Efficient Multi-Scale Simplicial Complex Generation for Mapper

Matt Piekenbrock*, Derek Doran†, and Ryan Kramer‡

Abstract. Mapper is amongst the most widely used algorithms for topological data analysis. In its simplest interpretation, Mapper takes as input a set of 'point cloud data' and a map defined on the data to produce a graph that approximates the topological structure of the data. It is a powerful tool for unsupervised data analysis and exploration, but relies on a number of parameters that heavily influence the quality of the resulting construction. One crucial parameter, sometimes called the *overlap* or *gain* parameter, controls the entire relational component of the output graph. Fixing this parameter restricts the scale of the analysis, and in practice there is sparse guidance and few guideposts to help an analyst determine what parameters may provide 'better' results. Subject to moderately weak assumptions that are often met in practical settings, we introduce an indexing structure able to produce all possible *mappers* of a dataset for any value of this crucial parameter. As a result, once an initial *mapper* is computed, alternative *mappers* at different scales require just a fraction of time to compute. We empirically demonstrate order-of-magnitude speed up across *mapper* constructions over multiple datasets and discuss other theoretical advancements that may follow from efficient multi-scale *mapper* constructions.

Key words. Mapper, Topology Theory, Unsupervised Learning

AMS subject classifications. 54H99, 54C05, 54J05

Introduction. Methods incorporating *topology theory* have rapidly emerged in the realm of machine learning and data analysis rapidly in recent years. Topology theory explores deformations of maps and spaces, and has been referred to as the natural evolution of the notions of proximity and continuity [14]. Drawing theory from fields such as computational geometry, algebraic topology, machine learning, and statistics, the development of topological methods and algorithms that utilize these theories are becoming increasingly useful in studying data-centric problems. The specific applications of these theories to data-centric analysis are collectively being referred to as *topological data analyses* (TDA). TDA has recently been used in applied research in domains such as microbiology [32], material science [17], and sensor networking [7].

Perhaps the most popular tool for applied TDA is Mapper, developed by Singh *et al.* [30]. Mapper is a coordinatization framework capable of summarizing high-dimensional data sets, or functions of them, and has proven its worth as a useful tool for data visualization [15], genetic recombination characterization [12], and dimensionality reduction [30]. Mapper has also found much success in commercial settings [1].[1] In its simplest interpretation, Mapper is a topologically-motivated framework for reducing high dimensional data into an interpretable graph $G(V, E)$ where vertices correspond to clusters of data, and edges correspond to in-

*Department of Computer Science, Wright State University, Dayton, OH (piekenbrock.5@wright.edu, http://www.wright.edu/~piekenbrock.5/).

†Department of Computer Science, Wright State University, Dayton, OH (derek.doran@wright.edu, http://knoesis.org/people/derek/).

‡U.S. Air Force Research Laboratory, Wright-Patterson Air Force Base, OH (ryan.kramer.3@us.af.mil).

[1] Mapper's framework lies at the core of the Ayasdi platform (https://www.ayasdi.com/solutions/)

teractions (non-empty intersections) between clusters. Mapper thus provides a simplified representation of high-dimensional data that not only *preserves* certain topological structures of interest, but is also amenable to qualitative analysis and visualization.

In this work, we derive closed-form expressions to compute the set of minimal parameter values that, under relatively weak assumptions on the cover, produce 'distinct' topological constructions with Mapper, with the exact definition of 'distinct' to be defined later in Section 4. Using these expressions, we create a new approach to generate *mappers* more efficiently, and we discuss some example applications our results may enable or enhance. We analyze the storage and runtime complexities of our approach, and demonstrate its efficiency with experiments performed on real-world data sets with realistic parameter settings often used in practice. Additionally, we outline the connections between our analysis in relation to other theoretical results involving Mapper.

