Implementing Persistence-Based Clustering of Point Clouds in the Topology ToolKit

Ryan Cotsakis, Jim Shaw, Julien Tierny, Joshua A Levine

Implementing Persistence-Based Clustering of Point Clouds in the Topology ToolKit

Ryan Cotsakis*, Jim Shaw*, Julien Tierny, and Joshua A. Levine

Abstract We show how the scalar field topology features of the Topology ToolKit (TTK) can be leveraged in a pipeline for persistence-based clustering of point clouds. While TTK provides numerous features for computing topological structures of scalar fields on unstructured meshes, prior to this work, it allowed for only basic point cloud input. In this work, we implemented two new modules in TTK: one for sampling scalar fields based on either distance or density of the point cloud and a second for computing persistence-based clusters. Both modules provide heuristics for automatically specifying key thresholds so as to simplify user interaction. This document outlines the implementation details of the two modules and provides experimental results that demonstrate their modularity and utility.

1 Introduction

Clustering is an important tool used widely in data analysis. We consider the problem of clustering point clouds. The input dataset is an unstructured collection of points that are a discrete subset of $\mathbb{R}^d$. Clustering seeks to partition the points into a set of logical groups (clusters) such that points in the same group are more similar to each other than they are to points in the other groups. From a data analysis perspective,

Ryan Cotsakis*
University of British Columbia, e-mail: ryancotsakis@gmail.com

Jim Shaw*
University of British Columbia, e-mail: jimshawster@gmail.com

Julien Tierny
CNRS / Sorbonne Université, e-mail: julien.tierny@sorbonne-universite.fr

Joshua A. Levine
University of Arizona, e-mail: josh@email.arizona.edu

*Both authors contributed equally to the work

clustering is often an important module for other downstream tasks that either directly utilize the clusters, or study how many, how large, or how spread out clusters are.

When the data is unstructured, as is the case of point clouds, clustering typically requires building a model based on other prior knowledge or assumptions on the dataset. This knowledge can be used to infer distance and/or similarity measures or otherwise make decisions about how to define what properties delineate grouping. As a result there is no singular goal for clustering applicable in all settings.

On the other hand, segmenting a scalar field has similar goals to clustering and provides an interesting counterpoint. Compared to segmenting a point cloud, segmenting a scalar field has a variety of tools that naturally decompose the field based on features encoded in the scalar field. Common techniques for this include watersheds in image segmentation [5, 6] and topological segmentation via the Morse complex [12, 26]. A particular advantage of topological segmentation techniques is the ability to provide a hierarchy of segmentations that respect a notion of simplification. Specifically, topological persistence allows one to rank the importance of segments, jointly simplifying the scalar field while leading to a coarser segmentation [13].

Leveraging the strengths of using scalar field topology for segmentation, one approach for segmenting point clouds is to first compute a scalar field from a given input point cloud. This scalar field serves to provide the additional prior information and provide structure for the dataset. For example, one simple proposition is to use a scalar field that estimates the density of the points themselves, resulting in clusters that relate to those computed by density-based clustering [14]. Of course, the choice of scalar field is a free parameter to the clustering algorithm and other scalar fields could be use that encode distance or similarity. After segmenting the scalar field, one can then associate each point with the labels assigned to nearby regions of the domain of the scalar field, and then use this assignment to produce the final clustering.

1.1 Contributions

In this work we leverage the features of scalar field segmentation for point cloud segmentation. We primarily follow the methodology of Chazal et al.'s algorithm ToMATo [9] for designing the scalar field, except we replace the domain as represented by a neighborhood graph with a simple triangulated grid on which we sample the field. Our main contributions are the implementation details for adapting this approach into a large framework for topological analysis, the Topology ToolKit (TTK) [27]. Specifically, our contributions are:

- We describe the implementation for persistence-based clustering of point clouds in TTK. This involved implementing two new modules: one for computing a scalar field from a point and a second for clustering via persistence.
- As we aim for a non-parametric method, we design heuristics for automatically selecting parameters involved in both modules that work well in most settings.

- We report experimental details on the efficacy of our heuristics as well as discuss important design considerations.

2 Related Work

Clustering methods are well-studied over the past forty years, with common approaches that include k-means [19], density-based clustering like DBSCAN and its variants [14, 24], mode-seeking methods [18] such as mean shift [10], and spectral clustering [30]. These methods have been used in a wide variety of applications in data analysis, image processing, computer graphics, and statistics. Thus, a variety of practical tools implement clustering, thus making it desirable for the users of TTK as well.

