\gcd(Q_{T'}, Q_T) > 1]$ **do**
      $\mathsf{col}_j \leftarrow \mathsf{col}_j - \left(\mathsf{col}_j[k] \times \overline{\mathsf{col}_{j'}[k]}^{T}\right) \cdot \mathsf{col}_{j'}$;
      $Q_T \leftarrow Q_T / \gcd(Q_{T'}, Q_T)$;
    end
    if $Q_T \neq 1$ **then** $\mathcal{P} \leftarrow \mathcal{P} \cup \{(k, j, Q_T)\}$; $Q_S \leftarrow Q_S / Q_T$;
    ;
  end
end

Correctness of the Multi-Field Algorithm. First, note that all operations processed on $\mathbf{R}$ correspond to left-to-right elementary column operations in the matrices $\mathbf{R}(\mathbb{Z}_{q_s})$ for all $s \in [r]$. By definition of the partial inverse, the column operation in line 7 can only reduce the value of $\mathrm{low}(j, Q_S)$. Moreover, one iteration of the `while` loop in line 3 either strictly reduces Q_S by dividing it by Q_T (in line 10) or set $(\mathsf{col}_j[k] \mod Q_S)$ to zero, hence reducing strictly $\mathrm{low}(j, Q_S)$. The later case happens when Q_T is set to 1 in line 8. Consequently, the algorithm terminates.

We prove recursively, on the numbers of columns, that each of the matrix $\mathbf{R}(\mathbb{Z}_{q_s})$ gets reduced to column echelon form. We fix an arbitrary field $\mathbb{Z}_{q_s}$: suppose that the $j-1$ first columns of $\mathbf{R}(\mathbb{Z}_{q_s})$ have been reduced at the end of iteration $j-1$ of the `for` loop in line 1. We prove that at the end of the j^{th} iteration of the `for` loop in line 1, the j first columns of the matrix $\mathbf{R}(\mathbb{Z}_{q_s})$ are reduced. Consider two cases. First suppose there is a triplet $(i, j, Q_T) \in \mathcal{P}$ for some $i < j$ and Q_T satisfying $q_s \mid Q_T$. This implies that the algorithm exits the `while` loop in line 5 with $q_s \mid Q_S$ (because by definition of Q_T, in line 4, $Q_T \mid Q_S$) and there is no $j' < j$ such that $[\mathrm{low}(j', Q_{T'}) = \mathrm{low}(j, Q_S)$ and $\gcd(Q_{T'}, Q_T) > 1]$. This in particular implies that there is no $j' < j$ such that $\mathrm{low}(j', q_s) = \mathrm{low}(j, q_s)$ and column j is reduced in $\mathbf{R}(\mathbb{Z}_{q_s})$.

Secondly, suppose that there is no such pair (i, j, Q_T) in $\mathcal{P}$, with q_s dividing Q_T. Consequently, during all the computation of the `while` loop in line 3, $q_s \mid Q_S$. When exiting this `while` loop, $\text{low}(j, Q_S)$ is undefined, implying in particular that $\text{low}(j, q_s)$ is undefined and column j of $\mathbf{R}(\mathbb{Z}_{q_s})$ is zero, and hence reduced.

8.3 Output-Sensitive Complexity Analysis

Arithmetic Complexity Model for Large Integers. During the reduction algorithm we perform arithmetic operations on big integers, for which we describe a complexity model [46]. Suppose that on our architecture, a memory word is encoded on w bits (on modern architectures, w is usually 64). Computer chips contain Arithmetic Logic Units that allow arithmetic operations on a 1-memory word integer in $O(1)$ machine cycles. Let the *length* of an integer z be defined by: $\lambda(z) = \lfloor \log_2 z/\mathsf{w} \rfloor + 1$, i.e. by the number of memory words necessary to encode z. We express the arithmetic complexity as a function of the length. For any positive integer z of length $\lambda(z) = \mathsf{B}$, operations in $\mathbb{Z}_z$ cost $\mathsf{A}_+(z) = O(\mathsf{B})$ for addition, $\mathsf{A}_\times(z) = O(M(\mathsf{B}))$ for multiplication and $\mathsf{A}_\div(z) = O(M(\mathsf{B}) \log \mathsf{B})$ for (extended) Euclidean algorithm, inversion and division, where $M(n)$ is a monotonic upper bound on the number of word operations necessary to multiply two integers of length B. By a result of [45], $M(\mathsf{B}) = O(\mathsf{B} \log \mathsf{B}\ 2^{O(\log^* \mathsf{B})})$, where $\log^* n$ is the iterated logarithm of n. In the following, we write $\mathsf{A}(z)$ for a bound on the complexity of arithmetic operations on integers smaller than z.

In the case of multi-field persistent homology, we are interested in the value of λ for an element in $\mathbb{Z}_Q$, $Q = q_1 \cdots q_r$, in the case where $\{q_1, \cdots, q_r\}$ are the first r prime numbers. We know [64] that $\ln Q < 1.01624 q_r$ and $q_r < r \ln(r \ln r)$ for $r \geq 6$. Consequently, $\lambda(Q) < \lfloor 1.46613 r \ln(r \ln r)/\mathsf{w} \rfloor + 1$. Note that $\lambda(Q) \ll r$ for $r \ln r \ll e^{\mathsf{w}}$, which is a reasonable assumption.

Complexity of the Modular Reconstruction Algorithm. Let $\mathbf{K}$ be a filtered complex of size n. The persistent homology algorithm described in Section 8.1, applied on $\mathbf{K}$ with coefficients in a field $\Bbbk$, requires $O(n^3)$ operations in $\Bbbk$. For a field $\mathbb{Z}_q$ these operations take constant time and the algorithm has complexity $O(n^3)$. The output of the algorithm is the persistence diagram, which has size $O(n)$ for any field.

For a set of prime numbers $\{q_1, \cdots, q_r\}$, let n' be the total number of distinct pairs in all persistence diagrams for the persistent homology of $\mathbf{K}$ with coefficient fields $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$. We express the complexity of the modular

reconstruction algorithm in terms of the size of its output (i.e. the multi-field persistence diagram of size n'), the number of fields r and the arithmetic complexity $\mathsf{A}(Q)$. First, note that, for a column j' in the reduced form of $\mathbf{R}$, the size of $\mathcal{L}(j')$ is equal to the number of triplets of the multi-field persistence diagram with death index j'; denote this quantity by $|\mathcal{L}(j')|$. Hence, when reducing column $j > j'$, the column $\mathsf{col}_{j'}$ is involved in a column operation $\mathsf{col}_j \leftarrow \mathsf{col}_j + \alpha \cdot \mathsf{col}_{j'}$ at most $|\mathcal{L}(j')|$ times. Consequently, reducing col_j requires $O(\sum_{j'<j} |\mathcal{L}(j')|) = O(n')$ column operations. There is a total number of $O(m \times m')$ column operations to reduce the matrix, each of them being computed in time $O(n \times \mathsf{A}(Q))$.

Computing the partial inverse of an element $x \in \mathbb{Z}_Q$ takes time $O(r \times \mathsf{A}(Q))$ in the general case, and only $O(\mathsf{A}(Q))$ if x is invertible in $\mathbb{Z}_Q$. The partial inverse of an element $x = \mathsf{col}_j[k]$ is computed only if there is a pair $(k, Q_T) \in \mathcal{L}(j)$. This element is not invertible in $\mathbb{Z}_Q$ iff $|\mathcal{L}(j)| > 1$. There are consequently $O(|n' - n|)$ non-invertible elements x that are at index $\mathrm{low}(j, Q_T)$ in some column j, for some Q_T. If we store the partial inverses when we compute them, the total complexity for computing all partial inverses in the modular reconstruction algorithm is $O((n + r \times (n' - n) \times \mathsf{A}(Q))$. We conclude that the total cost of the modular reconstruction algorithm for multi-field persistent homology is $O(\left[r \times (n' - n) + n^2 n'\right] \times \mathsf{A}(Q)) = O(\left[r \times (n' - n) + (n')^3\right] \times \mathsf{A}(Q))$, while the brute-force algorithm, consisting in computing persistence separately for every field $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$ requires $O(r \times n^3)$ operations.

Note that asymptotically in r, one arithmetic operation in $\mathbb{Z}_{Q_{[r]}}$ becomes more costly than r distinct arithmetic operations in $\mathbb{Z}_{q_1}, \cdots, \mathbb{Z}_{q_r}$, in which case the modular reconstruction approach developed in this chapter becomes worse than the brute-force algorithm (even when n' and n are close). This however happens for extremely big values of r (see Chapter 9) and had, in particular, no incidence on all experiments we have run in the next Chapter.

Remark 8.1. *On all datasets considered in our experiments, we have found no example where n' was significantly bigger than n. However, it is unclear whether many "short-lived torsion" might appear in general. This is not an issue as the persistent homology algorithm admits a complexity analysis depending on the length of bars in the output persistence barcode [42]. In this analysis, the cost for computing a pair (i, j) in the persistence diagram is function of the quantity $|j - i|$. This analysis adapts naturally to the modular reconstruction approach, showing that short-bars induce a smaller cost for computation.*

Chapter 9

Performance of Modular Reconstruction

In this section, we report on the performance of the modular reconstruction algorithm for multi-field persistent homology. We compute the persistent homology of Rips complexes built on a variety of both real and synthetic datasets. We use the *compressed annotation matrix* implementation of persistence cohomology presented in Chapter 5. The coefficients of the matrix are represented with the multi-precision integers of the `GMP` library [2]. We compare the performance of the modular reconstruction algorithm for multi-field persistent homology with the brute-force approach. By brute-force, we mean that we compute persistence independently in each field of coefficients. In this later case, we use standard `C++ int` to represent the coefficients. Note that, comparatively, memory allocation and arithmetic operations are slower with `GMP` integers, even if they fit in one memory word.

The datasets are listed in Figure 9.1 with the size of point sets $|\mathcal{P}|$, the threshold ρ for the Rips complex and the size of the complex $|\mathcal{K}|$. The values T_r for $r \in \{1, 50, 100, 200\}$ refers to the running time of the modular reconstruction algorithm for the r first prime numbers, and R_r refers to the ratio between the brute-force approach and the modular reconstruction algorithm.

9.1 Time Performance

Surprisingly, we have observed that, on all experiments, the number of differences between persistence diagrams with various coefficient fields was extremely small. Let n be the number of points in a persistent diagram and

Data	$\lvert\mathcal{P}\rvert$	ρ	$\lvert\mathcal{K}\rvert$	T_1	R_1	T_{50}	R_{50}	T_{100}	R_{100}	T_{200}	R_{200}
Bud	49,990	0.09	$127 \cdot 10^6$	96.3	0.51	110.3	22.2	115.9	42.3	130.7	75.0
Bro	15,000	0.04	$142 \cdot 10^6$	123.8	0.41	143.5	17.8	150.2	34.0	174.5	58.5
Cy8	6,040	0.8	$193 \cdot 10^6$	121.2	0.63	134.6	28.2	139.2	54.6	148.8	102.2
Kl	90,000	0.25	$114 \cdot 10^6$	78.6	0.52	89.3	23.0	93.0	44.1	105.2	78.0
S3	50,000	0.65	$134 \cdot 10^6$	125.9	0.40	145.7	17.2	152.6	32.8	177.6	50.3

Figure 9.1: Timings of the modular reconstruction algorithm vs brute-force.

n' the total number of points in the multi-field persistence diagram. The quantity $n' - n$, that appears in the complexity analysis of the modular reconstruction algorithm, can be considered as a very small constant in our experiments (≤ 10). We have also observed that these differences between persistence diagrams appeared for small prime numbers q_s.

Figure 9.1 presents the timings of the modular reconstruction approach for a variety of simplicial complexes ranging between 114 and 193 million simplices. We note that from $r = 1$ to $r = 200$ prime numbers, the time for computing multi-field persistence using the modular reconstruction approach only increases by 23 to 41%, when the brute-force approach requires about 200 times more time. This difference appears in the speedup expressed by the ratio R_r. For $r = 1$, the modular reconstruction approach is about twice

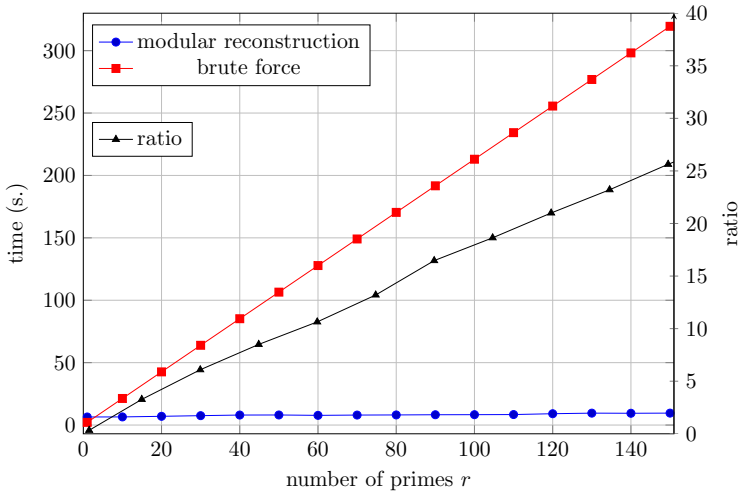


Figure 9.2: Timings for the modular reconstruction algorithm and brute force.

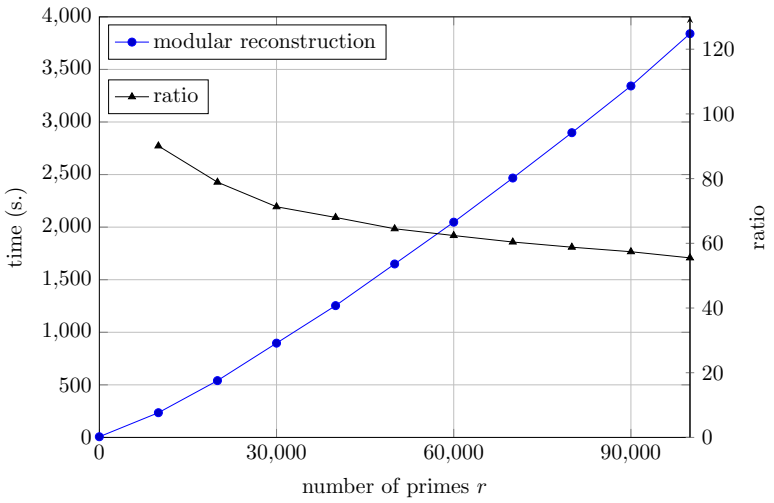


Figure 9.3: Asymptotic behavior of modular reconstruction and brute force.

slower than the standard persistent homology algorithm in a single field, because modular reconstruction is a more complex procedure and deals, in our implementation, with `GMP` integers that are slower than the classic `int` used in the standard persistent homology algorithm. However, this difference fades away as soon as $r > 1$ and the modular reconstruction is significantly more efficient than the brute-force algorithm: it is, in particular, between 50.3 and 102.2 times faster for $r = 200$.

Figures 9.2 and 9.3 present the evolution of the running time of the modular reconstruction approach and the brute-force approach for an increasing number of fields r (using the first r prime numbers). Persistence is computed for a Rips complex built on a set of 10000 points sampling a Klein bottle, which contains torsion in its integral homology, resulting in a simplicial complex of 6.14 million simplices. We analyze the result in terms of the complexity analysis of Section 8.3. The quantity m is fixed and m' is fixed for $r \geq 2$. The complexity of the brute-force algorithm is $O(r \times m^3)$ and we indeed observe a linear behavior when r increases. The complexity of the modular reconstruction approach is $O(\left[r \times (m' - m) + m'^3\right] \mathsf{A}(Q_{[r]}))$. The part $r \times (m' - m)$ of the complexity is negligeable because $m' - m$ is extremely small. For medium values of r (≤ 150), like in Figure 9.2, the arithmetic complexity $O(\mathsf{A}(Q_{[r]}))$ increases very slowly because $\lambda(Q_{[r]}) = \left\lfloor \log_2 Q_{[r]} / \mathsf{w} \right\rfloor + 1$ increases slowly. We consequently observe a very slow increasing of the time complexity compared to the one of brute-force.

9.2 Asymptotic Behavior

Figure 9.3 describes the asymptotic behavior of the modular approach, where the arithmetic operations become costly. We observe that the timings for the modular reconstruction approach follow a convex curve. The convexity comes from the growth of $\lambda(Q_{[r]})$, which is asymptotically $\Theta(r \log r))$ [64]. However, the increasing of the slope is very slow: all along this experiment, we have been unable to reach a value of r for which the modular approach is worse than the brute-force approach. For readability, the timings for the brute-force approach are implicitly represented through their ratio with the modular approach: all along the experiment, for $10000 \leq r \leq 100000$, the modular approach is between 55 and 90 times faster. Based on a linear interpolation of the timings for the brute-force approach, and a polynomial interpolation of the modular reconstruction timings, we expect the modular reconstruction to become worse than brute-force for a number of primes r bigger than 4.9 million. In the case of multi-field persistent homology however, there is no need to take r bigger than 200, because r is related to torsion coefficients, which are small in practice.

Part IV

Zigzag Persistence via Reflections and Transpositions

This Part is based on the following publications:

- Clément Maria and Steve Y. Oudot. Zigzag Persistence via Reflections and Transpositions. *SODA 2015* [52]

Chapter 10

The Theory of Zigzag Persistence

In the previous chapters of this dissertation, we have studied persistent homology and efficient methods to compute persistence diagrams. In particular, the material introduced offers tools to solve the topological inference problem. On the mathematical side, persistent homology allows one to infer with garantees the homology of an unknown topological space. On the algorithmic side, the techniques introduced (simplex tree, compressed annotation matrix and modular reconstruction) allow one to compute an approximation of the space and to compute its persistent homology with various coefficient fields. Moreover, the experimental chapters demonstrate that the full computation takes a few minutes for simplicial complexes with hundreds of million simplices.

However, the theory of persistence is limited. On one side, approximating, with garantees, a topological space from a point cloud using only distance computations leads to Čech complex-like constructions which are extremely big. Even the big complexes we have considered so far may not be sufficient to obtain a valid persistence diagram. On the other side, persistent homology is rigid in the sense that the left-to-right simplicial maps in a filtration restricts the area of application to the multi-scale approximation of topological spaces we have presented so far. This motivates the introduction of the theory of *zigzag persistent homology*, that generalizes the theory of persistence. We propose an algorithm to compute zigzag persistence in the next chapter.

The theory of zigzag persistence is a generalization of the theory of per-

sistent homology to filtrations of the following form:

$$\mathbf{K}_1 \longleftrightarrow \mathbf{K}_2 \longleftrightarrow \cdots \longleftrightarrow \mathbf{K}_{n-1} \longleftrightarrow \mathbf{K}_n$$

where bidirectional arrows indicate that the actual arrow orientations can be arbitrary. These filtrations are called *zigzag filtrations*. For a dimension d and a coefficient field $\Bbbk$, the *zigzag (persistence) module* associated to this sequence is:

$$\mathbf{H}_d(\mathbf{K}_1, \Bbbk) \longleftrightarrow \mathbf{H}_d(\mathbf{K}_2, \Bbbk) \longleftrightarrow \cdots \longleftrightarrow \mathbf{H}_d(\mathbf{K}_{n-1}, \Bbbk) \longleftrightarrow \mathbf{H}_d(\mathbf{K}_n, \Bbbk)$$

This is the target object of zigzag persistence. Before studying it formally, we introduce some background about quiver theory.

10.1 Introduction to Quiver Theory

Let $(\Bbbk, +, \cdot)$ be an arbitrary field. In the following, all vector spaces are finite dimensional $\Bbbk$-vector spaces.

Quivers and Representations. A *quiver* is a directed graph. Specifically, a quiver is a pair $\mathcal{Q} = (Q_0, Q_1)$ of a finite set of vertices Q_0 and a finite set of (directed) arrows between them. If $a \in Q_1$ is an arrow, then $\mathsf{t}a$ and $\mathsf{h}a$ denote its *tail* and its *head* respectively.

Let us fix a quiver $\mathcal{Q} = (Q_0, Q_1)$. A $\Bbbk$-*representation* $\mathbb{V}$ of $\mathcal{Q}$ is an assignement of a finite dimensional $\Bbbk$-vector space for each vertex of $\mathcal{Q}$:

$$\{V_x : x \in Q_0\}$$

together with an assignement of a $\Bbbk$-linear maps for each arrow:

$$\{v_a : V_{\mathsf{t}a} \to V_{\mathsf{h}a} : a \in Q_1\}$$

We write for short $\mathbb{V} = (V_x, v_a)$. In a sense, quiver representations generalize the idea of vector space to a collection of vector spaces connected by homomorphisms. It is an algebraic object per se. We define now *subrepresentations*, *summands* and *morphisms* of $\Bbbk$-representations, which are defined pointwise.

A *subrepresentation* $\mathbb{U}$ of a representation $\mathbb{V} = (V_x, v_a)$ of $\mathcal{Q}$ is a representation (U_x, u_a) of $\mathcal{Q}$ such that U_x is a subspace of V_x for every vertex $x \in Q_0$ and u_a is the restriction of the linear map v_a to the subspace $U_{\mathsf{t}a}$.

Let $\mathbb{V} = (V_x, v_a)$ and $\mathbb{W} = (W_x, w_a)$ be two $\Bbbk$-representations. A *morphism of representations* $\phi : \mathbb{V} \to \mathbb{W}$ is a set of linear maps $\{\phi_x : V_x \to$

$W_x\}_{x\in Q_0}$ such that, for every arrow $a \in Q_1$, the following diagram commutes:

$$\begin{array}{ccc} V_{\mathsf{t}a} & \xrightarrow{v_a} & V_{\mathsf{h}a} \\ {\scriptstyle \phi_{\mathsf{t}a}}\downarrow & & \downarrow{\scriptstyle \phi_{\mathsf{h}a}} \\ W_{\mathsf{t}a} & \xrightarrow{w_a} & W_{\mathsf{h}a} \end{array}$$

The morphism is called a *monomorphism* if every ϕ_x is injective, an *epimorphism* if every ϕ_x is surjective, and an *isomorphism* (denoted $\cong$) if every ϕ_x is bijective.

Morphisms between $\Bbbk$-representations are composed pointwise, by composing the linear maps at each vertex $x \in Q_0$ independently. That is the composition of $\phi\colon \mathbb{U} \to \mathbb{V}$ with $\psi\colon \mathbb{V} \to \mathbb{W}$ is the morphism $\psi\circ\phi\colon \mathbb{U} \to \mathbb{W}$ defined by $(\psi\circ\phi)_x = \psi_x\circ\phi_x$ for all $x \in Q_0$. Moreover, for each $\Bbbk$-representation $\mathbb{V}$ we have the *identity morphism* $\mathbb{1}_{\mathbb{V}}\colon \mathbb{V} \to \mathbb{V}$ defined by $(\mathbb{1}_{\mathbb{V}})_x = \mathbb{1}_{V_x}$ for all $x \in Q_0$. These definitions induce the following properties on the $\Bbbk$-representations of $\mathcal{Q}$:

- They contain a zero object, which is the $\Bbbk$-representation with all spaces and maps equal to 0.

- They have a *direct sum* defined for any $\mathbb{V}, \mathbb{W}$ as the $\Bbbk$-representation $\mathbb{V}\oplus\mathbb{W}$ with spaces $V_x \oplus W_x$ for all $x \in Q_0$ and maps $v_a \oplus w_a = \left(\begin{smallmatrix} v_a & 0 \\ 0 & w_a \end{smallmatrix}\right)$ for every arrow $a \in Q_1$. A non-trivial $\Bbbk$-representation $\mathbb{V}$ is called *decomposable* if it is isomorphic to the direct sum of two non-trivial $\Bbbk$-representations (called *summands*), and *indecomposable* otherwise.

- Every morphism $\phi\colon \mathbb{V} \to \mathbb{W}$ has a *kernel*, defined by $(\ker\phi)_a = \ker\phi_a$ for all a. Similarly, ϕ has an *image* and a *cokernel*, also defined pointwise.

- A morphism ϕ is a monomorphism if and only if $\ker\phi = 0$, an epimorphism if and only if $\operatorname{cok}\phi = 0$, and an isomorphism if and only if ϕ is both a monomorphism and an epimorphism.

