A ROADMAP FOR THE COMPUTATION OF PERSISTENT HOMOLOGY

NINA OTTER[†], MASON A. PORTER[†‡], ULRIKE TILLMANN[†], PETER GRINDROD[†], AND HEATHER A. HARRINGTON[†]

Abstract.
Persistent homology is a method used in topological data analysis to study qualitative features of data. It is robust to perturbations, independent of dimensions and scale, and provides a compact representation of the outputs. The computation of persistent homology, despite recent progress, remains a wide open area with numerous important and fascinating challenges. We investigate the challenges of computing persistent homology, and we navigate the various algorithms that can be used for it. Specifically, we evaluate the (currently available) open-source implementations of persistent-homology computations on a range of synthetic and real-world data sets, and we indicate which algorithms and implementations are best suited to these data. We provide guidelines for the computation of persistent homology, make our own implementations used in this study available to the public, and suggest measures to quantify the challenges of the computation of persistent homology.

Key words. persistent homology, software, computational algebraic topology, topological data analysis

AMS subject classifications. 55-04, 62-07 (Primary); 05C82, 55N35 (Secondary)

1. Introduction. The amount of available data has increased dramatically during the last several years, and this situation — which will only become more extreme — necessitates the development of innovative and efficient data-processing methods. Making sense of the vast amount of data is challenging: data are almost always noisy, high-dimensional, and incomplete. Although the use of techniques such as clustering and other ideas from areas such as computer science, and machine learning — along with mathematical and statistical models — are often very useful for data analysis (see, e.g., [33, 35, 40, 55] and many other references), recent mathematical developments are shedding new light on such 'traditional' ideas and forging new approaches of their own. For example, techniques from the relatively new discipline of 'topological data analysis' (TDA) [12] have provided a wealth of new insights in the study of data in an increasingly diverse set of applications, such as breast cancer [23, 50], viral evolution [16], natural images [15], contagion spread on networks [58], structure of amorphous materials [37], force networks in granular matter [41], and collective behaviour in biology [59].

The main goal of TDA is to use ideas and results from topology — the field of mathematics that is concerned with the study of shapes — to develop tools to study qualitative features of data. This requires precise definitions of the qualitative features, the computational tools to compute them, and the summaries of outputs. We can address all three points using a method in TDA called *persistent homology* (PH). For an introduction see the books on computational topology [27, 34, 62], and the article [26] for history. See `https://www.youtube.com/watch?v=h0bnG1Wavag` for Matthew Wright's introductory video on PH.

Data studied with PH usually takes the form of a point cloud. One can examine the shape of a point cloud by thickening the points at different scales of resolution and analyzing the evolution of the shape across the different resolution values. The

[†]Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK
[‡]CABDyN Complexity Centre, University of Oxford, Oxford OX1 1HP, UK

qualitative features are given by topological invariants, which are used to distinguish between shapes that cannot be continuously deformed one into the other. One can represent the variation of such invariants across the different resolution values in a compact way to yield the summary of the 'shape' of the data.

The first algorithm for the computation of PH was introduced for the computation over the field $\mathbb{F}_2$ in [29] and for general fields in [65]. Since then, several algorithms and optimization techniques have been presented, and there are now several powerful implementations of PH [3,6,48,49,57]. For a non-expert who wishes to try PH, it can be difficult to discern which implementation or algorithm is best suited for a given problem. The field of PH is evolving rapidly, and new software implementations are updated or released at a rapid pace. Not all of them are well-documented, and (as is well known in the TDA community), the computation of PH for large data sets is computationally very expensive. To our knowledge, there is neither an overview of the different computational methods nor a clear summary of the challenges that need to be addressed. In the present article, we close this gap: we introduce computation of PH to the non-expert, provide measures for evaluating PH implementations on different data sets, and offer guidelines for the computation of PH *.

The rest of this paper is organized as follows. In Section 2, we introduce the necessary homological algebra for PH. In Section 3, we provide an overview of algorithms and implementations of PH. In Section 4, we describe the synthetic and real-world data sets that we use as test cases and in Section 5 we describe the measures that we use to compare these implementations. We present the results of our analysis, including an evaluation of which implementations are best suited to the data sets, in Section 6. Finally, in Section 7, we outline points that we believe need to be addressed by the TDA community to overcome the current limitations and challenges of using PH for data analysis.

2. Homology and Persistent Homology. Topology is the study of shapes [36]. Two shapes are considered to be equivalent if one can be continuously deformed into the other — for example, by bending or stretching, but without tearing. To help determine when two shapes are equal, topologists use *invariants*, which are properties that are preserved under continuous deformations. Two useful invariants are the number of connected components and the number of holes of a shape, though of course there are many others. An easily computable method to determine such invariants is *homology*, a technique that assigns to a given shape a vector space whose rank is the number of connected components, holes, or other topological feature, of the shape. Persistent homology is the homology of a 'filtration' of shapes. If we are given a collection of nested shapes $S_1 \subset \cdots \subset S_l$, then we can use PH to study how the invariants of the shapes evolve along the filtration. We devote the rest of the present section to making the previous explanation precise.

2.1. Homology. To keep our exposition simple, we restrict our description to *simplicial homology* over the field $F = \mathbb{F}_2$ of two elements. For an introduction to homology theories over general rings, see [36]. For an overview of computational homology, see [38].

*A lot of research is being done on efficient data structures. However, for most implementations the end-user cannot choose the data structure to use, and moreover some of the data structures are language dependent. We therefore do not include a discussion of data structures because it would not help the end-user and would take us in waters better navigated by computer scientists than by applied mathematicians.

A *chain complex* over a field F is a tuple (C, d), where C is a collection $\{C_p\}_{p\in\mathbb{N}}$ of vector spaces indexed by the natural numbers, together with a collection d of F-linear maps $\{d_p \colon C_p \to C_{p-1}\}_{p\in\mathbb{N}}$ such that $d_{p-1} \circ d_p = 0$ for all natural numbers p. The maps d_p are called *boundary maps*. The *p-cycles* of the complex are the elements sent to 0 by the map d_p; the *p-boundaries* are the elements in the image of d_{p+1}. A *map of chain complexes* $f \colon (C, d) \to (C', d')$ is a collection $\{f_p \colon C_p \to C'_p\}_{p\in\mathbb{N}}$ of F-linear maps such that $f_{p-1} \circ d_p = d'_p \circ f_p$ for all natural numbers p. The p-cycles form a vector space, and so do the p-boundaries; we denote these vector spaces by Z_p and B_p, respectively. The *pth homology* of a chain complex (C, d) is the quotient vector space $H_p((C, d)) = Z_p/B_p$. The number $\dim H_p((C, d)) = \dim Z_p - \dim B_p$ is called the *pth Betti number* of (C, d) and is denoted by $\beta_p(C)$. Any map of chain complexes $f \colon (C, d) \to (C', d')$ induces a linear map on homology $H(f) \colon H_p((C, d)) \to H_p((C', d'))$.