Our work emphasizes using topological tools for clustering, via computing segmentations that can be simplified through topological persistence [13]. Topological analysis has led to a variety of tools for data analysis in general, and relies on the field of persistent homology to compute and rank features (see Edelsbrunner and Harer for a thorough introduction [11]). Chazal et al. were one of the pioneers of using topological persistence for guiding clustering of point clouds [9] and as mentioned previously our work is using a conceptually similar pipeline adapted to TTK [27]. This approach builds on key theoretical results where the same authors show one can recover structural information of scalar fields from samples [8].

Others have also pursued using topological analysis for clustering. Particularly, Beksi and Papnikolopoulos have shown the utility of clustering 3D point clouds [2, 4] and designing signatures for points [3]. Moon et al. design the persistence terrace to help provide a summary plot to guide the inference process [20]. These methods all focus on low-dimensional point sets (typically two- and three-dimensional data). Nevertheless, many applications of clustering focus on higher dimensional point clouds. Using topological analysis can also work in this setting, but there are certain challenges with computing and visualizing the topological structure of segments. Oesterling et al. show methods for capturing the topology of high-dimensional point cloud density fields [21] by constructing topological landscapes to provide a tangible metaphor [22].

3 Software Design Overview

We implemented persistence-based clustering for point clouds in TTK by creating two new modules. The first module, *ScalarFieldFromPointCloud*, computes a scalar field from a given input point cloud that will be used to analyze the structure of the point cloud. The second module, *PersistenceSimplification*, is used to simplify the scalar field in preparation for topological segmentation. Both modules enable a few

user controls for segmentation, but can also be used in an automatic mode where all parameters are set by default heuristics.

To complete the clustering, we wrap these modules in a pipeline with five steps: (1) read the input point cloud, (2) compute a scalar field, (3) simplify this field with persistence, (4) perform topological segmentation with the Morse complex, and (5) map the segmentation labels from the scalar field domain to the input point clouds. Steps (1), (4), and (5) leverage existing modules in both TTK as well as standard methods that exist in the ParaView [1]. Steps (2) and (3) are developed as standard TTK modules that we also enable as ParaView plugins for experimentation.

4 Computing Scalar Fields from Point Clouds

As previously mentioned, ScalarFieldFromPointCloud takes a point cloud as an input and constructs a triangulated regular grid surrounding the input point cloud. The grid size as well as boundary padding may be specified by the user. The grid is the domain of the scalar field that ScalarFieldFromPointCloud outputs. As with Chazal et al. [9] we experimented with two difference options for calculating the scalar values on each grid point: a kernel density estimation (KDE) or a distance field.

For the distance field, the scalar values on the regular grid can simply be taken to be the distance to the closest point in the point cloud. In effect, this will yield very small values for points with nearby neighbors, whereas the KDE will yield the largest values in regions of high density. Subsequently, in this document we only report on experiments using density.

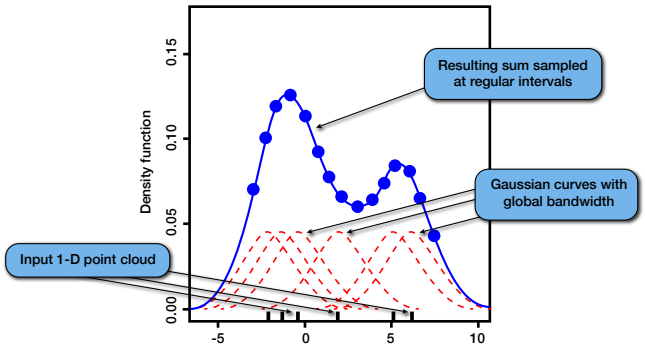

Fig. 1 Gaussian kernel density estimate. A Gaussian of a certain bandwidth is centered at each point. The Gaussians are added and the purple curve is the resulting probability distribution that is estimated from the samples. Image taken from Wikipedia

The scalar values for the KDE are calculated as follows: We construct identical Gaussian distributions centered at each point in the point cloud. A continuous scalar

function $f : \mathbb{R}^d \to \mathbb{R}$ defined on the ambient space can be constructed by taking the sum of all of the Gaussian distributions. The standard deviation or width of the Gaussians is denoted as the *bandwidth*, which is a parameter the user may choose. The value of the output scalar field for a particular vertex in the regular grid is determined to be the value of f at that point. See Figure 1 for a graphic explanation.

