Distributed Computation of Persistent Homology from Partitioned Big Data

Nicholas O. Malott, Rishi R. Verma, Rohit P. Singh, and Philip A. Wilsey
Dept. of EECS, University of Cincinnati, Cincinnati, OH 45221, USA
Email: malottno@mail.uc.edu, verma.rishiraj@gmail.com, singh2ro@mail.uc.edu, philip.wilsey@uc.edu

***Abstract*—*Topological Data Analysis* is a machine learning method that summarizes the topological features of a space. *Persistent Homology* (PH) can identify these topological features as they persist within a point cloud; persisting in respect to the connectedness of the point cloud at increasing distances. The utility of PH is apparent in several fields including bioinformatics, network security, and object classification. However, the memory complexity of PH limits the application to relatively small point clouds for low-dimensional topological feature identification. For this reason, numerous approaches to optimize and approximate the PH have been introduced for providing results over large point clouds. One solution, *Partitioned Persistent Homology (PPH)*, has shown favorable approximation on a single node with significant performance improvement. However, the single-node approach is limited by the available system memory, leading to the need for a distributed approach for additional (especially memory) resources. This paper studies a distributed version of PPH for use with large point clouds over a high-performance compute cluster. Experimental results of the distributed algorithm against previous studies is presented along with scalability of the distributed library.**

***Index Terms*—topological data analysis; persistent homology; data partitioning; distributed data mining; distributed computing**

I. Introduction

Topological Data Analysis (TDA) provides methods to analyze and classify data based on the shape of manifolds in a manner resilient to noise and deformation [1]–[3]. One of the primary tools of TDA is *Persistent Homology* (PH). PH provides a scalar view of the topological features of a point cloud [1], [4]–[7]. The approach is to identify the change in topological features — components, loops, holes, and voids of the space — as the connectedness of the point cloud increases. PH produces *persistence intervals* that summarize the topological features for classification and machine learning.

While the technique is straightforward, the worst-case time and space complexity of computing PH is $O(n_K^{2.376})$ [7] and $O(n_K^2)$ [8] respectively, where n_k is the total number of simplices in the complex. For the Vietoris-Rips complex, the complex type most commonly used in computing PH, the worst case number of simplices is $n!/(d+1)!(n-d-1)!$, where n is the number of points and d is the dimension of homology to compute up to [9]. As a result (and unless otherwise constrained), the computation of PH is limited to small data-sets of only a few thousand points in $\mathbb{R}^3$.

Several advances are being made to compute PH on larger point clouds. These include reduction of the simplicial complex [10]–[15] and optimization of the reduction algorithm [16]–[19]. Unfortunately these approaches have not yet provided the boost necessary to compute PH over big data in a manner that preserves both large and small topological features of the point cloud. There are several specific data analysis problems for which both large and small topological features have been shown to be important [20]–[22], making the preservation of both desirable.

One alternative approach for approximation of PH on very large point clouds involves: (i) partitioning to compute small topological features, and (ii) data reduction to locate the large features [23]. The method requires the construction of *ghost cells* [24] such that the PH computations can identify small topological features that lie on the boundary between partitions. Large features can be located in the reduced point cloud and refined using *upscaling*. This technique is called *Partitioned Persistent Homology (PPH)*.

This paper extends the technical approach outlined by Malott and Wilsey for computing PPH on big data and explores the implementation on a distributed cluster. The key advantage of this work over that of [23] is the exploitation of the multiplicity of memory resources to permit the efficient computation of PPH on big data. That is, in the worst case, the single compute node may require sequential execution of the PH computation for each partition (when a partition has space requirements equal to the RAM capacity of the compute node).

Distributed PPH is developed into the *Lightweight Homology Framework* (LHF) [25] open source software library. The solution uses an MPI-based communication structure to distribute the computational workloads and merge results. In order to demonstrate the effectiveness of the distributed approach, several traditional data-set examples [10], [12], [21], [23] are examined and their performance profiled. The scalability of the approach is evaluated with respect to the MPI implementation in LHF. Results show significant improvement in the memory footprint of the approach, enabling PH approximation for larger data and in higher dimensions.

The remainder of this paper is organized as follows. Section II contains background on TDA and specifically PH. Section III identifies work related to the high performance computation of PH. Section IV describes the technical approach for dis-

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.

tributing PPH computation. Section V details the development and implementation of distributed PPH. Section VI presents experimental results and relates the findings back to the method presented in this paper. Finally, Section VII remarks on the experimental study and future work with PPH.

II. Background

This section provides a brief introduction to PH along with distributed approaches for computing *persistence intervals* over large point clouds. Details on the sequential computation of PH are available at [4], [6], [7], [26], [27].

The output of PH is a set of persistence intervals H_d at each dimension d. H_0 persistence intervals represent connected components, H_1 represent loops, H_2 represent voids, and so on. Each persistence interval is a 2-tuple, $< \epsilon_{birth}, \epsilon_{death} >$, that describes when that unique topological feature appears (ϵ_{birth}) and disappears (ϵ_{death}). The distance between ϵ_{birth} and ϵ_{death} denotes the *persistence* of the feature, with larger distances typically indicating more significant features.

A set of persistence intervals characterizes the unique topological structure within a given point cloud. Two sets of persistence intervals are compared using a distance metric, such as the Bottleneck, Wasserstein, or Kernel metrics to enable classification and modeling. These comparisons may be carried out on a subset of the persistence intervals, such as those filtered by a persistence threshold or of a certain topological dimension of interest.

Several computationally expensive steps are necessary to extract persistence intervals from the input point cloud. Notably, construction of the *filtered simplicial complex* and subsequent *boundary matrix reduction* are the primary contributors to the time and space complexity of the algorithm. These complexities scale from the number of simplices, which grow exponentially in higher dimensions.

Distributed approaches can utilize the larger memory of a cluster to compute PH. Exploration of distributed PH is recent and has identified algorithmic improvements to compute the exact persistent homology [28], [29]. The approaches still suffer from the ever-increasing number of simplices in higher dimensions and the reliance of the boundary matrix to be reduced atomically, from left to right. This dependency limits the extension of the sequential approach to moderate performance improvements with distributed systems.

Partitioned Persistent Homology (PPH) provides a piece-wise approach that both reduces the overall memory footprint and exploits distributed memory for computing PH over larger point clouds and in higher dimensions. The first step of this technique involves decomposing the point cloud into overlapping regional partitions.[1] The partitions are analyzed separately to locate the smaller topological features in the point cloud. In addition to computing PH on the partitions, the centroids of the partitions are used to approximate the large persistent topological features that exist across the partitions of the point cloud. Once the separate PH computations are computed, the results are merged to form a characterization of the topological features in the entire point cloud. With this piece-wise technique, each of the PH computations can be performed concurrently and independently.

In addition, PPH can be applied as a general wrapper to alternative complex structures, approximations of the space, and boundary reduction optimizations. Utilization of the technique demonstrates the capability to further extend the size and dimensionality of point clouds examined by PH.

[1]The overlap is required to ensure that topological features on the boundary between the partitions are also identified.

