A Survey on the High-Performance Computation of Persistent Homology

Nicholas O. Malott, *Member, IEEE,* Shangye Chen, *Student Member, IEEE,*
and Philip A. Wilsey, *Senior Member, IEEE*
E-mail: malottno@mail.uc.edu, chen3se@mail.uc.edu, philip.wilsey@uc.edu

Abstract—*Persistent Homology* is a computational method of data mining in the field of Topological Data Analysis. Large-scale data analysis with persistent homology is computationally expensive and memory intensive. The performance of persistent homology has been rigorously studied to optimize data encoding and intermediate data structures for high-performance computation. This paper provides an application-centric survey of the High-Performance Computation of Persistent Homology. Computational topology concepts are reviewed and detailed for a broad data science and engineering audience.

Index Terms—Persistent Homology; High-Performance Computing; Topological Data Analysis; Data Science; Data Mining

1 Introduction

Researchers, scientists, and analysts need to extract information from big data. Data is being collected in enormous sizes and at ever-increasing rates, leading to an analysis of new tools, techniques, and best practices for data science. Industries are investing heavily in high performance computational tools for advanced analytics. Data analysis with topology has demonstrated excellent results in several fields [1], [2], [3], [4], [5]. Researchers are studying topological methods to mine data characteristics, especially for complex, multivariate data. This type of analysis is generally classified as *Topological Data Analysis* (TDA).

TDA techniques characterize the structure of data; they have been successfully used to classify embedded structures in large, complex data sets. For example, TDA approaches have been used in network analysis [6], [7], [8], brain artery classification [9], images and movies [10], [11], [12], [13], [14], [15], protein analysis [16], [17], [18], and genomic sequences [6], [19], [20], [21]. TDA techniques have the ability to identify structure despite certain deformations of a space, leading to discovery of relationships not discernible by conventional methods of analysis [2], [3].

One of the principal methods in TDA is *Persistent Homology* [22], [23]. Persistent homology is a technique for identifying the topological features of a point cloud at different spatial resolutions. More precisely, multiple views of the data are created by considering the connectivity of the points at different distances. These views are sorted by their connectivity distance and collectively called a *filtration* of the point cloud. Persistent homology then examines each member of the filtration sequentially and measures homologies (*i.e.,* features such as connected components, loops, voids, and so on) persisting through the different filtrations. Each individual feature is identified as it first appears (denoted *birth*) and when it disappears (denoted *death*) in the filtration. Persistent homology has specifically been used in fields such as bioinformatics [20], [24], [25], networking [26], [27], classification [28], [29], [30], pattern recognition [9], [31], [32], and more. Persistent homology can discover topological features (such as loops and voids) embedded in higher-dimensional spaces [33], [34].

Although persistent homology has demonstrated promising results for data analysis, the approach suffers from exponential space and run-time complexity [5]. This leads to the pursuit of high-performance approaches to be exploited for computing persistent homology on large and high-dimensional data sets. This paper surveys approaches for high-performance persistent homology to provide context to engineers and data scientists entering the field.

Various aspects of persistent homology have been previously covered through surveys and tutorials, most notably Chazal *et al* [1], Otter [5], Zhu [35], Fugacci *et al* [36], and Pun *et al* [37]. Chazal provides an introduction to topological data analysis including where PH fits within the tools of TDA. The survey covers the theoretical framework of the approach alongside an example with protein and sensor data. Otter surveys current techniques for persistent homology and highlights multiple libraries and their uses. A roadmap is presented for the computation of persistent homology with description of different data types, complex types, and statistical interpretation of topological summaries. Otter presents a comprehensive analysis of several benchmark data sets for persistent homology and results from several different software libraries. Zhu presents a brief tutorial on persistent homology as it relates to natural language processing, demonstrating a similarity filtration on several text examples. Fugacci contains a comprehensive background of persistent homology alongside an interactive web tool for introduction. Pun discusses the practical application of persistent homology based machine learning models and the combination of different topological feature selections with machine learning pipelines. Pun also details a roadmap for practical application of persistent homology-based

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Cincinnati, Cincinnati, OH, 45221–0030.*

machine learning models. Each of these studies, along with several other surveys [38], [39], [40], effectively describe the background concepts of persistent homology.

This survey introduces current techniques for implementing and optimizing the computation of persistent homology, including discussion of the key algorithm steps. While previous publications have focused on the theoretical background of persistent homology, this survey instead examines computation of persistent homology on sizable data sets to identify observed and measured bottlenecks of the approach. The material is intended for an engineering audience interested in the implementation methods for computing persistent homology. As a result, the more rigorous mathematical concepts and definitions of persistent homology are omitted to provide an entry-point for data scientists and engineers to study the practical implementation techniques. Several high-performance sequential, parallel, and distributed libraries are reviewed to introduce current state-of-the-art implementations of persistent homology.

The remainder of this paper is organized as follows. Section 2 illustrates several real-world applications of persistent homology. Section 3 provides technical description of the computation, coupled with related concepts. Section 4 details strategies for computing persistent homology. Section 5 presents several high-performance sequential, parallel, and distributed libraries. Section 6 examines the performance of persistent homology algorithms in various experimental scenarios. The key ideas and concepts of this survey are summarized in Section 7.

2 Application Background

Persistent Homology (PH) is a data mining technique to characterize topological features within a data set. Topological features are algebraic structures identified by their dimension: H_0 represents connected components in the data, H_1 algebraic loops, H_2 voids, and so on. PH provides a way to mine evolutionary relationships of the topological structures present in a point cloud with respect to filtration. That is, PH organizes the point cloud into a collection of graphs (each graph is a subset of a *complex* [41] defined in Section 3.1) evaluated at different filtration values. PH then analyzes the ordered collection of graphs to discover the filtrations at which each topological feature appears (denoted *birth*) and then disappears (denoted *death*) from the graph collection. Applications can then be constructed to classify the set of topological features as they persist across the filtrations.

Many applications of persistent homology examine the input data in terms of proximity: how near or far points are from one another in a metric space. This organization of data forms a point cloud in the dimension of the data. Proximity filtration examines the change in the point cloud as the distance between points being connected is increased. This creates many different topologies of the point cloud, each a graph representing all points being connected less than the filtration distance. In sequence, these topological spaces created by the filtration can be used to track the formation (and collapse) of the algebraic structures present. Features that exist for a large interval of the topological spaces are considered persistent, and can be used to estimate the homology of the space. This topic is explored further in Section 3.

In this section, the algorithm to compute PH is presented as a black-box with an input of a data set and an output of *persistence intervals* that track the topological structures found (Figure 1). The input data is evaluated with respect to a distance metric with PH to produce persistence intervals. In general, persistence intervals characterize the topological features identified as a 3-tuple $\langle H_d, \epsilon_{birth}, \epsilon_{death} \rangle$ where H_d is the dimension that the feature occupies, ϵ_{birth} is the connectivity distance that the feature first appears, and ϵ_{death} is the connectivity distance that the feature is no longer present. Persistence intervals are evaluated with respect to a distance metric and can be utilized to compare topological structures between data sets.

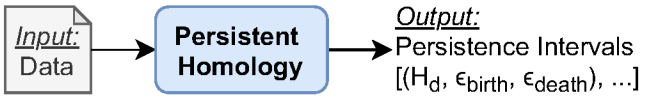


Fig. 1. Black-box diagram for PH application descriptions. The set of resultant persistence intervals represent topological structures identified from the input data.

There are numerous demonstrated uses for PH in scientific research. The remainder of this section details several examples for object classification, namely: brain artery tree analysis, and protein analysis. Object classification is a broad class of examining shapes, such as geometric objects, text, triangulated meshes, and high-dimensional features of a space. PH provides measures of these shapes that can characterize the underlying structures in the data. This approach is similar when examining brain artery trees representing $\mathbb{R}^3$ paths of brain arteries of labeled patient data. The underlying structure of the data enables classification from the output of PH. Finally, several protein analysis applications are described to connect the approach to biological data. PH can provide structural analysis over any type of data provided a distance metric between samples is defined.

This section provides brief examples of the utility of PH (a background in topology and homology groups is assumed [38], [39], [42]). Applications of PH are presented to familiarize the reader with its use in data mining and analysis prior to algorithm specification. The details of computing PH are continued in Section 3.

2.1 Object Classification

Object classification is common in the fields of data mining and machine learning. TDA, based heavily in graph theory, can provide relative characteristics of point clouds by examining the graphs constructed when connecting points. PH provides a representative view into these characteristics, often useful in differentiating dimensional structures. Some examples of object classification in PH include text recognition, $\mathbb{R}^2$ shape recognition, and triangulated mesh ($\mathbb{R}^3$) dissimilarity.

The MNIST data set [?] is a popular classification data set for algorithm evaluation, consisting of labeled images of handwritten digits for recognition. Classification with

PH typically examines the inferred homology of the data; that is, it is primarily concerned with persistent or "long" topological features identified from PH. In the domain of handwritten digits, a naive approach may recognize digits with a loop, such as "0"or "4", have an $\mathbb{R}^2$ (H_1) loop, while many digits have none. PH can classify some of the digits this way, while also including connected components (H_0) or additional features to further organize the space. Notably Garin *et al* recently demonstrated classification of the MNIST data set with a broader set of TDA techniques for supervised learning [**?**].

Triangulated meshes, such as points sampled from a continuous $\mathbb{R}^3$ manifold, can be analyzed with PH to classify different objects and structural representations. In these cases the long topological features represent salient features of the space; if the triangulated mesh densely approximates a continuous manifold, the long topological features will identify the homology of the manifold. Interestingly, the small topological features can provide additional insight into the structural composition [9], [43]. Not only can objects be classified by larger topological structures, such as manifolds encompassing voids of the space, but classification based on the distribution of points from smaller persistence intervals can differentiate results.

Persistent homology can be used to identify different clusters (through H_0 connected components) and shapes (H_1 loops) in $\mathbb{R}^2$ [34], [44]. PH has also been used for time series and spatial data clustering to extract significant features; these features may be utilized to compare topologically-similar objects, shapes, or clusters of a data set [34]. Transforms such as the time delay embedding (Takens embedding [45]) have been utilized to classify signals such as gravitational waves detection [46] and dynamic state detection [47]. In this case the embedding transforms the signal to identify periodic features using persistence intervals.

Construction of the complex with measures other than proximity can provide an alternate recognition of shapes, as demonstrated by Carlsson *et al* [23] through the use of filtered tangent complexes to recognize sharp corners and smooth edges. Shape recognition, in general, may require a broad analysis of different complex types and techniques to fit the desired application.

Moitra *et al* [21] introduce a method of classifying streaming data of real-world data sets examined under a sliding-window model. The technique is applicable to unbounded and evolving data streams, involving an online summarization of topological structure that can trigger an offline step to compute the full persistence intervals. Their study demonstrates the ability to identify reticulate genomic exchanges during the evolution of two viruses: Influenza A and HIV. The results are promising for unbounded streaming persistent homology applications.

Many more examples of PH can fall into the broad category of object classification. In particular, two studies that use PH are presented below to provide more concrete and detailed examples of the utility of PH for object classification.

2.2 Brain Artery Trees

Bendich *et al* [9] classifies patients from the PH of their respective brain artery trees. The study presents evidence that the computed persistence intervals correlate with characteristics of the examined patients. The input data for the study consists of labeled patients: each with a respective brain artery scan, sex, age, and dominant hand. The scanned trees represent blood vessels in the brain identified by Magnetic Resonance Angiography (MRA) images in $\mathbb{R}^3$. Bendich's study of the connected components and loops persisting in the brain artery trees identify relationships to the labeled patients for Sex and Age characteristics.

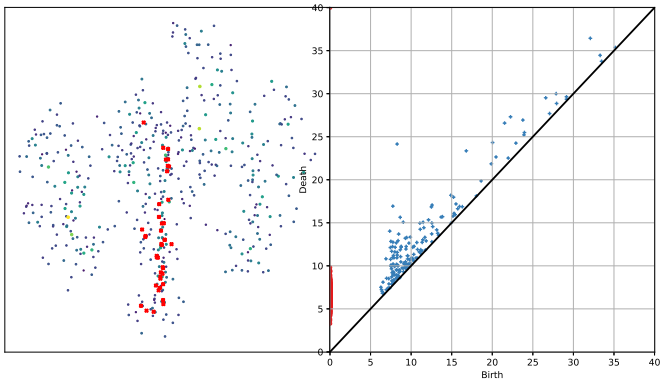


Fig. 2. $\mathbb{R}^2$ projection of MRA brain artery tree for a single patient (left) and persistence diagram (right) for H_0 (red) and H_1 (blue) features identified. Analysis of the most persistent features reveals a correlation of patients brain artery scans with their labeled features age and sex [9], [48].

The set of brain artery trees, consisting of roughly $100k$ points each, were individually subsampled to several thousand points to compute PH over. This subsampling preserves the spatial relationship of points in the brain artery trees while limiting the PH computation from exceeding memory limits of typical systems. This is a common approach when handling data beyond the memory limits of a system; exploration of this and other techniques for improving PH memory bounds are covered in Section 4. An example of the subsampled analysis of one patient's brain artery tree alongside the resultant persistence diagram is displayed in Figure 2. Correlation was identified between patient labels with the most-persistent features, those with the longest intervals, displayed away from the 45-degree line in the persistence diagram. Shorter intervals, generally considered noise within the point cloud, lie closer to the 45-degree line. The red points indicate H_0 connected components while blue H_1 points represent $\mathbb{R}^2$ loops oriented in the point cloud.