Remark 10.1. *A representation of $\mathcal{Q}$ has the structure of a module [38]. In the following, we use the terms (sub-)module and (sub-)representation indifferently.*

Quivers of Type A_n. In zigzag persistent homology, we focus on quivers of type A_n. An *A_n-type quiver* is a quiver which takes the following form:

$$\bullet_1 \longleftrightarrow \bullet_2 \longleftrightarrow \cdots \longleftrightarrow \bullet_{n-1} \longleftrightarrow \bullet_n$$

where bidirectional arrows indicate that the actual arrow orientations can be arbitrary. We define two total order relations on the indices $\{1, \ldots, n\}$ of the vertices of the quiver, depending on the orientation of the arrows. Let $\leq_{\mathbf{b}}$ be the order satisfying for every indices $i, j \in \{1, \ldots, n\}$, $i \leq_{\mathbf{b}} j$ iff:

$$\begin{cases} \text{either} & i = j, & & & \\ \text{or} & i < j & \text{and} & \bullet_{j-1} \to \bullet_j & \text{is forward,} \\ \text{or} & i > j & \text{and} & \bullet_{i-1} \leftarrow \bullet_i & \text{is backward.} \end{cases}$$

Symmetrically, we define the order $\leq_{\mathbf{d}}$ satisfying, for every indices $i, j \in \{1, \ldots, n\}$, $i \leq_{\mathbf{d}} j$ iff:

$$\begin{cases} \text{either} & i = j, & & & \\ \text{or} & i > j & \text{and} & \bullet_{j} \leftarrow \bullet_{j+1} & \text{is backward,} \\ \text{or} & i < j & \text{and} & \bullet_{i} \to \bullet_{i+1} & \text{is forward.} \end{cases}$$

We also define $\max_{\leq_{\mathbf{b}}}$ and $\max_{\leq_{\mathbf{d}}}$ which are the maximum function w.r.t. to the orders $\leq_{\mathbf{b}}$ and $\leq_{\mathbf{d}}$ respectively. For example, in the quiver:

$$\bullet_1 \longrightarrow \bullet_2 \longrightarrow \bullet_3 \longleftarrow \bullet_4 \longrightarrow \bullet_5 \longleftarrow \bullet_6$$

the indices satisfy $6 \leq_{\mathbf{b}} 4 \leq_{\mathbf{b}} 1 \leq_{\mathbf{b}} 2 \leq_{\mathbf{b}} 3 \leq_{\mathbf{b}} 5$ and $1 \leq_{\mathbf{d}} 2 \leq_{\mathbf{d}} 4 \leq_{\mathbf{d}} 6 \leq_{\mathbf{d}} 5 \leq_{\mathbf{d}} 3$. Note that the list of indices sorted according to increasing $\leq_{\mathbf{b}}$ is made of, first, the list of indices, in decreasing index order, that are the tail of a backward arrow, then the index 1, then the list of indices, in increasing index order, that are the head of a forward arrow. The list of indices sorted according to increasing $\leq_{\mathbf{d}}$ is made of, first, the list of indices, in increasing index order, that are the tail of a forward arrow, then the index n, then the list of indices, in decreasing index order, that are the head of a backward arrow. Note that these orders are a reformulation of the *birth-time* and *death-time indices* of [24].

In computational topology, the (sub-)representations of A_n-type quivers are also called *zigzag (sub-)modules*. We usually denote the set of vertices Q_0 by $\{1, \cdots, n\}$ and the homomorphism (of a representation) on the i^{th} arrow by f_i if the arrow is forward $\bullet_i \longrightarrow \bullet_{i+1}$ and by g_i if the arrow is backward $\bullet_i \longleftarrow \bullet_{i+1}$.

Let $\mathbb{V}$ be a $\Bbbk$-representation of an A_n-type quiver:

$$V_1 \longleftrightarrow V_2 \longleftrightarrow \cdots \longleftrightarrow V_{n-1} \longleftrightarrow V_n \tag{10.1}$$

Both the *Krull-Schmidt principle* and *Gabriel's theorem* hold and take the following form:

Theorem 10.1 (Krull-Schmidt, Gabriel)**.** *Every zigzag module* $\mathbb{V}$ *over a given field* $\Bbbk$ *is decomposable as a direct sum of indecomposable modules*

$$\mathbb{V} = \mathbb{V}^1 \oplus \mathbb{V}^2 \oplus \cdots \oplus \mathbb{V}^N, \tag{10.2}$$

where each indecomposable $\mathbb{V}^i$ *is isomorphic to some* interval module $\mathbb{I}[b_i; d_i]$*, defined as:*

$$\underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[1;b_i-1]} \overset{0}{\longleftrightarrow} \underbrace{\Bbbk \overset{\mathbb{1}}{\longleftrightarrow} \cdots \overset{\mathbb{1}}{\longleftrightarrow} \Bbbk}_{[b_i;d_i]} \overset{0}{\longleftrightarrow} \underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[d_i+1;n]} \tag{10.3}$$

Moreover, the decomposition is unique up to isomorphism and reordering of the terms $\mathbb{V}^i$*. We call* b_i *the* birth *and* d_i *the* death *of each interval module* $\mathbb{I}[b_i; d_i]$*.*

What this result says is that the algebraic structure of a zigzag module $\mathbb{V}$ is completely determined by the finite collection of intervals $[b_i; d_i]$ involved in its decomposition. This collection is called the *persistence barcode* of $\mathbb{V}$, and it is our target object when computing zigzag persistence. Similarly, we can define a persistence diagram and bottleneck distances between them.

For a $\Bbbk$-representation $\mathbb{V} = (V_i, f_i/g_i)_{i=1\ldots n}$ of an A_n-type quiver $\mathcal{Q}$, we define $\mathbb{V}[b; d]$ to be the representation $\mathbb{V} = (V_i, f_i/g_i)_{i=b\ldots d}$ of the quiver $\mathcal{Q}[b; d]$ restricted to the vertices (and arrows between them) of indices $b \leq i \leq d$. We call $\mathbb{V}[b; d]$ a *restriction* of the module $\mathbb{V}$ to the range $[b; d]$. If $b = 1$, we call $\mathbb{V}[1; d]$ a *prefix* of $\mathbb{V}$ and if $d = n$ we call $\mathbb{V}[b; n]$ a *suffix*. The following theorem states that the interval decomposition of the restriction $\mathbb{V}[b; d]$ is the direct sum of the intervals of the decomposition of $\mathbb{V}$ restricted to $[b; d]$.

Theorem 10.2 (Restriction Principle [24])**.** *Let* $\mathbb{V}$ *be a zigzag module that decomposes into:*

$$\mathbb{V} \cong \bigoplus_{j \in J} \mathbb{I}[b_i; d_i]$$

then the restriction $\mathbb{V}[b; d]$ *decomposes into:*

$$\mathbb{V}[b; d] \cong \bigoplus_{j \in J} \mathbb{I}\left([b_i; d_i] \cap [b; d]\right)$$

where $\mathbb{I}\left([b_i; d_i] \cap [b; d]\right)$ *is the interval module over the interval* $[b_i; d_i] \cap [b; d]$ *if* $[b_i; d_i] \cap [b; d] \neq \emptyset$ *and is the* 0 *module otherwise.*

10.2 Zigzag Persistent Homology

In practice, zigzag modules are obtained by computing the homology of finite sequences $\mathbb{K}$ of simplicial complexes connected by inclusion maps (or more generally simplicial maps), called *zigzag filtrations*:

$$\mathbb{K} := \quad \mathbf{K}_1 \longleftrightarrow \mathbf{K}_2 \longleftrightarrow \cdots \longleftrightarrow \mathbf{K}_{n-1} \longleftrightarrow \mathbf{K}_n \tag{10.4}$$

We call *standard persistence* the case where all arrows are oriented in the same direction. In this case, $\mathbb{K}$ is called a *standard filtration*.

As in standard persistence, a zigzag filtration induces a sequence $\mathbb{H}\mathrm{om}$ of homology groups with coefficients in a field $\Bbbk$:

$$\mathbf{H}_d(\mathbf{K}_1, \Bbbk) \longleftrightarrow \mathbf{H}_d(\mathbf{K}_2, \Bbbk) \longleftrightarrow \cdots \longleftrightarrow \mathbf{H}_d(\mathbf{K}_{n-1}, \Bbbk) \longleftrightarrow \mathbf{H}_d(\mathbf{K}_n, \Bbbk)$$

This is a $\Bbbk$-representation of an A_n-type quiver. It admits an interval decomposition by virtue of Theorem 10.1.

We focus in this part on the case where all maps of the zigzag filtration are elementary inclusions. Computing zigzag persistence consists in computing the interval decomposition of the zigzag module $\mathbb{H}\mathrm{om}$ corresponding to a zigzag filtration $\mathbb{K}$, given the sequence of simplex insertions and deletions of $\mathbb{K}$.

Algorithmic Aspects of Zigzag Persistence. Standard persistent homology has attracted a lot of attention in the recent years and, as mentioned previously, very efficient algorithms have been developed.

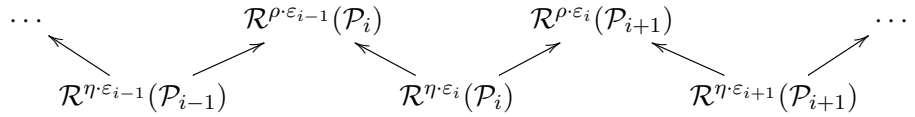
By contrast, the general zigzag case has received much less attention despite its growing interest in applications. Perhaps the main reason is that, unlike standard persistence modules, it is unknown whether general zigzag modules can be viewed as modules over a ring of polynomials. Hence, computing their interval decompositions as in Theorem 10.1 requires more elaborate machinery than mere matrix reduction. The so-called *right filtration functor* of Carlsson and de Silva [24] is an example of such machinery. Introduced originally as a tool to prove Gabriel's theorem, it was eventually turned into the first – and so far only – practical algorithm to decompose general zigzag modules [25]. This algorithm works with modules derived at the homology level from zigzag filtrations $\mathbb{K}$ in which each arrow corresponds to a single simplex insertion or deletion. It scans the zigzag filtration $\mathbb{K}$ from left to right, adding or removing one simplex at a time, and maintaining a compatible basis – in fact three, one for the cycles, one for the boundaries, and one for the killing chains – for the *right filtration* of the

zigzag prefix $\mathbb{K}[1;i]$ at iteration i (i.e. the zigzag filtration restricted to the i first complexes). Its implementation is available as part of the `C++` library `Dionysus` [56] and performs reasonably well in practice, however nowhere as efficiently as the aforementioned optimized algorithms for standard persistence. Its pseudo-code is also significantly longer and more intricate due to the required extra machinery. As a result, while reproducing it step by step is easy, grasping the higher-level picture of how and why it works is a comparatively challenging task. To what extent the approach (and its theoretical analysis) can be simplified and optimized is becoming an essential question.

We introduce in Chapter 11 an algorithm for computing zigzag persistence, and discuss the implications of such an algorithm in the concluding remarks of this dissertation.

Oscillating Rips Zigzag and Inference. When solving the topological inference problem, the classic approach in persistent homology is to reconstruct the unknown topological space at different scales with a Rips complex, using a scale parameter. The *oscillating Rips zigzag* is a zigzag filtration that exploits two parameters: scale and number of points.

Let $\mathcal{P}$ be a set of points in a metric space and let $[p_1, \cdots p_n]$ be an ordering of the points. We denote by $\mathcal{P}_i$ the i^{th} prefix $[p_1, \cdots, p_i]$ of the ordering, and by ε_i the Hausdorff distance $\mathsf{d}_{\mathcal{H}}(\mathcal{P}_i, \mathcal{P})$. The value ε_i is called the i^{th} geometric scale. Given two multipliers $\eta \leq \rho$, the *oscillating Rips zigzag* on the ordered points $\mathcal{P}$ is the following filtration:



where $\mathcal{R}^{\alpha}(\mathcal{P})$ is the Rips complex of threshold α and vertices the points of $\mathcal{P}$. Up-right arrows consist in adding more points, and up-left ones consist in increasing the threshold for the Rips; they are both inclusions but in different directions.

A standard ordering of the points is the *furthest point ordering* consisting in constructing the sequence of $\mathcal{P}_i$ incrementally:

- $\mathcal{P}_1 = (p_1)$, for an arbitrary point $p_1 \in \mathcal{P}$,
- if $\mathcal{P}_i$ has been computed, then pick p_{i+1} such that:

$$p_{i+1} = \arg\max_{p \in \mathcal{P}} \mathsf{d}(p, \mathcal{P}_i)$$

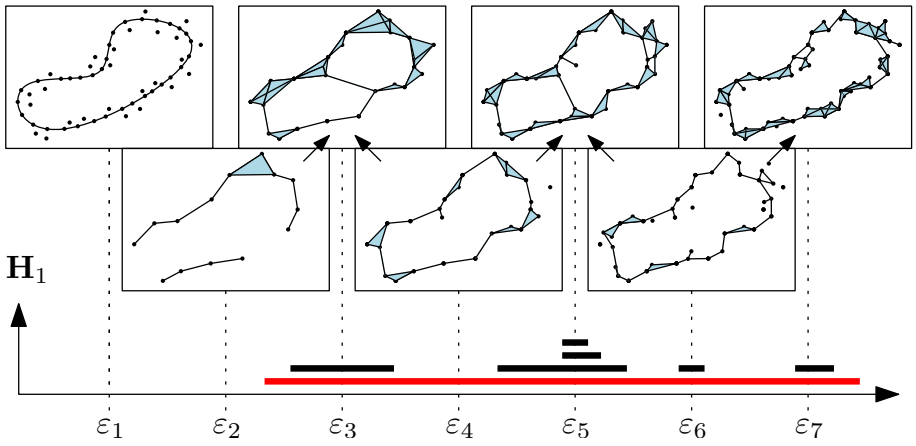


Figure 10.1: Zigzag persistent homology for topological inference.

Figure 10.1 pictures an application of zigzag persistence using an oscillating Rips filtration for topology inference. The same example has been solved with standard persistence in Figure 4.8. Because the threshold of the Rips complex adapts to the density of points, the simplicial complexes maintained along the filtration are much smaller than the ones in a Rips filtration for standard persistence. Note that the persistence diagram provides the right homology for the unknown topological space.

The oscillating Rips zigzag actually produces a provably good result. The following theorem is also based on the proximity of the Rips and the Čech complexes in $\mathbb{R}^D$ (Lemma 1.1) and the homotopy equivalent reconstruction result of Theorem 1.4:

Theorem 10.3 (Reconstruction with Oscillating Rips Zigzag)**.** *Let C be a compact set of $\mathbb{R}^D$, with a strictly positive topological feature size, and let $\mathcal{P} = \{p_1, \cdots, p_n\}$ be a set of points such that $\mathsf{d}_{\mathcal{H}}(C, \mathcal{P}) < \varepsilon$. There exist values of the multipliers η and ρ (depending only on the dimension D) such that, for a sufficiently small ε, there is a range of indices $[\ell, k] \subseteq [1; n]$ such that the persistence barcode of the oscillating Rips filtration, restricted to the geometric scales from ε_ℓ to ε_k, contains only full-length intervals and ephemeral (length 0) ones. The number of full-length intervals is equal the the Betti number of C.*

As a consequence, zigzag persistence furnishes all tools for solving the topological inference problem with a provably good result, while maintaining much smaller simplicial complexes.

Remark 10.2. *Zigzag persistence does not allow to consider several parameters (like scale and number of points) independently. Indeed, in the oscillating Rips zigzag we consider two parameters but make their values depend on each other. Hence, a zigzag filtration draws a path that explores a multi-dimensional space of parameters, but does not describe it entirely. Considering different parameters independently is known as* multi-dimensional persistence*, where the quivers considered are no longer zigzaging paths. Gabriel's theorem does not apply to these quivers, and finding and computing a direct-sum decomposition into indecomposables modules for particular families of multi-dimensional persistence modules is an active area of research.*

10.3 Representative Sequences and Encoding

We now introduce the main low-level object used to prove theorems and the validity of our algorithm.

Definition 10.1. *For an $\Bbbk$-representation $\mathbb{V} = (V_i, f_i/g_i)$ of an A_n-type quiver $\mathcal{Q}$, a* representative sequence*, denoted by $u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}$, is an n-tuple $(u^{(1)}, \ldots, u^{(n)}) \in V_1 \times \cdots \times V_n$ such that:*

(a) The index b, called the birth index*, satisfies either $b = 1$, or $\bullet_{b-1} \to \bullet_b$ is forward, or $g_{b-1}(u^{(b)}) = 0$. In addition, $u^{(i)} = 0$ for every $i < b$.*

(b) The index d, called the death index*, satisfies either $d = n$, or $\bullet_d \leftarrow \bullet_{d+1}$ is backward, or $f_d(u^{(d)}) = 0$. In addition, $u^{(i)} = 0$ for every $i > d$.*

(c) For all i, $b \leq i < d$, either $f_i(u^{(i)}) = u^{(i+1)}$ or $u^{(i)} = g_i(u^{(i+1)})$, depending on the direction of the arrow $\bullet_i \leftrightarrow \bullet_{i+1}$. In addition, $u^{(i)} \neq 0$ for every i, $b \leq i \leq d$.

The following easy propositions relates the representative sequences to the so-called *interval submodules* of $\mathbb{V}$, i.e. the submodules of $\mathbb{V}$ that are isomorphic to interval representations as in (10.3):

Proposition 10.1. *$\mathbb{U}$ is an interval submodule of $\mathbb{V}$ if and only if there exists a representative sequence $u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}$ such that $\mathbb{U}$ is equal to:*

$$0 \leftrightarrows \cdots 0 \leftrightarrows \langle u^{(b)} \rangle \leftrightarrows \cdots \leftrightarrows \langle u^{(d)} \rangle \leftrightarrows 0 \cdots \leftrightarrows 0$$

If an interval submodule of $\mathbb{V}$ is in direct sum, we call it an *interval summand*. As a direct consequence of the definitions, we deduce:

Proposition 10.2. *Let $\mathbb{V} = (V_i, f_i/g_i)$ be a representation of an A_n-type quiver and $\{ u_j^{(b_j)} \leftrightarrows \cdots \leftrightarrows u_j^{(d_j)} \}_{j \in J}$ be a family of representative sequences*

for $\mathbb{V}$. *If, for every index* $i \in \{1, \ldots, n\}$, *the set* $\{u_j^{(i)} : u_j^{(i)} \neq 0\}_{j \in J}$ *is a basis for* V_i, *then:*

$$\mathbb{V} \cong \bigoplus_{j \in J} \mathbb{I}[b_j; d_j]$$

In this case, we say that the family of representative sequences represents an interval decomposition *for* $\mathbb{V}$.

Arithmetic of Representative Sequences. Let $\mathbb{V} = (V_i, f_i/g_i)$ be a representation of an A_n-type quiver.

Definition 10.2. *Let* $\mathbf{u} = u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}$ *and* $\mathbf{v} = v^{(b')} \leftrightarrows \cdots \leftrightarrows v^{(d')}$ *be two representative sequences. Define* $b_m = \max_{\leq_{\mathbf{b}}}\{b, b'\}$ *and* $d_m = \max_{\leq_{\mathbf{d}}}\{d, d'\}$. *If the two sequences satisfy:*

(a) $[b; d] \cap [b'; d'] \neq \emptyset$,

(b) for all $j \in [b; d] \cap [b'; d']$, *the vectors* $u^{(j)}$ *and* $v^{(j)}$ *are linearly independent in* V_j,

(c) $b_m \leq d_m$,

then we define the binary operator $*$*:*

$$\begin{aligned}(u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}) * (v^{(b')} \leftrightarrows \cdots \leftrightarrows v^{(d')}) \quad := \\ u^{(b_m)} + v^{(b_m)} \leftrightarrows \cdots \leftrightarrows u^{(d_m)} + v^{(d_m)}\end{aligned}$$

of birth b_m *and death* d_m. *We also define, for any representative sequence* $u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}$ *and scalar* $\gamma \in \Bbbk$, *not equal to* 0, *the scalar multiplication:*

$$\gamma \cdot (u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}) := \gamma u^{(b)} \leftrightarrows \cdots \leftrightarrows \gamma u^{(d)}$$

Lemma 10.1. *For two representative sequences* $\mathbf{u}$ *and* $\mathbf{v}$ *satisfying conditions (a), (b) and (c) above, and a non-zero scalar* γ, $\mathbf{u} * \mathbf{v}$ *and* $\gamma \cdot \mathbf{u}$ *are representative sequences.*

Proof. The case of $\gamma \cdot \mathbf{u}$ is direct. Denote $\mathbf{u} = u^{(b)} \leftrightarrows \cdots \leftrightarrows u^{(d)}$ and $\mathbf{v} = v^{(b')} \leftrightarrows \cdots \leftrightarrows v^{(d')}$. We prove that $\mathbf{u} * \mathbf{v}$ is a representative sequence. First, we prove that definition 10.1 *(a)* is satisfied. If $b_{\mathrm{m}} = 1$ or $\bullet_{b_{\mathrm{m}}-1} \to \bullet_{b_{\mathrm{m}}}$ is forward, *(a)* is satisfied. Suppose now that $b_{\mathrm{m}} > 1$ and that the arrow $\bullet_{b_{\mathrm{m}}-1} \leftarrow \bullet_{b_{\mathrm{m}}}$ is backward. We prove that $g_{b_{\mathrm{m}}-1}(u^{(b_{\mathrm{m}})} + v^{(b_{\mathrm{m}})}) = 0$. Suppose, w.l.o.g., that $b_{\mathrm{m}} = b$. Hence $b' \leq_{\mathbf{b}} b$ which implies, together with $\bullet_{b_{\mathrm{m}}-1} \leftarrow \bullet_{b_{\mathrm{m}}}$ being backward:

- either $b = b'$, in which case $g_{b_{\mathrm{m}}-1}(u^{(b_{\mathrm{m}})}) = g_{b_{\mathrm{m}}-1}(v^{(b_{\mathrm{m}})}) = 0$,

- or $b > b'$, in which case $g_{b_{\mathrm{m}}-1}(u^{(b_{\mathrm{m}})}) = 0$ and $v^{(b_{\mathrm{m}})} = 0$.

Definition 10.1 (b) is satisfied with a similar proof.

We prove now that definition 10.1 *(c)* is satisfied. Definition 10.2*(c)* ensures that the interval $[b_{\mathrm{m}}; d_{\mathrm{m}}]$ is not empty and definition 10.2*(a)* implies that $[b_{\mathrm{m}}; d_{\mathrm{m}}] \subseteq [b; d] \cup [b'; d']$. As a consequence, for any index $j \in [b_{\mathrm{m}}; d_{\mathrm{m}}]$, $u^{(j)}$ and $v^{(j)}$ are not both equal to 0 and, by virtue of definition 10.2*(b)*, $u^{(j)} + v^{(j)} \neq 0$. Finally, we verify that for all $j \in [b_{\mathrm{m}}; d_{\mathrm{m}}]$, $f_j(u^{(j)} + v^{(j)}) = u^{(j+1)} + v^{(j+1)}$ or $g_j(u^{(j+1)} + v^{(j+1)}) = u^{(j)} + v^{(j)}$. For every $j \in [b_{\mathrm{m}}; d_{\mathrm{m}}] \setminus \{b - 1, b' - 1, d, d'\}$, definition 10.1 *(c)* is satisfied by linearity of f_j and g_j. Suppose, w.l.o.g., that the index b is contained within $(b_{\mathrm{m}}; d_{\mathrm{m}}]$. Hence, $b' < b$ and $b \leq_{\mathbf{b}} b'$, which implies that $\bullet_{b-1} \leftarrow \bullet_b$ is backward. Consequently, $g_{b-1}(u^{(b)}) = 0$ and $g_{b-1}(u^{(b)} + v^{(b)}) = u^{(b-1)} + v^{(b-1)}$. The case of d contained within $[b_{\mathrm{m}}; d_{\mathrm{m}})$ is similar.

□

The following lemma gives some properties of " $\cdot$ " and " $*$ ":

Lemma 10.2. *Let* $\mathbf{u}, \mathbf{v}$ *and* $\mathbf{w}$ *be three representative sequences pairwise satisfying conditions (a), (b) and (c) of definition 10.2. We have:*

1. $\mathbf{u} * \mathbf{v} = \mathbf{v} * \mathbf{u}$ *(commutativity),*

2. $(\mathbf{u} * \mathbf{v}) * \mathbf{w} = \mathbf{u} * (\mathbf{v} * \mathbf{w})$ *(associativity),*

3. $\gamma \cdot (\delta \cdot \mathbf{u}) = (\gamma\delta) \cdot \mathbf{u}$,

4. $1 \cdot \mathbf{u} = \mathbf{u}$.