III. Related Work

Computational PH has only recently become feasible, notably after the work of Carlsson *et al* [30]. Attempts to reduce the complexity of PH have been studied in several general areas: complex storage [31], [32], complex filtration and reduction [16], [19], [33], [34], complex approximation [11], [35], and general data reduction techniques [10], [12]–[15]. These optimizations have enabled PH for low-dimensional, moderately sized data-sets. Parallel execution of PH on GPGPUs has been explored in [36]. The GPU approach accelerates run-time, but provides no assistance to alleviate the more significant problem of memory pressure for computing PH.

Variations of the sequential persistent homology algorithm are common for optimizations and alternative approaches to computing PH. The implementation described in this paper utilizes the Vietoris–Rips filtration of the complex [37], computes the cohomology [38], and employs the twist and clear [19] and emergent pairs [18] optimizations. Implicit representation of the boundary matrix [18] is also implemented to provide compact representations and reduced memory footprints of each PH computation. While there are many additional optimizations, complex structures, and reduction techniques that can be paired with PPH, the above methods represent some of the best performance options currently available.

One method of distributed PH computation was introduced by Bauer *et al* [17] and implemented in DiPHA. The DiPHA library provides chunking and distribution of the boundary matrix to perform a distributed reduction step. While DiPHA has achieved notable performance improvements, it still suffers from the memory constraints of its internal representation of the boundary matrix [18]. Memory complexity has been targeted more recently with implicit representation models for Vietoris–Rips complexes [18] but have not yet been applied to a distributed approach.

Partitioned Persistent Homology (PPH) has several parallels to the witness complex [14], in which a set of landmark points is used to represent regions of the space with witness points locally similar to the landmarks. This representation results in a reduced complex with error bounded by the stability of persistence diagrams [39]. A similar approach using random samples of the point cloud to approximate the PH was introduced by Chazal *et al* [10]. These ideas were built upon by Moitra *et al* in the single-pass approximation of the large topological features using nano-clusters [12] and

reconstruction of the smaller topological features from partitions developed by Malott *et al* [23]. Unfortunately, each of these approaches result in some loss of the smaller topological features. PPH leverages the same sampling techniques of these approaches *and* utilizes the computation of PH on partitions of the data to recover the persistence intervals for the smaller topological features.

IV. Overview of the Approach

The distributed computation of PH enables TDA on larger data-sets where generated simplicial complexes grow beyond memory limits of a single node. DiPHA's distributed approach continues to build the complex for the original point cloud; thus, it still suffers from the exponential memory growth that prevents the computation of PH for big data. PPH operates on subsets of the original point cloud and thus, reduces the memory footprints of the simplicial complexes that are constructed for each partition. Previous results [12], [23], [40] have shown favorable approximations using PPH.

The PPH technique is intended to approximate PH on big data and in high dimensions. Partitions and centroid representations of the point cloud are analyzed to characterize the persistence intervals of data beyond current limitations. In the remainder of this paper the following symbols are used to describe the approach with respect to the partitions and their representative centroids:

- P, the point cloud,
- $\hat{P}$, the partitions,
- $n_{\hat{P}_i}$, the size (number of points) of each partition,
- P', the centroids,
- r_i, the distance from the partition centroid, $P'_i \in P'$, to the most distant point in that partition, and
- $r_{max} = max(r_i)$, the maximum r_i of all the partitions.

Distributed PPH requires an understanding of (i) the partitioning, (ii) approximation of the persistence intervals from the partitioned point cloud, (iii) the effects of the partitioning algorithm and number of generated partitions on the accuracy and performance of the distributed approach, and (iv) merging of partitioned persistence intervals into a singular set of persistence intervals. A high level outline of the steps of this paper are:

1) Partition the point cloud P such that each point is assigned to a single partition. The number of partitions generated should be scaled to fit within the memory bounds of the system; generally this bound can be estimated by the number of points and dimension of homology to compute. *The targeted number of partitions for this step is bounded by M, the maximum number of points in $\mathbb{R}^d$ at homology dimension H_{max} that PH can be performed.*

   There is an obvious trade-off between the number of partitions, k, and the size of each of those partitions, $n_{\hat{P}_i}$. When k is increased with a static number of input points, the value of $n_{\hat{P}_i}$ will decrease. A larger k will also preserve more of the salient topological features of the space [12]. This effect is described in Section IV-A.
2) Compute the PH of the centroid-replaced point cloud P'. The PH from the reduced data space will provide approximated persistence intervals bounded by an error of $2r_{max}$ [12]. Persistent topological features are thus preserved in the approximated point cloud. These approximated intervals can be further refined using the original point cloud and partition mapping (Section IV-B).
3) For each partition defined in Step 1, compute PH on a region of points within and around the partition (these PH computations can occur in parallel among each other and concurrently to Step 2). In particular, the size of this region is slightly larger than the points in the contained partition to ensure identification of features spanning multiple partitions (Section IV-A).
4) Due to the expansion of the partition boundaries for computing regional PH, the computations may produce duplicate persistence intervals from the overlapping regions between the partitions. Any duplicate features found by the regional PH computations can be eliminated by creating an arbitrary total order on the points in the original data-set. Each regional computation then reports only $< \epsilon_{birth}, \epsilon_{death} >$ intervals for topological features where the lowest ordered point in the convex hull of that feature is a member of that regional partition. A final step is to remove duplicate $< \epsilon_{birth}, \epsilon_{death} >$ intervals returned from the centroid based PH computation that are also discovered in a regional PH computation. The regionally computed $< \epsilon_{birth}, \epsilon_{death} >$ interval is preserved as it will generally have a more precise computation of the feature $< \epsilon_{birth}, \epsilon_{death} >$ interval than the centroid based PH computation (Section IV-D).

The remainder of this section describes the general steps to distribute PPH; details on their experimental implementation is provided in Section V.

A. Data Partitioning

PPH attempts to work on regional partitions of the original point cloud to identify smaller persistence intervals. These regional results are merged with an estimate of the larger topological features in the data using a representative point from each partition. In general, a partitioning assigns every point to a singular centroid. Formally, $\hat{P} = \{p \mid p \subset P\}$ is defined to be a partitioning of P if $\forall p, q \in \hat{P} \mid p \neq q$, then $p \cap q = \emptyset$ and $\cup_{p \in \hat{P}}\ p = P$.

Centroid-based clustering algorithms work surprisingly well in this scheme [40]. Classification with k-means++ [41] preserves both dense and sparse topological structures in point clouds up to a significant amount of reduction and can reasonably approximate the target persistence intervals in the approach.

The data partitions distributed for PH computation include not just the points in the partition mapped by k-means++ but

additional points within the error bound of the persistence intervals (as shown by Moitra *et al* [12]). Partitions are organized into distributable units by enumerating the points within the partition identified by k-means++, then subsequently adding points to the partition that are within some radius of the centroid of the partition. In particular, $2r_{max}$ provides an upper bound on the lost features shared between two partitions. This set of fuzzy partitions is referred to as $\hat{P}'$.