The brain artery trees were recently examined by Malott *et al* using *Partitioned Persistent Homology (PPH)* [48] to reconstruct the persistence intervals lost from subsampling of the data. The technique approximates the large topological features using centroids and smaller topological features using regional reconstructions about the partitions. In a similar manner, the results indicate that the differentiation of the brain artery trees can determine the patient's age and sex with significant accuracy. The regional reconstruction of smaller topological features introduces an additional increase in the classification, confirming the original study and extending the analysis further.

Brain artery trees are a fascinating use case where persis-

tent homology has been applied and correlated with labeled data. The structures of the arteries are identifiable through the H_0 and H_1 features over the point cloud filtration. These topological structures, captured in the output persistence intervals, have significant correlation with the labeled patient features.

2.3 Protein Analysis

Protein analysis is an inherently complex field that often requires massive computational resources for analysis. Combinations of proteins may be analyzed as basic primary structures such as amino acids or more complex structures such as conformational chain interactions. One recent topological classification method utilizing Molecular Topological Fingerprints (MTFs) was introduced by Xia *et al* [18] to track geometric origins of topological invariants of proteins. MTFs are demonstrated for protein characterization, identification, and classification. The work establishes a topology-function relationship of proteins that have been built on in several other experiments and tools to further topological protein analysis.

A study by Cang *et al* [16] explores the use of persistent homology characteristics for protein classification. They introduce a MTF-based support vector machine (SVM) classifier and validate the tool against several experiments: protein drug binding, classification of hemoglobin molecules, identification of protein domains, and classification of protein superfamilies. Features from the persistence intervals are carefully selected for the SVM model and are detailed in the study; these features differ slightly than those used for the standard MTF approach in [18].

Kovacev-Nikolic *et al* [49] apply PH to the maltose-binding protein (MBP), a complex bio-molecule with 370 amino acid residues. PH detects the conformational changes between closed and open forms of the MBP; the study confirms there is a statistically significant difference between these two forms. The approach also demonstrates how persistence landscapes [50], another method of analysis for persistence intervals, can be applied to machine learning methods such as SVM. Additionally, the authors determine the sites of interest correspond with the most persistent loop of the filtered complex, a finding not observed in the classical model. Chazal [1] provides a tutorial and python code for basic analysis of the data from [49] to compare the persistence diagrams of the MBP.

In Benzekry *et al* [51], a linear correlation is identified between PH and cancer patient data when examining the Betti numbers. The relationship predicted the most impactful protein on cancer progression within the protein-protein interaction network. In addition, by removing the individual protein node from the protein-protein interaction network and re-computing PH, researchers were able to evaluate whether this inhibition will improve patient survival rate.

Persistent homology has demonstrated capabilities in the field of protein analysis through several applications and continues to be of great interest to bioinformatics in general. The capabilities of PH to uncover topological features of the space can provide discernible insight into structures present in input data. Additional research into applications of PH continue to be uncovered and reported within the research community.

3 Persistent Homology

The application of Persistent Homology to various data mining and machine learning applications demonstrates promising results. However, the computational complexity of persistent homology is exponential in both time and space; this can limit the application of PH to relatively small data sets. In this section the detailed steps for computing persistent homology are provided and considered with respect to the computational complexity of the algorithm.

Figure 3 presents an overview of the main steps in the PH algorithm that are covered throughout this section. First, the PH algorithm is split into three parts, namely: *complex construction*, *complex filtration*, and *boundary matrix reduction*. Although the complex filtration may influence the complex construction technique, each of these steps is described sequentially. The results of PH remain the same as Section 2: compute and output persistence intervals that describe the topological features identified in the data. The persistence intervals can then be utilized for the desired application.

The encoding of the data into a complex is discussed in Section 3.1. This encoding is used throughout the computation and typically grows exponentially based on the number of points. The filtration of the complex is presented in Section 3.2. The filtration of the complex defines the metric space utilized to generate the boundary matrix. Boundary matrix reduction is the final step in the technique; it is detailed in Section 3.3. This step generates persistence intervals, that can then be interpreted and analyzed using the techniques described in Section 3.4.

3.1 Complex Construction

The first step in computing the persistent homology requires an encoding of the point cloud into a relational, graph-like structure referred to as the *complex* [41]. Complex construction and filtration requires selection of a metric to compare individual samples. In spatial data this is typically the Euclidean distance; in other studies the metric may need to be tailored to the application for measuring the difference between any two samples. This paper covers proximity complexes that examine the topological spaces filtered over the chosen metric, regardless of whether that metric describes spatial relationships of the vertices.

The *simplicial complex* [41] stores *simplices*; a *simplex* represents a basic structure within the point cloud. Simplices take the form of hyper-tetrahedra; more abstract representations of the space may provide alternate methods of analysis. Other domain-specific complexes exist such as cubical complexes [11] for image data. In this survey we will focus primarily on the simplicial complex and hyper-tetrahedra[1] to encode a point cloud.

The simplicial complex can be recognized as a higher-dimensional generalization of graphs. Simplicial complexes are applied to computational geometry to approximate continuous mathematical shapes such as surfaces and curves. The mathematical representations of simplicial complexes

1. A hyper-tetrahedron is the generalization of a triangle or tetrahedral region of space to d-dimensions. For example the 0-simplex is a point; the 1-simplex is a line segment composed of two points; the 2-simplex is a triangle face composed of three vertices; the k-simplex is a k-dimensional polytope composed of $k+1$ vertices.

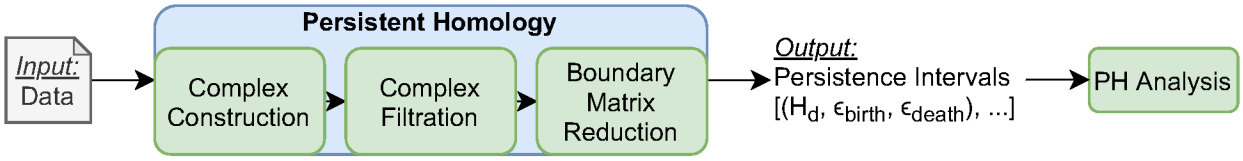


Fig. 3. The main steps of computing PH and processing PH results are shown. The steps with green backgrounds are discussed in this section. The computation of persistent homology is broken into its principal components, namely: Complex Construction, Complex Filtration, and Boundary Matrix Reduction. A short overview of persistence interval analysis is provided in Section 3.4 for PH applications.

have topological character and combinatoric specialty, making them beneficial for persistent homology. Higher-dimensional features can be recognized by constructing generalizations of the point cloud at different dimensional resolutions using simplices. This recognition and extraction of features is defined more succinctly in Section 3.3.

There are several methods to construct the complex including Vietoris–Rips [56], Witness [57], Čech [58], and Flag [56] complexes. The remainder of this section details these types of complexes and gives brief descriptions of how the constructed complex differs. Table 1 compares each complex with their principle use and includes a statement of their construction complexity, storage complexity, and a graph example for each. The complexity of the persistent homology algorithm begins with the size of the complex which can inhibit the computation on large data sets.

The *Čech complex* [58] (or proximity graph) is constructed from a point cloud in any metric space. The Čech complex is widely used when higher-order intersections of the points in the point cloud are needed to extract persistence intervals. The Čech complex first inserts the proximity graph into the complex data structure and then expands the complex if necessary. The construction complexity for the Čech complex is $O(N^2 + N2^n)$, where n is the average number of neighbors in each cell (simplex) and N is the number of cells (simplices) [52]. The storage complexity of the Čech complex is shown in Table 1 as functions of the cardinality of N [5].

The *Vietoris–Rips (VR) complex* [56], also called a *Rips complex*, is a simplicial complex induced from a one-skeleton graph. The vertices in the complex correspond to the input points; an edge is present if and only if the maximum distance is smaller than the bounding radius, ϵ_{max}. The Rips complex can then generalize proximity (ϵ-ball) graphs to higher dimensions through combinations of the lower dimensional one-skeleton graph. Construction requires first building the proximity graph and inserting it into the complex data structure. Then an expansion process adds the simplices corresponding to cliques until the dimension arrives at the maximum dimension for persistent homology computation. The Rips complex is an approximation of the Čech complex; construction of a Čech complex is computationally more expensive than the Rips complex. However, the VR complex will generally contain more higher-order simplices than the Čech, such as the tetrahedron in the respective example images of Table 1. The noted tetrahedron still forms in the Čech complex, but later in the filtration once the proximity balls have encompassed the tetrahedron.

The *Witness complex* [57] is a complex defined on two sets of points in R^D: the set of witnesses, W, and the set of landmark points, L. Typically the set of landmarks is a subset of the witnesses. Otter *et al* [5] states that the storage complexity of the witness complex is $O(2^{|L|})$, where $|L|$ is the number of landmark points. Boissonnat and Maria [53] show that the witness complex construction requires $O((|K| + |W|)k^2 D_m)$ insertions, where $|W|$ is then umber of witness points, D_m is $log|L|$, k is any integer in $\{1, ..., |L| - 1\}$ that defines the k-skeleton of the witness complex, and $|K|$ is the number of faces in the complex. Landmarks represent the vertices of the simplicial complex and witnesses can identify which simplices are inserted through a predicate. When the data set is large, the witness complex is a good choice for conducting data reduction on the data set and helps alleviate the memory constraints when directly applied to large data in R^D. As shown in Table 1, the example image for the witness complex demonstrates the construction process from left to right.

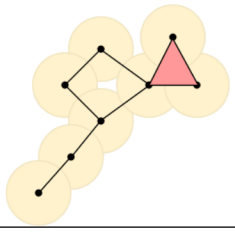
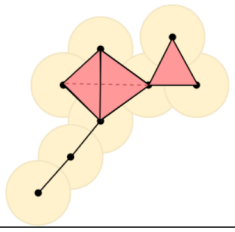
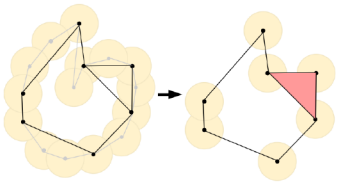
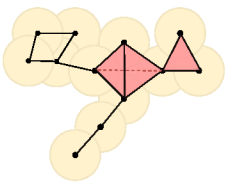
Another well-known complex is the *Flag (clique) complex* [56]. The Flag complex is a simplicial complex in which all minimal non-faces are two-element sets. In other words, if all edges of a potential face in a Flag complex are in the complex, the face should also be in the complex. In some conditions, if a set $\mathcal{S}$ of vertices that is not itself part of the complex but each pair of vertices in $\mathcal{S}$ belongs to some simplex in the complex, this simplicial complex is called an empty simplex. A Flag complex is a simplicial complex without empty simplices. Since the Flag complex is closely related to the graph, it is highly efficient in the field of studying directed networks in general and especially neural networks. Typically insertion into a flag complex is through a weighted graph; using the proximities of the complete graph would be identical to the VR complex. Chambers *et al* [55] show that the flag complex has the same storage complexity as the Rips complex. The construction complexity of a Flag complex is $O(n^k k^2)$, where k is the vertices number and n is the number of input points [54].

The construction of a simplicial complex is one of the most common ways to associate a filtration to a point cloud for computing persistent homology. The Vietoris–Rips complex provides fast construction alongside a compressed memory footprint that has proven to be useful in persistent homology. In addition the Rips complex can utilize a particular distance function or distance matrix to compute the corresponding filtrations without requiring physical positions. The remainder of this paper will focus on the Rips complex.

3.2 Complex Filtration

Filtration of the complex involves examining the nested subsets of simplices representing the appearance and dis-

TABLE 1
Complexity comparison of alternate representations for simplicial complexes. Graph example images depict structural differences between complexes, such as the inclusion of a tetrahedron in the Vietoris–Rips complex over the Čech.

Complex name	**Čech**	**Vietoris–Rips**	**Witness**	**Flag**
Used for	Points in Euclidean space	Distance matrix	Curves and surface in Euclidean space	Directed networks in Graphs
Construction Complexity	$O(N^2 + N2^n)$ [52]	$O(n_K^{2.376})$	$O((\|K\| + \|W\|)k^2 D_m)$ [53]	$O(n^k k^2)$ [54]
Storage Complexity	$O(2^N)$ [5]	$O(n_K^2)$	$O(2^{\|L\|})$ [5]	$O(n_K^2)$ [55]
Example image				

appearance of topological features. In a simplicial complex K, filtration is an ordering of the simplices of K such that all prefixes in this ordering must be the subcomplexes of K. In a sense the subcomplexes represent the addition of simplices (edges, triangles, tetrahedrons) to the point cloud as ϵ grows. A complex with filtration is denoted as a *filtered complex* K_F.

The filtration occurs over the metric used to construct the simplicial complex; when the Euclidean distance is used the connectedness of points in a point cloud are examined. The filtration is varied by a scalar parameter, ϵ. As ϵ increases the points become more connected, representing different topologies of the point cloud. Topological features identified during this filtration can be identified by their birth, recorded as ϵ_{birth}, and their disappearance, ϵ_{death}. Tracking of the individual features, births, and deaths requires examination of the filtered complex through the boundary matrix representation, detailed further in Section 3.3.