Proof. The commutativity (1.) is direct. Properties (3.) and (4.) follow directly from the definition of the multiplication " $\cdot$ ". We prove the associativity (2.). First, we prove that the sequences $(\mathbf{u} * \mathbf{v})$ and $\mathbf{w}$ satisfy conditions *(a)*, *(b)* and *(c)* of definition 10.2. Denote by $b_{\mathbf{u}}$, $b_{\mathbf{v}}$ and $b_{\mathbf{w}}$ the births of $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ respectively, and $d_{\mathbf{u}}$, $d_{\mathbf{v}}$ and $d_{\mathbf{w}}$ their deaths. The fact that the pairwise intersections of the intervals $[b_{\mathbf{u}}; d_{\mathbf{u}}]$, $[b_{\mathbf{v}}; d_{\mathbf{v}}]$ and $[b_{\mathbf{w}}; d_{\mathbf{w}}]$ is non-empty implies that their common intersection is non-empty[1]. Hence definition 10.2 *(a)* is satisfied. Definition 10.2 *(b)* and *(c)* are directly inherited from the fact they are pairwise satisfied by $b_{\mathbf{u}}$, $b_{\mathbf{v}}$ and $b_{\mathbf{w}}$. By symmetry, the result applies to the sequences $\mathbf{u}$ and $\mathbf{v} * \mathbf{w}$.

[1] This is a particular case of Helly's theorem.

Finally, associativity follows from the associativity of + in each vector space and the associativity of the functions $\max_{\leq_{\mathbf{b}}}$ and $\max_{\leq_{\mathbf{d}}}$.

□

For example, consider the following representation $\mathbb{V}$ of the quiver of A_6-type presented above:

$$\begin{array}{c} \Bbbk \xrightarrow{\mathbb{1}} \Bbbk \xrightarrow{\mathbb{1}} \Bbbk \xleftarrow{\pi_1} \Bbbk^2 \xrightarrow{\iota_z} \Bbbk^3 \xleftarrow{\iota_y} \Bbbk^2 \\ x \longmapsto x \longmapsto x \longmapsfrom (x,y) \longmapsto (x,y,z) \longmapsfrom (x,z) \end{array}$$

Naturally, $\mathbb{V} \cong \mathbb{I}[1;6] \oplus \mathbb{I}[4;5] \oplus \mathbb{I}[5;6]$. Let $x^{(1)} \leftrightarrows \cdots \leftrightarrows x^{(6)}$ and $y^{(4)} \leftrightarrows \cdots \leftrightarrows y^{(5)}$ be representative sequences for submodules respectively isomorphic to $\mathbb{I}[1;6]$ and $\mathbb{I}[4;5]$. We have:

$$\begin{array}{l} (\ x^{(1)} \leftrightarrows \cdots \leftrightarrows x^{(6)}\ ) * (\ y^{(4)} \leftrightarrows \cdots \leftrightarrows y^{(5)}\ ) = \\ (x+y)^{(1)} \leftrightarrows \cdots \leftrightarrows (x+y)^{(5)} \end{array}$$

In the following, the operators " $*$ " and " $\cdot$ " offer tools to manipulate interval submodules of an $\Bbbk$-representation.

Validity of the Persistence Algorithm. Our approach to computing the zigzag persistence of $\mathbb{V}$ in the next chapter is to maintain representative sequences for the interval summands of a direct sum decomposition of $\mathbb{V}$. Note that this is what the standard persistence algorithm does. Indeed, let $\mathbf{H}(\mathbb{K}) = 0 \longrightarrow \mathbf{H}_*(\mathbf{K}_1) \longrightarrow \cdots \longrightarrow \mathbf{H}_*(\mathbf{K}_n)$ be a standard persistence module, and let $\{\widehat{\tau}_i\}$, $i \in F \sqcup G \sqcup H$, be an encoding. The representative sequences induced by the cycles $\widehat{\tau}_f$ for $f \in F$ and $\widehat{\tau}_g$ for $g \in G$ paired with $h \in H$ are:

$$(0, \cdots, 0, [\widehat{\tau}_f]_f, \cdots, [\widehat{\tau}_f]_n) \quad \text{and} \quad (0, \cdots, 0, [\widehat{\tau}_g]_g, \cdots, [\widehat{\tau}_g]_{h-1}, 0, \cdots, 0)$$

where the notation $[\widehat{\tau}_i]_j$ for $i \leq j$ means that the cycle class is considered as an element of $\mathbf{H}_*(\mathbf{K}_j)$. This does not change the formal sum of the chain $\widehat{\tau}_i$ because, by definition of an encoding, its leading term is τ_i. These representative sequences represent the interval summands isomorphic to $\mathbb{I}[f;n]$ and $\mathbb{I}[g;h-1]$ in a direct-sum decomposition of $\mathbb{V}$.

Bibliographical Notes

We refer to [38] for an introduction to quiver theory, and to [60] for a detailed study of quiver theory and their use in persistent homology. This introduction to quiver theory is mostly inspired from the last reference.

The theory of zigzag persistence has been introduced in [24] together with an algorithm based on the concept of *right-filtration*. The definition of birth and death index comes from this source. A cubic algorithm for zigzag persistence has been then introduced in [25]. Without considering the algorithm presented in next chapter, this is the only existing practical algorithm for zigzag persistence and an implementation is available in the `Dionysus` [56] library. The theoretical complexity has been improved to matrix multiplication time in [54].

The oscillating Rips zigzag has been introduced by Morozov [56] for topological inference and analysed in [61] resulting in Theorem 10.3.

Chapter 11

Zigzag Persistence Algorithm

We introduce a new algorithm for computing zigzag persistence, designed in the same spirit as the standard persistence algorithm. Our algorithm reduces a single matrix, maintains an explicit set of chains encoding the persistent homology of the current zigzag, and updates it under simplex insertions and removals. The total worst-case running time matches the usual cubic bound.

A noticeable difference with the standard persistence algorithm is that we do not insert or remove new simplices "at the end" of the zigzag, but rather "in the middle". To do so, we use arrow reflections and transpositions, in the same spirit as reflection functors in quiver theory. Our analysis introduces new kinds of reflections in quiver representation theory: the "injective and surjective diamonds". It also introduces the "transposition diamond" which models arrow transpositions. For each type of diamond we are able to predict the changes in the interval decomposition and associated compatible bases. Arrow transpositions have been studied previously in the context of standard persistent homology, and we extend the study to the context of zigzag persistence. For both types of transformations, we provide simple procedures to update the interval decomposition and associated compatible homology basis.

For simplicity of exposition, we present the algorithm for homology with coefficients in the field $\mathbb{Z}_2$. The presentation generalizes naturally to arbitrary field coefficients $\Bbbk$. In particular, we prove the *diamond principles* in full generality in Chapter 12.

Overview of the Approach We introduce a new method for computing zigzag persistence in the same context as [25]. Our approach is inspired from another (and more ancient) proof of Gabriel's theorem [13], which performs

arrow reflections in sequence in a zigzag module and tracks the corresponding changes in its interval decomposition. Our algorithm scans the input zigzag filtration $\mathbb{K}$ from left to right as before, however it appends to the current prefix $\mathbb{K}[1;i] := \mathbf{K}_1 \leftrightarrow \cdots \leftrightarrow \mathbf{K}_i$ a *descending chain* of subcomplexes in which every arrow is backward and corresponds to a single simplex deletion:

$$\begin{array}{ccccccccc} \mathbf{K}_1 & \longleftarrow & \cdots & \longleftarrow & \mathbf{K}_i & \xleftarrow{\tau_m} & & & \\ \mathbf{K}_i \setminus \{\tau_m\} & \xleftarrow[\tau_{m-1}]{} & \mathbf{K}_i \setminus \{\tau_m, \tau_{m-1}\} & \xleftarrow[\tau_{m-2}]{} & \cdots & \xleftarrow[\tau_1]{} & \emptyset & & \end{array}$$

Adding such a descending chain is a way for us to anticipate the simplex deletions that will occur in the rest of the input zigzag, even though the actual order in which they will occur may not be the same as ours. Every new simplex insertion or deletion happens at the junction between the zigzag prefix and the descending chain. The corresponding changes in the zigzag filtration can be described as a combination of *arrow reflections* and *arrow transpositions* taking place in the descending chain. An *arrow reflection* is described by the diagram:

$$\begin{array}{ccccccc} & & & \mathbf{K} \cup \{\sigma\} & & & \\ & & \overset{\sigma}{\nearrow} & & \overset{\sigma}{\nwarrow} & & \\ \cdots \longleftrightarrow & \mathbf{K} & & & & \mathbf{K} & \longleftrightarrow \cdots \\ & & \underset{\mathbb{1}}{\nwarrow} & & \underset{\mathbb{1}}{\nearrow} & & \\ & & & \mathbf{K} & & & \end{array}$$

and an *arrow transposition* is described by the diagram:

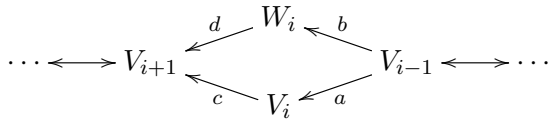
$$\begin{array}{ccccccc} & & & \mathbf{K} \cup \{\tau\} & & & \\ & & \overset{\sigma}{\swarrow} & & \overset{\tau}{\nwarrow} & & \\ \cdots \longleftrightarrow & \mathbf{K} \cup \{\sigma, \tau\} & & & & \mathbf{K} & \longleftrightarrow \cdots \\ & & \underset{\tau}{\nwarrow} & & \underset{\sigma}{\swarrow} & & \\ & & & \mathbf{K} \cup \{\sigma\} & & & \end{array}$$

The changes in the zigzag filtration consist in passing from the bottom representation to the top representation of either one of these diagrams.

These transformations of the zigzag filtration induce *diamonds* at the homology level. Specifically, an arrow reflection induces an *injective or surjective diamond*:

$$\begin{array}{ccccccc} & & & W & & & \\ & & \overset{f}{\nearrow} & & \overset{f}{\nwarrow} & & \\ \cdots \longleftrightarrow & V & & & & V & \longleftrightarrow \cdots \\ & & \underset{\mathbb{1}}{\nwarrow} & & \underset{\mathbb{1}}{\nearrow} & & \\ & & & V & & & \end{array}$$

depending whether f is injective or surjective. An arrow transposition induces a *transposition diamond*:



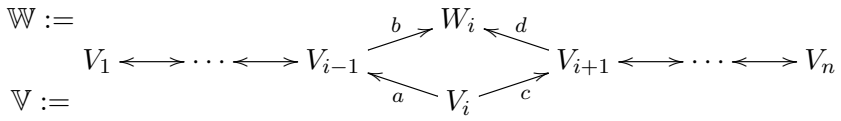
where a, b, c, d satisfy an exactness hypothesis [24]. We introduce and prove the *Injective Diamond Principle*, the *Surjective Diamond Principle* and the *Transposition Diamond Principle* in order to express the evolution in the interval decomposition of a zigzag module when passing one of the diamonds presented above. These diamond principles add up to the *Exact Diamond Principle* originally introduced by Carlsson and de Silva [24]. The diamonds corresponding to arrow transpositions were studied by Cohen-Steiner et al. [31] in the context of standard persistence. The transposition diamond principle generalizes the study to zigzag persistence.

Our algorithm to compute zigzag persistence is just one big sequence of diamond traversals. We handle each arrow reflection in time $O(m^2)$ and each arrow transposition in time $O(m)$, where m is the size of the largest simplicial complex in the zigzag filtration. Hence our algorithm can decompose zigzag modules into intervals in time $O(nm^2)$ as in [25]. Our preliminary experiments show a good behavior of our algorithm compared to the one in [25] in practice. Moreover, the similarity of our method to the standard persistence algorithm opens the door to all kinds of optimizations. These questions, and others, are discussed in the concluding remarks.

11.1 Diamond Principles.

In this section we relate the interval decompositions of two representations $\mathbb{V}$ and $\mathbb{W}$ related by a local change, called a *diamond*. We recall the *Exact Diamond Principle* of [24] and introduce the main theoretical results of the chapter, specifically the *Injective and Surjective Diamond Principles* and the *Transposition Diamond Principle*.

Exact Diamonds. Consider the diagram:



(11.1)

We say that the following diagram:

$$\begin{array}{ccc} V_{i+1} & \xrightarrow{d} & W_i \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle b} \\ V_i & \xrightarrow{a} & V_{i-1} \end{array} \tag{11.2}$$

is *exact* [24] if $\operatorname{im} D_1 = \ker D_2$ in the sequence $V_i \xrightarrow{D_1} V_{i-1} \oplus V_{i+1} \xrightarrow{D_2} W_i$, where $D_1(v) = (a(v), c(v))$ and $D_2(x, y) = b(x) - d(y)$. Note in particular that an exact diamond commutes, i.e. $b \circ a = d \circ c$.

If diagram 11.2 is exact we say that the representations $\mathbb{V}$ and $\mathbb{W}$ are related by an *exact diamond at index i*. We recall the *Exact Diamond Principle*:

Theorem 11.1 (Exact Diamond Principle [24])**.** *Given $\mathbb{V}$ and $\mathbb{W}$ related by an exact diamond at index i, there is a partial bijection between the intervals of the decompositions of $\mathbb{V}$ and $\mathbb{W}$:*

- *intervals $\mathbb{I}[i; i]$ are unmatched,*
- *for $b < i$, intervals $\mathbb{I}[b; i]$ are matched with intervals $\mathbb{I}[b; i-1]$ and vice versa,*
- *for $d > i$, intervals $\mathbb{I}[i; d]$ are matched with intervals $\mathbb{I}[i+1; d]$ and vice versa,*
- *intervals $\mathbb{I}[b; d]$ are matched with intervals $\mathbb{I}[b; d]$ in all other cases.*

Injective and Surjective Diamonds. For simplicity of exposition, we assume in the following that every representation $\mathbb{U}$ has an interval decomposition that satisfies the following:

- for every index $i > 1$, there is at most one interval with birth i,
- for every index $j < n$, there is at most one interval with death j.

These conditions are in particular satisfied in zigzag persistent homology (section 11.2).

We relate the interval decompositions of the bottom representation $\mathbb{V}$ and top representation $\mathbb{W}$ of the following diagram:

$$\begin{array}{ll} \mathbb{W} := & \\ & V_1 \longleftrightarrow \cdots \longleftrightarrow V \xrightarrow{f} W \xleftarrow{f} V \longleftrightarrow \cdots \longleftrightarrow V_n \\ \mathbb{V} := & \end{array} \tag{11.3}$$

where the diamond is located at index i in the module. We distinguish the case where f is injective of corank 1 and the case where f is surjective of nullity 1.

Theorem 11.2 (Injective Diamond Principle)**.** *Suppose f is injective of corank* 1*. Then,*

$$\mathbb{W} \cong \mathbb{V} \oplus \mathbb{I}[i; i]$$

Proof. $\mathbb{V}$ and $\mathbb{W}$ are related by an exact diamond. Indeed, by injectivity of f, $\text{im } D_1 = \ker D_2$ in the sequence $V \xrightarrow{D_1} V \oplus V \xrightarrow{D_2} W$ with $D_1 : v \mapsto (v, v)$ and $D_2 : (x, y) \mapsto f(x) - f(y)$. The nature of the maps and the partial bijection of the exact diamond principle 11.1 imply that there is only an extra summand $\mathbb{I}[i; i]$ in the decomposition of $\mathbb{W}$.

□

When f is surjective, the diamond is not exact as $\ker f \neq \{0\}$. For example, in the sequence $V \xrightarrow{D_1} V \oplus V \xrightarrow{D_2} W$, the couple $(u, 0)$ belongs to $\ker D_2 \setminus \text{im } D_1$ for any $u \in \ker f$, $u \neq 0$. Formulating the *Surjective Diamond Principle* requires an explicit expression of the interval submodules of the decomposition, using representative sequences. Recall that $\leq_{\mathbf{b}}$ and $\leq_{\mathbf{d}}$ are some prescribed orders on the indices $\{1, \ldots, n\}$ that depend only on the sequence of arrow orientations in the zigzags. In the following, $\leq_{\mathbf{b}}$ and $\leq_{\mathbf{d}}$ are defined on the quiver of the bottom diagram.

Theorem 11.3 (Surjective Diamond Principle)**.** *Suppose f is surjective of nullity* 1*, and ξ is a vector generating its kernel. Let $\{ u_j^{(b_j)} \nleftrightarrow \cdots \nleftrightarrow u_j^{(d_j)} \}_{j \in J}$ be a family of representative sequences representing an interval decomposition of $\mathbb{V}$. Up to a reordering of the indices in J, write ξ as:*

$$\xi = \alpha_1 u_1^{(i)} + \ldots + \alpha_p u_p^{(i)},$$

with $\alpha_j \neq 0$ for every $1 \leq j \leq p$ and $d_1 \leq_{\mathbf{d}} \ldots \leq_{\mathbf{d}} d_p$. Letting $b_{\ell_p} = \max_{\leq_{\mathbf{b}}} \{b_j\}_{j=1,\ldots,p}$, the modules $\mathbb{V}$ and $\mathbb{W}$ admit the following interval decompositions:

$$\begin{aligned} \mathbb{V} &\cong \mathbb{U} \oplus \textstyle\bigoplus_{1 \leq j \leq p} \mathbb{I}[b_j; d_j] \qquad \textit{and} \\ \mathbb{W} &\cong \mathbb{U} \oplus \mathbb{I}[b_{\ell_p}; i-1] \oplus \mathbb{I}[i+1; d_p] \oplus \\ &\quad \textstyle\bigoplus_{j=1}^{p-1} \mathbb{I}[b_{\ell_j}; d_j] \end{aligned}$$

where the pairing $(b_{\ell_j}, d_j)_{1 \leq j \leq p-1}$ is computed as follows (assuming b_{ℓ_p} and d_p are considered as already "paired"):

Algorithm 11.1: Pairing for Surjective Diamond

for j *from* 1 *to* $p-1$ **do**
  if b_j *not yet paired* **then**
    $b_{\ell_j} \leftarrow b_j$; pair b_{ℓ_j} with d_j;
  end
  else
    $b_{\ell_j} \leftarrow \max_{\substack{\leq_{\mathbf{b}} \\ k=1,\dots,p}} \{b_k : b_k \text{ not yet paired}\}$;
    pair b_{ℓ_j} with d_j;
  end
end

Sketch of Proof. The proof of the surjective diamond principle consists in working at the level of representative sequences in order to explicitly construct an internal direct sum decomposition of $\mathbb{W}$, from the one of $\mathbb{V}$. Intuitively, the underlying idea of the proof is to split the modules $\mathbb{V}$ and $\mathbb{W}$ at the index i of the surjective diamond, in order to consider their prefix $\mathbb{V}[1; i-1] = \mathbb{W}[1; i-1]$ and suffix $\mathbb{V}[i+1; n] = \mathbb{W}[i+1; n]$ separately. We manipulate the restricted representative sequences using $*$ and the scalar multiplication "$\cdot$" in order to "align" them with the kernel of f, and we join them back to represent submodules of $\mathbb{W}$. We prove the surjective diamond principle in section 12.2.

Consider the example, in computational topology, depicted in figure 11.1. The bottom and top zigzags of the diagram are related by a surjective diamond reflection and their interval decompositions at the homology level (dimension 1) are presented. Following the intuition, the decomposition for the bottom diagram corresponds to the addition and removal of each circular arc: for example, the birth of the closed interval $\mathbb{I}[2; 6]$ agrees with the insertion of the bottom arc (creating a "hole"), and its death agrees with the removal of the bottom arc when following the backward arrow $\bullet_6 \leftarrow \bullet_7$. By contrast, inserting the cap on top of the outer circle in the top zigzag links the three holes together and leads to a new pairing of the births and deaths of the corresponding intervals, according to theorem 11.3. In this example, the order relation $\leq_{\mathbf{b}}$ satisfies $7 \leq_{\mathbf{b}} 6 \leq_{\mathbf{b}} 4 \leq_{\mathbf{b}} 1 \leq_{\mathbf{b}} 2 \leq_{\mathbf{b}} 3 \leq_{\mathbf{b}} 5$ and $\leq_{\mathbf{d}}$ satisfies $1 \leq_{\mathbf{d}} 2 \leq_{\mathbf{d}} 4 \leq_{\mathbf{d}} 7 \leq_{\mathbf{d}} 6 \leq_{\mathbf{d}} 5 \leq_{\mathbf{d}} 3$. The arrow reflection first results in the apparition of intervals $\mathbb{I}[3; 3]$ and $\mathbb{I}[5; 5]$ (intervals $\mathbb{I}[b_{\ell_p}; i-1]$ and $\mathbb{I}[i+1; d_p]$ of the theorem). This induces a redistribution of birth and death indices. The interval dying at index 7 was born initially at index 3, which is

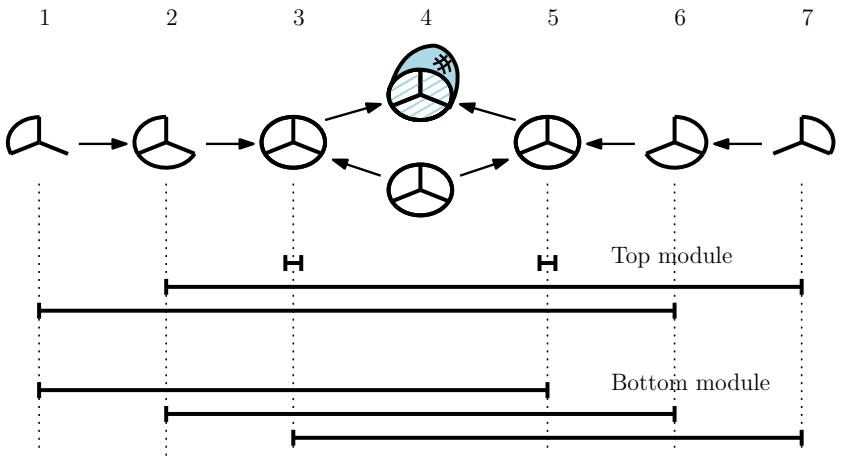


Figure 11.1: Two zigzags in computational topology, related by a surjective diamond at index 4 in 1-homology, and their interval decompositions. The decompositions are pictured as barcodes, where a bar from index i to index j represents an interval summand $\mathbb{I}[i; j]$.

already used in $\mathbb{I}[3; 3]$. It therefore gets assigned the largest available birth index w.r.t. $\leq_{\mathbf{b}}$, which is 2. Similarly, the interval dying at index 6 gets assigned 1 as new birth index.

Transposition Diamonds. Consider the diagram:

$$\begin{array}{ll} \mathbb{W} := & \\ & V_1 \longleftrightarrow \cdots \longleftrightarrow V_{i-1} \xrightarrow{b} W_i \xrightarrow{d} V_{i+1} \longleftrightarrow \cdots \longleftrightarrow V_n \\ \mathbb{V} := & \end{array} \qquad (11.4)$$

$$V_{i-1} \xrightarrow{a} V_i$$

We say that the representations $\mathbb{V}$ and $\mathbb{W}$ are related by a *transposition diamond* if the following diagram is *exact*:

$$\begin{array}{ccc} W_i & \xrightarrow{d} & V_{i+1} \\ {\scriptstyle b}\uparrow & & \uparrow{\scriptstyle c} \\ V_{i-1} & \xrightarrow{a} & V_i \end{array}$$

$V_{i-1} \xrightarrow{-D_1} V_i \oplus W_i \xrightarrow{-D_2} V_{i+1}$ with $D_1(v) = (a(v), b(v))$ and $D_2(x, y) = c(x) - d(y)$ such that $\text{im } D_1 = \ker D_2$

Note that the transposition diamond diagram (11.4) is similar to the exact diamond diagram (11.1) except that the diamond is "rotated by 90^o".