Unfortunately, with a large r_{max}, this upper bound can significantly increase the size of the partition to be distributed. For this reason scaling r_{max} to control partition size with a scale factor $0 \leq S \leq 2$ is explored in this work. When $S = 0$ no additional points will be added to the partition; at $S = 2$, all points within a radius of $2r_{max}$ will be added to the partition. A smaller scalar will result in smaller partitions but they may miss features that form up to $2r_{max}$.

B. Centroid Approximated Persistent Homology

The results of k-means++ are a set of centroids representing the geometric center of each partition. These representative points, P', provide an approximation that preserves the general shape of the original point cloud. Dense areas of the point cloud are thinned significantly, while sparse areas are preserved. This effect comes from the variance of points that are included in each partition when k-means++ attempts to minimize the inter-class variance, related to r_i and r_{max}. By choosing an algorithm that inherently minimizes the error induced in the persistence intervals, the approximate space can preserve the salient topological features of the space with a large degree of reduction.

Previous study of the centroid approximated PH has determined the resultant shift of the persistence interval to be no more than $2r_{max}$ [12]. This can be derived from the stability of persistence intervals examined by Chazal [42]. The theorem is adapted to the notation for partitions, and uses $Dg_{\mathbf{P}}$ and $Dg_{\mathbf{P}'}$ to represent the persistence intervals obtained from the original point cloud and centroid approximated cloud, respectively.

$$W_\infty(\mathrm{Dg}_{\mathbf{P}}, \mathrm{Dg}_{\mathbf{P}'}) \leq 2H(\mathbf{P}, \mathbf{P}') \tag{1}$$

where the Hausdorff distance, $H(\mathbf{P}, \mathbf{P}')$ represents the maximum distance from any point in the original space to the nearest neighbor in the centroid approximated space and the Wasserstein distance, $W_\infty(\mathrm{Dg}_{\mathbf{P}}, \mathrm{Dg}_{\mathbf{P}'})$, codifies the 'distance' or error between the persistence intervals. With a spherical clustering algorithm such as k-means++, the nearest centroid to any point is the assigned centroid; thus the maximum distance of any one point to respective centroid is r_{max}. Replacement into the stability theorem yields:

$$W_\infty(\mathrm{Dg}_{\mathbf{P}}, \mathrm{Dg}_{\hat{\mathbf{P}}}) \leq 2\, r_{max}. \tag{2}$$

This upper bound can be utilized to identify persistence intervals smaller than $2r_{max}$ within the individual partitions in most cases. Experimental results [12], [23], [40] show that large topological features, specifically when the feature is born after r_{max}, can be identified with significant reduction. In some cases, centroid approximations reducing the original point cloud by 95% still identify the large topological features.

Balancing the number of partitions, k, and the error induced becomes intuitive; with a larger k, the larger topological features will be identified with more accurate bounds. However, the use of boundary *upscaling* can permit the use of a smaller value of k [23]. With the mapping of centroids back to their constituent points in the original point cloud, *generators* [43] of the persistence intervals identified in the approximated space can be evaluated to refine the larger topological feature persistence intervals. This refinement of boundaries from the centroid map is an open area of study for the distributed structure, but presents significant memory requirements with larger and higher-dimensional topological features.

Iterative refinement, the process of continuously refining the larger topological features while balancing the available memory space, extends beyond the scope of this study but can utilize the distributed structure demonstrated. Additional details of iterative refinement are provided in Section VI-C3.

C. Distributed Computation

The distributed computation of persistence intervals for each of the partitions, $\hat{P}_i$, relies only on the data of each fuzzy partition. Persistence intervals identified within the partition are not identified in the approximated point cloud, P'. Each fuzzy partition and the set of centroids are distributed as independent units of work. The resulting persistence intervals are returned from each process. In this step it is necessary that *generators* [43] of identified topological features are computed by the distributed workers. Generators of the persistence intervals provide a method to compare results from the concurrent PH computations and remove duplicates. Each distributed partition reports the generators mapped into the original data indexing to enable merging of the distributed persistence intervals.

Fuzzy partitions require an additional step. Features may span multiple partitions, and thus each worker must filter the persistences intervals locally before reporting intervals for the backend merge step to prevent duplicates. When distributing the fuzzy partitions, the points belonging to the original partition $\hat{P}_i$ need to be identified. This enables filtering out instances when the generator $x_{min} \notin \hat{P}_i$ for a persistence interval. The given interval will thus only be reported by the $\hat{P}_i$ containing x_{min}, the minimum generator of the interval.

Once all partitions and the centroid set are evaluated, the persistence intervals are merged (see Section IV-D). The result of the distributed approach is a single set of persistence intervals that characterizes both the regional and large topological features of the point cloud.

D. Merging of Results

Merging the regional results with the centroid-approximated persistence intervals is fairly straightforward with proper mapping between the partitions, centroids, and an arbitrary total ordering of points. In this work, points are ordered based on their corresponding partition. Centroids are mapped to a separate partition for processing. Persistence intervals

along with their constituent boundary points (generators) are computed for each partition. These persistence intervals and generators are then merged (with duplicates removed) through one of two methods based on the interval's dimension.

For connected components, the minimum spanning tree (MST) of the entire point cloud must be recreated from the partitioned results. The MST represents the H_0 features (connected components) of the set. If each partition is considered without additional overlapping points, the H_0 results of $\hat{P}_i$ is the MST of that partition. By merging the results of all $\hat{P}$ the MST of each individual partition is obtained.

Since the PH computation is performed on overlapping partitions, the persistence homology for H_0 generates the MST of the internal partition and minimum connections to all points included that were originally outside of the partition. The minimum additional persistence interval represents the minimum connection for the partition to join with another partition; without bounds on the overlap we can obtain the complete MST of the distributed partitions. The bound of overlap between partitions determines whether the H_0 boundaries can be completely recreated. In the case where partitions have a distance between them greater than $2r_{max}$ but less than ϵ_{max}, the H_0 connection between the two partitions will not be found with the distributed merge scheme. As the value of the partition radius approaches ϵ_{max}, the accuracy of the H_0 persistence intervals will improve.

For higher dimensional components $H_d \mid d > 0$, the boundary points for each persistence interval are examined to locate the point marked as the lowest in the total ordering of the points. In particular, for each persistence interval reported from partition $\hat{P'}_i$, if the boundary point containing the ordered value was contained in the original partition $\hat{P}_i$ then the persistence interval is merged; otherwise the interval is discarded. This method ensures that a dimensional boundary identified by multiple partitions is only reported once.

Merging these two results — the H_0 connected component persistence intervals and the $H_d \mid d > 0$ — provides a suitable result set for computing PH using distributed partitions. The distributed partition results can then be merged with the approximated PH using cluster centroids by simply appending the $H_d \mid d > 0$ results from P' to the results. The H_0 intervals are already generated by the distributed computation in this case. Only larger topological features identified and refined through P' need to be added to the merged results.

V. Implementation

The distributed PPH scheme described above was developed into the *Lightweight Homology Framework (LHF)* [25]. LHF is a C++ library for studying PH including partitioning algorithms, different complexes, and alternative methods for computing persistence intervals and generators. The MPI framework was used to provide communication between multiple processes running on the same machine or in a distributed environment.