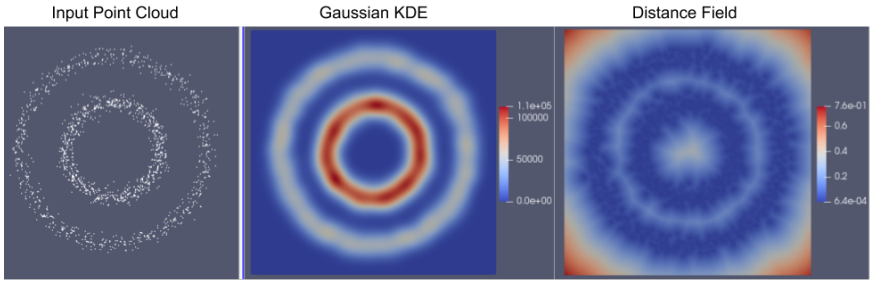

Fig. 2 The input to ScalarFieldFromPointCloud on the left, the output scalar field in the middle when using the Gaussian KDE option, and the output scalar field on the right when using the distance field option. For the KDE, the colour map is red at locations with a high density of points, and blue in sparse regions; in the distance field, the colour map is red far away from the points and blue near the points.

4.1 Parameter Setting

We also implemented a feature that automatically estimates what the bandwidth should be. For each point in the point cloud, we calculate the distance to the $k-$th closest neighbor. The parameter k is an integer, and we estimate it as follows

$$k = (0.587 \cdot n^{4/5})^{1/d}$$

and then rounding k up. n is the number of points in the point cloud, d is the dimension of the data (currently we allow $d = 2$ and $d = 3$). This method is modified from Equation 19 of [23] to include the $1/d$ scaling to the $n^{4/5}$ term. Finally, the mean distance to the $k-$th closest neighbor for every point in the point cloud is taken to obtain the final bandwidth which is used. This method of bandwidth selection can be used in ScalarFieldFromPointCloud module by selecting the "Automatic Bandwidth for KDE" option in ParaView.

We also considered a method that uses adaptively-sized bandwidths for the Gaussian kernels, proportional to the k-th nearest neighbor [7]. When exploring this variation, we found that the least persistent clusters would have an even smaller persistence with this algorithm, making it likely for PersistenceSimplification module to discard them as noise.

5 Persistence-Based Clustering

We briefly review the key concepts we used in persistent homology, for a full introduction we recommend Edelsbrunner and Harer [11]. Given a scalar field, $f : \mathbb{R}^d \to \mathbb{R}$, persistent homology can be used to study the evolution of the sublevel sets $f^{-1}(-\infty, a]$ for every real value a. Intuitively, as one sweeps through increasing values of a, the connectivity of the sublevel sets will change, and persistent homology helps to gauge both when and what types of changes occur. Specifically, one can observe when new components are created and destroyed, and how long (in terms of values for a) they exist. Such events will occur at values for a which correspond to critical points of f. These events can be grouped into what are commonly referred to as *persistence pairs* that correspond to a pair of critical points. For a persistence pair c_i and c_j such that $f(c_i) < f(c_j)$, we define the *persistence* as $f(c_j) - f(c_i)$, which corresponds to the span of function values for which the corresponding feature was present. The *persistence diagram* embeds these pairs into a useful tool for data analysis, as it describes all such pairs and conveniently arranges them so that pairs with higher persistence are made prominent.

Given the persistent diagram, a common task to filtering such features is to select a persistence threshold and retain all pairs whose persistence is above the threshold. The second module we implemented, PersistenceSimplification, performs this form of persistence simplification on an input scalar field. In our setting, simplification prepares the scalar field for topological segmentation and ultimately, clustering the point cloud. Tools for persistence simplification already exist in TTK, but our module encapsulates this process so as to make it easier for an end user. Specifically, our new module relies components from two existing modules: FTMTreePP [17] (which computes a merge tree that can be used to construct the persistence threshold for each persistence pair) and TopologicalSimplification [28] (which is used to compute a new, simplified scalar field that preserves the persistence pairs that are above threshold).

Previously, in TTK, a user would have to separately compute the persistence diagram, use thresholds to select a set of persistence pairs to preserve, and then run TopologicalSimplification to produce a simplified field. Our new module combines these features into a single module. First, the scalar field is given to the FTMTreePP object, and the persistence pairs are computed. The pairs are then sorted based on their persistence, and then a persistence threshold is decided based on user selected parameters (including automatic selection). Finally, TopologicalSimplification is called using the pairs with sufficiently high persistence as constraints on the input scalar field. Our module then computes this simplified field and returns it as output. Thus, unlike the separate components that exist in TTK, this module starts with a scalar field and produces a new, simplified one with no intermediate steps, greatly simplifying the process of simplification.

5.1 User Options

The PersistenceSimplification module makes this process easy by not exposing the persistence diagram to the user nor requiring manual pair selection. Instead, the user is given the option to have PersistenceSimplification choose which points are sufficiently persistent automatically as described below. Nevertheless, we consider a number of use cases for the PersistenceSimplification module. Although we are focused on clustering applications, we have made the module as versatile as possible. The user can override the automatic threshold by interacting with the GUI in ParaView to choose various threshold options:

- The user has the option to manually threshold which persistence pairs (min-saddle, max-saddle, or both) are used to simplify the scalar field based on a known persistence threshold.
- Alternatively, if a threshold is not known, the user can instead threshold based on the number of pairs they would like to simplify with. The most persistent points will be chosen. This is conceptually similar to setting the number of clusters, k, in k-means.