Theorem 11.4 (Transposition Diamond Principle)**.** *Given* $\mathbb{V}$ *and* $\mathbb{W}$ *related by a transposition diamond as above, we assume that the maps* a, b, c, d *are of two different types: injective of corank* 1 *and surjective of nullity* 1*. We have:*

1. *if* a *and* c *surjective of nullity* 1 *then* $\mathbb{V} \cong \mathbb{U} \oplus \mathbb{I}[b; i-1] \oplus \mathbb{I}[b'; i]$ *for some indices* $b, b' \leq i-1$*. Let* $(\dots, u, 0, 0\dots)$ *and* $(\dots v, a(v), 0\dots)$*,* $u, v \in V_{i-1}$*, be representative sequences for the interval summands* $\mathbb{I}[b; i-1]$ *and* $\mathbb{I}[b'; i]$ *respectively. There exists* $\alpha \in \Bbbk$ *such that* $v + \alpha u \in \ker b$ *and:*

(i) if $\alpha = 0$ *then* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[b; i] \oplus \mathbb{I}[b'; i-1]$*,*

(ii) if $\alpha \neq 0$ *then* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[\max_{\leq_{\mathbf{b}}}\{b, b'\}; i-1] \oplus \mathbb{I}[\min_{\leq_{\mathbf{b}}}\{b, b'\}; i]$*.*

2. *if* a *and* c *injective of corank* 1 *then* $\mathbb{V} \cong \mathbb{U} \oplus \mathbb{I}[i; d] \oplus \mathbb{I}[i+1; d']$ *for some indices* $d, d' \geq i+1$*. Let* $(0\dots 0, v, c(v), \dots)$ *and* $(0\dots 0, 0, u, \dots)$*,* $v \in V_i$ *and* $u \in V_{i+1}$*, be representative sequences for the interval summands* $\mathbb{I}[i; d]$ *and* $\mathbb{I}[i+1; d']$ *respectively. There exists* $\alpha \in \Bbbk$ *such that* $u + \alpha c(v) \in \operatorname{im} d$ *and:*

(i) if $\alpha = 0$ *then* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[i+1; d] \oplus \mathbb{I}[i; d']$*,*

(ii) if $\alpha \neq 0$ *then* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[i; \max_{\leq_{\mathbf{d}}}\{d, d'\}] \oplus \mathbb{I}[i+1; \min_{\leq_{\mathbf{d}}}\{d, d'\}]$*.*

3. *if* a *injective of corank* 1 *and* c *surjective of nullity* 1 *then:*
$\mathbb{V} \cong \mathbb{U} \oplus \mathbb{I}[i; d] \oplus \mathbb{I}[b; i]$ *and* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[i+1; d] \oplus \mathbb{I}[b; i-1]$*.*

4. *if* a *surjective of nullity* 1 *and* c *injective of corank* 1 *then:*
$\mathbb{V} \cong \mathbb{U} \oplus \mathbb{I}[i+1; d] \oplus \mathbb{I}[b; i-1]$ *and* $\mathbb{W} \cong \mathbb{U} \oplus \mathbb{I}[i; d] \oplus \mathbb{I}[b; i]$*.*

Sketch of Proof. The proof of the transposition diamond principle consists in turning a family of representative sequences that represents an interval decomposition of $\mathbb{V}$ into a family of representative sequences that represents an interval decomposition of $\mathbb{W}$. Specifically, every representative sequence for an interval summand of $\mathbb{V}$, with birth b and death d such that $b \notin \{i; i+1\}$ and $d \notin \{i-1, i\}$, is matched via a natural map with a representative sequence for an interval summand of $\mathbb{W}$ with same birth and death. Cases 1.*(i)*, 2.*(i)*, 3. and 4. consider the cases where two intervals with birth in $\{i; i+1\}$ and death in $\{i-1; i\}$ exchange their endpoints. Finally, the cases 1.*(ii)* and 2.*(ii)* are more intricate: given two representative sequences $\mathbf{u}$ and $\mathbf{v}$ with births in $\{i; i+1\}$ or deaths in $\{i-1; i\}$, we form the representative sequence $\mathbf{u} * (\alpha \cdot \mathbf{v})$. This explains the use of $\max_{\leq_{\mathbf{b}}}$ and $\max_{\leq_{\mathbf{d}}}$ to describe the birth and death of the new interval. We prove the transposition diamond principle in section 12.1.

11.2 Zigzag Persistent Homology Algorithm.

We assume familiarity with homology theory, referring the reader to [59] for an introduction. In the following we use simplicial homology with coefficients in a field $\Bbbk$.

A *simplicial complex* $\mathbf{K}$ on a finite set of *vertices* V is a collection of simplices $\{\sigma\}$, $\sigma \subseteq V$, such that $\tau \subseteq \sigma \in \mathbf{K} \Rightarrow \tau \in \mathbf{K}$. The dimension $d = |\sigma| - 1$ of σ is its number of elements minus 1. The group of d-chains, denoted $\mathbf{C}_d(\mathbf{K})$, of $\mathbf{K}$ is the group of formal sums of d-simplices with $\Bbbk$ coefficients. The *boundary operator* is a linear operator $\partial_d : \mathbf{C}_d(\mathbf{K}) \to \mathbf{C}_{d-1}(\mathbf{K})$ such that $\partial_d \sigma = \partial_d [v_0, \cdots, v_d] = \sum_{i=0}^{d} (-1)^i [v_0, \cdots, \widehat{v_i}, \cdots, v_d]$, where $\widehat{v_i}$ means v_i is deleted from the list. The kernel of ∂_d, denoted by $\mathbf{Z}_d(\mathbf{K})$, is the group of *d-cycles* and the image of ∂_d, denoted by $\mathbf{B}_{d-1}(\mathbf{K})$, is the group of *$(d-1)$-boundaries*. Observing $\partial_d \circ \partial_{d+1} = 0$, the d^{th} homology group $\mathbf{H}_d(\mathbf{K})$ of $\mathbf{K}$ is defined to be the quotient $\mathbf{H}_d(\mathbf{K}) = \mathbf{Z}_d(\mathbf{K}) / \mathbf{B}_d(\mathbf{K})$. We drop the dimension d by considering the external direct sum $\mathbf{C}(\mathbf{K}) = \bigoplus_d \mathbf{C}_d(\mathbf{K})$ and the boundary operator $\partial : \mathbf{C}(\mathbf{K}) \to \mathbf{C}(\mathbf{K})$ extended by linearity. We define $\mathbf{Z}(\mathbf{K})$, $\mathbf{B}(\mathbf{K})$ and $\mathbf{H}(\mathbf{K})$ similarly. Because the coefficients belongs to a field $\Bbbk$, these are vector spaces, and since $\mathbf{K}$ is a finite simplicial complex, they have finite dimensions. In this context, the insertion of a simplex σ of dimension d in a simplicial complex $\mathbf{K}$, $\mathbf{K} \hookrightarrow \mathbf{K} \cup \{\sigma\}$ may either create a homology class in dimension d, or destroy a homology class in dimension $d-1$. In the first case, the map induced at homology level is injective of corank 1. In the second case, it is surjective of nullity 1, and its kernel is spanned by $[\partial\sigma]$.

For ease of exposition, throughout the main body of the paper we assume the field of coefficients to be $\mathbb{Z}_2$. Nevertheless, our approach is not tied to $\mathbb{Z}_2$, and the proofs of the diamonds principles are written for an arbitrary field of coefficients $\Bbbk$.

Zigzag Filtrations. A *zigzag filtration* on an A_n-type quiver $\mathcal{Q}$ is an assignment of a simplicial complex $\mathbf{K}_i$ for each vertex $\bullet_i$, and of an elementary inclusion corresponding either to a simplex insertion $\mathbf{K}_i \hookrightarrow \mathbf{K}_{i+1}$ (i.e. $\mathbf{K}_{i+1} = \mathbf{K}_i \cup \{\sigma\}$) or to a simplex deletion $\mathbf{K}_i \hookleftarrow \mathbf{K}_{i+1}$ (i.e. $\mathbf{K}_{i+1} = \mathbf{K}_i \setminus \{\sigma\}$) for each arrow $\bullet_i \longleftrightarrow \bullet_{i+1}$. A zigzag filtration $\mathbb{K}$ induces a $\mathbb{Z}_2$-representation $\mathbf{H}(\mathbb{K})$ of $\mathcal{Q}$ at the homology level, whose homology groups and linear maps are induced by the simplicial complexes and elementary inclusions. Computing zigzag persistence consists in computing the interval decomposition of this zigzag module $\mathbf{H}(\mathbb{K})$, given the sequence of simplex

insertions and deletions. Note that for an index $i \in \{2, \cdots, n\}$ there is at most one interval with birth i in the interval decomposition of $\mathbf{H}(\mathbb{K})$, and for an index $i \in \{1, \cdots, n-1\}$ there is at most one interval with death i. There may be as many births equal to 1 and deaths equal to n. This is true generally when the maps between the simplicial complexes are elementary inclusions. We call *standard persistence* the case where all arrows are oriented in the same direction. In this case, $\mathbb{K}$ is called a *standard filtration*.

Standard Persistence and Matrix Reduction. We review the presentation of standard persistence as in [34], where explicit chains representing a compatible homology basis are maintained. For a standard filtration $\mathbb{K} = \mathbf{K}_m \xleftarrow{\tau_m} \cdots \xleftarrow{\tau_2} \mathbf{K}_1 \xleftarrow{\tau_1} \emptyset$, there exist chains $\widehat{\tau}_m, \cdots, \widehat{\tau}_1 \in \mathbf{C}(\mathbf{K}_m)$, a partition of the indices $\{1, \cdots, m\} = F \sqcup G \sqcup H$, and a bijective pairing $G \leftrightarrow H$, denoted by $\mathcal{P} \subseteq G \times H$, satisfying the following conditions:

1. for all i, $\mathbf{C}(\mathbf{K}_i) = \langle \widehat{\tau}_i, \cdots, \widehat{\tau}_1 \rangle$,
2. for all $f \in F$, $\partial \widehat{\tau}_f = 0$, and
3. for all pairs $(g, h) \in \mathcal{P}$, $\partial \widehat{\tau}_h = \widehat{\tau}_g$ and hence $\partial \widehat{\tau}_g = 0$.

We call such a partitioned set of chains an *encoding of the persistence module*. Condition 1. is equivalent to the fact that every $\widehat{\tau}_i$ admits τ_i as leading term (i.e. $\widehat{\tau}_i = \varepsilon_1 \tau_1 + \cdots + \varepsilon_i \tau_i$, with $\varepsilon_i \neq 0$). According to Theorem 2.6 of [34], an encoding of a standard persistence module encodes completely its interval decomposition. Indeed, for $f \in F$, $\widehat{\tau}_f$ is a cycle created at index f and whose homology class is non-zero in $\mathbf{K}_m$. For $g \in G$, paired with $h \in H$, $\widehat{\tau}_g$ is a cycle created at index g and whose homology class is non-zero from index g up to index $h-1$, after which it becomes the boundary of the chain $\widehat{\tau}_h$. One can read directly the persistent interval from this encoding. Indeed, let $\mathbf{H}(\mathbb{K}) = \mathbf{H}(\mathbf{K}_m) \longleftarrow \cdots \longleftarrow \mathbf{H}(\mathbf{K}_1) \longleftarrow 0$ be the corresponding persistence module. The representative sequences induced by the cycles $\widehat{\tau}_f$ for $f \in F$ and $\widehat{\tau}_g$ for $g \in G$ are respectively:

$$[\widehat{\tau}_f]^{(m)} \leftrightarrow \cdots \leftrightarrow [\widehat{\tau}_f]^{(f)} \quad \text{and} \quad [\widehat{\tau}_g]^{(h-1)} \leftrightarrow \cdots \leftrightarrow [\widehat{\tau}_g]^{(g)}$$

where $[\widehat{\tau}_i]^{(j)}$ refers to the homology class of $\widehat{\tau}_i$ in the simplicial complex $\mathbf{K}_j$. This is well-defined because $\widehat{\tau}_i$ has τ_i as leading term and $j \geq i$ in the previous sequences. Moreover, these homology classes are pointwise independent. By virtue of proposition 10.2, these sequences represent the interval decom-

position[1] of the zigzag module:

$$\mathbf{H}(\mathbf{K}) \cong \bigoplus_{f \in F} \mathbb{I}[m; f] \bigoplus_{(g,h) \in \mathcal{P}} \mathbb{I}[h-1; g]$$

In the following, we represent an encoding by an $(m \times m)$-matrix[2] $\mathbf{M}$ with $\mathbb{Z}_2$ coefficients, where each column j, denoted by col_j, represents the chain $\widehat{\tau}_j$ in the basis $\{\tau_1, \cdots, \tau_m\}$ of $\mathbf{C}(\mathbf{K}_m)$. Due to Condition 1., $\mathbf{M}$ is upper-triangular, with non-zero elements on the diagonal. For a non-zero column col, we denote by $\mathrm{low}(\mathsf{col})$ the row index of its lowest non-zero element. If the column is null, low is undefined.

For any chain $c \in \mathbf{C}(\mathbf{K}_m)$, represented as a column col in $\mathbf{M}$, and for any set of indices $I \subseteq \{1, \cdots, m\}$, we can express c as a linear combination of the chains $\{\widehat{\tau}_i\}_{i \in I}$ (whenever possible) using the following reduction:

Algorithm 11.2: `Reduction(col,`I`)`

while $\exists i_0 \in I$ *with* $\mathrm{low}(\mathsf{col}_{i_0}) = \mathrm{low}(\mathsf{col})$ **do**
  $\mathsf{col} \leftarrow \mathsf{col} + \mathsf{col}_{i_0}$;
end

If the output value of col is 0, then we have computed an expression $c + \sum_{i \in I'} \widehat{\tau}_i = 0$, where $I' \subseteq I$ is the set of indices i_0 picked in the **while** loop. Otherwise, if $\mathsf{col} \neq 0$, then $c \notin \langle \widehat{\tau}_i \rangle_{i \in I}$. The algorithm is valid because $\mathrm{low} : \{1, \cdots, m\} \to \{1, \cdots, m\}$ is injective in $\mathbf{M}$ (actually, the identity) and every column addition strictly reduces $\mathrm{low}(\mathsf{col})$. This reduction is at the heart of the standard persistence algorithm.

For any index $i \in \{1, \cdots m\}$, the boundary group $\mathbf{B}(\mathbf{K}_i)$ is generated by the cycles $\widehat{\tau}_g$, for $g \in G$ paired with an index $h \in H$ such that $g < h \leq i$. The cycle group $\mathbf{Z}(\mathbf{K}_i)$ is generated by all the cycles $\widehat{\tau}_j$ for $j \in F \sqcup G$ and $j \leq i$. In particular, for any chain $c \in \mathbf{C}(\mathbf{K}_m)$, we can express c in the basis $\{\widehat{\tau}_g\}_{g \in G}$ of $\mathbf{B}(\mathbf{K}_m)$ (or prove that it does not belong to $\mathbf{B}(\mathbf{K}_m)$) by representing c as a column col and running `Reduction(col,`G`)`. Similarly for expressing c in $\mathbf{Z}(\mathbf{K}_m)$ by running `Reduction(col,`$F \sqcup G$`)`.

[1] Since the persistence module is represented backwards, birth and death times are reversed compared to the standard persistence setting. For instance, in this example, m is the birth time and f is the death time.

[2] This matrix can be viewed as a compact encoding of the matrices R and V in the $R = DV$ decomposition of the boundary matrix D (see [34]).

Algorithm 11.3: Zigzag Persistence Algorithm

$\mathbf{M} \leftarrow \emptyset$;
foreach *arrow* $\bullet_i \xleftrightarrow{\sigma} \bullet_{i+1}$ **do**
  if *the arrow is forward* ($\xrightarrow{\sigma}$) **then**
    compute $\partial\sigma = \sum_{g\in G'} \widehat{\tau}_g + \sum_{f\in F'} \widehat{\tau}_f$ via
      $\texttt{Reduction}(\mathsf{col}_{\partial\sigma}, F \sqcup G)$;
    if $F' = \emptyset$ **then** $\texttt{injective\_diamond}(\mathbf{M},\ G')$;
    else $\texttt{surjective\_diamond}(\mathbf{M},\ F',\ G')$;
  end
  if *the arrow is backward* ($\xleftarrow{\sigma}$) **then**
    let $\xleftarrow{(\tau_{i_0})}$ be the backward arrow of $\mathbb{K}_i[m;0]$ s.t. $\tau_{i_0} = \sigma$;
    for $j = i_0 + 1 \cdots m$ **do** $\texttt{transposition\_diamond}(\mathbf{M},\ j-1,\ j)$;
    restrict the encoding to $\mathbb{K}_{i+1}[m-1;0]$;
  end
end

Overview of the Algorithm. Given an input zigzag filtration $\mathbb{K}$, we want to compute the intervals in a direct sum decomposition of the induced zigzag module $\mathbf{H}(\mathbb{K})$ at the homology level. By convention, we denote the complexes in $\mathbb{K}$ by $\mathbf{K}'_j$ for $1 \leq j \leq n$, so as $\mathbb{K}$ is written:

$$\mathbf{K}'_1 \longleftrightarrow \cdots \longleftrightarrow \mathbf{K}'_i \xleftrightarrow{\sigma} \mathbf{K}'_{i+1} \longleftrightarrow \cdots \longleftrightarrow \mathbf{K}'_n \tag{11.5}$$

For two indices b and d, $b \leq d$, we denote by $\mathbb{K}[b,d]$ the restriction of $\mathbb{K}$ to the simplicial complexes contained between complexes $\mathbf{K}'_b$ and $\mathbf{K}'_d$ (included). For a fixed index $i \in \{1,\ldots,n\}$, denote by m the number of simplices of $\mathbf{K}'_i$ and define $\mathbb{K}_i$ to be the following zigzag filtration:

(11.6)

$$\begin{array}{ccccccccc} \mathbf{K}'_1 & \longleftrightarrow & \cdots & \longleftrightarrow & \mathbf{K}'_i = \mathbf{K}_m & \xleftarrow{\tau_m} & & & \\ \mathbf{K}_{m-1} & \xleftarrow{\tau_{m-1}} & \mathbf{K}_{m-2} & \xleftarrow{\tau_{m-2}} & \cdots & \xleftarrow{\tau_1} & \emptyset & & \end{array}$$

where the prefix made of the restriction of $\mathbb{K}_i$ to its i leftmost vector spaces is equal to $\mathbb{K}[1;i]$, and the suffix of $\mathbb{K}_i$, denoted by $\mathbb{K}_i[m;0]$, made of the $m+1$ rightmost vector spaces contains only backward arrows. The simplices

$\tau_1, \cdots, \tau_m$ of $\mathbf{K}'_i$ are removed in an arbitrary order[3] in the suffix $\mathbb{K}_i[m;0]$. Note that we refer to the set of i leftmost indices of the zigzag filtration $\mathbb{K}_i$ using the interval notation $[1;i]$, and we refer to the set of its $m+1$ rightmost indices using the interval notation $[m;0]$. The algorithm is iterative and maintains, at the end of iteration i, a set of chains of $\mathbf{K}'_i$ that encodes an interval decomposition of the standard persistence module $\mathbb{K}_i[m;0]$ and is compatible with the whole zigzag module $\mathbb{K}_i$. We call this set a *compatible basis of* $\mathbb{K}_i$ and detail its definition later. During iteration $i+1$, there are two cases:

1. $\mathbf{K}'_i \xrightarrow{\sigma} \mathbf{K}'_{i+1}$ is forward: in this case, we turn the set of chains of $\mathbf{K}_i$ forming a compatible basis of $\mathbb{K}_i$, in the following bottom module, into a set of chains of $\mathbf{K}'_{i+1}$ forming a compatible basis of $\mathbb{K}_{i+1}$, in the following top module:

$$\begin{array}{ccccccccccc} & & & & \mathbf{K}'_{i+1} & & & & & & \\ & & & \overset{\sigma}{\nearrow} & & \overset{\sigma}{\nwarrow} & & & & & \\ \mathbf{K}'_1 \cdots \longleftrightarrow & \mathbf{K}'_i & & & & & \mathbf{K}'_i & \xleftarrow{\tau_m} & \mathbf{K}_{m-1} & \xleftarrow{\tau_{m-1}} & \cdots \emptyset \\ & & & \underset{\mathbb{1}}{\nwarrow} & & \underset{\mathbb{1}}{\nearrow} & & & & & \\ & & & & \mathbf{K}'_i & & & & & & \end{array} \tag{11.7}$$

Note that the bottom module is simply the module $\mathbb{K}_i$ with two extra identity arrows added for convenience. In particular, it does not change the interval decomposition, up to a shift in the indices. The top module is a valid form for $\mathbb{K}_{i+1}$. The transformation between the two modules is an *arrow reflection*. At the homology level, the zigzag modules corresponding to the bottom and top zigzag filtration are related by an injective diamond if the map $\mathbf{H}(\mathbf{K}'_i) \to \mathbf{H}(\mathbf{K}'_i \cup \{\sigma\})$ induced by the insertion of σ is injective, and are related by a surjective diamond if the map is surjective.

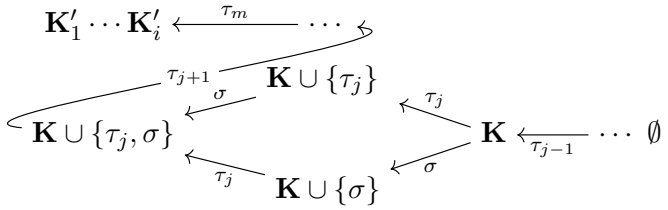
2. $\mathbf{K}'_i \xleftarrow{\sigma} \mathbf{K}'_{i+1}$ is backward: in this case, there exists an index $i_0 \in [m;1]$ such that $\sigma = \tau_{i_0}$. For every j from i_0+1 to m, we transpose the consecutive arrows $\xleftarrow{(\tau_j)}$ and $\xleftarrow{(\tau_{i_0}=\sigma)}$, so as to get in the end:

$$\begin{array}{l} \mathbf{K}'_1 \cdots \mathbf{K}'_i \xleftarrow{\tau_{i_0}=\sigma} \mathbf{K}'_{i+1} \xleftarrow{\tau_m} \mathbf{K}'_{i+1} \setminus \{\tau_m\} \xleftarrow{\tau_{m-1}} \\ \cdots \xleftarrow[\tau_{i_0+1}]{} \mathbf{K}'_{i+1} \setminus \{\tau_m, \ldots, \tau_{i_0+1}\} \xleftarrow[\tau_{i_0-1}]{} \cdots \emptyset \end{array}$$

Under each arrow transposition, we update the compatible basis of $\mathbb{K}_i$. We finally restrict the basis to the simplices of $\mathbf{K}'_{i+1}$, that is contained in $\mathbf{K}'_i$.

[3] In the algorithm described below, the simplices are removed in the reverse order of their insertion.

Note that an arrow transposition consists in going from the bottom to the top module in the following diagram:



At the homology level, the zigzag modules corresponding to the bottom and top zigzag filtrations are related by a transposition diamond by virtue of the Mayer-Vietoris theorem [59] (see also [24] for the case of the exact diamond).

Compatible Basis for a Zigzag Modules. Given $\mathbb{K}_i$ as in (11.6), we suppose that we have $\widehat{\tau}_m, \ldots, \widehat{\tau}_1 \in \mathbf{C}(\mathbf{K}_m)$, a partition $\{1, \ldots, m\} = F \sqcup G \sqcup H$, and a bijective pairing $\mathcal{P} \subseteq G \times H$ that give an encoding of the module $\mathbb{K}_i[m; 0]$, which is a standard persistence module. We say that this encoding is *compatible* with the whole zigzag module $\mathbb{K}_i$ iff there exists a direct sum decomposition $\mathbb{K}_i = \bigoplus_j \mathbb{U}_j$ into interval summands $\mathbb{U}_j \cong \mathbb{I}[b_j; d_j]$, together with bijective matching between:

- F and the interval summands $\mathbb{U}_j$ with $b_j \in [1; i]$ and $d_j \in [m; 0]$,
- G and the interval summands $\mathbb{U}_j \cong \mathbb{I}[b_j; d_j]$ with $b_j, d_j \in [m-1; 0]$,

such that the submodules $\mathbb{U}_j$ are represented by sequences

$$\mathbb{U}_j = (\#, \ldots, \#, [\widehat{\tau}_f]_m, \ldots, [\widehat{\tau}_f]_f, 0, \ldots)$$

for f in F, and

$$\mathbb{U}_j = (\ldots, [0]_m, \ldots, [0]_h, [\widehat{\tau}_g]_{h-1}, \ldots, [\widehat{\tau}_g]_g, 0, \ldots)$$

for $g \in G$. The symbol # indicates that the vector space element at this position in the sequence can be arbitrary. Additionally, we maintain the *birth* $b[\mathsf{col}_j]$ of each column col_j which is equal to the birth of the interval submodule associated to $\widehat{\tau}_j$ in the bijection mentioned above.