For the experimental study, the implementation relied on a master-worker relationship between the nodes. The master process partitions the source point cloud and distributes the data to each of the worker nodes, tracking the relationship of the distributed partition to the representative centroid and original point cloud. Each node is assigned an equal number of partitions to evaluate. Worker nodes wait to receive partitioned data, compute the persistence intervals, and report the results back to the master node before exiting. The master node is also assigned a set of partitions to evaluate.

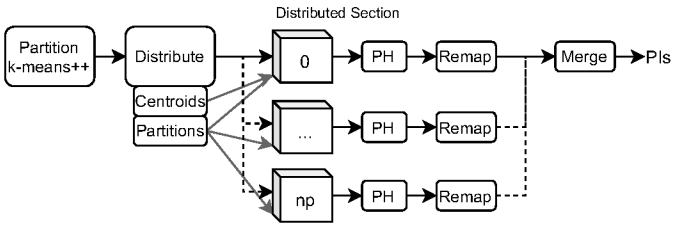

Fig. 1. LHF distribution implementation with MPI, where np is the number of worker nodes. Partitions are distributed out to worker nodes, executed on, remapped, and returned to the master node before merging.

Figure 1 depicts the data flow diagram for the LHF MPI implementation for distributed PPH. Scaling based on the number of nodes will have an improvement for the performance and run-time of the system. However, the size of the generated simplicial complexes from a partition may still exceed the memory of a node in the distributed system. It is necessary to study the distribution of points and partition sizes prior to using PPH in order to balance the memory limit (Section VI).

VI. Experimental Results

An evaluation of PPH requires: (i) analysis of the accuracy of persistence intervals compared to known results, and (ii) scalability studies of the distributed PPH computation. Small point clouds have been used to compare the results of approximate PH computations to exact PH libraries [10], [12], [23], [40]. Point clouds studied for the scalability portion of this paper are well beyond the limitations of current tools and it is, therefore, not possible to compare PPH to conventional (non-approximate) PH computation engines.

Three experiments are performed to demonstrate the effectiveness of distributed PPH. First, the Triangulated Mesh datasets [44] are compared to show improvement in the recognition of features when classifying 3D objects over previous subsampling and reduction methods [10], [12], [23]. Next, analysis of brain artery trees used by Bendich *et al* [21] are evaluated and correlated against patients to contrast the subsampling used in that study for estimating the H_1 topological features of the data. Finally the run-time and memory scalability of distributed PPH is presented.

LHF was designed to perform the partitioning, distribution, and merging of persistence intervals described in Section V. Input point clouds are partitioned with k-means++, distributed to worker nodes for persistence interval results, and merged together to compute the aggregate persistence intervals. In each experiment the run-time, memory, and output persistence intervals are recorded and compared for analysis.

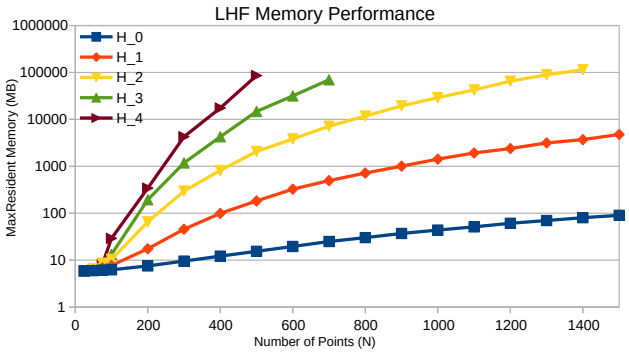

Fig. 2. Limitations of LHF based on the number of points on a single compute node with 128GB of memory. Each series represents the maximum dimension of homology to compute up to, H_{max}.

Prior to executing the PPH pipeline on any of the data-sets, the size of the generated simplicial complex must be evaluated. This analysis determines a suitable value for k, the number of partitions, that creates partitions of size less than or equal to the maximum memory bounds for the system. In the distributed case, each process must not exceed the system memory, which is directly tied to the number of points being evaluated in the partition. Figure 2 depicts the limitations of LHF using 128GB of memory at varying homology dimensions, H_d. The memory limitations of the system constrict the size of the generated simplicial complex as the dimension of homology to compute up to, H_{max}, increases

A. Dissimilarity Analysis of Persistence Landscapes

Comparison of persistence intervals from the triangulated mesh data-set has been studied previously [10], [12], [23]. The triangulated mesh data-sets represent several different mesh representations of animals in $\mathbb{R}^3$ of varying point cloud sizes from $n = 5,000$ to $n = 50,000$. Computing the PH up to H_2 to identify voids in the triangulated mesh point clouds is beyond the limitations of the traditional approach. Previous studies have used only a subset of sample points from the full point cloud to approximate the persistence intervals of the point cloud. These studies have shown favorable results.

For distributed PPH computation of the triangulated mesh data-sets, a heterogeneous cluster of four compute nodes is used. Three nodes have Intel(R) Xeon(R) CPU E5-1620 v2 @ 3.70GHz with 4-cores and 64GB of RAM. The fourth node is an AMD Ryzen Threadripper 1950X 16-Core Processor with 128GB of RAM. Throughout the experiments 5 processes are executed concurrently, one on each of the Xeon(R) machines and two on the AMD machine to allow each process up to 64GB of addressable memory. When the total number of partitions exceeds 5, the additional processes are scheduled round-robin to the nodes. Values of k were chosen independently for each data-set to ensure the partition sizes do not exceed available system memory per node.

The use of a *dissimilarity matrix* provides a general level of difference between structural components in point clouds. In the case of PH, there are several interesting structures that can be compared including the H_0 connected components, the H_1 loops in the point cloud, the H_2 voids, and higher dimensional structures that may provide greater differentiation. For this reason, the analysis separates the dimensional features identified from each of the point clouds to examine the dissimilarity of H_1 and H_2 features between each of the triangulated mesh models.

The k-centroid subsamples of each point cloud are used as input to the sequential PH method to establish a baseline of comparison because conventional PH tools cannot compute H_2 homologies at the scale of these test cases. Features identified in the centroid-approximated approach will be identical to the PPH approach, with additional features being identified in PPH due to the regional reconstruction of smaller persistence intervals. This allows determination of the improvement in dissimilarity between the models using the base case of centroid-approximated PH.

The dissimilarity plots for the centroid-approximated and PPH approaches using the Sliced-Wasserstein (SW) distance [45] are shown in Figure 3. The first row of plots shows results using the conventional PH algorithm; the second row shows results using PPH; and the third row show results using PPH with a fuzzy partition scale of $S = 0.5$. The first column compares all H_d persistence intervals, while the other two columns show, respectively, the dissimilarities of H_1 and H_2 features (H_0 is not shown due to space constraints, but the results are consistent with the others). A larger value of dissimilarity is represented by a larger SW distance — indicating that the persistence diagrams between the compared row and column data-sets are less similar. Lower values of dissimilarity indicate the generated persistence intervals are a closer match.