Figure 4 illustrates the VR complex filtration of a point cloud along with the corresponding persistence intervals. The top portion of the figure plots a synthetic point cloud at varying levels of ϵ distance. The corresponding homology groups H_0 (connected components) and H_1 (loops) identified in the point cloud are charted in the lower portion of Figure 4. Each individual bar in the plot represents a topological feature. Topological features are captured in persistence intervals, which include the dimension of the topological feature H_d, the birth of the feature ϵ_{birth}, and the death of the feature ϵ_{death}. This filtration demonstrates how features in the point cloud become more connected as ϵ is increased from 0.

The size of the VR complex grows exponentially in size with both dimension and number of points. This filtration can be limited by setting a bounding radius, ϵ_{max}, which restricts the range of connectivity distances to a smaller subset of simplices; this limit leads to a reduced space-time complexity. With a proper selection of ϵ_{max}, connections that do not affect the desired topological features may be removed and not reported in the persistence intervals. This filtration can thus significantly reduce the memory and runtime complexities of persistent homology but limits the identification of topological features to those smaller than ϵ_{max}.

The simplicial complex paired with the filtration, K_F, is required to extract the boundary matrix and perform reduction to identify persistence intervals. This process is described in the next section.

3.3 Boundary Matrix Reduction

A convenient way to extract the persistence intervals from the filtered simplicial complex is to reduce a matrix recording the incidences between dimensional simplices. Given a simplicial complex K, the d^{th} boundary matrix can be represented as the rows with σ_{d-1}-simplices and the columns with σ_d-simplices. Incidences where the σ_{d-1}-simplex is a face of the σ_d-simplex in the matrix are recorded and subsequently reduced to extract ϵ_{birth} and ϵ_{death} times for features in the complex. This process is repeated for each dimension up to H_{max} (a user defined parameter that is bounded above by the dimension of the point cloud).

In summary, the boundary matrix representation enables identification of algebraic chains reducing to zero, indicating the presence of loops, voids, and so on. These chains can be recognized from the face relationship of the simplex; the birth and death times are the filtration values of the complex where the chain first appears (and reduces to zero) and disappears (is covered by additional simplices). Multiple features can be identified simultaneously, in multiple dimensions, and of different sizes in the space with the technique, demonstrating the utility of PH.

Typically the approach has two parts: generation of the boundary matrix and extraction of persistence intervals using matrix reduction. Complex representations are designed to provide incident simplices efficiently for fast enumeration of the boundary matrix. Matrix reduction optimizations for extracting persistence intervals are generally algorithmic improvements that include co-homology [59], [60], twist and clear [61], co-reduction [62], and implicit matrix representation [63].

While each approach improves the performance of the boundary matrix reduction step, the boundary matrix itself is still very large, representing an array of size σ^2, where

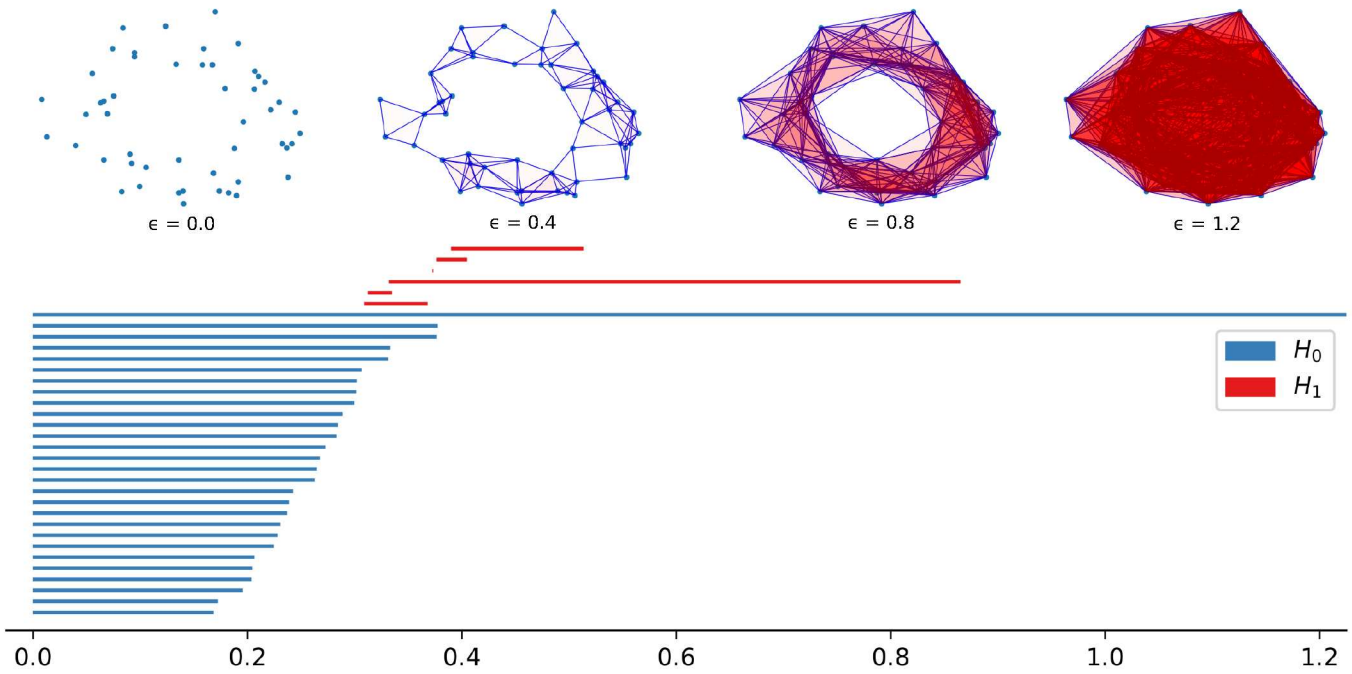


Fig. 4. Filtration of a point cloud (*top*) and the corresponding barcodes (*bottom*) showing birth and death of features at different filtrations of a Vietoris–Rips complex. The subcomplex relationship as $0 \leq \epsilon \leq \epsilon_{max}$ increases is depicted through the point cloud plots from left to right.

σ represents the total number of simplices in the complex. Storing the entire boundary matrix in memory prior to reduction is costly and typically limits the computation due to memory bounds of the system. Bauer [63] introduces a method to represent the Vietoris–Rips boundary matrix implicitly due to the combinatoric structure of dimensional simplices in the VR complex. This implicit representation of the boundary matrix significantly reduces the memory footprint of persistent homology, but has only been demonstrated with VR filtrations. These optimization concepts are discussed in greater detail in Section 4.3.

Computing the persistent homology on big data is still difficult with these optimizations. Using substantial computing hardware, current state-of-the-art tools can still only compute higher dimensional homologies for a few hundred points. Techniques to continue to improve the memory footprint and space-time complexity of the persistent homology algorithm are necessary in the approach for high-performance computing. Boundary matrix reduction and optimizations to the approach are a highly active area of study within persistent homology research.

3.4 Analysis of Persistent Homology Output

The output of the persistent homology computation is a set of *persistence intervals*. Persistence intervals provide a direct measure of the topological features identified within a point cloud. The resulting persistence intervals can be interpreted in several ways, most notably through diagrams and distance metrics. Visual diagrams, such as the persistence diagram and barcode diagram [5], give a qualitative analysis of the topological features. A comparison between two sets of persistence intervals with a distance metric can give a quantitative difference between the topological features identified [37], [64], [65]. Both have significance in TDA; there are varying methods to represent and utilize either of these outputs.

Each topological feature is represented by a persistence interval of the form: $\langle H_d, \epsilon_{birth}, \epsilon_{death} \rangle$, where H_d is the dimension of the feature and ϵ is a scale parameter. The values can be defined as: $\epsilon = (\epsilon_0, \epsilon_1, \cdots \epsilon_m)$ and $\epsilon_i < \epsilon_j \; for \; 0 \leq i < j \leq m$. Each 3-tuple of $\langle H_d, \epsilon_{birth}, \epsilon_{death} \rangle$ represents a single persistence interval. Depending on the topological structures present in the data, the full set of persistence intervals may contain zero or more intervals at each homology dimension H_1 to H_{max}; each persistence interval exists iff there is a corresponding topological feature in that dimension.

Direct visualization of high-dimensional data is difficult to provide. Several methods have been developed to extract the embedded structures from the data set and identify those structures qualitatively. For example, the seeds point cloud in Figure 5 is used to generate the *barcode* diagram and *persistence diagram* displayed. In the barcode representation, each persistence interval is plotted as a horizontal line colored by the dimension of the feature. The horizontal line begins at ϵ_{birth} and ends at ϵ_{death}. Generally long bars represent salient topological features while shorter intervals are interpreted as noise.

Alternatively in the persistence diagram representation (Figure 5), the birth and death time are plotted over the x-axis and y-axis respectively, colored again by dimension. Salient topological features in the persistence diagram representation lie far away from the 45-degree line; the line where the birth and death of the feature are relatively close. In general, persistent homology can fetch the birth and death times of the topological features and visualize the results

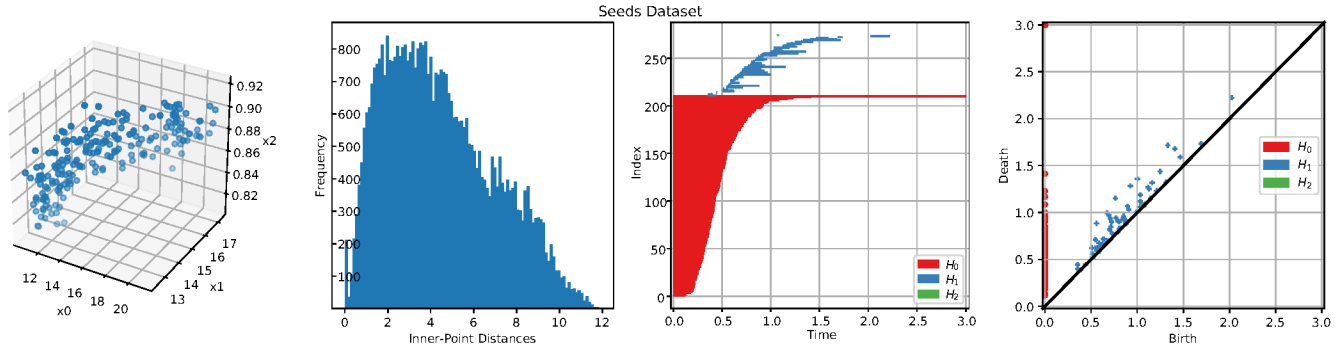


Fig. 5. Example of barcode and persistence diagrams on the seeds classification data set. From left to right: (i) 3D scatter plot for the leading 3 dimensions of the data set, (ii) the distribution of inner-point euclidean distances, and resulting persistence intervals (ii) as a barcode diagram and (iv) as a persistence diagram.

in the form of barcode or persistence diagrams to quickly identify homologies of the space.

Several measures are available for quantitative analysis between sets of persistence intervals, such as when classifying or determining the dissimilarity. The most notable measures include the Bottleneck distance, Wasserstein distance, and Heat Kernel distance. These metrics can be utilized for machine learning models such as SVM or other learning algorithms.

The *Bottleneck distance* [66] measures the largest difference between two sets of persistence intervals. The measurement is a bijection from one persistence diagram to another. For example, if Dgm_x and Dgm_y are two persistence diagrams, the bottleneck distance records the supremum of the distance between Dgm_x and Dgm_y and take the infimum over the bijection. The bottleneck distance is bounded by the Hausdorff distance, a popular measure in graph analysis.

The *Wasserstein distance* is more sensitive to details in the diagram compared to the Bottleneck distance, but requires additional properties to be stable. The Wasserstein distance is defined between probability distributions with a metric space M. It holds similar stability results and computes the maximum weighted matching in bipartite graphs. The Wasserstein distance represents the total difference in paired mappings between two sets of persistence intervals.

While persistence diagrams and the Wasserstein distance can form a metric space, sometimes the Wasserstein distance fails to directly apply persistent homology to the large class of machine learning techniques (such as SVM or PCA). As a result, several different kernel metrics have been proposed to provide stable learning metrics. One such approach is the continuous *Heat-based Kernel Distance (HKD)* metric [67]. HKD provides a multi-scale kernel based on the scale space theory. The kernel is used for the set of persistent diagrams and is defined through a feature map. Since the feature map is Lipschitz continuous with respect to the 1-Wasserstein distance the stability of persistent homology is preserved. The scale parameter in the kernel has the ability to control the robustness to noise. HKD provides a method to approximate the geodesic distance between two sets of persistence intervals. There are several other kernel metrics used for machine learning that have varying qualities pertaining to certain applications. A detailed analysis of kernel metrics with persistent homology is available in Pun *et al* [37].

4 High-Performance Computation of Persistent Homology

Persistent homology has computationally evolved as faster, more efficient techniques are discovered for various steps in the process. While these approaches provide moderate speed and memory improvements, the underlying concepts to build a complex, perform the filtration, and reduce the boundary matrix remain largely the same. This section expands upon the descriptions of these steps in Section 3 by detailing specifications used for building persistent homology applications in practice.

High-performance computation of persistent homology relies on balancing the memory footprint and the run-time for extracting persistence intervals. Compact data structures representing the complex and reduction matrix are necessary to control the memory footprint. Relatively small data sets can generate millions of simplices in higher dimensions, requiring efficient storage through representative complexes. These representations must also provide fast access for the generation of the boundary matrix for reduction.

Several steps are required to compute the persistent homology of an input point cloud. The first is to generate the complex from the input data. This process includes cleansing and normalization of the data, selection of an appropriate distance metric, dimensionality encoding using the distance matrix, and insertion into the complex.