Algorithm. The algorithm is purely online: we assume the input to be a stream of couples $(\sigma_i, \partial\sigma_i)_{i \geq 1}$ of a simplex σ_i and its boundary $\partial\sigma_i$, together with a flag specifying the direction of the arrow. At the beginning of iteration $i+1$ of the algorithm, we suppose we have a compatible basis of $\mathbb{K}_i$. The

Algorithm 11.4: `injective_diamond`($\mathbf{M}$, G')

let H' be the set of indices $h \in H$ paired with some $g \in G'$ in $\mathcal{P}$;
$\widehat{\tau}_{m+1} \leftarrow \sigma + \sum_{h \in H'} \widehat{\tau}_h$ and let col_{m+1} represent $\widehat{\tau}_{m+1}$; set $F \leftarrow F \cup \{m+1\}$;
add column col_{m+1} to $\mathbf{M}$ and row_{m+1} that is 0 everywhere except at index $m+1$;

algorithm is iterative and maintains, at the beginning of step i, a matrix $\mathbf{M}$ that represents an encoding of $\mathbb{K}_i[m;0]$ that is compatible with $\mathbb{K}_i$ as defined previously. The procedure is described in algorithm 11.3.

Note that the intervals are computed when *restricting* the set of chains of the compatible basis at the end of the processing of a backward arrow. Indeed, after the sequence of calls to `transposition_diamond`, the matrix $\mathbf{M}$ maintains a compatible basis for $\mathbb{K}_i$ after transposing the arrows. The chains of the compatible basis are defined on $\mathbf{K}'_i$, and we restrict them to $\mathbf{K}'_{i+1} = \mathbf{K}'_i \setminus \{\sigma\}$. If there exists a chain $\widehat{\tau}_m$ with $m \in F$ in the encoding, and let b_m be its birth, we record an interval $\mathbb{I}[b_m; i]$ in the decomposition of the zigzag persistence module $\mathbf{H}(\mathbb{K})$. Algorithmically, the restriction consists in removing the rightmost column and the bottom row of matrix $\mathbf{M}$.

Complexity. Denote by n the total number of arrows in the quiver and by m the maximal number of simplices of a simplicial complex in the zigzag filtration. The matrix $\mathbf{M}$ contains at most m columns and m rows. The subroutine `Reduction` proceeds to at most $O(m)$ column additions and hence $O(m^2)$ operations. We prove in section 11.3 that the cost of the procedure `surjective_diamond` is $O(m^2)$ operations and the cost of the procedure `injective_diamond` is $O(1)$. We also prove that the cost of `transposition_diamond` is $O(m)$, and this subroutine is called $O(m)$ times during an iteration of the algorithm. Finally, the time complexity of the algorithm to compute zigzag persistent homology is $O(nm^2)$, and its memory complexity is $O(m^2)$.

11.3 Arrow Reflections and Transpositions.

Recall that, for ease of exposition, we assume the field of coefficients to be $\mathbb{Z}_2$. Our proofs are however written for an arbitrary field $\Bbbk$.

Algorithm 11.5: `surjective_diamond`($\mathbf{M}$, F', G')

set $\mathsf{col}_{f_p} \leftarrow \mathsf{col}_{f_1} + \cdots + \mathsf{col}_{f_p}$; set $b_{\ell_p} \leftarrow \max_{\leq_{\mathbf{b}}}\{b_1, \cdots, b_p\}$;
for *j from* 1 *to* $p-1$ **do**
  set $b \leftarrow b_{f_j}$;
  while *there exists* $j_0 < j$ *or* $j_0 = p$ *such that* $b_{\ell_{j_0}} = b$ **do**
    $\mathsf{col}_{f_j} \leftarrow \mathsf{col}_{f_j} + \mathsf{col}_{f_{j_0}}$; $b \leftarrow \mathrm{pred}_{\mathbf{b}}(b)$;
  end
  set $b_{\ell_j} \leftarrow b$;
end
$F \leftarrow F \setminus \{f_p\}$; $G \leftarrow G \cup \{f_p\}$;
let H' be the set of indices $h \in H$ paired with some $g \in G'$ in $\mathcal{P}$;
$\widehat{\tau}_{m+1} \leftarrow \sigma + \sum_{h \in H'} \widehat{\tau}_h$ and let col_{m+1} represent $\widehat{\tau}_{m+1}$;
add column col_{m+1} to $\mathbf{M}$ and row_{m+1} that is 0 everywhere except at index $m+1$;
$H \leftarrow H \cup \{m+1\}$; and pair f_p and $m+1$ in $\mathcal{P}$

Arrow Reflection. As said before, computing an arrow reflection consists in traversing a diamond from bottom to top. We distinguish between the case where the linear map, induced at homology level by the simplex insertion $\xrightarrow{\sigma}$, is injective (of corank 1), and the case where the map is surjective (of nullity 1).

Injective Diamond. If $\partial\sigma = \sum_{g \in G'} \widehat{\tau}_g$ is a sum of boundaries, its insertion creates a new cycle class at the homology level, represented by $\widehat{\tau}_{m+1}$ and constructed in algorithm 11.4. The induced map at the homology level is injective.

It is easy to verify that, after the update, $\mathbf{M}$ represents an encoding of $\mathbb{K}_{i+1}[m+1;0]$. The sequence with one non zero element equal to $[\widehat{\tau}_{m+1}]$ at index $i+1$ is a representative sequence for an interval submodule of $\mathbf{H}(\mathbb{K}_{i+1})$ isomorphic to $\mathbb{I}[i+1;i+1]$. It is a summand because the element $[\widehat{\tau}_{m+1}]$ is linearly independent of the elements $\{[\widehat{\tau}_f]\}_{f \in F}$ in $\mathbf{H}(\mathbf{K}_{i+1})$. Finally, the classes $[\widehat{\tau}_f]_{f \in F}$ remain independent in $\mathbf{H}(\mathbf{K}_{i+1})$ because the map induced at the homology level by the insertion of σ is injective. Consequently, we conclude, using the injective diamond principle 11.2, that the encoding represented by $\mathbf{M}$ is compatible with $\mathbb{K}_{i+1}$.

Surjective Diamond. If $\partial\sigma = \sum_{f \in F'} \widehat{\tau}_f + \sum_{g \in G'} \widehat{\tau}_g$, with $F' \subseteq F$ and $G' \subseteq G$, the kernel of the morphism induced at the homology level by

Algorithm 11.6: `transposition_diamond`($\mathbf{M}$, i, $i+1$)

$\mathbf{z} \leftarrow \mathbf{M}[i+1][i]$; transpose row_i and row_{i+1};
if $\mathbf{z} = 0$ **then** transpose col_i and col_{i+1};
exchange i and $i+1$ in F, G, H and $\mathcal{P}$;
else
  $\mathbf{M}[i] \leftarrow \mathsf{col}_i + \mathsf{col}_{i+1}$;
  switch $\widehat{\tau}_i, \widehat{\tau}_{i+1}$ **do**
    case *(both cycles)* $i_0 \leftarrow \operatorname{argmin}_{\leq_{\mathbf{d}}}\{b_i, b_{i+1}\}$;
    $\mathbf{M}[i+1] \leftarrow \mathsf{col}_{i_0}$;
    case *(exactly one cycle)* let $\widehat{\tau}_{i_0}$, $i_0 \in \{i, i+1\}$, be the cycle;
    $\mathbf{M}[i+1] \leftarrow \mathsf{col}_{i_0}$;
    case *(no cycle)* $i, i+1 \in H$ are paired with $g_i, g_{i+1} \in G$;
    let $i_0, i_1 = i, i+1$ such that $g_{i_0} < g_{i_1}$;
    $\mathbf{M}[g_{i_1}] \leftarrow \mathsf{col}_{g_i} + \mathsf{col}_{g_{i+1}}$; $\mathbf{M}[i+1] \leftarrow \mathsf{col}_{i_0}$;
    if $i_0 \neq i+1$ **then** exchange i and $i+1$ in F, G, H and $\mathcal{P}$;
  endsw
end

the insertion of σ is spanned by $[\partial\sigma] = \sum_{f \in F'}[\widehat{\tau}_f]$, and this morphism is surjective. Write $F' = \{f_1, \ldots, f_p\}$, where the f_j are ordered such that $d_{f_1} \leq_{\mathbf{d}} \ldots \leq_{\mathbf{d}} d_{f_p}$. We define, for the set of births $\{b_{f_1}, \ldots, b_{f_p}\}$, the function $\mathrm{pred}_{\mathbf{b}}(b)$ that returns the predecessor of the index b w.r.t the order $\leq_{\mathbf{b}}$ among the set $\{b_{f_1}, \ldots, b_{f_p}\}$. The procedure described in algorithm 11.5 gives the change of compatible encoding underlying the new pairing presented in the surjective diamond principle 11.3. Note that the births b_{ℓ_j} are the ones described in the surjective diamond principle 11.3. For details on the correction of the algorithm, we refer to section 12.2. In particular, the column operations done in algorithm 11.5 are exactly the column operations done when reducing matrix $\mathbf{X}$ in algorithm 12.1 (when matching col_{f_p} with $\mathbf{x}_0$ and, for $j \geq 0$, col_{f_j} with $\mathbf{x}_j$).

In conclusion, we observe that algorithm 11.5 performs only left to right column additions. Hence, $\mathbf{M}$ is upper-triangular and its is easy to verify that the output matrix $\mathbf{M}$ stores an encoding of $\mathbf{K}_{i+1}[m+1; 0]$. It is compatible with the zigzag $\mathbb{K}_{i+1}$ by virtue of the surjective diamond principle. We prove in lemma 12.3 that this algorithm proceeds to a linear number of column operations and hence has complexity $O(m^2)$.

Arrow Transposition. Algorithmically, transposing the consecutive arrows $\prec(\tau_{i+1})\text{—}\bullet\prec(\tau_i)\text{—}$ consists in transposing rows $i+1$ and i in the matrix $\mathbf{M}$, corresponding to simplices τ_{i+1} and τ_i (see figure 11.2).

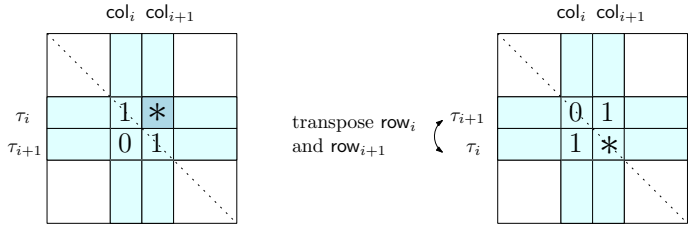


Figure 11.2: Row transposition

We must maintain the property that $\mathbf{M}$ represents an encoding of the suffix $\mathbb{K}_i[m;0]$ of the zigzag filtration after arrow transposition. Moreover, the new encoding must agree with the interval decomposition described by the transposition diamond principle 11.4. After transposition, $\mathbf{M}$ is not upper triangular anymore. If $\mathbf{z}=0$, we fall into one of cases $1.(i)$, $2.(i)$, (3) or (4) of the transposition diamond principle. Hence, for $\mathbf{z}=0$, we transpose the columns col_i and col_{i+1} and update the pairing $\mathcal{P}$ in consequence (the chains represented by col_i and col_{i+1} are paired with the same chains, but are now represented by the $(i+1)^{\text{st}}$ and i^{th} columns of $\mathbf{M}$ respectively). To avoid confusion with the notation col_j that may not correspond to the j^{th} column of $\mathbf{M}$ after transposition, we denote the j^{th} column of $\mathbf{M}$ by $\mathbf{M}[j]$.

If $\mathbf{z}\neq 0$, we fall into one of the cases $1.(ii)$, $2.(ii)$, (3) or (4) of the transposition diamond principle 11.4. Hence, we set $\mathbf{M}[i]$ to be the sum $\mathsf{col}_i+\mathsf{col}_{i+1}$, and $\mathbf{M}[i+1]$ to be either col_i or col_{i+1} depending on birth and death indices. We distinguish between the cases where both col_i and col_{i+1} represent cycles (i.e. $i,i+1\in F\sqcup G$), where only one represents a cycle (i.e. $i\in F\sqcup G$ and $i+1\in H$ or the other way around), and where none represents a cycle $(i,i+1\in H)$. We present the procedure in algorithm 11.6.

It is easy to verify that these updates turn $\mathbf{M}$ into an encoding of $\mathbb{K}_i[m;0]$ after arrow transposition. Using the transposition diamond principle 11.4, this encoding is compatible with the whole zigzag. In particular, the sum of cycles corresponds to the "sum" of representative sequence computed in the proof of the transposition diamond principle (section 12.1). The algorithm `transposition_diamond` proceeds to a constant number of column additions in $\mathbf{M}$, and hence has complexity $O(m)$.

Data	$\lvert V\rvert$	d	η	ρ	$\dim \mathbf{K}$	$\max \lvert\mathbf{K}\rvert$	nb. arrows	T_{RT}	$T_{\texttt{Dio}}$
Cli	2000	4	3	3.2	3	66172	7408112	99 sec.	272 sec.
Bro	595	25	3	3.2	3	1219926	277610862	2498 sec.	65331 sec.

Figure 11.3: Timings for the zigzag persistence algorithms.

Experiments. As a proof of concept, we have implemented our zigzag persistence algorithm in `C++`. The implementation relies on a sparse matrix representation, as in the standard persistence algorithm [42]. In figure 11.3, we compare the performance of our implementation with the one of the `Dionysus` library [56]. The **Cli** data set is a set of points from the *Clifford dataset* described in [61], which admits as underlying spaces a topological circle at small scales, a torus at larger scales and a 3-sphere at even larger scales. The **Bro** data set contains 5×5 *high-contrast patches* derived from natural images, interpreted as vectors in $\mathbb{R}^{25}$, from the Brown database [26].

We construct *oscillating Rips zigzags* on these sets of points. Details on the constructions are listed in figure 11.3, like the number of points $|V|$, the ambient dimension d, the parameter η and ρ for the oscillating Rips, the maximal dimension of the complex $\dim \mathbf{K}$, the maximal size of a complex in the filtration $\max |\mathbf{K}|$, the total number of arrows of the zigzag filtration "nb. arrows". The timings for the zigzag persistence algorithm with reflections and transpositions is denoted by T_{RT} and the timings for the algorithm, based on the right-filtration [24, 25], implemented in `Dionysus` is denoted by $T_{\texttt{Dio}}$.

The speed-up is encouraging. In particular, the performance of the two algorithms is comparable on the **Cli** data set, but the algorithm introduced in this chapter scales much better to the data set **Bro** compared to `Dionysus`. Note that in `Dionysus`, the processing of backward arrows is much slower than the processing of forward arrows in the case of **Bro**. This is partly due to implementation issues. However, counting the timings of `Dionysus` as twice the running time for forward arrows, our algorithm remains faster.

Chapter 12

Proofs of the Diamond Principles

12.1 Proof of the Transposition Diamond Principle.

Let $\mathbb{I}[b;d]$ be an interval summand of $\mathbb{V}$. If $b > i+1$ or $d < i-1$, the restriction principle 10.2 implies that the interval is matched with an interval summand $\mathbb{I}[b;d]$ of $\mathbb{W}$.

If $b < i$ and $d > i$, we prove that the interval is matched with an interval summand $\mathbb{I}[b;d]$ of $\mathbb{W}$. Note that if $(\dots u, a(u), c \circ a(u) \dots)$ is a representative sequence for the submodule of $\mathbb{V}$ isomorphic to the summand $\mathbb{I}[b;d]$, we use $(\dots u, b(u), d \circ b(u) \dots)$ as representative sequence for the submodule of $\mathbb{W}$, where the "$\dots$" are the same prefixes and suffixes in the two sequences. The new sequence is a representative sequence because, by commutativity, $d \circ b(u) = c \circ a(u)$ and $b(u) \neq 0$. We prove that the values at index i of the representative sequences for $\mathbb{W}$ remain independent. Let $\{ u_j^{(b_j)} \leftrightarrows \cdots \leftrightarrows u_j^{(d_j)} \}_{j\in J}$ be representative sequences for the interval summands of $\mathbb{V}$ satisfying $i \in (b_j; d_j)$. In particular, for every $j \in J$, $u_j^{(i)} = a(u_j^{(i-1)})$ and $u_j^{(i+1)} = c \circ a(u_j^{(i-1)})$. Suppose there exists a family of scalars $\{\alpha_j\}_{j\in J}$, not all zero, such that $\sum_{j\in J} \alpha_j b(u_j^{(i-1)}) = 0$. By commutativity, this implies that $\sum_{j\in J} \alpha_j d \circ b(u_j^{(i-1)}) = \sum_{j\in J} \alpha_j c \circ a(u_j^{(i-1)}) = \sum_{j\in J} \alpha_j u_j^{(i+1)} = 0$, which is in contradiction with the fact that the family $\{u_j^{(i+1)}\}_{j\in J}$ is free.

We study now the evolution of interval summands $\mathbb{I}[b;d]$, with $b \in \{i, i+1\}$ and $d \in \{i-1, i\}$, when passing a transposition diamond. For all cases

of the theorem, we exhibit representative sequences that, together with the ones defined above, represent an interval decomposition for $\mathbb{W}$.

Case 1. Recall that, by commutativity of the diamond, b and d must be both surjective of nullity 1. We first describe $\ker b$. As $c \circ a(v) = 0$, the element $(a(v), 0)$ belongs to $\ker D_2$ and hence belongs to $\operatorname{im} D_1$ by exactness. The preimage $a^{-1}(\{a(v)\})$ is $v + \langle u \rangle$, thus there exists $\alpha \in \Bbbk$ such that $\ker b = \langle v + \alpha u \rangle$ (b has nullity 1). Note that this implies that $b(u) \neq 0$.

(i) If $\alpha = 0$, we consider the representative sequences $(\dots u, b(u), 0 \dots)$ and $(\dots v, 0, 0 \dots)$.

(ii) If $\alpha \neq 0$, we consider the representative sequences $(\dots v, 0, 0 \dots) * \alpha \cdot (\dots u, b(u), 0 \dots) = (\dots v + \alpha u, 0, 0 \dots)$ (with birth $\max_{\leq_{\mathbf{b}}}\{b, b'\}$) and $(\dots w, b(w), 0 \dots)$ (with birth $\min_{\leq_{\mathbf{b}}}\{b, b'\}$), where $w \in V_{i-1}$ is taken to be u if $b \leq_{\mathbf{b}} b'$ and v otherwise. Note that the $*$ is well-defined.

Let $\{b(u_j^{(i-1)})\}_{j \in J}$ be the family of i^{th} elements of all representative sequences of $\mathbb{W}$ traversing index i (as defined above). In case (i), $\{b(u_j^{(i-1)})\}_{j \in J} \cup \{b(u)\}$ is a basis for W_i. Indeed, the family is free. Suppose otherwise: there exist scalars α, α_j such that $\alpha b(u) + \sum_{j \in J} \alpha_j b(u_j^{(i-1)}) = 0$, i.e. $\alpha u + \sum_{j \in J} \alpha_j u_j^{(i-1)} = \beta v$, as $\ker b = \langle v \rangle$. This a contradiction with the fact that $\{u, v\} \cup \{u_j^{(i-1)}\}_{j \in J}$ is free. Additionally, $\dim W_i = 1 + |J|$.

In case (ii), we prove similarly that $\{b(w)\} \cup \{b(u_j^{(i-1)})\}_{j \in J}$ is a basis for W_i. We conclude using proposition 10.2.

Case 2. Recall that, by commutativity of the diamond, b and d must be both injective of nullity 1. Let $(\dots 0, v, c(v), \dots)$ and $(\dots 0, 0, u, \dots)$, $v \in V_i$ and $u \in V_{i+1}$, be representative sequences for the interval submodules of $\mathbb{V}$ isomorphic to $\mathbb{I}[i; d]$ and $\mathbb{I}[i+1; d']$ respectively. First, we prove that $c(v) \notin \operatorname{im} d$. Suppose otherwise: there exists $w \in W_i$ such that $d(w) = c(v)$. But then (v, w) belongs to $\ker D_2$ and hence to $\operatorname{im} D_1$ by exactness. This is in contradiction with the fact $v \notin \operatorname{im} a$.

Because d has corank 1, there exists $\alpha \in \Bbbk$ such that $u + \alpha c(v) \in \operatorname{im} d$. Note that for any α, $u + \alpha c(v) \notin \operatorname{im} d \circ b$. Suppose otherwise: $u + \alpha c(v)$ belongs to $\operatorname{im} c \circ a$ which implies $u + \alpha c(v) \in \operatorname{im} c$ and finally $u \in \operatorname{im} c$, a contradiction. We distinguish the two cases:

(i) If $\alpha = 0$ then there exists $w \in W_i \setminus \operatorname{im} b$ such that $d(w) = u$. We consider the representative sequences $(\dots 0, w, u \dots)$ and $(\dots 0, 0, c(v) \dots)$.

(ii) If $\alpha \neq 0$ then $u \notin \operatorname{im} d$ and there exists $w \in W_i \setminus \operatorname{im} b$ such that $d(w) = u + \alpha c(v)$. We consider the representative sequences $(\dots 0, w, u + \alpha c(v) \dots)$ (with death $\max_{\leq_{\mathbf{d}}}\{d, d'\}$) and $(\dots 0, 0, x, \dots)$ (with death $\min_{\leq_{\mathbf{d}}}\{d, d'\}$), where $x \in V_{i+1}$ is taken to be u if $d' \leq_{\mathbf{d}} d$ and $c(v)$ otherwise.

In both cases (i) and (ii) we can verify that the family containing w and $\{b(u_j^{(i-1)})\}_{j\in J}$ is free for otherwise the family $\{u, v\} \cup \{d \circ b(u_j^{(i-1)}) = u_j^{(i+1)}\}_{j\in J}$ would not be free in V_{i+1}. We conclude using proposition 10.2.
Case 3. We prove that d is injective (of corank 1). Suppose otherwise: there exists $w \neq 0$ in W_i such that $d(w) = 0$. Hence, $(0, w)$ belongs to $\ker D_2$ and belongs to im D_1 by exactness. This implies the existence of $x \neq 0$ in V_{i-1} such that $a(x) = 0$, a contradiction with a injective. To preserve dimensions, b is thus surjective of nullity 1.

We now prove that there is no summand $\mathbb{I}[i; i]$ in $\mathbb{V}$. Suppose otherwise: there exists $x \in V_i \setminus$ im a and $c(x)$. Hence, $(x, 0) \in \ker D_2 =$ im D_1 while x does not belong to im a, a contradiction.

Consequently, let $(0 \dots 0, v, c(v) \dots)$ and $(\dots, u, a(u), 0 \dots)$, $u \in V_{i-1}$ and $v \in V_i$, be representative sequences for the interval submodules of $\mathbb{V}$ isomorphic to $\mathbb{I}[i; d]$ and $\mathbb{I}[b; i]$ respectively. We prove that $c(v) \notin$ im d. Suppose otherwise: there exists $w \in W_i$ such that $d(w) = c(v)$. We use, as previously, the exactness to show that this would imply that v has a preimage through a.

Finally, we prove that $b(u) = 0$. Indeed, the element $(0, b(u))$ belongs to $\ker D_2$ because $c \circ a(u) = d \circ b(u) = 0$ and hence belongs to im D_1 by exactness. Because a is injective, $a^{-1}(0) = 0$, which implies that $b(u)$ must be 0.

In conclusion, we consider the representative sequences $(\dots 0, 0, c(v) \dots)$ and $(\dots, u, 0, 0 \dots)$ for interval submodules of $\mathbb{W}$ that are isomorphic to $\mathbb{I}[i + 1; d]$ and $\mathbb{I}[b; i - 1]$ respectively. We conclude using proposition 10.2.
Case 4. Case 4. is deduced by symmetry from case 3.

