DBSpan: Density-Based Spanner for Clustering Complex Data, With an Application to Persistence Diagrams*

Brittany Terese Fasy†, David L. Millman†, Elliott Pryor†, and Nathan Stouffer‡

Abstract. Since its introduction in the mid-1990s, DBSCAN has become one of the most widely used clustering algorithms. However, one of the steps in DBSCAN is to perform a range query, a task that is difficult in many spaces, including the space of persistence diagrams. In this paper, we introduce a spanner into the DBSCAN algorithm to facilitate range queries in such spaces. We provide a proof-of-concept implementation, and study time and clustering performance for two data sets of persistence diagrams.

Key words. clustering, distance approximation, DBSCAN, persistent homology, spanners

AMS subject classifications. 55N31 Persistent homology and applications, topological data analysis, 62H30 Classification and discrimination; cluster analysis (statistical aspects), 62R40 Topological data analysis, and 68T09 Computational aspects of data analysis and big data.

1. Introduction.

A common task in data analysis is clustering. Since its introduction in the mid-1990s, DBSCAN [16] has become one of the most widely used clustering algorithms. Range queries—finding all data points that satisfy a certian (usually geometric) property, such as all points within distance ϵ of a point x—are a central step in the DBSCAN algorithm. However, computing range queries and pairwise distance matrices is a task that is often cumbersome in high-dimensions and in non-Euclidean spaces, such as spaces of: persistence diagrams, Reeb graphs, point clouds, and curves in space.

In this paper, we are interested in studying the clustering of persistence diagrams. The space of persistence diagrams under the bottleneck distance has infinite doubling dimension [17], making it not suitable for many clustering algorithms. To get around this issue, approximating distances (and defining new distances) for the task of clustering persistence diagrams has been the focus of several recent papers in topological data analysis (TDA) [8, 13, 14, 17, 32]. We present DBSpan, a modified version of the DBSCAN algorithm, that replaces the range query with an approximate range query by using a spanner technique introduced by Kerber and Nigmetov [25] for approximating expensive distances. When used to cluster persistence diagrams, we demonstrate performance improvements over the standard DBSCAN algorithm.

This paper begins, in Section 2, by introducing preliminaries needed to understand our algorithm, then presents the algorithm itself in Section 3. Furthermore, we provide a proof-of-concept implementation in Python[1] and demonstrate the utility of the algorithm with experimental results in the space of persistence diagrams under the bottleneck distance in Section 4.

2. Preliminaries.

In this section, we introduce notation and definitions needed to understand the contributions of this short paper. Relevent references are provided where the interested reader can find more details on these preliminaries.

Notation. S is a finite metric space, with corresponding distance metric $d\colon S \times S \to \mathbb{R}$. For a point $x \in S$ and radius $\epsilon > 0$, we use $N_\epsilon(x)$ to denote the set of points in $S \setminus \{x\}$ within the closed ϵ-neighborhood or ϵ-neighborhood of x; that is, $N_\epsilon(x) := \{y \neq x \in S \mid d(x,y) \leq \epsilon\}$.

Clustering (and Other Tasks) with Persistence Diagrams. While the algorithm presented in this paper is applicable to many types of data, we are particularly motivated by using persistence diagrams to compare data, and the difficulties that arise in clustering data using persistence diagrams.

*Submitted to the editors 8 March 2022.

Funding: BTF was partially supported by NSF grants DMS 1664858, DMS 1854336 and CCF 2046730.

†Montana State University, Bozeman, MT (brittany.fasy@montana.edu, david.millman@montana.edu, elliott.pryor@student.montana.edu).

‡onXmaps, Inc., Bozeman, MT

[1]The implementation is publicly available via MIT license at: https://github.com/compTAG/dbspan

Geometrically, a persistence diagram is a multiset of points in the extended plane (the extended plan $\overline{\mathbb{R}}^2$ allows infinite coordinates). To compare two persistence diagrams D_1 and D_2, we use the bottleneck distance between them, denoted $d_\infty(D_1, D_2)$. We defer formal definitions of persistence diagrams and distances between them to [15].