In addition the user can choose to simplify the scalar field using the maximum-saddle pairs and the minimum-saddle pairs separately, or both together. For density-based clustering, we expect maxima to delineate centroids for clustering, but this module provides more flexibility capabilities to be used in other settings too.

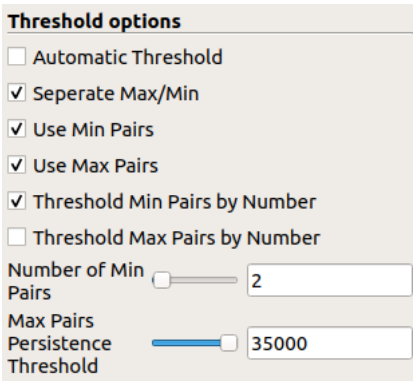

Fig. 3 GUI for PersistenceSimplification. By checking “Separate Max/Min", the user may select minimum-saddle pairs and maximum-saddle pairs in different ways. By ticking “Use Min/Max Pairs" the module will choose to retain minimum/maximum-saddle pairs. Since “Threshold Min Pairs by Number" is specified, we can choose the number of minimum-saddle pairs to retain. Alternatively, since “Threshold Max Pairs by Number" is not specified, we retain maximum-saddle pairs by thresholding persistence instead.

5.2 Automatic Parameter Setting

We developed our automatic threshold mechanism based on the concept of finding large jumps in persistence that separate topological noise (with relatively low persistence) from topological signal (with relatively high persistence). We define such jumps by first sorting all pairs by their index and looking for locations where the tangent of the curve does a poor job of predicting the persistence of the next highest pair, relative to how much this curve increases overall. For an illustration, see Figure 4.

Specifically, let p_n be the persistence of the most persistent critical pair in our dataset. To identify gaps automatically, we examine the persistence of each pair in increasing order. Let $a = 0.2$ and $b = 0.025$ be two fitting parameters (a describes the exponential growth in persistence pairs and b weights relative to the maximum). For a critical pair with persistence p_0, we ask if (1) the next most persistent pair has persistence greater than $(1+a)p_0 + bp_n$ and if (2) the pair after that has persistence greater than $(1 + 2a)p_0 + 2bp_n$. If the answer to both questions is yes, then the persistence threshold is determined to be $(1 + a)p_0 + bp_n$. If the answer to either question is no, then we remove the initial pair and begin the analysis again on the next pair, until a persistence threshold is decided.

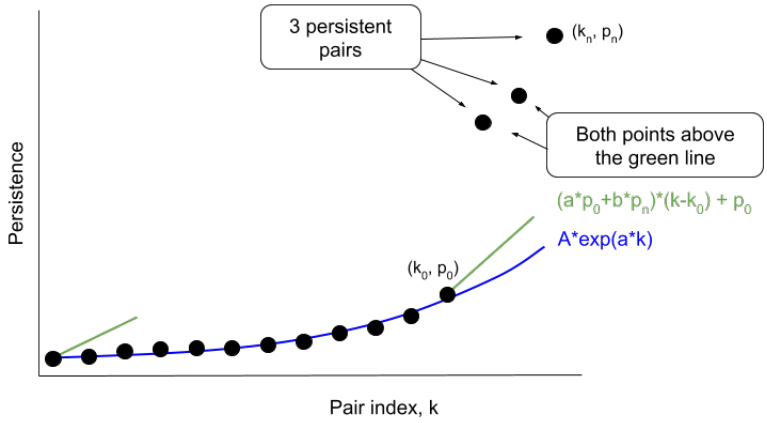

Fig. 4 Automatic persistence thresholding. Each black point is a critical point pair. We sort the pairs by persistence. We assume the noise assumes an exponential shape. $(a \cdot p_0 + b \cdot p_n)(k - k_0) + p_0$ is the tangent line to the exponential plus some slope $b \cdot p_n$, where p_n is the persistence of the most persistent critical pair. The algorithm essentially looks point by point to see if the next point is above this green line. If the next two points lie above the line, then the all further points are persistent enough.

This progressive approach is based on the observation that that noisy persistent pairs fit an exponential function quite well. Thus, the two parameters we select model this process. Specifically, a is chosen to be the parameter that describes the exponential growth of persistence in Figure 4. The parameter b is chosen as a value

that captures discontinuities in these curves. Ideally, one would directly estimate a and b using data fitting, or let the user vary them. As heuristics, we found they performed quite well for our data even with fixed parameters.

