Schreiber Hannah, MSc.

Algorithmic Aspects in standard and non-standard Persistent Homology

DOCTORAL THESIS

to achieve the university degree of

Doktorin der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Dr., Kerber Michael

Institute of Geometry

Graz, July 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date

Signature

Abstract

Persistent homology enables the analysis of evolving topological properties of general data sets through different scales. The standard case is persistent homology of *filtrations*, a sequence of nested complexes $\mathcal{K}_0 \hookrightarrow ... \hookrightarrow \mathcal{K}_m$. The persistent homology is represented by a barcode consisting of the birth and death times of the different cycle classes evolving through the growing complex. Filtrations can be generalized, two examples are *towers* and *zigzag filtrations*. The first allows more general simplicial maps instead of inclusion maps and the latter allows the inclusion map to also go backwards. This thesis is about algorithms for standard filtrations, but also for those different types of sequences.

We first give an efficient way to transform a tower into a filtration with the same persistent homology. Using an improved version of the coning strategy defined by Dey, Fang and Wang [29], we managed to give a construction whose resulting filtration has a size m of at most $O(\Delta \cdot n \log n_0)$, where Δ is the dimension, n the number of inclusions and n_0 the number of included vertices in the original tower. Moreover we give a streaming algorithm to compute the filtration and its persistent homology in $O(\Delta \cdot m \cdot C_\omega)$ time and space complexity $O(\Delta \cdot \omega)$, where ω is the size of the largest complex in the original tower and C_ω the cost of an operation in a dictionary with ω elements.

The second part yields to reduce the proceeded data size for zigzag filtrations using *discrete Morse theory*. The idea is to pair cells in the complexes together such that the removal of those pairs does not affect the homology of the complex, reducing this way the size of the complex. The complexity of the pairing of each new inserted face depends mainly on the complexity of computing the boundary and coboundary of the simplex in the Morse complex, and is compensated by the new size of the filtration.

Finally, we discuss the average complexity of the standard persistence algorithm, through different variations of filtrations. We give promising experimental results and first theoretical results for two kinds of filtration.

Acknowledgements

I would like to sincerely thank my supervisor Michael Kerber and my coauthor Clément Maria, as well as the members of the GUDHI Project at INRIA, who helped me to integrate the group as contributor. I am also very thankful to Tamal K. Dey and Steve Y. Oudot for accepting to review this thesis.

I want also to thank Wilfried Imrich for introducing me to the subject of symmetry breaking in graphs, which lead to two interesting publications together with Svenja Hüning, Judith Kloas and Thomas W. Tucker.

Contents

Chapter 1

Introduction

1.1 TDA and persistent homology

Topological Data Analysis (TDA) aims to analyze a dataset by first associating it to a "shape" (i.e., its topology), which is then interpreted (e.g. by comparison). A main tool for TDA is *persistent homology*, an algebraic method which enables to study the evolution of the topology in a sequences of spaces $X_1 \subseteq \ldots \subseteq X_n$, called a *filtration*. At each inclusion, it enables to keep track of the births and deaths of cycle classes in different dimensions — we will go deeper into details of the idea in Section 1.3. The birth/death pairs are then stored in what is called the *barcode* or *persistence diagram* of the filtration. It found many applications in various fields. To cite a few examples:

- In material science, TDA was used to study granular crystallization arising from irregular clusters. The crystallization was represented by 3- dimensional packings of frictional spheres and was studied at the grain-scale. The center of the spheres were measured by X-ray tomography, which were then used to build a Čech filtrations [56].

- TDA can also be a tool for shape classification. Using the tangents and information about the curvature of each point in the shape, one can define a tangent complex and a filtered tangent complex [11].

- In [15], a clustering scheme is defined which combines a common cluster algorithm with a cluster merging step guided by persistence.

- In genetics, persistence homology can help to characterize vertical evolution (mutations over a number of generations) and horizontal evolution (mixture of genomic material between individuals of different lineages). Both can be represented as a graph with a poset structure whose 1-homology can be studied. Horizontal evolution can also be represented as a filtration with a growing parameter representing the genetic distance for a given population of genomic sequences [12].

- In medicine, TDA support non supervised diagnostic of symptoms with distinct signatures, as for a particular type of breast cancer [53].

The success of persistent homology relies on sound theoretical foundations [33, 34, 59], favorable stability properties [4, 14, 19], but also on fast algorithms in

practice despite a theoretical cubical[1] worst case complexity. The worst case complexity arises from the reduction — a variation of Gaussian elimination — of the *boundary matrix*, representing the boundaries of the cells in the last complex in order of inclusion. One reason for the good practical results in practice is a range of improvements discovered in the last years which improve the runtime. One line of research exploits the special structure of the boundary matrix to speed up the reduction process [16]. This idea has led to efficient parallel algorithms for persistence in shared [1] and distributed memory [2]. Moreover, of same importance as the reduction strategy is an appropriate choice of data structures in the reduction process as demonstrated by the `PHAT` library [3]. A parallel development was the development of dual algorithms using persistent cohomology, based on the observation that the resulting barcode is identical [23]. The *annotation algorithm* [29, 5] is an optimized variant of this idea realized in the `GUDHI` library [58]. It is commonly considered as an advantage of annotations that only a cohomology basis must be kept during the process, making it more space efficient than reduction-based approaches.

Generalizations. The annotation algorithm [29] just cited was in fact developed for a more general type of filtration: *simplicial map filtrations* or also called *towers*, which allows not only inclusions but also contractions of vertices (or compositions of the two). It keeps track of the cohomology basis with help of *annotations* assigned to each simplex. When including a simplex, it is assigned a annotation depending of the annotations of its boundary. Then the annotation of the other simplices are updated depending on the new annotation. For the contraction of two vertices, it uses the homology preserving property of the link condition to update correctly the annotation of the simplices before contracting. In practice, towers allow for example to simplify approximation schemes for Vietoris-Rips and Čech filtrations [8, 29], which exact computing is greatly space consuming.

Another type of generalization are *zigzag filtrations* which allows the inclusion maps to not only go forwards, but also to go backwards. It produces for example better information in topology inference [54] as standard persistence or can be useful to connect non-related spaces together through their union $\mathbb{K}_i \to \mathbb{K}_i \cup \mathbb{K}_{i+2} \leftarrow \mathbb{K}_{i+2}$ (or intersection), as for example in level set zigzag persistence [10]. The theory of zigzag persistence was introduced in [9] and led to theoretical [48] and practical [10, 45] algorithms to compute its barcode. The latter are also based on matrix operations, as for standard persistence, but the matrix encodes more information, needed because for the eventual removal of a face. Moreover, in the standard case, the birth times were recorded in the order of inclusion, but the removal of a face in a complex can lead to a reordering of the birth times of cycle classes which are not dead yet. Some older columns can therefore be updated and even reordered.

1.2 Our contributions

This work will be a recollection of three different contributions in the just mentioned subjects about persistence homology.

[1] The current best upper bound is actually $O(n^\omega)$, where ω is the matrix multiplication exponent [48], but the corresponding algorithm is not used in practice.

From towers to filtrations. The first contribution will concern an algorithm for towers. As mentioned in Section 1.1, there already exists an algorithm for such type of sequences by Dey, Fang and Wang [29] which uses persistent cohomology. But the standard persistent homology for filtrations were extensively studied and optimized, and therefore, we were interested by making use of it. Our contribution is an algorithm which transforms a tower $\mathcal{T}$ into a standard filtration $\mathcal{F}$ with same persistence barcode. We then use the known algorithms to compute the persistence from the resulting filtration. The method we used is a slightly modified version of the coning strategy that Dey, Fang and Wang introduced in [29]. The original version could lead to an exponential growth of the resulting filtration and thus we introduced two simple heuristics, which gave us a good guarantee: if n is the number of simplices included in $\mathcal{T}$, n_0 the number of vertices and Δ the maximal dimension of any complex in $\mathcal{T}$, then the size of $\mathcal{F}$ is in the order of $O(\Delta \cdot n \cdot \log n_0)$. In our experiments, the resulting filtration was even way smaller. Then the algorithm we use to compute the final barcode is based on the usual persistence algorithm but transformed into a streaming algorithm. The reason is that, contrary to standard filtration, the maximal width of a tower does not depend of its length. So, we can build a very long tower without having systematically memory issues and thus we would like to be able to compute its barcode without loading the whole filtration. Additionally, we use some simple characteristic of a tower to maintain a memory use in the order of $O(\omega^2)$, with ω being the width of the tower. The streaming algorithm has then a time complexity in the order of $O(\omega^2 \cdot \Delta \cdot n \cdot \log n_0)$.

Zigzag filtrations and discrete Morse theory. Our second contribution proposes a preprocess for the algorithms for zigzag persistence. Even though the worst case complexity is the same than for standard persistence, in practice, zigzag persistence does not perform as well. To partially remedy to this problem, we use *discrete Morse theory* to "shrink" all the complexes in the zigzag filtration, such that less has to be processed by a zigzag persistence algorithm. The idea of discrete Morse theory is to pair together different cells of a complex such that removing those cells does not modify the homology of the complex. The resulting complex is called a *Morse complex*. This strategy was already applied for standard persistence [49] by reducing the set of cells added at each inclusion. The difficulty with zigzag persistence is that a cell which was included can change its status in the zigzag filtration when it is removed. If this cell was paired, its assigned pair needs to be included in the Morse complex, which is not a trivial operation. Escolar and Hiraoka [36] went around this problem by making sure that their do not pair cells together which are not removed at the same time. But this require to look at the whole filtration. As for towers, the complexes in a zigzag filtration can remain relatively small, such that we want to be able to proceed very long filtrations. Our contribution is a streaming like algorithm which reduces the complexes and is able to deal with the eventual separation of a pair, such that it only has to know the currently proceeded complex in the zigzag filtration.

Standard filtrations and average complexity. All algorithms used in practice for standard persistence are based on a left-to-right column additions, a variation of the classical Gaussian elimination process. Therefore the worst

case complexity is cubic in the number of simplices. Unfortunately, there exists filtrations reaching this worst case. But the reason that this algorithm is nevertheless so popular is that in practice, we observe a behavior which seems to be linear and thus far from being cubical. We measured experimentally different interesting parameters for four different types of filtrations. The most "random" filtration behaves around a squared complexity, whereas more "organized" filtrations are clearly below. We also give two theoretical results concerning two opposite types of filtration. We prove that the first has a quasi-linear complexity, whereas a variant of the second looses in average an order of magnitude with respect to its worst case.

Software. We would like to point out that both algorithms for towers and zigzag filtrations were implemented by the author, with help from Clément Maria[2] for the latter, and integrated in the in the generic open source library GUDHI [58] (to be released).

Outline. In the remaining of this work, the next chapters will specify our contributions. But first, the general topic is introduced in more details and more formally in the next section. Then, Chapter 2 will cover our contribution for towers, Chapter 3 the one for zigzag filtrations and finally, we will discuss and analyze in Chapter 4 the expected complexity for standard persistence.

1.3 From data to barcode

The typical TDA pipeline is composed of three steps:

1. Convert the input data (usually a point cloud, for example obtained from scanning a 3D object) into a convenient representation and practical data structure from which the necessary topological information can be retrieved;

2. Compute a range of topological information;

3. Analyze the results.

The first step arises from the necessity of discretizing an element in order to digitally analyze it. As a simplified example, take an annulus-shaped object you would like to create a digital model of. As first step, you could scan it. For obvious technical reasons, you cannot expect retrieving every single point of the surface of your object, but only a (hopefully) representative sample. The result is what is called a *point cloud*, a finite set of coordinates in a metric space, as in the left part of Figure 1.1. But now that you are only left with a bunch of points, how do we retrieve the connection between them? Because an annulus is not an annulus without a hole in the middle — if it seems obvious for human eyes to identify the lack of points in the middle of the Figure as being a hole, it is not obvious at all for the computer. We first need to associate a shape to those point.

[2]INRIA Sophia Antipolis-Méditerranée, France – clement.maria@inria.fr

Simplicial complexes. Given a finite *vertex set* V, a *simplex* is a non-empty subset of V. We call it a d-simplex if the subset contains exactly $d+1$ vertices; d is then the *dimension* of the simplex. A vertex is therefore a 0-dimensional simplex and we call a 1-dimensional simplex an *edge*, and 2-dimensional simplex a *triangle*. Let σ be a d-simplex and τ a d'-simplex. We say that τ is a face of σ and σ a coface of τ, if $\tau \subseteq \sigma$. When the inclusion is strict, we talk about a *proper* face or coface. Moreover, τ is a facet of σ and σ a cofacet of τ, if $d = d' + 1$.

An *(abstract) simplicial complex* $\mathbb{K}$ over V is a set of simplices that is closed under taking faces, i.e., if $\sigma \in \mathbb{K}$ and $\tau \subset \sigma$, then $\tau \in \mathbb{K}$. The *dimension* of $\mathbb{K}$ is the maximal dimension of its simplices. A simplex $\sigma \in \mathbb{K}$ is *maximal* in $\mathbb{K}$ if none of its cofaces is contained in $\mathbb{K}$.

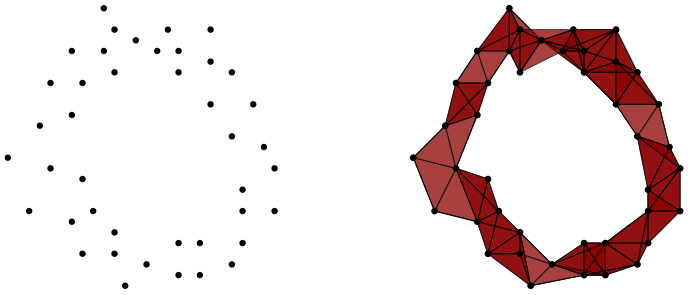


Figure 1.1: Example of 2D point cloud shaped as an annulus (left) and of a simplicial complex defined on those points (right).

Simplicial complexes are commonly used as data structure in TDA. Back to our point cloud, an example of simplicial complex over the set of points is shown in the right part of Figure 1.1. We will mainly make use of simplicial complexes, but we will also need a more general definition of complexes:

Cell Complexes. An *abstract complex* over a principal ideal domain $\mathbf{D}$ (such as the ring of integers $\mathbb{Z}$ or a field $\mathbb{Z}/p\mathbb{Z}$ for p prime) is a graded finite collection $X = \bigsqcup_{d\in\mathbb{Z}} X_d$ of elements, called *cells* or *faces*, together with an *incidence function* $[\cdot:\cdot]^X : X \times X \to \mathbf{D}$. The *dimension* of a cell $\sigma \in X_d$ is $\dim \sigma = d$. The incidence function satisfies, for any cells σ, τ, and μ:

$$[\sigma:\tau]^X \neq 0 \Rightarrow \dim\sigma = \dim\tau + 1 \quad \text{and} \quad \sum_{\tau\in X} [\sigma:\tau]^X \cdot [\tau:\mu]^X = 0.$$

Then, τ is a *facet* of σ, and σ a *cofacet* of τ, if $[\sigma:\tau]^X \neq 0$. And a cell is called *maximal*, if it has no cofacet.

In this new context, a simplicial complex is a complex, where the cells of dimension d are d-simplices with a fixed orientation. The principal ideal domain $\mathbf{D}$ is then the ring of integers $\mathbb{Z}$, and the incidence function takes values in $\{-1, 0, 1\} \subset \mathbb{Z}$.

In this work, the topological invariant we are interested to measure is the homology of our data (or more exactly, we will look at the persistent homology as we will see later). For this purpose, we will need the notion of boundary:

Boundary Operator. For a field $\mathbb{F}$ of coefficients, we associate to a complex X a *chain complex* $C(X,\mathbb{F}) = \bigoplus_d C_d(X,\mathbb{F})$, where $C_d(X,\mathbb{F})$ is the $\mathbb{F}$-vector space freely generated by the d-dimensional cells in X_d of X. To simplify the notations, we fix $\mathbb{F}$ from now on and remove it from any notation. For every dimension d, the *boundary operator* $\partial_d^X : C_d(X) \to C_{d-1}(X)$ is generated by:

$$\partial_d^X \sigma = \sum_{\tau \in X_{d-1}} [\sigma : \tau]^X \cdot \tau.$$

When there is no ambiguity, we remove the superscript X from the notation: ∂_d. To put emphasis on the boundary operator, we denote a complex by (X, ∂), where $\partial\colon C(X) \to C(X)$ is $\partial = \bigoplus_d \partial_d^X$. For a cell σ, we call $\partial\sigma$ the *boundary* of σ. We say that a cell τ is in the boundary of σ if $\dim \sigma = \dim \tau + 1$ and the coefficient of τ is non-zero in $\partial\sigma$.

By a small abuse of notation, we refer also to X as a chain complex provided there is no ambiguity in the definition of their incidence function and boundary maps.

Homology. In a complex (X, ∂), the *d-cycles* are defined as $Z_d(X) = \ker \partial_d$ and the *d-boundaries* as $B_d(X) = \operatorname{im} \partial_{d+1}$. Then the d^{th} *homology group* is the quotient

$$H_d(X) = {}^{Z_d(X)}\!\!\diagup_{B_d(X)}.$$

We also use the notation $H(X)$, when the dimension does not matter and the context or properties can be applied to any dimension.

To visualize, homology gather cycles which can be continuously transformed into each other through the faces in a same class. In Figure 1.1 (right), the complex has one non-trivial 0-homology class, which can be seen as the representative of the single connected component. It also has one non-trivial 1-homology class representing the cycles going around the middle hole, and therefore are not continuously contractible to a point.

Another way to express the homology of a complex X are *Betti numbers*. The d^{th} Betti number $\beta_d(X)$ is the rank of $H_d(X)$. For Figure 1.1 (right), we have $\beta_0 = \beta_1 = 1$ and $\beta_i = 0$ for $i \geq 2$.

But now, we are still left with a problem: how do we determine the right complex with the right homology for a set of point? If we look at Figure 1.2, why should the complex on the right be more or less valid than the complex on the left for our initial set of points? The solution is quit simple: we construct not one complex, but a whole range of complexes — filtrations — and see which homology features persist the most through the different complexes — persistent homology.

Filtrations. A *filtration* $\mathcal{F}$ of length m is sequence of complexes $X_0 \subseteq ... \subseteq X_m$ connected by inclusion maps i_j, $j \in \{0, ..., m-1\}$:

$$\mathcal{F}: \; X_0 \xhookrightarrow{i_0} X_1 \xhookrightarrow{i_1} \cdots \xhookrightarrow{i_{m-2}} X_{m-1} \xhookrightarrow{i_{m-1}} X_m \; .$$

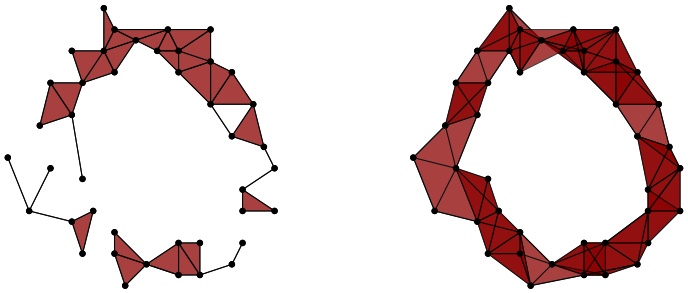


Figure 1.2: Example of two different simplicial complexes defined on the same set of vertices.

We can generalize a filtration with a *zigzag filtration* $\mathcal{Z}$ for which the inclusion maps are also allowed to go backwards:

$$\mathcal{Z}: \; X_0 \overset{i_0}{\longleftrightarrow} X_1 \overset{i_1}{\longleftrightarrow} \cdots \overset{i_{m-2}}{\longleftrightarrow} X_{m-1} \overset{i_{m-1}}{\longleftrightarrow} X_m \;,$$

where an arrow $\leftrightarrow$ is either going forward or going backward. Sometimes, we will call a filtration a *standard filtration*, to clearly distinguish it from a zigzag filtration.

If the complexes are simplicial complexes, we can also generalize a filtration with a *tower* $\mathcal{T}$ or also called a *simplicial map filtration*. As the second name indicates, in this case, the inclusion maps are replaced by more general simplicial maps $(s_j)_{j\in\{0,\dots,m-1\}}$, which we will define later:

$$\mathcal{T}: \; \mathbb{K}_0 \xrightarrow{s_0} \mathbb{K}_1 \xrightarrow{s_1} \cdots \xrightarrow{s_{m-2}} \mathbb{K}_{m-1} \xrightarrow{s_{m-1}} \mathbb{K}_m \;.$$

The *dimension* of $\mathcal{F}$, $\mathcal{Z}$ and $\mathcal{T}$ is the maximal dimension among the complexes respectively composing them, and their *width* is the maximal size of those complexes. For filtrations, dimension and width are determined by the dimension and size of X_m, but this is not necessarily true for towers.

Standard filtrations are the most commonly used contructions, as for example Vietoris-Rips and Čech filtrations which take account of the distances between the vertices to be constructed. One reason for their popularity is that the algorithm for persistent homology is then not only quite simple, but also well performing in practice, see Chapter 4. But their properties are clearly limited and it would be interesting to also have well performing algorithms for the other two types of filtrations. That is the purpose of Chapter 2 and 3 for towers and zigzag filtrations respectively.

Modules and morphisms. For our fixed base field $\mathbb{F}$ and a complex X, it is well-known that $H_d(X)$ is a $\mathbb{F}$-vector space and that an inclusion or simplicial map $f : X \to X'$ induces a linear map $f^* : H_d(X) \to H_d(X')$. In categorical terms, the equivalent statement is that homology is a functor from the category of simplicial complexes and simplicial maps to the category of vector spaces and linear maps. Therefore, on each of the above defined filtrations, we can apply the homology functor to obtain the corresponding *persistence module*:

$$H(\mathcal{F}): \; H(X_0) \overset{f_0^*}{\longleftrightarrow} H(X_1) \overset{f_1^*}{\longleftrightarrow} \cdots \overset{f_{m-2}^*}{\longleftrightarrow} H(X_{m-1}) \overset{f_{m-1}^*}{\longleftrightarrow} H(X_m) \;,$$

where each f_j^* is a linear map induced by the initial inclusion or simplicial map. Persistence modules are realizations of representations:

An *A_m-type quiver* $\mathcal{Q}$ is a directed graph:

$$\bullet_1 \longleftrightarrow \bullet_2 \longleftrightarrow \cdots \longleftrightarrow \bullet_{m-1} \longleftrightarrow \bullet_m \ ,$$

where bidirectional arrows are either going forward or going backward.

An *$\mathbb{F}$-representation* of $\mathcal{Q}$ is an assignment of a finite dimensional $\mathbb{F}$-vector space V_i for every node $\bullet_i$ and an assignment of a linear map $f_i : V_i \leftrightarrow V_{i+1}$ for every arrow $\bullet_i \leftrightarrow \bullet_{i+1}$, the orientation of the map being the same as that of the arrow. We denote such a representation by $\mathbb{V} = (V_i, f_i)$. In our context, an $\mathbb{F}$-representation of an A_m-type quiver is called a *zigzag module* (Or just *module* if all arrows are going forward).

Let $\mathbb{V} = (V_i, f_i)$ and $\mathbb{W} = (W_i, g_i)$ be two $\mathbb{F}$-representations of a same quiver $\mathcal{Q}$. A *morphism of representations* $\Phi : \mathbb{V} \to \mathbb{W}$ is a set of linear maps $\{\Phi_i : V_i \to W_i\}_{i=1\ldots n}$ such that the following diagram commutes for every arrow of $\mathcal{Q}$:

$$\begin{array}{ccc} V_i & \xleftrightarrow{f_i} & V_{i+1} \\ {\scriptstyle \Phi_i}\downarrow & & \downarrow{\scriptstyle \Phi_{i+1}} \\ W_i & \xleftrightarrow{g_i} & W_{i+1} \end{array} \ .$$

The morphism is called an *isomorphism* (denoted by $\cong$) if every Φ_i is bijective. $\mathbb{V}$ and $\mathbb{W}$ are then said to be *isomorphic*.

The *direct sum* of two $\mathbb{F}$-representations $\mathbb{V} = (V_i, f_i)$, $\mathbb{W} = (W_i, g_i)$, denoted by $\mathbb{V} \oplus \mathbb{W}$, is the representation of $\mathcal{Q}$ with spaces $V_i \oplus W_i$ for every node $\bullet_i$, and with map $f_i \oplus g_i = \begin{pmatrix} f_i & 0 \\ 0 & g_i \end{pmatrix}$ for every arrow $\bullet_i \leftrightarrow \bullet_{i+1}$. An $\mathbb{F}$-representation $\mathbb{V}$ is *decomposable* if it can be written as the direct sum of two non-trivial representations. It is otherwise said to be *indecomposable*.

Persistent Homology. For any $1 \leq b \leq d \leq m$, define the *interval representation* $\mathbb{I}[b; d]$ as follows:

$$\underbrace{0 \xleftrightarrow{0} \cdots \xleftrightarrow{0} 0}_{b-1 \text{ times}} \xleftrightarrow{0} \underbrace{\mathbb{F} \xleftrightarrow{\mathbb{1}} \cdots \xleftrightarrow{\mathbb{1}} \mathbb{F}}_{d-b+1 \text{ times}} \xleftrightarrow{0} \underbrace{0 \xleftrightarrow{0} \cdots \xleftrightarrow{0} 0}_{m-d \text{ times}},$$

where the maps 0 and $\mathbb{1}$ stand respectively for the null map and the identity map.

Theorem 1.1 (Krull-Remak-Schmidt, Gabriel)**.** *Every $\mathbb{F}$-representation $\mathbb{V}$ of an A_m-type quiver can be decomposed as a direct sum of indecomposables: $\mathbb{V} \cong \mathbb{V}^1 \oplus \mathbb{V}^2 \oplus \cdots \oplus \mathbb{V}^N$, where each indecomposable $\mathbb{V}^j$ is isomorphic to some interval representation $\mathbb{I}[b_j; d_j]$. This decomposition is unique up to permutation of the indecomposables.*

Such a decomposition of a persistence module is called an *interval decomposition*. Because this decomposition is uniquely defined up to isomorphisms and re-ordering, the pairs (b_1, d_1), ..., (b_N, d_N) are an invariant of the persistence module, called its *barcode* or *persistence diagram*. Important homological features are represented by pairs (b, d), where $d - b$ is big enough: the higher the

value of $d-b$, the longer the corresponding feature persists through the filtration when read from left to right. Therefore, computing the persistent homology of a filtration means computing exactly those pairs, called *persistence pairs*. See Figure 1.3 as example of barcode.

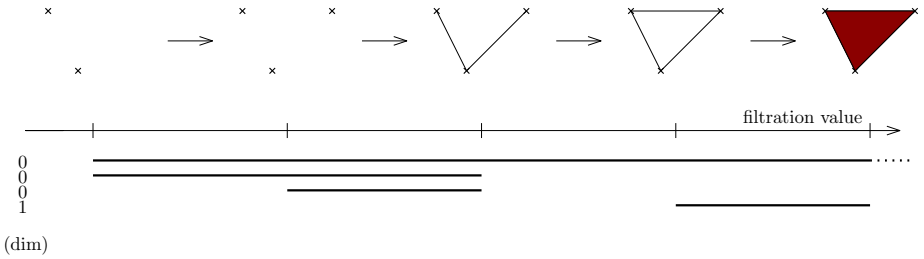


Figure 1.3: Example of filtration and its persistence barcode: each bar corresponds the appearance and disappearance of a cycle class.

Matrix Reduction. As already mentioned, computing the persistent homology of a standard filtration is relatively simple. All algorithm used in practice are based on *matrix reduction*. To simplify the explanation, assume we have a filtration $\mathcal{F} : X_0 \to ... \to X_m$ where $X_i \setminus X_{i-1} = \{\sigma_i\}$ for $i \in \{1, ..., m\}$, that is, X_i has exactly one more cell than X_{i-1}. Then $\mathcal{F}$ can be encoded by a *boundary matrix* B: each column and row represent a cell in X_m, and are ordered with respect to the order the cells where inserted in $\mathcal{F}$. Therefore, the column i represents the cell σ_i and the row j the cell σ_j. Then the matrix cell (i, j) contains the coefficient of σ_j in $\partial\sigma_i$ if σ_j is a facet of σ_i, and zero otherwise. In other words, the i^{th} column of B encodes the facets of σ_i, and the j^{th} row of B encodes the cofacets of σ_j. Note that B is upper-triangular. Moreover, we will sometimes identify rows and columns in B with the corresponding cell in X_m.

Adding the d-cell σ_i to X_i either introduces one new homology class of dimension d in X_i, or turns a non-trivial homology class of dimension $d-1$ trivial. We call σ_i and the i-th column of B *positive* or *negative* respectively (with respect to the given filtration).

In an arbitrary matrix M, a *left-to-right column addition* is an operation of the form $M_k \leftarrow M_k + M_\ell$ with $\ell < k$, where M_k and M_ℓ are respectively the k^{th} and ℓ^{th} columns of the matrix. The *pivot* of a non-zero column is the largest non-zero index of the corresponding column. A non-zero entry (i, j) is also called a pivot if j is the pivot of the column i. A matrix R is called a *reduction* of M if R is obtained by a sequence of left-to-right column additions from M and no two columns in R have the same pivot. Usually, R is computed as shown in Algorithm 1.

It is well-known that, although B does not have a unique reduction, the pivots of all its reductions are the same. And those pivots (b_1, d_1), ..., (b_N, d_N) are exactly the persistence pairs of $\mathcal{F}$ we wanted to compute. Note, that a direct consequence is that a cell σ_i is positive if and only if the i^{th} column in the reduction of B is zero.

Algorithm 1: Reduction algorithm for the boundary matrix M.

input : M
output: R
The function `pivot` returns the index of the lowest non-zero entry of the given column.
for $i = 1 \ldots n_{d+1}$ **do**
  while $\exists\, j \in \{1, ..., i-1\}$ `pivot`$(M[j])$ == `pivot`$(M[i])$ **do**
    $M[i] \leftarrow M[i] + M[j]$;
  end
end

Chapter 2

Towers to Filtrations

This chapter is based on the paper *Barcodes of Towers and a Streaming Algorithm for Persistent Homology* [43] which is joint work with Michael Kerber[1].

2.1 Introduction

Motivation and problem statement. In this chapter, we consider the persistent homology of towers. We recall that a tower of length m is a sequence of simplicial complexes $(\mathbb{K}_i)_{i=0,\dots,m}$ and simplicial maps $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ connecting them. Applying the homology functor with an arbitrary field, we obtain a *persistence module*, a sequence of vector spaces connected by linear maps. Such a module decomposes into a *barcode*, a collection of intervals, each representing a homological feature in the tower that spans over the specified range of scales.