Once the complex is formed the filtration is performed to construct the boundary matrix. The boundary matrix is then reduced to extract the persistence intervals. The boundary matrix reduction has been largely targeted to accelerate the extraction of persistence intervals; several optimizations have been studied over the standard algorithm [68]. Recent optimizations can significantly increase performance, including twist and clear [61], computing the co-homology [59], Morse matching [69], and emergent and apparent pairs [63] . One of the highest performance tools for computing the Vietoris–Rips persistence intervals using these enhancements is Ripser [63]. Ripser has made numerous advancements in the sequential computation of persistent homology and has inspired several multiprocessor and GPU-accelerated variants that have enabled computation on even larger data sets; these include Ripser++ [93], giotto [?], [94], and LHF [48], [76].

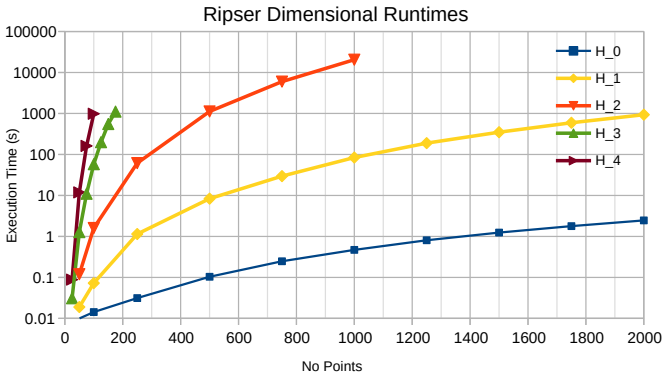


Fig. 6. Dimensional limitations of Ripser based on number of points in source point cloud on Ryzen Threadripper 1950X with 128GB of RAM.

Even with the reduced memory footprint and speedup of Ripser, the approach suffers as H_{max} and the number of points increase. The limitations of Ripser on a Ryzen Threadripper 1950X with 128GB of RAM on a synthetically generated d-sphere are shown in Figure 6. Detection of H_2 voids in the point cloud with no limiting radius (ϵ_{max}) exceeds the system's RAM capacity after several hundred points. Higher dimensions are even more severely limited (as the Figure 6 shows). Further exploration of the computational limitations of sequential persistent homology are explored in Section 6.1. The sequential algorithm for PH is limited to fairly small data sets, especially for higher dimensions of homology.

Parallel and distributed approaches provide more resources for the computation but still suffer based on the number of simplices generated in the simplicial complex. As the dimension of homology groups increases (H_{max}), the generated simplices grow exponentially and can quickly bound the algorithm. These memory limitations are the study of several optimizations and approximations proposed to reduce the complexity.

The remainder of this section details specific steps in the computation of persistent homology. Section 4.1 describes the creation of the simplicial complex, including normalization, distance metric and distance matrix construction, different complex types, and construction and storage of the complex. Section 4.2 describes filtration of the complex and boundary matrix construction in the approach. Finally, Section 4.3 identifies the steps for reducing the boundary matrix and provides brief detail on several of the notable optimizations for reduction.

4.1 Creation of Simplicial Complex

Several design decisions in a persistent homology approach are necessary for the construction of the complex. In the case of a simplicial complex, qualification of simplices and storage for the persistent homology algorithm need to be determined. These include defining a distance metric for quantifying connectedness of the space, a choice of the type of complex to be constructed, and the storage of the complex into an efficient data structure enabling fast processing and a low-memory footprint.

Storage of the simplicial complex requires fast insertion of simplices into the complex. Insertion of a single point can affect several or all members of the complex, depending on the graph complexity and ϵ_{max} limitation. The simplex tree [53] is a compressed data structure that is efficient in memory and management of the complex. Bauer provides a more efficient VR algorithm that eliminates the need for a static reduction matrix by storing the co-face relationships between the simplices [63]. In some cases the complex may need to be maintained online and updated as new data is discovered, necessitating a compressed and maintainable data structure. Fast construction of the simplicial complex combined with necessary interfaces for boundary matrix reduction are an area of study to reduce the complexity of the algorithm.

This section provides detailed information on construction of the complex, specifically using proximity-based simplicial complexes to represent an input point cloud. Various design decisions at this step affect the resulting persistence intervals and performance of the algorithm. The complex construction does not typically dominate the run-time and memory footprints in the standard approach, although inefficient data structures can become limiting as the complex grows in simplices.

4.1.1 Normalization of Input Point Cloud

One common step in any data analysis is some normalization of the input data to prevent features from dominating the distance metric used between points. Normalization techniques have been thoroughly explored in other studies, such as feature normalization and z-score normalization. Using a normalization technique may be necessary to scale and interpret the data. Figure 7 compares the resultant barcodes and distance matrix histogram for the *seeds* data set. *FeatureNormalization* scales all features into the range of 0 to 1. Alternatively using *ZScoreNormalization* centers the mean of each feature on 0.0 with a standard deviation of 1.0. The right plots showing the histogram distribution of the distance matrix indicate the normalization reduces inner-point distances while retaining the general distribution of distances between points.

Normalization notably has an effect on the persistence intervals produced by the persistent homology computation. Few directed studies on the effect of normalization on the persistent homology of a point cloud have been examined. Normalization can create additional small features which may be considered as noise as shown in Figure 7 in the barcode diagrams. While the plots show slight differences in the persistence intervals the overall concept preserves the significant topological features of the space. These topological features may experience slight shifts in their birth or death times due to the normalization. However limiting the scale of ϵ using normalization can lead to an easier selection of ϵ_{max} while preventing features of larger scale from dominating the distance between related points in the space.

4.1.2 Distance Metric and Matrix Construction

The complex encodes the spatial relationships of points within the point cloud. A distance metric is defined to measure the proximities of all points in the space; the Euclidean

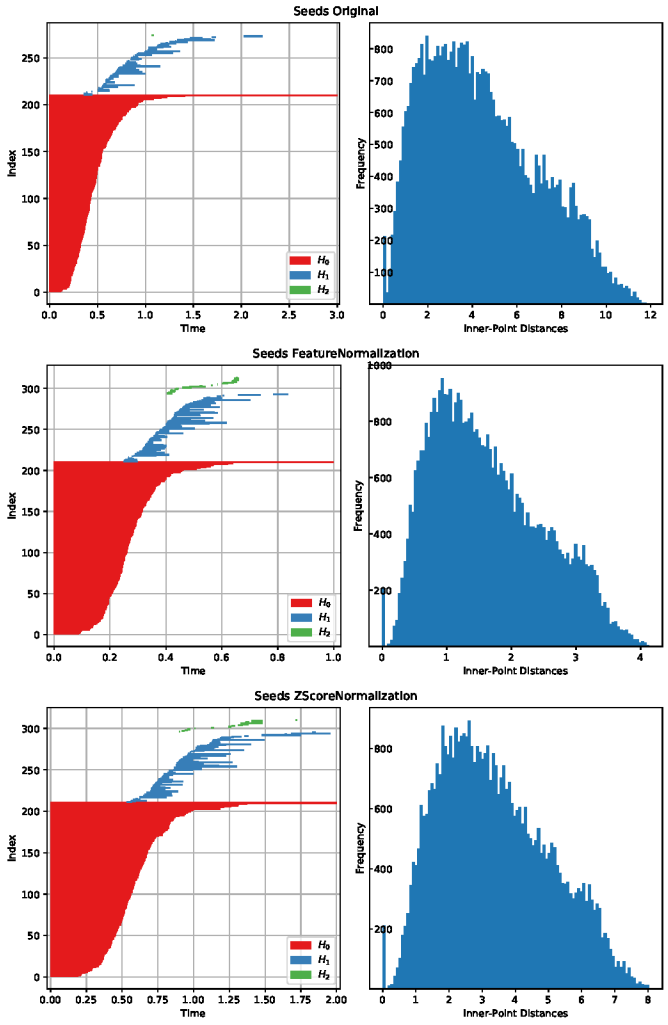


Fig. 7. Barcode Diagrams (left) and Distance Matrix Histograms (right) for the seeds data set with normalization prior to persistent homology. The top row shows the raw data results, the middle row with $FeatureNormalization$, and the bottom with $ZScoreNormalization$.

distance in this example. The inner point distances can be represented with a distance matrix for fast lookup when inserting points and distances for filtration in the complex. This simply involves computing the upper (or lower) triangular matrix entries due to the inherent symmetry of the distance matrix.

In the case of creating a triangular distance matrix, the distance metric computation requires $O(n \log n)$ insertions. Determination of the distance metric used can affect the speed of this representation. Euclidean distance requires exponential functions which are costly from a computation perspective; the Manhattan L_1 distance may provide better performance for larger point clouds and higher dimensions. Using an alternative distance metric will affect the scale of the resulting persistence intervals. Generally the distance matrix construction does not dominate the run-time or memory requirements of computing PH.

In some applications a different type of distance metric may be computed and used for the simplicial complex construction. The distance metric must be filterable with ϵ such that $0 \leq \epsilon \leq \epsilon_{max}$. Non-numeric data may need encoding or alternative distance metric definition such as hash functions to compute PH. These should be considered separately, and may be inserted as a preprocessed distance matrix for persistent homology.

4.1.3 Simplicial Complex and Filtrations

Several different simplicial complexes can be utilized to build the graph of the point cloud. A simplicial complex K requires that every simplex σ_i from K is also in K, and the non-empty intersection of two simplices, $\sigma_1, \sigma_2 \in K$, is a face of both σ_1 and σ_2. This representation precedes the algebraic construction of the boundary matrix.

Notable proximity complexes within topology include the *Vietoris–Rips* [56], *Witness* [57], *Čech* [58], and *Flag* [56] complexes. Each complex provides slightly different interpretations of the point cloud leading to differing resultant persistence intervals. Performance of the complex, in both construction and storage, should be considered when efficiency is required. Characteristic differences and uses of several notable complexes are described in Section 3.1.

The design of a persistent homology application must determine a specified type of complex to implement. Several libraries, such as GUDHI [70], provide users a choice of complex to represent the point cloud. The run-time and memory complexity of each type can become an integral factor in designing high-performance persistent homology applications. For example, the *Vietoris–Rips* complex is similar in nature to the *Čech* complex, but only requires pairwise intersection of simplices to form a higher dimension simplex. With the *Čech* complex the common point of intersection must be included in the ϵ-balls in order to consider the higher dimensional simplex. While fewer simplices will be inserted with the *Čech* complex, the evaluation of intersecting ϵ-balls is inefficient in higher dimensions.

It is evident that the choice of complex will lead to structural differences and change the resultant persistence intervals; in some cases a choice of complex may be influenced by the type of data being analyzed. Performance and initialization of the complex is an important design decision as well. Enumeration of higher-dimensional simplices becomes costly in some complex types, such as the *Čech* complex, whereas fast algorithms for higher-dimension construction of the *Vietoris–Rips* complex are readily available. For example, the incremental construction of the *Vietoris–Rips* complex detailed by Zomorodian [71] provides faster construction than the inductive or maximal construction for higher dimensional simplices. As the dimension of simplices generated increases the need for fast complex construction becomes readily apparent.

4.1.4 Simplicial Complex Construction and Storage

Construction of a simplicial complex from the distance matrix can be expensive. The number of generated simplices becomes exponential with higher-dimensional persistent homology and requires a fast insertion structure coupled with a low-memory footprint. Several techniques have been identified to store the simplicial complex: the simplex array list, the simplex tree, and the compressed annotation matrix. These are each detailed further to provide the strengths of each structure. An alternative approach to enumerating the

Vietoris–Rips complex alongside implicit boundary matrix representation is detailed in Section 4.2.

The *simplex array list* stores the simplices into lists based on the dimension of the simplex. For computing persistent homology in low dimensions, such as H_0 or H_1, the simplex array list is easily maintained. However, as the dimensions of persistent homology grow, the number of entries in each of the higher dimensional lists are exponential and quickly become costly to insert, sort, and retrieve for filtration and boundary matrix construction. The simplex array list is not well suited for high-dimensional persistent homology analysis, and is limited even in low dimensions by the number of simplices inserted. A faster structure is often necessary if the simplicial complex is to be constructed and stored.

The *simplex tree* stores the simplices into a tree structure that is significantly more efficient than the simplex array list [53]. The tree provides fast insertion, sorting, and memory-efficient storage of the simplices and their weights. The structure of the tree also enables fast lookup of cofaces, one of the primary approaches in constructing the boundary matrix for reduction.

Further improvement can be achieved using the *Compressed Annotation Matrix (CAM)* [72] which stores the simplicial complex in a separate representation from the cohomology groups. Results with CAM have shown considerable improvement in both time and memory performance for extracting persistence intervals. An implementation of CAM is available in the GUDHI library.

Ultimately the data structure and storage of the simplicial complex depends on several factors. First, the construction of the complex needs to be fast, indicating a structure that can provide efficient insertion of new points, simplices, and respective faces or homology groups. Once the complex is constructed, the reduction of the boundary matrix from the filtered simplicial complex needs to be also considered for fast access of faces and simplices of the complex.

4.2 Filtration and Boundary Matrix Construction

Filtration of the complex provides identification of the features present as the proximity parameter, ϵ, is varied from 0 to ϵ_{max}. Fortunately there are several techniques that reduce the complexity of examining the complex at each level of connectedness through the boundary matrix. The boundary matrix represents all incidences between simplices within the complex. For any simplicial complex K, the d^{th} boundary matrix is represented by rows of σ_{d-1}-simplices incident with columns of σ_d-simplices. Reduction of the boundary matrix identifies the algebraic relationships captured through simplices in the complex.