12.2 Proof of the Surjective Diamond Principle.

Consider the diagram:

$$\begin{array}{ll} \mathbb{W} := & \\ & V_1 \longleftrightarrow \cdots \longleftrightarrow V \overset{f}{\nearrow} W \overset{f}{\nwarrow} V \longleftrightarrow \cdots \longleftrightarrow V_n \\ \mathbb{V} := & \end{array} \tag{12.1}$$

where the diamond is located at index i in the module and where f is surjective of nullity 1. Note that the arrows at index i in $\mathbb{V}$ are reverted compared to diagram 11.3. Because the morphisms on these arrows are the identity, it does not change the decomposition of $\mathbb{V}$. Suppose $\mathbb{V}$ is isomorphic to:

$$\mathbb{V} \cong \bigoplus_{j\in J} \mathbb{I}[b_j; d_j].$$

All along the proof, we fix a family of representative sequences

$$\{\, u_j^{(b_j)} \nleftrightarrow \cdots \nleftrightarrow u_j^{(d_j)} \,\}_{j\in J}$$

that represents the interval decomposition of $\mathbb{V}$. From these representative sequences, we construct explicit representative sequences for the interval submodules in the direct sum decomposition of $\mathbb{W}$. The construction is ad hoc as it depends on the chosen representative sequences of $\mathbb{V}$. However, it provides an interval decomposition of $\mathbb{W}$ that is canonical due to theorem 10.1.

Contributing Intervals. Let $\xi \neq 0$ be a vector in $\ker f$. Up to a reordering of the indices in J, write ξ as:

$$\xi = \alpha_1 u_1^{(i)} + \cdots + \alpha_p u_p^{(i)},$$

with $\alpha_j \neq 0$ for every $1 \leq j \leq p$ and $d_1 \leq_{\mathbf{d}} \cdots \leq_{\mathbf{d}} d_p$. We say that such interval $\mathbb{I}[b_j; d_j]$ of the decomposition, with $1 \leq j \leq p$, *contributes* to the kernel of f.

Lemma 12.1. *Every interval $\mathbb{I}[b; d]$ of the decomposition of $\mathbb{V}$ that does not contribute to $\ker f$ is matched with an interval $\mathbb{I}[b; d]$ in the decomposition of $\mathbb{W}$.*

Proof. Let $\mathbb{V} \cong \bigoplus_{j=1}^{p} \mathbb{I}[b_j; d_j] \oplus \mathbb{U}$, where $\mathbb{U}$ contains all intervals not contributing to $\ker f$. Consider the morphism $\phi : \mathbb{V} \to \mathbb{W}$ defined as:

$$\begin{array}{ccccccccccccc}
\mathbb{V}: & & V_1 & \leftrightarrow & \cdots & \leftrightarrow & V & \xrightarrow{\mathbb{1}} & V & \xleftarrow{\mathbb{1}} & V & \leftrightarrow & \cdots & \leftrightarrow & V_n \\
{\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\mathbb{1}} & & & & \downarrow{\scriptstyle\mathbb{1}} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle\mathbb{1}} & & & & \downarrow{\scriptstyle\mathbb{1}} \\
\mathbb{W}: & & V_1 & \leftrightarrow & \cdots & \leftrightarrow & V & \xrightarrow{f} & W & \xleftarrow{f} & V & \leftrightarrow & \cdots & \leftrightarrow & V_n
\end{array}$$

The first isomorphism theorem implies that:

$$\mathbb{V}/\ker\phi \cong \operatorname{im}\,\phi = \mathbb{W}$$

by surjectivity of ϕ. We conclude that:

$$\mathbb{W} \cong \left(\bigoplus_{j=1}^{p} \mathbb{I}[b_j; d_j] \right) / \ker\phi \oplus \mathbb{U}.$$

□

Consequently, we assume in the following that all the intervals in the decomposition of $\mathbb{V}$ contain index i and contribute to $\ker f$. Specifically, the decomposition of $\mathbb{V}$ is $\mathbb{V} \cong \bigoplus_{j=1}^{p} \mathbb{I}[b_j; d_j]$ and it is represented by

$\{\ u_j^{(b_j)} \leftrightarrows \cdots \leftrightarrows u_j^{(d_j)}\ \}_{j=1,\ldots,p}$, with $b_j < i < d_j$ for all j and $p = \dim V$.

Formal Sums of Sequences. We consider the restrictions $\mathbb{V}[1; i-1]$ and $\mathbb{V}[i+1; n]$ of the representation $\mathbb{V}$. By mean of the restriction principle 10.2, their interval decompositions are:

$$\mathbb{V}[1; i-1] \cong \bigoplus_{j=1}^{p} \mathbb{I}[b_j; i-1] \text{ and } \mathbb{V}[i+1; d_j] \cong \bigoplus_{j=1}^{p} \mathbb{I}[i+1; d_j]$$

and are represented by

$$\{\ u_j^{(b_j)} \leftrightarrows \cdots \leftrightarrows u_j^{(i-1)}\ \}_{j=1,\ldots,p} \text{ and } \{\ u_j^{(i+1)} \leftrightarrows \cdots \leftrightarrows u_j^{(d_j)}\ \}_{j=1,\ldots,p}$$

respectively. For short, we denote by $\mathbf{x}_j$ the sequence $u_j^{(b_j)} \leftrightarrows \cdots \leftrightarrows u_j^{(i-1)}$ and by $\mathbf{y}_j$ the sequence $u_j^{(i+1)} \leftrightarrows \cdots \leftrightarrows u_j^{(d_j)}$. These sequences also represent interval decompositions for $\mathbb{W}[1; i-1]$ and $\mathbb{W}[i+1; n]$ as $\mathbb{V}$ and $\mathbb{W}$ differ only by a vector space at index i. We call a representative sequence in $\mathbb{V}[1; i-1]$ whose death is $i-1$ a *right sequence*, and we call a representative sequence in $\mathbb{V}[i+1; n]$ whose birth is $i+1$ a *left sequence*.

Let $\mathcal{U}$ be the set of all *formal sums* of $\{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$ with coefficients in $\Bbbk$, i.e.

$$\mathcal{U} = \{\ \mathbf{x}\ :\ \mathbf{x} = \gamma_1\mathbf{x}_1 + \ldots + \gamma_p\mathbf{x}_p,\ \ \gamma_j \in \Bbbk\ \},$$

and let $\mathcal{V}$ be the set of all formal sums of $\{\mathbf{y}_1, \ldots, \mathbf{y}_p\}$ with coefficients in $\Bbbk$, i.e.

$$\mathcal{V} = \{\ \mathbf{y}\ :\ \mathbf{y} = \gamma_1\mathbf{y}_1 + \ldots + \gamma_p\mathbf{y}_p,\ \ \gamma_j \in \Bbbk\ \}.$$

Naturally, $\mathcal{U}$ and $\mathcal{V}$ are $\Bbbk$-vector spaces. Recall that $\leq_{\mathbf{b}}$ and $\leq_{\mathbf{d}}$ are total order relations defined on the indices $\{1, \ldots, n\}$ of the bottom quiver in diagram 12.1. Recall also that "$\cdot$" is a scalar multiplication for representative sequences and $*$ is a binary operator.

Proposition 12.1. *Every formal sum $\gamma_1\mathbf{x}_1 + \ldots + \gamma_p\mathbf{x}_p \neq 0$ of $\mathcal{U}$ defines a right sequence:*

$$\gamma_1 u_1^{(b)} + \ldots + \gamma_p u_p^{(b)} \leftrightarrows \cdots \leftrightarrows \gamma_1 u_1^{(i-1)} + \ldots + \gamma_p u_p^{(i-1)}$$

where the birth b is equal to $\max_{\leq_{\mathbf{b}}}\{b_j : \gamma_j \neq 0\}_{j=1,\dots,p}$. *Similarly, every formal sum* $\gamma_1\mathbf{y}_1 + \ldots + \gamma_p\mathbf{y}_p \neq 0$ *of* $\mathcal{V}$ *defines a left sequence:*

$$\gamma_1 v_1^{(i+1)} + \ldots + \gamma_p v_p^{(i+1)} \nleftrightarrow \cdots \nleftrightarrow \gamma_1 v_1^{(d)} + \ldots + \gamma_p v_p^{(d)}$$

where the death d is equal to $\max_{\leq_{\mathbf{d}}}\{d_j : \gamma_j \neq 0\}_{j=1,\dots,p}$.

Proof. We prove the result for $\mathcal{U}$ only, the case of $\mathcal{V}$ being symmetric. Let $i_1, \ldots, i_k$ be the indices such that $\gamma_{i_j} \neq 0$. We prove the proposition by induction on j for the formal sum $\gamma_{i_1}\mathbf{x}_{i_1} + \ldots + \gamma_{i_j}\mathbf{x}_{i_j}$. For $j = 1$, $\gamma_{i_1}\mathbf{x}_{i_1}$ defines the right sequence $\gamma_{i_1} \cdot \mathbf{x}_{i_1}$, where " $\cdot$ " is the scalar multiplication of definition 10.2. Suppose the property true up to j: $\gamma_{i_1}\mathbf{x}_{i_1} + \ldots + \gamma_{i_j}\mathbf{x}_{i_j}$ defines a right sequence $\mathbf{x} =$

$$\gamma_{i_1} u_{i_1}^{(b)} + \ldots + \gamma_{i_j} u_{i_j}^{(b)} \nleftrightarrow \gamma_{i_1} u_{i_1}^{(i-1)} + \ldots + \gamma_{i_j} u_{i_j}^{(i-1)}$$

We prove that $\mathbf{x}$ and $\gamma_{i_{j+1}} \cdot \mathbf{x}_{i_j}$ satisfy conditions *(a)*, *(b)* and *(c)* of definition 10.2 and show that $\mathbf{x} * (\gamma_{i_{j+1}} \cdot \mathbf{x}_{i_j})$ is the right sequence associated to $\gamma_{i_1}\mathbf{x}_{i_1} + \ldots + \gamma_{i_{j+1}}\mathbf{x}_{i_{j+1}}$. Conditions *(a)* and *(c)* are directly satisfied considering the two sequences have same death $i-1$, and $i-1$ is the largest index in the restricted module $\mathbb{V}[1; i-1]$. We prove condition *(b)* by contradiction: suppose there exists and index $k \in [b; i-1] \cap [b_{i_{j+1}}; i-1]$ such that, for some $\alpha, \beta \in \Bbbk$, not both 0, we have:

$$\alpha(\gamma_{i_1} u_{i_1}^{(k)} + \ldots + \gamma_{i_j} u_{i_j}^{(k)}) + \beta\gamma_{i_{j+1}} u_{i_{j+1}}^{(k)} = 0$$

By hypothesis on k, not all $u_i^{(k)}$ in the sum are 0 and hence we have a contradiction with the fact that the family $\{u_j^{(k)} : u_j^{(k)} \neq 0\}$ is free in V_k.

Consequently, $\mathbf{x} * (\gamma_{i_{j+1}} \cdot \mathbf{x}_{i_j})$ is a well-defined right sequence. Its birth is $\max_{\leq_{\mathbf{b}}}\{b_{i_\ell}\}_{\ell=1,\dots,j+1}$ and its death is $i-1$.

Finally, using the commutativity and associativity of $*$ (lemma 10.2), we conclude that the representative sequence constructed does not depend on the order in which the sequences are added.

□

By a small abuse of notation, we use the same notation $\mathbf{x}$ for the formal sum and for the representative sequence it defines.

Recall that $\xi = \alpha_1 u_1^{(i)} + \cdots + \alpha_p u_p^{(i)}$ generates the kernel of f. We define the right and left sequences $\xi^{\mathcal{U}}$ and $\xi^{\mathcal{V}}$ to be equal to $\alpha_1\mathbf{x}_1 + \cdots + \alpha_p\mathbf{x}_p$ and $\alpha_1\mathbf{y}_1 + \cdots + \alpha_p\mathbf{y}_p$ respectively.

The Join Operator. Let $u^{(b)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow u^{(i-1)}$ and $v^{(i+1)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow v^{(d)}$ be respectively a right sequence in $\mathbb{V}[1; i-1]$ and a left sequence in $\mathbb{V}[i+1; n]$. Recall that f is the surjective map in the diamond. If the sequences satisfy $f(u^{(i-1)}) = f(v^{(i+1)})$ (we denote this element by w), we define the *join* of these right and left sequences by the representative sequence of $\mathbb{W}$ equal to:

$$\begin{gathered}(u^{(b)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow u^{(i-1)}) \bowtie (v^{(i+1)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow v^{(d)}) := \\ u^{(b)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow u^{(i-1)} \rightarrow w \leftarrow v^{(i+1)} \leftarrow\!\!\!\rightarrow \cdots \leftarrow\!\!\!\rightarrow v^{(d)}\end{gathered}$$

The following lemma is a direct consequence of the linearity of f:

Lemma 12.2. *Let $\mathbf{x}, \mathbf{x}' \in \mathcal{U}$, $\mathbf{y}, \mathbf{y}' \in \mathcal{V}$ such that $\mathbf{x} \bowtie \mathbf{y}$ and $\mathbf{x}' \bowtie \mathbf{y}'$ are well-defined. For any $\alpha, \beta \in \Bbbk$, the join $(\alpha\mathbf{x} + \beta\mathbf{x}') \bowtie (\alpha\mathbf{y} + \beta\mathbf{y}')$ is well-defined.*

Greedy Pairing of Births and Deaths. Up to a reordering of the indices $\{1, \ldots, p\}$, suppose that the deaths satisfy $d_1 \leq_{\mathbf{d}} \ldots \leq_{\mathbf{d}} d_p$. Define π to be the permutation of $\{1, \cdots, p\}$ satisfying $b_{\pi(1)} \leq_{\mathbf{b}} \ldots \leq_{\mathbf{b}} b_{\pi(p)}$, breaking ties arbitrarily. In particular d_p is the maximum w.r.t. $\leq_{\mathbf{d}}$ among all the death indices $\{d_1, \cdots, d_p\}$ and $b_{\pi(p)}$ is the maximum w.r.t. $\leq_{\mathbf{b}}$ among all the birth indices $\{b_1, \cdots, b_p\}$.

We construct new bases $\{\overline{\mathbf{x}}_0, \ldots, \overline{\mathbf{x}}_{p-1}\}$ for $\mathcal{U}$ and $\{\overline{\mathbf{y}}_0, \cdots, \overline{\mathbf{y}}_{p-1}\}$ for $\mathcal{V}$ such that:

1. the p right sequences $\mathbf{x}_j$, $0 \leq j \leq p-1$, have births exactly the set $\{b_1, \ldots, b_p\}$,
2. the p left sequences $\mathbf{y}_j$, $0 \leq j \leq p-1$, have deaths exactly the set $\{d_1, \ldots, d_p\}$,
3. for every j such that $1 \leq j \leq p-1$, the join between $\mathbf{x}_j$ and $\mathbf{y}_j$ is well-defined,
4. the joins $\mathbf{x}_0 \bowtie 0$ and $0 \bowtie \mathbf{y}_0$ are well-defined, where 0 refers to the sequence where all elements are 0. $\mathbf{x}_0 \bowtie 0$ has death $i-1$ and $0 \bowtie \mathbf{y}_0$ has birth $i+1$.

We pick $\overline{\mathbf{x}}_0 = \xi^{\mathcal{U}}$ and $\overline{\mathbf{y}}_0 = \xi^{\mathcal{V}}$ as first elements for the new bases. They satisfy condition (4.) above. Note that $\{\overline{\mathbf{x}}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{p-1}\}$ and $\{\overline{\mathbf{y}}_0, \mathbf{y}_1, \cdots, \mathbf{y}_{p-1}\}$ are bases for $\mathcal{U}$ and $\mathcal{V}$ respectively. Note also that, by definition of the $\mathbf{x}_j$ and the $\mathbf{y}_j$, conditions (2.) and (3.) are satisfied, but not condition (1.)

Algorithm 12.1: Reduction to column echelon form of $\mathbf{X}$ and $\mathbf{Y}$.

Data: Matrices $\mathbf{X}$ and $\mathbf{Y}$
for $j = 1$ *to* $p-1$ **do**
  while *there exists* $j_0 < j$ *with* $\mathrm{low}_{\mathbf{X}}(j_0) = \mathrm{low}_{\mathbf{X}}(j)$ **do**
    $\overline{\mathbf{x}}_j \leftarrow \overline{\mathbf{x}}_j + \left(\frac{-\gamma_{\mathrm{low}_{\mathbf{X}}(j)}}{\gamma_{\mathrm{low}_{\mathbf{X}}(j_0)}}\ \overline{\mathbf{x}}_{j_0}\right)$;
    if $j_0 \neq 0$ **then** $\overline{\mathbf{y}}_j \leftarrow \overline{\mathbf{y}}_j + \left(\frac{-\gamma_{\mathrm{low}_{\mathbf{X}}(j)}}{\gamma_{\mathrm{low}_{\mathbf{X}}(j_0)}}\ \overline{\mathbf{y}}_{j_0}\right)$;
  end
end

as $\xi^{\mathcal{U}}$ does not have, in general, birth b_p. The construction is algorithmic[1] and consists of a simultaneous basis change in the vector spaces $\mathcal{U}$ and $\mathcal{V}$. Let $\mathbf{X}$ be the $(p \times p)$ matrix, with $\Bbbk$ coefficients, representing the basis $\{\xi^{\mathcal{U}}, \mathbf{x}_1, \cdots, \mathbf{x}_{p-1}\}$ of $\mathcal{U}$ into the basis $\{\mathbf{x}_{\pi(1)}, \cdots, \mathbf{x}_{\pi(p)}\}$, and let $\mathbf{Y}$ be the $(p \times p)$ matrix, with $\Bbbk$ coefficients, representing the basis $\{\xi^{\mathcal{V}}, \mathbf{y}_1, \cdots, \mathbf{y}_{p-1}\}$ of $\mathcal{V}$ into the basis $\{\mathbf{y}_1, \cdots, \mathbf{y}_p\}$, as depicted in figure 12.1. Note that columns are labeled from 0 to $p-1$ and rows are labeled from 1 to p, in order to match with the indices of the bases elements.

For a matrix $\mathbf{M}$, we define $\mathrm{low}_{\mathbf{M}}(j)$ to be the row index of the lowest non-zero coefficient of the j^{th} column, for $0 \leq j \leq p-1$. Let $\gamma_{\mathrm{low}(j)} \in \Bbbk$ be the value of this lowest coefficient in matrix $\mathbf{X}$. We denote the reduced columns of $\mathbf{X}$ and $\mathbf{Y}$ by $\overline{\mathbf{x}}_i$ and $\overline{\mathbf{y}}_i$ respectively. The algorithm is a reduction to column echelon form of the matrix $\mathbf{X}$, meaning that at the end of the algorithm, any row index j admits a unique column k such that $j = \mathrm{low}_{\mathbf{X}}(k)$. The process is presented in algorithm 12.1. The reduction is done by mean of elementary column operations, that are reproduced almost identically in $\mathbf{Y}$ (**if** condition). In the algorithm, for any $0 \leq j \leq p-1$, we denote by $\gamma_{\mathrm{low}_{\mathbf{X}}(j)}$ the value of the lowest non-zero coefficient of column j in matrix $\mathbf{X}$.

Because the algorithm consist in a change of basis in vector spaces $\mathcal{U}$ and $\mathcal{V}$, the algorithm is valid and, at the end of the reduction, $\mathbf{X}$ is in column echelon form. The **if** condition of the reduction ensures that the function $\mathrm{low}_{\mathbf{Y}}$ does not change and $\mathbf{Y}$ is in column echelon form as well. Because the rows are sorted by increasing birth values w.r.t. $\leq_{\mathbf{b}}$ in $\mathbf{X}$, the birth of $\overline{\mathbf{x}}_j$ is the birth $b_{\pi(k)}$ such that $k = \mathrm{low}_{\mathbf{X}}(j)$. Consequently, as the function

[1]Note that these algorithm are theoretical and are not used in the zigzag persistence algorithm presented in section 11.2. They are meant to prove the surjective diamond principle, which offers a simpler computational rule for updating the interval decomposition.

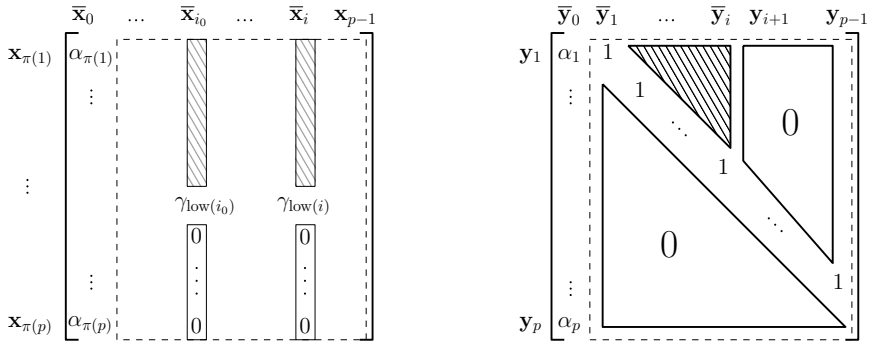


Figure 12.1: Matrices $\mathbf{X}$ and $\mathbf{Y}$ during the j^{th} iteration of algorithm 12.1.

$\mathrm{low}_{\mathbf{X}}$ is bijective ($\mathbf{X}$ is in column echelon form and is full-rank), condition (1.) is satisfied. With a similar argument on $\mathbf{Y}$, condition (2.) is satisfied. Finally, condition (3.) is satisfied because, for any $j \geq 1$ and all along the procedure, the vector at index $i-1$ in the right sequence $\overline{\mathbf{x}}_j$ and the vector at index $i+1$ in the left sequence $\overline{\mathbf{y}}_j$ are either identical, or differ by $\gamma\xi$. They consequently have the same image by f.

Proposition 12.2. *The family* $\{\overline{\mathbf{x}}_0 \bowtie 0, 0 \bowtie \overline{\mathbf{y}}_0, \overline{\mathbf{x}}_1 \bowtie \overline{\mathbf{y}}_1, \cdots, \overline{\mathbf{x}}_{p-1} \bowtie \overline{\mathbf{y}}_{p-1}\}$ *of representative sequences of* $\mathbb{W}$ *represents an interval decomposition of* $\mathbb{W}$.

Proof. For any index j, $0 \leq j \leq p-1$, denote $\mathbf{x}_j$ and $\overline{\mathbf{x}}_j$ by, respectively, $u^{(b_j)} \twoheadleftarrow \cdots \twoheadleftarrow u^{(i-1)}$ and $\overline{u}_j^{(b_{\ell_j})} \twoheadleftarrow \cdots \twoheadleftarrow \overline{u}_j^{(i-1)}$. We prove that, for any index $k \leq i-1$, the family $\{\overline{u}_j^{(k)} : \overline{u}_j^{(k)} \neq 0\}_{j=0,\ldots,p-1}$ is a basis for V_k. By virtue of proposition 12.1, the family $\{\overline{u}_j^{(k)} : \overline{u}_j^{(k)} \neq 0\}_{j=0,\ldots,p-1}$ is a subset of the family $\{\mathbf{X}u_j^{(k)} : u_j^{(k)} \neq 0\}_{j=0,\ldots,p-1}$, where $\mathbf{X}$ is in column echelon form and is invertible, and the set of non-zero $u_j^{(k)}$ is free. Consequently, for any index $k \leq i-1$, the family of non-zero $u_j^{(k)}$ is free. Because, the births of the representative sequences $\{\overline{\mathbf{x}}_0 \bowtie 0, 0 \bowtie \overline{\mathbf{y}}_0, \overline{\mathbf{x}}_1 \bowtie \overline{\mathbf{y}}_1, \cdots, \overline{\mathbf{x}}_{p-1} \bowtie \overline{\mathbf{y}}_{p-1}\}$ are exactly the set $\{b_1, \ldots, b_p\}$, the set $\{\overline{u}_j^{(k)} : \overline{u}_j^{(k)} \neq 0\}_{j=0,\ldots,p-1}$ contains $\dim V_k$ elements and is a basis of V_k. Denoting $\overline{\mathbf{y}}_j = \overline{v}_j^{(i+1)} \twoheadleftarrow \cdots \twoheadleftarrow \overline{v}_j^{(d_j)}$, we prove similarly that the family of non-zero $\overline{v}_j^{(k)}$ is a basis for V_k, for any index $k \geq i+1$.

Finally, at index $k = i$, the non zero vectors $\overline{u}_j^{(i)}$ form a basis because the decomposition is aligned with the kernel of f. We conclude using proposition 10.2.

□

Finally, we study a bit closer the structure of the matrix $\mathbf{X}$ during the reduction, in order to deduce the simpler pairing rule described in algorithm 11.1 of the surjective diamond principle 11.3.