Example 1. Consider the following chain complex over F:

$$0 \underset{d_4}{\longrightarrow} \underset{=C_3}{F} \overset{\Delta}{\underset{d_3}{\longrightarrow}} \underset{=C_2}{F^2} \overset{0}{\underset{d_2}{\longrightarrow}} \underset{=C_1}{F} \underset{d_1}{\longrightarrow} 0\,,$$

where Δ is the diagonal map that sends any $r \in F$ to the pair (r, r) and $C_i = 0$ for $i \leq 0$ and $i \geq 4$. We have

$$H_1((C, d)) = d_1^{-1}\{0\}/\mathrm{im}(d_2) = F$$

$$H_2((C, d)) = d_2^{-1}\{0\}/\mathrm{im}(d_3) = F^2/F \cong F,$$

and $H_i((C, d)) = 0$ for all other natural numbers i.

A finite (abstract) *simplicial complex* is a collection K of non-empty subsets of a finite set K_0 such that $\tau \subset \sigma$ and $\sigma \in K$ guarantees that $\tau \in K$ and such that $\{v\} \in K$ for all $v \in K_0$. The elements of K_0 are called *vertices* of K, the elements of K are called *simplices*, and we say that a simplex has *dimension p* or is a *p-simplex* if it has a cardinality of $p + 1$. The collection of p-simplices is denoted by K_p. If τ and σ are simplices such that $\tau \subset \sigma$, then we call τ a *face* of σ. A *map of simplicial complexes* $f : K \to L$ is a map $f \colon K_0 \to L_0$ such that $f(\sigma) \in L$ for all $\sigma \in K$. Every p-simplex can be realized geometrically as the convex hull of $p+1$ affinely independent points in $\mathbb{R}^p$ (see Fig. 2.1). A simplicial complex with N vertices can be realized geometrically in $\mathbb{R}^N$ as the union of the realization of its simplices when the N vertices have been identified using the standard basis.

Given a simplicial complex K, we can assign to it the chain complex (C_K, d_K) over $\mathbb{F}_2$ such that C_{Kp} is the $\mathbb{F}_2$-vector space whose basis is given by the p-simplices of K and whose boundary maps are

$$d_{Kp} \colon C_{Kp} \to C_{Kp-1} \colon \sigma \mapsto \sum_{\tau\subset\sigma, \tau\in K_{p-1}} \tau\,.$$

Any map $f : K \to L$ of simplicial complexes induces a map $(C_K, d_K) \to (C_L, d_L)$ of chain complexes. The pth homology of the chain complex (C_K, d_K) is called the *pth simplicial homology* of K and is denoted by $H_p(K)$.

From a geometric perspective, one can construe the dimension of $H_p(K)$ as the number of 'p-dimensional holes' of K. In particular, the following geometric interpretations are very helpful:

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

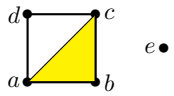

Fig. 2.1: (Left) A simplicial complex K with five 0-simplices, five 1-simplices, and one 2-simplex. (Right) Geometric realization of K. The Betti numbers of its associated simplicial complex are $\beta_0(K) = 2$, $\beta_1(K) = 1$, and $\beta_2(K) = 0$.

- The 0th Betti number counts the number of connected components.
- The 1st Betti number counts the number of loops.
- The 2nd Betti number counts the number of voids.

The *dimension* of a simplicial complex is the maximum of the dimensions of its simplices. If K is a simplicial complex of dimension d, then $H_p(K) = 0$ for all $p > d$.

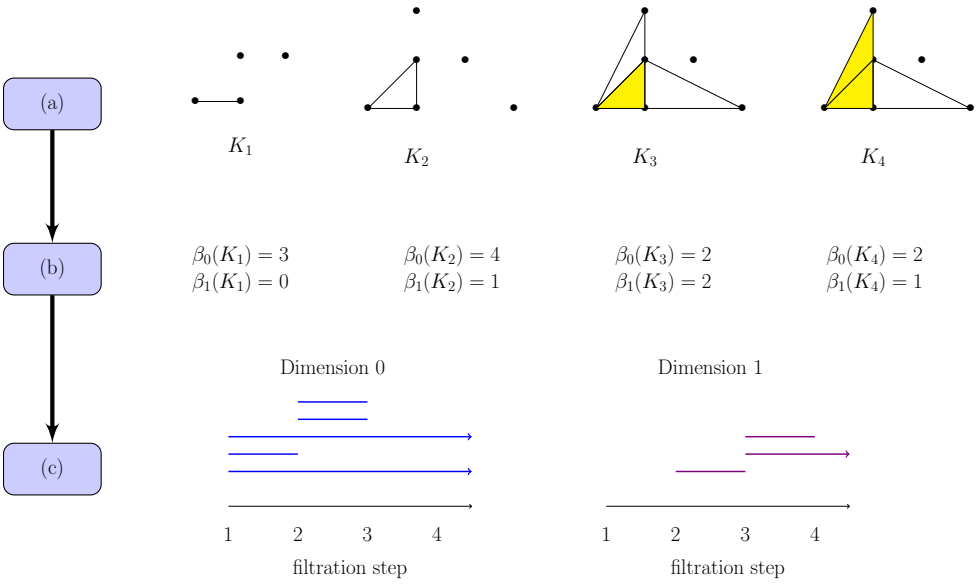

Fig. 2.2: Example of persistent homology. (a) We start with a finite-dimensional filtered simplicial complex. That is, we start with a sequence of finite simplicial complexes $K_1, \ldots, K_l$ such that $K_i \subset K_j$ whenever $i \leq j$ for $i, j = 1, \ldots, l$. For each $i = 1, \ldots, l$, we call the subcomplex K_i the *ith filtration step*. (b) We compute the simplicial homology of each complex K_i. (c) By the Correspondence Theorem for Persistent Homology [65, Theorem 3.1], we obtain barcodes that represent the lifetime of homological features across the filtration. Each bar in dimension p corresponds to the lifetime of a p-homology class: its left endpoint represents the smallest filtration step i at which the class first appears, and its right endpoint represents the smallest filtration step $j > i$ at which the representing cycle becomes a boundary. We say that the class is 'born' at i and 'dies' at j. Features that are born but do not die at the final filtration step are represented by arrows starting at the birth of that feature and pointing to the right.

2.2. Persistent Homology. We now introduce some basic notions in persistent homology (PH). See [27,34,62] for introductions to PH.

A *filtration* of a finite simplicial complex K is a collection of finite simplicial complexes $K_1, \dots, K_l$ such that $K_1 \subseteq \cdots \subseteq K_l = K$. We use the term *filtered simplicial complex* for a simplicial complex together with a filtration. For all $i \leq j$, the inclusion maps $K_i \to K_j$ induce F-linear maps $f_{i,j} \colon H_p(K_i) \to H_p(K_j)$ on simplicial homology. For $0 \neq x \in H_p(K_i)$, we say that x *dies* in $H_p(K_j)$ if $j > i$ is the smallest index for which $f_{i,j}(x) = 0$. We say that $0 \neq x \in H_p(K_i)$ is *born* in $H_p(K_i)$ if $f_{k,i}^{-1}(x) = 0$ for all $k < i$. We can represent the lifetime of x using the half-open interval $[i, j)$. If $f_{i,j}(x) \neq 0$ for all $i < j \leq l$, we say that x *lives forever* and we represent its lifetime by the interval $[i, \infty)$.