The faces of a simplex represent these incidences with σ_{d-1}-simplices. Storage of the faces enables fast construction of the boundary matrix from the simplices, which each are ordered by their weight and represent the augmented boundary matrix by face relationships. This can lead to a fast boundary matrix construction that enables extraction of the persistence intervals by ordered filtration. Reduction of the boundary matrix from left to right performs the filtration based on the weights (ϵ) of the simplices resulting in the output persistence intervals as described in Section 4.3.

Persistent homology requires construction of a large portion of the boundary matrix from the simplicial complex during reduction. The full boundary matrix construction creates a matrix of size $\sum_{d=0}^{H_{max}} \|\sigma_d\|^2$, which is exponential in number of points and H_{max}. The size of the boundary matrix becomes a major constraint on the memory footprint to perform a full reduction. One approach is to represent the boundary matrix as a set of cascading matrices by dimension; *i.e.,* the first boundary matrix would examine the σ_0 simplices and respective cofaces, σ_1, the second σ_1 cross σ_2, and so on. This still results in a large representation of the boundary matrix, especially as H_{max} increases and higher-order boundary matrices are necessary. However, this method (with various optimizations) is a common implementation of the boundary matrix for persistent homology.

More recent optimizations have been implemented with an implicit representation of the boundary matrix in order to construct and reduce the boundary matrix for Vietoris–Rips filtrations [63]. The approach takes advantage of several key features of the Vietoris–Rips filtration to only compute necessary cofaces for boundary matrix reduction (Section 4.3). Implicit representation of the boundary matrix can reduce the memory footprint even further, consequently improving the run-time performance of the algorithm.

Construction of the boundary matrix is one of the primary bottlenecks for persistent homology. Implicit representation of the boundary matrix provides a reduced memory footprint and faster run-times with several optimizations; however, it may be less suitable for maintaining a complex in a streaming or evolving approach [21]. Attempts to provide fast construction of the complex and boundary matrix continue to yield further enhancements to the persistent homology pipeline for both exact and approximate methods to extract persistence intervals.

4.3 Boundary Matrix Reduction

Once the boundary matrix is constructed it must be reduced to identify algebraic loops, voids, and higher order topological features. The standard concept for homology consists of attaching to a topological space a sequence of homology groups to obtain the global topological features. These global topological features represent topological structures of the continuous shape, including holes, curves, and so on. The background theory consists of topological spaces, homology groups and an evolution scheme; these topics are discussed more thoroughly in [1], [5].

A generalized approach to extract persistence intervals, the *standard approach*, utilizes a tracking array with a slot for each simplex in the filtration [68]. This approach relies on the observation that if pivots in the boundary matrix are eliminated in decreasing order the entire description can be identified from row echelon form without the need to reduce to normal form. Algebraic chains identified from the reduction can then be analyzed for their birth and death times by examining the minimal face of the topological feature and collapsing filtration value. These results describe a general approach to computing persistence intervals up to H_d.

The boundary matrix reduction step is a computationally expensive process due to the number of the generated simplices in the complex. The standard approach for reduction of the boundary matrix quickly becomes the dominant performance inhibitor when computing persistent homology on larger data sets. Many optimizations for reduction of the boundary matrix have been proposed; however, several provide a current best approach to boundary matrix reduction. These include twist and clear, cohomology, implicit matrix reduction, and emergent pairs. Tools using these optimizations such as Ripser [73], GUDHI [70] and Ripserer [90] perform comparably well.

Co-reduction is an algorithm that can compute the homology of large cubical and simplicial complexes [62]. It is essential for low-dimensional topological sets embedded in high dimensions. The algorithm is based on the theory of one space homology, enabling the dual process of co-reduction. The experimental results in [62] demonstrate that the algorithm performs much faster than other homology algorithms for low-dimensional sets embedded in high dimensions.

de Silva *et al* [59] suggested using the co-homology for persistence computation due to the close relationship between absolute and relative persistent co-homology. de Silva establishes that the homology and cohomology groups of a filtered cell complex contain equivalent information, leading to a reduced *dual* algorithm for computing persistent cohomology. While cohomology itself provides a notable performance increase over traditional persistent homology, it implicitly employs the clearing algorithm to achieve additional gain demonstrated through the study.

The standard boundary matrix reduction algorithm fails to exploit the special structure of a boundary matrix in which the boundaries are always cycles. This phenomenon introduces a large number of unnecessary matrix operations in the reduction. The twist optimization [61] avoids the computation of cycles in decreasing dimension by "killing" or "zeroing" the higher-order pivot columns (set them to zero) without reduction. This optimization can improve the performance by reducing number of columns processed in the boundary matrix. The algorithm processes the complex in decreasing dimension, noting that columns can be killed (set to 0) when the corresponding $d+1$ boundary matrix row is fully reduced in the boundary matrix.

While the twist and clear observation is identified as the dimension decreases, increasing dimensions produces a similar optimization; the null space of $d+1$ are formed by the pivots columns of d. This indicates when a pivot column is found in increasing dimension of boundary matrix d, the corresponding row in the boundary matrix $d+1$ can be removed with no effect on the persistence intervals. This increasing-dimension approach is generally referred to as the *clearing* algorithm. Interestingly, the clearing algorithm is included in de Silva's introduction of co-homology; it accounts for a portion of the improvement in de Silva's technique.

The use of clearing and computing the cohomology with an efficient data structure, such as the simplex tree, provide a sufficient approach for computing lower-dimensional persistent homology on several thousand points. The approach is still generally limited to several thousand points when $H_{max} < 2$, and several hundred points in higher dimensions. Bottlenecks to the approach in this manner are the construction of the boundary matrix from the complex and reduction of the boundary matrix when the number of simplices becomes increasingly large. This has led to an alternative representation of the complex to refactor the construction of the complex into a more space-efficient approach.

Implicit representation of the boundary matrix is a recent optimization that reduces the need for constructing the entire boundary matrix by representing coface incidences through operations on the complex. Bauer introduces this implicit representation in Ripser and describes the required preliminaries and ordering to perform the approach [63]. Apparent and emergent pairs are also utilized in Ripser and described thoroughly in [63].

Combinations of these optimizations have led to changes in both the structure of the simplicial complex and representation of the boundary matrix. Optimal VR approaches now utilize implicit matrix representation to provide reduced memory consumption and fast reduction of the coboundary matrix. Persistent cohomology has become, in some ways, synonymous with persistent homology in implementations due to the speedups obtained. Continued performance increases in the boundary matrix reduction and portions of the persistent homology pipeline create a need for users to understand the key computational components.

4.4 Approximate Methods for PH

Methods to approximate the persistent homology of a point cloud can be separated into two primary categories: approximation of the original point and approximation of the complex. Approximation of the point cloud utilizes methods such as sampling and dimensionality reduction to provide a smaller input point cloud to PH that approximates the original data. Approximation of the complex uses sparsification to limit the number of complex elements that need to be analyzed. Both of these approaches have demonstrated significant advancements in the computation of PH on larger and higher-dimensional data sets.

Approximation of the input point cloud has been studied by Chazal *et al* [29] using sampled data and computing persistent homology on multiple independent samplings. Work by Moitra *et al* [75] and Malott *et al* [76], [77] have expanded on Chazal's findings, examining directed subsampling through partitioning algorithms to approximate salient topological features of the point cloud. In [75], the presented cluster-based data reduction is related to subsampling, but the approach only needs to take one approximation from the original data, reducing the computational complexity. Figure 8 demonstrates the cluster-based data reduction technique using k-means++ to reduce the size of the point cloud prior to persistent homology. The results indicate large topological features, such as the H_2 void inside of the triangulated mesh model, are preserved with bounded error.

Several additional studies of the cluster-based reduction have been examined with different approximations of the input point cloud [48], [76], [77]. The reduction of the point cloud can preserve salient topological features in cases of

extreme reduction leading to reduced complexity of the approach, a significant development in approximating high-dimensional topological features.

There have also been various attempts to reduce the dimensionality of data passed into persistent homology through projections [78], [79], [80]. Ramamurthy *et al* perform detailed experiments on the effect of random projection on persistence intervals and Betti numbers [80]. The remaining two studies present theoretical limits of dimensionality reduction for persistent homology [78], [79]. Another recent study on the effects of projection on PH attempts to identify and recreate high-dimensional features in their original space from a lower dimensional projection of the data [**?**]. While projection may play an important role in understanding and recognizing features in high dimensional spaces, this is one area that persistent homology excels in naturally. Due to the point cloud being reduced to a distance matrix, any dimensionality reduction only impacts the inner point distances. In many cases this distance can be bounded; if the perturbation of the distance matrix is small enough the resulting persistence intervals may be largely unaffected.

Approximation of the complex representation has been introduced to decrease the space-time computational complexity [57]. For example, reduction through simplicial collapse of the complex has demonstrated capabilities for processing PH on larger point clouds [81], [82], [83], [84]. Dey *et al* [85] conducted the approximation of persistent homology via simplicial batch collapse, providing a technique for larger topological feature identification. The approach has been implemented in the SimBa Library [85] which is an efficient and fast algorithm for approximating the persistent homology of Rips filtration. Several additional techniques, such as morse matching [86], [87] and sparsification [88], [89] have been proposed to further decrease the size of a complex.

Approximation methods may introduce some error to the resultant persistence intervals. Characterizing approaches for approximating persistent homology requires careful analysis of both data and results. However, in cases where data sizes are too large for exact applications of persistent homology, it may be necessary to approximate the point cloud or complex to compute within run-time or memory requirements.

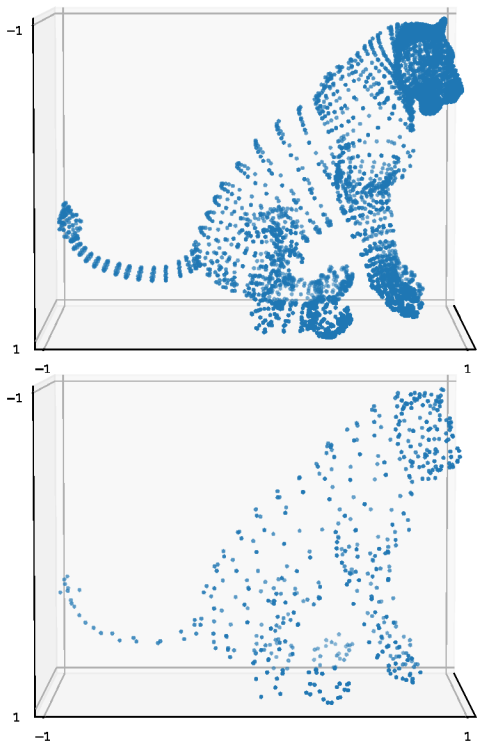


Fig. 8. Approximation of a triangulated mesh point cloud. Original Lion model (top) and 90% Reduction of Lion model with k-means++ centroid replacement (bottom). Large structures such as the hollow void in the center of the lion are preserved and identified with persistent homology after significant reduction to the point cloud size.

5 Libraries for Persistent Homology

There are several libraries that are immediately available for computing persistent homology (see Table 2). Some persistent homology libraries target specific purposes, such as computing the Vietoris–Rips filtration of the input data. Others provide choices of complex storage and filtrations that can be utilized for varying types of data. This section summarizes some of the existing libraries and respective uses for each. In addition, the optimization strategies and steps covered in Section 4 are detailed as they pertain to each of these libraries.

When selecting an appropriate library for high-performance persistent homology there are several determining factors: the characteristics of the input data, the implementation architecture (serial, parallel, distributed), and integration with additional tools for data mining or machine learning. This section provides information for integrating persistent homology into a larger application using existing libraries. There are persistent homology libraries available in most modern languages with more being developed; implementation decisions need to be based on the aforementioned characteristics of analysis.

The type of input data can immediately limit the application of existing libraries depending on the use case. Otter *et al* introduce several guidelines for best-suited implementations based on the characteristic data set [5]. For example points in Euclidean space or a distance matrix can be best computed with the Vietoris–Rips, Čech, or Witness complexes, leading to the selection of Ripser or GUDHI for high performance. Image data, which is cubical in nature, benefits from the use of a cubical complex to represent the relationships which is directly available from GUDHI or DIPHA. Other types of data, such as networks or gradient maps may require an alternate selection of persistent homology library or preprocessing to form a suitable metric space. Understanding the data and underlying structure is key to computing and interpreting the resultant persistence intervals.

The implementation architecture can also have implications on the most-suitable library to use. For example, if additional computing hardware is available, such as multiple processors, a distributed heterogeneous or homogeneous system, or GPUs, different approaches may show significant performance gains over a library with non-parallel implementations. In a distributed environment the DIPHA library provides chunking and distribution of the boundary

TABLE 2
A brief overview of current persistent homology libraries with respective optimizations.