Our computational problem is to compute the barcode of a given tower efficiently. As considerable amount of work went into the study of fast algorithms for the more prominent filtration case, see Section 1.1, we will make use of it. The more general case of towers recently received growing interest in the context of sparsification technique for the Vietoris-Rips and Čech complexes; see the related work section below for a detailed discussion.

Results. As our first result, we show that any tower can be *efficiently* converted into a *small* filtration with the same barcode. Using the well-known concept of *mapping cylinders* from algebraic topology [41], it is easy to see that such a conversion is possible in principle. Dey, Fan, and Wang [29] give an explicit construction, called "coning", for the generalized case of *zigzag towers*. Using a simple variant of their coning strategy, we obtain a filtration whose size is only marginally larger than the length of the tower in the worst case. Furthermore, we experimentally show that the size is even smaller on realistic instances.

To describe our improved coning strategy, we discuss the case that a simplicial map in the tower contracts two vertices u and v. The coning strategy by Dey et al. proposes to join u with the closed star of v, making all incident simplices of v incident to u without changing the homotopy type. The vertex u

[1]Graz University of Technology, Austria – kerber@tugraz.at

is then taken as the representative of the contracted pair in the further processing of the tower. We refer to the number of simplices that the join operation adds to the complex as the *cost* of the contraction. Quite obviously, the method is symmetric in u and v, and we have two choices to pick the representative, leading to potentially quite different costs. We employ the self-evident strategy to pick the representative that leads to smaller costs (somewhat reminiscent of the "union-by-weight" rule in the union-find data structure [21, §21]). Perhaps surprisingly, this idea leads to an asymptotically improved size bound on the filtration. We prove this by an abstraction to path decompositions on weighted forest which might be of some independent interest. Altogether, the worst-case size of the filtration is $O(\Delta \cdot n \cdot \log(n_0))$, where Δ is the maximal dimension of any complex in the tower, and n/n_0 is the number of simplices/vertices added to the tower.

We also provide a conversion algorithm whose time complexity is roughly proportional to the total number of simplices in the resulting filtration. One immediate benefit is a generic solution to compute barcodes of towers: just convert the tower to a filtration and apply one of the efficient implementations for barcodes of filtrations. Indeed, we experimentally show that on not-too-large towers, our approach is competitive with, and sometimes outperforms Simpers, an alternative approach that computes the barcode of towers with *annotations*, a variant of the persistent cohomology algorithm.

Our second contribution is a space-efficient version of the just mentioned algorithmic pipeline that is applicable to very large towers. To motivate the result, let the *width* of a tower denote the maximal size of any simplicial complex among the $\mathbb{K}_i$. Consider a tower with a very large length (say, larger than the number of bytes in main memory) whose width remains relatively small. In this case, our conversion algorithm yields a filtration that is very large as well. Most existing implementations for barcode computation read the entire filtration on initialization and must be converted to streaming algorithm to handle such instances. Moreover, algorithms based on matrix reduction are required to keep previously reduced columns because they might be needed in subsequent reduction steps. This leads to a high memory consumption for the barcode computation.

We show that with minor modifications, the standard persistent algorithm can be turned into a streaming algorithm with smaller space complexity in the case of towers. The idea is that upon contractions, simplices become *inactive* and cannot get additional cofaces. Our approach makes use of this observation by modifying the boundary matrix such that columns associated to inactive simplices can be removed. Combined with our conversion algorithm, we can compute the barcode of a tower of width ω keeping only up to $O(\omega)$ columns of the boundary matrix in memory. This yields a space complexity of $O(\omega^2)$ and a time complexity of $O((\Delta \cdot n \cdot \log(n_0))\omega^2)$ in the worst case. We implemented a practically improved variant that makes use of additional heuristics to speed up the barcode computation in practice and resembles the *chunk algorithm* presented in [1].

We tested our implementation on various challenging data sets. The source code of the implementation is available[2] and the software was named `Sophia`.

[2] at `https://bitbucket.org/schreiberh/sophia/` and soon in the open library `GUDHI` [58]

Related work. Dey et al. [29] described the first efficient algorithm to compute the barcode of towers. Instead of the aforementioned coning approach explained in their paper, their implementation handles contractions with an empirically smaller number of insertions, based on the link condition. Recently, the authors have released the SimPers library[3] that implements their annotation algorithm from the paper.

The case of towers has received recent attention in the context of approximate Vietoris-Rips and Čech filtrations. The motivation for approximation is that the (exact) topological analysis of a set of n points in d-dimensions requires a filtration of size $O(n^{d+1})$ which is prohibitive for most interesting input sizes. Instead, one aims for a filtration or tower of much smaller size, with the guarantee that the approximate barcode will be close to the exact barcode ("close" usually means that the bottleneck distance between the barcodes on the logarithmic scale is upper bounded; we refer to the cited works for details). The first such type of result by Sheehy [57] resulted in a approximate filtration; however, it has been observed that passing to towers allows more freedom in defining the approximation complexes and somewhat simplifies the approximation schemes conceptually. See [29, 8, 44, 18] for examples. Very recently, the `SimBa` library [30] brings these theoretical approximation techniques for Vietoris-Rips complexes into practice. The approach consists of a geometric layer to compute a tower, and an algebraic layer to compute its barcode, for which they use `SimPers`. Our approach can be seen as an alternative realization of this algebraic layer.

Outline. We introduce the necessary basic concepts in Section 2.2. We describe our conversion algorithm from general towers to barcodes in Section 2.3. The streaming algorithm for persistence is discussed in Section 2.4.

2.2 Background

Joins and Simplicial Subcomplexes. We denote the *vertex set of a simplicial complex* $\mathbb{K}$ over the vertex set V by $\mathcal{V}(\mathbb{K})$ and define it to be exactly V. A simplicial complex $\mathbb{L}$ is a *subcomplex* of the complex $\mathbb{K}$ if $\mathbb{L} \subseteq \mathbb{K}$. Given $\mathcal{W} \subseteq \mathcal{V}(\mathbb{K})$, the *induced subcomplex* by $\mathcal{W}$ is the set of all simplices σ in $\mathbb{K}$ with $\sigma \subseteq \mathcal{W}$. For a simplex σ and a vertex $v \notin \sigma$, we define the *join* $v * \sigma$ as the simplex $\{v\} \cup \sigma$. And for a subcomplex $\mathbb{L} \subseteq \mathbb{K}$ and a vertex $v \in \mathcal{V}(\mathbb{K}) \setminus \mathcal{V}(\mathbb{L})$, we define the join $v * \mathbb{L} := \{v * \sigma \mid \sigma \in \mathbb{L}\}$. For a vertex $v \in \mathbb{K}$, the *star* of v in $\mathbb{K}$, denoted by $\mathrm{St}(v, \mathbb{K})$, is the set of all cofaces of v in $\mathbb{K}$. In general, the star is not a subcomplex, but we can make it a subcomplex by adding all faces of star simplices, which is denoted by the *closed star* $\overline{\mathrm{St}}(v, \mathbb{K})$. Equivalently, the closed star is the smallest subcomplex of $\mathbb{K}$ containing the star of v. The *link* of v, $\mathrm{Lk}(v, \mathbb{K})$, is defined as $\overline{\mathrm{St}}(v, \mathbb{K}) \setminus \mathrm{St}(v, \mathbb{K})$. It can be checked that the link is a subcomplex of $\mathbb{K}$. When the complex is clear from context, we will sometimes omit the $\mathbb{K}$ in the notation of stars and links.

Simplicial maps. A map $s : \mathbb{K} \to \mathbb{K}'$ between simplicial complexes is called *simplicial* if, with $\sigma = \{v_0, \ldots, v_k\} \in \mathbb{K}$, $s(\sigma)$ is equal to $\{s(v_0), \ldots, s(v_k)\}$

[3] `http://web.cse.ohio-state.edu/~tamaldey/SimPers/Simpers.html`

and $s(\sigma)$ is a simplex in $\mathbb{K}'$. By definition, a simplicial map maps vertices to vertices and is completely determined by its action on the vertices. Moreover, the composition of simplicial maps is again simplicial.

A simple example of a simplicial map is the inclusion map $s : \mathbb{L} \hookrightarrow \mathbb{K}$ where $\mathbb{L}$ is a subcomplex of $\mathbb{K}$. If $\mathbb{K} = \mathbb{L} \cup \{\sigma\}$ with $\sigma \notin \mathbb{L}$, we call s an *elementary inclusion*. The simplest example of a non-inclusion simplicial map is $s : \mathbb{K} \to \mathbb{K}'$ such that there exist two vertices $u, v \in \mathbb{K}$ with $\mathcal{V}(\mathbb{K}') = \mathcal{V}(\mathbb{K})\backslash\{v\}$, $s(u) = s(v) = u$, and s is the identity on all remaining vertices of $\mathbb{K}$. We call s an *elementary contraction* and write $(u, v) \rightsquigarrow u$ as a shortcut. These notions were introduced by Dey, Fan and Wang in [29] and they also showed that any simplicial map $s : \mathbb{K} \to \mathbb{K}'$ can be written as the composition of elementary contractions[4] and inclusions.

We recall that a *tower* of length m is a collection of simplicial complexes $\mathbb{K}_0, \ldots, \mathbb{K}_m$ and simplicial maps $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ for $i = 0, \ldots, m-1$. From this initial data, we obtain simplicial maps $s_{i,j} : \mathbb{K}_i \to \mathbb{K}_j$ for $i \leq j$ by composition, where $s_{i,i}$ is simply the identity map on $\mathbb{K}_i$. The term "tower" is taken from category theory, where it denotes a (directed) path in a category with morphisms from objects with smaller indices to objects with larger indices. Indeed, since simplicial complexes form a category with simplicial maps as their morphisms, the specified data defines a tower in this category. Remind that the *dimension* of a tower is the maximal dimension among the $\mathbb{K}_i$, and the *width* of a tower is the maximal size among the $\mathbb{K}_i$.

Homology and Collapses. We will make use of the following homology-preserving operation: a *free face* in $\mathbb{K}$, is a simplex with exactly one proper coface in $\mathbb{K}$. An *elementary collapse* in $\mathbb{K}$ is the operation of removing a free face and its unique coface from $\mathbb{K}$, yielding a subcomplex of $\mathbb{K}$. We say that $\mathbb{K}$ *collapses to* $\mathbb{L}$, if there is a sequence of elementary collapses transforming $\mathbb{K}$ into $\mathbb{L}$. The following result is then well-known:

Lemma 2.1. *Let $\mathbb{K}$ be a simplicial complex that collapses into the complex $\mathbb{L}$. Then, the inclusion map $s : \mathbb{L} \hookrightarrow \mathbb{K}$ induces an isomorphism s^* between $H(\mathbb{L})$ and $H(\mathbb{K})$.*

Matrix reduction. In this chapter, we assume for simplicity that the computation of the barcode is done over the base field $\mathbb{Z}_2$, and interpret the coefficients of the boundary matrix accordingly.

The standard persistence algorithm processes the columns from left to right; recall Algorithm 1. In the j^{th} iteration, as long as the j^{th} column is not empty and has a pivot that appears in a previous column, it performs a left-to-right column addition. We will use a simple improvement of this algorithm that is called *compression*: before reducing the j^{th} column, it first scans through the non-zero entries of the column. If a row index i corresponds to a negative simplex (i.e., if the i^{th} column is not zero at this point in the algorithm), the row index can be deleted without changing the pivots of the matrix. After this initial scan, the column is reduced in the same way as in the standard algorithm.

[4] They talk about "collapses" instead of "contractions", but this notion clashes with the standard notion of simplicial collapses of free faces that we use later. Therefore, we decided to use "contraction", even though the edge between the contracted vertices might not be present in the complex.

See [1, §. 3] for a discussion (we remark that this optimization was also used in [59]).

2.3 From towers to filtrations

We phrase now our first result which says that any tower can be converted into a filtration of only marginally larger size with a space-efficient streaming algorithm:

Theorem 2.2 (Conversion Theorem)**.** *Let*

$$\mathcal{T}: \ \mathbb{K}_0 \xrightarrow{s_0} \mathbb{K}_1 \xrightarrow{s_1} \dots \xrightarrow{s_{m-1}} \mathbb{K}_m$$

be a tower where, w.l.o.g., $\mathbb{K}_0 = \varnothing$ and each s_i is either an elementary inclusion or an elementary contraction. Let Δ denote the dimension and ω the width of the tower, and let $n \leq m$ denote the total number of elementary inclusions, and n_0 the number of vertex inclusions. Then, there exists a filtration

$$\mathcal{F}: \ \hat{\mathbb{K}}_0 \hookrightarrow \hat{\mathbb{K}}_1 \hookrightarrow \dots \hookrightarrow \hat{\mathbb{K}}_m,$$

where the inclusions are not necessarily elementary, such that $\mathcal{T}$ and $\mathcal{F}$ have the same barcode, and the width of the filtration is:

$$|\hat{\mathbb{K}}_m| = O(\Delta \cdot n \log n_0).$$

Moreover, $\mathcal{F}$ can be computed from $\mathcal{T}$ with a streaming algorithm with a complexity of

$$\begin{aligned} &O(\Delta \cdot |\hat{\mathbb{K}}_m| \cdot C_\omega) && \textit{(time)} \\ &O(\Delta \cdot \omega) && \textit{(space)}, \end{aligned}$$

where C_ω is the cost of an operation in a dictionary with ω elements.

The remainder of the section is organized as follows. We define $\mathcal{F}$ in Section 2.3.1 and prove that it yields the same barcode as $\mathcal{T}$ in Section 2.3.2. In Section 2.3.3, we prove the upper bound on the width of the filtration. In Section 2.3.4, we explain the algorithm to compute $\mathcal{F}$ and analyze its time and space complexity.

2.3.1 Active and small coning

Coning. We briefly revisit the *coning strategy* introduced by Dey, Fan and Wang [29]. Let $s : \mathbb{K} \to \mathbb{L}$ be an elementary contraction $(u, v) \rightsquigarrow u$ and define

$$\mathbb{L}^* = \mathbb{K} \cup \left(u * \overline{\mathrm{St}}(v, \mathbb{K})\right).$$

An example is shown in Figure 2.1.

Dey et al. show that $\mathbb{L} \subseteq \mathbb{L}^*$ and that the map induced by inclusion is an isomorphism between $H(\mathbb{L})$ and $H(\mathbb{L}^*)$. By applying this result at any elementary contraction, this implies that every zigzag tower can be transformed into a zigzag filtration with identical barcode.

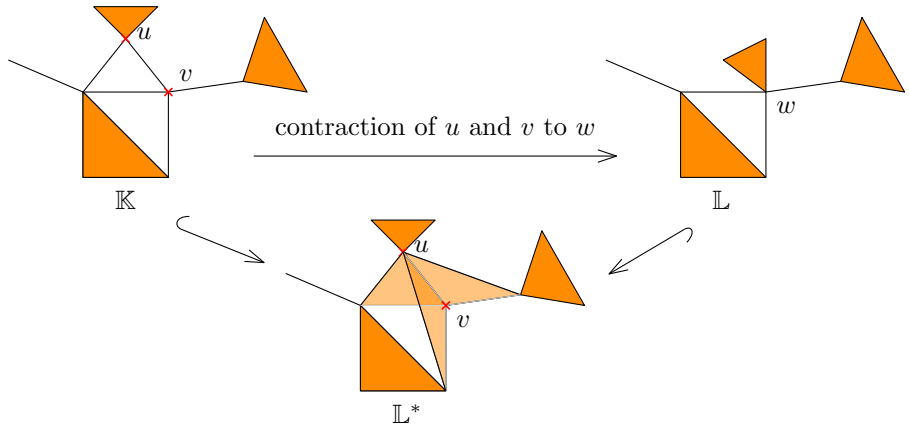


Figure 2.1: Construction example of $\mathbb{L}^*$, were u and v in $\mathbb{K}$ are contracted to w in $\mathbb{L}$

Given a tower $\mathcal{T}$, we can also obtain an non-zigzag filtration using coning, if we continue the operation on $\mathbb{L}^*$ instead of going back to $\mathbb{L}$. More precisely, we set $\tilde{\mathbb{K}}_0 := \mathbb{K}_0$ and if s_i is an inclusion of simplex σ, we set:

$$\tilde{\mathbb{K}}_{i+1} := \tilde{\mathbb{K}}_i \cup \{\sigma\}.$$

If s_i is a contraction $(u, v) \rightsquigarrow u$, we set:

$$\tilde{\mathbb{K}}_{i+1} := \tilde{\mathbb{K}}_i \cup \left(u * \overline{\mathrm{St}}(v, \tilde{\mathbb{K}}_i)\right).$$

Indeed, it can be proved that $(\tilde{\mathbb{K}}_i)_{i=0,\dots,m}$ has the same barcode as $\mathcal{T}$. However, the filtration will not be small, and we will define a smaller variant now.

Our new construction yields a sequence of complexes $\hat{\mathbb{K}}_0, \dots, \hat{\mathbb{K}}_m$ with $\hat{\mathbb{K}}_i \subseteq \hat{\mathbb{K}}_{i+1}$. During the construction, we maintain a flag for each vertex in $\hat{\mathbb{K}}_i$, which marks the vertex as *active* or *inactive*. A simplex is called *active* if all its vertices are active, and *inactive* otherwise. For a vertex u and a complex $\hat{\mathbb{K}}_i$, let $\mathrm{Act}\overline{\mathrm{St}}(u, \hat{\mathbb{K}}_i)$ denote its *active closed star*, which is the set of active simplices in $\hat{\mathbb{K}}_i$ in the closed star of u.

The construction is inductive, starting with $\hat{\mathbb{K}}_0 := \varnothing$. If $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ is an elementary inclusion with $\mathbb{K}_{i+1} = \mathbb{K}_i \cup \{\sigma\}$, set:

$$\hat{\mathbb{K}}_{i+1} := \hat{\mathbb{K}}_i \cup \{\sigma\}.$$

If σ is a vertex, we mark it as active. It remains the case that $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ is an elementary contraction of the vertices u and v. If $|\mathrm{Act}\overline{\mathrm{St}}(u, \hat{\mathbb{K}}_i)| \leq |\mathrm{Act}\overline{\mathrm{St}}(v, \hat{\mathbb{K}}_i)|$, we set

$$\hat{\mathbb{K}}_{i+1} := \hat{\mathbb{K}}_i \cup \left(v * \mathrm{Act}\overline{\mathrm{St}}(u, \hat{\mathbb{K}}_i)\right)$$

and mark u as inactive. Otherwise, we set

$$\hat{\mathbb{K}}_{i+1} := \hat{\mathbb{K}}_i \cup \left(u * \mathrm{Act}\overline{\mathrm{St}}(v, \hat{\mathbb{K}}_i)\right)$$

and mark v as inactive. This ends the description of the construction. We write $\mathcal{F}$ for the filtration $(\hat{\mathbb{K}}_i)_{i=0,\dots,m}$.

There are two major changes compared to the construction of $(\tilde{\mathbb{K}}_i)_{i=0,\dots,m}$. First, to counteract the potentially large growth of the involved cones, we restrict coning to active simplices. We will show below that the subcomplex of $\hat{\mathbb{K}}_i$ induced by the active vertices is isomorphic to $\mathbb{K}_i$. As a consequence, we add the same number of simplices by passing from $\hat{\mathbb{K}}_i$ to $\hat{\mathbb{K}}_{i+1}$ as in the approach by Dey et al. does when passing from $\mathbb{K}$ to $\mathbb{L}^*$.

A second difference is that our strategy exploits that an elementary contraction of two vertices u and v leaves us with a choice: we can either take u or v as the representative of the contracted vertex. In terms of simplicial maps, these two choices correspond to setting $s_i(u) = s_i(v) = u$ or $s_i(u) = s_i(v) = v$, if s_i is the elementary contraction of u and v. It is obvious that both choices yield identical complexes $\mathbb{K}_{i+1}$ up to renaming of vertices. However, the choices make a difference in terms of the size of $\hat{\mathbb{K}}_{i+1}$, because the active closed star of u to v in $\hat{\mathbb{K}}_i$ might differ in size. Our construction simply choose the representative which causes the smaller $\hat{\mathbb{K}}_{i+1}$.

2.3.2 Topological equivalence

We make the following simplifying assumption for $\mathcal{T}$. Let s_i be an elementary contraction of u and v. If our construction of $\hat{\mathbb{K}}_{i+1}$ turns v inactive, we assume that $s_i(u) = s_i(v) = u$. Otherwise, we assume $s_i(u) = s_i(v) = v$. Indeed, this is without loss of generality because it corresponds to a renaming of the simplices in each $\mathbb{K}_i$ and yields equivalent persistence modules. The advantage of this convention is the following property, which follows from a straight-forward inductive argument.

Lemma 2.3. *For every i in $\{0, ..., m\}$, the set of vertices of $\mathbb{K}_i$ is equal to the set of active vertices in $\hat{\mathbb{K}}_i$.*

This allows us to interpret $\mathbb{K}_i$ and $\hat{\mathbb{K}}_i$ as simplicial complexes defined over a common vertex set. In fact, $\mathbb{K}_i$ is the subcomplex of $\hat{\mathbb{K}}_i$ induced by the active vertices:

Lemma 2.4. *A simplex σ is in $\mathbb{K}_i$ if and only if σ is an active simplex in $\hat{\mathbb{K}}_i$.*

Proof. We use induction on i. The statement is true for $i = 0$, because $\mathbb{K}_0 = \varnothing = \hat{\mathbb{K}}_0$.

So assume first $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ is an elementary inclusion that adds a d-simplex $\sigma = (v_0, \dots, v_d)$ to $\mathbb{K}_{i+1}$. If σ is a vertex, it is set active in $\hat{\mathbb{K}}_{i+1}$ by construction. Otherwise, $v_0, \dots, v_d$ are active in $\hat{\mathbb{K}}_i$ by induction and stay active in $\hat{\mathbb{K}}_{i+1}$. In any case, σ is active in $\hat{\mathbb{K}}_{i+1}$. The equivalence for the remaining simplices is straight-forward.

If s_i is an elementary contraction $(u, v) \rightsquigarrow u$, we prove both directions of the equivalence separately. For "$\Rightarrow$", fix a d-simplex $\sigma \in \mathbb{K}_{i+1}$. It suffices to show that $\sigma \in \hat{\mathbb{K}}_{i+1}$, as in this case, it is also active by Lemma 2.3. If $\sigma \in \mathbb{K}_i$, this follows at once by induction because $\mathbb{K}_i \subseteq \hat{\mathbb{K}}_i \subseteq \hat{\mathbb{K}}_{i+1}$. If $\sigma \notin \mathbb{K}_i$, u must be a vertex of σ. Moreover, writing $\sigma = \{u, v_1, ..., v_d\}$ and $\sigma' = \{v, v_1, \dots, v_d\}$ we have that $\sigma' \in \mathbb{K}_i$ and $s_i(\sigma') = \sigma$. In particular, the vertices $v_1, \dots, v_d$ are

active in $\hat{\mathbb{K}}_i$ by induction, hence $\{v_1, \ldots, v_d\}$ is in the active closed star of v in $\hat{\mathbb{K}}_i$. By construction, $\{u, v_1, ..., v_d\} = \sigma$ is in $\hat{\mathbb{K}}_{i+1}$.

For "$\Leftarrow$", let $\sigma \in \hat{\mathbb{K}}_{i+1} \setminus \mathbb{K}_{i+1}$. We show that σ is an inactive simplex in $\hat{\mathbb{K}}_{i+1}$. By Lemma 2.3, this is equivalent to show that σ contains a vertex not in $\mathbb{K}_{i+1}$.

Case 1: $\sigma \in \hat{\mathbb{K}}_i$. If σ is inactive in $\hat{\mathbb{K}}_i$, it stays inactive in $\hat{\mathbb{K}}_{i+1}$. So, assume that σ is active in $\hat{\mathbb{K}}_i$ and thus $\sigma \in \mathbb{K}_i$ by induction. But $\sigma \notin \mathbb{K}_{i+1}$, so σ must have v as a vertex and is therefore inactive in $\hat{\mathbb{K}}_{i+1}$.

Case 2: $\sigma \in \hat{\mathbb{K}}_{i+1} \setminus \hat{\mathbb{K}}_i$. By construction, σ is of the form $\{u, v_1, \ldots, v_d\}$ such that $\{v_1, \ldots, v_d\}$ is in the active closed star of v in $\hat{\mathbb{K}}_i$. Assume for a contradiction that $v \neq v_j$ for all $j = 1, \ldots, d$. Then, $\sigma' = \{v, v_1, \ldots, v_d\}$ is active in $\hat{\mathbb{K}}_i$ and thus, by induction, a simplex in $\mathbb{K}_i$. But then, $s_i(\sigma') = \sigma \in \mathbb{K}_{i+1}$ which is a contradiction to our choice of σ. It follows that v is a vertex of σ which proves our claim. □

Lemma 2.5. *For every $0 \leq i \leq m$, the complex $\hat{\mathbb{K}}_i$ collapses to $\mathbb{K}_i$.*

Proof. We use induction on i. For $i = 0$, $\mathbb{K}_0 = \hat{\mathbb{K}}_0$, and the statement is trivial. Suppose that the statement holds for $\hat{\mathbb{K}}_i$ and $\mathbb{K}_i$ using the sequence c_i of elementary collapses. Note that these collapses only concern inactive simplices in $\hat{\mathbb{K}}_i$. For an inactive vertex $v \in \hat{\mathbb{K}}_i$, the construction of $\hat{\mathbb{K}}_{i+1}$ ensures that v does not gain any additional coface. This implies that c_i is still a sequence of elementary collapses for $\hat{\mathbb{K}}_{i+1}$, yielding a complex $\hat{\mathbb{K}}^*_{i+1}$ with $\mathbb{K}_{i+1} \subseteq \hat{\mathbb{K}}^*_{i+1} \subseteq \hat{\mathbb{K}}_{i+1}$. In particular, $\hat{\mathbb{K}}^*_{i+1}$ only contains vertices that are still active in $\hat{\mathbb{K}}_i$. If s_i is an elementary inclusion, $\hat{\mathbb{K}}^*_{i+1} = \mathbb{K}_{i+1}$, because any all vertices in $\hat{\mathbb{K}}_i$ remain active in $\hat{\mathbb{K}}_{i+1}$. For s_i being an elementary contraction $(u, v) \rightsquigarrow u$, set $S := \hat{\mathbb{K}}^*_{i+1} \setminus \mathbb{K}_{i+1}$ as the remaining set of simplices that still need to be collapsed to obtain $\mathbb{K}_{i+1}$. All simplices of S have v as vertex. More precisely, S is the set of all simplices of the form $\{v, v_1, \ldots, v_d\}$ with $v_1, \ldots, v_d$ active in $\hat{\mathbb{K}}_{i+1}$. We split $S = S_u \cup S_{\neg u}$ where $S_u \subset S$ are the simplices in S that contain u as vertex, and $S_{\neg u} = S \setminus S_u$.

We claim that the mapping that sends $\{u, v, v_1, \ldots, v_d\} \in S_u$ to $\{v, v_1, \ldots, v_d\} \in S_{\neg u}$ is a bijection. This map is clearly injective. If $\sigma = \{v, v_1, \ldots, v_d\} \in S_{\neg u}$, then $\sigma \in \hat{\mathbb{K}}_i$ (because every newly added simplex in $\hat{\mathbb{K}}_{i+1}$ contains u). Also, $\sigma \in \hat{\mathbb{K}}^*_{i+1}$, and is therefore active in $\hat{\mathbb{K}}_i$. By construction, $\{u, v, v_1, \ldots, v_d\} \in \hat{\mathbb{K}}_{i+1}$, proving surjectivity.

We now define a sequence of elementary collapses from $\hat{\mathbb{K}}^*_{i+1}$ to $\mathbb{K}_{i+1}$. Choose a simplex $\sigma = \{v, v_1, \ldots, v_d\} \in S_{\neg u}$ of maximal dimension, and let $\tau = \{u, v, v_1, \ldots, v_d\}$ denote the corresponding simplex in S_u. Then σ is indeed a free face in $\hat{\mathbb{K}}^*_{i+1}$, because if there was another coface $\tau' \neq \tau$, it takes the form $\{w, v, v_1, \ldots, v_d\}$ with $w \neq u$ active. So, $\tau' \in S_{\neg u}$, and τ' has larger dimension than σ, a contradiction. Therefore, the pair (σ, τ) defines an elementary collapse in $\hat{\mathbb{K}}^*_{i+1}$. We proceed with this construction, always collapsing a remaining pair in $S_{\neg u} \times S_u$ of maximal dimension, until all elements of S have been collapsed. □

Proposition 2.6. *$\mathcal{T}$ and $\mathcal{F}$ have the same barcode.*

Proof. Let $\hat{s}_i : \hat{\mathbb{K}}_i \to \hat{\mathbb{K}}_{i+1}$ denote the inclusion map from $\hat{\mathbb{K}}_i$ to $\hat{\mathbb{K}}_{i+1}$. By combining Lemma 2.5 with Lemma 2.1, we have an isomorphism $i_i^* : H(\mathbb{K}_i) \to$

$H(\hat{\mathbb{K}}_i)$, for all $0 \leq i \leq m$, induced by the inclusion maps $i_i : \mathbb{K}_i \to \hat{\mathbb{K}}_i$, and therefore the following diagram connecting the persistence modules induced by $\mathcal{T}$ and $\mathcal{F}$:

$$\begin{array}{ccccccc} H(\mathbb{K}_0) & \xrightarrow{s_0^*} & H(\mathbb{K}_1) & \xrightarrow{s_1^*} & \cdots & \xrightarrow{s_{m-1}^*} & H(\mathbb{K}_m) \\ \downarrow{\scriptstyle i_0^*} & & \downarrow{\scriptstyle i_1^*} & & & & \downarrow{\scriptstyle i_m^*} \\ H(\hat{\mathbb{K}}_0) & \xrightarrow{\hat{s}_0^*} & H(\hat{\mathbb{K}}_1) & \xrightarrow{\hat{s}_1^*} & \cdots & \xrightarrow{\hat{s}_{m-1}^*} & H(\hat{\mathbb{K}}_m) \end{array} . \tag{2.1}$$

The *Persistence Equivalence Theorem* [33, p.159] asserts that $(\mathbb{K}_j)_j$ and $(\hat{\mathbb{K}}_j)_j$, with $j = 0, ..., m$, have the same barcode if (2.1) commutes, that is, if $i_{i+1}^* \circ s_i^* = \hat{s}_i^* \circ i_i^*$, for all $0 \leq i < m$.