The *pth PH vector spaces* of a filtered simplicial complex K are defined as $H_p^{i,j} = \mathrm{im}(f_{i,j})$, and the *total pth PH of* K is defined as $\oplus_{i=1}^{l} H_p(K_i)$. By the Correspondence Theorem of Persistent Homology [65, Theorem 3.1], for every p, we can assign a finite well-defined collection of half open intervals — its so-called *barcode* — to the total pth persistent homology vector space. We illustrate this assignment in Fig. 2.2. An alternative way to represent PH graphically is to use *persistence diagrams*, in which an interval $[i, j)$ is represented by the point (i, j) in $\mathbb{R}^2$.

3. Computation of PH from Point Clouds. A point cloud is a finite metric space (X, d). One can use PH to analyze data in the form of a point cloud. For applications, PH is very appealing: data can be studied along multiple scales and the results of PH are stable with respect to small perturbations in measurements. (For stability results, see [11,18].) In Fig. 3.1, we illustrate the pipeline for the computation of PH for a point cloud, and in Fig. 3.2 we give an example.

In the following subsections, we describe each step and state-of-the-art algorithms and data structures for the computation of PH.

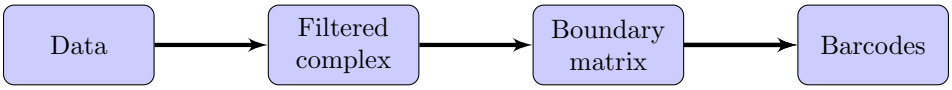

Fig. 3.1: Pipeline for the computation of PH from a point cloud.

3.1. From Data to a Filtered Simplicial Complex. Given a point cloud (X, d), there are several ways to associate a simplicial complex to it.

The Vietoris–Rips complex is one of the most popular complexes in PH. For a non-negative real number ϵ, the *Vietoris–Rips complex* $V(X, \epsilon)$ *at scale* ϵ is defined as follows:

$$V(X, \epsilon) = \{\sigma \subset X \mid d(x, y) \leq \epsilon \text{ for all } x, y \in \sigma\}.$$

For $\epsilon \leq \epsilon'$, we have $V(X, \epsilon) \subseteq V(X, \epsilon')$, so considering different values of the scale ϵ yields a filtered simplicial complex. The dimension of the Vietoris–Rips complex is bounded only by the size of X, so in practice it is necessary to put a limit on the dimension of the simplices that one allows in the construction of the Vietoris–Rips complex. Let N denote the cardinality of X. In the worst case, the Vietoris–Rips complex includes $N!/p!(N-p)!$ simplices of dimension p, so the number of simplices can grow exponentially with the number of input points in the cloud.

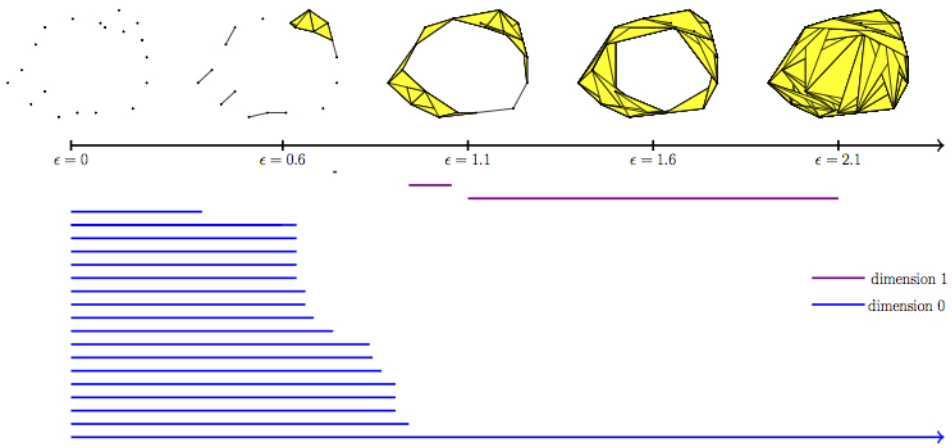

Fig. 3.2: Computation of PH for a point cloud using the Vietoris–Rips complex.

Two kinds of optimizations are known to reduce the complexity of the computation of simplicial complexes from a point cloud:

(A) Simplicial complexes whose number of simplices grows slowier and whose homology approximates that of the original complex.

(B) Reduction techniques to obtain a smaller simplicial complex with the same homology as the original complex.

We summarise these optimisations in Table 3.1. Fast implementations of the computation of simplicial complexes are crucial; fast algorithms for the computation of the Vietoris–Rips complex were introduced in [63]. For some recent work on efficient data structures see [2] for generalized Vietoris–Rips complexes and [9] for general simplicial complexes.

Complex K	Size of K	(A) Approximation	(B) Reduction
Čech [32]	$2^{\mathcal{O}(n)}$	–	tidy set [64] Morse red. [47]
Vietoris–Rips (VR) [60]	$2^{\mathcal{O}(n)}$	linear-size appr. [56] (lazy) witness [20] (LW) GIC [24] HOPES [43] (only 1-D)	tidy set Morse red.
α [31]	$n^{\mathcal{O}(\lceil d/2 \rceil)}$ (n points in $\mathbb{R}^d$)	HOPES [43] (only 1-D)	tidy set Morse red.
weight rank clique filtration (WRCF) [53]	$2^{\mathcal{O}(n)}$	-	tidy set Morse red.

Table 3.1: We list several types of complexes used for PH. We indicate the worst case size of the complex in function of the cardinality n of the point cloud and (A) optimisation in terms of a complex whose size grows slowlier and whose homology has been proved to approximate that of the original complex (B) optimisation that reduces the size of the complex without changing the homology. For the α-complex there is a well-known speed-up in 1-D which uses a duality between 0-dimensional and 1-dimensional persistence for α-complexes [27] (see [42] for an implementation).

3.2. From a Filtered Simplicial Complex to a Boundary Matrix. To compute the PH of a filtered simplicial complex K, we need to associate to it a matrix — the so-called *boundary matrix* — that stores information about the faces of every simplex. To do this, we place a total ordering on the simplices of the complex that is compatible with the filtration in the following sense:

- a face of a simplex precedes the simplex;
- a simplex in the ith complex K_i precedes simplices in $K \setminus K_i$.

Let n denote the total number of simplices in the complex and denote the simplices with respect to this ordering by $\sigma_1, \ldots, \sigma_n$. We construct a square matrix δ of dimension n by storing a 1 in $\delta(i, j)$ if the simplex σ_i is a face of simplex σ_j of codimension 1; otherwise, we store a 0 [†].