Library	Language	Filtrations	Optimizations	Complex Representations	Related Libraries
Ripser [63]	C++, Python	VR, Cubical	Morse, Implicit, Twist, Clear, Dual	Implicit	Ripser.py
Ripserer [90]	Julia	VR, Cubical, Alpha	Morse, Implicit, Twist, Clear, Dual, Involuted, Critical Simplex	Implicit	
Ripser++ [93]	C++	VR, Cubical	GPU, Clearing, Compression		Ripser
GUDHI [70]	C++, R Python	VR, Witness, Cubical, Alpha, Čech, Cover, Tangential	Dual, Sparsification Subsampling, Implicit Edge Collapse	SimplexTree, Toplex Map, Skeleton Blocker CAM	
JavaPlex [**?**]	Java	VR, Landmark, Witness	Zigzag, Dual		
Eirene [74]	Julia	VR	Morse, Sheaf Dual	Boundary Matrix	
Dionysus [**?**]	C++, R Python	VR, Alpha, Čech	Dual, Zigzag	Boundary Matrix	
Perseus [13]	C++	VR, Cubical	Morse	Boundary Matrix	
PHAT [91]	C++, R	VR, Cubical	Twist, Chunk, Spectral Sequence Dual	Boundary Matrix	
DIPHA [92]	C++	VR, Cubical	Twist, Clearing, Dual, Distributed	Boundary Matrix	PHAT
LHF [76]	C++	VR, Alpha	Partitioned PH, Morse, Implicit, Twist, Clear, Dual, *Upscaling*, Involuted	SimplexTree, Boundary Matrix, Implicit	
giotto [94]	Python	VR, Flag, Čech Cubical, Alpha	Edge Collapse, Image Filtrations, Ripser/GUDHI Integration		Ripser, GUDHI

matrix to perform a distributed reduction step, leading to less restriction on the size of the input data based on memory of a single node. For distributed or parallel environments the LHF library provides partitioning methods to attack the exponential memory growth of the complex [76]. Alternatively, GPU approaches such as Ripser++ can accelerate the boundary matrix reduction step on a single node by employing an accelerated GPU algorithm [95]. The sequential approach these optimizations may be necessary in cases where the architecture of the system can employ these techniques and the size of input data is cumbersome.

Finally, the integration of additional tools that wrap the persistent homology library are important in selecting a tool to use. Several of the applications have python bindings into the libraries, which enable fast startup and testing for implementation. These approaches are valuable for system prototyping and research, but may not be a standard choice for machine learning pipelines that require high performance and throughput. In these cases the library should work well with the machine learning pipeline; a decision on how well the library integrates may deem suitability.

Table 2 presents several libraries that can be used to compute persistent homology. While there are many more libraries that exist and continue to be developed, this collection focuses on state-of-the-art libraries that can be easily integrated into a larger machine learning pipeline for high-performance data science. Several features of each library are noted including the complex filtrations available through the library, various optimizations employed, and the internal representation of the complex and boundary matrix for reduction. The latter is key to computing persistent homology for larger point clouds and higher dimensions as the size of the boundary matrix is a primary bottleneck for modern computers.

There is no single approach that fits every application; the libraries discussed provide coverage of several common applications for machine learning and data analysis pipelines. Understanding of the different components of persistent homology as described in this survey attempt to guide readers in their selection of design criteria and corresponding libraries for various applications.

5.1 Sequential Libraries

This section highlights some of the sequential libraries for computing persistent homology. The libraries offer different complex representations and tuning parameters for experimentation. These parameters are described in detail in each respective library's documentation pages.

One of the more recent libraries for Vietoris–Rips filtrations is Ripser [63]. Ripser uses an implicit boundary matrix representation to reduce the required memory for the reduction step, subsequently reducing the run-time of the approach. Ripser has been integrated into python bindings through the Ripser.py PyPi package. There have been various extensions to the Ripser approach (*e.g.,* Ripser++ and Ripserer) each wrapping the optimizations set forth in Ripser in new and innovative ways. Ripser++ implements parallel boundary matrix reduction for GPU acceleration [95].

GUDHI [70] is a more generalized approach to computing persistent homology. GUDHI provides interfaces for several different complexes including Alpha, Čech, Vietoris–Rips, Witness, Cubical, and more. The library exhibits similar performance to Ripser when computing Vietoris–Rips filtrations of a point cloud. However, GUDHI has additional filtrations and complex representations that may be beneficial in practice, such as sparsification of the complex.

Eirene [74] is an implementation of the Vietoris–Rips filtration in Julia that uses sheaf cohomology to identify persistence intervals quickly. Eirene is not as performant as Ripser and GUDHI, but computes the representative generators of the persistence intervals which identify constituent points of a feature. Ripserer, another julia PH library, includes a method of identifying representative cycles with *involuted* homology [90], [96]. The use of the representative cycles can enable tracking of topological features in the original point cloud. These and other notable features are identified in Table 2.

5.2 Parallel/Distributed Libraries

Parallel and distributed libraries combat the memory and run-time limitations of persistent homology. This section introduces several parallel and distributed persistent homology libraries. As mentioned previously, a variant of Ripser called Ripser++ enables GPU acceleration of the algorithm through parallel reduction of the boundary matrix in steps [95]. Giotto, which wraps a newer version of Ripser using a lock-free algorithm [**?**], combines several of the aforementioned sequential tools into a single python library for interoperability [94].

DIPHA [92] is a distributed library that utilizes OpenMP to enable reduction of the boundary matrix on a distributed compute cluster. DIPHA employs several similar optimizations to PHAT [91] and can process much larger complexes than a sequential approach. In cases where the size of the point cloud or dimension of homology to compute to is too large for a single computer, distribution of the work to several worker nodes may provide suitable evaluation of the persistence intervals. HYPHA [93] extends DIPHA further by implementing GPU accelerated scanning and compression of the boundary matrix that can further increase performance.

DIPHA provides an exact evaluation of the persistence intervals; the size of the boundary matrix grows exponentially with the number of points. The construction and storage of the boundary matrix is often the bottleneck for persistent homology [63]. For this reason, approximate approaches, such as the clustering approach described in Section 4.4 and depicted in Figure 8 can have significant impact on the underlying performance of the approach. An extension to this clustering technique in a distributed environment is to partition the point cloud and distribute the partitions to approximate the topological features of the space [76]. This approach, called *Partitioned Persistent Homology*, is implemented into the LHF library for parallel and distributed approaches [48]. LHF provides several different optimizations in a pipelined architecture for study of the affects of partitioning and *upscaling* for reconstruction of feature boundaries. LHF also implements the sequential Ripser algorithm, alongside several different complex types and experimental tools for PH evaluation.

6 Experimental PH Performance

In this section the capabilities of persistent homology are evaluated with respect to complex sizes, implementation of optimizations, and hardware performance. Persistent homology is generally limited by memory; the size of the complex and boundary matrix reduction steps quickly grow beyond memory capacity, even on high-end systems and servers. The experimental results provide insight into how, and when, the persistent homology algorithm is applicable in high-performance systems.

First, the complex performance is analyzed with respect to the memory capacity of the system in Section 6.1. Additional SWAP space is utilized to measure the effectiveness of disk space for the algorithm and run-time implications. Synthetic data is generated to measure the memory and run-time performance. These methods are presented to demonstrate how the complex quickly exceeds memory limitations, especially in higher dimensions of persistent homology.

Once the complex is constructed the boundary matrix is reduced to extract persistence intervals. Section 6.2 examines several notable techniques to improve the algorithm alongside measurements of their respective impact. These techniques are evaluated against selected TDA data sets of varying dimensions. The use of multiple optimizations to the boundary matrix reduction step can significantly accelerate the overall performance.

Section 6.3 demonstrates the impact of hardware decisions for computing persistent homology. Specifically, the memory size, bandwidth, channels, and controllers are examined in the context of computational PH and provide dedicated system suggestions for high-performance computation. The analysis yields a road-map for selecting high-performance architectures for computational persistent homology.

6.1 Complex Performance

The complex constructed for persistent homology grows exponentially in size by the number of points and the maximum dimension of homology. Persistent homology is restricted in practice to relatively small data sets and lower dimensions of homology, often bounded by available memory of the system. Overflow into disk memory, such as expanding the system's SWAP space, can provide additional space for the storage of the complex at the cost of run-time performance of the algorithm. However, even with large amounts of memory availability the exponential growth of the complex can easily outpace expansions of memory.

Synthetic data was generated for capturing the memory and run-time limitations of Ripser at different homology dimensions. Each data set utilizes points sampled from the surface of a d-sphere, where d is equal to the maximum homology dimension. This creates a topological feature at H_{max} to be identified by the algorithm. The number of points sampled from the d-sphere were varied to analyze the impact of point cloud size and homology dimension on the memory and run-time performance.

Figure 9 depicts the dimensional limitations of Ripser with 32GB of RAM alongside 32GB of swap, doubling the effective memory space to 64GB. The memory limitation of the system still significantly inhibits the algorithm for identifying higher dimensional features. Figure 10 plots the corresponding execution time for each test. Once SWAP space begins to be utilized the run-time performance suffers due to slower disk access.

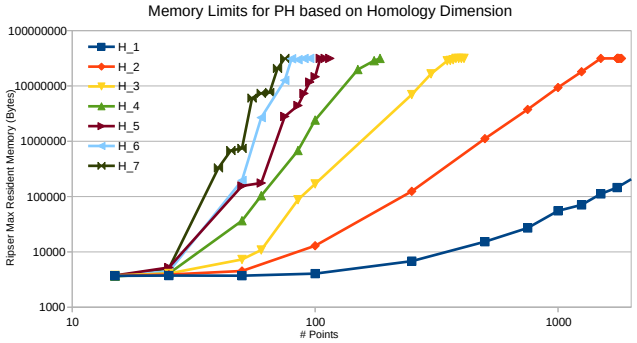


Fig. 9. Memory limits for Ripser based on number of points in point cloud and the maximum dimension of homology to compute. Higher dimensions of homology ($H_{max} > 2$) require significant resources for small point clouds.

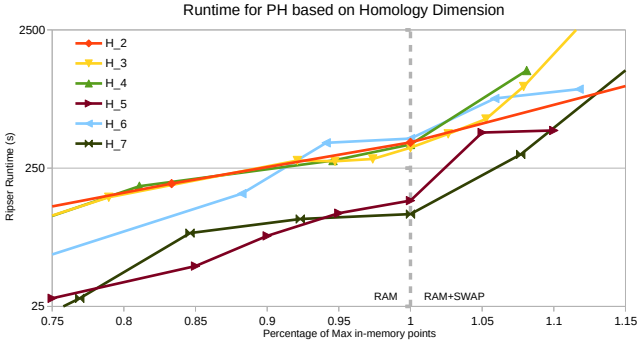


Fig. 10. Ripser run-time as the percentage of the maximum in-memory points is exceeded, leading to SWAP usage. Once swap begins to be used (> 1.0), the run-times increase and can vary significantly from thrashing.

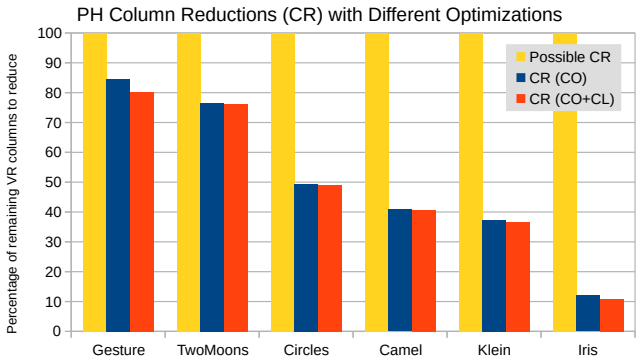


Fig. 11. This graph shows the percentage impact of different optimizations against the baseline of the standard homology computation (Possible CR). The dual algorithm (CR (CO)), which computes the cohomology, results in a significant decrease in column reductions. The clearing algorithm paired with the dual algorithm (CR (CO + CL) provides a slight additional decrease in the remaining columns for these data sets.

Approaches to reduce the size of the complex, such as sparsification, collapses, and point cloud approximations attempt to preserve the topological features of the point cloud while reducing the total number of simplices to analyze. As indicated in Section 3.1, different filtrations of the point cloud will change the size of the complex as well. In the discussion of the reduction of the complex, the remainder of this section utilizes the *Vietoris–Rips (VR)* complex as the baseline for reduction. This represents the complete graph; the set of all possible simplices that can be formed from the point cloud.

6.2 Boundary Matrix Reduction Performance

Optimizations to the boundary matrix reduction target the processing time and frequency of columns analyzed. There are four primary techniques for Vietoris–Rips complexes: cohomology, clearing, morse matching, and emergent and apparent pairs. Several of these techniques are analyzed independently and in conjunction to provide relative impact of the optimizations in this section. This approach characterizes the impact on the boundary matrix reduction algorithm by counting the number of reduced columns from independent and combinations of the standard approaches.

As Figure 11 shows, the dual algorithm [59], which computes the cohomology of the point cloud, achieves massive performance speedup due to a large reduction in the number of column operations on the boundary matrix. In addition, the twist and clear algorithm [61] reduces the number of columns in subsequent dimensions of the boundary matrix reduction by noting columns that are zeroed in the next dimension. This effect can remove a significant number of columns from needing to be reduced, leading to better algorithm performance in both space and run-time efficiency. Paired with clearing the dual algorithm can add an additional performance boost in the reduction of the boundary matrix due to less column reductions.

The magnitude of simplices generated during complex construction is large, and grows larger with higher dimensions of homology. This directly impacts the number of column reductions in the boundary matrix, although the optimizations described can help reduce the required columns. Table 3 presents an experimental analysis of column reductions to the boundary matrix performed over several different data sets. Figure 11 compares the total number of boundary column rows in the homology computation, the cohomology number of rows, and the rows after clearing.