The difficulty of dealing with persistence diagrams in tasks such as clustering or range queries largely boils down to two issues. First, the computation of the distance itself is expensive. To compute $d_\infty(D_1, D_2)$, one must minimize over the set of all matchings between point sets D_1 and D_2. While heuristic improvements on computation have been made to prune this search space in practice [23], it is still a time-consuming task. Thus, for large data sets, computing all pairwise distances is impractical. Second, the doubling dimension (that is, the log of the number of balls of radius $\frac{r}{2}$ needed to cover a ball of radius r; see [11]) of the space of persistence diagrams is infinite [17, 24, 25]. Often, algorithms for computing nearest neighbors or range queries assume low doubling dimension. And, even in spaces of constant doubling dimension d, to use cover trees for computing nearest neighbors is $O(d^{12} \log n)$ for a data set with n points [2].

DBSCAN. The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithm was first introduced in [16] and has seen many improvements and implementations [4, 6, 7, 20, 27]. (Most of the implementations are optimized for datasets lying in $\mathbb{R}^d$). The intuition of this algorithm is that, given $\epsilon > 0$, we: (1) for each point x, compute $N_\epsilon(x)$, the set of all points whose distance is at most ϵ from x; (2) construct a neighborhood graph on these neighborhoods, restricting to the "dense" points (i.e., the vertices of this graph are the points whose ϵ-neigborhoods meet a minimum threshold of neighbors, and edges exist between vertices if their distance is at most 2ϵ). The output of DBSCAN is a labeling of each vertex with its cluster, or as 'noise' (that is, non-dense points whose ϵ-neighborhoods do not contain any dense point). The vertices of the connected components of this graph (along with additional points "at the boundary" of these clusters) are the clusters that DBSCAN computes.

The central step of DBSCAN is the computation of the ϵ-neighborhoods. This step is an example of a *range query*: given a shape in the domain (such as a disc in $\mathbb{R}^2$), find all data points that are contained in that shape. In $\mathbb{R}^2$, a data structure can be created to support range queries of ℓ_∞ balls with $O(n \log n)$ precomputation such that a range query whose range has k points will take $O(\log n + k)$ distance calculations [38]. Assuming the ϵ-neighborhoods in DBSCAN have constant size, we observe that the computation of all ϵ neighborhoods takes $O(n \log n)$. In spaces with low doubling dimension, rather than calculating the ϵ-neighborhoods exactly, approximating these using approximate nearest neighbor graphs can be used. For example, [39] replaces the range queries in DBSCAN with locality sensitive hashing. And, a paper posted to ArXiv in 2020 [9] uses approximate k-nn graphs using random projections to provide a speedy approximation of the ϵ-neighborhoods in DBSCAN. However, in spaces with large doubling dimension, a single range query (in practice) resorts to scanning through all data points; hence, the computation of the ϵ-neighborhoods requires $O(n^2)$ distance calculations.

DBSCAN is familiar to the TDA community, as papers have been published using DBSCAN. Within this context, DBSCAN has been used to cluster points within a persistence diagram [26, 29], as a comparison against TDA-based techniques for clustering [12, 21, 28, 31], and as a clustering subroutine in a TDA pipeline [3]. In the latter, a pairwise distance matrix is required as input, which makes using DBSCAN prohibitive for large data sets of persistence diagrams.

Spanners. In DBSCAN, when spatial range queries or other ways to speed-up the calculation of ϵ-neighborhoods are not possible, we rely on the pairwise distance matrix; that is, we need to store (or at least compute) the distance between every pair of input data. Thinking of this matrix as a large graph (a vertex for each datum and an edge for each pairwise distance), spanners help to simplify the graph, while maintaining approximate distances. In particular, a $(1+\delta)$-spanner of a discrete metric space (S, d) is a weighted graph $G = (S, E, \omega)$ such that for all pairs $p, q \in S$,

$$d(p, q) \leq d_G(p, q) \leq (1 + \delta) d(p, q), \tag{2.1}$$

where $d_G(p, q)$ is the length of the shortest path between p and q in G. Spanners were originally introduced to simplify the pairwise distance matrix for points in the plane [10], but have since been used in other metric spaces as well [1, 19, 25]. However, the underlying assumption is often that distances are fast to compute and that we can have access to the entire pairwise distance matrix to compute the spanner.