3.3. From a Boundary Matrix to a Barcode. The standard algorithm for the computation of PH was introduced in [29] for the field $\mathbb{F}_2$ and for general fields in [65]. For every $j = 1, \ldots, n$, define $\text{low}(j)$ to be the index of the lowest row that contains a 1 in column j. If column j only contains 0 entries, then the value of $\text{low}(j)$ is undefined. We say that the boundary matrix is *reduced* if the map low is injective on its domain of definition. In Fig. 3.3, we illustrate the standard algorithm for reducing the boundary matrix. Because this algorithm operates on columns of the matrix from left to right, it is also called 'column algorithm' in the literature. In the worst case, the complexity of the algorithm is cubic in the number of simplices.

```
for i = 1 to n do
   while there exists i < j with low(i) = low(j) do
      add column i to column j
   end while
end for
```

Fig. 3.3: The standard algorithm for the reduction of the boundary matrix to barcodes.

Once the boundary matrix is reduced, we can read off the intervals in a barcode by pairing the simplices.

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

Following the standard algorithm several new algorithms and optimisations have been introduced. For all algorithms the worst case running time is cubic in the number of simplices, however the actual running times can vary tremendously depending on which algorithm is used.

The dual algorithm is a sequential algorithm and is based on the computation of persistent cohomology [21, 22]. The twist algorithm (also called the 'row algorithm' because it operates on rows of the boundary matrix) is a sequential algorithm and takes a shortcut in the reduction of the boundary matrix [17]. The spectral sequence algorithm divides the boundary matrix into blocks and reduces them from the diagonal outwards in different phases that can be executed in parallel [28, Section VII.4]. The

[†] Recent publications on efficient data structures for the boundary matrix are [6] and [8].

chunk algorithm implements the first two phases of the reduction of the spectral sequence, then eliminates pairs that have already been found, and in the end applies the twist agorithm [4]. The distributed algorithm combines a generalisation of the spectral sequence algorithm with the twist algorithm, in such a way that the reduction can be executed in parallel in a distributed setting [5].

For existing performance studies of these algorithms we refer the reader to [6, 21, 46]. In particular, in [21] the authors compare the performance of the dual algorithm against the standard algorithm and suggest that whenever possible one should use the dual algorithm instead of the standard algorithm when computing PH.

The previous algorithms can be used to reduce the boundary matrix in case one has a sequence of complexes with inclusion maps going all in the same direction, as in the following diagram:

$$\cdots \to K_{i-1} \to K_i \to K_{i+1} \to \dots$$

An algorithm for the computation of PH in case the inclusion maps do not all go in the same direction, as e.g. in the following diagram

$$\cdots \to K_{i-1} \to K_i \leftarrow K_{i+1} \to \dots$$

was introduced in [13], and is called the zigzag algorithm.

In the more general setting in which the maps are not inclusions one can still compute PH using the simplicial map algorithm [25].

3.4. Software. There are currently several implementations for the computation of PH that are publicly available. We summarise the properties of some of these in Table 4.1, and report on performance for a selection of them in Section 6. For a comprehensive list of programs see `https://people.maths.ox.ac.uk/otter/PH_programs`.

The software package javaPlex [57], which was developed by the computational topology group at Stanford University, is based on the Plex library [52], which to our knowledge is the first piece of software to implement the computation of PH. Perseus [49] was developed to implement theoretical developments in the theory of discrete Morse theory [47]. jHoles [7] is a Java library aimed at the computation of the weight rank clique filtration for weighted undirected networks [53]. Dionysus [48] is the first software package to implement the dual algorithm [21, 22]. PHAT [6] is a library that implements several algorithms and data structures for the fast computation of barcodes from the boundary matrix, and it is the first software to implement a matrix-reduction algorithm that can be executed in parallel. DIPHA [3], a spin-off of PHAT, implements a distributed computation of the matrix-reduction algorithm. GUDHI [46], the most recently developed software of the set that we examine, implements new data structures for the simplicial complex and the boundary matrix, and the multifield algorithm, which allows the simultaneous computation of PH over several fields [10]. SimpPers [54] implements the simplicial map algorithm.

4. Data sets. We compute PH for several data sets by constructing the filtered Vietoris–Rips complex up to the maximum distance between any two points. We use two types of data sets: point clouds and weighted undirected networks. To obtain a point cloud from a weighted undirected network we compute shortest paths, using the weights on the edges as distances between nodes; for the real-world networks we take the inverse of the weights.

We list the data sets, of which 1-2 are synthetic and 3-6 are real-world data sets:

Software	javaPlex	Perseus	jHoles	Dionysus	PHAT	DIPHA	GUDHI	SimpPers
(A) Language	Java	C++	Java	C++	C++	C++	C++	C++
(B) Algorithms for PH	standard, dual, zigzag	Morse reductions, standard	standard (uses javaPlex)	standard, dual, zigzag	standard, dual, twist, chunk, spectral seq.	twist, dual, distributed	dual, multifield	simplicial map
(C) Coeff. field	$\mathbb{Q}$ $\mathbb{F}_p$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$ (standard, zigzag) $\mathbb{F}_p$ (dual)	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$
(D) Homology	simplicial, cellular	simplicial, cubical	simplicial	simplicial	simplicial, cubical	simplicial, cubical	simplicial	simplicial
(E) Filtrations computed	VR LW W	VR	WRCF	VR α Čech	–	VR, lower star	VR	–
(F) Filtrations as input	simplicial complex, zigzag, CW	simplicial complex, cubical complex	–	simplicial complex, zigzag	boundary matrix of simpl. complex	boundary matrix of simpl. complex	–	map of simpl. complexes
(G) Additional features	Computes some hom. alg. constructions	–	–	vineyards, circle-valued functions	–	–	–	–
(H) Precompiled	✓	✓	✓	–	–	–	–	✓
(I) Visualisation	barcodes	persistence diagram	–	–	–	persistence diagram	–	–

Table 4.1: Overview of currently existing published software for the computation of PH. The symbol – means that the respective feature is not implemented, and ✓ means that it is implemented. For each software package we indicate: (A) The language in which it is written. (B) The implemented algorithms for the computation of barcodes from the boundary matrix. (C) The coefficient fields for which PH is computed; the letter p denotes any prime number in the coefficient field $\mathbb{F}_p$. (D) The type of homology computed. (E) The filtered complexes that are computed and (F) the ones that can be given as input. We use the notation for the complexes introduced in Table 3.1. javaPlex supports the input of a filtered CW complex for the computation of cellular homology [36]; in constrast with simplicial complexes there are at present no algorithms to assign a cell complex to point cloud data. DIPHA implements the computation of the lower star filtration [30] of a cubical complex for image data [61]. (G) Additional features implemented by the library; javaPlex supports the computation of some constructions from homological algebra (see [57] for details), while Dionysus implements the computation of vineyards [14] and circle-valued functions [22]. (H) If executable files are provided and (I) if visualisation of the output is provided.