The results of Figure 3 show that PPH provides significantly better dissimilarity between the triangulated mesh models. A majority of this improvement is the result of the regional reconstruction where smaller persistence intervals are identified. In higher dimensions such as H_1 and H_2, the regional reconstruction with PPH also improves the overall dissimilarity between the models indicating structural differences that can provide greater insight into classification. This result is significant for classification, pattern, and object recognition in machine learning applications. A slight increase in dissimilarity can be observed with the addition of partition overlap with $S = 0.5$, which may be useful for additional discernment in some applications if desired.

B. Regional Reconstruction of Brain Artery Trees

Topological data analysis is one technique that has been used to analyze brain artery trees. In particular, Bendich *et al* [21] studied brain artery tree data-sets describing 98 patient MRA paths in $\mathbb{R}^3$ using the H_0 and H_1 features of PH. Interestingly, PH has been shown to provide correlations with the age and gender of the patients in that study. However, several of the correlations are constructed on the subsampled brain artery tree data; the sizes of the scans exceed 10^5

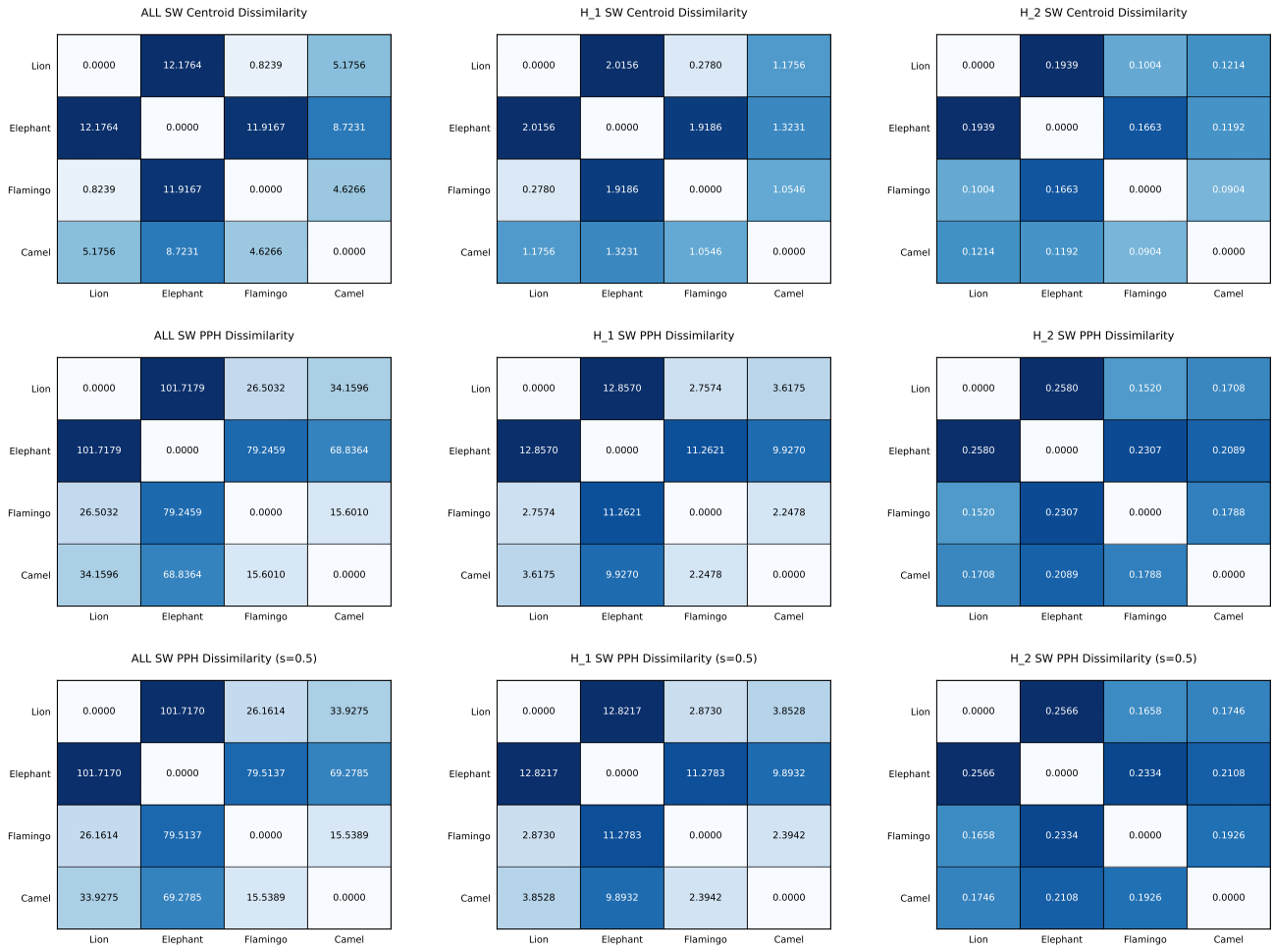

Fig. 3. Dissimilarity plots for the triangulated mesh point clouds.

points and are too large to compute with existing tools. In their approach, the brain artery trees are each subsampled to $3,000$ points and evaluated to find H_1 loops within the point cloud. PPH can improve this recognition of small H_1 features embedded in the subsampling that provides more clarity to the reported results.

In this paper the H_1 features are identified using PPH to reconstruct the smaller loops embedded in the partitions. These loops represent the loops within brain artery paths of the MRA point clouds; the birth of the loop represents when a feature was first formed over the point cloud and the death represents the coverage of the feature. The Bendich study has indicated an inverse relationship between the persistence ($\epsilon_{death} - \epsilon_{birth}$) of the longest intervals and the age of the subject. The experiment in this paper attempts to re-evaluate the correlation of the reconstructed H_1 persistence intervals using PPH against the subsampled method studies.

PPH results were compared to a centroid-approximated PH of $k = 3,000$. The persistence intervals for all 98 patients were computed to evaluate the Pearson Correlation Coefficient of the first principal component (PCA1) of persistence intervals against age and sex. Coefficients of greater magnitude indicate stronger correlations. Significant results for age are shown in Table I. PPH provides similar age correlation as the H_1 persistences demonstrated by Bendich. The set of persistence intervals between 150 and 250 when ordered by persistence also demonstrate significant correlation with the age of the patients. Importantly, other factors had a much more significant correlation with age, such as the H_1 ϵ_{birth} and ϵ_{death} times from the PPH results, indicating a substantial improvement over the subsampled approach.

Measurement	$Corr(P')$	$Corr(PPH)$
Top 150 H_1 ϵ_{birth}	0.24	0.12
Top 150 H_1 ϵ_{death}	0.43	0.63
Top 150 H_1 $persistence$	0.61	0.56
Top 150 to 250 H_1 ϵ_{birth}	0.27	0.63
Top 150 to 250 H_1 ϵ_{death}	0.07	0.61
Top 150 to 250 H_1 $persistence$	0.60	0.61

TABLE I
PEARSON CORRELATION COEFFICIENTS AND THE FIRST PRINCIPAL COMPONENT (PCA1) BETWEEN AGE AND VARIOUS SUBSETS OF THE PERSISTENCE INTERVALS.