On the other hand, Kerber and Nigmetov [25] studied spanners in metric spaces where the distances are expensive to compute, such as in the space of persistence diagrams. Given a parameter δ, they provide algorithms to compute a $(1 + \delta)$-spanner for a pairwise distance graph. The spanners do not require computing all $|S|^2$ distances, and in practice, tend to be linear in size. While we defer the details of the algorithm to [25], we note that their algorithm is an iterative method that maintains, for each $p, q \in S$, a lower and an upper bound, $a(p, q)$ and $b(p, q)$ for the distance $d(p, q)$. When selecting which distance to compute next (that is, which edge to add to the spanner), they offer two heuristics, BlindGreedy and BlindRandom. Both of these heuristics select which edge based on the ratio $\frac{b(p,q)}{a(p,q)}$, with BlindGreedy selecting the edge that maximizes that ratio and BlindRandom selecting randomly among edges whose ratio is at least $1 + \delta$.

3. Algorithm.

We present DBSpan, an approximation of DBSCAN for metric spaces where the distances are expensive to compute. The algorithm, given in Algorithm 3.1, mirrors DBSCAN, but differs in two key places:

1. We replace the ϵ-neighborhood $N_\epsilon(p)$ with an approximate nearest-neighbor ball $\widetilde{N}^\delta_\epsilon(p)$ computed using distances in the spanner graph from [25] using the BlindRandom heuristic.
2. Rather than computing the ϵ-neighborhoods on the fly, the approximate ϵ-neighborhoods from the spanner are precomputed in Line 3.

Note that the ϵ-neighborhoods for DBSCAN could also be pre-computed; however, all $\Theta(|S|^2)$ distances between input data will still need to be computed first. By using the spanner from [25], DBSpan reduces the number of pairwise distances that need to be computed.

Implementation. Along with this paper, we are releasing an open-source Python implementation (see link in Footnote 1). This code uses Scikit-TDA [34] for computing persistence diagrams on Rips filtrations and computing the bottleneck distance between persistence diagrams. For the most part, the Python code mirrors Algorithm 3.1. We note that, for clarity of exposition and for adhering to variable naming practices in Python, several variables have different names in the pseudocode in this paper and the Python code: S in the pseudocode is called `data` in the Python code, ϵ is called `eps`, m is called `min_samples`, and Q is called `queue`. In addition, the Python code has one performance improvement: rather than pushing p onto the queue in Line 13, p is directly processed and Q is initialized to $N_\epsilon(p)$. This saves one range query.

Relationship between Clusters Found by DBSCAN and DBSpan. By Equation (2.1), several relationships between the clusters found by DBSCAN and DBSpan arise that allow us to ensure that 'nearby' points are clustered together and 'far away' points are not. In particular:

1. If $p, q \in S$ such that DBSpan(S, ϵ, m, δ) clusters them into the same cluster and their ϵ-neighborhoods have at least m points, then p and q will be in the same cluster using DBSCAN(S, ϵ, M) for all $M \geq m$.
2. If $p, q \in S$ such that DBSpan(S, ϵ, m, δ) clusters them into different clusters, C_p and C_q, if there are no outliers, and if the minimum inter-cluster distance between all pairs of clusters is greater than $(1+\delta)\epsilon$, then p and q will not be in the same cluster using DBSCAN(S, ϵ, m).

These statements follow from the fact that the approximate ϵ-neighborhoods are computed from an $(1 + \delta)$-spanner over S.