The data sets compared are samplings in different dimensions to demonstrate how the clearing algorithm significantly improves in higher dimensions at removing a large percentage of candidate columns in the boundary matrix. Paired with the cohomology algorithm the clearing optimization can limit the number of column reductions, significantly impacting the speed of the approach. The magnitude of column reductions, even after cohomology and clearing, can have a large effect on performance of applications if not managed with compact data structures.

In the most extreme case, the Iris data set, when computed up to $H_{max} = 5$ on 100 points, can generate over a billion total columns in the boundary matrix for reduction. Cohomology requires only a fraction of the columns, while cohomology with the clearing optimization can reduce the number of columns further. In addition to the clearing and cohomology algorithms, additional techniques such as morse matching and emergent and apparent pairs

TABLE 3
Sample data sets used to demonstrate the column reductions for persistent homology, depicted in Figure 11. The *CR (CO)* column represents the remaining columns with cohomology, and the *CR (CO + CL)* column represents the remaining columns with cohomology and clearing.

Data Set	# Points	Data Dimension	H_{max}	Possible CR	CR (CO)	CR (CO + CL)
Gesture	100	32	4	79,375,495	67,098,259	63,644,877
TwoMoons	2000	2	1	2,001,000	1,528,942	1,526,943
Circles	1200	10	2	288,001,000	142,147,056	141,616,051
Camel	1200	3	2	288,001,000	117,556,212	117,059,110
Klein	375	3	2	8,789,375	3,271,474	3,223,812
Iris	100	4	5	1,271,427,895	152,659,308	138,731,360

can provide reduction in the computational complexity as detailed in [63]. These optimizations are not column level; measurement of their impact requires detailed analysis of the individual techniques.

One recent optimization utilized in the Ripser algorithm is implicit representation of the boundary matrix. The approach takes advantage of the combinatoric structure of the Vietoris–Rips complex to inductively construct and compute persistent homology. Ripser works alongside the dual algorithm and clearing optimization to provide the fastest sequential approach to persistent homology of VR complexes. This is the primary reason Ripser is evaluated as a baseline throughout this experimental section.

6.3 Hardware Performance

Persistent homology is a memory-intensive algorithm. Fast access memory, such as RAM, provides necessary resources for computing persistent homology over higher dimensions and larger point clouds in reasonable time. As discussed in Section 6.1, the use of SWAP to expand the memory of the system can alleviate the memory strain. Additional hardware specifications, such as memory capacity, bandwidth, channels, and controllers, are equally important to the overall performance of the algorithm.

Memory capacity of the system is examined closely in Section 6.1, demonstrating the exponential growth of the complex and need for significant memory resources for persistent homology. While memory capacity limits the size of complexes that can be evaluated, other characteristics of the system architecture can hinder the run-time performance. Two machines with the same memory capacity can evaluate the same sizes for persistent homology but may have wildly varying run-times.

Memory bandwidth, channels, and controllers can contribute to performance differences between machines. Memory bandwidth is intuitive — newer, faster RAM provides accelerated read and write times during the algorithm. These read and writes happen frequently; thus the number of channels and controllers available can bottleneck the speed of the algorithm in larger memory spaces. This observation was originally identified while evaluating expanding memory space in the run-time performance of the algorithm. The remainder of this section characterizes how, and when, these memory attributes will affect the performance of persistent homology applications.

Figure 12 plots experimental data from several different computer architectures when computing persistent homology. The four classes of machines are plotted as separate

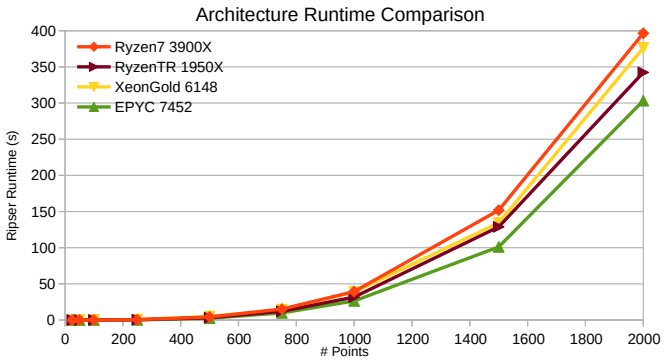


Fig. 12. Experimental performance over computer architectures listed in Table 4. Characteristics of the memory architecture play an important role in the run-time performance of persistent homology.

series in the chart. Each machine configuration is detailed in Table 4 for reference. All limits were taken when computing up to H_2 homology groups, generating d-spheres as described in Section 6.1.

One interesting difference between the evaluated systems is in the number of RAM channels and controllers. Due to the memory intensive construction of the complex and boundary matrix reduction the RAM channels and controllers quickly bound the run-time of the approach. The memory architecture organization is more detrimental in higher dimensions of PH; computing over thousands of points may take several hours when possible. Newer processors such as the AMD EPYC 7452 build provide better run-time performance due to an alleviated memory bottleneck over systems with the same memory capacity but with lower memory throughput.

While the persistent homology algorithm requires massive system resources to run, the hardware architecture can have significant implications on the performance. Utilizing systems with more memory channels and controllers, especially a 1-to-1 ratio, will yield considerable speedup when computing persistent homology. Processors such as the new AMD EPYC 7452 continue to provide alleviation of the memory bottleneck on the technique. Architecture of the system can play an important role in implementing persistent homology for experimental studies.

7 Conclusion

The study of persistent homology continues to uncover fascinating relationships within data unseen by traditional

TABLE 4
System architectures for evaluated Persistent Homology limitations. Variations in memory architectures contribute to performance differences between these systems as depicted in Figure 12.

CPU Model	CPU Frequency Base (GHz)	CPU Frequency Max (GHz)	RAM Capacity (GB)	RAM Frequency (MHz)	# RAM Channels	# RAM Controllers
AMD Ryzen7 3800X	3.9	4.5	64GB	2400	2	2
AMD RyzenTR 1950X	3.4	4.0	128GB	2666	4	4
Intel XeonGold 6148	2.4	3.7	192GB	2666	6	2
AMD EPYC 7452	2.35	3.35	256GB	3200	8	8

machine learning algorithms. While the memory and runtime complexities of the algorithm currently inhibit use for big data, optimizations and strategies to approximate the persistent homology are being explored.

This survey has detailed several applications of persistent homology alongside the technical considerations for designing high-performance applications of the approach. Utilization of persistent homology for data mining and machine learning can provide alternative representation of point clouds and identify relationships often missed by traditional methods of data analysis. Experimentation within the data science and engineering communities will continue to reveal opportunities for the technique in various fields.

Unfortunately there remains no single solution for computing persistent homology on any point cloud. Selection of complexes, storage types, and analysis of the data depends heavily on the domain being analyzed. These decisions need to be made when designing and implementing machine learning applications using persistent homology and should be appropriately studied prior to integration. Existing tools may provide suitable approaches for prototyping, experimental analysis, and further data studies.

ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under grant IIS–1909096. In addition, this research was supported in part through research cyberinfrastructure resources and services provided by the Advanced Research Computing (ARC) center at the University of Cincinnati, Cincinnati, OH, USA.

REFERENCES

[1] F. Chazal and B. Michel, "An introduction to topological data analysis: Fundamental and practical aspects for data scientists," *ArXiv e-prints*, Feb. 2021.
[2] R. Ghrist, "Barcodes: The persistent topology of data," *Bulletin of the American Mathematical Society*, vol. 45, no. 1, pp. 61–75, 2008.
[3] P. Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson, and G. Carlsson, "Extracting insights from the shape of complex data using topology," *Scientific Reports*, vol. 3, Feb. 2013.
[4] G. Singh, F. Memoli, and G. Carlsson, "Topological methods for the analysis of high dimensional data sets and 3D object recognition," in *Eurographics Symposium on Point-Based Graphics*, M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, Eds. The Eurographics Association, 2007.
[5] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington, "A roadmap for the computation of persistent homology," *EPJ Data Science*, vol. 6, no. 1, Aug. 2017.
[6] G. Petri, M. Scolamiero, I. Donato, and F. Vaccarino, "Topological strata of weighted complex networks," *PLOS ONE*, vol. 8, no. 6, pp. 1–8, Jun. 2013.
[7] D. Horak, S. Maletić, and M. Rajković, "Persistent homology of complex networks," *Journal of Statistical Mechanics: Theory and Experiment*, Mar. 2009.
[8] M. Hajij, B. Wang, C. E. Scheidegger, and P. Rosen, "Visual detection of structural changes in time-varying graphs using persistent homology," *IEEE Pacific Visualization Symp*, pp. 125–134, 2018.
[9] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer, "Persistent homology analysis of brain artery trees," *The Annals of Applied Statistics*, vol. 10, no. 1, pp. 198–218, Mar. 2016.
[10] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, ser. Applied Mathematical Sciences. Springer-Verlag New York, 2004, vol. 157.
[11] H. Wagner, C. Chen, and E. Vuçini, "Efficient computation of persistent homology for cubical data," in *Topological Methods in Data Analysis and Visualization II: Theory, Algorithms, and Applications*, R. Peikert, H. Hauser, H. Carr, and R. Fuchs, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 91–106.
[12] P. Bendich, H. Edelsbrunner, and M. Kerber, "Computing robustness and persistence for images," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1251–1260, Nov. 2010.
[13] K. Mischaikow and V. Nanda, "Morse theory for filtrations and efficient computation of persistent homology," *Discrete & Computational Geometry*, vol. 50, no. 2, pp. 330–353, 2013.
[14] V. Nanda, "Discrete morse theory for filtrations," Ph.D. dissertation, Department of Mathematics, Rutgers University, Oct. 2012. [Online]. Available: http://people.maths.ox.ac.uk/nanda/source/Thesis.pdf
[15] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian, "On the local behavior of spaces of natural images," *International Journal of Computer Vision*, vol. 76, no. 1, pp. 1–12, Jan. 2008.
[16] Z. Cang, L. Mu, K. Wu, K. Opron, K. Xia, and G.-W. Wei, "A topological approach for protein classification," *Molecular Based Mathematical Biology*, vol. 3, no. 1, Nov. 2015.
[17] T. K. Dey and S. Mandal, "Protein classification with improved topological data analysis," in *18th International Workshop on Algorithms in Bioinformatics*, ser. WABI 2018, Aug. 2019, pp. 6:1–6:13.
[18] K. Xia and G.-W. Wei, "Persistent homology analysis of protein structure, flexibility, and folding," *Int Journal for Numerical Methods in Biomedical Engineering*, vol. 30, no. 8, pp. 814–844, 2014.
[19] P. G. Camara, D. I. S. Rosenbloom, K. J. Emmett, A. J. Levine, and R. Rabadan, "Topological data analysis generates high-resolution, genome-wide maps of human recombination," *Cell systems*, vol. 3, no. 1, pp. 83–94, 2016.
[20] J. M. Chan, G. Carlsson, and R. Rabadan, "Topology of viral evolution," *Proceedings of the National Academy of Sciences*, vol. 110, no. 46, pp. 18 566–18 571, 2013.
[21] A. Moitra, N. O. Malott, and P. A. Wilsey, "Persistent homology on streaming data," in *2020 International Conference on Data Mining Workshops (ICDMW)*, ser. ICDMW '20, Nov. 2020, pp. 636–643.
[22] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological persistence and simplification," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000.
[23] G. Carlsson, A. Zomorodian, A. Collins, and L. J. Guibas, "Persistence barcodes for shapes," *International Journal of Shape Modeling*, vol. 11, no. 02, pp. 149–187, 2005.
[24] L. Li, W.-Y. Cheng, B. S. Glicksberg, O. Gottesman, R. Tamler, R. Chen, E. P. Bottinger, and J. T. Dudley, "Identification of type 2 diabetes subgroups through topological analysis of patient similarity," *Science translational medicine*, vol. 7, no. 311, Oct. 2015.
[25] M. Nicolau, A. J. Levine, and G. Carlsson, "Topology based data analysis identifies a subgroup of breast cancers with a unique mu-