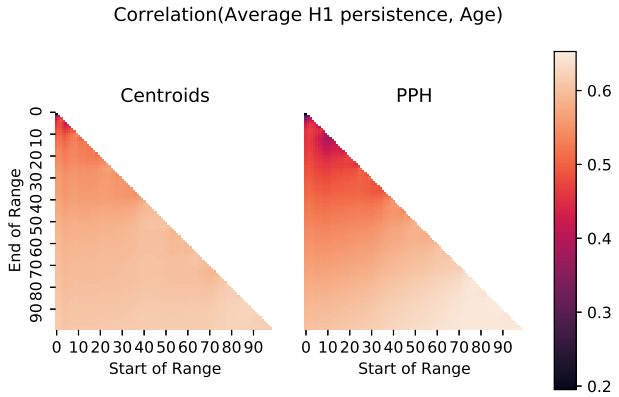

Fig. 4. Centroid approximated and PPH reconstructed H_1 Correlation (PCA1, Age). Higher values indicate a greater correlation between the patient's age and the range of persistences (x, y).

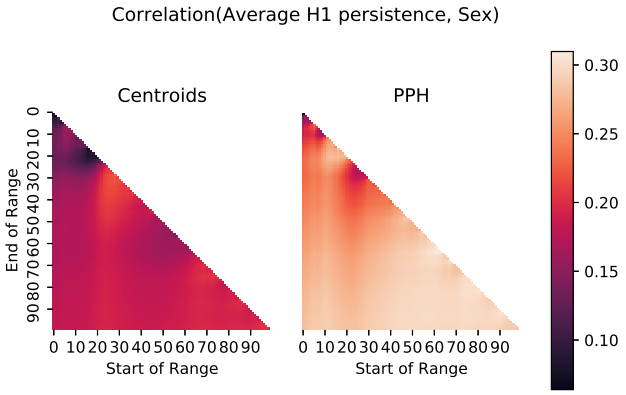

Fig. 5. Centroid approximated and PPH reconstructed H_1 Correlation (PCA1, Sex). Higher values indicate a greater correlation between the patient's sex and the range of persistences (x, y).

Measurement	$p(P')$	$p(PPH)$
Top 100 H_1 ϵ_{birth}	0.36	0.00
Top 100 H_1 ϵ_{death}	0.48	0.00
Top 100 H_1 $persistence$	0.73	0.06

TABLE II
P-VALUES FOR THE PERMUTATION TEST BETWEEN MALES AND FEMALES FOR TOP 100 H_1 FEATURES IN THE CENTROID AND PPH RESULTS.

The comparison was also evaluated for the p-values for a permutation test between males and females, as performed by Bendich *et al* [21]. The top 100 H_1 persistence intervals show a dramatic decrease in the p-value for determining sex of the individual from the brain artery tree data, indicating statistically significant results as depicted in Table II. This indicates that the PPH approach, which includes granular detail of the connections from the regional reconstruction step, can provide additional differentiating information with the smaller persistence intervals. This information is useful in studies to provide a fine-grained analysis of the topological features present in big data.

Figures 4 and 5 show heatmaps comparing the first principal component of ranges of H_1 features against patient age and sex for centroid sampling and PPH. The range (x, y) selects the top xth to yth barcodes when ordered by persistence. Results show an increase in the correlation for the largest 50 barcodes over the subsampled approach. This indicates more granularity and a greater determined relationship between the PPH persistences of patients and the sex of the patient. While these results confirm Bendich's initial findings of the correlation of the H_1 persistences, PPH provides better recognition due to regional reconstruction of small persistence intervals.

The findings in this section reaffirm the claims by Bendich *et al* [21] that the top H_1 persistence intervals have a significant relationship with the age and sex of the patients. Reconstruction of the regional persistence intervals through PPH amplify the significance of the relationship and provide detailed recognition of the represented topological features.

C. Performance

The performance of the distributed approach can only be analyzed from a scalability perspective; that is, the ground truths for larger datasets are not available and cannot be computed with current tools. For this reason the aim of the performance experiments are to characterize the impact of parameter selection and data-set size on both the time and space complexity of the approach. A study of the distributed implementation and performance scalability of PPH relies on several parameters:

- k, the number of partitions (clusters) to generate,
- H_{max}, the max dimension of homology to compute,
- np, the number of processes utilized, and
- n, the number of points in the original data-set.

The input parameters to distributed PPH will have several effects on the overall performance. The number of partitions, k, describes how many partitions will be created in the first step. This controls the number of partitions distributed, the size of each of those partitions, and the number of points used to estimate large topological features. Naturally, the number of partitions will affect r_{max} and, when fuzzy partitions are used, the number of additional points brought into each partition for identifying features spanning multiple partitions.

The maximum dimension of homologies to compute, H_{max}, determines the size of the complex generated from the point cloud alongside the complexity of extracting the persistence intervals. Each distributed process receives several partitions and H_{max}; the process computes PH up to H_{max} and returns the relevant persistence intervals. The master process then merges the results from each partition to report the persistence intervals computed by the distributed PPH algorithm.

The number of points, n, will also play a significant role in measuring the performance of the distributed system. As n increases, the number of points per partition will increase. The inclusion of points within a radius of $2r_{max}$ in each partition will duplicate points between different partitions while the

number of partitions, k, remains constant. This indicates the system can serve two purposes: a large master node that computes with a large k, with smaller partitions distributed to worker nodes, or a smaller k used for the master node and larger partitions distributed to worker nodes for evaluation.

Collectively, the k, H_{max}, and n directly affect the generated number of simplices in the simplicial complex. In the sequential approach with a Vietoris–Rips complex, the number of generated simplices is $O(n^{H_{max}+1})$. Storing, filtering, and reducing the boundary matrix from such a large number of simplices is not possible on a single node. However, partitioning and separating the larger complex into smaller complexes enables the distributed approach presented in this study with a significantly reduced memory footprint for increasingly smaller complex reductions in PH computation.

A larger compute cluster was used to evaluate the scalability of the approach. The ARCC High Performance cluster offers up to 36 total nodes, each with an Intel Xeon Gold 6148 CPU and 192GB RAM with an Omnipath Networking infrastructure. In the scalability experiments, the MPI-enabled PPH approach was evaluated up to 16 compute nodes for speedup results and 32 nodes for a large-scale example.

The remainder of this section is organized as follows. Section VI-C1 approaches the scalability of the system by first identifying the memory limits of PPH and measured parallel speedup. The speedup is then analyzed in terms of sequential overhead for partitioning and merging of results in Section VI-C2. Finally a brief summary of Iterative PPH for further partitioning and refinement of topological features is provided in Section VI-C3. Each of these experiments utilizes synthetic d-spheres generated at different sizes and dimensions to characterize the overall performance of distributed PPH.