4. Experimental Results.

Experimental Methods. In the experiments, we compare clustering using the scikit-learn [30] implementation of DBSCAN to the clustering obtained using DBSpan (Algorithm 3.1). To validate the clustering, we treat the clusters from DBSCAN as the *ground truth* and compute the adjusted Rand index (ARI) [37] for DBSpan. Similar to the standard Rand index [33], if the clusters match

Algorithm 3.1 DBSpan(S, ϵ, m, δ)

Input: Dataset S; neighborhood radius $\epsilon \geq 0$; core size $m \geq 0$; and spanner parameter δ
Output: A cluster label is attributed to each point in S

For all $p \in S$, initialize $p.cluster = -1$ {All points are considered unlabeled initially}
Compute spanner of S
For all $p \in S$, compute $\widetilde{N}_\epsilon^\delta(p)$ using the spanner
$c = 0$ {number of clusters; also current cluster number}
for $p \in S$ **do**
  if p is labeled **then**
    continue
  else if $|\widetilde{N}_\epsilon^\delta(p)| < m$ **then**
    $p.cluster \leftarrow 0$ {p is labeled as noise}
    continue
  end if
  $c \leftarrow c + 1$
  $Q.push(p)$ {initialize queue of points to be added to current cluster}
  while Q is not empty **do**
    $q \leftarrow Q.pop()$
    if $q.cluster \geq 1$ **then**
      continue
    end if
    $q.cluster \leftarrow c$ {q is now considered labeled}
    if $|\widetilde{N}_\epsilon^\delta(q)| \geq m$ **then**
      Add $N_\epsilon(q)$ to Q
    end if
  end while
end for
return

exactly between DBSCAN and DBSpan (up to relabeling), then the adjusted index is one. The more disagreement between the two clusterings, the closer the index is to zero.

Experiment 1: Shape Dataset. In this experiment, we study how in the shape dataset, varying the approxiation of the true distance, δ, changes the quality of the clustering (as measured by ARI).

We consider a simple data set of shapes in $\mathbb{R}^4$. Using TaDAsets from Scikit-TDA [34], we create 74 shapes from three different classes: 30 torii, 30 spheres, and 14 Swiss rolls. For each shape, we sample 100 points form the shape, apply Gaussian noise with $\sigma = .1$, and compute the persistence diagram corresponding to the Vietoris-Rips filtration. Then, the 1d diagram is used as input into DBSCAN and DBSpan with common parameters $\epsilon = 0.3$ and $m = 15$.

In Table 1, we see the result of varying the spanner parameter δ and comparing the speed and accuracy of DBSpan. The first column, δ, is the approximation for the spanner. The second column is the ARI of the DBSpan output as compared to the DBSCAN ground truth. The third column is the number of edges in the resulting spanner. The fourth column is the proportion of edges in the spanner to the complete graph. Note that the complete graph has 2701 edges. The fifth column is the time (in seconds) for running DBSpan. The sixth column is the speedup of DBSpan. Note that the DBSCAN took 74.99s to complete.

Observe that for $\delta = \{.1, 1\}$, DBSpan and DBSCAN have a ARI index of 1, which means that they produced the same output. Unfortunately, when $\delta = .1$, DBSpan is slightly slower than DBSCAN. In this case, even though we only compute 83% of all distances, the cost of determining which distances can be omitted overwhelms the gains (as could be expected). For this problem, $\delta = 10$ strikes a nice balance between speed and accuracy as Rand index is .98 with a 4x speedup.

Table 1: Performance metrics comparing DBSCAN and varying δ values for DBSpan. For this experiment, DBSCAN ran in 74.99 seconds and the complete graph contains 2701 edges. We compare accuracy between the clusterings with the adjusted Rand index (ARI) in column 2 and measure DBSpan time in column 5 in seconds.

δ	ARI	Num Edges	% Possible Edges	DBSpan time	Speedup
0.1	1.00	2243	83.0	80.91	0.9
1.0	1.00	1114	41.2	48.00	1.6
10.0	0.98	442	16.4	18.74	4.0
50.0	0.84	398	14.7	16.58	4.5
100.0	0.86	375	13.9	15.66	4.8
500.0	0.95	421	15.6	18.62	4.0
1000.0	0.84	388	14.4	17.44	4.3

Table 2: Performance metrics comparing DBSCAN and DBSpan on varying size inputs. We compare accuracy between the clusterings with the adjusted Rand index (ARI) in column 2. The time to run DBSCAN and DBSpan are meaused in seconds and reported in columns 4 and 5, respectively.