Two simplicial maps $s : \mathbb{K} \to \mathbb{K}'$ and $s' : \mathbb{K} \to \mathbb{K}'$ are *contiguous* if, for all $\sigma \in \mathbb{K}$, $s(\sigma) \cup s'(\sigma) \in \mathbb{K}'$. Two contiguous maps are known to be homotopic [51, Theorem 12.5.] and thus equal at homology level, that is, $s^* = s'^*$. We show that $i_{i+1} \circ s_i$ and $\hat{s}_i \circ i_i$ are contiguous. This implies that (2.1) commutes, because, by functoriality, $i_{i+1}^* \circ s_i^* = (i_{i+1} \circ s_i)^* = (\hat{s}_i \circ i_i)^* = \hat{s}_i^* \circ i_i^*$.

To show contiguity, fix $\sigma \in \mathbb{K}_i$ and observe that $(\hat{s}_i \circ i_i)(\sigma) = \sigma$ because $\hat{s}_i$ and i_i are inclusions. If $s_i(\sigma) = \sigma$, $(i_{i+1} \circ s_i)(\sigma) = \sigma$ as well, and the contiguity condition is clearly satisfied. So, let $s_i(\sigma) \neq \sigma$. Then s_i is an elementary contraction $(u, v) \rightsquigarrow u$, and σ is of the form $\{v, v_1, \ldots, v_d\}$, where one of the v_j might be equal to u. Then, $(i_{i+1} \circ s_i)(\sigma) = \{u, v_1, ..., v_d\}$. Consequently, $(i_{i+1} \circ s_i)(\sigma) \cup (\hat{s}_i \circ i_i)(\sigma) = \{u, v, v_1, ..., v_d\}$. By Lemma 2.4, $\sigma = \{v, v_1, \ldots, v_d\}$ is in the active closed star of v in $\hat{\mathbb{K}}_i$, and by construction $\{u, v, v_1, ..., v_d\} \in \hat{\mathbb{K}}_{i+1}$, which proves contiguity of the maps. □

2.3.3 Size analysis

The contracting forest. Let $\mathcal{T}_j$ denote the prefix of length $j \leq m$, of $\mathcal{T}$:

$$\mathcal{T}_j : \ \varnothing = \mathbb{K}_0 \xrightarrow{s_0} \ldots \xrightarrow{s_{j-1}} \mathbb{K}_j$$

We associate a rooted labeled forest F_j to $\mathcal{T}_j$ inductively as follows: For $j = 0$, F_0 is the empty forest. Let F_{j-1} be the forest of $\mathcal{T}_{j-1}$. If s_{j-1} is an elementary inclusion of a d-simplex, we have two cases: if $d > 0$, set $F_j := F_{j-1}$. If a vertex v is included, $F_j := F_{j-1} \cup \{x\}$, with x a single node tree labeled with v. If s_{j-1} is an elementary contraction contracting two vertices u and v in $\mathbb{K}_{j-1}$, there are two trees in F_{j-1}, whose roots are labeled u and v. In F_j, these two trees are merged by making their roots children of a new root, which is labeled with the vertex that u and v are mapped to.

We can read off from the construction immediately that the roots of F_i are labeled with the vertices of the complex $\mathbb{K}_i$, for every $i = 0, \ldots, m$. Moreover, each leaf corresponds to the inclusion of a vertex in $\mathcal{T}_i$, and each internal node corresponds to a contraction of two vertices. In particular, F_i is a *full* forest, that is, every node has 0 or 2 children.

Let $F := F_m$ denote the forest of the tower $\mathcal{T} = \mathcal{T}_m$. Let Σ denote the set of all simplices that are added at elementary inclusions in $\mathcal{T}$, and recall that $n = |\Sigma|$. A d-simplex $\sigma \in \Sigma$ added by s_i is formed by $d + 1$ vertices, which correspond to $d + 1$ roots of F_{i+1}, and equivalently, to $d + 1$ nodes in F. For a node x in F, we denote by $E(x) \subseteq \Sigma$ the subset of simplices with at least one

vertex that appears as label in the subtree of F rooted at x. If y_1 and y_2 are the children of x, $E(y_1)$ and $E(y_2)$ are both subsets of $E(x)$, but not disjoint in general. However, the following relation follows at once:

$$|E(x)| \;\geq\; |E(y_1)| + |E(y_2) \setminus E(y_1)| \tag{2.2}$$

We say that the set N of nodes in F is *independent*, if there are no two nodes $x_1 \neq x_2$ in N, such that x_1 is an ancestor of x_2 in F. A vertex in $\mathbb{K}_i$ appears as label in at most one F-subtree rooted at a vertex in the independent set N. Thus, a d-simplex σ can only appear in up to $d+1$ E-sets of vertices in N. That implies:

Lemma 2.7. *Let N be an independent set of vertices in F. Then,*

$$\sum_{x \in N} |E(x)| \leq (\Delta + 1) \cdot n,$$

The cost of contracting. Recall that a contraction $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ yields an inclusion $\hat{\mathbb{K}}_i \hookrightarrow \hat{\mathbb{K}}_{i+1}$ that potentially adds more than one simplex. Therefore, in order to bound the total size of $\hat{\mathbb{K}}_m$, we need to bound the number of simplices added in all these contractions.

We define the *cost* of a contraction s_i as $|\hat{\mathbb{K}}_{i+1} \setminus \hat{\mathbb{K}}_i|$, that is, the number of simplices added in this step. Since each contraction corresponds to a node x in F, we can associate these costs to the internal nodes in the forest, denoted by $c(x)$. The leaves get cost 0.

Lemma 2.8. *Let x be an internal node of F with children y_1, y_2. Then,*

$$c(x) \leq 2 \cdot |E(y_1) \setminus E(y_2)|.$$

Proof. Let $s_i : \mathbb{K}_i \to \mathbb{K}_{i+1}$ denote the contraction that is represented by the node x, and let w_1 and w_2 be the labels of its children y_1 and y_2, respectively. By construction, w_1 and w_2 are vertices in $\mathbb{K}_i$ that are contracted by s_i. Let $C_1 = \overline{\mathrm{St}}(w_1, \mathbb{K}_i) \setminus \overline{\mathrm{St}}(w_2, \mathbb{K}_i)$ and $C_2 = \overline{\mathrm{St}}(w_2, \mathbb{K}_i) \setminus \overline{\mathrm{St}}(w_1, \mathbb{K}_i)$. By Lemma 2.4, $\overline{\mathrm{St}}(w_1, \mathbb{K}_i) = \mathrm{Act}\overline{\mathrm{St}}(w_1, \hat{\mathbb{K}}_i)$, and the same for w_2. So, because the simplices the two active closed stars have in common will not influence the cost of the contraction, we have,

$$c(x) \leq \min\{|C_1|, |C_2|\}.$$

In particular, $c(x) \leq |C_1|$. We will show that $|C_1| \leq 2 \cdot |E(y_1) \setminus E(y_2)|$.

For every d-simplex $\sigma \in \mathbb{K}_i$, there is some d-simplex $\tau \in \Sigma$ that has been added in an elementary inclusion s_j with $j < i$, such that $s_{i-1} \circ s_{i-2} \circ \ldots \circ s_{j+1}(\tau) = \sigma$. We call τ an *origin* of σ. There might be more than one origin of a simplex, but two distinct simplices in $\mathbb{K}_i$ cannot have a common origin. Moreover, for every vertex v of σ, the tree in F_i whose root is labeled with v contains exactly one vertex v' of τ as label. We omit the proof which works by simple induction.

We prove the inequality by a simple charging argument. For each vertex in C_1, we charge a simplex in $|E(y_1) \setminus E(y_2)|$ such that no simplex is charged more than twice. Note that $C_1 = (\mathrm{St}(w_1, \mathbb{K}_i) \cup \mathrm{Lk}(w_1, \mathbb{K}_i)) \setminus \overline{\mathrm{St}}(w_2, \mathbb{K}_i)$. If $\sigma \in \mathrm{St}(w_1, \mathbb{K}_i)$, then fix an origin τ of σ. Then τ has a vertex that is a label in the subtree rooted at y_1, so $\tau \in E(y_1)$. At the same time, since w_2 is

not a vertex of σ, τ has no vertex in the subtree rooted at y_2, so $\tau \notin E(y_2)$. We charge τ for the existence of σ. Because different simplices have different origins, every element in $E(y_1) \setminus E(y_2)$ is charged at most once for $\mathrm{St}(w_1, \mathbb{K}_i)$. If $\sigma \in \mathrm{Lk}(w_1, \mathbb{K}_i)$, $\sigma' := w_1 * \sigma \in \mathrm{St}(w_1, \mathbb{K}_i)$, and we can choose an origin τ' of σ'. As before, $\tau' \in E(y_1) \setminus E(y_2)$ and we charge τ' for the existence of σ. Again, each element in $E(y_1) \setminus E(y_2)$ is charged at most once among all elements in the link. This proves the claim. □

An *ascending path* $(x_1, ..., x_L)$, with $L \geq 1$, is a path in a forest such that x_{i+1} is the parent of x_i, for $1 \leq i < L$. We call L the *length* of the path and x_L its *endpoint*. For ascending paths in F, the *cost* of the path is the sum of the costs of the nodes. We say that the set P of ascending paths is *independent*, if the endpoints in P are pairwise different and form an independent set of nodes. We define the *cost* of P as the sum of the costs of the paths in P.

Lemma 2.9. *Let $(x_1, ..., x_L)$ be an ascending path and P an independent set of ascending paths in F. Then the cost of both is respectively:*

$$c(x_1, ..., x_L) \leq 2 \cdot |E(x_L)|$$
$$c(P) \leq 2 \cdot (\Delta + 1) \cdot n.$$

Proof. For the first statement, let p be the ascending path $(x_1, ..., x_L)$. Without loss of generality, we can assume the the path starts with a leaf x_1, because otherwise, we can always extend the path to a longer path with at least the same cost. We let $p_i = (x_1, \ldots, x_i)$ denote the subpath ending at x_i, for $i = 1, \ldots, L$, so that $p_L = p$. We let $c(p_i)$ denote the cost of the path p_i and show by induction that $c(p_i) \leq 2 \cdot |E(x_i)|$. For $i = 1$, this follows because $c(p_1) = 0$. For $i = 2, \ldots, L$, x_i is an internal node, and its two children are x_{i-1} and some other node x'_{i-1}. Using induction and Lemma 2.8, we have that

$$\begin{aligned} c(p_i) &= c(p_{i-1}) + c(x_i) \\ &\leq 2 \cdot (|E(x_{i-1})| + |E(x'_{i-1}) \setminus E(x_{i-1})|) \\ &\leq 2 \cdot |E(x_i)|, \end{aligned}$$

where the last inequality follows from (2.2). The second statement follows from Lemma 2.7 because the endpoints of the paths form an independent set in F. □

Ascending path decomposition. An *only-child-path* in a binary tree is an ascending path starting in a leaf and ending at the first encountered node that has a sibling, or at the root of the tree. An only-child-path can have length 1, if the starting leaf has a sibling already. Examples of only-child-paths are shown in Figure 2.2. We observe that no node with two children lies on any only-child-path, which implies that the set of only-child-paths forms an independent set of ascending paths.

Consider the following pruning procedure for a full binary forest F. Set $F_{(0)} \leftarrow F$. In iteration i, we obtain the forest $F_{(i)}$ from $F_{(i-1)}$ by deleting the only-child-paths of $F_{(i-1)}$. We stop when $F_{(i)}$ is empty; this happens eventually because at least the leaves of $F_{(i-1)}$ are deleted in the i^{th} iteration. Because we start with a full forest, the only-child-paths in the first iteration are all of length 1, and consequently $F_{(1)}$ arises from $F_{(0)}$ by deleting the leaves of $F_{(0)}$.

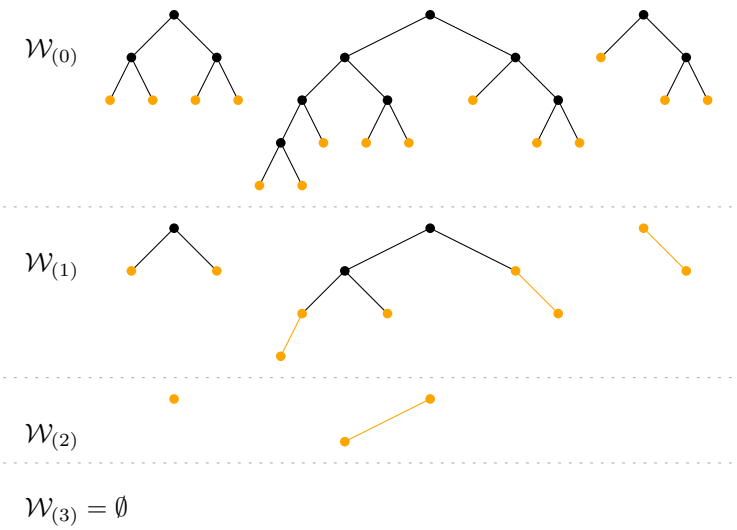


Figure 2.2: Iterations of the pruning procedure; the only-child-paths are marked in color.

Note that the intermediate forests $F_{(1)}, F_{(2)}, \ldots$ are not full forests in general. Figure 2.2 shows the pruning procedure on an example.

To analyze this pruning procedure in detail, we define the following integer valued function for nodes in F:

$$r(x) = \begin{cases} 1, & \text{if } x \text{ is a leaf} \\ r(y_1) + 1, & \text{if } x \text{ has children } y_1, y_2 \text{ and } r(y_1) = r(y_2) \\ \max\{r(y_1), r(y_2)\}, & \text{if } x \text{ has children } y_1, y_2 \text{ and } r(y_1) \neq r(y_2) \end{cases}$$

Lemma 2.10. *A node x of a full binary forest F is deleted in the pruning procedure during the $r(x)$-th iteration.*

Proof. We prove the claim by induction on the tree structure. If x is a leaf, it is removed in the first iteration, and $r(x) = 1$. If x is an internal node with children y_1 and y_2, let $r_1 := r(y_1)$ and $r_2 := r(y_2)$. By induction, y_1 is deleted in the r_1-th iteration and y_2 is deleted in the r_2-th iteration. There are two cases: if $r_1 = r_2$, x still has two children after $r_1 - 1$ iterations. This implies that x does not lie on an only-child- path in the forest $F_{(r_1-1)}$ and is therefore not deleted in the r_1-th iteration. But because its children are deleted, x is a leaf in $F_{(r_1)}$ and therefore deleted in the $(r_1 + 1)$-th iteration. It remains the second case that $r_1 \neq r_2$. Assume without loss of generality that $r_1 > r_2$, so that $r(x) = r_1$. In iteration r_1, y_1 lies on an only-child-path in $F_{(r_1-1)}$. Because $y_2 \notin F_{(r_1-1)}$, y_1 has no sibling in $F_{(r_1-1)}$, so the only-child-path extends to x. Consequently, x is deleted in the r_1-th iteration. □

Lemma 2.11. *For a node x in a full binary forest, let $s(x)$ denote the number of nodes in the subtree rooted at x. Then $s(x) \geq 2^{r(x)} - 1$. In particular, $r(x) \leq \log_2(s(x) + 1)$.*

Proof. We prove the claim by induction on $r(x)$. Note that $r(x) = 1$ if and only if v is a leaf, which implies the statement for $r(x) = 1$. For $r(x) > 1$, it is

sufficient to prove the statement assuming that v has a minimal s-value among all nodes with the same r-value. Since x is not a leaf, it has two children y_1 and y_2. They satisfy $r(y_1) = r(y_2) = r(x) - 1$, because otherwise $r(x) = r(y_1)$ or $r(x) = r(y_2)$, contradicting the minimality of x. By induction hypothesis, we obtain that

$$\begin{aligned} s(x) &= 1 + s(y_1) + s(y_2) \\ &\geq 1 + \left(2^{r(y_1)} - 1\right) + \left(2^{r(y_2)} - 1\right) \\ &= 2^{r(x)} - 1. \end{aligned}$$

□

Proposition 2.12. *Let n_0 be the number of vertices included in $\mathcal{T}$. Then:*

$$\begin{aligned} |\hat{\mathbb{K}}_m| &\leq n + 2 \cdot (\Delta + 1) \cdot n \cdot (1 + \log_2(n_0)) \\ &= O(n \cdot \Delta \cdot \log_2(n_0)). \end{aligned}$$

Proof. Applying the pruning procedure to the contraction forest F of $\mathcal{T}$, we obtain in every iteration a set of independent ascending paths of F, and the cost of this set is bounded by $2 \cdot (\Delta + 1) \cdot n$ by Lemma 2.9. Because F has at most $2 \cdot n_0 - 1$ nodes, any node x satisfies $r(x) \leq \log_2(2 \cdot n_0)$ by Lemma 2.11. It follows that the pruning procedure ends after $1 + \log_2(n_0)$ iterations, so the total cost of the contraction forest is at most $2 \cdot (\Delta + 1) \cdot n \cdot (1 + \log_2(n_0))$. By definition, this cost is equal to the number of simplices added in all contraction steps. Together with the n simplices added in inclusion steps, the bound follows. □

2.3.4 Algorithm

We will make frequent use of the following concept: a *dictionary* is a data structure that stores a set of *items* of the form `(k,v)`, where `k` is called the *key* and `v` is called the *value* of the item. We assume that all keys stored in the dictionary are pairwise different. The dictionary support three operations:

- `insert(k,v)`: adds a new item to the dictionary,
- `delete(k)`: removes the item with key `k` from the dictionary (if it exists),
- `search(k)`: returns the item with key `k`, or returns that no such item exists.

Common realizations are balanced binary search trees [21, §12] and hash tables [21, §11].

Simplicial complexes by dictionaries. The main data structure of our algorithm is a dictionary D that represents a simplicial complex. Every item stored in the dictionary represents a simplex, whose key is the list of its vertices. Every simplex σ itself stores an dictionary CoF_σ. Every item in CoF_σ is a pointer to another item in D, representing a cofacet τ of σ. The key of the item is a vertex identifier (e.g., an integer) for v such that $\tau = v * \sigma$.

How large is D for a simplicial complex $\mathbb{K}$ with n simplices of dimension Δ? Observe that D stores n items, and each key is of size $\leq \Delta + 1$. That yields a size of $O(n\Delta)$ plus the size of all CoF_σ. Since every simplex is the cofacet of

at most Δ simplices, the size of all these inner dictionaries is also bounded by $O(n\Delta)$ (assuming that the size of a dictionary is linear in the number of stored elements).

We can insert and delete simplices efficiently in D using dictionary operations. For instance, to insert a simplex σ given as a list of vertices, we insert a new item in D with the key. Then we search for the Δ facets of σ (using dictionary search in D), and notify each facet τ of σ about the insertion by adding a pointer to σ to CoF_τ, using the vertex $\sigma \setminus \tau$ as the key. The deletion procedure works similarly. Each simplex insertion and deletion requires $O(\Delta)$ dictionary operations. In what follows, it is convenient to assume that dictionary operations have unit costs; we multiply the time complexity with the cost of a dictionary operation at the end to compensate for this simplification.

The conversion algorithm. We assume that the tower $\mathcal{T}$ is given to us as a stream where each element represents a simplicial map s_i in the tower. Specifically, an element starts with a token {`INCLUSION`, `CONTRACTION`} that specifies the type of the map. In the first case, the token is followed by a non-empty list of vertex identifiers specifying the vertices of the simplex to be added. In the second case, the token is followed by two vertex identifiers u and v, specifying a contraction of type $(u, v) \rightsquigarrow u$.

The algorithm outputs a stream of simplices specifying the filtration $\mathcal{F}$. Specifically, while handling the i^{th} input element, it outputs the simplices of $\hat{\mathbb{K}}_{i+1} \setminus \hat{\mathbb{K}}_i$ in increasing order of dimension (to ensure that every prefix is a simplicial complex). For simplicity, we assume that output simplices are also specified by a list of vertices — the algorithm can easily be adapted to return the boundary matrix of the filtration in sparse list representation with the same complexity bounds.

We use an initially empty dictionary D as above, and maintain the invariant that after the i^{th} iteration, D represents the active subcomplex of $\hat{\mathbb{K}}_i$, which is equal to $\mathbb{K}_i$ by Lemma 2.4.

If the algorithm reads an inclusion of a simplex σ from the stream, it simply adds σ to D (maintaining the invariant) and writes σ to the output stream.

If the algorithm reads a contraction of two vertices u and v, from $\mathbb{K}_i$ to $\mathbb{K}_{i+1}$, we let c_i denote the cost of the contraction, that is, $c_i = |\hat{\mathbb{K}}_{i+1} \setminus \hat{\mathbb{K}}_i|$. The first step is to determine which of the vertices has the smaller active closed star in $\hat{\mathbb{K}}_i$, or equivalently, which vertex has the smaller closed star in $\mathbb{K}_i$. The size of the closed star of a vertex v could be computed by a simple graph traversal in D, starting at a vertex v and following the cofacet pointers recursively, counting the number of simplices encountered. However, we want to identify the smaller star with only $O(c_i)$ operations, and the closed star can be much larger. Therefore, we change the traversal in several ways:

First of all, observe that $|\overline{\mathrm{St}}(u)| \leq |\overline{\mathrm{St}}(v)|$ if and only if $|\mathrm{St}(u)| \leq |\mathrm{St}(v)|$ (in $\mathbb{K}_i$). Now define $\mathrm{St}(u, \neg v) := \mathrm{St}(u) \setminus \mathrm{St}(v)$. Then, $|\mathrm{St}(u)| \leq |\mathrm{St}(v)|$ if and only if $|\mathrm{St}(u, \neg v)| \leq |\mathrm{St}(v, \neg u)|$, because we subtracted the intersection of the stars on both sides. Finally, note that $\min\{|\mathrm{St}(u, \neg v)|, |\mathrm{St}(v, \neg u)|\} \leq c_i$, as one can easily verify. Moreover, we can count the size of $\mathrm{St}(u, \neg v)$ by a cofacet traversal from u, ignoring cofacets that contain v (using the key of CoF_*), in $O(|\mathrm{St}(u, \neg v)|)$ time. However, this is still not enough, because counting both sets independently gives a running time of $\max\{|\mathrm{St}(u, \neg v)|, |\mathrm{St}(v, \neg u)|\}$. The last trick is that we count

the sizes of $\mathrm{St}(u, \neg v)$ and $\mathrm{St}(v, \neg u)$ at the same time by a simultaneous graph traversal of both, terminating as soon as one of the traversal stops. The running time is then proportional to $2 \cdot \min\{|\mathrm{St}(u, \neg v)|, |\mathrm{St}(v, \neg u)|\} = O(c_i)$, as required.

Assume w.l.o.g. that $|\overline{\mathrm{St}}(u)| \leq |\overline{\mathrm{St}}(v)|$. Also in time $O(c_i)$, we can obtain $\mathrm{St}(u, \neg v)$. We sort its elements by increasing dimension, which can be done in $O(c_i + \Delta)$ using integer sort. For each simplex $\sigma = \{u, v_1, \ldots, v_k\} \in \mathrm{St}(u, \neg v)$ in order, we check whether $\{v, v_1, \ldots, v_k\}$ is in D. If not, we add it to D and also write it to the output stream. Then, we write $\{u, v, v_1, \ldots, v_k\}$ to the output stream (note that we do not have to check it existence in $\hat{\mathbb{K}}_i$, because it does not by construction, and there is no need to add it to D because of the next step). At the end of the loop, we wrote exactly the simplices in $\mathbb{K}_{i+1} \setminus \mathbb{K}_i$ to the output stream, which proves correctness.

It remains to maintain the invariant on D. Assuming still that $|\overline{\mathrm{St}}(u)| \leq |\overline{\mathrm{St}}(v)|$, u turns inactive in $\hat{\mathbb{K}}_{i+1}$. We simply traverse over all cofaces of u and remove all encountered simplices from D. After this operations, the invariant holds. This ends the description of the algorithm.

Complexity analysis. By applying the operation costs on the above described algorithm, we obtain the following statement. Combined with Propositions 2.6 and 2.12, it completes the proof of Theorem 2.2.

Proposition 2.13. *The algorithm requires $O(\Delta \cdot \omega)$ space and $O(\Delta \cdot |\hat{\mathbb{K}}_m| \cdot C_\omega)$ time, where $\omega = \max_{i=0,\ldots,m} |\mathbb{K}_i|$ and C_ω is the cost of an operation in a dictionary with at most ω elements.*

Proof. The space complexity follows at once from the invariant, because the size of D is at most $O(\Delta|\mathbb{K}_i|)$ during the i^{th} iteration.

For the time bound, we set $S := |\hat{\mathbb{K}}_m|$ for convenience and show that the algorithm finishes in $O(\Delta \cdot S)$ steps, assuming dictionary operations to be of constant cost. We have one simplex insertion per elementary inclusion which requires $O(\Delta)$ operations. Thus, all inclusions are bounded by $O(n\Delta)$, which is subsumed by our bound as $n \leq S$. For the contraction case, we need $O(c_i)$ to identify the smaller star, $O(c_i + \Delta)$ to get a sorted list of $\mathrm{St}(u, \neg v)$ (or vice versa), and $O(\Delta \cdot c_i)$ to add new vertices to D. Moreover, we delete the star of u from D; the cost for that is $O(\Delta \cdot d_i)$, where d_i is the number of deleted simplices. Thus, the complexity of a contraction is $O(\Delta \cdot (c_i + d_i))$.

Since c_i is the number of simplices added to the filtration at step i, the sum of all c_i is bounded by $O(S)$. Moreover, because every simplex that ever appears in D belongs to $\hat{\mathbb{K}}_m$ and every simplex is inserted only once, the sum of all d_i is bounded by $O(S)$ as well. Therefore, the combined cost over all contractions is $O(\Delta \cdot S)$ as required. □

Note that the dictionary D has lists of identifiers of length up to $\Delta + 1$ as keys, so that comparing two keys has cost $O(\Delta)$. Therefore, using balanced binary trees as dictionary, we get a complexity of $C_\omega = O(\Delta \log \omega)$. Using hash tables, we get an expected worst-case complexity of $C_\omega = O(\Delta)$.

Experimental results. The following tests where made on a 64-bit Linux (Ubuntu) HP machine with a 3.50 GHz Intel processor and 63 GB RAM. The programs were all implemented in C++ and compiled with optimization level

-O2. Our algorithm was implemented in the software `Sophia`[5] and will soon appear in a release of `GUDHI` [58].

To test its performance, we compared it to the software `Simpers`[6] (downloaded in August 2017), which is the implementation of the Annotation Algorithm from Dey, Fan and Wang described in [29]. Simpers computes the persistence of the given filtration, so we add to our time the time the library `PHAT` [3] (version 1.5) needs to compute the persistence from the generated filtration. `PHAT` was used with its default parameters and its '--ascii --verbose' options activated. `Simpers` also used its default parameters except for the dimension parameter which was set to 5.

	c	n	n_0	Δ	ω
data1	495	4 833	500	4	2 908
data2	795	7 978	800	4	4 816
data3	794	8 443	800	5	5 155
GPS	1 746	8 585	1 747	3	1 747
KB	22 499	95 019	22 500	3	22 500
MC	23 074	143 928	23 075	3	28 219
S3	252 995	1 473 580	252 996	4	252 996
PC25	14 999	10 246 125	15 000	3	2 191 701

	Sophia + PHAT			Simpers	
	filtration size	time (s)	mem. peak (kB) Sophia / total	time (s)	mem. peak (kB)
data1	19 747	0.07	4 752 / 6 472	0.54	10 030
data2	35 253	0.20	5 286 / 9 259	2.82	19 876
data3	38 101	0.21	5 638 / 9 487	3.88	25 104
GPS	9 063	0.02	4 234 / 5 027	0.07	5 849
KB	133 433	0.30	10 036 / 14 484	0.51	25 392
MC	185 447	0.51	13 730 / 18 792	0.77	26 718
S3	1 824 461	8.50	85 128 / 151 860	10.78	247 956
PC25	12 283 003	135.02	994 400 / 1 439 664	∞	-

Table 2.1: Experimental results. The symbol ∞ means that the calculation time exceeded 12 hours.

The results of the tests are in Table 2.1. The timings for File IO are not included in any process time except the input reading of `Sophia`. The memory peak was obtained via the '/usr/bin/time -v' Linux command. Each command was repeated 10 times and the average was token. The first three towers in the table, `data1-3`, were generated incrementally on a set of n_0 vertices: In each iteration, with 90% probability, a new simplex is included, that is picked uniformly at random among the simplices whose facets are all present in the complex, and with 10% probability, two randomly chosen vertices of the complex

[5] `https://bitbucket.org/schreiberh/sophia/`

[6] `http://web.cse.ohio-state.edu/~tamaldey/SimPers/Simpers.html`

are contracted. This is repeated until the complex on the remaining k vertices forms a $k-1$-simplex, in which case no further simplex can be added. The remaining data was generated from the `SimBa` (downloaded in June 2016) library with default parameters using the point clouds from [30]. To obtain the towers that `SimBa` computes internally, we included a print command at a suitable location in the `SimBa` code.

To verify that the space consumption of our algorithm does not dependent on the length of the tower, we constructed an additional example whose size exceeds our RAM capacity, but whose width is small: we took 10 random points moving on a flat torus in a randomly chosen fixed direction. When two points get in distance less than t_1 to each other, we add the edge between them (the edge remains also if the points increase their distance later on). When two points get in distance less than t_2 from each other with $t_2 < t_1$, we contract the two vertices and let a new moving point appear somewhere on the torus. This process yields a sequence of graphs, and we take its flag complex as our simplicial tower. In this way, we obtain a tower of length about $3.5 \cdot 10^9$ which has a file size of about 73 GB, but only has a width of 367. Our algorithm took about 2 hours to convert this tower into a filtration of size roughly $4.6 \cdot 10^9$. During the conversion, the virtual memory used was constantly around 22 MB and the resident set size about 3.8 MB only, confirming the theoretical prediction that the space consumption is independent of the length of the tower. The information about the memory use was taken from the '/proc/<pid>/stat' system file every 100 000 insertions/contractions during the process.

2.3.5 Tightness and lower bounds

The conversion theorem (Theorem 2.2) yields an upper bound of $O(\Delta \cdot n \log n_0)$ for the size of a filtration equivalent to a given tower. It is natural to ask whether this bound can be improved. In this section, we assume for simplicity that the maximal dimension Δ is a constant. In that case, it is not difficult to show that our size analysis cannot be improved:

Proposition 2.14. *There exist an example of a tower with n simplices and n_0 vertices such that our construction yields a filtration of size $\Omega(n \log n_0)$.*

Proof. Let $p = 2^k$ for some $k \in \mathbb{N}$. Consider a graph with p edges (a_i, b_i), with $a_1, \ldots, a_p$, $b_1, \ldots, b_p$ $2p$ distinct vertices. Our tower first constructs this graph with inclusions (in an arbitrary order). Then, the a-vertices are contracted in a way such that the contracting forest is a fully balanced binary tree — see Figure 2.3 for an illustration.