1) PPH Memory Limits: Serial execution of LHF for memory and runtime performance was previously presented in Figure 2. On a single node the limiting factor is the available amount of memory; if the maximum available memory is exceeded, PPH fails. With each partition being treated as a single unit of work, if the size of any partition exceeds the memory limitations, the approach will also fail. In a distributed memory approach, each partition must be sized appropriately to fit within any of the nodes of the cluster.

Figure 6 presents the partition size limitation based on available system memory to demonstrate the effect of the minimum node's memory space. Any individual partition can be assigned to any node; if the largest partition is assigned to a node with insufficient memory, PPH will fail. Fortunately an iterative application of PPH on an individual partition can be performed to address this problem.

In the most extreme case of PPH, each of the k partitions contains M points, where $k \leq M$. In practice, the sizes of the partitions vary significantly and we therefore achieve slightly less scale for the parallel speedup. This indicates that with a maximum partition size of M in a partitioning where $k = M$ and $n_{\hat{P}_i} = M \ \forall \hat{P}_i \in \hat{P}$, the maximum point cloud size that PPH can perform on is of size M^2. Standard PPH enables the distributed PH of point clouds beyond 100k points, and

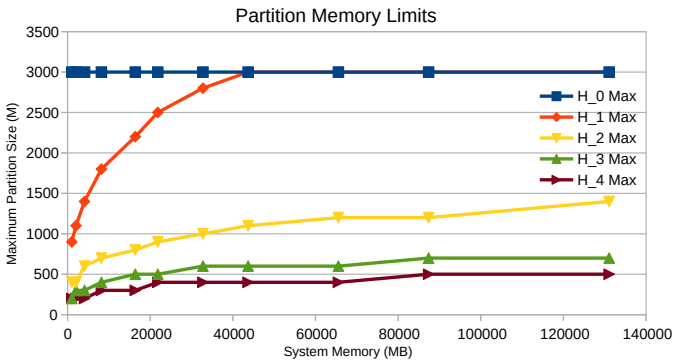

Fig. 6. Measured maximum partition point size (M) based on available system memory of a single node, up to M=3000.

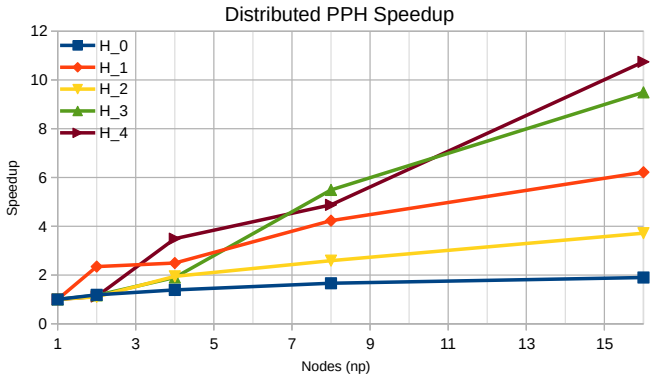

Fig. 7. Measured parallel speedup of distributed PPH based on number of nodes executed on.

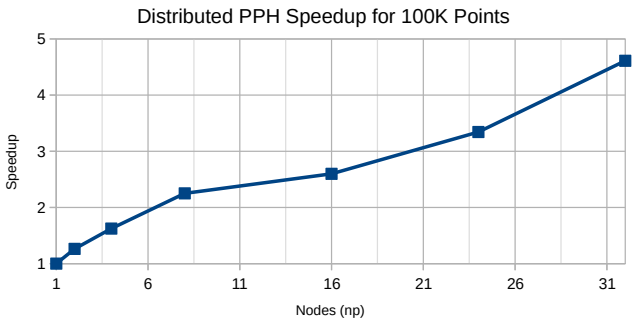

Fig. 8. Distributed PPH speedup for 100k points up to H_2 based on number of nodes. A seeded k-means++ with $k = 1200$ was used to generate partitions and scalar $S = 0.5$ was utilized for regional reconstruction.

in some cases beyond 1m points. An iterative PPH, described in Section VI-C3, can extend this limit even further. This is a significant step towards computing PH on big data.

Parallel speedup from the PPH approach is intuitive; each partition is distributed to one of the np nodes and concurrently executed. Figure 7 presents the parallel speedup from a multi-node distributed approach on synthetically generated data. In higher dimensions the system RAM limitation constrains the maximum data size for PH and consequently the speedup

n	k	H_{max}	k-means++ (s)	PPH (s)	MPI (s)	Overhead (%)
20000	250	1	191.65	336.60	4.16	36.78
20000	250	2	189.50	9580.35	18.26	2.12
10000	250	2	112.45	5789.67	16.72	2.18
5000	250	3	26.97	18639.82	9.31	0.19
1000	100	4	1.55	2134.13	12.21	0.64

TABLE III
A SAMPLING OF SEQUENTIAL TIMES FOR COMPUTING k-MEANS++, RUNNING PPH COMPUTATIONS, AND MPI OVERHEAD FOR DISTRIBUTION AND MERGING OF PARTITIONS. ALL TIMES WERE TAKEN FROM A SYNTHETIC D-SPHERE OF DIMENSION 8.

attained. Evaluation with larger system memory on more distributed nodes will provide additional scalability as the size of the point cloud increases in higher dimensions.

Figure 8 explores the runtime of PPH up to H_2 with $100,000$ points using different node configurations. This size of data is well beyond the limitations of current PH approaches in reasonable space-time complexity. Additional nodes and RAM can continue to expand the capabilities for higher dimensions and larger point clouds. Distributed PPH demonstrates significant improvement in both memory space and runtime over the single-node approach and enables approximate topological data analysis on big data beyond current limitations of the exact approach as demonstrated in this scalability study.

2) Distributed Overhead: Distributed PPH exploits the independence of each generated partition to compute the persistent homology by parts. As indicated in Figure 1, the parallel portion that benefits from distributed processing includes the individual PH computations and partition-local filtering and remapping of identified intervals. Sequential functions of the system, such as partitioning of the input data-set, distribution of the partitioned sets, and merging of results can potentially limit the scalability of the system.

Table III provides the sequential processing time over several workloads with distributed PPH. The k-means++ and MPI portions represent the partitioning and sequential distribution steps. The PPH time is the total execution time of PH on the partitions. The overhead percentage is the percentage of total processing time in the k-means++ or MPI modes.

In general the sequential overhead portions of the technique are minimal compared to the persistent homology computations required for analysis. This is a result of the implementation only broadcasting the partitioned data at startup and gathering after each process has finished their set of workloads. A majority of the processing time for higher dimensional persistent homology, especially as H_{max} exceeds 2, is spent on PH computations for each of the partitions. In low dimensions, such as H_0, the overhead significantly limits scalability.

Notably the $O(nkd)$ runtime complexity of k-means++ is significant in the larger datasets, such as the $100k$ d-sphere in Figure 8. Alternate partitioners such as distributed k-means++ may be employed to take advantage of multiple nodes during the partitioning step.

3) Distributed Iterative PPH: While the method utilized in the scalability studies of this paper focuses on a single partitioning and distribution of units of work to worker nodes, an additional method has been previously introduced in [23] to iteratively repartition distributed sets when the number of points exceeds system memory limits. This Distributed Iterative PPH has been implemented into the LHF library; workloads of each worker node are repartitioned accordingly when space does not permit a full reporting of results.