6 Experimental Results

We test the effectiveness of our automatic parameter selection technique and analyze where it fails. For the following experiments, we only cluster with parameters selected through our automatic selection technique. We used a collection of different datasets and benchmarks. For evaluating automatic feature detection efficacy, we used the Ultsch's Fundamental Clustering Problems Suite (FCPS) [29][1] and Fränti and Sieranoja's k-means clustering testing suite [16][2]. For comparisons against other clustering methods, we evaluated on the scikitlearn clustering datasets [25][3].

6.1 Automatic Feature Detection

Results for our clustering method with automatic parameter selection are shown in Figures 5 and 6. On the two k-means datasets, our method was quite effective at separating individual clusters of both different sizes and shapes.

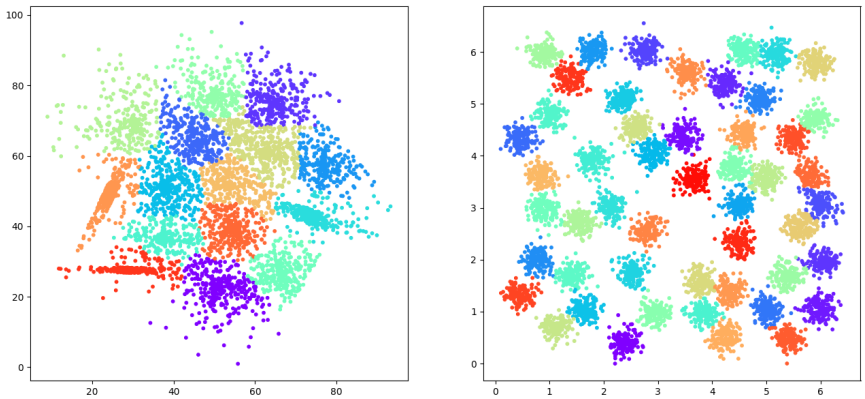

Fig. 5 Our automatic persistence clustering, applied to two datasets from the k-means clustering suite. Left: dataset S4, Right: dataset A3. Each colour indicates a different cluster, as labeled by our algorithm.

[1] Downloaded from https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/data

[2] Downloaded from http://cs.joensuu.fi/sipu/datasets/

[3] Extracted directly from `sklearn.datasets`

For the FCPS datasets we chose, our technique encountered additional challenges that stem from the automatic selection of persistence thresholds. While generally reasonable, we found our thresholds to produce a few additional clusters than necessary, particularly for the TwoSuns (we end up with 4 instead of 2 clusters) and Lsun (we end up with 5 instead of 3 clusters). Note that if we manually set the number of clusters to the appropriate number, or method produced correct clusterings for both. The Wingnut dataset proved a bit challenging when the data sparsity made it challenging to separate the top from the bottom, as shown in the few points that are not separated well. We discuss this issue further in Section 6.2.

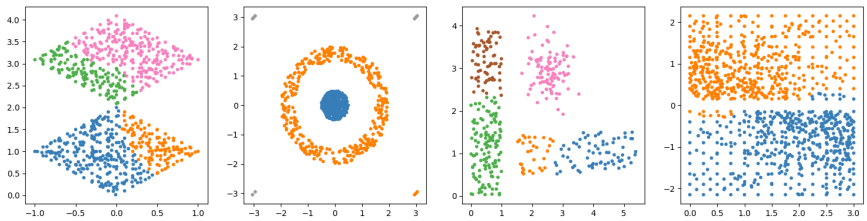

Fig. 6 Our automatic persistence clustering, applied to four datasets from FCPS. From left-to-right, we evaluated the TwoSuns, Target, Lsun, and Wingnut datasets. Each colour indicates a different cluster, as labeled by our algorithm.

6.2 Comparison Against Other Clustering Methods

We also evaluated our clustering method against a variety of other techniques[4], as shown in Figure 7. Note that the data generation process in scikitlearn involves randomness, which explains the minor discrepancies between Figure 7 and figures on their website.

In the datasets, we are given the ground truth for what the clusters should be. This means we can apply the Fowlkes-Mallows score for clustering [15]. The Fowlkes-Mallow score is a goodness of clustering score from 0 to 1, with 1 being a perfect clustering and smaller values being bad clusterings. We have labeled each method and each dataset with a Fowlkes-Mallows score and summed up the scores for all datasets and each algorithm in Table 1.