To bound the costs, it suffices to count the number of edges added between a-vertices and b-vertices in each step. We call them ab-edges from now on. Define the *level* of a contraction to be its level in the contracting tree, where 0 is the level of the leaves, and k is the level of the root. On level 1, a contraction of a_i and a_j yields exactly one new ab-edge, either (a_i, b_j) or (a_j, b_i). The resulting contracted vertex has two incident ab-edges. A contraction on level 2 yields two novel ab-edges, and a vertex with four incident ab-edges. By a simple induction, we observe that a contraction on level i introduces 2^{i-1} new ab-edges, and hence has a cost of at least 2^{i-1}, for $i = 1, \ldots, k$. This means that the sum of the costs of all level i contractions is exactly $\frac{p}{2}$. Summing up over all i yields a cost of at least $k\frac{p}{2}$. The result follows because $k = \log(\frac{n_0}{2})$ and $p = \frac{n}{3}$. □

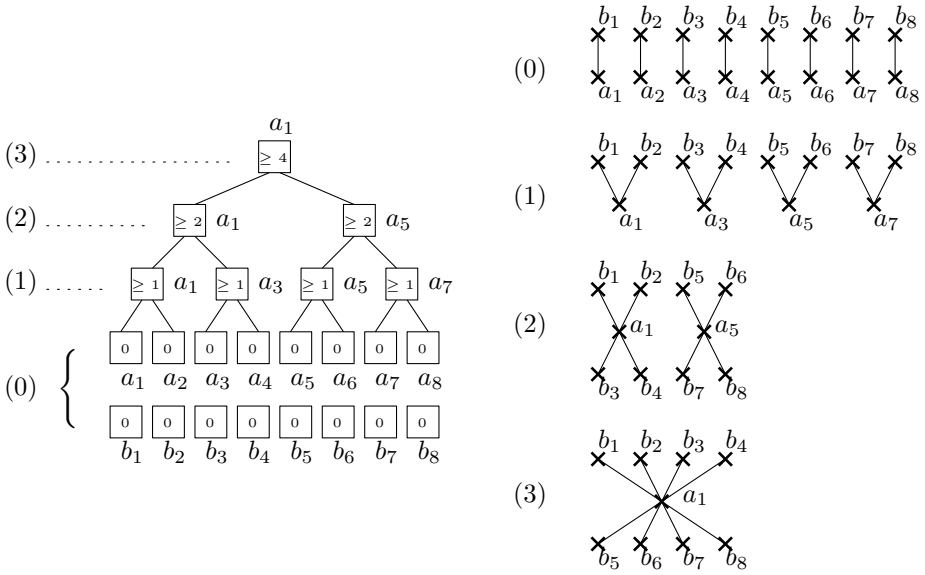


Figure 2.3: [In proof of Proposition 2.14] Sequence of contractions in the described construction for $p = 8$ (right) and the corresponding contracting forest (left), whose nodes contains the cost of the contractions

Another question is whether a different approach could convert a tower into a filtration with an (asymptotically) smaller number of simplices. For this question, consider the *inverse persistence computation problem*: given a barcode, find a filtration of minimal size which realizes this barcode. Note that solving this problem results in a simple solution for the conversion problem: compute the barcode of the tower first using an arbitrary algorithm; then compute a filtration realizing this barcode. While useful for lower bound constructions, we emphasize the impracticality of this solution, as the main purpose of the conversion is a faster computation of the barcode.

Let b be the number of bars of a barcode. An obvious lower bound for the filtration size is $\Omega(b)$, because adding a simplex causes either the birth or the death of exactly one bar in the barcode. For constant dimension, $O(b)$ is also an upper bound:

Lemma 2.15. *For a barcode with b bars and maximal dimension Δ, there exists a filtration of size $\leq 2^{\Delta+2}b$ realizing this barcode.*

Proof. Begin with an empty complex. Now consider the birth and death times represented by the barcode one by one. The first birth will be the one of a 0-dimensional class, so add a vertex v_0 to the complex. From now, every time a 0-dimensional homology class is born add a new vertex to the complex. When a 0-dimensional homology class dies, link the corresponding vertex to v_0 with an edge.

When a k-dimensional homology class is born, with $k > 0$, add the boundary of a $(k+1)$-simplex to the complex that is incident to v_0 and to k novel vertices. When this homology class dies, add the corresponding $(k + 1)$-simplex. This

way, the resulting filtration realizes the barcode. For a bar of the barcode in dimension k, we have to add all proper faces of a $(k+1)$-simplex (except for one vertex). Since that number is at most $2^{k+2}-3$, the result follows. □

If a tower has length m, what is the maximal size of its barcode? If the size of the barcode is $O(m)$, the preceding lemma implies that a conversion to a filtration of linear size is possible (for constant dimension). On the other hand, any example of a tower yielding a super-linear lower bound would imply a lower bound for any conversion algorithm, because a filtration has to contain at least one simplex per bar in its barcode.

To approach the question, observe that a single contraction might destroy many homology classes at once: consider a "fan" of t empty triangles, all glued together along a common edge ab (see Figure 2.4 (a)). Clearly, the complex has t generators in 1-homology. When a and b are contracted, the complex transforms to a star-shaped graph which is contractible. Moreover, a contraction can also create many homology classes at once: consider a collection of t disjoint triangulated spheres, all glued together along a common edge ab. For every sphere, remove one of the triangles incident to ab (see Figure 2.4 (b)). The resulting complex is acyclic. The contraction of the edge ab "closes" each of the spheres, and the resulting complex has t generators in 2-homology. Finally, a contraction might not not affect the homology at all — see Figure 2.4 (c).

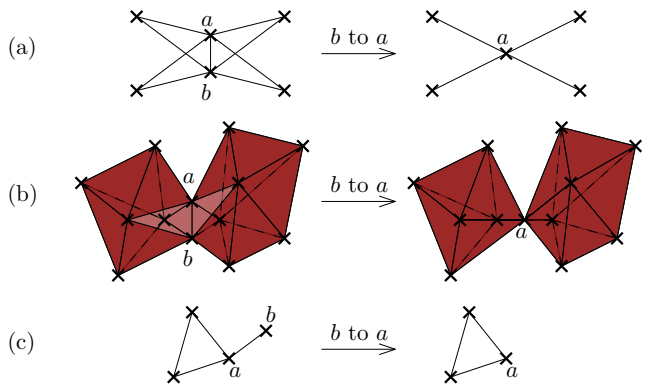


Figure 2.4: Examples of the influence of contractions on the homology classes: (a) destruction of four 1-homology generators, (b) creation of two 2-homology generators, and (c) no influence at all

The above examples show that a single contraction can create and destroy many bars. For a super-linear bound on the barcode size, however, we would have to construct an example where sufficiently many contractions create a large number of bars. So far, we did neither succeed in constructing such an example, nor are we able to show that such an example does not exist.

2.4 Persistence by streaming

Even if the complex we need to maintain in memory during the conversion is relatively small, at the end, the algorithm to compute the final persistence still needs to memorize the whole filtration we give it as input. If the complex in

the original tower has an interesting maximum size during the whole process, we should be able to compute its persistence even if the tower is extremely long. So we design here a streaming variation of the reduction algorithm that computes the barcode of filtrations, such that it has an efficient memory use. More precisely, we will prove the following theorem:

Theorem 2.16. *With the same notation as in Theorem 2.16, we can compute the barcode of a tower $\mathcal{T}$ in worst-case time $O(\omega^2 \cdot \Delta \cdot n \cdot \log n_0)$ and space complexity $O(\omega^2)$.*

We describe the algorithm in Section 2.4.1 and prove the complexity bounds in Section 2.4.2. The described algorithm requires various adaptations to become efficient in practice, and we describe an improved variant in Section 2.4.3. We present some experiments in Section 2.4.4.

2.4.1 Algorithmic description

On a high level, our algorithm converts the tower into an equivalent filtration and computes the barcode of that filtration. We focus on the second part, which we describe as a streaming algorithm. The input to the algorithm is a stream of elements, each starting with a token {`ADDITION`, `INACTIVE`} followed by a simplex identifier which represents a simplex σ. If the token is `ADDITION`, this is followed by a list of simplex identifiers specifying the facets of σ. In other words, the element encodes the next column of the boundary matrix. If the token is `INACTIVE`, it means that σ has become inactive in the complex. In particular, no subsequent simplex in the stream will contain σ as a facet. It is not difficult to modify the algorithm from Section 2.3.4 to return a stream as required, within the same complexity bounds.

The algorithm uses a matrix data type M as its main data structure. We realize M as a dictionary of columns, indexed by a simplex identifier. Each column is a sorted linked list of identifiers corresponding to the non-zero row indices of the column. In particular, we can access the pivot of the column in constant time and we can add two columns in time proportional to the maximal size of the involved lists. Note that most algorithms store the boundary matrix as an array of columns, but we use dictionaries for space efficiency.

There are two secondary data structures that we mention briefly: given a row index r, we have to identify the column index c of the column that has r as pivot in the matrix (or to find out that no such column exists). This can be done using a dictionary with key and value type both simplex identifiers. Finally, we maintain a dictionary representing the set of simplex identifiers that represent active simplices of the complex, plus a flag denoting whether the corresponding simplex is positive or negative. It is straight-forward to maintain these structures during the algorithm, and we will omit the description of the required operations.

The algorithm uses two subroutines. The first one, called `reduce_column`, takes a column identifier j as input is defined as follows: iterate through the non-zero row indices of j. If an index i is the index of an inactive and negative column in M, remove the entry from the column j (this is the "compression" described at the end of Section 2.2). After this pre-processing, reduce the column: while the column is non-empty, and its pivot i is the pivot of another column $k < j$ in the matrix, add column k to column j.

The second subroutine, `remove_row`, takes a index ℓ as input and clears out all entries in row ℓ from the matrix. For that, let j be the column with ℓ as pivot. Traverse all non-zero columns of the matrix except column j. If a column $i \neq j$ has a non-zero entry at row ℓ, add column j to column i. After traversing all columns, remove column j from M.

The main algorithm can be described easily now: if the input stream contains an addition of a simplex, we add the column to M and call `reduce_column` on it. If at the end of that routine, the column is empty, it is removed from M. If the column is not empty and has pivot ℓ, we report (ℓ, j) as a persistence pair and check whether ℓ is active. If not, we call `remove_row(`ℓ`)`. If the input stream specifies that simplex ℓ becomes inactive, we check whether j appears as pivot in the matrix and call `remove_row(`ℓ`)` in this case.

Proposition 2.17. *The algorithm computes the correct barcode.*

Proof. First, note that removing a column from M within the procedure `remove_row` does not affect further reduction steps: Since the pivot ℓ of the column is inactive, no subsequent column in the stream will have an entry in row ℓ. Moreover, the reduction process cannot introduce an entry in row ℓ because the routine has removed all such entries.

Note that `remove_row` might also include right-to-left column additions, and we also have to argue that they do not change the pivots. Let R denote the matrix obtained by the standard compression algorithm that does not call `remove_row` (as described in Section 2.2). Just before our algorithm calls `reduce_column` on the j^{th} column, let M_{j-1} denote the matrix with $(j-1)$ columns that is represented by M. It is straight-forward to verify by an inductive argument that every column of M_{j-1} is a linear combination of $R_1, \ldots, R_{j-1}$, where R_i is the i^{th} column in R. `reduce_column` adds a subset of the columns of M_{j-1} to the j^{th} column. Thus, the reduced column can be expressed by a sequence of left-to-right column additions in R, and thus yields the same pivot as the standard compression algorithm. □

2.4.2 Complexity analysis

We analyze how large the structure M can become during the algorithm. After every iteration, the matrix represents the reduced boundary matrix of some intermediate complex $\hat{\mathbb{L}}$ with $\hat{\mathbb{K}}_i \subseteq \hat{\mathbb{L}} \subseteq \hat{\mathbb{K}}_{i+1}$ for some $i = 0, \ldots, m$. Moreover, the active simplices define a subcomplex $\mathbb{L} \subseteq \hat{\mathbb{L}}$ and there is a moment during the algorithm where $\hat{\mathbb{L}} = \hat{\mathbb{K}}_i$ and $\mathbb{L} = \mathbb{K}_i$, for every $i = 0, \ldots, m$. We call this the i^{th} *checkpoint*. We will make frequent use of the following simple observation.

Lemma 2.18. $|\hat{\mathbb{K}}_{i+1} \setminus \hat{\mathbb{K}}_i| \leq |\mathbb{K}_i| \leq \omega$.

Lemma 2.19. *At every moment, the number of columns stored in M is at most 2ω.*

Proof. It can be verified easily that, throughout the algorithm, a column is stored in M only if not zero, and its pivot is active. So, assume first that we are at the i^{th} checkpoint for some i. Since $\mathbb{L} = \mathbb{K}_i$, there are not more than $|\mathbb{K}_i| \leq \omega$ active simplices. Since each column has a different active pivot, their number is also bounded by ω. If we are between checkpoint i and $i+1$, there have been not more than ω columns added to M since the i^{th} checkpoint from Lemma 2.18. The bound follows. □

The number of rows is more difficult to bound because we cannot guarantee that each column in M corresponds to an active simplex. Still, the number of rows is asymptotically the same as for columns:

Lemma 2.20. *At every moment, the number of rows stored in M is at most 4ω.*

Proof. Consider a row index ℓ and a time in the algorithm where M represents $\hat{\mathbb{L}}$. By the same argument as in the previous lemma, we can argue that there are at most 2ω active row indices at any time. Therefore, we restrict our attention to the case that ℓ is inactive and distinguish three cases. If ℓ represents a negative simplex, we observe that its row should have been was removed due to the compression optimization, and after ℓ became inactive, no simplex can have it as a facet either. It follows that row ℓ is empty in this case. If ℓ is positive and was paired with another index j during the algorithm, then `remove_row` was called on ℓ, either at the moment the pair was formed, or when ℓ became inactive. Since the procedure removes the row, we can conclude that row ℓ is empty also in this case. The final case is that ℓ is positive, but has not been paired so far. It is well-known that in this case, ℓ is the generator of an homology class of $\hat{\mathbb{L}}$. Let

$$\beta(\hat{\mathbb{L}}) := \sum_{i=0}^{\Delta} \beta_i(\hat{\mathbb{L}})$$

denote the sum of the Betti numbers of the complex. Then, it follows that the number of such row indices is at most $\beta(\hat{\mathbb{L}})$.

We argue that $\beta(\hat{\mathbb{L}}) \leq 2\omega$ which proves our claim. Assume that $\hat{\mathbb{K}}_i \subseteq \hat{\mathbb{L}} \subset \hat{\mathbb{K}}_{i+1}$. We have that $\beta(\hat{\mathbb{K}}_i) = \beta(\mathbb{K}_i)$ by Lemma 2.5, and since $\mathbb{K}_i$ has at most ω simplices, $\beta(\mathbb{K}_i) \leq \omega$. Since we add at most ω simplices to get from $\hat{\mathbb{K}}_i$ to $\hat{\mathbb{L}}$, and each addition can increase β by at most one, we have indeed that $\beta(\hat{\mathbb{L}}) \leq 2\omega$. □

Proposition 2.21. *The algorithm has time complexity $O(\omega^2 \cdot \Delta \cdot n \cdot \log n_0)$ and space complexity $O(\omega^2)$.*

Proof. The space complexity is immediately clear from the preceding two lemmas, as M is the dominant data structure in terms of space consumption. For the time complexity, we observe that both subroutines `reduce_column` and `remove_row` need $O(\omega)$ column additions and $O(\omega)$ dictionary operations in the worst case. A column addition costs $O(\omega)$, and a dictionary operation is not more expensive (since the dictionaries contain at most $O(\omega)$ elements and their keys are integers). So, the complexity of both methods is $O(\omega^2)$. Since each routine is called at most once per input element, and there are $O(\Delta \cdot n \cdot \log n_0)$ elements by Theorem 2.2, the bound follows. □

2.4.3 Implementation

The algorithm described in Section 2.4.1 is not efficient in practice for three reasons: First, it has been observed as a general rule that the *clearing optimization* [3] significantly improves the standard algorithm. However, in the above form, that optimization is not usable because it requires to process the columns

in a non-incremental way. Second, the `remove_row` routine scans the entire matrix M; while not affecting the worst-case complexity, frequently scanning the matrix should be avoided in practice. Finally, the above algorithm uses lists to represent columns, but it has been observed that this is a rather inefficient way to perform matrix operations [3].

We outline a variant of the above algorithm that partially overcomes these drawbacks and behaves better in practice. In particular, our variant can be implemented with all column representations available in the `PHAT` library. The idea is to perform a "batch" variant of the previous algorithm: We define a *chunk size* C and read in C elements from the stream; we insert added columns in the matrix, removing row entries of already known inactive negative columns as before, but not reducing the columns yet. After having read C elements, we start the reduction of the newly inserted columns using the clearing optimization. That is, we go in decreasing dimension and remove a column as soon as its index becomes the pivot of another column; see [16] for details. After the reduction ends, except for the last chunk, we go over the columns of the matrix and check for each pivot whether it is active. If it is, we traverse its row entries in decreasing order, but skipping the pivot. Let ℓ be the current entry. If ℓ is the inactive pivot of some column j, we add j to the current column. If ℓ is inactive and represents a negative column, we delete ℓ from the current column. After performing these steps for all remaining columns of the matrix, we go over all columns again, deleting every column with inactive pivot. (By the way, also cleaning up the secondary data structures described in 2.4.1.)

It remains the question of how to choose the parameter C. The chunk provides a trade-off between time and space efficiency. Roughly speaking, the matrix can have up to $O(\omega + C)$ columns during this reduction, but the larger the chunks are, the more benefit one can draw from the clearing optimization (the clearing optimization fails for pairs where the simplices are in different chunks). We therefore recommend to choose C rather large, but making sure that the matrix will still fit into memory.

2.4.4 Experimental evaluation

The tests were made with the same setup as in Section 2.3.4. Figure 2.5 shows the effect of the chunk size parameter C on the runtime and memory consumption of the algorithm. The data used is `S3` (see Section 2.3.4); we also performed the tests on the other examples from Table 2.1, with similar outcome. The File IO operations are included in the measurements. Confirming the theory, as the chunk size decreases, our implementation needs less space but more computation time (while the running time seems to increase slightly again for larger chunk sizes).

For the $4.6 \cdot 10^9$ inclusions tower from Section 2.3.4, with $C = 200\,000$, the algorithm took around 4.5 hours, the virtual memory used was constantly around 68 MB and the resident set size constantly around 49 MB, confirming the theoretical statement that the memory size does not depend on the length of the filtration.

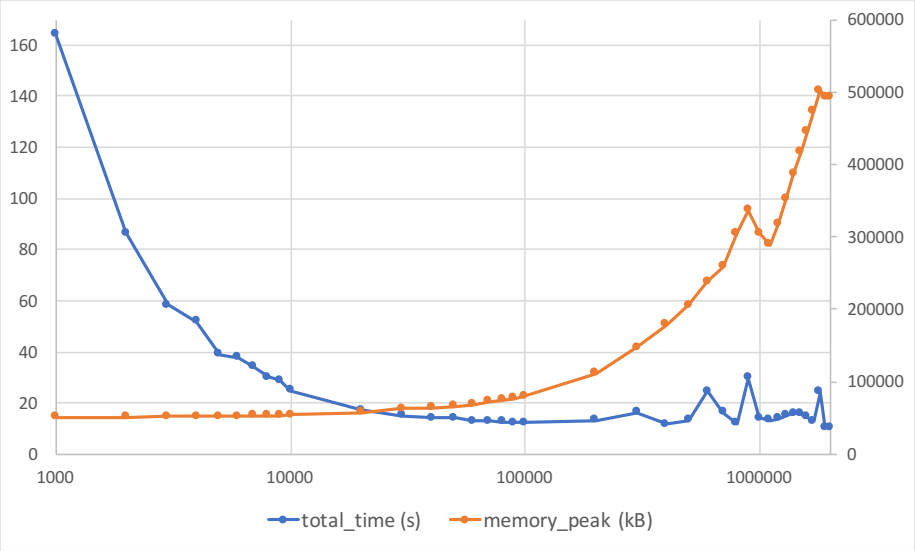


Figure 2.5: Evolution of processing time (left Y-axis in sec) and process memory peak (right Y-axis in kB) depending on the chunk size (logarithmic X-axis)

2.5 Conclusion

In the first part of the paper, we have presented an efficient algorithm to reduce the computation of the barcode of a simplicial tower to the computation of a barcode of a filtration with slightly larger size. With our approach, every algorithmic improvement for persistence computation on the filtration case becomes immediately applicable to the case of towers as well. In the second part of the paper, we present a streaming variant of the classical persistence algorithm for the case of towers. In here, the extra information provided by the towers allows a more space efficient storage of the boundary matrix.

There are various theoretical and practical questions remaining for further work: as already exposed in Section 2.3.5, the question of how large can the barcode of a tower become has immediate consequences on the conversion from towers to filtrations. Moreover, while we focused on constant dimension in Section 2.3.5, we cannot exclude the possibility that our algorithm achieves a better asymptotic bound for non-constant dimensions.

We made our software publicly available in the `Sophia` library. There are several open questions regarding practical performance. For instance, our complex representation, based on hash table, could be replaced with other variants, such as the Simplex tree [6]. Moreover, it would be interesting to compare our streaming approach for the barcode computation with a version that converts to a filtration and subsequently computes the barcode with the annotation algorithm. The reason is that the latter algorithm only maintains a cohomology basis of the currently active complex in memory and therefore avoids the storage of the entire boundary matrix.

Since both the simplex tree and annotation algorithm are part of the `GUDHI` library [58], we plan to integrate our conversion algorithm in an upcoming ver-

sion of the GUDHI library, both to increase the usability of our software and to facilitate the aforementioned comparisons.

Chapter 3

Zigzag Morse Filtrations

This chapter is based on the paper *Discrete Morse Theory for Computing Zigzag Persistence* [47] which is joint work with Clément Maria[1].

3.1 Introduction

As in Chapter 2, this chapter aims to optimize the computation of barcodes, but this time for zigzag filtrations.

Problem statement and related work. The theory of zigzag persistence was introduced in [9], and theoretical [48] and practical [10, 45] algorithms have been introduced to compute it. Zigzag persistence has great applicative potential, considering it provably produces better topological information in topology inference [54], while maintaining the homology of smaller complexes thanks to deletions of faces, and more generally allows a finer approach to data analysis, such as density estimation and topological bootstrapping [9].

However, while computing the barcode of filtrations improved drastically in the last years — as much theoretically [17, 20, 24, 48] as experimentally [1, 2, 5, 16] — computing zigzag persistence remains tedious. It is more intricate that computing standard persistent homology, essentially due to the fact that the full sequence of insertions and deletions of faces is unknown, which requires the maintenance and update of heavier data structures. As a consequence, none of the optimizations of persistence algorithms adapt to the zigzag case. The relatively poor performance of zigzag persistence implementations, compared with persistent homology ones, is a major hindrance to its practical use.

But there is another approach to fast computation: it consists of preprocessing the input filtration in order to drastically reduce the size of the complexes, while preserving the interval decomposition of the persistence module [7, 31, 49, 55]. This approach has the double advantage of reducing both time and memory complexity. This goal has successfully been reached for standard filtrations by the use of *discrete Morse theory* [31, 37, 49] (see also [22, 40]), and led to the implementation of the efficient software, such as `Perseus` [52] and `Diamorse` [25]. Additionally, noticeable successes, at the crossroad of persistence and discrete Morse theory, have been reached in the study of 3D images [55], allowing drastic

[1]INRIA Sophia Antipolis-Méditerranée, France – `clement.maria@inria.fr`

improvements in memory and time performance, as well as the study of data ranging from medical imaging to material science [26, 27, 38]. Therefore, we will now adapt the theory used for standard filtrations for zigzag filtrations.

Motivation and applications for zigzag persistence. We give two important applications of zigzag persistence on which we test the experimental performance of our method.

1. *Topology inference from data points P.* A standard approach [33] consists of computing the persistent homology of the Rips complex $\mathcal{R}^\rho(P)$ on the set of points P, for an increasing threshold $\rho \geq 0$. We compute instead the zigzag persistence of oscillating Rips zigzag filtrations [54]. These filtrations add data points progressively while reducing the scale of reconstruction in order to adapt to a more and more dense set of points. Specifically,

$$\leftarrow\cdots- \mathcal{R}^{\mu\varepsilon_i}(P_i) \hookrightarrow \mathcal{R}^{\nu\varepsilon_i}(P_i \cup \{p_{i+1}\}) \hookleftarrow \mathcal{R}^{\mu\varepsilon_i}(P_i \cup \{p_{i+1}\}) -\cdots\rightarrow \ , \quad (3.1)$$

where $\mathcal{R}^\alpha(P)$ is the Rips complex of threshold α on points P, and ε_i a measure of the "sparsity" of the set of points $P_i := \{p_1, \ldots, p_i\}$ that decreases when points are added. Finally, $0 < \mu \leq \nu$ are parameters. This filtration is known to furnish provably correct persistence diagrams, with much less noise than standard persistence [54], while naturally maintaining much smaller complexes during computation. This application is of importance in data analysis [13, 15].

2. *Levelset persistence of images.* Given a function $f\colon X \to \mathbb{R}$ on a domain X, classical persistence studies the persistent homology of sublevel sets $f^{-1}(-\infty, \rho]$ for an increasing ρ. Levelset persistence [10] studies instead the zigzag persistence of of the pre-images of intervals, for appropriate $s_1 \leq s_2 \leq \ldots$,

$$\leftarrow\cdots- f^{-1}[s_{i-1}, s_i] \hookrightarrow f^{-1}[s_{i-1}, s_{i+1}] \hookleftarrow f^{-1}[s_i, s_{i+1}] -\cdots\rightarrow \ . \quad (3.2)$$

From the levelset persistence, one can recover the sublevel set persistence [10], while maintaining again much smaller structures. This application is of particular importance for medical imaging and material science [26, 27, 38].

Streaming model and memory efficiency. A main advantage of zigzag persistence is to consequently maintain much smaller complexes over the computation. To formalize this notion, we adopt a streaming model for the computation of zigzag persistence. The input is given by a stream of insertions and deletions of faces, with no knowledge of the entire zigzag filtration, and zigzag persistence is computed "on the fly". In particular, the memory complexity of our algorithms depends solely on the maximal size of any complex in the filtration as opposed to the entire number of insertions and deletions of faces, which is generally much larger.

Contributions and existing results. In the spirit of [49], we introduce a preprocessing reduction of a zigzag filtration based on discrete Morse theory [37]. After introducing some background in Section 3.2, we introduce in Section 3.3 a *zigzag Morse filtration* that generalizes the filtered Morse complex [49] of standard persistence, and we prove that it has same persistent homology as the input zigzag filtration. Because of removal of cells not agreeing with the Morse

decomposition, the zigzag Morse filtration contains chain maps that are not inclusions. We study the effect of those maps on the boundary operator of the Morse complex in Section 3.4, and design a persistence algorithm for zigzag Morse complexes in Section 3.5. Finally, we report on the experimental performance of the zigzag persistence algorithm for Morse complexes in Section 3.6.

Note that a similar approach to adapt discrete Morse theory to zigzag persistence was followed by Escolar and Hiraoka [36]. Adapting [49], they define a *global* zigzag filtered Morse complex for a zigzag filtration, and study its interval decomposition. The main limitation of their approach is that the user must know the entirety of the input zigzag filtration to compute the Morse pairing, canceling the benefit of using "small complexes" in zigzag persistence. On the contrary, our approach requires no other than local knowledge of the input zigzag filtration, and all computation are done "on the fly" in the streaming model.

3.2 Background

Complexes and chain maps. In practice, it is common to work with specific complexes, such as simplicial or cubical complexes. However, Morse reductions (introduced below) produce general complexes, which forces us to work in the general setting as we defined in Section 1.3, Paragraph "Cell Complexes".

For a (general) complex X, we denote by $\langle \cdot, \cdot \rangle : C(X) \times C(X) \to \mathbb{Z}$ the inner product on $C(X)$ making the canonical basis of cells $\{\sigma\}_{\sigma \in X}$ orthonormal. In particular, if τ is in the boundary of σ, $\langle \partial\sigma, \tau \rangle = [\sigma : \tau]^X$ in (X, ∂). For a chain $c \in C(X)$, we say that c *contains* a cell σ, and write $\sigma \in c$, if the coefficient of σ is non-zero in c.

Definition 3.1. *Let X and X' be two complexes; X is included in X' if $X \subseteq X'$ as sets of cells, and $[\cdot : \cdot]^{X'}\Big|_X = [\cdot : \cdot]^X$. We also denote the inclusion of complexes by $X \subseteq X'$.*

Finally, a *chain map* $\phi \colon C(X) \to C(X')$ is a map that commutes with the boundary operators of X and X'. It induces a morphism $\phi^* \colon H(X) \to H(X')$ of homology groups.

Notations 3.1. *Let X, X', Y, Y' be complexes, such that $X \subseteq X'$ and $Y \subseteq Y'$, and let $\phi \colon C(X) \to C(Y)$ and $\phi' \colon C(X') \to C(Y')$ be chain maps. If the following diagram commutes, we allow ourselves to use the same notation ϕ for both ϕ and ϕ', when there is no ambiguity on their domain and codomain:*

$$\begin{array}{ccc} C(X) & \hookrightarrow & C(X') \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\phi'} \\ C(Y) & \hookrightarrow & C(Y') \end{array} .$$

Notations 3.2. *By a small abuse of notations, when two complexes X and $X \cup \{\sigma\}$ differ by a single cell σ, we use the notation $X \overset{\sigma}{\hookrightarrow} X \cup \{\sigma\}$ to name the chain map induced by the inclusion. When they differ by a set of cells Σ, we use the notation $X \overset{\Sigma}{\hookrightarrow} X \cup \Sigma$.*

Discrete Morse theory. We refer the reader to [37] for an extended introduction to discrete Morse theory, and to [49] for its application in persistent homology. We follow the general presentation of [49].

The incidence function of a complex induces a *face partial ordering* $<$ on X by taking the transitive closure of the relation $\prec$ defined by

$$\tau \prec \sigma \quad \text{iff} \quad [\sigma : \tau]^X \neq 0.$$

A *partial matching* of X is a partition $X = \mathcal{A} \sqcup \mathcal{Q} \sqcup \mathcal{K}$ of the cells of the complex, together with a bijective pairing $\mathcal{Q} \leftrightarrow \mathcal{K}$, such that if $(\tau, \sigma) \in \mathcal{Q} \times \mathcal{K}$ are paired, then $\dim \sigma = \dim \tau + 1$, and $[\sigma : \tau]^X \neq 0$ is a unit in the PID $\mathbf{D}$ (e.g., 1 or -1 if $\mathbf{D} = \mathbb{Z}$). We call such pair of cells a *Morse pair*. We denote the bijection $\omega \colon \mathcal{Q} \to \mathcal{K}$, such that Morse pairs are of the form $(\tau, \omega(\tau))$.