Aside from the aforementioned contributions, we also make available an R [28] implementation of the data structure we use to construct the *mappers*, discussed further in Section 4, for public use.[2] To our knowledge, this is the first open-source implementation of such a structure for Mapper and incidentally, for the specific covers we consider, also the first open-source implementation to our knowledge of the so-called *multiscale mapper* construction [8]. Finally, of separate interest is the reproducibility of our experiments and results. All datasets, benchmarks, examples, and source code associated with this research is publicly available online and fully reproducible[3].

In the next section, we elaborate on a few of the motivations for this effort and discuss some of the related works. In Section 1, we discuss some prerequisite background material including preliminary notation that will be used throughout the paper. We also recall Mapper's formal definition, and follow by giving an informal or algorithmic overview of the Mapper framework. In Section 4, we derive a closed form solution for finding the minimal set of values for the overlap setting. In Section **??**, examples are given and discussed in detail to further convey the motivation, significance, and utility of our solution. Finally, we discuss future applications and work related to Mapper in Section 8.

Motivation. Mapper is a general framework capable of describing heterogeneous formats of data, and hence naturally requires a number of parameters. Provided a reference map f, a covering over a metric space $\{\mathcal{U}\}$, a clustering algorithm $\mathcal{C}$ using some distance metric (M, d), and their corresponding hyper-parameters, Mapper deterministically generates G. Mapper requires several choices to be made by the user based on the goal of the analysis at hand. Naturally, a large parameter space generally implies a large solution space: changing the value of any particular parameter in may lead to a different set of solutions, which may in turn lead to inconsistent interpretations and analyses. This is true for the construction Mapper produces as well. Solutions which reduce this complexity would not only enable more robust analyses of the topological solutions produced by Mapper, but may further accelerate the adoption and reach of TDA tools and methods for data analysis.

The focus of this effort is to analyze the *overlap* parameter associated with a specific

[2]The Mapper source code is available here: https://github.com/peekxc/Mapper

[3]The source code the experiments and corresponding data is available here: https://github.com/peekxc/IndexedMapper

class of covers often used in practice with Mapper, which we describe further in Section 1. We are interested in this parameter, as it solely determines the connectivity of the output graph G, and thus the relational qualities of the data that are modeled. At the smallest possible overlap value (0) the graph is completely disconnected, and as the overlap increases, the graph becomes ever more connected. Unfortunately, there is little practical guidance on how to determine this parameter value given a particular data set. If overlap is too high, spurious connections may be shown in the output graph, conveying a relational structure that is misrepresentative of the underlying data topology. Conversely, if it is too low, subtle yet important connections may be missed.

Is there an optimal scale that best captures the topology of the data with respect to Mapper's overlap parameter? To answer this, it's worth exploring how such scaling choices are performed for other types of TDA. Consider a well-known topological construction of data X, such as the Vietoris-Rips Complex at scale ϵ, which is a simplicial complex obtained by adding n-dimensional simplices spanning the points of X when the pairwise intersection of $B(X_1, \epsilon) \cap B(X_2, \epsilon) \cap \cdots \cap B(X_n, \epsilon)$ is non-empty, where $B(X_i, \epsilon) = \{x \in X \mid d(X_i, x) \leq \epsilon\}$. As the sole parameter which determines connectivity, the ϵ parameter is analogous to the overlap parameter of Mapper, and as Ghrist argues: *"Despite being both computable and insightful, the homology of a complex associated to a point cloud at a particular ϵ is insufficient: it is a mistake to ask which value of ϵ is optimal"* [13]. We extend this assertion to Mapper: a fixed scale parameter is insufficient if the task is study the structure in the context of *homology*. Rather, the solution is to look at the *continuum* of possible values towards understanding how the complex changes across ranges of parameter settings. Intuitively, substructures which seem to "persist" across parameter ranges forms a foundation for many approaches in Persistent Homology [24]. This notion extends to exploratory use-cases of Mapper as well—rather than choose an optimal scale to extract the "best" output of Mapper, a more practical solution may be to allow an analyst to explore the space of possible parameter values. By viewing multiple topological representations of the data at once, an analyst could compare and contrast solutions, derive insights, and extract richer knowledge on the structure of the data. Even better, perhaps the ideal solution may be to allow the analyst to *move between representations* fluidly, such that they may view how the topological structure of the data changes with scale. The analyst could then understand how sensitive the topological representation of the data (i.e. the output graph of Mapper) is to changes in other settings of interest, pointing to directions of stability and instability, and supplementing the theoretical guarantees persistence lends.