Lemma 12.3. *The reduction algorithm 12.1 satisfies the following properties.*

(i) During its reduction, any column col *of* $\mathbf{X}$ *is either equal to* $[0, \dots, 0, 1, 0, \dots 0]^T$ *or to* $\gamma \times [\alpha_{\pi(1)}, \dots, \alpha_{\pi(\mathrm{low}_{\mathbf{X}}(\mathsf{col}))}, 0, \dots, 0]^T$ *for some* $\gamma \neq 0$.

(ii) The reduction of $\mathbf{X}$ *performs* $O(p)$ *column operations in* $\mathbf{X}$.

(iii) The pairing procedure in algorithm 11.1 of the surjective diamond principle 11.3 is correct.

Proof. **(i)** We prove the result by induction on the column index j. For $j = 0$, $\overline{\mathbf{x}}_0 = [\alpha_{\pi(1)}, \dots, \alpha_{\pi(p-1)}]^T$ and the property is satisfied. Suppose all columns $\overline{\mathbf{x}}_k$, for $k \leq j$, satisfy the property. We reduce column $\mathbf{x}_{j+1}$, originally equal to $[0, \dots, 0, 1, 0, \dots, 0]$. If there is no column $\overline{\mathbf{x}}_k$, with $k < j + 1$, such that $\mathrm{low}_{\mathbf{X}}(k) = \mathrm{low}_{\mathbf{X}}(j + 1)$, then the column $\mathbf{x}_{j+1}$ is reduced and satisfies the property. Otherwise, let $\overline{\mathbf{x}}_k$ be such that $k \leq j$ and $\mathrm{low}_{\mathbf{X}}(k) = \mathrm{low}_{\mathbf{X}}(j + 1)$. By induction, $\overline{\mathbf{x}}_k$ satisfies the property. Because the matrix has full-rank, $\overline{\mathbf{x}}_k$ must be equal to $\gamma \times [\alpha_{\pi(1)}, \dots, \alpha_{\pi(\mathrm{low}_{\mathbf{X}}(j+1))}, 0, \dots, 0]^T$ for some $\gamma \neq 0$.

We reduce $\mathbf{x}_{j+1}$ using $\overline{\mathbf{x}}_k$ as in the **while** loop. Denote $\mathrm{low}_{\mathbf{X}}(j + 1)$ by ℓ. We get $\overline{\mathbf{x}}_{j+1} = \gamma' \times [\alpha_{\pi(1)}, \dots, \alpha_{\pi(\ell-1)}, 0, \dots, 0]^T$ for some $\gamma' \neq 0$. By induction, and because $\mathbf{X}$ has full-rank, the remainder of the reduction of $\overline{\mathbf{x}}_{j+1}$ involves only columns of the form $[0, \dots, 0, 1, 0, \dots, 0]^T$. Consequently, the reduced column $\overline{\mathbf{x}}_{j+1}$ satisfies the property.

(ii) The property follows by noticing that, during the whole reduction, a column $\mathbf{x}_k$ is picked at most once in the **while** loop condition for reducing another column (i.e. $j_0 \leftarrow k$).

(iii) For each column operation $\overline{\mathbf{x}}_j \leftarrow \overline{\mathbf{x}}_j + \gamma \overline{\mathbf{x}}_{j_0}$, the index $\mathrm{low}_{\mathbf{X}}(j)$ decreases by exactly 1. Recall that the birth of the representative sequence defined by $\overline{\mathbf{x}}_j$ is $b_{\pi(\mathrm{low}_{\mathbf{X}}(j))}$. Recall also that, by definition of the permutation π, the rows are ordered by increasing birth w.r.t. $\leq_{\mathbf{b}}$, i.e. $b_{\pi(1)} \leq_{\mathbf{b}} \dots \leq_{\mathbf{b}} b_{\pi(p)}$. We prove by induction on j that algorithm 11.1 of the surjective diamond principle 11.3 is correct. More precisely, we prove that every reduced column $\overline{\mathbf{x}}_j$ equal to $[0, \dots, 0, 1, 0, \dots, 0]^T$ has birth $b_{\ell_j} = b_j$ and every reduced column $\overline{\mathbf{x}}_j$ equal to $\gamma \times [\alpha_{\pi(1)}, \dots, \alpha_{\pi(\mathrm{low}_{\mathbf{X}}(j))}, 0, \dots, 0]^T$ has birth b_{ℓ_j} equal to the maximal birth index, w.r.t. $\leq_{\mathbf{b}}$, available during its reduction. For

$j = 0$, $\overline{\mathbf{x}}_0$ has birth $b_{\pi(p)}$ which is the maximum w.r.t. $\leq_{\mathbf{b}}$. Suppose the property true for all $k \leq j$. Consider column $\overline{\mathbf{x}}_{j+1}$. If its reduced form is $[0, \ldots, 0, 1, 0, \ldots, 0]^T$, then its representative sequence has indeed birth b_{j+1}. Otherwise, there exists a column $\overline{\mathbf{x}}_k = \gamma \times [\alpha_{\pi(1)}, \ldots, \alpha_{\pi(\mathrm{low}_{\mathbf{X}}(j+1))}, 0, \ldots, 0]^T$ for $k \leq j$. Hence, by induction, none of the births larger than b_{j+1} are available. By reducing $\mathrm{low}_{\mathbf{X}}(j+1)$ by exactly 1 for each column operation, $\overline{\mathbf{x}}_{j+1}$ is assigned the largest birth w.r.t. $\leq_{\mathbf{b}}$ that is available. □

This concludes the proof of the surjective diamond principle.

Part V

Computational Library for Topology

This Part is based on the following publications:

- Clément Maria, Jean-Daniel Boissonnat, Marc Glisse and Mariette Yvinec: The Gudhi Library: Simplicial Complexes and Persistent Homology. *International Congress on Mathematical Software* [51]

and the computational library is available at

- Clément Maria: `Gudhi`, Simplicial Complexes and Persistent Homology Packages, `https://project.inria.fr/gudhi/software/` [50]

Chapter 13

Generic Design

We present the main algorithmic and design choices that have been made to represent complexes and compute persistent homology in the `Gudhi` library. The `Gudhi` library (Geometric Understanding in Higher Dimensions) is a generic `C++` library for computational topology. Its goal is to provide robust, efficient, flexible and easy to use implementations of state-of-the-art algorithms and data structures for computational topology. We present the different components of the software, their interaction and the user interface. We justify the algorithmic and design decisions made in `Gudhi` and provide benchmarks for the code.

The challenge is twofold. On the one hand we need to design a generic library in computational topology, in order to adapt to the various configurations of the problem: nature of the complexes (simplicial, cubical, etc) and their representation, nature of the maps between them (inclusions, edge contractions, etc), ordering of the maps (linear, zigzag, etc) and types of algorithm for persistence (homology, cohomology). On the other hand, we need to implement a high-performance library to handle complex practical examples.

In Section 13.1 we describe the challenges when designing a library for computational topology and Section 13.2 presents the design of the `Gudhi` library. In Section 13.3, we discuss the implementation choices and the user interface. Specifically, simplicial complexes are implemented with a simplex tree data structure (Chapter 2. The simplex tree is an efficient and flexible data structure for representing general (filtered) simplicial complexes. The persistent homology of a filtered simplicial complex is computed by means of the persistent cohomology algorithm implemented with a compressed annotation matrix (Chapter 5). The persistent homology package provides the

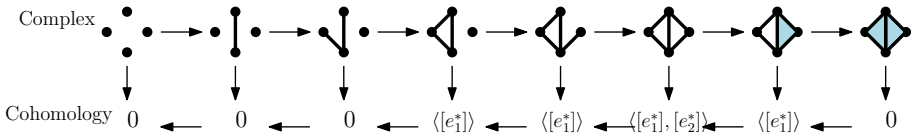


Figure 13.1: Standard computation of persistence, with simplicial complexes connected by left to right inclusions.

computation of persistence with different coefficient fields, including the implementation of the multi-field persistence algorithm (Chapter 8). Finally, in we discuss the future components of the library and their integration in the design.

13.1 Philosophy of the Library

Suppose we are given a set of points and a metric, approximating a topological space whose homology groups we want to compute. We can compute the Rips complex on top of these points using a simplex tree, then compute the persistent homology of the filtered complex with multi-field coefficients and hence get, with garantees, the homology groups (integral Betti numbers and prime divisors of torsion coefficients) of the topological space (see Figure 13.1).

In light of the experiments in Parts I, II and III, the time complexity of such computation would be of order 10^{-8} seconds per simplex for the construction of the simplex tree and 10^{-6} seconds per simplex for the computation of multi-field persistence for the first few hundred prime numbers on a desktop machine. Memorywise, the size of the simplicial complex is the bottleneck of the computation, and we can store a simplicial complex in a simplex tree of up to a few hundred million simplices. Consequently, we need only a few minutes to compute the persistent homology of a simpli-

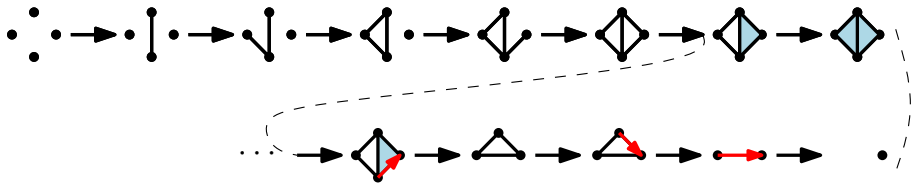


Figure 13.2: Filtration of simplicial complexes connected by general simplicial maps. In particular, the edge contractions allows the computation of the same changes at homology level while reducing the size of the simplicial complex.

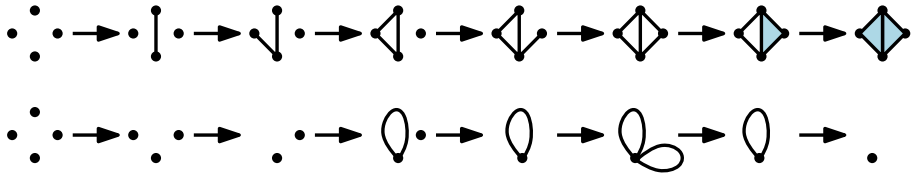


Figure 13.3: Simplification of a filtered simplicial complex. The result is a filtration whose persistent homology is identical to the original one. The complexes are now general cell complexes, but individually contain fewer cells that the original simplicial complexes.

cial complex that fills up the memory entirely: memory is the limitation of the approach, independently of what data structures are used. Indeed, the simplex tree has an optimal size to represent a filtered simplicial complex. Moreover, the theoretical bounds (Theorem 1.4) on the threshold of a Rips complex to obtain a valid topological approximation of a topological space lead to gigantic simplicial complexes in practice.

Designing new approaches to topological inference, that solve the memory issue while preserving good computational timings in practice, is a recent and major challenge for computational topology. Recently, a few methods have been introduced. The first one consists in defining new types of filtrations that give good topological approximations while remaining small. For example, in [66] the author introduce a linear-size (in the number of vertices) Rips complex that gives a provably good persistence diagram, and in [40] the authors show how to compute the persistent homology of the filtration using general simplicial maps (in particular edge contractions, see Figure 13.2).

Another approach is to simplify the filtered simplicial complex. In particular, such an approach as been developed using Morse simplifications [55], which preserve the homology. By simplifying the filtered simplicial complex within ranges of simplices with same filtration value, one obtains a filtered

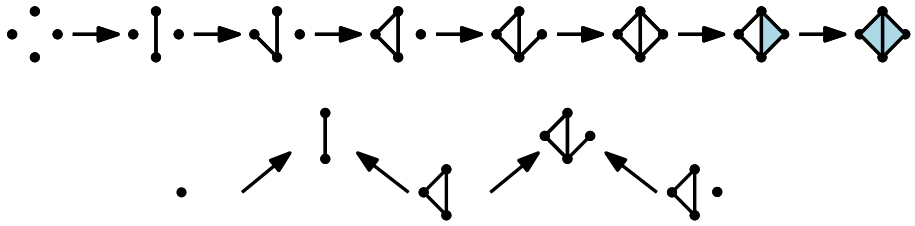


Figure 13.4: Zigzag filtration of a complex.

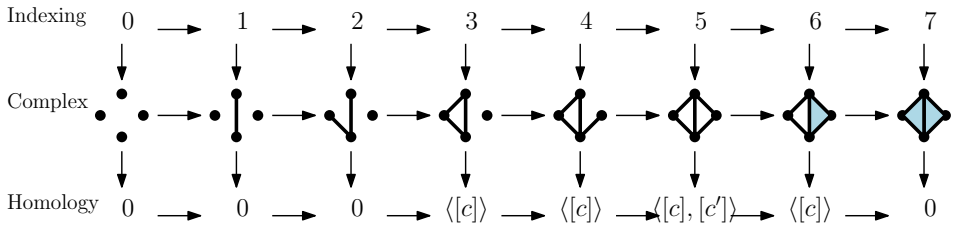


Figure 13.5: Indexing of eight simplicial complexes and corresponding sequence of homology groups in dimension 1.

cell complex with fewer cells (see Figure 13.3).

Finally, zigzag persistence, which has been introduced in Part IV, is a generalization of persistent homology. Zigzag persistent homology not only allows the computation of persistence on smaller complexes but also open new areas for applications. Oudot and Sheehy prove in [61] that the filtration of the oscillating Rips zigzag gives provably good persistence diagram while maintaining small simplicial complexes along the computation (under some reasonable geometric assumptions). We refer to Figure 13.4 for an illustration of a zigzag filtration and to Chapter 14 for a practical example of topological inference and a comparison of standard persistence and zigzag persistence (in particular on the size of the simplicial complexes). Part IV of this dissertion presents an algorithm to compute zigzag persistence more efficiently and with a simpler approach, and falls within the category of these recent works.

The common point of these approaches is that they require generalizations of some aspects of standard persistence, which may be either a general type of simplicial maps in the filtration – as opposed to inclusions –, a general type of cell complexes – as opposed to simplicial complexes – or maps going in both directions – instead of left-to-right only. We describe in the following a flexible design for a generic library for computational topology. In particular, the design adapts to the different improvements on the theory of persistence that are described above. The library is available at [50] and implements all algorithms presented in this dissertation.

Components of the Theory. Figure 13.5 illustrates the three components of the theory of persistence and their connections. The top one is what we call an *indexing scheme* and is a quiver of type A_n, together with a left to right traversal order. From the computational point of view, there are two types of indexing schemes of interest in persistent homology: *lin-*

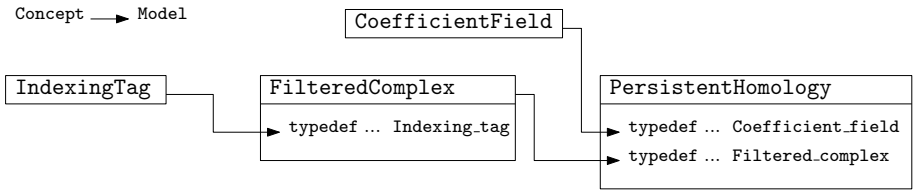


Figure 13.6: Overview of the design of the library.

ear ones $\bullet \longrightarrow \bullet \longrightarrow \cdots \longrightarrow \bullet \longrightarrow \bullet$ in persistent homology (Chapter 4), and *zigzag* ones $\bullet \longrightarrow \bullet \longleftarrow \cdots \longrightarrow \bullet \longleftarrow \bullet$ in zigzag persistent homology (Chapter 10). We describe indexing schemes with ranges and iterators, plus an indication of the orientation – forward or backward – of the arrows.

In the following, we consider the case where the indexing scheme is induced by a filtration. A *filtration* of a simplicial complex is a function $f : \mathbf{K} \to \mathbb{R}$ satisfying $f(\tau) \leq f(\sigma)$ whenever $\tau \subseteq \sigma$. Ordering the simplices by increasing filtration values (breaking ties so as a simplex appears after its subsimplices of same filtration value) provides a linear indexing scheme.

The middle of Figure 13.5 represents the sequence of simplicial complexes connected by simplicial maps, *ie* the filtration. Finally, the bottom represents the persistence module.

Remark 13.1. *The separation into indexing, filtration and persistence module is close in spirit to the categorification of persistent homology. In particular, the vertical arrows in Figure 13.5 represent functors of categories. We refer to [22] for more details on the category theory approach of persistence.*

13.2 Design of the Library

A *concept* is a set of requirements (valid expression, associated types, etc) for a type. If a type satisfies these requirements, it is a *model* of the concept. The general idea under our design is to associate a concept per component presented above: the three components of the theory (indexing, complex and homology) are illustrated in Figure 13.5. Given two components related by a vertical arrow in Figure 13.5, and two models `A` and `B` of their respective associated concepts, we connect `B` with `A` through a template argument `B<A>`.

IndexingTag Concept. In order to describe the indexing scheme, we use a tag `IndexingTag` that is either `linear_indexing_tag` or `zigzag_indexing_tag`, corresponding to the two indexing schemes of interest mentioned above.

The tag is passed as a template argument to a model of the concept `FilteredComplex`, representing filtered cell complexes.

FilteredComplex Concept. We define the concept `FilteredComplex` that describes the requirement for a type to implement a filtered cell complex. We use the vocabulary of simplicial complexes, but the concept is valid for any type of cell complex. The main requirements are the definition of:

1. type `Indexing_tag`, which is a model of the concept `IndexingTag`, describing the nature of the indexing scheme,
2. type `Simplex_handle` to manipulate simplices,
3. method `int dimension(Simplex_handle)` returning the dimension of a simplex,
4. type and method `Boundary_simplex_range boundary_simplex_range(Simplex_handle)` that returns a range giving access to the codimension 1 subsimplices of the input simplex, as well as the coefficients $(-1)^i$ in the definition of the operator ∂. The iterators have value type `Simplex_handle`,
5. type and method `Filtration_simplex_range filtration_simplex_range ()` that returns a range giving access to all the simplices of the complex read in the order assigned by the indexing scheme,
6. type and method `Filtration_value filtration(Simplex_handle)` that returns the value of the filtration on the simplex represented by the handle.

Figure 13.7 illustrates the use of a model of the concept `FilteredComplex`. It sketches the algorithm used for computing persistent homology via the annotation algorithm of Chapter 5.

PersistentHomology Concept. The concept `PersistentHomology` describes the requirement for a type to compute the persistent homology of a filtered complex. The requirement are the definition of:

1. a type `Filtered_complex`, which is a model of `FilteredComplex` and provides the type of complex on which persistence is computed,
2. a type `Coefficient_field`, which is a model of `CoefficientField` and provides the coefficient field on which homology is computed.

```
void compute_persistent_homology( FilteredComplex cpx ) {
    for( Simplex_handle sh : cpx.filtration_simplex_range() ) {
        int dim = cpx.dimension(sh);
        update_cohomology_groups( dim, sh, cpx );
            //inside update_cohomology_groups
            for( Simplex_handle b_sh :
                cpx.boundary_simplex_range(sh) ) {...}
            //out
} } }
```

Figure 13.7: Sample code for the computation of persistence, illustrating the use of a model of concept `FilteredComplex`.

The requirements of the concept `CoefficientField` are essentially the definition of field operations (addition, multiplication, inversion, etc).

We refer to Figure 13.6 for a presentation of the concepts and their connections.

13.3 Implementation

In this section we describe how these concepts are implemented. The code is available at [50].

Simplicial Complex. We use a *simplex tree* (Chapter 2) to represent simplicial complexes. The class `Simplex_tree` is a model of `FilteredComplex` and hence furnishes all requirements of the concept. Moreover, it furnishes algorithms to construct efficiently simplicial complexes, and in particular flag complexes.

Persistent Homology. We use the *compressed annotation matrix* (Chapter 5) to implement the persistent cohomology algorithm [35, 40] for persistence. This leads to the class `Persistent_cohomology`, which is a model of `PersistentHomology`. The class `Persistent_cohomology` allows the computation of the persistence diagram of a filtered complex, using the method `compute_persistent_homology` (see Figure 13.7).

The coefficient fields available as models of `CoefficientField` are `Field_Zp` for $\mathbb{Z}_p$ (for any prime p) and `Multi_field` for the multi-field persistence algorithm – computing persistence simultaneously in various coefficient fields – described in Chapter 8.

Future Components. The library may be extended in various directions that fit naturally in the design. The first direction is to allow zigzag indexing schemes, by the creation of a tag `zigzag_indexing_tag`. In this case, the method `filtration_simplex_range` must indicate the direction of the arrows. The implementation of the zigzag persistence algorithm presented in Chapter 11 is at the prototype stage, and will be integrated within the `Gudhi` library soon.

New implementations and models for `FilteredComplex` will be added. For example, the construction of witness complexes will be added to the class `Simplex_tree`. Additionnaly, new types of complexes (like cubical complexes) and new data structures to represent them may be added to the library: in order to compute their persistent homology, they only need to satisfy the requirements of the concept `FilteredComplex`.

So far, only inclusions have been considered for simplicial maps between simplicial complexes. As explained in [40], any simplicial map may be implemented with a sequence of inclusions and edge contractions. We will consequently add edge contractions as updates in the class `Simplex_tree` and implement the induced updates in the class `Persistent_cohomology` (an algorithm for edge contraction in a simplex tree is described in Chapter 2 and an algorithm for updating an annotation matrix at cohomology level is described in [40]). This way, we will be able to compute persistent homology of simplicial maps. In this case, the range provided by `filtration_simplex_range` must indicate the nature of the map between complexes.

Future works include also the implementation of a class `Field_Q`, model of concept `CoefficientField`, for homology with $\mathbb{Q}$ coefficients. Finally the interface between complexes and persistent homology allows us to implement more persistent homology algorithms.

Ongoing projects include also the fast computation of nearest neighbors in metric spaces, the computation of the bottleneck distance between persistence diagrams and an interface for the `Gudhi` library in the `R` language.

Chapter 14

Computational Topology in Action

14.1 Example of Use of the Library

We study in more details the *Clifford data set* [61], which consists of 2000 points evenly spaced along the line $y = 20x \mod 2\pi$ in the flat torus $(\mathbb{R} \mod 2\pi)^2$, then mapped onto the Clifford torus in $\mathbb{R}^4$ via the embedding $f\colon (u, v) \mapsto (\cos u, \sin u, \cos v, \sin v)$. This data set admits three non-trivial underlying spaces: at small scales, a topological circle (*ie* the helicoidal curve), at larger scales, the torus and at even larger scales the 3-sphere on which the torus is sitting. This is a data set of interest for persistent homology, because of the different levels of reconstruction. We use the `Gudhi` library, described in the previous chapter and using the algorithms and data structures developed in this dissertation, to infer the topology of the Clifford data set.

Figure 14.1 illustrates the user interface for constructing the 4-skeleton of a flag complex from a graph (we use a Rips graph for the experiments), using a simplex tree (Chapter 2), and computing its persistent homology with various coefficient fields, using the compressed annotation matrix (Chapter 5) with modular reconstruction (Chapter 8).

We have been able to store and compute the multi-field persistent homology of a 4-dimensional simplicial complex with up to 491 million simplices before running out of memory. Precisely, we have constructed the 4-skeleton of the Rips complex (with threshold $\rho = 0.9$) on the Clifford data set in 20 seconds using a simplex tree. The multi-field persistent homology of the complex has been computed in 16 minutes for all coefficient fields $\mathbb{Z}_p$ for the

```
Graph g; ...  //compute the graph
Simplex_tree< linear_indexing_tag > st; //linear ordering
st.insert(g); //insert the graph as 1-skeleton of the complex
st.expand(4); //construct the 4-skeleton of the associated flag
    complex
Persistent_cohomology< Simplex_tree<linear_indexing_tag>,
    Multi_field > pcoh; //persistence with "multi field
    coefficients" defined on a simplex tree
pcoh.compute_persistent_homology(st,2,1223); //compute persistent
    homology of st in all fields Zp for p prime between 2 and 1223
```

Figure 14.1: User interface for the construction of a filtered flag complex with a simplex tree and the computation of its persistent homology.

200 first prime numbers p (ie $p \in [2; 1223]$). Note that the total computation per simplex is 2.0×10^{-6} seconds per simplex, including construction of the simplex tree and, for every simplex in the complex, computation of the boundary in the simplex tree and update of the compressed annotation matrix with multi-field coefficients.

We have noticed no difference between the persistent features of the different persistence diagram for the various coefficient fields. This fact garantees that the integral homology groups of the underlying spaces have no torsion subgroup $\mathbb{Z}_{q^k}$, for any $q \leq 1223$. The absence of torsion was expected, as we know that the underlying spaces are a topological circle, a torus and a 3-sphere. However, in a practical case where the underlying spaces are unknown, it is very unlikely that the homology we infer contains torsion subgroups which such big prime divisor of torsion coefficient ($ie > 1223$). We discuss this point in the concluding remarks of the dissertion.