Call $\mathcal{H}$ the *oriented Hasse diagram* of $(X, <)$: each vertex represents a cell of X and there is an edge between two vertices representing respectively the cells σ and τ if and only if $\tau \prec \sigma$. The edges are oriented from the higher dimensional cell to the lower dimensional cell, i.e., if $\tau \prec \sigma$, then the arrow goes from σ to τ, except for the arrows between cells of Morse pairs $(\tau, \sigma) \in \mathcal{Q} \times \mathcal{K}$, oriented in the other direction.

A *Morse matching* of a complex X is a partial matching that induces an *acyclic* oriented Hasse diagram $\mathcal{H}$ for X. We denote a Morse matching with a partition $\mathcal{A} \sqcup \mathcal{Q} \sqcup \mathcal{K}$ and pairing $\omega \colon \mathcal{Q} \to \mathcal{K}$ by $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$. Note that a Morse matching can also be defined on a subset Σ of cells of a complex X. By convention, we denote $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$ Morse matchings for a *complex*, and $(\hat{\mathcal{A}}, \hat{\mathcal{Q}}, \hat{\mathcal{K}}, \hat{\omega})$ Morse matchings for a *set of faces* not forming a complex.

In a complex with a Morse matching, a *gradient path* between a $(d+1)$-dimensional cell ν and a d-dimensional cell μ is a simple directed path in $\mathcal{H}$ from ν to μ alternating between d and $(d+1)$-dimensional cells[2]. Every gradient path γ is consequently simple and of the form:

$$\gamma = \nu \searrow_{\tau_1} \nearrow^{\omega(\tau_1)} \searrow_{\tau_2} \nearrow^{\omega(\tau_2)} \cdots \searrow_{\tau_r} \nearrow^{\omega(\tau_r)} \searrow_{\mu} \begin{matrix} \dim d+1 \\ \dim d. \end{matrix} \tag{3.3}$$

We denote by $\Gamma(\nu, \mu)$ the set of all distinct gradient paths from ν to μ, and we define for every path γ (with the notations of Diagram (3.3)) its *multiplicity* $m(\gamma)$:

$$m(\gamma) := [\nu : \tau_1]^X \cdot (-1)^r \cdot \prod_{i=1}^{r} \left([\omega(\tau_i) : \tau_i]^X \right)^{-1} \cdot \prod_{i=1}^{r-1} [\omega(\tau_i) : \tau_{i+1}]^X \cdot [\omega(\tau_r) : \mu]^X$$

and $m(\gamma) = [\nu : \mu]^X$ for the one-edge path $\gamma = (\nu, \mu)$, if it exists. In other words, the multiplicity is the product of incidences for downward arrows, times the product of minus the inverse of incidences for upward arrows in the path.

Given a complex X and a Morse matching $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$, the *Morse complex* $(\mathcal{A}, \partial^{\mathcal{A}})$ associated to the matching is the complex based on the cells of $\mathcal{A}$, called the *critical cells*, with incidence function $[\cdot : \cdot]^{\mathcal{A}} : \mathcal{A} \times \mathcal{A} \to \mathbf{D}$ defined, for two

[2]Note that our definition differs from the original reference [37], where gradient paths connect cells of same dimension.

critical cells $\nu, \mu \in \mathcal{A}$, by

$$[\nu : \mu]^{\mathcal{A}} := \sum_{\gamma \in \Gamma(\nu,\mu)} m(\gamma).$$

The dimension of a critical cell σ in $\mathcal{A}$ is the same as the dimension of σ in the original complex X. We denote the set of d-dimensional cells of $\mathcal{A}$ by $\mathcal{A}_d$. As a complex, the boundary operator of $\mathcal{A}$ is defined, for $\sigma \in \mathcal{A}_d$ a critical cell of dimension d, by

$$\partial_d^{\mathcal{A}} : \mathcal{A}_d \to \mathcal{A}_{d-1}, \quad \text{such that} \quad \partial_d^{\mathcal{A}} \sigma = \sum_{\mu \in \mathcal{A}_{d-1}} [\sigma : \mu]^{\mathcal{A}} \cdot \mu.$$

We finally have the fundamental theorem of discrete Morse theory:

Theorem 3.1 (Forman [37])**.** *A complex (X, ∂^X) and a Morse complex $(\mathcal{A}, \partial^{\mathcal{A}})$, for a Morse matching $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$ of X, have isomorphic homology groups*[3]*.*

Persistent homology and discrete Morse theory. Let $X_1 \subseteq \ldots \subseteq X_m$ be a standard filtration of complexes. A *standard Morse filtration* (called *filtered Morse complex* in [49]) for this filtration is a collection of Morse matchings $(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)_{i=1\ldots m}$ for each X_i, with Morse complex $(\mathcal{A}_i, \partial^{\mathcal{A}_i})$ on the critical cells, and Morse pairs $\omega_i \colon \mathcal{K}_i \xrightarrow{\text{bij.}} \mathcal{Q}_i$, satisfying:

$$\begin{aligned} &\mathcal{A}_i \subseteq \mathcal{A}_{i+1}, \quad \mathcal{Q}_i \subseteq \mathcal{Q}_{i+1}, \quad \mathcal{K}_i \subseteq \mathcal{K}_{i+1}, \\ &\omega_{i+1}\big|_{\mathcal{Q}_i} = \omega_i, \quad \partial^{\mathcal{A}_{i+1}}\big|_{\mathcal{A}_i} = \partial^{\mathcal{A}_i}. \end{aligned} \tag{3.4}$$

A filtered Morse complex consequently forms a filtration $\mathcal{A}_1 \subseteq \ldots \subseteq \mathcal{A}_m$ of Morse complexes connected by inclusions. It induces naturally a persistence module:

$$H(\mathcal{A}_1) \longrightarrow H(\mathcal{A}_2) \longrightarrow \cdots \longrightarrow H(\mathcal{A}_{n-1}) \longrightarrow H(\mathcal{A}_m) \ .$$

Forman's isomorphism between homology groups of complexes and Morse complexes extends to persistent homology groups within this framework:

Theorem 3.2 (Forman [37], Mischaikow and Nanda [49])**.** *For a standard filtration $X_1 \subseteq \ldots \subseteq X_m$, let $(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)_{i=1\ldots m}$ be a standard Morse filtration. There exist collections of chain maps $(\psi_i : C(X_i) \to C(\mathcal{A}_i))_{i=1\ldots m}$ and $(\varphi_i : C(\mathcal{A}_i) \to C(X_i))_{i=1\ldots m}$ for which the following diagrams commute for every i:*

$$\begin{array}{ccc} C(X_i) & \hookrightarrow & C(X_{i+1}) \\ {\scriptstyle \psi_i}\downarrow & & \downarrow{\scriptstyle \psi_{i+1}} \\ C(\mathcal{A}_i) & \hookrightarrow & C(\mathcal{A}_{i+1}) \end{array} \qquad \begin{array}{ccc} C(X_i) & \hookrightarrow & C(X_{i+1}) \\ {\scriptstyle \varphi_i}\uparrow & & \uparrow{\scriptstyle \varphi_{i+1}} \\ C(\mathcal{A}_i) & \hookrightarrow & C(\mathcal{A}_{i+1}) \end{array}$$

and φ_i and ψ_i induce isomorphisms at the homology level, that are inverses of each other. Consequently, these maps induce isomorphisms between the persistent modules of the filtration and the Morse filtration.

[3] In fact, the complexes are *homotopy equivalent*.

Without expressing them explicitly, we use the following properties of the map ψ (see [49] for explicit formulations):

Properties 3.1. *Let X be a complex with a Morse matching $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$. The chain map $\psi\colon C(X) \to C(\mathcal{A})$ can be expressed as the composition of elementary chain maps over all Morse pairs (τ, σ), taken in an arbitrary order,*

$$\psi = \prod_{(\tau,\sigma),\ s.t.\ \sigma=\omega(\tau)} \psi_{\tau,\sigma}\ ,$$

where $\psi_{\tau,\sigma}\colon C(X') \to C(X' \setminus \{\tau, \sigma\})$ is defined on a "partially reduced" complex X' to $X' \setminus \{\tau, \sigma\}$, with incidence functions induced by the partial matching. More specifically, X' is a Morse complex of X for a matching $(\mathcal{A}', \mathcal{Q}', \mathcal{K}', \omega')$, such that $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathcal{K}' \subseteq \mathcal{K}$, and the restriction of ω to $\mathcal{Q}'$ is equal to ω'. The complex $X' \setminus \{\tau, \sigma\}$ is the Morse complex of X with one more Morse pair (τ, σ). The set of Morse pairs already considered in $\mathcal{Q}' \times \mathcal{K}'$ is dependent of the order in which the maps are composed.

The map $\psi_{\tau,\sigma}$ satisfies:

1. *$\psi_{\tau,\sigma}(\sigma) = 0$,*
2. *$\psi_{\tau,\sigma}(\tau)$ is a linear combination of facets of σ in X', and*
3. *$\psi_{\tau,\sigma}(\mu) = \mu$ for all $\mu \neq \sigma, \tau$.*

Similarly, the map $\varphi\colon C(\mathcal{A}) \to C(X)$ can be decomposed into

$$\varphi = \prod_{(\tau,\sigma),\ s.t.\ \sigma=\omega(\tau)} \varphi_{\tau,\sigma}\ ,$$

such that $\varphi_{\tau,\sigma}\colon C(X' \setminus \{\tau, \sigma\}) \to C(X')$ and $\psi_{\tau,\sigma}\colon C(X') \to C(X' \setminus \{\tau, \sigma\})$ induce isomorphisms at the homology level, that are inverse of each other (defined on the appropriate domain and codomain).

3.3 Zigzag Morse filtration and persistence

For a zigzag filtration of complexes $\mathcal{Z}$, we introduce in this article a canonical zigzag Morse filtration $\mathcal{M}$ of Morse complexes admitting the same persistent homology.

3.3.1 Zigzag Morse filtration

Without loss of generality, consider the zigzag filtration

$$\overline{\mathcal{Z}}:\ \overline{X}_1 \xhookrightarrow{\Sigma_1} \overline{X}_2 \xhookleftarrow{\Sigma_2} \cdots \xhookrightarrow{\Sigma_{2k-1}} \overline{X}_{2k-1} \xhookleftarrow{\Sigma_{2k}} \overline{X}_{2k}\ , \tag{3.5}$$

where the $\overline{X}_i$ are complexes, $\overline{X}_1 = \overline{X}_{2k} = \varnothing$, and the i^{th} arrow is an inclusion, either forward (i odd) or backward (i even), where complexes $\overline{X}_i$ and $\overline{X}_{i+1}$ differ by a set of cells Σ_i (possibly empty). We now further decompose $\overline{\mathcal{Z}}$.

Atomic operations. For each forward arrow $\bullet_i \to \bullet_{i+1}$, i odd, let $(\hat{\mathcal{A}}_i, \hat{\mathcal{Q}}_i, \hat{\mathcal{K}}_i, \hat{\omega}_i)$ be a Morse matching of the set of cells Σ_i.

Because Morse matchings are acyclic, there exists a total ordering of the cells of Σ_i, compatible with the face partial ordering of Σ_i, such that paired cells in $(\hat{\mathcal{A}}_i, \hat{\mathcal{Q}}_i, \hat{\mathcal{K}}_i, \hat{\omega}_i)$ are consecutive with regard to that order. We can consequently decompose a forward inclusion $\overline{X}_i \subseteq \overline{X}_{i+1}$ into a sequence of inclusions of a single critical cell $\sigma \in \hat{\mathcal{A}}_i$, and of inclusions of a single Morse pair of cells $(\tau, \sigma) \in \hat{\mathcal{Q}}_i \times \hat{\mathcal{K}}_i$, with $\sigma = \hat{\omega}_i(\tau)$.

For every backward arrow $\bullet_i \leftarrow \bullet_{i+1}$, i even, the Morse matchings $(\hat{\mathcal{A}}_j, \hat{\mathcal{Q}}_j, \hat{\mathcal{K}}_j, \hat{\omega}_j)$, for smaller odd indices $j < i$, induce a Morse matching on the cells of $\overline{X}_i$. To avoid ambiguity, if a cell is reinserted in the filtration after being removed it is considered as a different element. By restriction, they consequently induce a valid Morse matching on all cells of Σ_i, except on those cells $\sigma \in \Sigma_i$ that form a Morse pair (τ, σ), with $\tau \notin \Sigma_i$. We decompose backward arrows into a sequence of removals of a single critical cell, of removals of a single Morse pair of cells, and of removals of a non-critical cell σ, without its paired cell $\tau \notin \Sigma_i$.

In summary, given an input filtration $\overline{\mathcal{Z}}$ as above, and the Morse matchings $(\hat{\mathcal{A}}_i, \hat{\mathcal{Q}}_i, \hat{\mathcal{K}}_i, \hat{\omega}_i)$, we defined an *atomic* zigzag filtration

$$\mathcal{Z} : \ (\varnothing =) X_1 \longleftrightarrow X_2 \longleftrightarrow \cdots \longleftrightarrow X_{m-1} \longleftrightarrow X_m \, (= \varnothing) \ ,$$

where all arrows are of the following three types:

$$X \xleftrightarrow{\ \sigma\ } X' \tag{3.6}$$

$$X \xleftrightarrow{\ \{\tau,\sigma\}\ } X' \tag{3.7}$$

$$X \xrightarrow{\ \mathbb{1}\ } X \xleftarrow{\ \sigma\ } X \setminus \{\sigma\} \tag{3.8}$$

where σ is in each case a maximal cell in X, Diagrams (3.6) and (3.7) are forward or backward insertions of a critical cell or a Morse pair (τ, σ) of cells, respectively, and Diagram (3.8) is the removal of the cell σ from a Morse pair (τ, σ), where the cell τ is not removed. The identity arrow in this last diagram is a technicality that is clarified later. Naturally, one can recover the persistent homology of the zigzag filtration $\overline{\mathcal{Z}}$ from the one of $\mathcal{Z}$. We work with $\mathcal{Z}$ for the rest of the chapter.

Morse filtration. Given a zigzag filtration $\overline{\mathcal{Z}}$, Morse matchings $(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)$, and an associated atomic filtration $\mathcal{Z}$ as above, we define a *zigzag Morse filtration*

$$\mathcal{M} : \ (\varnothing =) \mathcal{A}_1 \longleftrightarrow \mathcal{A}_2 \longleftrightarrow \cdots \longleftrightarrow \mathcal{A}_{m-1} \longleftrightarrow \mathcal{A}_m \, (= \varnothing) \ ,$$

of Morse complexes $(\mathcal{A}_i, \partial^{\mathcal{A}_i})$ of the complexes (X_i, ∂^{X_i}) of $\mathcal{Z}$ inductively. Note that the maps of the zigzag Morse filtration are not all inclusions. Specifically, for a critical cell σ in both X_i and X_{i+1}, in general $\partial^{\mathcal{A}_i}(\sigma) \neq \partial^{\mathcal{A}_{i+1}}(\sigma)$.

All $X_1, X_m, \mathcal{A}_1$ and $\mathcal{A}_m$ are empty complexes. The zigzag Morse filtration is constructed inductively for the insertion of a critical cell (Diagram (3.6)) and the

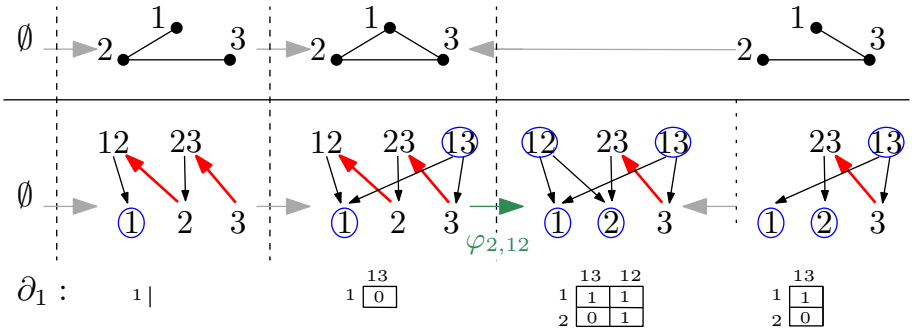


Figure 3.1: Zigzag filtration (top) and its Morse filtration (bottom), given by Hasse diagrams and (Morse) boundary maps. Upward arrows in Hasse diagrams represent Morse matchings, critical faces are circled. Note that the rightmost operation illustrates Diagram (3.10), with a non trivial modification of $\partial_1(\{1,3\})$.

insertion of a Morse pair (Diagram (3.7)) as for standard Morse filtrations [49]:

$$\begin{array}{ccc} C(X) & \xhookrightarrow{\sigma'} & C(X\cup\{\sigma'\}) \\ {\scriptstyle\psi}\downarrow & & \downarrow{\scriptstyle\psi} \\ C(\mathcal{A}) & \xhookrightarrow{\sigma'} & C(\mathcal{A}\cup\{\sigma'\}) \end{array} \qquad \begin{array}{ccc} C(X) & \xhookrightarrow{\{\tau,\sigma\}} & C(X\cup\{\tau,\sigma\}) \\ {\scriptstyle\psi}\downarrow & & \downarrow{\scriptstyle\psi_{\tau,\sigma}\circ\psi} \\ C(\mathcal{A}) & \xrightarrow{\mathbb{1}} & C(\mathcal{A}) \end{array} , \tag{3.9}$$

where all horizontal arrows are inclusions of complexes, and in particular the boundary maps of $\mathcal{A}$ and $\mathcal{A}\cup\{\sigma'\}$ are equal when restricted to the cells of $\mathcal{A}$. The removal of critical cells and Morse pairs is symmetrical. The chain maps ψ and $\psi_{\tau,\sigma}$ are the ones of Theorem 3.2 and Properties 3.1, and are used later.

For the removal of a non-critical cell σ without its paired cell τ (Diagram (3.8)), which is specific to zigzag persistence, the Morse filtration is constructed with:

$$\begin{array}{ccccc} C(X) & \xrightarrow{\mathbb{1}} & C(X) & \xhookleftarrow{\sigma} & C(X\setminus\{\sigma\}) \\ {\scriptstyle\psi_{\tau,\sigma}\circ\psi}\downarrow & & \downarrow{\scriptstyle\psi} & & \downarrow{\scriptstyle\psi} \\ C(\mathcal{A},\partial) & \xrightarrow{\varphi_{\tau,\sigma}} & C(\mathcal{A}\cup\{\tau,\sigma\},\partial') & \xhookleftarrow{\sigma} & C(\mathcal{A}\cup\{\tau\},\partial'') \end{array} . \tag{3.10}$$

The main technicality is that the boundary maps ∂ and ∂' differ in a non trivial way, that we study in Section 3.4. The map ∂'' is equal to the restriction of ∂' to the critical cells $\mathcal{A}\cup\{\tau\}$ (the right arrow is a backward inclusion of complexes). The chain maps $\psi_{\tau,\sigma}$ and $\varphi_{\tau,\sigma}$ are the ones from Theorem 3.2 and Properties 3.1, and ψ is the compositions of all maps $\psi_{\mu,\omega(\mu)}$ over the Morse pairs $(\mu,\omega(\mu))$ of the Morse matching of X, except the pair (τ,σ). We give an example of zigzag Morse filtration in Figure 3.1.

Diagrams (3.9) are studied in [49]. We now focus on the study of Diagram (3.10).

Remark 3.1. Note that a key point for the proofs of theorems in [49] is that filtered Morse complexes in standard persistence satisfy $(\mathcal{A}_i,\partial)\subset(\mathcal{A}_{i+1},\partial)$. This fact also allows the standard persistent homology algorithm [34, 59] to work directly for filtered Morse complexes. This property is not satisfied by

zigzag Morse filtrations, which explains why our approach is more atomic than the one of [49] (see Section 3.3.2), and that we have to design a new homology matrix algorithm to implement operation (3.10) (see Sections 3.4 and 3.5).

3.3.2 Isomorphism of zigzag modules

Theorem 3.2 implies that the atomic operations of Diagrams (3.9) induce commuting diagrams in homology, with vertical maps being isomorphisms as proved in [49]:

Lemma 3.3. *Let X be a complex and $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$ a Morse complex obtained from X. Let σ' be a cell, and (τ, σ) a Morse pair, such that $(\mathcal{A} \cup \{\sigma'\}, \mathcal{Q}, \mathcal{K}, \omega)$ and $(\mathcal{A}, \mathcal{Q} \cup \{\tau\}, \mathcal{K} \cup \{\sigma\}, \omega)$ are valid Morse complexes. Then there exist isomorphisms ψ^* and $(\psi_{\tau,\sigma})^*$ such that the following diagrams commute:*

$$\begin{array}{ccc} H(X) & \xrightarrow{\sigma'^*} & H(X \cup \{\sigma'\}) \\ {\scriptstyle \psi^*}\downarrow & & \downarrow{\scriptstyle \psi^*} \\ H(\mathcal{A}) & \xrightarrow{\sigma'^*} & H(\mathcal{A} \cup \{\sigma'\}) \end{array} \qquad \begin{array}{ccc} H(X) & \xrightarrow{\sigma^* \circ \tau^*} & H(X \cup \{\tau, \sigma\}) \\ {\scriptstyle \psi^*}\downarrow & & \downarrow{\scriptstyle (\psi_{\tau,\sigma})^* \circ \psi^*} \\ H(\mathcal{A}) & \xrightarrow{\mathbb{1}} & H(\mathcal{A}) \end{array}$$

where σ'^* and $\sigma^* \circ \tau^*$ are the maps induced at homology level by the insertion of σ' and $\{\tau, \sigma\}$ respectively. The maps ψ^* and $(\psi_{\tau,\sigma})^*$ are the isomorphisms induced by chain maps ψ and $\psi_{\tau,\sigma}$ of discrete Morse theory (see Theorem 3.2).

We prove the following lemma, which is specific to our zigzag Morse filtration.

Lemma 3.4. *Let X be a complex and $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$ a Morse complex obtained from X. Let σ be a maximal cell of X not in $\mathcal{A}$, which therefore forms a Morse pair with a cell τ, $[\sigma : \tau]^X \neq 0$. There exist isomorphisms ψ^*, $(\psi_{\tau,\sigma})^*$, and $(\varphi_{\tau,\sigma})^*$ such that the following diagram commutes:*

$$\begin{array}{ccccc} H(X) & \xrightarrow{\mathbb{1}} & H(X) & \xleftarrow{\sigma^*} & H(X \setminus \{\sigma\}) \\ {\scriptstyle (\psi_{\tau,\sigma})^* \circ \psi^*}\downarrow & & \downarrow{\scriptstyle \psi^*} & & \downarrow{\scriptstyle \psi^*} \\ H(\mathcal{A}) & \xrightarrow{(\varphi_{\tau,\sigma})^*} & H(\mathcal{A} \cup \{\tau, \sigma\}) & \xleftarrow{\sigma^*} & H(\mathcal{A} \cup \{\tau\}) \end{array}$$

where σ^* is the map induced at homology level by the removal of σ. The maps ψ^*, $(\psi_{\tau,\sigma})^*$, and $(\varphi_{\tau,\sigma})^*$ are the isomorphisms induced at homology level by, respectively, the chain maps ψ, $\psi_{\tau,\sigma}$, and $\varphi_{\tau,\sigma}$ of discrete Morse theory (see Theorem 3.2).

Proof. Apply the homology functor to Diagram (3.10). The right square commutes, being induced by horizontal inclusions. Because the maps induced at homology level by $\psi_{\tau,\sigma}$ and $\varphi_{\tau,\sigma}$ are isomorphisms, inverse of each other (see Theorem 3.2), we get $(\varphi_{\tau,\sigma})^* \circ (\psi_{\tau,\sigma})^* \circ \psi^* = \psi^*$ and the left square commutes. □

We conclude,

Theorem 3.5. *The zigzag filtrations $\mathcal{Z}$ and $\mathcal{M}$ have same persistent homology.*

Proof. Applying the homology functor to $\mathcal{Z}$ and $\mathcal{M}$, we get the zigzag modules

$$\begin{array}{lccccccc} H(\mathcal{Z}): & H(X_0) & \longleftrightarrow & H(X_1) & \longleftrightarrow & \cdots & \longleftrightarrow & H(X_m) \\ & \downarrow{\scriptstyle \psi_0^*} & & \downarrow{\scriptstyle \psi_1^*} & & & & \downarrow{\scriptstyle \psi_m^*} \\ H(\mathcal{M}): & H(\mathcal{A}_0) & \longleftrightarrow & H(\mathcal{A}_1) & \longleftrightarrow & \cdots & \longleftrightarrow & H(\mathcal{A}_m) \end{array}$$

where, by construction, every $\mathcal{A}_i$ is a Morse complex of X_i, and the ψ_i^* are the isomorphisms induced by the chain maps $\psi_i \colon C(X_i) \to C(\mathcal{A}_i)$, connecting a complex and its Morse reduction (Theorem 3.2). By Theorem 3.2 and Lemma 3.4, all squares commute and are compatible with each other, and the $(\psi_i^*)_i$ define an isomorphism of zigzag modules. □

3.4 Boundary of the Morse complex

Let X be a complex with incidence function $[\cdot:\cdot]^X$, together with a Morse matching $(\mathcal{A}, \mathcal{Q}, \mathcal{K}, \omega)$, inducing an orientation of the Hasse diagram $\mathcal{H}$ of the complex, and a Morse complex $(\mathcal{A}, \partial)$.

In this section, we track the evolution of the boundary operators in Morse complexes under the evaluation of the map $\varphi_{\tau,\sigma} \colon (\mathcal{A}, \partial) \to (\mathcal{A} \cup \{\tau, \sigma\}, \partial')$ from Diagram (3.10). Both complexes are Morse complexes of the same X, whose matchings differ by exactly one pair (τ, σ), i.e., the Morse partition of complex $\mathcal{A} \cup \{\tau, \sigma\}$ is $(\mathcal{A} \cup \{\tau, \sigma\}) \sqcup (\mathcal{Q} \setminus \{\tau\}) \sqcup (\mathcal{K} \setminus \{\sigma\})$. We denote this last complex by $(\mathcal{A}', \partial')$, with incidence function $[\cdot:\cdot]^{\mathcal{A}'}$ in the following. We prove:

Lemma 3.6. *Let ν be a cell of the complex $(\mathcal{A}, \partial)$. Then, in the complex $(\mathcal{A}', \partial')$,*

$$\partial'(\nu) = \partial(\nu) + \left([\sigma:\tau]^X\right)^{-1} [\nu:\tau]^{\mathcal{A}'} \cdot \partial'\sigma. \tag{3.11}$$

Proof. First, note that σ is maximal in X, and so it is maximal in $\mathcal{A} \cup \{\tau, \sigma\}$.

Let $\mathcal{H}$ and $\mathcal{H}'$ be the Hasse diagrams of X induced by the Morse matchings of $\mathcal{A}$ and $\mathcal{A}'$, respectively. Because the matchings differ by a single Morse pair (τ, σ), $\mathcal{H}$ and $\mathcal{H}'$ only differ by the orientation of the edge $\tau \leftrightarrow \sigma$.

For a critical cell $\nu \in \mathcal{A}$, we have:

$$\partial\nu = \sum_{\substack{\mu\in\mathcal{A} \\ \gamma\in\Gamma(\nu,\mu)}} m(\gamma)\cdot\mu = \underbrace{\sum_{\substack{\mu\in\mathcal{A}, \\ \gamma\in\Gamma_{\tau\to\sigma}(\nu,\mu)}} m(\gamma)\cdot\mu}_{(\star)} + \underbrace{\sum_{\substack{\mu\in\mathcal{A}, \\ \gamma\in\Gamma_{\tau\nrightarrow\sigma}(\nu,\mu)}} m(\gamma)\cdot\mu}_{\partial'\nu - [\nu:\tau]^{\mathcal{A}'}\cdot\tau},$$

where $\Gamma_{\tau\to\sigma}(\nu,\mu)$ are the gradient paths from ν to μ in $\mathcal{H}$ containing the upward arrow $\tau \to \sigma$, and $\Gamma_{\tau\nrightarrow\sigma}(\nu,\mu)$ are the ones not containing it. Assume τ is of dimension d, and σ of dimension $d+1$.

Because σ is critical in $\mathcal{A}'$, it has no ingoing arrow from cells of dimension d in $\mathcal{H}'$. Consequently, $\Gamma_{\tau\nrightarrow\sigma}(\nu,\mu)$ contains exactly all gradient paths from ν to $\mu \neq \tau$ in $\mathcal{H}'$. Hence, the sum over $\Gamma_{\tau\nrightarrow\sigma}(\nu,\mu)$, for $\mu \in \mathcal{A}$, gives $\partial'\nu - [\nu:\tau]^{\mathcal{A}'}\tau$. Note that σ cannot appear in $\partial'\nu$ because σ is maximal by hypothesis.

Now, studying the left term $(\star)$, and splitting gradient paths passing through edge (τ, σ), then factorizing, we get

$$
\begin{aligned}
(\star) &= \sum_{\substack{\mu\in\mathcal{A},\\ \gamma_1\in\Gamma(\nu,\tau),\\ \gamma_2\in\Gamma(\sigma,\mu)}} m(\gamma_1)\cdot\Big(-[\sigma:\tau]^X\Big)^{-1} m(\gamma_2)\cdot\mu \\
&= -\Big([\sigma:\tau]^X\Big)^{-1} \underbrace{\sum_{\mu\in\mathcal{A}}\left(\sum_{\gamma_2\in\Gamma(\sigma,\mu)} m(\gamma_2)\cdot\mu\right)}_{(\star_2)\ =\ \partial'\sigma-[\sigma:\tau]\cdot\tau} \cdot \underbrace{\left(\sum_{\gamma_1\in\Gamma(\nu,\tau)} m(\gamma_1)\right)}_{(\star_1)}.
\end{aligned}
$$

The sum $(\star_1)$ over $\Gamma(\nu,\tau)$ is independent of μ, and equal to $[\nu:\tau]^{\mathcal{A}'}$ by definition.

Because τ is critical in $\mathcal{A}'$, it has no outgoing arrow towards cells of dimension $d+1$ in $\mathcal{H}'$. Consequently, $\Gamma(\sigma,\mu)$ contains exactly all gradient paths from σ to μ in $\mathcal{H}'$, where $\mu\neq\tau$. Hence, the sum $(\star_2)$ over $\Gamma(\sigma,\mu)$ gives $\partial'\sigma - [\sigma:\tau]^X\cdot\tau$.