This notion of studying topological constructions on a continuum of parameter ranges is not new. Methods measuring some notion of 'stability' or 'persistence' of a given real-valued function with respect to its domain have proven useful in both theoretical and practical instances of TDA [13]. In the context of the Mapper construction explicitly, theoretical extensions which analyze Mapper at multiple resolutions are emerging [8, 9]. Nonetheless, using such extensions to performing TDA on empirical data requires explicit assumptions of how a Mapper construction is parameterized. Such types of empirical analyses may only be enabled tractably, however, if the the solutions to various parameterizations can be computed efficiently.

1. Background. This section provides a succinct review of the relevant notation and topological constructs needed to describe Mapper and our corresponding extensions. We give a concise summary of the Mapper algorithm, leaving the reader to refer to the original paper by Singh *et. al* [30] for deeper background material. An even greater treatment of the topological ideas underlying Mapper can be found in Carlsson's exposition [5]. Still further, Munkres et al. [26] provides a review of set theory and other foundational topological theory, Merkulov et al. [21] provides more detailed information on simplicial complexes, and Munch [24] and Ghrist [14] may be consulted for either introductory or high-level overviews TDA and its applications, respectively.

1.1. Simplicial Complex. We start by defining a simplicial complex: the principle outcome of Mapper. A *simplicial complex* is a pair $K = (V, S)$ where V is a finite set of 0-simplexes whose elements we'll refer to as *vertices* of K, and S is a set of non-empty subsets of V that satisfies that following two conditions:

1. $p \in V \Rightarrow \{p\} \in S$
2. $\sigma \in S, \tau \subset \sigma \Rightarrow \tau \in S$

Each element of $\sigma \in S$ is called a *simplex* of K (or s-simplex, where $|\sigma| = s + 1$). Given an s-simplex σ, its *faces* are the $(s-1)$-simplices τ in S such that $\tau \in \sigma$. The dimension of the simplicial complex K is the largest k such that S is a k-simplex. Further define the j-skeleton $K^{(j)} = \{\sigma \in K \mid \dim \sigma \leq j\}$ of K as the simplicial complex comprising all simplices of K that are of dimension j or less. Since $K^{(1)}$ is composed of the 0 and 1 simplexes of K, it is structured like a graph, and we will use the terms 'vertex' and 'edge' to describe its 0 and 1-simplexes, respectively.

1.2. Covers. Define a *cover* of a set X as a collection of open sets U_α whose union contains X as a subset, i.e. $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ is a cover of X if $X \subseteq \bigcup_{\alpha \in A} U_\alpha$ for some index set A. We will always assume that each U_α is path-connected and a cover means a finite open cover. We sometimes will subscript the cover with a parameter if the collection of its sets are depend on the value of the parameter, e.g. we may write $\mathcal{U}_\epsilon$ to mean the collection of sets parameterized by ϵ whose union forms a cover. We will often index sets with $\mathbb{Z}^+$ to imply order, i.e. for a given collection of sets $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$, when we write $U_1, U_2, \ldots, U_k$, its implied the index set A is well-ordered.