Num Dgms	ARI	% Possible Edges	DBSCAN time	DBSpan time	Speedup
30	1.00	66.5	283.25	185.14	1.5
45	1.00	66.3	654.45	429.65	1.5
60	0.79	64.5	1007.53	656.37	1.5
75	0.94	55.9	1762.89	998.35	1.8
90	0.90	56.2	2435.76	1369.58	1.8
105	0.96	54.3	3476.13	1942.88	1.8
120	0.86	49.3	4457.90	2269.63	2.0

For $\delta > 10$, however, the sparsification (and along with it the loss of accuracy and speed up) seems to plateau at computing $15\% \pm 1.5\%$ of the possible edges. We have seen this plateau in other tests cases as well and would like to investigate further.

Experiment 2: Prostate Cancer Dataset. In this experiment, we study the scaling of DBSpan. For a fixed distance approximation, we compare various size datasets. As in the previous experiment, we treat the clusters from DBSCAN as ground truth and assess accuracy with ARI.

We consider a subset of images from the Kaggle Prostate cANcer graDe Assessment (PANDA) Challenge [5], partitioned into small 512×512 pixel regions of interest (ROIs). For each ROI, we use the Histocartography Python library [22] to extract nuclei centroids, and we use GUDHI [36] to compute a persistence diagram using the Vietoris-Rips filtration on these centroids. So that we consider interesting diagrams, we remove any diagram with less that 100 1d-persistence points. The resulting set of 8595 1d-diagrams are available on the Open Science Framework [18] in the DBSpan Test Dataset project [35]. The distribution of the number of points in these diagrams are imbalanced, so when we sample from this set, we create three levels: diagrams with less than 150 points, diagrams with between 150 and 200 points, and diagrams with more than 200 points; and perform disproportionate stratified random sampling. Then, the sampled diagrams are used as input into DBSCAN and DBSpan with common parameters $\epsilon = 10$ and $m = 5$.

Then, we pick a reasonable delta value for this experiment by sampling 30 diagrams from our dataset and identifying a δ in which the ARI is still close to 1. For this dataset, at $\delta = 1$, we have a ARI of .867 while only computing about 64% of the pairwise distance matrix.

In Table 2, we see the result of varying the number of diagrams and compare the speed and accuracy of DBSpan to DBSCAN. The first column is the number of diagrams to cluster. The second column is the ARI of the DBSpan output as compared to the DBSCAN ground truth. The third column is the proportion of edges in the spanner to the complete graph. The fourth column is the time (in seconds) for running DBSCAN. The fifth column is the time (in seconds) for running DBSpan. The sixth column is the speedup of DBSpan.

Observe that as the input gets larger we increase the speedup from 1.5 to 2. As one would expect, speed seems to increase with the reduction of the number of edges. Perhaps, unexpectedly, the proportion of edges computed decreases as the as we increase the number of nodes. For all but two trials, the ARI is above .9. Note that from the previous experiment, one would not expect the ARI to change substantially for a fixed δ. It seems that further study is required to understand how to pick a more predictable δ for clustering.

5. Conclusions. In this paper, we introduced a spanner into the DBSCAN algorithm to facilitate range queries in spaces with expensive distance computations. In the practical side, we saw that even when we remove the computation of a small fraction of distances, we see improvements in performance with little deviation from the output of DBSCAN. We believe that we could see an even larger speedups by optimizing the implementation and by parallelizing the construction of the spanner. Kerber and Nigmetov [25] proposed multiple heuristics for building up the spanner without computing all distances. In this paper, we only considered one method. Are there better ways to add edges to the spanner when the task is approximating neighborhoods? Finally, our process for picking δ is done in a somewhat ad-hock fashion. It would be useful to develop a more theoretically guided method.