All algorithms in Figure 7 except ours which is outlined in red and denoted “Persistence" needs to have parameters specified. For these other algorithms, we used the same parameters as in scikit-learn.org/stable/modules/clustering.html, which were picked to perform well on these datasets. The best algorithms according to the

[4] See scikit-learn.org/stable/modules/clustering.html for information on the clustering methods we compared to

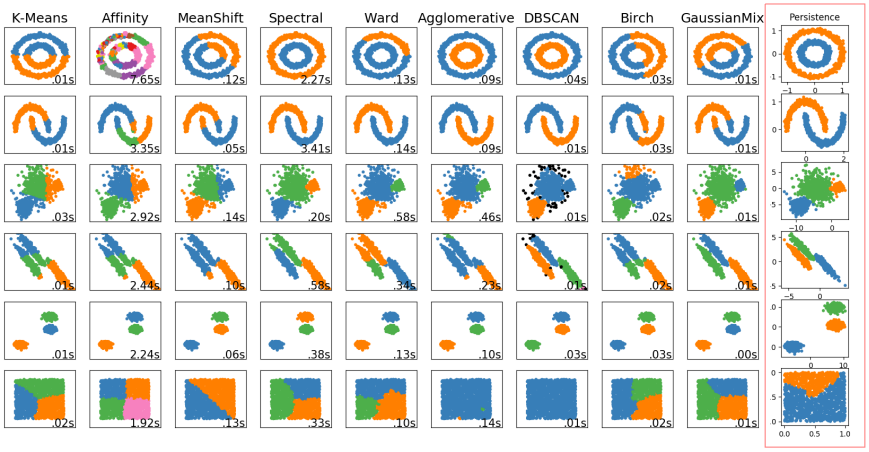

Fig. 7 A comparison of clustering methods. Each colour indicates a different labeling according to the algorithm. Our method, “Persistence", is outlined in red.

	K-Means	Affinity	Mean Shift	Spectral	Ward	Agglom.	DBSCAN	Birch	Gauss. Mix	**Persistence**
Circles	0.50	0.36	0.42	1.0	0.66	1.0	1.0	0.51	0.50	**1.0**
Moons	0.74	0.67	0.77	1.0	1.0	1.0	1.0	0.57	0.75	**1.0**
Varied Density	0.87	0.88	0.90	0.96	0.95	0.95	0.75	0.74	**0.98**	0.96
Anisotropic	0.73	0.74	0.75	0.97	0.79	0.69	0.98	0.72	1.0	**1.0**
Blobs	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**1.0**
No Structure	0.58	0.50	0.71	0.58	0.59	0.99	**1.0**	0.59	0.58	0.77
Sum of Scores	*4.42/6.0*	*4.15/6.0*	*4.55/6.0*	*5.51/6.0*	*4.99/6.0*	*5.63/6.0*	*5.73/6.0*	*4.13/6.0*	*4.81/6.0*	***5.73/6.0***

Table 1 The Fowlkes-Mallows score, a metric for goodness of clustering, for the corresponding algorithm (column) and dataset (row). The metric ranges from 0 to 1, with 0 being a bad clustering and 1 being a perfect clustering. The datasets correspond in order to the datasets in Figure 7.

sum of scores is Persistence and DBSCAN. DBSCAN, however, does not classify all points as seen by the black outliers in the figure.

In Figure 7 we see that the only data set for which our automatic clustering method fails drastically is the last data set on the bottom row, which has only 1 cluster. The data itself is not ideal for our method when compared to the other clusters in Figure 7. Particularly, it suffers from the fact that it is sampled from a uniform distribution on a square and thus there are no major topological variations in the resulting density field. This behavior is also byproduct of estimating density by summed Gaussians, which would naturally weight dense areas higher than sparse areas. As a result, the one singular cluster has a large area and is ultimately undersampled.

These two conditions (uniform sampling and summing Gaussians) result in “patchy-ness” in the density estimation. Figure 8 illustrates the clustering procedure for the final data set. Despite the relatively low quality density estimation, our automatic threshold for persistence almost worked. In this test, a in the automatic persistence algorithm was set to 0.2. In reality, by fitting an exponential curve to

the data, we get that a should be 0.33 instead. This leads to a more appropriate clustering.