Finally, putting terms together, the following allows us to conclude:

$$
\begin{aligned}
\partial\nu &= \Big(\partial'\nu - [\nu:\tau]^{\mathcal{A}'}\cdot\tau\Big) - \frac{[\nu:\tau]^{\mathcal{A}'}}{[\sigma:\tau]^X}\Big(\partial'\sigma - [\sigma:\tau]^X\cdot\tau\Big) \\
&= \partial'\nu - \Big([\sigma:\tau]^X\Big)^{-1}[\nu:\tau]^{\mathcal{A}'}\partial'\sigma.
\end{aligned}
$$

□

3.5 Persistence algorithm for zigzag Morse complexes

We describe in this section, our implementation of the algorithm to compute the persistence diagram of a zigzag Morse filtration as defined in Section 3.3. It consists of adapting the zigzag persistence algorithm [45], used in our experiments, to our Morse framework, relying on the results of Sections 3.3 and 3.4. Our approach could be adapted for implementing algorithm [9, 10].

3.5.1 Zigzag persistence algorithm

We first explain briefly the algorithms for computing zigzag persistence.

Existing zigzag persistence algorithms. There are currently two practical[4] approaches to compute zigzag persistent homology [9, 10, 45]. They can both be formulated in a unified framework [46]. Given an input zigzag filtration:

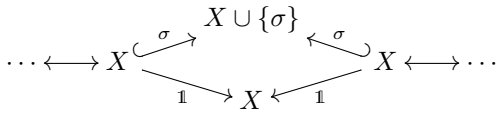
$$
X_1 \hookrightarrow X_2 \hookleftarrow \cdots \hookrightarrow X_{n-1} \hookleftarrow X_n\ , \tag{3.12}
$$

both algorithms are iterative. At step i of the computation, they maintain a homology basis of $H(X_i)$ that is *compatible* (defined later) with the interval decomposition of the zigzag module associated to a zigzag filtration of the form

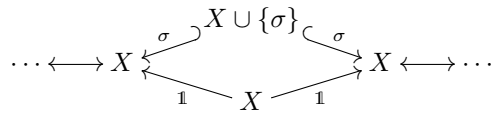
$$
X_1 \longleftrightarrow \cdots \longleftrightarrow X_i \longleftrightarrow X'_{i+1} \longleftrightarrow \cdots \longleftrightarrow X'_{i+m-1} \longleftrightarrow X'_{i+m}\ , \tag{3.13}
$$

[4]Putting aside [48], which is essentially of theoretical nature.

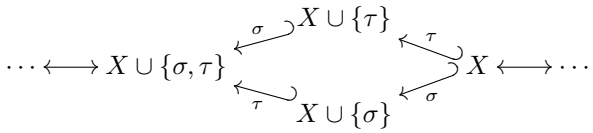
The first i complexes and $i-1$ maps in (3.12) and (3.13) are identical, and the remaining complexes and maps of (3.13) are algorithm dependent. Both algorithms consist of updating a homology basis in order to maintain its compatibility when operating (a subset of) the following three local transformations of the zigzag filtration/module in sequence:



(3.14)



(3.15)



(3.16)

where each arrow represents the insertion of a cell. These transformations are called *reflection diamonds* for (3.14) and (3.15), and *transposition diamonds* for (3.16), and their effect on the interval decomposition of the zigzag module have been characterized for general zigzag filtrations of complexes in [46, 45].

We now focus on the algorithm introduced in [45] that we use in our experiments.

The zigzag algorithm of [45]. For a zigzag filtration or a zigzag Morse filtration $\mathcal{S}$ of length m, denote by $\mathcal{S}[p;q]$, $1 \leq p \leq q \leq m$, the restriction of $\mathcal{S}$ to spaces with indices $i \in [p;q]$, and the maps between them.

Let $\mathcal{Z} : X_1 \leftrightarrow X_2 \leftrightarrow \cdots \leftrightarrow X_m$ be the input zigzag filtration, where *all* arrows are forward or backward inclusions of a *single* cell. The algorithm reads $\mathcal{Z}$ from left to right iteratively, one map after another. At step j, let $\mathcal{Z}_j$ be:

$$\mathcal{Z}_j : \; X_1 \longleftrightarrow X_2 \longleftrightarrow \cdots \longleftrightarrow X_j \xleftarrow{\sigma_1} X'_{j+1} \xleftarrow{\sigma_2} \cdots \xleftarrow{\sigma_{m-1}} X'_{j+m-1} \xleftarrow{\sigma_m} X'_{j+m} = \varnothing.$$

where $\mathcal{Z}_j[1;j] = \mathcal{Z}[1;j]$ and $\mathcal{Z}_j[j+1;j+m]$ is a "backward" filtration where every cell σ_i in X_j is removed in order of insertion. We will now define how to go to the next step $j+1$, i.e., how to construct $\mathcal{Z}_{j+1}$ from $\mathcal{Z}_j$ by using reflection and transposition diamonds. We have two cases:

• If $X_j \xrightarrow{\sigma} X_{j+1}$ is forward in $\mathcal{Z}$, consider $\overline{\mathcal{Z}}_j$, that is $\mathcal{Z}_j$ with two extra identity arrows:

$$\overline{\mathcal{Z}}_j : \; X_1 \longleftrightarrow \cdots \longleftrightarrow X_j \xrightarrow{\mathbb{1}} X_j \xleftarrow{\mathbb{1}} X_j \xleftarrow{\sigma_1} X'_j \xleftarrow{\sigma_2} \cdots \xleftarrow{\sigma_m} X'_{j+m} = \varnothing \; ,$$

We can then apply a reflection diamond (3.14) at X_j to obtain $\mathcal{Z}_{j+1}$:

$$\mathcal{Z}_{j+1} : \; X_1 \longleftrightarrow \cdots \longleftrightarrow X_j \xrightarrow{\sigma} X_{j+1} \xleftarrow{\sigma} X_j \xleftarrow{\sigma_1} X'_{j+1} \xleftarrow{\sigma_2} \cdots \xleftarrow{\sigma_m} X'_{j+m} = \varnothing \; .$$

Studying the effect of a reflection diamond on homology, algorithm [45] updates

a homology matrix (defined below in this framework) at X_j, compatible with $\mathcal{Z}_j$ (and also $\overline{\mathcal{Z}}_j$), into a homology matrix at X_{j+1}, compatible with $\mathcal{Z}_{j+1}$ defined above.

• If $X_j \xleftarrow{\sigma} X_{j+1}$ is backward in $\mathcal{Z}$, there exists an index ℓ such that $\sigma = \sigma_\ell$ in the part $\mathcal{Z}_j[j; j+m]$ of the filtration $\mathcal{Z}_j$. By applying successively transposition diamonds (3.16) in $\mathcal{Z}_j[j; j+m]$ to move up the removal arrow of σ_ℓ until reaching X_j, we obtain $\mathcal{Z}_{j+1}$:

$$\cdots X_j \xleftarrow{\sigma_\ell=\sigma} X_{j+1} \xleftarrow{\sigma_1} X'_{j+1} \setminus \{\sigma\} \cdots \xleftarrow{\sigma_{\ell-2}} X'_{j+\ell-2} \setminus \{\sigma\} \xleftarrow{\sigma_{\ell-1}} X'_{j+\ell} \xleftarrow{\sigma_{\ell+1}} \cdots ,$$

Studying the effect of transposition diamonds on homology, algorithm [45] updates a homology matrix at X_j, compatible with $\mathcal{Z}_j$, into a homology matrix at X_{j+1}, compatible with $\mathcal{Z}_{j+1}$ defined above.

3.5.2 Adaptation to zigzag Morse filtrations

Using notations from Section 3.3, let $\overline{\mathcal{Z}}$ be a general zigzag filtration:

$$\overline{\mathcal{Z}} : \ (\varnothing =) \overline{X}_1 \xhookrightarrow{\Sigma_1} \overline{X}_2 \xhookleftarrow{\Sigma_2} \cdots \xhookrightarrow{\Sigma_{2k-1}} \overline{X}_{2k-1} \xhookleftarrow{\Sigma_{2k}} \overline{X}_{2k} (= \varnothing) ,$$

together with Morse matchings $(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)$ on the set of cells Σ_i of every *forward* inclusion $\overline{X}_i \xrightarrow{\Sigma_i} \overline{X}_{i+1}$, i odd.

Let $\mathcal{Z}$ be the associated *atomic* zigzag filtration of complexes where all maps are forward or backward inclusions of a single cell: $\mathcal{Z} : X_1 \leftrightarrow \cdots \leftrightarrow X_m$.

Algorithm [45] can update a homology matrix for a general complex using reflection and transposition diamonds to implement the insertion and deletion of cells pictured in Diagrams (3.9). We now implement the operation of Diagram (3.10), introducing the chain map $\varphi_{\tau,\sigma}$.

At step j of the algorithm, we maintain a zigzag Morse filtration $\mathcal{M}_j$ for the filtration $\mathcal{Z}_j$. At space X_j, the filtration satisfies:

Properties 3.2 (Zigzag Morse filtration $\mathcal{M}_j$)**.**
1. The filtration $\mathcal{M}_j[1; j]$ is a general zigzag Morse filtration (defined in Section 3.3.1) for $\mathcal{Z}[1; j]$ and its Morse matchings $\{(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)\}_{i=1\ldots j}$,

2. the filtration $\mathcal{M}_j[j; j+m]$ is a standard Morse filtration (defined in [49] and Equation (3.4)*) for the* standard *filtration $\mathcal{Z}_j[j; j+m]$.*

Before exhibiting the filtrations, we prove the following simple property of the zigzag persistence algorithm,

Lemma 3.7. *Let τ, σ be cells of X_j, and let $X_p \xrightarrow{\tau} X_{p+1}$ and $X_q \xrightarrow{\sigma} X_{q+1}$ be the two maps in $\mathcal{Z}$ that have the largest indices $1 \leq p, q < j$ for which a forward inclusion of τ and σ, respectively, happens in $\mathcal{Z}[1; j]$.*

Let $X'_{j+r-1} \xleftarrow{\tau} X'_{j+r}$ and $X'_{j+s-1} \xleftarrow{\sigma} X'_{j+s}$, for indices $1 \leq r, s \leq m$, be the backward inclusions of τ and σ in the part $\mathcal{Z}_j[j; j+m]$ of the filtration $\mathcal{Z}_j$. Then,

$$p < q \quad \textit{iff} \quad s < r.$$

In other words, if τ is inserted before σ, it is removed after σ.

Proof. The only "new" arrows in the diagram are brought by the reflection diamonds (3.14) applied at index j of the algorithm, on $\mathcal{Z}_j$, which induces the desired symmetry in forward and backward arrows for the insertion of a given cell. We refer to [45] for details on the algorithm. □

Now, consider the following diagram, where (τ, σ) are cells of X_j which are paired in the Morse matching of X_j induced by the Morse matchings $\{(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)\}_{i=1\dots j}$ of the filtration,

$$
\begin{array}{rccccccccccc}
\mathcal{Z}_j: & \hookleftarrow & X_j & \xrightarrow{\mathbb{1}} & X_j & \longleftarrow & X'_{j+1} & \longleftarrow\cdots\longleftarrow & X'_{j+r} & \longleftarrow & X, \sigma, \tau \xleftarrow{\sigma} X, \tau \xleftarrow{\tau} X & \longleftarrow & X_{j+r-2} & \hookleftarrow \\
 & & \downarrow{\scriptstyle \psi_{\tau,\sigma}\circ\psi} & & \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi} \quad \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi} & \\
\overline{\mathcal{M}}_j: & \hookleftarrow & \mathcal{A}_j & \xrightarrow[\varphi_{\tau,\sigma}]{} & \mathcal{A}_j, \sigma, \tau & \longleftarrow & \mathcal{A}'_{j+1}, \sigma, \tau & \longleftarrow\cdots\longleftarrow & \mathcal{A}'_{j+r}, \sigma, \tau & \longleftarrow & \mathcal{A}, \sigma, \tau \xleftarrow[\sigma]{} \mathcal{A}, \tau \xleftarrow[\tau]{} \mathcal{A} & \longleftarrow & \mathcal{A}'_{j+r-2} & \hookleftarrow \\
 & & \downarrow{\scriptstyle \mathbb{1}} & & \downarrow{\scriptstyle \psi_{\tau,\sigma}} & & \downarrow{\scriptstyle \psi_{\tau,\sigma}} & & \downarrow{\scriptstyle \psi_{\tau,\sigma}} & & \downarrow{\scriptstyle \psi_{\tau,\sigma}} & & \downarrow{\scriptstyle \mathbb{1}} \quad \downarrow{\scriptstyle \mathbb{1}} & \\
\mathcal{M}_j: & \hookleftarrow & \mathcal{A}_j & \longrightarrow & \mathcal{A}_j & \longleftarrow & \mathcal{A}'_{j+1} & \longleftarrow\cdots\longleftarrow & \mathcal{A}'_{j+r} & \longleftarrow & \mathcal{A} & \longleftarrow & \mathcal{A} & \longleftarrow & \mathcal{A}'_{j+r-2} & \hookleftarrow
\end{array}
\tag{3.17}
$$

where arrows without label are simple inclusions of complexes. Simplifying notations, we denote by X the complex X'_{j+r-1}, by $\mathcal{A}$ the complex $\mathcal{A}'_{j+r-1}$, and union of a complex and some cells by X, σ, τ, instead of $X \cup \{\sigma, \tau\}$. We use this diagram until the end of the section, and define its various components progressively.

Lemma 3.7 ensures that τ and σ, that are consecutively inserted (Morse pair, Diagram (3.7)), are consecutively removed in $\mathcal{Z}_j[j; j+m]$, as pictured above. The filtration $\mathcal{Z}_j$ appears on top, where two arrows (curved horizontal) are further decomposed for convenience.

By induction, let $\mathcal{M}_j$ be the zigzag Morse filtration maintained by the algorithm at step j, and satisfying Properties 3.2. Performing reflection diamonds (3.14) at index j, and transposition diamonds (3.16) at indices $j+r$, $r > 0$, maintains the Properties 3.2. Consequently, at the level of the zigzag Morse filtration, the zigzag algorithm [45] can implement insertions and deletions of critical cells (Diagrams (3.9)) with no further modification, while maintaining a Morse filtration $\mathcal{M}_j \mapsto \mathcal{M}_{j+1}$ satisfying the algorithmic invariant Properties 3.2.

The only obstruction to using the zigzag persistence algorithm is the operation introduced in Diagram (3.10). Consequently, consider the next operation in $\mathcal{Z}$ to be the removal $X_j \xleftarrow{\sigma} X_{j+1}$ of a non-critical cell σ, paired with a cell τ in the Morse matching of X_j, such that τ is not removed. The cell σ cannot be "directly removed" as it does not appear in $\mathcal{M}_j[j; j+m]$. We focus the rest of this section to the definition and study of the zigzag Morse filtration $\overline{\mathcal{M}}_j$ of Diagram (3.17).

Let $\overline{\mathcal{M}}_j$ be as in the diagram above, where the map $\varphi_{\tau,\sigma}$ is the map defined in Diagram (3.10), and the chain maps ψ between $\mathcal{Z}_j$ and $\overline{\mathcal{M}}_j$ are the ones of Diagrams (3.9) and (3.10). By Theorem 3.5, these maps induce an isomorphism of zigzag modules $H(\mathcal{Z}_j) \to H(\overline{\mathcal{M}}_j)$, and the filtrations have same persistent homology. Additionally, $\overline{\mathcal{M}}_j$ is a zigzag Morse filtration and a standard Morse filtration from space $\mathcal{A}_j, \sigma, \tau$ on to the right, i.e., it satisfies Properties 3.2. Finally, σ is critical in $\mathcal{A}_j, \sigma, \tau$, and can be removed with the zigzag persistence algorithm to obtain $\mathcal{M}_{j+1}$.

Compatible homology matrix. We design an algorithm to turn a homology matrix at $\mathcal{A}_j$, compatible with $\mathcal{M}_j$, into a homology matrix at $\mathcal{A}_j \cup \{\tau, \sigma\}$, compatible with $\overline{\mathcal{M}}_j$, in Diagram (3.17).

Consider X_j in $\mathcal{Z}_j$ containing m cells:

Definition 3.2 ([23])**.** *Let X be a cell complex of size m and $\mathcal{B} = \{c_0, \ldots, c_{m-1}\}$ be a collection of m chains of $C(X)$. We say that $\mathcal{B}$ is a* homology matrix *at X if there exists an ordering $\sigma_0, \ldots, \sigma_{m-1}$ of the m cells of X such that:*

0. for all $0 \leq r < m$, the restriction $\{\sigma_0, \ldots, \sigma_r\} \subset X$ is a subcomplex of X,

1. for all $0 \leq r < m$, the leading term of c_r is σ_r for the chosen ordering, i.e., $c_r = \varepsilon_0\sigma_0 + \ldots + \varepsilon_{r-1}\sigma_{r-1} + \sigma_r$, for some $\varepsilon_i \in \mathbb{F}$,

and there exists a partition $\{0, \ldots, m-1\} = F \sqcup G \sqcup H$, and a bijective pairing $G \leftrightarrow H$, satisfying:

2. for all indices $f \in F$, $\partial^{X_j} c_f = 0$,

3. for all pairs $g \leftrightarrow h$ of $G \times H$, $\partial^{X_j} c_h = c_g$.

This data encodes [23] the persistent homology of the (standard) filtration $\mathcal{Z}_j[j; j+m]$. In particular, the homology groups of X_j are equal to $\langle [c_f] : f \in F \rangle$. It is convenient to see this data as a matrix $M_{\mathcal{B}}$ with cycle c_i as i^{th} column, expressed in the basis $\{\sigma_i\}_{i=1\ldots m}$ for rows. In this case, condition (1) of the definition is equivalent to the matrix being upper triangular, with no zero entry in the diagonal.

Additionally,

Definition 3.3 ([45])**.** *We denote by $\bigoplus_\ell \mathbb{I}[b_\ell; d_\ell]$ the interval decomposition of $H(\mathcal{Z}_j)$. A homology matrix $\mathcal{B} = \{c_0, \ldots, c_{m-1}\}$ at X_j is* compatible *with the filtration $\mathcal{Z}_j$ iff there exists a zigzag module isomorphism $\Phi^* \colon H(\mathcal{Z}) \to \bigoplus_\ell \mathbb{I}[b_\ell; d_\ell]$ such that $\Phi_j^* \colon H(X_j) \to \mathbb{F}^{|F|}$ sends $\{[c_f] : f \in F\}$ to the canonical basis of $\mathbb{F} \times \cdots \times \mathbb{F}$.*

The Morse theory algorithm for persistent homology of [49] can be applied to maintain a compatible homology matrix for a Morse filtration under the operations pictured in Diagrams (3.9). We design the update for the new operation of Diagram (3.10). Consider:

$$\mathcal{M}_j : \ \mathcal{A}_1 \longleftrightarrow \cdots \longleftrightarrow \mathcal{A}_j \quad \text{and}$$
$$\mathcal{Z}_j : \ X_1 \longleftrightarrow \cdots \longleftrightarrow X_j \ ,$$

such that $\mathcal{M}_j$ is a zigzag Morse filtration for $\mathcal{Z}_j$. Assume $\mathcal{A}_j$ has m cells, and let $\mathcal{B} = \{c_0, \ldots, c_{m-1}\}$ be a homology matrix at $\mathcal{A}_j$ compatible with $H(\mathcal{M}_j)$. Following Diagram (3.10), consider:

$$\overline{\mathcal{M}}_j : \ \mathcal{A}_1 \longleftrightarrow \cdots \longleftrightarrow \mathcal{A}_j \longrightarrow \mathcal{A}_j \cup \{\tau, \sigma\} \quad \text{and}$$
$$\overline{\mathcal{Z}}_j : \ X_1 \longleftrightarrow \cdots \longleftrightarrow X_j \xrightarrow{\mathbb{1}} X_j$$

such that $\overline{\mathcal{M}}_j$ is a zigzag Morse filtration for $\overline{\mathcal{Z}}_j$. From $\mathcal{B}$, we define a homology matrix $\overline{\mathcal{B}} := \{c'_0, \ldots, c'_{m-1}, c_\tau, c_\sigma\}$ at $\mathcal{A}_j \cup \{\tau, \sigma\}$ that is compatible with $H(\overline{\mathcal{M}}_j)$.

Denote the two last complexes and their boundary maps in $\overline{\mathcal{M}}_j$ by $(\mathcal{A}_j, \partial)$ and $(\mathcal{A}'_j, \partial')$, with $\mathcal{A}'_j := \mathcal{A}_j \cup \{\tau, \sigma\}$. Then:

- for all indices $i \in F \sqcup H$, define

$$c'_i := c_i - \left([\sigma : \tau]^{X_j}\right)^{-1} \left(\sum_{\nu \in c_i} [\nu : \tau]^{\mathcal{A}'}\right) \cdot \sigma,$$

where the sum is taken over all cells ν in the support of chain c_i,

- define $c_\tau := \partial'\sigma$, and $c_\sigma := \sigma$, and put the index of c_τ in G, the index of c_σ in H, and pair them together,

- the pairing $G \leftrightarrow H$ inherited from $\mathcal{B}$ remains unchanged, and so does F.

Lemma 3.8. *The collection $\overline{\mathcal{B}}$ is a homology matrix at $\mathcal{A}_j \cup \{\tau, \sigma\}$ in Diagram* (3.17).

Proof. We prove that $\overline{\mathcal{B}}$ satisfies the conditions of Definition 3.2.

0. Because a Morse matching induces an acyclic Hasse Diagram, there exists r such that $\sigma_0, \ldots, \sigma_r, \tau, \sigma, \sigma_{r+1}, \ldots, \sigma_{m-1}$ is an ordering of the cells of $\mathcal{A}_j \cup \{\tau, \sigma\}$ such that the first k cells form a subcomplex, for any k, as in Definition 3.2.

1. **Case c_τ, c_σ.** The leading term of c_σ is σ. We prove that the leading term of c_τ is τ in the ordering defined above. Let $\mathcal{H}$ be the oriented Hasse diagram of X_j for the Morse matching where (τ, σ) forms a Morse pair (complex $\mathcal{A}_j$), and $\mathcal{H}'$ for the matching where τ and σ are critical (complex $\mathcal{A}_j \cup \{\tau, \sigma\}$); they differ by the orientation of arrow $\sigma \leftrightarrow \tau$. First, $\langle \partial'\sigma, \tau \rangle^{\mathcal{A}_j \cup \{\tau,\sigma\}} \neq 0$ because there exists a unique gradient path from critical cell σ to critical cell τ in $\mathcal{A}_j \cup \{\tau, \sigma\}$, which is the one edge path $\gamma = (\tau, \sigma)$. The path γ exists because τ is a facet of σ in X_j. If there were another distinct gradient path from σ to τ in $\mathcal{H}'$, not containing the edge $\sigma \to \tau$, this path would exist in $\mathcal{H}$ and form a cycle with edge $\tau \to \sigma$ in $\mathcal{H}$; a contradiction with the definition of Morse matchings. Second, if $\mu \in \mathcal{A}_j \cup \{\tau, \sigma\}$, is critical such that $[\sigma : \mu]^{\mathcal{A}_j \cup \{\tau,\sigma\}} \neq 0$, then μ appears before σ (and τ) in the ordering. Indeed, there exists a gradient path $\gamma = (\sigma, \mu_1, \omega(\mu_1), \ldots, \omega(\mu_{r-1}), \mu_r = \mu)$ from σ to μ in $\mathcal{H}'$. The cells $(\mu_i, \omega(\mu_i))$ of a pair are inserted consecutively by construction, and, for all i, μ_i is inserted before $\omega(\mu_{i-1})$ because it is a facet in X_j. By transitivity, μ is inserted before σ.

Case c'_i. The leading term of c'_i is σ_i. If $c'_i = c_i$, it is direct. Otherwise, by construction, $c'_i = c_i + \alpha \cdot \sigma$, $\alpha \neq 0$, and the chain c_i contains cells ν in its support such that $[\nu : \tau]^{\mathcal{A}_j \cup \{\tau,\sigma\}} \neq 0$, i.e., cofacets of τ in $\mathcal{A}_j \cup \{\tau, \sigma\}$. With a similar transitivity argument as above, τ (and σ) must consequently appear before such ν in the ordering of cells defined. The leading term of c'_i is then unchanged.

2. Let c_i be a chain such that $i \in F \sqcup H$. By Lemma 3.6, it is a direct calculation from the definition of c'_i that $\partial' c'_i = \partial c_i$. Consequently, Conditions (2) and (3) of Definition 3.2 are satisfied for those chains. The pairing $G \leftrightarrow H$ remains valid, because $\partial' c'_h = \partial c_h = c_g = c'_g$ for $g \leftrightarrow h$, $(g, h) \in G \times H$.

3. By definition, $\partial' c_\sigma = c_\tau$, their indices are in $H \times G$ and paired together. □

We now prove the compatibility condition:

Algorithm 2: Zigzag persistence algorithm for Morse filtrations

input : atomic zigzag filtration
$\mathcal{Z}: (\varnothing =) X_1 \longleftrightarrow X_2 \longleftrightarrow \cdots \longleftrightarrow X_{n-1} \longleftrightarrow X_m (= \varnothing)$
output: persistence diagram of $\mathcal{Z}$

set $M_{\mathcal{B}} \leftarrow \varnothing$;
for $j = 1 \dots m-1$ **do**
  if $X_j \xleftrightarrow{\sigma} X_{j+1}$, $\sigma \in X_j$ *critical* **then**
    use `zigzag_persistence_algorithm(`$M_{\mathcal{B}}$, $\mathcal{M}_j$, σ`)` to add or remove σ;
  end
  if $X_j \xleftarrow{\{\tau,\sigma\}} X_{j+1}$, (τ, σ) *Morse pair* **then**
    do nothing;
  end
  if $X_j \xrightarrow{\mathbb{1}} X_j \xleftarrow{\sigma} X_{j+1}$, σ *paired with* τ, τ *not removed* **then**
    set $M_{\mathcal{B}} \leftarrow M_{\overline{\mathcal{B}}}$ as described above;
    use `zigzag_persistence_algorithm(`$M_{\mathcal{B}}$,$\overline{\mathcal{M}}_j$,σ`)` to remove σ;
  end
end

Lemma 3.9. *The homology matrix $\overline{\mathcal{B}}$ at $\mathcal{A}_j \cup \{\tau, \sigma\}$ is compatible with $\overline{\mathcal{M}}_j$ in Diagram* (3.17)*.*

Proof. By hypothesis, $\mathcal{B} = \{c_0, \dots, c_{m-1}\}$ is a homology matrix at A_j, compatible with $\mathcal{M}_j$; let $\Omega\colon H(\mathcal{M}_j) \to \oplus_\ell \mathbb{I}[b_\ell; d_\ell]$ be a zigzag module isomorphism such that Ω_j sends $\{[c_f] : f \in F\}$ to the canonical basis of $\mathbb{F} \times \dots \times \mathbb{F}$.

Note that, none of the c'_i have an entry τ, except for c_τ, whose index is in G by construction. Consequently, by Properties 3.1, the chain map $\psi_{\tau,\sigma}\colon C(\mathcal{A}_j, \sigma, \tau) \to C(\mathcal{A}_j)$ simply cancels the entry σ in every c'_f, $f \in F$, and $\psi_{\tau,\sigma} c'_f = c_f$. Consequently, consider the chain maps between $\overline{\mathcal{M}}_j$ and $\mathcal{M}_j$ in Diagram (3.17). Each square commutes by virtue of Theorem 3.2 (for inclusions) and Lemma 3.4 (for $\varphi_{\tau,\sigma}$), and they induce an isomorphism $\Phi^*\colon H(\overline{\mathcal{M}}) \to H(\mathcal{M})$ of zigzag modules. The isomorphism $\Omega \circ \Phi^*\colon H(\overline{\mathcal{M}}) \to \oplus_\ell \mathbb{I}[b_\ell; d_\ell]$ sends $\{[c'_f] : f \in F\}$ to the canonical basis of $\mathbb{F} \times \dots \times \mathbb{F}$, and $\overline{\mathcal{B}}$ is compatible with $\overline{\mathcal{M}}$. □

In conclusion, for an input atomic zigzag operation $\mathcal{Z}$, with three atomic maps pictured in Diagrams (3.6), (3.7), and (3.8), the Morse algorithm for computing the zigzag persistence of $\mathcal{Z}$ is given in Algorithm 2, where the routine `zigzag_persistence_algorithm(`$M_{\mathcal{B}}$, $\mathcal{M}_j$, σ`)` is the zigzag persistence algorithm of [45] to handle forward or backward insertions of a single cell in a homology matrix $M_{\mathcal{B}}$ at complex $\mathcal{A}_j$, compatible with the filtration $\mathcal{M}_j$ (see Diagram (3.17)). Each iteration of the `for` loop turns a homology matrix $M_{\mathcal{B}}$ at complex $\mathcal{A}_j$, compatible with the filtration $\mathcal{M}_j$, into a homology matrix at complex $\mathcal{A}_{j+1}$, compatible with the filtration $\mathcal{M}_{j+1}$, where $\mathcal{M}_{j+1}$ is a zigzag Morse filtration for $\mathcal{Z}_{j+1}$, and $\mathcal{A}_j$ and $\mathcal{A}_{j+1}$ are respectively Morse complexes for X_j and X_{j+1}.

Implementation and complexity. We represent $\mathcal{B} = \{c_0, \dots, c_{m-1}\}$ by an $(m \times m)$-sparse matrix data structure $M_{\mathcal{B}}$. Assume computing boundaries and coboundaries in a Morse complex of size m is given by an oracle of complexity $\mathcal{C}(m)$. We implement the transformation $\mathcal{B} = \{c_0, \dots, c_{m-1}\} \to \overline{\mathcal{B}} = \{c'_0, \dots, c'_{m-1}, c_\tau, c_\sigma\}$ presented above by:

- computing the boundary $\partial'\sigma$ of σ in $\mathcal{A}_j \cup \{\tau, \sigma\}$, and the coboundary $\{\nu : [\nu : \tau]^{\mathcal{A}_j \cup \{\tau,\sigma\}} \neq 0\}$ of τ, in $O(\mathcal{C}(m))$ operations,

- adding columns c_τ and c_σ to the matrix in $O(m)$ operations,

- computing c'_i for all i, in $O(m^2)$. We can restrict the transformation to those c_i containing a cell of the coboundary of τ.

Consequently, we can perform the transformation above in $O(m^2 + \mathcal{C}(m))$ operations on a $(m \times m)$-matrix. The zigzag persistence algorithm of [10, 45] deals with forward and backward insertions of a single cell in $O(m^2)$ operations.