Acknowledgments. We thank users of the R package TDA[2] for questions on how to improve pairwise distance computations that inspired us to work in this direction. We thank Jordan Schupbach for pre-computing persistence diagrams, and Ben Holmgren and Braeden Sopp for their thoughtful feedback. We also thank Michael Kerber for conversations after we read [25], a key result needed for DBSpan.

REFERENCES

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete & Computational Geometry, 9 (1993), pp. 81–100.
[2] A. Beygelzimer, S. Kakade, and J. Langford, *Cover trees for nearest neighbor*, in Proceedings of the 23rd international conference on Machine learning, 2006, pp. 97–104.
[3] S. Biasotti, A. Cerri, D. Giorgi, and M. Spagnuolo, *PHOG: Photometric and geometric functions for textured shape tetrieval*, in Computer Graphics Forum, vol. 32, Wiley Online Library, 2013, pp. 13–22.
[4] D. Birant and A. Kut, *ST-DBSCAN: An algorithm for clustering spatial-temporal data*, Data & knowledge engineering, 60 (2007), pp. 208–221.
[5] W. Bulten, K. Kartasalo, P.-H. C. Chen, P. Ström, H. Pinckaers, K. Nagpal, Y. Cai, D. F. Steiner, H. van Boven, R. Vink, et al., *Artificial intelligence for diagnosis and gleason grading of prostate cancer: the panda challenge*, Nature medicine, (2022), pp. 1–10.
[6] R. J. Campello, D. Moulavi, and J. Sander, *Density-based clustering based on hierarchical density estimates*, in Pacific-Asia conference on knowledge discovery and data mining, Springer, 2013, pp. 160–172.
[7] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, *Hierarchical density estimates for data clustering, visualization, and outlier detection*, ACM Transactions on Knowledge Discovery from Data (TKDD), 10 (2015), pp. 1–51.
[8] S. Chen and Y. Wang, *Approximation algorithms for 1-Wasserstein distance between persistence diagrams*, in 19th International Symposium on Experimental Algorithms (SEA), 2021.
[9] Y. Chen, W. Ruys, and G. Biros, *KNN-DBSCAN: A DBSCAN in high dimensions*, arXiv preprint arXiv:2009.04552, (2020).
[10] L. P. Chew, *There is a planar graph almost as 600d as the complete graph*, 2 (1986), pp. 169–177.
[11] K. L. Clarkson, *Nearest-neighbor searching and metric space dimensions*, in Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, G. Shakhnarovich, T. Darrell, and P. Indyk, eds., MIT Press, 2006, p. 1559.
[12] R. Cotsakis, J. Shaw, J. Tierny, and J. A. Levine, *Implementing persistence-based clustering of point clouds in the Topology ToolKit*, in Topological Methods in Data Analysis and Visualization VI, Springer, 2021, pp. 343–357.
[13] T. K. Dey and S. Zhang, *Approximating 1-Wasserstein distance between persistence diagrams by graph sparsification*, in 2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX), SIAM, 2022, pp. 169–183.

[2] https://cran.r-project.org/web/packages/TDA/vignettes/article.pdf

[14] P. Dłotko and N. Hellmer, *Bottleneck profiles and discrete prokhorov metrics for persistence diagrams*, (2021). arXiv preprint arXiv:2106.02538.

[15] H. Edelsbrunner and J. Harer, *Computational Topology: An Introduction*, American Mathematical Society, 2010.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., *A density-based algorithm for discovering clusters in large spatial databases with noise*, in kdd, vol. 96, 1996, pp. 226–231.

[17] B. T. Fasy, X. He, Z. Liu, S. Micka, D. L. Millman, and B. Zhu, *Approximate nearest neighbors in the space of persistence diagrams*, 2018. arXiv preprint arXiv:1812.11257.

[18] E. D. Foster and A. Deardorff, *Open science framework (OSF)*, Journal of the Medical Library Association, 105 (2017), pp. 203–206.