tational profile and excellent survival," *Proceedings of the National Academy of Sciences*, vol. 08, no. 17, pp. 7265–7270, 2011.
[26] R. Ghrist and A. Muhammad, "Coverage and hole-detection in sensor networks via homology," in *Fourth International Symposium on Information Processing in Sensor Networks*, ser. IPSN 2005. IEEE, Apr. 2005, pp. 254–260.
[27] V. de Silva and R. Ghrist, "Homological sensor networks," *Notices of the American Math Soc*, vol. 54, no. 1, pp. 10–17, Jan. 2007.
[28] A. Adcock, D. Rubin, and G. Carlsson, "Classification of hepatic lesions using the matching metric," *Computer vision and image understanding*, vol. 121, pp. 36–42, Apr. 2014.
[29] F. Chazal, B. T. Fasy, F. Lecci, B. Michel, A. Rinaldo, and L. Wasserman, "Subsampling methods for persistent homology," in *International Conference on Machine Learning*, ser. ICML 2015, Lille, France, Jul. 2015.
[30] H. Lee, H. Kang, M. K. Chung, B.-N. Kim, and D. S. Lee, "Persistent brain network homology from the perspective of dendrogram," *IEEE Transactions on Medical Imaging*, vol. 31, no. 12, pp. 2267–2277, Dec. 2012.
[31] G. Carlsson, "Topological pattern recognition for point cloud data," *Acta Numerica*, vol. 23, pp. 289–368, 2014.
[32] P. Frosini and C. Landi, "Persistent betti numbers for a noise tolerant shape-based approach to image retrieval," *Pattern Recognition Letters*, vol. 34, no. 8, pp. 863–872, Jun. 2013.
[33] Y. Dabaghian, F. Mémoli, L. Frank, and G. Carlsson, "A topological paradigm for hippocampal spatial map formation using persistent homology," *PLoS Comput Biol*, vol. 8, no. 8, 2012.
[34] C. M. Pereira and R. F. de Mello, "Persistent homology for time series and spatial data clustering," *Expert Systems with Applications*, vol. 42, no. 15–16, pp. 6026–6038, Sep. 2015.
[35] X. Zhu, "Persistent homology: An introduction and a new text representation for natural language processing," in *IJCAI*, 2013, pp. 1953–1959.
[36] U. Fugacci, S. Scaramuccia, F. Iuricich, and L. D. Floriani, "Persistent homology: a step-by-step introduction for newcomers." in *Smart Tools and Apps for Graphics – Eurographics Italian Chapter Conference*, G. Pintore and F. Stanco, Eds. The Eurographics Association, 2016, pp. 1–10.
[37] C. S. Pun, K. Xia, and S. X. Lee, "Persistent-homology-based machine learning and its applications – a survey," Nov. 2018.
[38] G. Carlsson, "Topology and data," *Bulletin of the American Mathematical Society*, vol. 46, no. 3, pp. 255–308, Apr. 2009.
[39] H. Edelsbrunner and J. Harer, "Persistent homology — a survey," *Surveys on Discrete and Computational Geometry*, vol. 453, pp. 257–282, 2008.
[40] F. Hensel, M. Moor, and B. Rieck, "A survey of topological machine learning methods," *Frontiers in Artificial Intelligence*, vol. 4, p. 52, May 2021.
[41] E. H. Spanier, *Algebraic Topology*. Springer Science & Business Media, 1989.
[42] R. Ghrist, *Elementary Applied Topology*. Createspace, 2014.
[43] H. Adams, T. Emerson, M. Kirby, R. Neville, C. Peterson, P. Shipman, S. Chepushtanova, E. Hanson, F. Motta, and L. Ziegelmeier, "Persistence images: A stable vector representation of persistent homology," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 218–252, Jan. 2017.
[44] C. M. M. Pereira and R. F. de Mello, "Data clustering using topological features," in *2014 Brazilian Conference on Intelligent Systems*, Oct. 2014, pp. 360–365.
[45] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980*, ser. Lecture Notes in Mathematics, D. Rand and L.-S. Young, Eds., vol. 898. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 366–381.
[46] C. Bresten and J.-H. Jung, "Detection of gravitational waves using topological data analysis and convolutional neural network: An improved approach," *ArXiv e-prints*, Oct. 2019.
[47] A. Myers, E. Munch, and F. A. Khasawneh, "Persistent homology of complex networks for dynamic state detection," *Physical Review E*, vol. 100, no. 2, p. 022314, 2019.
[48] N. O. Malott, R. R. Verma, and P. A. Wilsey, "Parallel computation of partitioned persistent homology," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 344–354.
[49] V. Kovacev-Nikolic, P. Bubenik, D. Nikolić, and G. Heo, "Using persistent homology and dynamical distances to analyze protein binding," *Statistical applications in genetics and molecular biology*, vol. 15, no. 1, pp. 19–38, 2016.
[50] P. Bubenik, "Statistical topological data analysis using persistence landscapes," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 77–102, Jan. 2015.
[51] S. Benzekry, J. A. Tuszynski, E. A. Rietman, and G. L. Klement, "Design principles for cancer therapy guided by changes in complexity of protein-protein interaction networks," *Biology direct*, vol. 10, no. 1, pp. 1–14, 2015.
[52] N.-K. Le, P. Martins, L. Decreusefond, and A. Vergne, "Construction of the generalized czech complex," in *2015 IEEE 81st Vehicular Technology Conference*, ser. (VTC Spring). IEEE, 2015, pp. 1–5.
[53] J.-D. Boissonnat and C. Maria, "The simplex tree: An efficient data structure for general simplicial complexes," *Algorithmica*, vol. 70, no. 3, pp. 406–427, Nov. 2014.
[54] A. Zomorodian, "The tidy set: A minimal simplicial set for computing homology of clique complexes," in *Twenty-Sixth Annual Symposium on Computational Geometry*, 2010, pp. 257–266.
[55] E. W. Chambers, J. Erickson, and P. Worah, "Testing contractibility in planar rips complexes," in *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*, 2008, pp. 251–259.
[56] H. Edelsbrunner and J. Harer, *Computational Topology, An Introduction*. American Mathematical Society, 2010.
[57] V. de Silva and G. Carlsson, "Topological estimation using witness complexes," in *Eurographics Symposium on Point-Based Graphics*, ser. SPBG '04, M. Gross, H. Pfister, M. Alexa, and S. Rusinkiewicz, Eds. The Eurographics Association, 2004.
[58] U. Bruzzo, "Introduction to algebraic topology and algebraic geometry," 2002.
[59] V. de Silva, D. Morozov, and M. Vejdemo-Johansson, "Dualities in persistent (co)homology," *Inverse Problems*, vol. 27, no. 12, 2011.
[60] T. K. Dey, F. Fan, and Y. Wang, "Computing topological persistence for simplicial maps," in *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, ser. SOCG'14. New York, NY, USA: ACM, Jun. 2014, pp. 345–354.
[61] C. Chen and M. Kerber, "Persistent homology computation with a twist," in *Proceedings 27th European Workshop on Computational Geometry (EuroCG'11)*, 2011, pp. 197–200.
[62] M. Mrozek and B. Batko, "Coreduction homology algorithm," *Discrete & Computational Geometry*, vol. 41, no. 1, pp. 96–118, Jan. 2009.
[63] U. Bauer, "Ripser: efficient computation of vietoris-rips persistence barcodes," 2019.
[64] F. Chazal, B. Fasy, F. Lecci, B. Michel, A. Rinaldo, A. Rinaldo, and L. Wasserman, "Robust topological inference: Distance to a measure and kernel distance," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5845–5884, 2017.
[65] J. M. Phillips, B. Wang, and Y. Zheng, "Geometric inference on kernel density estimates," *arXiv preprint arXiv:1307.7760*, 2013.
[66] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, "Stability of persistence diagrams," *Discrete & computational geometry*, vol. 37, no. 1, pp. 103–120, 2007.
[67] J. Reininghaus, S. Huber, U. Bauer, and R. Kwitt, "A stable multi-scale kernel for topological machine learning," in *2015 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR, Jun. 2015, pp. 4741–4748.
[68] A. Zomorodian and G. Carlsson, "Computing persistent homology," *Discrete Comput Geom*, vol. 33, no. 2, pp. 249–274, Feb. 2005.
[69] M. Joswig and M. E. Pfetsch, "Computing optimal morse matchings," *SIAM Journal on Discrete Mathematics*, vol. 20, no. 1, pp. 11–25, 2006.
[70] C. Maria, J.-D. Boissonnat, M. Glisse, and M. Yvinec, "The gudhi library: Simplicial complexes and persistent homology," INRA, Tech. Rep. RR-8548, 2014. [Online]. Available: https://hal.inria.fr/hal-01005601v2
[71] A. Zomorodian, "Fast construction of the vietoris–rips complex," *Computer and Graphics*, pp. 263–271, 2010.
[72] J.-D. Boissonnat, T. K. Dey, and C. Maria, "The compressed annotation matrix: an efficient data structure for computing persistent cohomology," *CoRR*, vol. abs/1304.6813, 2013. [Online]. Available: http://arxiv.org/abs/1304.6813
[73] U. Bauer. (2018) Ripser. The Technical University of Munich. [Online]. Available: http://www.cs.umd.edu/mount/ANN/
[74] G. Henselman. (2019) Eirene: julia library for homological persistence. [Online]. Available: https://github.com/Eetion/Eirene.jl
[75] A. Moitra, N. Malott, and P. A. Wilsey, "Cluster-based data reduction for persistent homology," in *2018 IEEE International Conference on Big Data*, ser. Big Data 2018, Dec. 2018, pp. 327–334.

[76] N. O. Malott and P. A. Wilsey, "Fast computation of persistent homology with data reduction and data partitioning," in *2019 IEEE International Conference on Big Data*, ser. Big Data 2019, Dec. 2019, pp. 880–889.
[77] N. O. Malott, A. Sens, and P. A. Wilsey, "Topology preserving data reduction for computing persistent homology," in *International Workshop on Big Data Reduction*, 2020.
[78] S. Arya, J.-D. Boissonnat, K. Dutta, and M. Lotz, "Dimensionality reduction for k-distance applied to persistent homology," in *36th International Symposium on Computational Geometry*, ser. SoCG 2020, Jun. 2020, pp. 10:1–10:15.
[79] D. R. Sheehy, "The persistent homology of distance functions under random projection," in *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, ser. SOCG'14. New York, NY, USA: ACM, 2014, pp. 328–334.
[80] K. N. Ramamurthy, K. R. Varshney, and J. J. Thiagarajan, "Computing persistent homology under random projection," in *IEEE Workshop on Statistical Signal Processing*, Jun. 2014, pp. 105–108.
[81] M. K. Chari, "On discrete morse functions and combinatorial decompositions," *Discrete Mathematics*, vol. 217, no. 1-3, pp. 101–113, Apr. 2000.
[82] D. N. Kozlov, "Discrete morse theory for free chain complexes," *Comptes Rendus Mathematique*, vol. 340, no. 12, pp. 867–872, Jun. 2005.
[83] R. Malgouyres and A. R. Francés, "Determining whether a simplicial 3-complex collapses to a 1-complex is np-complete," in *International Conference on Discrete Geometry for Computer Imagery*, ser. Lecture Notes in Computer Science, D. Coeurjolly, I. Sivignon, L. Tougne, and F. Dupont, Eds., vol. 4992. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 177–188.
[84] M. Tancer, "Recognition of collapsible complexes is np-complete," *Discrete & Computational Geometry*, vol. 55, no. 1, pp. 21–38, Jan. 2016.
[85] T. K. Dey, D. Shi, and Y. Wang, "Simba: An efficient tool for approximating rips-filtration persistence via simplicial batch-collapse," *24th Annual European Symposium on Algorithms (ESA 2016)*, 2016.
[86] R. Forman, "Morse theory for cell complexes," *Advances in Mathematics*, vol. 134, no. 1, pp. 90–145, 1998.
[87] H. King, K. Knudson, and N. M. Kosta, "Birth and death in discrete morse theory," *Journal of Symbolic Computation*, vol. 78, no. C, pp. 41–60, Jan. 2017.
[88] M. Kerber and R. Sharathkumar, "Approximate čech complex in low and high dimensions," in *International Symposium on Algorithms and Computation*, ser. Lecture Notes in Computer Science, L. Cai, S.-W. Cheng, and T.-W. Lam, Eds., vol. 8283. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 666–676.
[89] D. R. Sheehy, "Linear-size approximations to the vietoris–rips filtration," *Discrete & Computational Geometry*, vol. 49, no. 4, pp. 778–796, Jun. 2013.
[90] M. Čufar, "Ripserer.jl: flexible and efficient persistent homology computation in julia," *Journal of Open Source Software*, vol. 5, no. 54, 2020.
[91] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner, "Phat — persistent homology algorithms toolbox," *Journal of Symbolic Computation*, vol. 78, pp. 76–90, January-February 2017.
[92] U. Bauer, M. Kerber, and J. Reininghaus, "Distributed computation of persistent homology," in *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2014, pp. 31–38.
[93] S. Zhang, M. Xiao, C. Guo, L. Geng, H. Want, and X. Zhang, "HYPHA: A framework based on separation of parallelisms to accelerage persistent homology matrix reduction," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19. New York, NY, USA: ACM, Jun. 2019, pp. 69–81.
[94] G. Tauzin, U. Lupo, L. T. andJulian Burella Pérez, M. Caorsi, A. M. Medina-Mardones, A. Dassatti, and K. Hess, "giotto-tda:: A topological data analysis toolkit for machine learning and data exploration." *Journal of Machine Learning Research*, vol. 22, no. 39, pp. 1–6, 2021.
[95] S. Zhang, M. Xiao, and H. Wang, "Gpu-accelerated computation of vietoris-rips persistence barcodes," *arXiv preprint arXiv:2003.07989*, 2020.
[96] M. Čufar and Žiga Virk, "Fast computation of persistent homology representatives with involuted persistent homology," 2021.

Nicholas O. Malott is a Graduate Assistant with the University of Cincinnati EECS department and completing his PhD in Computer Engineering. He received a BS and MS degree in Computer Engineering from University of Cincinnati in 2016 and 2020 respectively. His research includes Approximate Algorithms, Topological Data Analysis, and Distributed System Architecture and Design.

Shangye Chen is a Graduate Assistant with the University of Cincinnati EECS department and pursuing her PhD degree in Computer Science. She received the BS in Computer Science from Northwest University, Xi'an, China in 2016 and the MS degree in Computer Science from Miami University in 2019. Her research focuses on Topological Data Analysis and Streaming Data.

Philip A. Wilsey (M'82, SM'98) received the BS in Mathematics from Illinois State University in 1981 and the MS and PhD degrees in Computer Science from the University of Louisiana at Lafayette in 1985 and 1987. Currently he is a Professor in the EECS department at the University of Cincinnati. He studies High Performance Computing with applications to Topological Data Analysis and Parallel Simulation.