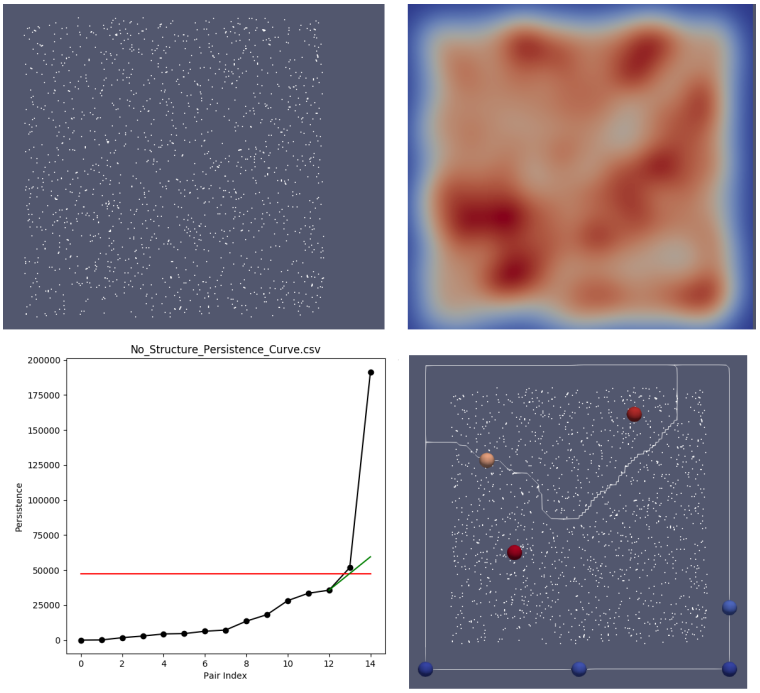

Fig. 8 Clustering of a data set which is sampled from a uniform distribution on a square. Top Left - Original data set. Top Right - Kernel density estimation; red is high, blue is low. Bottom Left - Automatic persistence thresholding, with the red line representing the persistence at which we threshold. Each black dot represents a maximum-saddle pair. We order the pairs by persistence. The point just above the red threshold lies above the green line, hence the algorithm terminates. Bottom Right - Segmented domain, with each segment representing a cluster. The red spheres are maxima, blue minima, and yellow saddle.

It turns out that the TwoSuns datasets in Figure 6 fails to automatically cluster for the same reasons as above. The same patchiness in the scalar field results leading to an incorrect thresholding. Interestingly, fitting an exponential curve to the persistence curve yields $a = 0.53$, which we found would indeed lead to the correct two clusters.

It seems for both data sets that the bandwidth is too small for the density estimation. However, we noticed that the automatic bandwidth selection on the data sets in Figure 5 produced bandwidths that could not be increased without yielding incorrect clusterings. This observation points out some of the challenges with setting parameters, but in general our automatic method worked well as a heuristic for the data sets we experimented with.

7 Discussion

Our work demonstrates the effectiveness of using scalar field topology for clustering point clouds. Moreover, we also discussed implementing this technique in TTK, creating new modules to both ease the process for a user as well as designing user heuristics for automatically specifying parameters. In general, clustering is often an extremely useful first step in the data analysis pipeline, but as the problem is ill-constrained there is no single solution that works in all settings.

Our method allows for both distance fields and KDE to building the scalar field from point clouds. While KDE is a natural choice, its use does restrict the clustering approach to be sensitive to the density of the input point cloud, similar to DBSCAN. Of course, there is further control in terms of the persistence and other user-defined parameters. In some contexts, this control is desirable, but we leave it to future work to explore what other scalar fields might be better suited in different contexts.

On the computational side, we made an up front choice to limit our approach to working with point clouds in low-dimensions. This report demonstrates the efficacy on mainly two-dimensional datasets, but we have also experimented with both three-dimensional point clouds and point clouds sampled from surfaces embedded in three dimensions. Our results do extend, although our method for automatically specifying parameters in both modules does have a dependence on the underlying dimension of the data. We leave it to future work to specify these parameters (k, a, and b) in terms of the dimension of the data. For higher dimensional data, utilizing a similar pipeline is also work in progress, but it has significant challenges with visualizing the resulting clustering. One method that shows promise is to first project the input data to a lower dimensional manifold, as is frequently done in other settings.

Since we sample our scalar field on a triangulated grid, the computational workload will ultimately be dependent on the resolution of this grid. TTK is optimized for fast access on triangulations, which is advantageous compared to other alternative simplicial complexes we could use such as Rips complexes. While we require a sampling density sufficient for capturing the separation between clusters, it would be interesting to consider specifying a non-uniform triangulation that adapts to the data so as to reduce the computational burden. It is future work as how best to balance clustering accuracy with computational cost.

Finally, our implemented modules are available for download, as well as the ParaView state files used to produce the results prsesented in this paper [5].

Acknowledgements This work is partially supported by the European Commission grant ERC-2019-COG *"TORI"* (ref. 863464). This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number(s) DE-SC-0019039.