In conclusion, let $\overline{\mathcal{Z}} = (\overline{X}_i \overset{\Sigma_i}{\longleftrightarrow} \overline{X}_{i+1})_{i=1\dots 2k}$ be a general zigzag filtration (Diagram (3.5)), and let $\mathcal{M}$ be a zigzag Morse filtration as defined in Section 3.3, for a collection of Morse matchings $(\mathcal{A}_i, \mathcal{Q}_i, \mathcal{K}_i, \omega_i)$ on Σ_i, i odd. And:

- denote by n the total number of insertions and deletions critical cells in $\mathcal{M}$, and by $|\mathcal{A}_m|$ the maximal number of critical cells of a complex in $\mathcal{M}$,

- denote by N the total number of insertion and deletion of cells in $\overline{\mathcal{Z}}$, and by $|X_m|$ the maximal number of cells of a complex in $\overline{\mathcal{Z}}$.

Additionally, we compute Morse matchings using the fast coreduction algorithm of Mrozek and Batko [50]. Even if computing optimal Morse matchings is hard in general [42], this heuristic gives experimentally very small Morse complexes, with constant amortized cost per cell considered. We compute boundaries and coboundaries in a Morse complex $\mathcal{A}$ of a complex X by a linear traversal of the Hasse diagram of X. We store in memory the homology matrix of the Morse complex and the complex X. Consequently, the total cost of the algorithm is:

Theorem 3.10. *The persistent homology of $\overline{\mathcal{Z}}$ can be computed in*

$$O(n \cdot |\mathcal{A}_m|^2 + n \cdot |X_m| + N) \qquad \textit{(time)}$$
$$O(|\mathcal{A}_m|^2 + |X_m|) \qquad \textit{(space)},$$

In comparison, running the (practical) zigzag persistence algorithms [9, 10, 45] require $O(N \cdot |X_m|^2)$ operation and memory $O(|X_m|^2)$.

3.6 Experiments

In this section, we report on the performance of the zigzag persistence algorithm [45] with and without Morse reduction. The corresponding code will be avaible in a future release of the open source library `GUDHI` [58].

The following tests are made on a 64-bit Linux (Ubuntu) HP machine with a 3.50 GHz Intel processor and 63 GB RAM. The programs are all implemented in C++ and compiled with optimization level `-O2` and `gcc-8`. Memory peaks are obtained via the `/usr/bin/time -f` Linux command, and timings are measured

via the C++ `std::chrono::system_clock::now()` method. The timings for File IO are not included in any process time.

We run two types of experiments: homology inference from point clouds, using oscillating Rips zigzag filtrations, and levelset persistence of 3D-images. Both applications are described in the introduction in Section 3.1.

For homology inference, we use both synthetic and real data points. The point clouds `KlBt5`, `Spi3`, `Sph3`, and `To3` are synthetic samples of respectively the 5-dimensional Klein bottle, a 3-dimensional spiral wrapped around a torus, the 3-dimensional sphere, and the 3-dimensional torus. The point cloud `MoCh` and `By` are 3-dimensional measured samples of surface models: the MotherChild model, and the Stanford bunny model from the Stanford Computer Graphics Laboratory. The results with corresponding parameters are presented in Table 3.1.

	Without Morse reduction				With Morse reduction			
	N $\times 10^6$	$\lvert X_m\rvert$	time (s) cpx + pers	mem. peak (GB)	n $\times 10^6$	$\lvert \mathcal{A}_m\rvert$	time (s) cpx + pers	mem. peak (GB)
`KlBt5`	63.3	187096	403 + 2912	4.7	4.9	11272	394 + 448	1.1
`Spi3`	66.1	47296	435 + 4438	5.2	3.8	12810	382 + 343	1.1
`MoCh`	75.7	37709	460 + 4680	5.8	4.1	11975	450 + 318	1.1
`Sph3`	99.4	66848	430 + 3498	7.5	4.2	13432	665 + 853	1.3
`To3`	32.8	32903	117 + 847	2.4	1.6	7570	173 + 79	0.47
`By`	30.5	18764	153 + 951	2.3	5.2	8677	165 + 287	0.96

Table 3.1: Experimental results for the oscillating Rips zigzag filtrations. For each experiment, the maximal dimension is 10, $\mu = 4$, $\nu = 6$, except for `Sph3`, where $\nu = 7$. The number of vertices is 2000.

Levelset persistence is computed for a function $f : [0;1]^3 \to \mathbb{R}$, were f is a Fourier sum with random coefficients, as proposed in the `DIPHA` library[5] as representative of smooth data. The cube $[0;1]^3$ and function f are discretized into equal size voxels. For some tests, we also added random noise to the values of f. The values of $s_1 \leq s_2 \leq \ldots$ are spaced out equally such that $s_{i+1} - s_i = \epsilon$ for all i.The results with corresponding parameters are presented in Table 3.2.

In all experiments, timings are decomposed into 'cpx' for computation dedicated to the complex (construction, computation of (co)boundaries and of Morse matchings) and 'pers' for the computation of zigzag persistence.

Analysis of the results. The results show a significant improvement when using Morse reduction. For homology inference (Table 3.1), the total running time is between 2.5 and 6.7 times faster when using Morse reduction. Moreover, most of the computation is transferred onto the computation of the Morse complex, which opens new roads to improvement in future implementation, such as parallelization of the Morse reduction [39] (note that parallelization of the

[5] `github.com/DIPHA/dipha/blob/master/matlab/create_smooth_image_data.m`

		Without Morse reduction				With Morse reduction			
ϵ	max. noise	N $\times 10^6$	$\lvert X_m \rvert$	time (s) cpx + pers	mem. peak (GB)	n $\times 10^6$	$\lvert \mathcal{A}_m \rvert$	time (s) cpx + pers	mem. peak (GB)
0.1	0	34	286780	563 + 1725	3.9	6.3	48578	224 + 29	2.7
0.15	0	-	-	∞	-	9.3	115558	756 + 44	3.6
0.15	0.5	36.5	315305	417 + 3248	4.2	4.7	36144	221 + 59	2.8
0.2	0	-	-	∞	-	15.5	245360	2097 + 68	4.7
0.2	0.5	-	-	∞	-	5.6	56500	392 + 47	3.4

Table 3.2: Experimental results for the level set zigzag filtrations. For each experiment, the function $f : [0; 1]^3 \to [-14, 21]$ is applied to $129^3 = 2\,146\,689$ cells and the persistence is computed for maximal dimension 3. The interval size is denoted by ϵ. The infinity symbol ∞ corresponds to more than 12 hours computing time.

computation of zigzag persistence is not possible in the streaming model). In particular, the computation of zigzag persistence is from 3.3 to 14.7 times faster. The better performance is due to filtrations being from 5.8 to 23.5 times shorter than the original ones (quantities n vs N in the complexity analysis) and smaller complexes, from 2.2 to 16.6 times smaller with the Morse reduction (quantities $|\mathcal{A}_m|$ and $|X_m|$ in the complexity analysis). Note that the memory consumption with Morse reduction is from 2.4 and up to 5.6 times smaller, which is critical on complex examples in practice.

For levelset persistence (Table 3.2), the total running time is at least 9 times faster, and the computation of zigzag persistence alone is itself approximatively 55 times faster, when the computation without Morse reduction finished. On those cases that finish, the filtration size is from 5.5 to 7.7 times shorter with Morse reduction, the maximal size of the complexes between 5.9 and 8.7 times smaller, and the memory consumption around 50% more efficient.

Additionally, using Morse reduction allows to handle cases where the standard zigzag algorithm never finishes (more than 12 hrs). On these examples, the Morse algorithm does not take more than 36 min. for the entire computation.

These results agree with the complexity analysis (Section 3.5) where terms $O(|\mathcal{A}_m|^2)$ and $O(|X_m|^2)$ dominate both time and memory complexities.

3.7 Conclusion

We adapted the use of discrete Morse theory to the algorithms for zigzag filtrations. The difficulty was to handle the possibility of separation of a Morse pair to maintain a streaming like algorithm. We implemented our solution for a particular zigzag persistence algorithm [45] and had satisfying results for interesting applications.

But the limitations of the strategy is obvious: the size reduction does only improve the algorithm when a lot of cells are included simultaneously in the filtration. Otherwise enough pairs cannot be formed. Therefore we still need strategies for less particular zigzag filtrations.

Another interesting remaining question is if we could obtain better results by using another pairing method than discrete Morse theory. Finding a perfect Morse matching is NP-hard and so we have to rely on heuristics and an incomplete matching to fasten the pairings. But Morse matchings have to take account of adjacency, which is perhaps not a property we need when we know that we are removing a whole set of pairs. We could pair non-adjacent cells together under the condition that all cells "in between" will also be paired and removed. A first idea in this direction could be to look at the standard persistence pairs of the "sub-filtration" formed by the simultaneously included cells.

Chapter 4

Reduction and Average Complexity

This chapter is based on joint work with Michael Kerber[1].

4.1 Introduction

Motivation and problem statement. As stated in the introductory chapter, one reason of the success of (standard) persistent homology is its good practical performances: despite all efficient algorithms used in practice having a cubic worst case complexity, practical experiments are surprisingly fast and seem to tend to a more linear behavior. Even though the question of why is quite natural, the answer is not trivial to find and little to no work was done on the subject in this context.

The aim of this chapter will be to make a first step to study the average complexity of the persistence algorithm, or more precisely of the matrix reduction algorithm defined at the end of Section 1.3. For simplicity, we work only with $\mathbb{Z}_2$-coefficients.

First results. We restrict the study to four particular but common randomized filtration types over random point sets in low dimension. We made a series of experiments to measure several parameters such as the addition cost, number of addition or maximal column size. This gives us a good intuition of the process. Among others, good performances seems to be related to relative small reduced columns (columns with few non-zero elements). We show then two theoretical results corresponding to two of the filtration types. First, we show that the sum of the addition costs for persistence homology in dimension d for a lower star filtration is in the order of $O(d^2n)$, if n is the number of simplices involved. Secondly, we show that the expected complexity of a modified version of the shuffled filtration is at least an order of magnitude lower than its worst case complexity.

[1]Graz University of Technology, Austria – kerber@tugraz.at

(Quasi-)Related work. As mentioned, as to our knowledge, there is no work which was already done on this topic or even for the Gaussian elimination algorithm in general, on which the reduction algorithm is based on. But we can mention alternative algorithms which were designed for particular filtrations and which have a good worse complexity, as for example for height persistence in $\mathbb{R}^3$ which can be computed in $O(n \log n)$ time, where n is the number of simplices [28].

A similar algorithm to the reduction algorithm is the diagonalization algorithm, that is bringing a matrix in its Smith normal form, which can be used to compute the homology of a complex. If n is again the number of simplices of the complex and Δ its dimension, for "spare enough" matrices, it can be shown that the expected running time is in $O(\Delta n^2)$ [32]. The subtle but crucial difference from the persistence reduction algorithm is that the column order does not matter and the algorithm does not have to restrict it-self to left-to-right column additions.

We can also detour to probabilistic graph theory and the study of *phase transitions* with respect to the largest connected component in a random graph of n_0 vertices. A simple non-oriented graph can be seen as a 1-dimensional simplicial complex or as the 1-skeleton of a higher dimensional simplicial complex. Adding randomly edges over a set of vertices can therefore define a random filtrations. P. Erdős and A. Rényi [35] showed that the size of the largest connected component changes from $O(\log n_0)$ to $O(n_0)$ when the expected degree of the graph passes through 1. This phenomenon is referred to as the "emergence of the giant component", because we transition from a graph which is mainly acyclic (i.e., with high probability practically a forest) to a giant connected component with only few leftover trees. If this does not directly help to resolve our problem, it gives us a nice explanation for the aspect of the boundary matrix after reduction for such random filtrations: the first phase corresponds to a phase where adding an edges manly kills 0-homology, whereas the second phase mainly gives birth to new 1-dimensional cycle classes. This reflects on the reduced matrix, where all the non-trivial columns concentrate on the left of the matrix — and that in any dimension.

Outline. In Section 4.2, we define the different filtrations we will be working on. Then in Section 4.3, we present our experimental results. In Section 4.4.1, we show our result for lower star filtrations and in Section 4.4.2 the result for shuffled filtrations. Finally, in Section 4.5, we will discuss our results and future work.

4.2 Common randomized filtrations models.

Filtrations defined in practice are following particular construction processes because of useful properties or for technical reasons. This can influence on the expected running time of the persistence algorithm. Therefore, it makes sense to analyze the algorithm for particular types of filtrations instead as for all filtrations in general. In this section we will define several models of filtrations we want to study. We will restrict them to simplicial complexes of maximal dimension 2, i.e., the complexes will contain vertices, edges and triangles. All

filtrations are defined on a fixed vertex set of size n_0 and each possible simplex is included (i.e. a total of $\binom{n_0}{2}$ edges and $\binom{n_0}{3}$ triangles), one by one.

Lower Star filtrations. For this type of filtration, the vertices are added in random order and a simplex is added as soon as all its facets are included with priority to higher dimensional simplices (e.g. if both an edge and a triangle can be included, the triangle is included first). Moreover, when more than one same dimensional simplex can be included, they are included in lexicographical order defined on the vertices of the simplices in the order of appearance. The filtration is therefore determined by the number of vertices up to relabeling. The beginning of a lower Star filtration is shown in Figure 4.1.

This type of filtration is very specific, but it is easy to analyze and the number of bit operations can be expressed by a deterministic formula (see Section 4.4.1). It is probably the model which produces the lowest complexity and is thus interesting to use when one is free to choose the order of a subset of simplices. It is therefore also a good "control sample" to compare the other filtrations to.

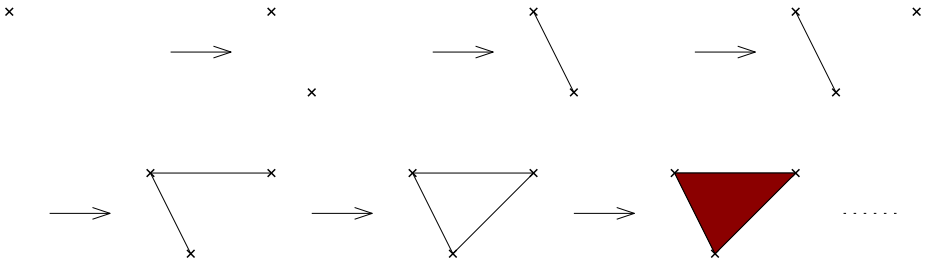


Figure 4.1: Example of the beginning of a lower star filtration.

Shuffled filtrations. We first include randomly all vertices, then randomly all edges and finally randomly all triangles. With "randomly" we mean that each simplex can be included with same probability. The simplices are included by dimension to ensure that we have a valid simplicial complex at each step of the filtration. By construction of the boundary matrix in a particular dimension, only the order of the simplices within same dimensional simplices matters for the running time complexity and therefore such filtrations are good representatives for all uniformly randomly picked filtrations.

Erdős-Rényi filtrations. This model is based on the model of Erdős-Rényi graphs. The Erdős-Rényi model defines the construction of a random graph with two fixed parameters: the number of vertices and a probability p. It starts with all vertices and then each edge is included with probability p independent of the other edges. In our case, we want to include every possible edge, therefore it is equivalent to choose with equiprobability a random order on the edges. We also want to include triangles: one is inserted in between the edges as soon as their boundary edges are inserted.

Rips filtrations. For n_0 random points in the unit square in $\mathbb{R}^2$, the edges are inserted in the order of their length. The triangles are inserted as soon as

their respective boundaries are inserted. Rips filtrations are the most commonly used filtration types among the four presented here. An example is shown in Figure 4.2.

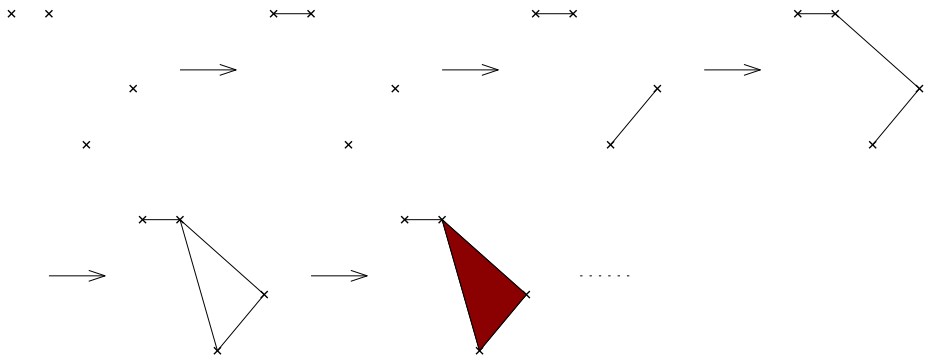


Figure 4.2: Example of the beginning of a Rips filtration.

4.3 Experimental results.

To have a first intuition, we experimentally measured different parameters for the four filtrations types defined previously. We only looked at the boundary matrix in dimension 1, which means the rows only represents the edges and the columns only the triangles. Therefore, we define the input size to be $n = \binom{n_0}{2} + \binom{n_0}{3}$, if n_0 is the number of vertices. Let $m = \binom{n_0}{2}$ be the number of edges and thus the height of the matrix. The width of the matrix is then in the order of $O(m^{\frac{3}{2}})$. The experiments were repeated for an increasing n_0, and for a particular n_0, the result is the average over 50 to 100 different runs — except for the lower star filtration which does not contain any real probabilistic aspect: one run is sufficient. We recall that we work only with coefficients in $\mathbb{Z}_2$ and therefore the matrix contains only 0's and 1's.

Bit addition operations. We define as a *bit addition operation* the switch of a matrix cell from 0 to 1 or vice versa when adding a column c' on another column c during the reduction process. In our experiments, the number of bit addition operations which results from one column addition is then the number of 1's in c'.

The first parameter we measured was the total number A of bit addition operations during the reduction process. The running time complexity of the reduction algorithm is mainly defined by the number and the cost of a column addition and thus, A is a good indicator. The solid lines in Figure 4.3 shows the evolution of A with respect to n with a logarithmic scale for the y-axis. We distinguish for all four filtration types a distinct polynomial growth. So, we used linear regression to infer the empirical asymptotic behavior of A, which is represented by the dotted lines in Figure 4.3:

- Lower star filtrations: $O(n^{1.10})$,
- Rips filtrations: $O(n^{1.31})$,

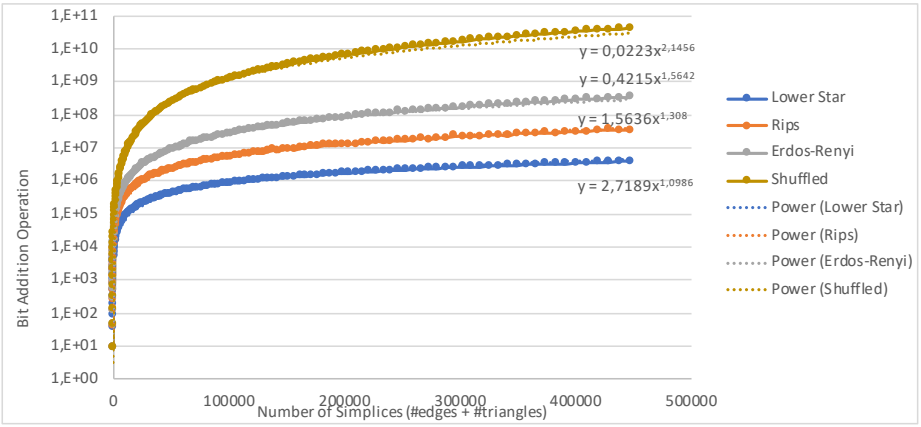


Figure 4.3: Diagram representing the accumulated number of bit addition operations (y-axis, logarithmic scale) with respect to the number of simplices (x-axis). From top to bottom: shuffled filtration, Erdős-Rényi filtration, Rips filtration, lower star filtration.

- Erdős-Rényi filtrations: $O(n^{1.56})$,
- Shuffled filtrations: $O(n^{2.15})$.

Those results are rather positive: a common filtration type as the Rips filtration seems to tend more to a linear than a quadratic behavior and even if one picks randomly a filtration, we can expect it to be better than the worst case (even though the dispersion is not clear).

To better understand the evolution of A, we then looked at the number of column additions made and at the maximal number of bit addition operations for a single column addition. We define the *size* of a column to be the number of 1's in this column.

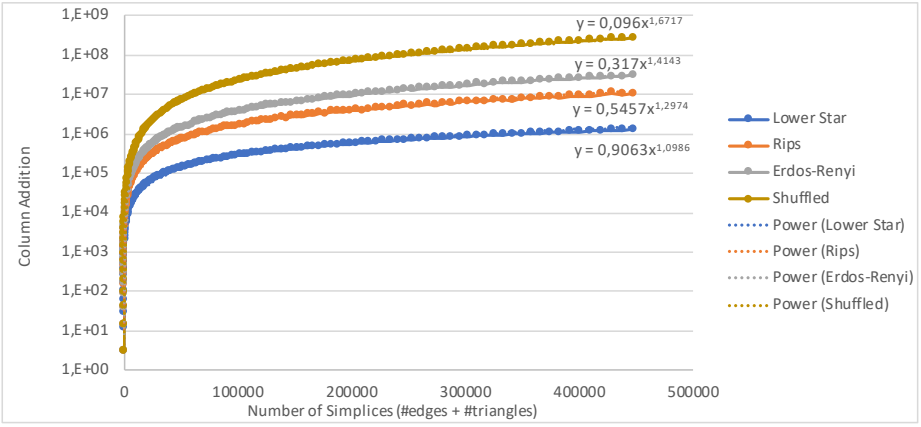


Figure 4.4: Diagram representing the number of column additions (y-axis, logarithmic scale) with respect to the number of simplices (x-axis). From top to bottom: shuffled filtration, Erdős-Rényi filtration, Rips filtration, lower star filtration.

Number of column additions. The solid lines in Figure 4.4 shows the evolution of the number of column additions with respect to n with a logarithmic scale for the y-axis. Again, we used linear regression to infer the empirical asymptotic behavior, which is represented by the dotted lines in Figure 4.4:

- Lower star filtrations: $O(n^{1.10})$,
- Rips filtrations: $O(n^{1.30})$,
- Erdős-Rényi filtrations: $O(n^{1.41})$,
- Shuffled filtrations: $O(n^{1.67})$.

For the lower star and Rips filtration, the result is almost the same than for A. The reason is obvious for the lower star filtration, but for the Rips filtration, this could indicate that the general size of the columns is small enough to not weight to much in the complexity.

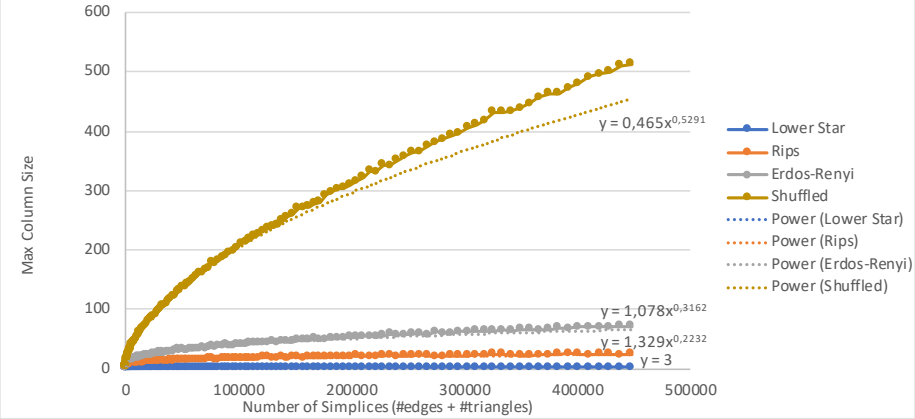


Figure 4.5: Diagram representing the maximal size of a reduced column (y-axis) with respect to the number of simplices (x-axis). From top to bottom: shuffled filtration, Erdős-Rényi filtration, Rips filtration, lower star filtration.

Maximal size of a column. We measured the maximum size among all columns in the completely reduced matrix. Its evolution with respect to n is shown by the solid lines in Figure 4.5. The dotted lines represent the empirical asymptotic behavior of the maximal size:

- Lower star filtrations: 3 (constant).
- Rips filtrations: $O(n^{0.22})$,
- Erdős-Rényi filtrations: $O(n^{0.32})$,
- Shuffled filtrations: $O(n^{0.53})$.

There is a clear difference between the Erdős-Rényi filtration and the Shuffled filtration, even though the edges were randomly ordered in both cases. That the triangles kill as soon as possible the 1-dimensional cycles in the Erdős-Rényi filtration (similarly as in the lower star filtration) seems to keep the column rather small. This could also be one of the reasons for the tendencies of the maximal size in the Rips filtration.

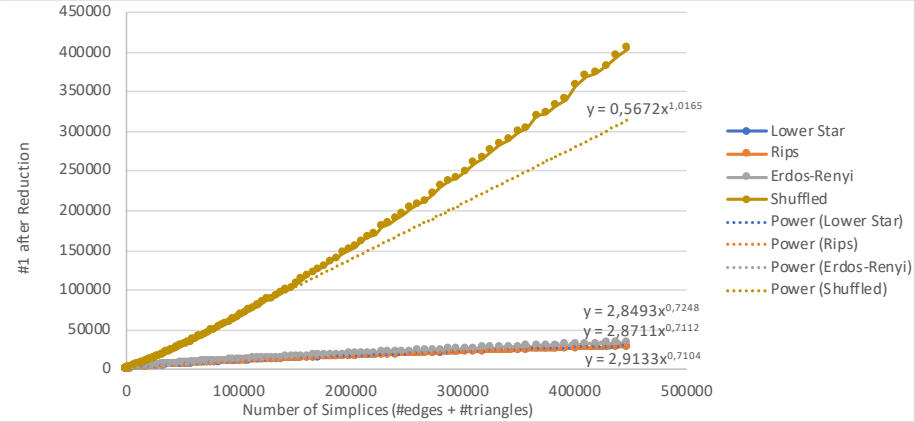


Figure 4.6: Diagram representing the total number of 1's in the final reduced matrix (y-axis) with respect to the number of simplices (x-axis). From top to bottom: shuffled filtration, Erdős-Rényi filtration, Rips filtration, lower star filtration.

Number of 1's in the reduced matrix. Finally, we counted the number of 1's in the final reduced matrix, which value has, as for the precedent parameters, a very low coefficient of variation over the different runs. This gives us a global idea of how big a column is expected to become, or how likely it is that its size is close to the maximal size measured before. Again, the evolution with respect to n is shown by the solid lines in Figure 4.6 and the dotted lines represent the empirical asymptotic behavior:

- Lower star filtrations: $O(n^{0.71})$,
- Rips filtrations: $O(n^{0.71})$,
- Erdős-Rényi filtrations: $O(n^{0.72})$,
- Shuffled filtrations: $O(n^{1.02})$.

The results are linear, if not sublinear, with respect to n. But more interesting is that the average amount of 1's in the Rips and Erdős-Rényi filtration are very close to the amount of 1's in the lower star filtration, which has the minimal amount of 1's: all the non-trivial column have exactly size 3 and the amount of non- trivial columns is the same than for all other three filtration types. This seems to indicate, that, despite a few bigger columns, most columns should have a size $O(1)$ with respect to n or at least with a power near 0.

Sum of squared bar lengths. Another interesting parameter is the sum Σ of the square of the length of each bar in the final barcode, because it is well-known that the time complexity of the reduction algorithm is at most in the order of Σ. If the sum in the experiments is close to the total number of bit addition operations, then searching for the expected value of Σ gives another interesting approaches to the question. Again, the experimental results are shown in Figure 4.7. The evolution with respect to n is shown by the solid lines and the dotted lines represent the empirical asymptotic behavior:

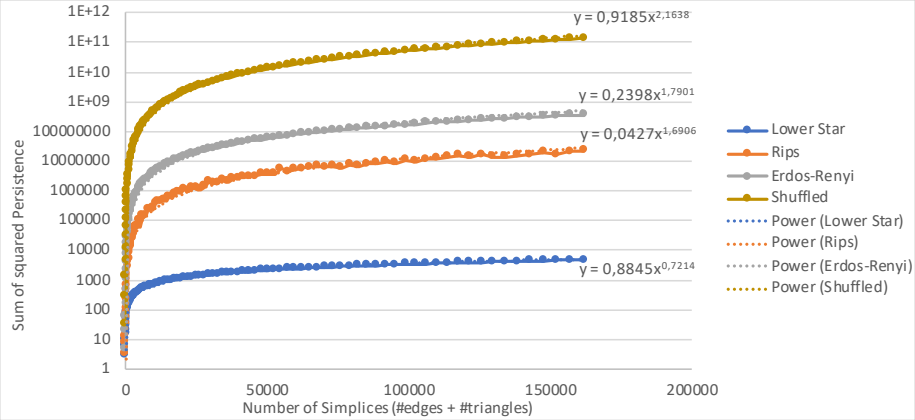


Figure 4.7: Diagram representing the sum of squared bar lengths (y-axis) with respect to the number of simplices (x-axis). From top to bottom: shuffled filtration, Erdős-Rényi filtration, Rips filtration, lower star filtration.

- Lower star filtrations: $O(n^{0.72})$,
- Rips filtrations: $O(n^{1.69})$,
- Erdős-Rényi filtrations: $O(n^{1.79})$,
- Shuffled filtrations: $O(n^{2.16})$.

4.4 Theoretical results.

We focused on the lower star filtrations and the shuffled filtrations and on persistence in a fixed dimension for our theoretical analysis. Therefore we restrict the boundary matrix to the necessary simplices: a boundary matrix for dimension d only records $(d+1)$-simplicies as its column and d-simplicies as its rows. We recall that a column which is non-trivial after reduction is called a *negative* column and *positive* otherwise.

4.4.1 Lower star filtrations

Even if the construction of a lower star filtration was described for maximal dimension 2, the way to extent it to higher dimension is obvious. So, we look at a boundary matrix M for dimension $d > 0$ of a lower star filtration of maximal dimension $\Delta \geq d$. Let R be M after reduction. We can prove the following lemmas:

Lemma 4.1. *A positive d-simplex τ, $d > 0$, in a lower star filtration always creates at least one d-cycle which is the boundary of a $(d+1)$-simplex. Moreover the d^{th}-homology group of the current complex was trivial just before the insertion of τ.*

Proof. We prove the statement by induction on the dimension $d > 0$.

Base case: $d = 1$. Let e be a positive edge inserted at step i. Let v be the last vertex included before step i. By construction, v is in the boundary

of e and just before the inclusion of v, every possible edge and triangle for the current set of vertices is already included. Let C_0 be a cycle created by the insertion of e. Note that v has to be in C_0. If C_0 has length 3, then it is the boundary of a triangle t and t will be inserted before the next edge after step i. So the statement holds. Assume C_0 has length greater than 3. Then it contains a vertex $w \neq v$ with two neighbors u_1 and u_2 in C_0 different from v. All three vertices were included before v, so the triangle $\{u_1, w, u_2\}$ and its boundary are already included. So e also creates the cycle C_1 homological to C_1, such that C_1 is identical to C_0 except that the edges u_1w and wu_2 are replaced by the edge u_1u_2. The length of C_1 is strictly shorter than the length of C_0 and if it still greater than 3, we can repeat the same argument to construct a shorter homological cycle C_2. So, we can easily show with induction that e is contained in the boundary of a triangle, which is not included yet. A similar argument can be applied to show that every non-trivial 1-cycle created between the insertion of v and e has to be killed before the insertion of e.