1. Klein bottle: The Klein bottle is a one-sided nonorientable surface (Fig. 4.1). We linearly sample points from the Klein bottle using its 'figure-8' immersion in $\mathbb{R}^3$ and size samples of 400 and 900 points. We vary the upper bound of the dimension of the Vietoris–Rips complex from 2 to 3.
   Note that the image of the immersion of the Klein bottle does not have the same homotopy type of the Klein bottle, but they have the same singular homology[‡] with $\mathbb{Z}_2$ coefficients; we have $H_0(B) = F$, $H_1(B) = F \oplus F$ and $H_2(B) = F$, where B denotes the Klein bottle and $H_i(B)$ is the ith singular homology group with $F = \mathbb{Z}_2$ coefficients.
2. Random VR complexes [39]: the parameters for this model are positive integers N and d. The random VR complex $V(N;\epsilon)$ is the VR complex $V(X,\epsilon)$, where X is a set of N points sampled from $\mathbb{R}^d$. Here the sample is taken with respect to any distribution on $\mathbb{R}^d$ that has a bounded Lebesque-measurable density function. Equivalently, $V(N;\epsilon)$ is the clique complex on the random geometric graph $G(N,\epsilon)$ [51]. We sample N points uniformly at random from $[0,1]^d$. We choose $(N,d) = (100,4)$ and $(1000,8)$. The 'Vanishing/non-vanishing threshold theorem' [39] for random VR complexes says that if $d \geq 2$, $k \geq 1$ and $\epsilon \in o\left(N^{-1/d}\right)$ then for $\epsilon \in o\left(N^{-\frac{2k+2}{d(2k+1)}}\right)$ we have $H_k(VR(N;\epsilon)) = 0$ a.a.s. and for $\epsilon \in \omega\left(N^{-\frac{2k+2}{d(2k+1)}}\right)$ a.a.s. $H_k(VR(N;\epsilon)) \neq 0$.
3. Genomic sequences of the HIV virus: we obtain point clouds from the independent and concatenated sequences of the three largest genes `gag`, `pol` and `env` of the HIV genome together with the Hamming distance. We use the aligned sequences studied using PH in [16] (the authors of that paper retrieved the sequences from [44]).
4. Stanford Dragon graphic: we sample points uniformly at random from three-dimensional (3D) scans of the dragon [45], whose reconstruction we show in Fig. 4.1. The sample size varies between 2000 and 3000 points, and we vary the dimension cap for the Vietoris–Rips complex between 2 and 3.
5. *C. elegans* neuronal network [53]: a weighted, undirected network in which each node is a neuron and edges represent synapses or gaps junctions.
6. Human genome: a weighted, undirected network representing a sample of the human genome. We use the network studied using PH in [53] (the authors of that paper created the sample using data retrieved from [19]). Each node represents a gene, and weighted edges between nodes represent the correlation of the expression level of two genes.

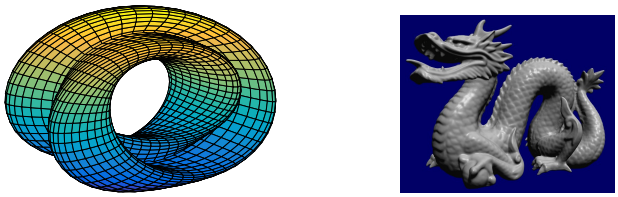

Fig. 4.1: The image of the figure-8 immersion of the Klein bottle and the reconstruction of the Stanford Dragon (retrieved from [45]).

[‡] *Singular homology* is a method that assigns to every topological space homology groups encoding invariants of the space, in an analogous way as simplicial homology assigns homology groups to simplicial complexes. We refer the reader to [36] for an account of singular homology.

5. Methods. In this section, we first explain the reasons behind our choice of software to include in the study. We then outline our performance analysis.
We decided to include only software with a published paper, and only the most recent version of every library. All libraries that satisfy these two criteria are listed in Table 4.1, and we describe the complexes and algorithms they implement in Section 3. To study the performance of the packages we decided to restrict our focus to the algorithms that are implemented by the largest number of libraries. We thus restrict to the computation of the Vietoris–Rips complex and the dual and standard algorithms and therefore only study the software javaPlex, Perseus, Dionysus, DIPHA, and GUDHI, because they all implement the computation of the Vietoris–Rips complex. PHAT only takes a boundary matrix as input, so it is not possible to conduct a direct comparison with the other implementations. However, the fast data structures and algorithms implemented in PHAT are also implemented in its spin-off software DIPHA, which we include in our study. The software jHoles is aimed at the computation of PH using the WRCF for weighted undirected networks, while SimpPers takes as input a map of simplicial complexes, and thus cannot be directly compared to the other libraries.

We examine the software packages javaPlex, Perseus, Dionysus, DIPHA, and GUDHI using both synthetic and real-world data from four different perspectives:

1. Performance measured in CPU seconds and wall-time (i.e., elapsed time) seconds.
2. Memory required by the process.
3. Maximum size of simplicial complex allowed by the software.
4. Phases of computation of PH supported by the software.

5.1. Machines. We tested all of the software on a cluster with 1728 (i.e., 108×16) cores of 2.0GHz Xeon SandyBridge, RAM of 64 GiB in 80 nodes and 128 GiB in 4 nodes, and a scratch disk of 20 TB.[§]

6. Tests and results. We ran the software on a total of 9 different data sets for each software. We repeated each computation five times[¶] and report average timings and memory measurements. Of the five software that we study, four implement the computation of the dual algorithm, and four implement the standard algorithm. We therefore compare the two sets of four libraries running either the dual algorithm or the standard algorithm. DIPHA is the only software to implement a distributed computation; we chose as a default to run the software on one node and 16 cores; we only increased the number of nodes and cores employed when the machine ran out of memory. For the other software the computations were running on a single node with one core. Augmenting the number of nodes can make the computations faster (in terms of CPU seconds) also for smaller complexes [‖]; we see this in our experiments,

[§]We also tested the four libraries that do not not depend on a distributed setting, on an IBM shared memory system with 16 (i.e. 2) cores of 3.3GHz, and a RAM of 768 GB, and further on a shared memory system of 64 (8×8) cores of 2.67GHz Xeon/Westmere EX cores and a RAM of 1TB. The major difference in running shared algorithms on such systems vs. the distributed memory system is that each node has much more RAM available than in the distributed system.

[¶]We repeated each computation only five times since PH computations can take from fractions of seconds to several weeks, or months.

[‖]Based on the results of our tests, we think of small, medium, and large complexes as complexes having a size in the order of magnitude of up to 10 million simplices, between 10 million and 100 million simplices, and between 100 million and a billion simplices, respectively. At present there are no software packages that can handle complexes with 10 billion simplices or more.

and it is also discussed in [5]. Due to space constraints we only report results for five data sets; these are however a representative sample of the overall results.

6.1. Performance. In Table 6.1, we report the timings of the different software packages for a few of the data sets. We measured the elapsed and CPU time by using the `time` function in Unix.