[5] https://github.com/bluenote-1577/ttk

References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large-data visualization. The Visualization Handbook pp. 717–731 (2005)
2. Beksi, W.J., Papanikolopoulos, N.: 3d point cloud segmentation using topological persistence. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 5046–5051. IEEE (2016)
3. Beksi, W.J., Papanikolopoulos, N.: Signature of topologically persistent points for 3d point cloud description. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1–6. IEEE (2018)
4. Beksi, W.J., Papanikolopoulos, N.: A topology-based descriptor for 3d point cloud modeling: Theory and experiments. Image and Vision Computing **88**, 84–95 (2019)
5. Bertrand, G.: On topological watersheds. Journal of Mathematical Imaging and Vision **22**(2-3), 217–230 (2005)
6. Beucher, S.: Watersheds of functions and picture segmentation. In: ICASSP'82. IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 7, pp. 1928–1931. IEEE (1982)
7. Breiman, L., Meisel, W., Purcell, E.: Variable kernel estimates of multivariate densities. Technometrics **19**(2), 135–144 (1977)
8. Chazal, F., Guibas, L.J., Oudot, S.Y., Skraba, P.: Scalar field analysis over point cloud data. Discrete & Computational Geometry **46**(4), 743 (2011)
9. Chazal, F., Guibas, L.J., Oudot, S.Y., Skraba, P.: Persistence-based clustering in riemannian manifolds. Journal of the ACM (JACM) **60**(6), 41 (2013)
10. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. IEEE Transactions on Pattern Analysis & Machine Intelligence **24**(5), 603–619 (2002)
11. Edelsbrunner, H., Harer, J.: Computational Topology: An Introduction. American Mathematical Society (2009)
12. Edelsbrunner, H., Harer, J., Natarajan, V., Pascucci, V.: Morse-smale complexes for piecewise linear 3-manifolds. In: Proceedings of the nineteenth annual symposium on Computational geometry, pp. 361–370. ACM (2003)
13. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. In: Proceedings 41st Annual Symposium on Foundations of Computer Science, pp. 454–463. IEEE (2000)
14. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of Knowledge Discovery and Data Mining, pp. 226–231 (1996)
15. Fowlkes, E.B., Mallows, C.L.: A method for comparing two hierarchical clusterings. Journal of the American statistical association **78**(383), 553–569 (1983)
16. Fränti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets. Applied Intelligence **48**(12), 4743–4759 (2018)
17. Gueunet, C., Fortin, P., Jomier, J., Tierny, J.: Task-based augmented merge trees with fibonacci heaps. In: 2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV), pp. 6–15. IEEE (2017)
18. Koontz, W.L.G., Narendra, P.M., Fukunaga, K.: A graph-theoretic approach to nonparametric cluster analysis. IEEE Trans. Computers **25**(9), 936–944 (1976)
19. Lloyd, S.: Least squares quantization in PCM. IEEE transactions on information theory **28**(2), 129–137 (1982)
20. Moon, C., Giansiracusa, N., Lazar, N.A.: Persistence terrace for topological inference of point cloud data. Journal of Computational and Graphical Statistics **27**(3), 576–586 (2018)
21. Oesterling, P., Heine, C., Janicke, H., Scheuermann, G., Heyer, G.: Visualization of high-dimensional point clouds using their density distribution's topology. IEEE Transactions on Visualization and Computer Graphics **17**(11), 1547–1559 (2011)
22. Oesterling, P., Heine, C., Weber, G.H., Scheuermann, G.: Visualizing nd point clouds as topological landscape profiles to guide local data analysis. IEEE transactions on visualization and computer graphics **19**(3), 514–526 (2012)

23. Orava, J.: K-nearest neighbour kernel density estimation, the choice of optimal k. Tatra Mountains Mathematical Publications **50**(1), 39–50 (2011)
24. Sander, J., Ester, M., Kriegel, H.P., Xu, X.: Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data mining and knowledge discovery **2**(2), 169–194 (1998)
25. Scikitlearn: Clustering. https://scikit-learn.org/stable/modules/clustering.html (2019). [Online; accessed 02-January-2019]
26. Thom, R.: Sur une partition en cellules associée à une fonction sur une variété. COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L ACADEMIE DES SCIENCES **228**(12), 973–975 (1949)
27. Tierny, J., Favelier, G., Levine, J.A., Gueunet, C., Michaux, M.: The Topology ToolKit. IEEE Trans. on Visualization and Computer Graphics (Special Issue IEEE VIS 2017: SciVis) **24**(1), 832–842 (2018)
28. Tierny, J., Pascucci, V.: Generalized topological simplification of scalar fields on surfaces. IEEE transactions on visualization and computer graphics **18**(12), 2005–2013 (2012)
29. Ultsch, A.: Clustering with SOM: U*C. Proc. Workshop on Self-Organizing Maps pp. 75–82 (2005)
30. Von Luxburg, U.: A tutorial on spectral clustering. Statistics and computing **17**(4), 395–416 (2007)