[19] L.-A. Gottlieb and L. Roditty, *An optimal dynamic spanner for doubling metric spaces*, in European Symposium on Algorithms, Springer, 2008, pp. 478–489.

[20] M. Hahsler, M. Piekenbrock, and D. Doran, *dbscan: Fast density-based clustering with R*, Journal of Statistical Software, 91 (2019), pp. 1–30.

[21] U. Islambekov and Y. R. Gel, *Unsupervised space–time clustering using persistent homology*, Environmetrics, 30 (2019), p. e2539.

[22] G. Jaume, P. Pati, V. Anklin, A. Foncubierta, and M. Gabrani, *Histocartography: A toolkit for graph analytics in digital pathology*, in MICCAI Workshop on Computational Pathology, 2021, pp. 117–128.

[23] M. Kerber, D. Morozov, and A. Nigmetov, *Geometry helps to compare persistence diagrams*, Journal of Experimental Algorithmics (JEA), 22 (2017), pp. 1–4.

[24] M. Kerber and A. Nigmetov, *Spanners for topological summaries*, June 2018. CG Week Young Researcher's Forum.

[25] M. Kerber and A. Nigmetov, *Metric spaces with expensive distances*, International Journal of Computational Geometry & Applications, 30 (2020), pp. 141–165.

[26] A. Lawson, T. Hoffman, Y.-M. Chung, K. Keegan, and S. Day, *A density-based approach to feature detection in persistence diagrams for firn data*, Foundations of Data Science, (2021).

[27] L. McInnes, J. Healy, and S. Astels, *HDBSCAN: Hierarchical density based clustering*, Journal of Open Source Software, 2 (2017), p. 205.

[28] M. Michel, J. Le Deunf, N. Debese, L. Bazinet, and L. Dejoie, *Multibeam outlier detection by clustering and topological persistence approach, ToMATo algorithm*, in OCEANS 2021: San Diego–Porto, IEEE, 2021, pp. 1–8.

[29] R. R. Moraleda, W. Xiong, N. A. Valous, and N. Halama, *Segmentation of biomedical images based on a computational topology framework*, in Seminars in Immunology, vol. 48, Elsevier, 2020, p. 101432.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[31] J. A. Pike, A. O. Khan, C. Pallini, S. G. Thomas, M. Mund, J. Ries, N. S. Poulter, and I. B. Styles, *Topological data analysis quantifies biological nano-structure from single molecule localization microscopy*, Bioinformatics, 36 (2020), pp. 1614–1621.

[32] Y. Qin, B. T. Fasy, C. Wenk, and B. Summa, *A domain-oblivious approach for learning concise representations of filtered topological spaces for clustering*, IEEE Transactions on Visualization and Computer Graphics, 28 (2021), pp. 302–312.

[33] W. M. Rand, *Objective criteria for the evaluation of clustering methods*, Journal of the American Statistical association, 66 (1971), pp. 846–850.

[34] N. Saul and C. Tralie, *Scikit-TDA: Topological data analysis for python*, 2019, https://doi.org/10.5281/zenodo.2533369, https://doi.org/10.5281/zenodo.2533369.

[35] J. Schupbach, *Dbspan test dataset*, Apr 2022, osf.io/uj32d.

[36] The GUDHI Project, *GUDHI User and Reference Manual*, GUDHI Editorial Board, 3.5.0 ed., 2022, https://gudhi.inria.fr/doc/3.5.0/.

[37] N. X. Vinh, J. Epps, and J. Bailey, *Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance*, Journal of Machine Learning Research, 11 (2010), p. 28372854.

[38] D. S. Willard, *New data structures for orthogonal range queries*, SIAM Journal on Computing, 14 (1985), pp. 232–253.

[39] Y.-P. Wu, J.-J. Guo, and X.-J. Zhang, *A linear DBSCAN algorithm based on LSH*, in 2007 International Conference on Machine Learning and Cybernetics, vol. 5, IEEE, 2007, pp. 2608–2614.