Data set	C. elegans		Klein		HIV		Dragon 1		Dragon 2	
size of complex	4.4×10^6		1.1×10^7		2.1×10^8		1.7×10^8		1.3×10^9	
javaPlex st	84	284	747	1031	-	-	-	-	-	-
Dionysus st	474	473	1830	1824	-	-	-	-	-	-
DIPHA st	6	68	90	1360	1631	25950	7356	117417	142559	1489615
Perseus	543	542	1978	1974	-	-	-	-	-	-
javaPlex d **										
Dionysus d	513	513	145	145	-	-	4360	4362	-	-
DIPHA d	4	39	6	73	81	1276	75	1176	2358	37572
GUDHI	6	4	11	10	249	248	285	283	15419	3151

Table 6.1: Performance of the software packages measured in wall-time and CPU seconds for the *C. elegans*, Klein bottle, HIV and Dragon data sets. For each data set, we indicate the size of the simplicial complex. We indicate the implementation of the standard algorithm by the abbreviation 'st' following the name of the package, and the implementation of the dual algorithm by the abbreviation 'd'. Perseus implements only the standard algorithm, and GUDHI implements only the dual algorithm. We ran DIPHA on one node and 16 cores for the data sets *C. elegans*, Klein and Dragon 1; on 2 nodes of 16 cores for the HIV data set, and for Dragon 2 on 2 and 3 nodes of 16 cores for the dual and standard implementations, respectively.

6.2. Memory. In Table 6.2, we report the memory used by the processes in terms of maximum resident set size (RSS), i.e. the maximum amount of real RAM the program has used during its execution. Due to technical issues, we cannot include memory measurements for the software javaPlex. However, one can infer memory requirements for this software package by the value of the maximal heap size necessary to perform the computations; we report this in Table 6.2.

Data set	C. elegans	Klein	HIV	Dragon 1	Dragon 2
size of complex	4.4×10^6	1.1×10^7	2.1×10^8	1.7×10^8	1.3×10^9
javaPlex st	< 5	< 15	> 120	> 120	> 120
Dionysus st	1.3	11.6	-	-	-
DIPHA st	0.1	0.2	2.7	2.4	4.9
Perseus	5.1	12.7	-	-	-
javaPlex d					
Dionysus d	0.5	1.1	-	16.8	-
DIPHA d	0.1	0.2	1.8	1.8	13.8
GUDHI	0.2	0.6	9.9	9.2	64.5

Table 6.2: Memory usage in GB for the same data sets of Table 6.1. For javaPlex, we indicate the value of the maximum heap size that was sufficient to perform the computation; thus the value we give is an upper limit to memory usage. For DIPHA, we indicate the maximum memory used by a single core (considering all cores). For the difference in memory usage between the dual and standard implementation of DIPHA for the data set Dragon 2, note that we used 32 cores for the dual implementation and 48 for the standard implementation.

**Due to a technical issue, we are currently not able to report results for this implementation.

6.3. Maximum size of simplicial complex and steps implemented. In Table 6.3, we give the maximum size of the simplicial complex for which we were able to compute PH with each software package and we report the steps implemented by the libraries in Table 6.4.

Software	javaPlex		Dionysus		DIPHA		Perseus	GUDHI
	st	d	st	d	st	d	st	d
max. size	$1 \cdot 10^7$		$1.1 \cdot 10^7$	$3.2 \cdot 10^8$	$1.3 \cdot 10^9$	$1.3 \cdot 10^9$	$1 \cdot 10^7$	$1.3 \cdot 10^9$

Table 6.3: Maximal size of simplicial complex supported by the software thus far.

Software	javaPlex	Perseus	Dionysus	DIPHA	GUDHI
Installation	✓	✓	–	–	–
Complex	✓	✓	✓	✓	✓
Boundary matrix	✓	✓	✓	✓	✓
Barcodes	✓	✓	✓	✓	✓
Visualisation	✓	✓	–	✓	–

Table 6.4: Roughly, one can divide the pipeline for the computation of PH into five steps: (1) installation of the software, (2) computation of the complex from input data, (3) computation of the boundary matrix, (4) computation of barcodes, and (5) visualisation of outputs. For each software package, we indicate which of these steps are supported.

6.4. Conclusions. Our tests show that GUDHI and DIPHA are the most powerful libraries currently available, and can handle well large complexes. The dual implementation in Dionysus is suited to medium complexes, while javaPlex, Perseus and the standard implementation in Dionysus can only handle small complexes. Furthermore, there is a huge disparity between implementations of dual and standard algorithms: dual implementations outperform standard ones not only in terms of CPU and wall-time seconds but also in the amount of memory used. This significant difference in performance and memory usage was already revealed for the software package Dionysus in [21].

For the computation of PH for samples of points from the Klein bottle we were only partially able to recover the expected homology; PH for the sample of size 400 reveals a long bar in 0th persistent homology, only a long bar (instead of two) in first PH, but no significant bar in dimension 2. We obtained similar barcodes for the sample of size 900 by bounding the dimension of the Vietoris–Rips complex to 2, while we were not able to finish the computations for this sample by including also 3-simplices (the VR complex contains about 27 billion simplices in this case) [††]. Even though the estimates for the random VR complexes given in [39] are asymptotic, we were able to recover them with our computations: we report the barcodes for dimension 1 and 2 for the random VR complex on 100 points in $[0, 1]^4$ in Figure 6.1.

To conclude, the best-performing software packages — DIPHA and GUDHI — are also the ones that can handle the largest number of simplices (about one billion). However, for small complexes, the software packages Perseus and javaPlex are

[††]In such a case, one could compute 'partial' barcodes by constructing the Vietoris-Rips complex only up to a certain value for ϵ.

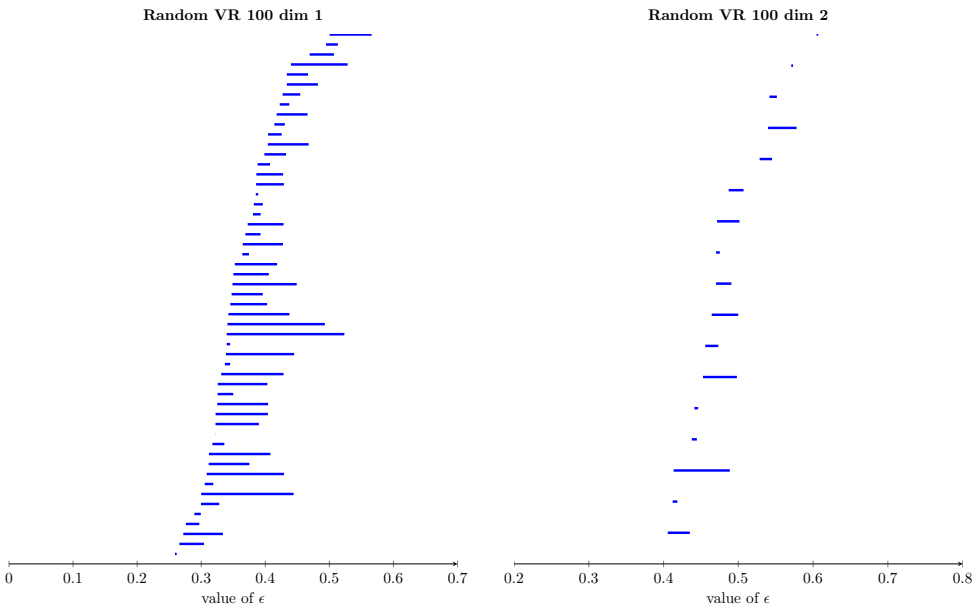

Fig. 6.1: Barcode of random VR complex on 100 points in $[0,1]^4$ for dimensions 1 and 2.

best suited because they are the easiest ones to use (see Table 6.4). Since the library JAVAPLEX implements the computation of a variety of complexes and algorithms, it is the best software for a first approach to PH computation.