Induction: assume that, for a $d > 0$, the statement is true for every d-simplex. We show that is it also true for every $(d+1)$-simplex.

Let σ be a $(d+1)$-simplex and τ the last d-complex inserted before σ. Say that τ is inserted at step i. By construction, τ is a positive simplex which created the boundary of σ, otherwise σ would not have been included at this point. We will show that if σ is positive, then it also creates the boundary of a $(d+2)$-simplex. The insertion of τ could have created boundaries for other $(d+1)$-simplices than σ. So, let Σ be the set of $(d+1)$-simplices whose boundaries appear with the inclusion of τ. All those boundaries are in the same cycle class created by τ, because it is the only non-trivial class by induction hypothesis. Again by construction, at step $i+1$ one of the simplices, say σ_0, in Σ has to be inserted and this first simplex will kill the cycle class created by τ. So, if σ is positive, $\sigma \in \Sigma \setminus \sigma_0$. Before including any other d-simplex in the filtration, every remaining simplex in Σ has to be included first and no new non-trivial 1-cycle can be created. So, every simplex in $\Sigma \setminus \{\sigma_0\}$ will be positive. Therefore, assume without loss of generality that σ is the j^{th} simplex in $\Sigma \setminus \{\sigma_0\}$ to be included. Then σ_0 and σ have exactly $d+1$ vertices in common, to be exact, the $d+1$ vertices of τ. Let $u_0 = \sigma_0 \setminus \sigma_j$ and $u_j = \sigma_j \setminus \sigma_0$. Both u_0 and u_j were inserted before v by assumption, so any remaining $(d+1)$-simplex formed by $d+2$ vertices in $\tau \cup \{u_0, u_j\}$ is also already included. Thus, with σ_0 and σ we have all the facets of the $(d+2)$-simplex μ formed by all the vertices of $\tau \cup \{u_0, u_j\}$. Moreover this means that μ will be included by construction before the next $(d+1)$-simplex is inserted. This being true for every j, we can guarantees that the $(d+1)^{th}$-homology group will be trivial before the insertion of any $(d+1)$-simplex in Σ. □

Lemma 4.2. *Every negative column in R did not undergo any column addition.*

Proof. By Lemma 4.1, a positive d-simplex produces a new d-cycle which is the boundary of at least one $(d+1)$-simplex. This immediately leads by construction to the inclusion of one of the corresponding $(d+1)$-simplices. Therefore every new cycle class will always be killed at time "birth+1". If a positive d-simplex τ leads to the inclusion of the negative $(d+1)$-simplex σ in the next step, then τ will the youngest simplex in the boundary of σ and thus will correspond to its pivot. Because τ appears then for the first time in a boundary, the column corresponding to σ will not be modified by any column addition. □

Remark 4.1. The lexicographical order does not intervene in the proofs of Lemma 4.2 and 4.1. Both lemmas are still true, if the simplices of same dimension which can be included simultaneously are included randomly.

This is not true for the next Lemma. If the d-simplices are included randomly, then a column can have up to m column additions, if m is the number of $(d-1)$-simplices included until the position of the column.

Lemma 4.3. *Every positive column in R was obtain from exactly $d+2$ column additions.*

Proof. By Lemma 4.2, every negative column was not modified by any column addition and therefore a negative column still represents exactly the boundary of the corresponding $(d+1)$-simplex. By Lemma 4.1, a positive $(d+1)$-simplex τ gives birth to a $(d+1)$-cycle which is exactly the boundary of a $(d+2)$-simplex. Let v_0 be the first vertex included in the filtration. Because of the lexicographical order, every negative simplex will contain v_0, whereas the positive simplices are exactly those not containing v_0. So, the boundary created by τ is either formed of only positive simplices or of exactly τ and $d+2$ negative simplices. In the latter case, because the negative column are in their initial state, we will need exactly $d+2$ column addition from the corresponding $d+2$ negative columns to reduce the column of τ. If all simplices in the boundary are positive, let $\tau = \tau_0, \tau_1, \ldots, \tau_{d+2}$ be those simplices and σ the $(d+2)$-simplex with the positive boundary. Each τ_i, $0 \geq i \geq d+2$, is also included in a boundary of a $(d+2)$-simplex σ_i composed of only negative simplices except of τ_i. Then τ_i will correspond to the pivot of σ_i. Because our coefficients are in $\mathbb{Z}_2$, the sum of every σ_i is equal to σ. So, again, when reducing the column corresponding to τ, we will have to add all $d+2$ different negative columns appearing in the boundaries of the σ_i's. □

Therefore, the next theorem follows:

Theorem 4.4. *Let M be the boundary matrix for dimension d of a lower star filtration with n_0 vertices and maximal dimension $\Delta \geq d$. Let $d_* = d+2$. The number of bit addition operations of the reduction algorithm M is exactly:*

$$d_*^2 \cdot \sum_{k=0}^{d_*} (-1)^{d_*-k} \binom{n_0}{k}.$$

Proof. By Lemma 4.2, every negative column in the reduced matrix was obtained from zero bit addition operations and therefore their size is exactly $d+2 = d_*$. Thus, by Lemma 4.3, the sum of the bit addition operations for a single positive column is d_*^2. By a simple recursive argument, we can show that the number of positive columns is $\sum_{k=0}^{d_*} (-1)^{d_*-k} \binom{n_0}{k}$. Thus the result follows. □

Let $n = \binom{n_0}{d+2} + \binom{n_0}{d+1} = O(n_0^{d+2})$ be the total number of simplices involved in Theorem 4.4. Then the complexity is in the order of $\Theta(d^2 \cdot n)$.

4.4.2 Shuffled filtrations

For shuffled filtrations, we focus on persistence in dimension 1. Let M be the boundary matrix for dimension 1 of some shuffled filtrations with n_0 vertices and

$m = \binom{n_0}{2}$ edges. Keeping track of the evolution of the columns through additions in M is way more difficult than for the lower star filtration. Therefore we will first look at some simplified model for a boundary matrix and the reduction algorithm.

Expended matrix. Define a 3-*column* in a matrix as a column with exactly three non-zero entries. For instance, every column in M is a 3-column. We define $\tilde{M}$ as a $m \times \binom{m}{3}$-matrix over $\mathbb{Z}_2$ with m rows and all possible 3-columns arranged in a random order (every column order is equiprobable). $\tilde{M}$ can be interpreted as the boundary matrix of a cell complex consisting of one vertex v, m distinct self-loops attached to v, and $\binom{m}{3}$ 2-cells bounded by three of the self-loops. Note that $\tilde{M}$ contains significantly more columns than M with the same number of edges.

If we apply the reduction algorithm 1 to $\tilde{M}$, the worst case complexity is in the order of $O(m^3) \cdot O(m) \cdot O(m) = O(m^5)$.

But to analyze $\tilde{M}$, we also define a variation of the reduction algorithm: If during the reduction, we encounter an input column c with the same pivot as a previously reduced column c', we first reduce c and then replace c' with the unreduced version of c. The idea is that c has initially only 3 non-zero entries, so subsequent reduction steps are less expensive. When enough reduced columns have been replaced, the reduction of the remaining columns becomes considerably cheaper than in the naive version. We call this new algorithm the *expended reduction algorithm.*

Lemma 4.5. *The expended reduction algorithm results in the same persistence pairs than the standard reduction algorithm.*

Proof. Let $\tilde{M}_k$ be the matrix $\tilde{M}$ restricted to the k first columns. Reducing the $(k+1)^{th}$ boundary $\partial\sigma$ with columns from left-to-right has the purpose to express the new boundary with a linear combination of columns to the left, i.e. basis cycles which already exists in the "prefix" complex represented by $\tilde{M}_k$. If $\partial\sigma$ can be completely expressed by the basis in $\tilde{M}_k$, than a higher dimensional cycle is born. Otherwise, the remaining is a cycle which is killed and becomes a basis element in $\tilde{M}_{k+1}$. But which base of $\tilde{M}_k$ is used does not matter, the persistence pairs are known to be always the same. Let c be the column which has the same pivot than $\partial\sigma$ in $\tilde{M}_k$. Switching c with $\partial\sigma$ after reducing $\partial\sigma$ is simply a base change in $\tilde{M}_{k+1}$ by replacing an element with an existing element which is clearly linear independent of the other base elements without c. □

Contrary to M, $\tilde{M}$ has enough columns for the expended reduction algorithm to make a difference in the average, as we will see in the next lemma. For this purpose, we have to note that in the reduced version of $\tilde{M}$, we will have exactly m different pivots, one for each row. We say that a pivot p is *contained* in a column c when the pivot of c is p.

Lemma 4.6. *Let $t > 2$ be a row index and X the random variable representing the smallest c, such that all pivots from t to m are contained in the columns 1 to c in $\tilde{M}$. Then the expected value of X:*

$$\mathbb{E}[X] \leq 5 \cdot \frac{\binom{m}{3}}{t}.$$

Proof. Let A^i_j be the event corresponding to the existence of a pivot in row i between columns 1 and j and $N = \binom{m}{3}$ be the number of columns. Then:

$$\mathbb{E}[X] = \sum_{j=0}^{N-1} P(X \geq j+1) = \sum_{j=0}^{N-1} P(\bigcup_{i=t}^{m} \neg A^i_j) \;\leq\; \sum_{j=0}^{N-1}\sum_{i=t}^{m} P(\neg A^i_j).$$

Let $K_i = \binom{i-1}{2}$ be the number of columns with pivot i. Then:

$$P(\neg A^i_j) = \frac{\binom{N-K_i}{j}}{\binom{N}{j}} = \prod_{\ell=0}^{j-1} 1 - \frac{K_i}{N-\ell}.$$

Therefore:

$$\begin{aligned}
\mathbb{E}[X] &\leq \sum_{j=0}^{N-1}\sum_{i=t}^{m}\prod_{\ell=0}^{j-1} 1 - \frac{K_i}{N-\ell} \\
&\leq \sum_{j=0}^{N-1}\sum_{i=t}^{m}(1-\frac{K_i}{N})^j = \sum_{i=t}^{m}\sum_{j=0}^{N-1}(1-\frac{K_i}{N})^j \\
&\leq \sum_{i=t}^{m}\sum_{j=0}^{\infty}(1-\frac{K_i}{N})^j = \sum_{i=t}^{m}\frac{1}{1-(1-\frac{K_i}{N})} = \sum_{i=t}^{m}\frac{N}{K_i}
\end{aligned}$$

We have:

$$\sum_{i=t}^{m}\frac{1}{K_i} = \sum_{i=t}^{m}\frac{1}{\binom{i-1}{2}} = \sum_{i=t}^{m}\frac{1}{(i-1)(i-2)}.$$

For $t > 3$, $\frac{1}{(i-1)(i-2)}$ is strictly decreasing for each i. So, we can upper bound the sum with the corresponding integral:

$$\sum_{i=t}^{m}\frac{1}{(i-1)(i-2)} \leq \int_{t-1}^{m}\frac{1}{(i-1)(i-2)}\,\mathrm{d}i = \log\left(\frac{m-2}{m-1}\right) - \log\left(\frac{t-3}{t-2}\right).$$

Because $\frac{m-2}{m-1}$ and $\frac{t-3}{t-2}$ are both smaller than 1:

$$\log\left(\frac{m-2}{m-1}\right) - \log\left(\frac{t-3}{t-2}\right) \leq -\log\left(\frac{t-3}{t-2}\right) = \log\left(\frac{t-2}{t-3}\right).$$

Using the property that $\log x \leq x-1$ for $x > 0$:

$$\log\left(\frac{t-2}{t-3}\right) \leq \frac{t-2}{t-3} - 1 = \frac{1}{t-3} \leq \frac{4}{t}.$$

When $t = 3$:

$$\sum_{i=3}^{m}\frac{1}{(i-1)(i-2)} = \frac{1}{2} + \sum_{i=4}^{m}\frac{1}{(i-1)(i-2)} \leq \frac{1}{2} + \frac{4}{4} = \frac{3}{2} \leq \frac{5}{3} = \frac{5}{t}.$$

So, $\mathbb{E}[X] \leq 5 \cdot \frac{N}{t}$. □

From this lemma we can deduce a first upper-bound:

Lemma 4.7. *Let t be a fixed row index in M. The time complexity of the expended reduction algorithm has as upper-bound $O(\frac{m^5}{t} + t^2m^3)$ in expectation.*

Proof. Let X be the random variable representing the smallest c, such that all pivots from t to m are contained in the columns 1 to c in $\tilde{M}$. In Figure 4.8, $\tilde{M}$ is represented after reduction: we can distinguish two types of columns for a fixed row index t. The bigger columns in the figure are the reduced columns which pivot is larger or equal to t. We know that they appear before column X and that they all have size 3 when we reduce columns after index X. The smaller columns represented in the figure have pivots smaller than t.

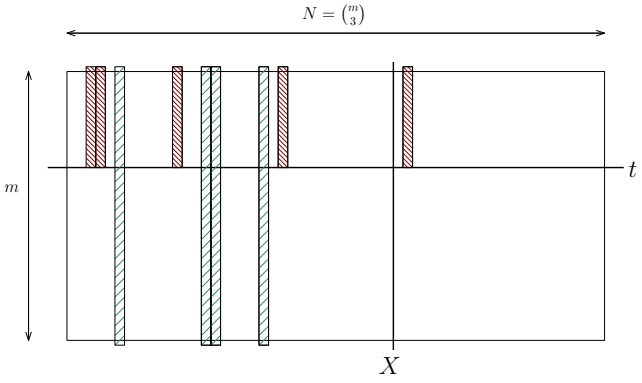


Figure 4.8: Visualization of $\tilde{M}$ after reduction.

While reducing columns placed before column X, we do not have any guarantee on the already reduced columns, so we have to assume the worst case: for X columns we have to make a maximum of m additions of columns of maximal size m. While reducing columns placed after column X, we add the two different kind of columns: for $N - X$ columns we have to make maximal t additions of maximal size t and $m - t$ additions of size exactly 3.

This gives us a maximal cost of:

$$\mathcal{C} = Xm^2 + (N - X)(t^2 + 3(m - t)).$$

By Lemma 4.6, $\mathbb{E}[X] = O(\frac{m^3}{t})$, so the expected value of $\mathcal{C}$ is:

$$\begin{aligned}
\mathbb{E}[\mathcal{C}] &= \mathbb{E}[X]m^2 + (N - \mathbb{E}[X])(t^2 + 3(m - t)) \\
&= O(\frac{m^3}{t})m^2 + O(m^3 - \frac{m^3}{t})(t^2 + O(m - t)) \\
&= O(\frac{m^5}{t}) + O(\frac{m^3t^3 + m^4t}{t}) \\
&= O(\frac{m^5}{t} + m^3t^2 + m^4)
\end{aligned}$$

It is not difficult to verify that $\frac{m^5}{t} + m^3t^2$ reaches its minimum for $t = O(m^{\frac{2}{3}})$ with a value of $O(m^{\frac{13}{3}})$ which is higher than m^4. So, $\mathcal{C} = O(\frac{m^5}{t} + m^3t^2)$ in expectation. □

As seen in the proof this upper-bound gives us at best $O(m^{\frac{13}{3}})$ for $t = O(m^{\frac{2}{3}})$. But instead of using just one threshold t, we could refine the analysis done in the proof by using a sequence of thresholds: $\mathbb{E}[X]$ will become smaller when t increases in value, such that we can improve the analyses for columns placed before X as we can see in Figure 4.9. This way we can show the next theorem.

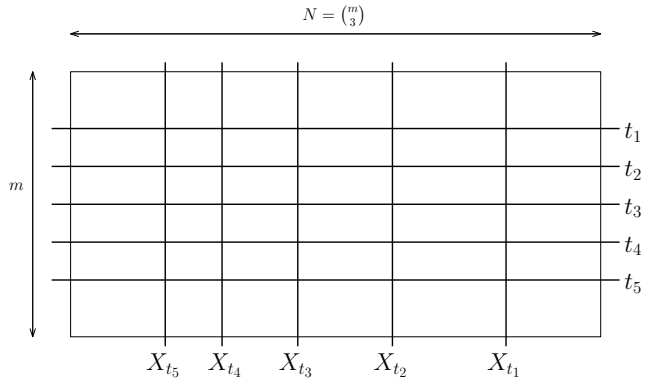


Figure 4.9: Visualization of $\tilde{M}$ with a sequence of thresholds t_1.

Theorem 4.8. *Let $\epsilon > 0$ be a constant. The time complexity of the expended reduction algorithm is $O(m^{4+\epsilon})$ in expectation.*

Proof. Let $\epsilon > 0$ and $k = \lceil \log_2(\frac{1}{2\epsilon} + \frac{1}{2}) \rceil$. For large enough $m >> k$, define the row index $t_{k-s} = m^{1-(2^{s+1}-1)\epsilon}$ and its corresponding random variable X_{k-s} as defined in Lemma 4.6, for $s \in \{0, ..., k-1\}$. Let $e_s = \mathbb{E}[X_s]$ for each $s \in \{1, ..., k\}$, then $t_1 < t_2 < ... < t_k$ and $e_1 > e_2 > ... > e_k$, by Lemma 4.6. Then, during the expended reduction algorithm, we have for the columns:

– from 1 to X_k: m column additions of maximal size m,

– from $X_s + 1$ to X_{s-1}, for $s \in \{1, ..., k\}$: t_s additions of maximal size t_s and $(m - t_s)$ of size 3,

– from $X_1 + 1$ to N: t_1 additions of maximal size t_1 and $(m - t_1)$ of size 3.

Therefore, the complexity is:

$$\mathcal{C} = O(X_k m^2 + \sum_{\ell=1}^{k-1} X_\ell t_{\ell+1}^2 + m \sum_{\ell=1}^{k-1} X_\ell + m^3 t_1^2 + m^4).$$

By Lemma 4.6, $e_s = O(\frac{m^3}{t_s})$ for $s \in \{1, ..., k\}$ and by replacing each t_s by its

value, we have:

$$\begin{aligned}\mathbb{E}[\mathcal{C}] &= O(e_k m^2 + \sum_{\ell=1}^{k-1} e_\ell t_{\ell+1}^2 + m\sum_{\ell=1}^{k-1} e_\ell + m^3 t_1^2 + m^4) \\ &= O(m^{4+\epsilon} + \sum_{\ell=1}^{k-1} m^{4+\epsilon} + m\sum_{\ell=1}^{k-1} m^{2+(2^{\ell+1}-1)\epsilon} + m^{4+\epsilon} + m^4) \\ &= O((k+1)m^{4+\epsilon} + m^3\sum_{\ell=1}^{k-1} m^{(2^{\ell+1}-1)\epsilon} + m^4).\end{aligned}$$

It is easy to verify that $\sum_{\ell=1}^{k-1} m^{(2^{\ell+1}-1)\epsilon} = O((k-1)m^{1+\epsilon})$. Moreover, $k << m$. Thus, $\mathcal{C} = O(m^{4+\epsilon})$ in expectation. □

4.5 Discussion.

The experimental results are pointing to nice distinct polynomial behaviors for each of the filtration types, way lower than cubic and therefore support the existence of a sub-cubical average complexity. But on the theoretical side, some work has still to be done.

Lower star filtration. The time complexity for lower star filtrations shows us that a nearly linear behavior is possible, but it is obviously quite restrictive. On the other hand, it is easy to construct, and even within a particular framework with a need of more sophisticated or generalized filtration types, it is not unusual that we are free to choose the order of some subset of simplices. For example, when a bunch of simplices are added simultaneously. Even if working on a subset is not the same than working on a complex — because it does not have to be a complex — it supports the effectiveness of the lexicographical order, which is often used heuristically.

Shuffled filtration. A shuffled filtration is a good representative for a randomly picked filtration and therefore particularly interesting to analyze. The expended version seems to be a first step in this direction, even if it (seemingly) cannot be directly linked to the time complexity of a real shuffled filtration. Another approach we will be looking at is to upper-bound the expected value of the sum of squared bar length in the barcode, even though it is yet uncertain if this approach is really easier. To simplify the problem, we could also look at a matrix which columns where randomly produced with a certain distribution: we would toss a coin for each cell of a column to decide if we put a 1 or a 0 inside. Those column won't represent any triangles anymore, but choosing a probability $p = \frac{3}{m}$, with m the number of rows, yields 3 as the expected number of 1's. The problem becomes even more simpler, if we concentrate on the second half of the matrix: we observed that all negative columns appear quite soon in the process, therefore we can assume that every pivot is already present. Using the observation that the number of 1's in the total of negative columns is linear to the number of simplices, we can use a new q-distribution to simulate those negative columns, with $q = \frac{1}{\sqrt{m}}$. But even then, the resulting expectations are

quite tedious to approximate into a readable nice result as for the expended shuffled filtration.

Rips filtration. We do not have any result for the Rips filtration yet, but it is particular interesting for its geometrical aspect. How much does the use of the underlying metric influence the expected running time of the reduction algorithm and on which aspects does it operate?

Acknowledgment. The author was supported by the Austrian Science Fund (FWF) grant number P 29984-N35.

Bibliography

[1] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and Compress: Computing Persistent Homology in Chunks. In *Topological Methods in Data Analysis and Visualization III*, Mathematics and Visualization, pages 103–117. Springer, 2014.

[2] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed Computation of Persistent Homology. In *Algorithm Engineering and Experiments (ALENEX)*, pages 31–38, 2014.

[3] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat — Persistent Homology Algorithms Toolbox. *Journal of Symbolic Computation*, 78:76–90, 2017.

[4] Ulrich Bauer and Michael Lesnick. Induced Matchings and the Algebraic Stability of Persistence Barcodes. *Journal of Computational Geometry*, 6(2):162–191, 2015.

[5] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The Compressed Annotation Matrix: An Efficient Data Structure for Computing Persistent Cohomology. *Algorithmica*, 2014.

[6] Jean-Daniel Boissonnat and Clément Maria. The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. *Algorithmica*, 70:406–427, 2014.

[7] Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek. Strong Collapse for Persistence. In *European Symposium on Algorithms (ESA) 2018*, pages 67:1–67:13, 2018.

[8] Magnus B. Botnan and Gard Spreemann. Approximating Persistent Homology in Euclidean Space through Collapses. *Applied Algebra in Engineering, Communication and Computing*, 26(1-2):73–101, 2015.

[9] Gunnar E. Carlsson and Vin de Silva. Zigzag Persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[10] Gunnar E. Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag Persistent Homology and Real-valued Functions. In *Symposium on Computational Geometry (SoCG)*, pages 247–256, 2009.

[11] Gunnar E. Carlsson, Afro J. Zomorodian, Anne D. Collins, and Leonidas J. Guibas. Persistence Barcodes for Shapes. *International Journal of Shape Modeling*, 11(2):149–187, 2005.

[12] Joseph Minhow Chan, Gunnar E. Carlsson, and Raul Rabadan. Topology of Viral Evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

[13] Huang-Wei Chang, Sergio Bacallado, Vijay S. Pande, and Gunnar E. Carlsson. Persistent Topology and Metastable State in Conformational Dynamics. *PLoS ONE*, 8, 04 2013.

[14] Frédéric Chazal, Vin de Silva, Marc Glisse, and Steve Y. Oudot. *The Structure and Stability of Persistence Modules*. Springer Briefs in Mathematics. Springer, 2016.

[15] Frédéric Chazal, Leonidas J. Guibas, Steve Y. Oudot, and Primoz Skraba. Persistence-Based Clustering in Riemannian Manifolds. *Journal of the ACM*, 60(6):41:1–41:38, November 2013.

[16] Chao Chen and Michael Kerber. Persistent Homology Computation with a Twist. In *European Workshop on Computational Geometry (EuroCG)*, pages 197–200, 2011.

[17] Chao Chen and Michael Kerber. An Output-sensitive Algorithm for Persistent Homology. *Computational Geometry*, 46(4):435–447, 2013.

[18] Aruni Choudhary, Michael Kerber, and Sharath Raghvendra. Polynomial-Sized Topological Approximations Using The Permutahedron. In *32nd International Symposium on Computational Geometry (SoCG)*, pages 31:1–31:16, 2016.

[19] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of Persistence Diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[20] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and Vineyards by Updating Persistence in Linear Time. In *Symposium on Computational Geometry (SoCG)*, pages 119–126, 2006.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[22] Justin Curry, Robert Ghrist, and Vidit Nanda. Discrete Morse Theory for Computing Cellular Sheaf Cohomology. *Foundations of Computational Mathematics*, 16(4):875–897, 2016.

[23] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in Persistent (Co)Homology. *CoRR*, abs/1107.5665, 2011.

[24] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent Cohomology and Circular Coordinates. *Discrete & Computational Geometry*, 45(4):737–759, 2011.

[25] Olaf Delgado-Friedrichs and Vanessa Robins. Diamorse.

[26] Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian P. Sheppard. Morse Theory and Persistent Homology for Topological Analysis of 3D Images of Complex Materials. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 4872–4876, 2014.

[27] Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian P. Sheppard. Skeletonization and Partitioning of Digital Images Using Discrete Morse Theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):654–666, 2015.

[28] Tamal K. Dey. Computing Height Persistence and Homology Generators in $\mathbb{R}^3$ Efficiently. *CoRR*, abs/1807.03655, 2018.

[29] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing Topological Persistence for Simplicial Maps. In *ACM Symposium on Computational Geometry (SoCG)*, SoCG'14, pages 345:345–345:354, New York, NY, USA, 2014. ACM.

[30] Tamal K. Dey, Dayu Shi, and Yusu Wang. SimBa: An efficient tool for approximating Rips-filtration persistence via Simplicial Batch-collapse. In *European Symposium on Algorithms (ESA)*, pages 35:1–35:16, 2016.

[31] Paweł Dlotko and Hubert Wagner. Computing Homology and Persistent Homology Using Iterated Morse Decomposition. *CoRR*, abs/1210.1429, 2012.

[32] Bruce Randall Donald and David Renpan Chang. On the Complexity of Computing the Homology Type of a Triangulation. *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 650 – 661, 11 1991.

[33] Herbert Edelsbrunner and John Harer. *Computational Topology: an Introduction*. American Mathematical Society, 2010.

[34] Herbert Edelsbrunner, David Letscher, and Afra J. Zomorodian. Topological Persistence and Simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[35] Paul Erdős and Alfréd Rényi. On the Evolution of Random Graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[36] Emerson Escolar and Yasuaki Hiraoka. Morse Reduction for Zigzag Persistence. *Journal of the Indonesian Mathematical Society*, 20(1):47–75, 2014.

[37] Robin Forman. Morse Theory for Cell Complexes. *Advances in Mathematics*, 134:90–145, 1998.

[38] David Günther, Jan Reininghaus, Ingrid Hotz, and Hubert Wagner. Memory-Efficient Computation of Persistent Homology for 3D Images Using Discrete Morse Theory. In *2011 24th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 25–32, 2011.

[39] Attila Gyulassy, Peer-Timo Bremer, and Valerio Pascucci. Shared-Memory Parallel Computation of Morse-Smale Complexes with Improved Accuracy. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1183–1192, 2019.

[40] Shaun Harker, Konstantin Mischaikow, Marian Mrozek, and Vidit Nanda. Discrete Morse Theoretic Algorithms for Computing Homology of Complexes and Maps. *Foundations of Computational Mathematics*, 14(1):151–184, 2014.

[41] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[42] Michael Joswig and Marc E. Pfetsch. Computing Optimal Morse Matchings. *SIAM Journal on Discrete Mathematics*, 20(1):11–25, 2006.

[43] Michael Kerber and Hannah Schreiber. Barcodes of Towers and a Streaming Algorithm for Persistent Homology. *Discrete & Computational Geometry*, 61(4):852–879, Jun 2019.

[44] Michael Kerber and Raghvendra Sharathkumar. Approximate Čech Complex in Low and High Dimensions. In *International Symposium on Algortihms and Computation (ISAAC)*, pages 666–676, 2013.

[45] Clément Maria and Steve Y. Oudot. Zigzag Persistence via Reflections and Transpositions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2015*, pages 181–199, 2015.

[46] Clément Maria and Steve Y. Oudot. Computing Zigzag Persistent Cohomology. *CoRR*, abs/1608.06039, 2016.

[47] Clément Maria and Hannah Schreiber. Discrete Morse Theory for Computing Zigzag Persistence. In *Algorithms and Data Structures Symposium (WADS) 2019*, 2019.

[48] Nikola Milosavljevic, Dmitriy Morozov, and Primoz Skraba. Zigzag Persistent Homology in Matrix Multiplication Time. In *Symposium on Computational Geometry (SoCG)*, 2011.

[49] Konstantin Mischaikow and Vidit Nanda. Morse Theory for Filtrations and Efficient Computation of Persistent Homology. *Discrete & Computational Geometry*, 50(2):330–353, 2013.

[50] Marian Mrozek and Bogdan Batko. Coreduction Homology Algorithm. *Discrete & Computational Geometry*, 41(1):96–118, 2009.

[51] James Munkres. *Elements of Algebraic Topology*. Perseus Publishing, 1984.

[52] Vidit Nanda. Perseus.

[53] Monica Nicolau, Arnold J. Levine, and Gunnar E. Carlsson. Topology Based Data Analysis Identifies a Subgroup of Breast Cancers with a Unique Mutational Profile and Excellent Survival. In *Proceedings of the National Academy of Sciences*, pages 7265 – 7270, 2011.

[54] Steve Y. Oudot and Donald R. Sheehy. Zigzag Zoology: Rips Zigzags for Homology Inference. *Foundations of Computational Mathematics*, 15(5):1151–1186, 2015.

[55] Vanessa Robins, Peter J. Wood, and Adrian P. Sheppard. Theory and Algorithms for Constructing Discrete Morse Complexes from Grayscale Digital Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1646–1658, 2011.

[56] Mohammad Saadatfar, Hiroshi Takeuchi, Vanessa Robins, Nicolas Francois, and Yasuaki Hiraoka. Pore Configuration Landscape of Granular Crystallization. *Nature Communications*, 8:15082, May 2017.

[57] Donald R. Sheehy. Linear-size approximation to the Vietoris-Rips Filtration. *Discrete & Computational Geometry*, 49:778–796, 2013.

[58] The GUDHI Project. GUDHI, 2015.

[59] Afra J. Zomorodian and Gunnar E. Carlsson. Computing Persistent Homology. *Discrete & Computational Geometry*, 33:249–274, 2005.