In the distributed iterative PPH scheme any partition such that $n > M$ where n is the source number of points and M is the maximum number of points for PH in dimension H_{max} can be further partitioned and reconstructed. The iteration follows the same steps as the standard approach requiring partitioning, PH computation, remapping, and merging for any partition that requires further reduction to fit into system memory.

VII. CONCLUSIONS

The exponential memory complexity of computing PH inhibits the use of Topological Data Analysis on large data-sets. While attempts to approximate or distribute the computation of PH have been studied, few have successfully exploited speedup through the multiplicity of memory resources to permit the efficient computation of PH on big data. The partitioning and approximation of PPH has demonstrated significant reduction of the memory footprint and functions as a wrapper to the PH algorithm and future optimizations in the computation. By piece-wise approximation of the data-set and merging of the results, PPH permits the computation of PH on significantly larger data-sets. The lack of incremental communication by the regional PH computations permits the deployment on large compute clusters that scales efficiently.

Furthermore, a remapping of the spaces rectifies lost persistence intervals, providing an improved approximation beyond the current memory limitations. Merging and upscaling of partitioned persistence intervals as described in this paper is a significant step towards the complete identification of the topological features in the space. This continues to be an active area of study and may uncover the ability to completely recreate the persistence intervals from partitioned datasets.

While k-means++ was used in this study, the approach has been shown to work with other partitioning algorithms such as agglomerative, DBScan, and Mean-shift [40]. Depending on the underlying structure of the data there may be features identified with some partitioners that are lost with others. Performance becomes a concern as the method is extended into the big data scope, which may require a more efficient or distributed partitioning algorithm. However, k-means++ scales well and, in general, provides very good results.

The results in this paper show promise for the piece-wise approximation to reduce the overall memory limitations for PH. This study has characterized the accuracy and performance for several synthetic and real-world data-sets and explored synthetic data-sets beyond the current limitations of exact PH libraries. Improvements to the iterative refinement of large topological features and merging of higher dimensional persistence intervals provide a framework for further improvement of the persistence intervals, should that be desired.

REFERENCES

[1] G. Carlsson, "Topology and data," *Bulletin of the American Mathematical Society*, vol. 46, no. 3, pp. 255–308, Apr. 2009.

[2] 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.

[3] R. Ghrist, "Barcodes: The persistent topology of data," *Bulletin of the American Mathematical Society*, vol. 45, no. 1, pp. 61–75, 2008.

[4] F. Chazal and B. Michel, "An introduction to topological data analysis: Fundamental and practical aspects for data scientists," *ArXiv e-prints*, Oct. 2017.

[5] H. Edelsbrunner and J. Harer, "Persistent homology — a survey," *Surveys on Discrete and Computational Geometry*, vol. 453, pp. 257–282, 2008.

[6] C. S. Pun, K. Xia, and S. X. Lee, "Persistent-homology-based machine learning and its applications – a survey," Nov. 2018.

[7] 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.

[8] T. K. Dey, D. Shi, and Y. Wang, "SimBa: An efficient tool for approximating rips-filtration persistence via simplicial batch collapse," *ACM Journal of Experimental Algorithmics*, vol. 24, pp. 1.5:1–1.5:16, Jan. 2019.

[9] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington, "A roadmap for the computation of persistent homology," *arXiv preprint arXiv:1506.08903*, Aug. 2015.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] U. Bauer, M. Kerber, and J. Reininghaus, "Clear and compress: Computing persistent homology in chunks," in *Topological Methods in Data Analysis and Visualization III*, P. T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, Eds. Springer International Publishing, Mar. 2014, pp. 103–117.

[17] ——, "Distributed computation of persistent homology," in *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2014, pp. 31–38.

[18] U. Bauer, "Ripser: efficient computation of vietoris-rips persistence barcodes," 2019.

[19] 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.

[20] 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.

[21] 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.

[22] P. Bubenik, M. Hull, D. Patel, and B. Whittle, "Persistent homology detects curvature," *Inverse Problems*, vol. 36, no. 2, Jan. 2020.

[23] 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.

[24] J. M. Patchett, B. Nouanesengesy, J. Pouderoux, J. Ahrens, and H. Hagen, "Parallel multi-layer ghost cell generation for distributed unstructured grids," in *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, Oct. 2017, pp. 84–91.

[25] Researchers at The High Performance Computing Laboratory. (2020) LHF: Lightweight homology framework. The University of Cincinnati. [Online]. Available: https://github.com/wilseypa/lhf

[26] 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.

[27] X. Zhu, "Persistent homology: An introduction and a new text representation for natural language processing," in *IJCAI*, 2013, pp. 1953–1959.

[28] D. Morozov and A. Nigmetov, "Towards lockfree persistent homology," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '20, Jul. 2020, pp. 555–557.

[29] 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.

[30] 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.

[31] 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.

[32] 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

[33] M. Mrozek and B. Batko, "Coreduction homology algorithm," *Discrete & Computational Geometry*, vol. 41, no. 1, pp. 96–118, Jan. 2009.

[34] U. Bauer. (2018) Ripser. The Technical University of Munich. [Online]. Available: http://www.cs.umd.edu/mount/ANN/

[35] J. A. Barmak and E. G. Minian, "Strong homotopy types, nerves and collapses," *Discrete & Computational Geometry*, vol. 47, no. 2, pp. 301–328, Mar. 2012.

[36] S. Zhang, M. Xiao, and H. Wang, "Gpu-accelerated computation of vietoris-rips persistence barcodes," *arXiv preprint arXiv:2003.07989*, 2020.

[37] A. Zomorodian, "Fast construction of the vietoris–rips complex," *Computer and Graphics*, pp. 263–271, 2010.

[38] V. de Silva, D. Morozov, and M. Vejdemo-Johansson, "Dualities in persistent (co)homology," *Inverse Problems*, vol. 27, no. 12, 2011.

[39] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, "Stability of persistence diagrams," *Discrete & computational geometry*, vol. 37, no. 1, pp. 103–120, 2007.

[40] 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.

[41] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[42] F. Chazal, V. de Silva, M. Glisse, and S. Oudot, "The structure and stability of persistence modules," *arXiv preprint arXiv:1207.3674*, 2012.

[43] O. Busaryev, T. K. Dey, and Y. Wang, "Tracking a generator by persistence," in *Computing and Combinatorics (COCOON)*, ser. Lecture Notes in Computer Science, vol. 6196. Berlin, Heidelberg: Springer Verlag, 2010, pp. 278–287.

[44] R. W. Sumner and J. Popovic, "Mesh data from deformation transfer for triangle meshes," 2004. [Online]. Available: https://people.csail.mit.edu/sumner/research/deftransfer/data.html

[45] M. Carriere, M. Cuturi, and S. Oudot, "Sliced wasserstein kernel for persistence diagrams," *arXiv preprint arXiv:1706.03358*, 2017.