7. Considerations and Outlook. We compared the algorithms in publicly available software with the aim to test the implementations that are currently available to the mathematical community. Our results should be viewed as a guideline for people who want to analyze data by computing PH. Even though a fair comparison of performance of methods can be difficult to achieve [1], we call on the TDA community for a concerted effort to test the state-of-the-art methods. Some of the existing comparisons available in the TDA literature that we cited in the present paper rely on different implementations and architectures, and we have endeavored to take a step towards more direct comparisons.

We conclude by outlining some challenges that we believe must be addressed by the TDA community to overcome current limitations and issues. There is a need for the creation of a user-friendly computational topology library and the definition and construction of benchmark data sets for the test of new algorithms and data structures. Furthermore, input and output formats vary from one software package to another, and we thus call for a uniformization of input and output type across the different implementations. Although recent progress in the optimization of the reduction of the boundary matrix has made the computation of barcodes extremely fast, this progress is hindered by the issues related to the computation of simplicial complexes, which have not received the same attention. Finally, new techniques are required to compute PH for streams of data. With such techniques at hand, one could not only make sense of the vast amount of streams of data produced, but also tackle challenges related to the computation of 'static' data sets (i.e., which do not come in real-time streams).

8. Acknowledgements. We thank the Rabadan Lab at Columbia University for providing the HIV sequences used in [16] and Giovanni Petri for sharing the data sets used in [53]. We thank Adrian Clough, Patrizio Frosini, Florian Klimm, Vitaliy Kurlin, Robert MacKay and Dane Taylor for helful comments on the first version of this paper. NO thanks Ulrich Bauer, Michael Lesnick, Hubert Wagner and Matthew Wright for helpful discussions, and Florian Klimm, Vidit Nanda, and Bernadette Stolz for precious advice. We thank the participants of the workshop 'Algebraic Topology: Computation, Data Analysis, and Applications', held at the Mathematical Institute, University of Oxford on 24 Febraury 2015, for putting forward some of the challenges that we outline in the previous section. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work (`http://dx.doi.org/10.5281/zenodo.22558`).
NO and PG are grateful for support from the EPSRC grant EP/G065802/1 (The Digital Economy HORIZON Hub). HAH gratefully acknowledges EPSRC Fellowship EP/K041096/1.

REFERENCES

[1] Nature Methods, 12 (2015), p. 273. Editorial.
[2] Dominique Attali, André Lieutier, and David Salinas, *Efficient data structure for representing and simplifying simplicial complexes in high dimensions*, International Journal of Computational Geometry and Applications, 22 (2012), pp. 279–303.
[3] Ulrich Bauer, Michael Kerber, and Jan Reininghaus, *DIPHA (a distributed persistent homology algorithm)*. Software available at `https://code.google.com/p/dipha/`.
[4] ———, *Clear and compress: Computing persistent homology in chunks*, in Topological Methods in Data Analysis and Visualization III, Peer-Timo Bremer, Ingrid Hotz, Valerio Pascucci, and Ronald Peikert, eds., Mathematics and Visualization, Springer International Publishing, 2014, pp. 103–117.
[5] ———, *Distributed computation of persistent homology*, in 2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX), Society for industrial and applied mathematics, 2014, ch. 3, pp. 31–38.
[6] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner, *PHAT: Persistent homology algorithms toolbox*, in Mathematical Software, ICMS 2014, Hoon Hong and Chee Yap, eds., vol. 8592 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 137–143. Software available at `https://code.google.com/p/phat/`.
[7] Jacopo Binchi, Emanuela Merelli, Matteo Rucco, Giovanni Petri, and Francesco Vaccarino, *jHoles: A tool for understanding biological complex networks via clique weight rank persistent homology*, Electronic Notes in Theoretical Computer Science, 306 (2014), pp. 5 – 18. Proceedings of the 5th International Workshop on Interactions between Computer Science and Biology (CS2Bio14).
[8] Jean-Daniel Boissonnat, TamalK. Dey, and Clément Maria, *The compressed annotation matrix: An efficient data structure for computing persistent cohomology*, in Algorithms - ESA 2013, HansL. Bodlaender and GiuseppeF. Italiano, eds., vol. 8125 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 695–706.
[9] Jean-Daniel Boissonnat and Clément Maria, *The simplex tree: An efficient data structure for general simplicial complexes*, in Algorithms - ESA 2012, Leah Epstein and Paolo Ferragina, eds., vol. 7501 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 731–742.
[10] ———, *Computing persistent homology with various coefficient fields in a single pass*, in Algorithms - ESA 2014, Andreas S. Schulz and Dorothea Wagner, eds., vol. 8737 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 185–196.
[11] Peter Bubenik and Jonathan A. Scott, *Categorification of persistent homology*, Discrete & Computational Geometry, 51 (2014), pp. 600–627.
[12] Gunnar Carlsson, *Topology and Data*, Bulletin of the American Mathematical Society, 46 (2009), pp. 255–308.
[13] Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov, *Zigzag persistent homology and real-valued functions*, in Proceedings of the Twenty-fifth Annual Symposium on Compu-