1.3. Other notation. We will use lowercase symbols to represent scalars, capital symbols to represent sets (or random variables, based on the context), and **bold** lowercase and capital symbols to represent *row* vectors ($1 \times m$) matrices ($n \times m$), respectively. Unless otherwise stated, we will use $\circ$ to denote the Hadamard product to expressing element-wise operations between vectors and matrices. We make heavy use of other element-wise operations on equally-sized vectors, and thus extend the inequality relations $R \in \{\leq, \geq, <, >\}$ to vectors such that $\boldsymbol{x}\ R\ \boldsymbol{y}$ implies $x_i\ R\ y_i\ \forall\ i \in \{1, 2, \ldots, n\}$ when $n = |\boldsymbol{x}| = |\boldsymbol{y}|$.

2. Mapper. We first introduce the Mapper algorithm [30, 5]. It is often useful to discuss the algorithm in terms of its resulting topological construction/object [24]; we will use and *emphasized* text whenever referring to a Mapper construction. We follow Singh *et al.*'s notion by first defining Mapper using the *topological* version of the definition, and then the analogous *statistical* version for point cloud data. We begin by with defining the Nerve of a cover.

Definition 2.1 (Nerve of a cover). *Given a finite cover $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ of a topological space X, define the* nerve *of the cover $\mathcal{U}$ to be the simplicial complex $N(\mathcal{U})$ whose vertex set is the index set A, and where any subset $\{\alpha_0, \alpha_1, \ldots, \alpha_k\} \subseteq A$ spans a k-simplex in $N(\mathcal{U})$ if and only if $U_{\alpha_0} \cap U_{\alpha_1} \cap \cdots \cap U_{\alpha_k} \neq \emptyset$.*

Suppose one is given a continuous map $f : X \to Z$ where Z is equipped with a cover $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$. For each α, introduce an inverse map $f^{-1}(U_\alpha) = \{x \in X : f(x) \in U_\alpha\}$ and write it as a decomposition of its path connected components $\{V(\alpha, i)\}$, so that $f^{-1}(U_\alpha) = \bigcup_{i=1}^{j_\alpha} V(\alpha, i)$, where j_α is the number of such path connected components. Define $f^*(\mathcal{U})$ as the cover of X where $\bigcup_\alpha \{V(\alpha, i)\}$ are its open sets. Obtaining a cover of X this way is sometimes referred to as the *pullback*[4] cover of X induced by $\mathcal{U}$ via f. With these components, Mapper yields a *topological construction* as the nerve of the pullback cover $f^*(\mathcal{U})$.

Definition 2.2 (Mapper). *Let X and Z be topological spaces and let $f : X \to Z$ be a continuous map. Let $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ be a finite open cover of Z. The* mapper construction of X, *or* mapper *for short, is defined to be the nerve of the pullback cover:*

$$M(\mathcal{U}, f) := N(f^*(\mathcal{U}))$$

If one considers a real-valued function $f : X \to \mathbb{R}$ as the map and a cover $\mathcal{U}_\epsilon$ of Z consisting of intervals of length ϵ, the corresponding *mapper* $M(\mathcal{U}_\epsilon, f)$ may be thought of as an approximation of the Reeb graph (or, when f is a multivariate mapping, the Reeb space), where the degree of the approximation is determined by the size of ϵ. In fact, Mapper was conjectured to recover the Reeb graph precisely when $\epsilon \to 0$ in its original exposition [30]. More recently, a theoretical study of Mappers convergence to the Reeb graph (and space, resp.) of f as $\epsilon \to 0$ has been explored by Munch et al. [25].

2.1. Algorithmic description. In what follows we give a more informal, step-wise description of Mapper from the point-of-view where one has a set of point cloud data (PCD) $X \subset \mathbb{R}^d$ one is interested in performing a TDA on. A *mapper* is constructed algorithmically as follows:

1. Define a continuous map $f : X \to Z$ where Z is some reference metric space. Note that because Z is a metric space, it's assumed that it is possible to compute inter-point distances between the points in the data of Z.
2. Select a finite covering $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ of Z.
3. Construct the subsets $X_\alpha = f^{-1}U_\alpha$. Since f is continuous, the union of these sets for all α form an open covering of X.
4. Given a clustering algorithm $\mathcal{C}$, construct the set of clusters by applying $\mathcal{C}$ to each set X_α. This induces a new covering of X parametrized by pairs (α, c) where c is one of the clusters of X_α.
5. Construct a simplicial complex $N(\mathcal{U})$ whose vertex set is the set of all possible such pairs (α, c) and where a family $\{(\alpha_0, c_0), (\alpha_1, c_1), \ldots, (\alpha_k, c_k)\}$ spans a k-simplex if and only if the corresponding clusters have a non-empty $(k+1)$-fold intersection.