We obtain the persistence diagram presented in Figure 14.2 (only the most persistent bars are pictured; the bars for topological noise are all shorter than 0.063). We note that we approximate properly the circle (from $\rho = 0.0629$ to $\rho = 0.31$), then the torus (from $\rho = 0.316$ to the end of the computation with $\rho = 0.9$). Despite the size of the compex, we are enable to find a feature in 3-dimensional homology. The impossibility of infering the 3-sphere with standard persistence had already been observed in [61], but for the 4-skeleton of a Rips complex with much smaller threshold $\rho = 0.625$, which results into a simplicial complex with less than 24 million simplices.

Consequently, it seems out of reach to infer the topology of a 3-sphere from the Clifford data set using standard persistence. We use our prototype implementation of the zigzag persistent homology algorithm, presented in Chapter 11, in order to compute a more informative persistence diagram.

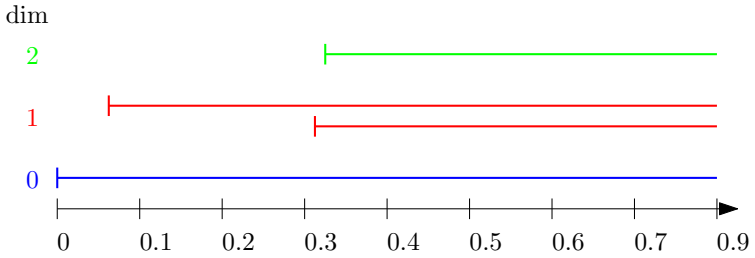


Figure 14.2: Persistence barcode for the standard persistence of the 4-skeleton of the Rips complex ($\rho = 0.9$) on the Clifford data set. The filtration contains 491 millions insertions of simplices.

We use an oscillating Rips zigzag filtration with the multipliers $\eta = 3$ and $\rho = 3.2$, and ordering the points of the Clifford data set using a furthest point strategy. We obtain the persistence diagram pictures in Figure 14.3. The computation took 139 seconds for a filtration containing 14 million arrows and simplicial complexes containing at most 175 thousands simplices. We observe that the 3-sphere appears clearly in the persistence barcode, with a persistent bar $[0.65; 0.74]$, for a much smaller filtration (in number of arrows and maximal complex size) than standard persistence. Note that the zigzag persistence algorithm is efficient in practice, but is nowhere as fast as the persistence cohomology algorithm implemented with a compressed annotation matrix. Specifically, the average computation time per arrow for updating the encoding in zigzag persistence homology is 1.0×10^{-5} seconds (we use $\mathbb{Z}_2$ as coefficient field). The average time per arrow for updating the encoding in standard persistence with a compressed annotation matrix is, on the same example, 5.1×10^{-7} seconds per arrow (we use only $\mathbb{Z}_2$ coefficient and do not consider boundary computation for this timing). We discuss the possibility of adapting all optimizations (compressed annotation matrix and modular reconstruction), developed in this dissertation, to the algorithm for zigzag persistence of Chapter 11 in the concluding remarks.

14.2 Additional Experiments.

Figure 14.4 presents timings T_{st} for the construction of flag complexes with a simplex tree using the algorithm of Chapter 2, $T^{\text{ph}}_{\mathbb{Z}_2}$ and $T^{\text{ph}}_{\mathbb{Z}_{1223}}$ for the computation of persistent homology with coefficient is $\mathbb{Z}_2$ and $\mathbb{Z}_{1223}$ respectively, using the implementation of Chapter 5, and $T^{\text{ph}}_{\mathbb{Z}^2_{1223}}$ for the simultaneous computation of persistent homology in the 200 coefficient fields $\mathbb{Z}_p$ with p prime,

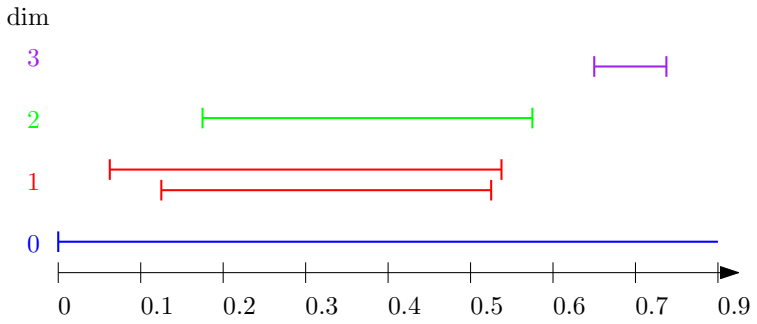


Figure 14.3: Persistence barcode for the zigzag persistence of the 4-skeleton of the oscillating Rips zigzag ($\eta = 3$ and $\rho = 3.2$) on the Clifford data set. The filtration contains 14 million insertion and deletion of simplices, and the maximal size of a simplicial complex is 175 thousand simplices.

for $2 \leq p \leq 1223$, using the multi-field persistent homology algorithm described in Chapter 8. Experiments have been realized on Rips complexes built on a variety of data points. Datasets are listed in Figure 14.4 with the size of points sets $\#\mathcal{P}$, the ambient dimension D and intrinsic dimension d of the sample points ("?" if unknown), the parameter ρ for the Rips complex and the size of the complex $\#\mathbf{K}$.

The average timings per simplex of the various algorithms are ranging between $4.08 \cdot 10^{-8}$ and $7.28 \cdot 10^{-8}$ seconds per simplex for the construction of the simplex tree, between $1.27 \cdot 10^{-6}$ and $2.00 \cdot 10^{-6}$ seconds per simplex for the computation of persistent homology with coefficient field $\mathbb{Z}_2$ or $\mathbb{Z}_{1223}$ (including the computation of boundaries in the simplex tree, for each simplex insertion), and between $1.68 \cdot 10^{-6}$ and $3.52 \cdot 10^{-6}$ seconds per simplex for the computation of multi-field persistent homology in all fields $\mathbb{Z}_p$ for p prime, $2 \leq p \leq 1223$. Note that most of the time to computate the persistent homology is spent in computing boundaries in the simplex tree.

Data	$\#\mathcal{P}$	D	d	r	$\#\mathbf{K}$	T_{st}	$T^{\text{ph}}_{\mathbb{Z}_2}$	$T^{\text{ph}}_{\mathbb{Z}_{1223}}$	$T^{\text{ph}}_{\mathbb{Z}^2_{1223}}$
Bud	49,990	3	2	0.09	$127 \cdot 10^6$	5.7	161	161	252
Bro	15,000	25	?	0.04	$142 \cdot 10^6$	5.8	252	252	380
Cy8	6,040	24	2	0.8	$193 \cdot 10^6$	8.4	249	249	325
Kl	90,000	5	2	0.25	$114 \cdot 10^6$	8.3	228	227	401
S3	50,000	4	3	0.65	$134 \cdot 10^6$	7.2	176	176	310

Figure 14.4: Timings in seconds for the various algorithms.

Appendix A

General Algebra

We recall in this chapter the definitions and properties of basic algebraic structures.

Groups

A *group* is a nonempty set G, called the *underlying set* of the group, together with a binary operation on G, noted + in the additive case, with satisfies the following properties:

(i) **Associativity**: For all $x, y, z \in \mathrm{G}$, $(x + y) + z = x + (y + z)$.

(ii) **Identity**: There exists an element $0_{\mathrm{G}} \in \mathrm{G}$, called the *identity* element of the group, for which: $0 + x = x + 0 = x$ for all $x \in \mathrm{G}$.

(iii) **Inverse**: For each $x \in \mathrm{G}$, there is an element $-x \in \mathrm{G}$, called the inverse of x, for which: $x + (-x) = (-x) + x = 0_{\mathrm{G}}$.

A *subgroup* H of G is a group with the binary relation + of G which underlying set is a non-empty subset of the underlying set of G. Two elements $x, y \in \mathrm{G}$ *commute* if $x+y = y+x$, and a group G is *Abelian*, or *commutative*, if every pair of elements commute. A group is *finite* if the underlying set G is a finite set; otherwise, it is *infinite*. The *order* of a group is the cardinality of the underlying set G, denoted by $|\mathrm{G}|$.

The group we concider in the following are Abelian and, unless mentioned otherwise, are written additively. When there is no ambiguity, the identity 0_{G} of G is denoted 0. If n is a non-negative integer ($n \geq 0$) and x is an element of G, nx denotes the n-fold sum $x + \cdots + x$, where $0x = 0_{\mathrm{G}}$, and $(-n)x$ denotes $n(-x)$.

The *order* of an element $x \in \mathrm{G}, x \neq 0$, is the smallest positive interger n such that $nx = 0$. If no such n exists, x is said to have *infinite order*.

Homomorphisms. A group homomorphism is a relation preserving function between two groups. Let G and H be two groups. A function $f : \mathrm{G} \to \mathrm{H}$ is a *group homomorphism*.

Free Abelian Groups. Let G be an Abelian group. A family $\{x_\alpha\}$ of elements G *generates* G if every element $x \in \mathrm{G}$ can be written as a sum $\sum_\alpha n_\alpha x_\alpha$ where finitely many integers n_α are non-zero. If G is generated by a finite family $\{x_1, \cdots, x_n\}$ of elements, we say that G is *finitely generated* and we write $\mathrm{G} = \langle x_1, \cdots, x_n \rangle$.

A family $\{x_\alpha\}$ of elements in G is a *basis* of G if every $x \in \mathrm{G}$ can be written *uniquely* as a finite sum $\sum_\alpha n_\alpha x_\alpha$. Uniqueness implies that each element x_α has infinite order; that is, x_α generates an infinite cyclic subgroup $\langle x_\alpha \rangle$ of G. A group G is *free* if it has a *basis*.

For an abelian group G, the set of elements $x \in \mathrm{G}$ of finite order form a subgroup T of G, called the *torsion subgroup*. If T vanishes, we say G is *torsion-free*. A free abelian group is necessary torsion-free.

The following theorem classifies finitely-generated groups up to isomorphism:

Theorem A.1 (Structure Theorem of Finitely Generated Abelian Groups)**.** *Let* G *be a finitely-generated Abelian group, then* G *is isomorphic to:*

$$\mathrm{G} \cong \mathbb{Z}^\beta \bigoplus_{q \text{ prime}} \left(\mathbb{Z}_{q^{k_1}} \oplus \cdots \oplus \mathbb{Z}_{q^{k_{t(p,q)}}} \right)$$

for unique integers β *and* q^{k_i}*.*

Ring

A *ring* $(\mathcal{R}, +, \times)$ is a set $\mathcal{R}$ together with two binary operations (addition $+$ and multiplication $\times$) such that:

(i) $(\mathcal{R}, +)$ is a group,

(ii) the multiplication is associative – *ie* $(x \times y) \times z = x \times (y \times z)$ – and distributive over addition – *ie* $x \times (y+z) = x \times y + x \times z$ and $(x+y) \times z = x \times z + y \times z$.

We shall consider only rings which are *commutative*, *ie* for all $x, y \in \mathcal{R}$, $x \times y = y \times x$, and which have an identity element 1, *ie* $\exists 1 \in \mathcal{R}$ such that $1 \times x = x \times 1 = x$ for every $x \in \mathcal{R}$. All the rings we consider satisfy $0 \neq 1$ and the product $x \times y$ of two nonzero elements $x, y \in \mathcal{R} \setminus \{0\}$ is nonzero. Such rings are called *integral domains*.

Principal Ideal Domain. An *ideal* I of $\mathcal{R}$ is a subset of $\mathcal{R}$ that is an additive subgroup and such that for every $x \in \mathcal{R}$ and every $y \in I$, $x \times y \in I$. An ideal is *principal* if it is generated by a single element, *ie* there exists an element $x_0 \in I$ such that I is equal to $x_0 \times \mathcal{R}$, the smallest ideal containing $\mathbf{x}_0$.

A *principal ideal domain* (PID for short) is an integral domain in which every ideal is principal.

Module

Let $\mathcal{R}$ be a ring. A *$\mathcal{R}$-module* $\mathcal{M}$ is an abelian group (written additively) on which $\mathcal{R}$ acts linearly: more precisely, it is a pair $(\mathcal{M}, \mu)$, where $\mathcal{M}$ is an abelian group and μ is a mapping of $\mathcal{R} \times \mathcal{M}$ into $\mathcal{M}$ such that, if we write $r \cdot x$ for $\mu(r, x)$, $r \in \mathcal{R}, x \in \mathcal{M}$, the following axioms are satisfied: for every $r, r' \in \mathcal{R}, x, y \in \mathcal{M}$,

(i) $r \cdot (x + y) = r \cdot x + r \cdot y$,

(ii) $(r + r') \cdot x = r \cdot x + r' \cdot x$,

(iii) $(r \times r') \cdot x = r \cdot (r' \cdot x)$.

Module Homomorphism. Let $\mathcal{M}$ and $\mathcal{N}$ be two $\mathcal{R}$-modules. A mapping $f : \mathcal{M} \to \mathcal{N}$ is an *$\mathcal{R}$-module homomorphism* if: for every $r \in \mathcal{R}, x, y \in \mathcal{M}$,

(i) $f(x + y) = f(x) + f(y)$,

(ii) $f(r \cdot x) = r \cdot f(x)$.

Thus f is a homomorphism of abelian groups which commutes with the action of each $r \in \mathcal{R}$.

Bibliographical Notes

We refer to [59, 63] for more details on Abelian groups, and to [5] for more details on rings and modules.

Concluding Remarks and Open Problems

We have presented in this dissertation a coherent set of data structures and algorithms for computational topology. The contributions range from the representation and manipulation of topological spaces to the computation of the persistent and zigzag persistent homology of filtrations. The methods presented are diverse and solve algorithmic questions in combinatorics, computer algebra and computer arithmetics. We summarize the contributions and discuss the perspective they open.

Simplex Tree. We have first considered the problem of representing and manipulating topological spaces in computational topology. We have introduced the *simplex tree* to represent simplicial complexes. The structure requires only $O(1)$ memory word per simplex, which is optimal when representing general filtered simplicial complexes. In particular, filtered simplicial complexes are the ones we are interested in when computing persistence. The simplex tree allows a wide range of computations on the simplicial complex, from incidence retrieval to construction of simplicial complexes.

An interesting, yet not well understood, question is the effect of the labelling of the vertices, in the simplex tree, on the complexity of some operations. Indeed, the time complexity for computing cofaces and contracting edges depends crucially on the number of nodes containing a label ℓ at a certain depth in the tree. However, the smaller ℓ, the more unlikely it is to find a node deep in the tree with label ℓ, because the vertices of each simplex are ordered. It is an open question to understand the repartition of labels in the simplex tree, depending on the combinatorics of the underlying simplicial complex and the labelling of its vertices.

Compressed Annotation Matrix. We have introduced the *compressed annotation matrix* for computing the persistent cohomology of a filtration. The data structure is a sparse matrix with an extra compression phase. The compressed annotation matrix allows a compact representation of an encoding in persistence, and leads to an implementation with a sparse complexity analysis and with a practical linear-time behavior. This work raises two questions.

1. The complexity of the persistent homology algorithm in cubic, and this bound is tight. However, there is no good complexity model to explain the practical behavior of the persistent (co)homology algorithm. In particular, why do the quantities $c_{\max}$ and $r_{\max}$ (measuring the "sparsity" of the annotation matrix) remain small ? An interesting direction is the analysis of the reordering strategy for iso-simplices, presented in this dissertation as a heuristic. Under certain properties of the simplicial complex (like *shellability*), the iso-simplices reordering tends to insert successively the two simplices of collapsible pairs, which leads to constant time updates in the annotation matrix. Can we generalize this idea to a wider class of simplicial complexes (beyond shellable) ?

   There are similarities between these questions and the Morse simplification approach for persistent homology [55]. A gradient vector field in discrete Morse theory pairs codimension 1 simplices and defines a simplification order. To which extent is the iso-simplices ordering related to a Morse simplification ? Morse simplifications have been well-studied [23], and relating our algorithm with them is of interest.

2. The effect of the compression of the annotation matrix reduces drastically the number of operations. It is not well-understood why. As presented earlier, in 0-cohomology two vertices have the same annotation vector *iff* they belong to the same connected component. Can we generalize the criterion to higher dimensions, so as to explain why so many simplices have the same annotation vector in practice?

Modular Reconstruction. We have introduced the modular reconstruction method for multi-field persistent homology. This work is the only attempt we know to compute persistence with different coefficient fields efficiently.

In order to infer properly the prime divisor of torsion coefficients, we have assumed a bound on the torsions to be known. This bound may be computed with Hadamard's formula or the ovals of Cassini, but these bounds

are usually big. In the context of topological inference, given a set of points on a compact, and under reasonnable sampling conditions, is it possible to improve the bound on the torsion coefficient, using only the number of points and geometric quantities ?

Zigzag Persistence. We have introduced an algorithm for zigzag persistence. The novelty of the approach and of the mathmetical tools introduced raises some questions:

1 A main motivation for introducing our zigzag persistent algorithm was its simplicity and proximity with the standard persistence algorithm. A direct consequence is the possibility to adapt the methods presented in Parts I, II and III to the zigzag framework, in order to obtain a very efficient zigzag persistence algorithm in practice. In particular, this requires to define a cohomology version of the zigzag algorithm. This is a natural extension to the work presented in this dissertation.

2 On the theoretical side, the weak diamond principles we have introduced hold in a fairly specific setting, but the techniques developed in our proofs (in particular the vector space of representative sequences) may be used in a larger context. One goal is to handle weak diamonds with arbitrary maps. Such a result would allow a zigzag persistence algorithm with general simplicial maps in the filtration.

3 Our arrow permutations are a special type of vineyards [31], obtained by sequences of transpositions in the descending chain of the zigzag. Devising general rules for transpositions of simplex insertions and deletions in a general position in an arbitrary zigzag seems within our reach, as such transpositions reduce basically to three kinds of diamonds, depending on the arrow orientations: weak diamonds and transposition diamonds as presented, and Mayer-Vietoris diamonds as in [24].

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] GMP, the GNU Multiple Precision arithmetic library. http://gmplib.org/.

[3] AIM@SHAPE. Neptune. http://www.aimatshape.net/.

[4] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[5] Michael Atiyah and Ian MacDonald. *Introduction To Commutative Algebra*. Addison-Wesley series in mathematics. Westview Press, 1994.

[6] Dominique Attali, André Lieutier, and David Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *Int. J. Comput. Geometry Appl.*, 22(4):279–304, 2012.

[7] Dominique Attali, André Lieutier, and David Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *Int. J. Comput. Geometry Appl.*, 22(4):279–304, 2012.

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

[9] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Phat, 2013. https://code.google.com/p/phat/.

[10] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization III*, pages 103–117. 2014.

[11] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology. In *ALENEX*, pages 31–38, 2014.

[12] Jon Louis Bentley and Robert Sedgewick. Fast algorithms for sorting and searching strings. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana.*, pages 360–369, 1997.

[13] I. N. Bernstein, I. M. Gelfand, and V. A. Ponomarev. Coxeter functors and Gabriel's theorem. *Russian Mathematical Surveys*, 28(2):17–32, 1973.

[14] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22:5–19, 2002.

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

[16] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. *Algorithmica*, 2015.

[17] Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. Relaxed Delaunay triangulations over finite universes.

[18] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *ESA*, pages 731–742, 2012.

[19] Jean-Daniel Boissonnat and Clément Maria. Computing persistent homology with various coefficient fields in a single pass. In *ESA*, 2014.

[20] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427, 2014.

[21] Erik Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete & Computational Geometry*, 9:387–426, 1993.

[22] Peter Bubenik and Jonathan A. Scott. Categorification of persistent homology. *Discrete & Computational Geometry*, 51(3):600–627, 2014.

[23] Benjamin A. Burton, Thomas Lewiner, João Paixão, and Jonathan Spreer. Parameterized complexity of discrete Morse theory. In *Symposium on Computational Geometry*, pages 127–136, 2013.

[24] Gunnar E. Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[25] Gunnar E. Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Symposium on Computational Geometry*, pages 247–256, 2009.

[26] Gunnar E. Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76(1):1–12, 2008.

[27] Frédéric Chazal, David Cohen-Steiner, and André Lieutier. A sampling theory for compact sets in euclidean space. *Discrete & Computational Geometry*, 41(3):461–479, 2009.

[28] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, 2011.

[29] Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. *Comput. Geom.*, 46(4):435–447, 2013.

[30] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[31] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Symposium on Computational Geometry*, pages 119–126, 2006.

[32] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[33] Vin de Silva. A weak characterisation of the delaunay triangulation. *Geometriae Dedicata*, 135(1):39–64, 2008.

[34] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co)homology. *CoRR*, abs/1107.5665, 2011.

[35] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete & Computational Geometry*, 45(4):737–759, 2011.

[36] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes. In *Symposium on Computational Geometry*, pages 232–239, 1993.

[37] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771–784, 1995.

[38] Harm Derksen and Jerzy Weyman. Quiver representations. *Notices of the AMS*, 52(2):200–206, 2005.

[39] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S*, 66:23–45, 1998.

[40] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. In *Symposium on Computational Geometry*, page 345, 2014.

[41] Jean-Guillaume Dumas, B. David Saunders, and Gilles Villard. On efficient sparse integer matrix smith normal form computations. *J. Symb. Comput.*, 32(1/2):71–99, 2001.

[42] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.

[43] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[44] R. Engelking. *General topology*. Monografie matematyczne. PWN, 1977.

[45] Martin Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[46] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[47] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 1 edition, December 2001.

[48] Ann B. Lee, Kim Steenstrup Pedersen, and David Mumford. The nonlinear statistics of high-contrast patches in natural images. *International Journal of Computer Vision*, 54(1-3):83–103, 2003.

[49] Pascal Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Int. J. Comput. Geometry Appl.*, 4(3):275–324, 1994.

[50] Clément Maria. Gudhi, simplicial complexes and persistent homology packages. `https://project.inria.fr/gudhi/software/`.

[51] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The Gudhi library: Simplicial complexes and persistent homology. In *International Congress on Mathematical Software*, pages 167–174, 2014.

[52] Clément Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. In *SODA*, 2015.

[53] S. Martin, A. Thompson, E.A. Coutsias, and J. Watson. Topology of cyclo-octane energy landscape. *J Chem Phys*, 132(23):234115, 2010.

[54] Nikola Milosavljevic, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Symposium on Comp. Geom.*, 2011.

[55] Konstantin Mischaikow and Vidit Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry*, 50(2):330–353, 2013.

[56] Dmitriy Morozov. Dionysus. http://www.mrzv.org/software/dionysus/.

[57] Dmitriy Morozov. Persistence algorithm takes cubic time in worst case. *BioGeometry News, Dept. Comput. Sci., Duke Univ*, 2005.

[58] David M. Mount and Sunil Arya. ANN, Approximate Nearest Neighbors Library. http://www.cs.sunysb.edu/ algorith/implement/ANN/implement.shtml.

[59] James R. Munkres. *Elements of algebraic topology.* Addison-Wesley, 1984.

[60] Steve Y. Oudot. Persistence theory: from quiver representations to data analysis. Book in preparation, 2014.

[61] Steve Y. Oudot and Donald R. Sheehy. Zigzag Zoology: Rips Zigzags for Homology Inference. *J. Foundations of Computational Mathematics*, 2014. To appear.

[62] The Stanford 3D Scanning Repository. Happy Buddha. http://graphics.stanford.edu/data/3Dscanrep/.

[63] Steven Roman. *Fundamentals of Group Theory: An Advanced Approach*. SpringerLink : Bücher. Springer Science+Business Media, LLC, 2011.

[64] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *ijm*, 6:64–94, 1962.

[65] Harlan Sexton and Mikael Vejdemo Johansson. JPlex, 2009. http://comptop.stanford.edu/programs/jplex/.

[66] Donald Sheehy. Linear-size approximations to the vietoris-rips filtration. *Discrete & Computational Geometry*, 49(4):778–796, 2013.

[67] Cary Webb. Decomposition of Graded Modules. *Proceedings of the American Mathematical Society*, 94.4:565–571, 1985.

[68] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). *New Results and New Trends in Computer Science*, 555:359 – 370, 1991.

[69] Afra Zomorodian. The tidy set: a minimal simplicial set for computing homology of clique complexes. In *Symposium on Computational Geometry*, pages 257–266, 2010.

[70] Afra Zomorodian and Gunnar E. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

Index