tational Geometry, SCG '09, New York, NY, USA, 2009, ACM, pp. 247–256.
[14] ———, *Zigzag persistent homology and real-valued functions*, in Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry, SCG '09, New York, NY, USA, 2009, ACM, pp. 247–256.
[15] Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian, *On the local behavior of spaces of natural images*, International Journal of Computer Vision, 76 (2008), pp. 1–12.
[16] Joseph Minhow Chan, Gunnar Carlsson, and Raul Rabadan, *Topology of viral evolution*, Proceedings of the National Academy of Sciences, 110 (2013), pp. 18566–18571.
[17] Chao Chen and Michael Kerber, *Persistent homology computation with a twist*, in Proceedings 27th European Workshop on Computational Geometry, 2011.
[18] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer, *Stability of persistence diagrams*, Discrete & Computational Geometry, 37 (2007), pp. 103–120.
[19] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25. `http://www.cise.ufl.edu/research/sparse/matrices`.
[20] Vin de Silva and Gunnar Carlsson, *Topological estimation using witness complexes*, in Proceedings of the First Eurographics conference on Point-Based Graphics, Eurographics Association, 2004, pp. 157–166.
[21] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson, *Dualities in persistent (co)homology*, Inverse Problems, 27 (2011).
[22] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson, *Persistent cohomology and circular coordinates*, Discrete & Computational Geometry, 45 (2011), pp. 737–759.
[23] D. DeWoskin, J. Climent, I. Cruz-White, M. Vazquez, C. Park, and J. Arsuaga, *Applications of computational homology to the analysis of treatment response in breast cancer patients*, Topology and its Applications, 157 (2010), pp. 157 – 164. Proceedings of the International Conference on Topology and its Applications 2007 at Kyoto; Jointly with 4th Japan Mexico Topology Conference.
[24] Tamal Krishna Dey, Fengtao Fan, and Yusu Wang, *Graph induced complex on point data*, in Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry, SoCG '13, New York, NY, USA, 2013, ACM, pp. 107–116.
[25] Tamal K. Dey, Fengtao Fan, and Yusu Wang, *Computing topological persistence for simplicial maps*, in Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG'14, New York, NY, USA, 2014, ACM, pp. 345:345–345:354.
[26] H. Edelsbrunner and Morozov D., *Persistent homology: Theory and practice*, in Proceedings of the European Congress of Mathematics, 2012, pp. 31–50.
[27] H. Edelsbrunner and J. Harer, *Persistent Homology — A Survey*, in Surveys on Discrete and Computational Geometry, vol. 453, Amer Mathematical Society, 2008, p. 257.
[28] Herbert Edelsbrunner and John Harer, *Computational Topology: An Introduction*, Applied mathematics, American Mathematical Society, 2010.
[29] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian, *Topological persistence and simplification*, Discrete & Computational Geometry, 28 (2002), pp. 511–533.
[30] Herbert Edelsbrunner, Dmitriy Morozov, and Valerio Pascucci, *Persistence-sensitive simplification functions on 2-manifolds*, in Proceedings of the Twenty-second Annual Symposium on Computational Geometry, SCG '06, New York, NY, USA, 2006, ACM, pp. 127–134.
[31] Herbert Edelsbrunner and Ernst P. Mücke, *Three-dimensional alpha shapes*, ACM Trans. Graph., 13 (1994), pp. 43–72.
[32] S. Eilenberg and N.E. Steenrod, *Foundations of algebraic topology*, Princeton Mathematical Series, Princeton University Press, 1952.
[33] Guojin Gan, Chaoqun Ma, and Jianhong Wu, *Data Clustering: Theory, Algorithms, and Applications*, SIAM, Philadelphia, PA, 2007.
[34] R. Ghrist, *Elementary Applied Topology*, Createspace, 2014. ed. 1.0.
[35] A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi, *A survey of statistical network models*, Found. Trends Mach. Learn., 2 (2010), pp. 129–233.
[36] Allen Hatcher, *Algebraic Topology*, Cambridge University Press, Cambridge, New York, 2002.
[37] Y. Hiraoka, A. Hirata, E. G. Escolar, and Y. Nishiura, *Persistent homology and many-body atomic structure for medium-range order in the glass*, Nanotechnology, 26 (2015).
[38] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, no. Bd. 157 in Applied Mathematical Sciences, Springer, 2004.
[39] Matthew Kahle, *Random geometric complexes*, Discrete & Computational Geometry, 45 (2011), pp. 553–573.

[40] Leonard Kaufman and Peter J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley and Sons, Inc., New York, NY, 1990.
[41] M. Kramar, A. Goullet, L. Kondic, and K. Mischaikow, *Persistence of force networks in compressed granular media*, Phys. Rev. E, 87 (2013), p. 042207.
[42] V. Kurlin, 2015. `http://kurlin.org/projects/persistent-skeletons.cpp`.
[43] V. Kurlin, *A one-dimensional homologically persistent skeleton of an unstructured point cloud in any metric space*, Computer Graphics Forum, (2015).
[44] Los Alamos National Laboratory, *HIV Database*. `http://www.hiv.lanl.gov/content/index`.
[45] Stanford University Computer Graphics Laboratory, *The stanford 3d scanning repository*. `https://graphics.stanford.edu/data/3Dscanrep`.
[46] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec, *The gudhi library: Simplicial complexes and persistent homology*, in Mathematical Software, ICMS 2014, Hoon Hong and Chee Yap, eds., vol. 8592 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 167–174. Software available at `https://project.inria.fr/gudhi/software/`.
[47] Konstantin Mischaikow and Vidit Nanda, *Morse theory for filtrations and efficient computation of persistent homology*, Discrete & Computational Geometry, 50 (2013), pp. 330–353.
[48] Dmitriy Morozov, *Dionysus*. Software available at `http://www.mrzv.org/software/dionysus/`.
[49] Vidit Nanda, *Perseus, the persistent homology software*. Software available at http://www.sas.upenn.edu/ vnanda/perseus.
[50] Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson, *Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival*, Proceedings of the National Academy of Sciences of the United States of America, 108 (2011), pp. 7265–7270.
[51] Mathew Penrose, *Random Geometric Graphs*, Oxford University Press, Oxford, UK, 2003.
[52] P. Perry and V. de Silva, *Plex*, 2000–2006. `http://mii.stanford.edu/research/comptop/programs/`.
[53] Giovanni Petri, Martina Scolamiero, Irene Donato, and Francesco Vaccarino, *Topological strata of weighted complex networks*, PLoS ONE, 8 (2013).
[54] Jyamiti research group (Prof. Tamal K. Dey) Ohio State University, *SimpPers*, 2014. `http://web.cse.ohio-state.edu/~tamaldey/SimpPers/SimpPers-software/`.
[55] Satu Elisa Schaeffer, *Graph Clustering*, Computer Science Review, 1 (2007), pp. 27–64.
[56] Donald R. Sheehy, *Linear-size approximations to the Vietoris-Rips filtration*, Discrete & Computational Geometry, 49 (2013), pp. 778–796.
[57] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams, *JavaPlex: A research software package for persistent (co)homology*, in Proceedings of ICMS 2014, Han Hong and Chee Yap, eds., Lecture Notes in Computer Science 8592, 2014, pp. 129–136. Software available at `http://appliedtopology.github.io/javaplex/`.
[58] D. Taylor, F. Klimm, H. A. Harrington, M. Kramar, K. Mischaikow, M. A. Porter, and P. J. Mucha, *Topological data analysis of contagion maps for examining spreading processes on networks*, Nature Communications, 6 (2015).
[59] C. M. Topaz, L. Ziegelmeier, and T. Halverson, *Topological Data Analysis of Biological Aggregation Models*, ArXiv e-prints, (2014).
[60] L. Vietoris, *Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen*, Mathematische Annalen, 97 (1927), pp. 454–472.
[61] Hubert Wagner, Chao Chen, and Erald Vuçini, *Efficient computation of persistent homology for cubical data*, in Topological Methods in Data Analysis and Visualization II, Ronald Peikert, Helwig Hauser, Hamish Carr, and Raphael Fuchs, eds., Mathematics and Visualization, Springer Berlin Heidelberg, 2012, pp. 91–106.
[62] A.J. Zomorodian, *Topology for Computing*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2009.
[63] Afra Zomorodian, *Technical section: Fast construction of the Vietoris-Rips complex*, Comput. Graph., 34 (2010), pp. 263–271.
[64] ———, *The tidy set: A minimal simplicial set for computing homology of clique complexes*, 2010. SCG10.
[65] Afra Zomorodian and Gunnar Carlsson, *Computing persistent homology*, Discrete Comput. Geom., 33 (2005), pp. 249–274.