Proceeding item-wise through these steps, one begins a TDA with Mapper by defining a reference map, sometimes referred to as a *filter* function, which takes as input a set of PCD

[4]The term *pullback* has also been expressed as an *operation*[8]. We will also refer to it this way.

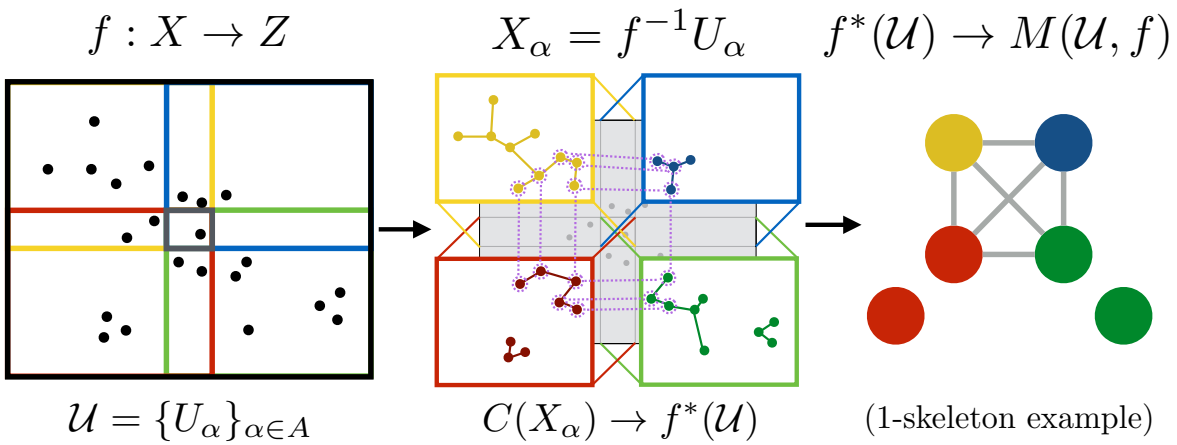

Figure 1: Mapper example. The left panel shows the mapping of a given data set $X \to Z$ via f (where $Z \in \mathbb{R}^2$) and four overlapping sets that construct the covering $\mathcal{U}$ of Z. The center figure shows the isolation the points in each of these sets that is used for partial clustering and the connected components represent the results of this clustering. Points which appear in multiple subsets are connected by dashed lines. The final figure on the right represents the 1-skeleton realization of the simplicial complex constructed from the nerve of the covering.

defined in the *data space* and maps to an appropriate metric space, which we'll refer to as the *filter space* or Z-space. The *mapper* is highly dependent on this function. The specific choice of this function is inevitably application-specific, however details on common functions that may of interest are discussed in [6, 30] and also in Section 6.1. In step 2, a parameterized covering is constructed on the Z-space, partitioning the filtered data into (often overlapping) subsets based on which set of the covering U_α they intersect. We will frequently refer to these sets as *level sets*, due to their relaxed interpretation as such (this is elaborated on in the next section). Because level sets generally overlap, any given data point may be within multiple level sets. Observe that since f is continuous, the constructed cover of Z also forms a covering of X, via the *pullback* operation $f^*(\mathcal{U})$. Mapper relies on the idea of *partial clustering*, wherein subsets of X are clustered independently of other subsets as a means of simplifying the data into topological prototypes, where each prototype is realized as a vertex in the resulting construction, and each vertex represents *cluster* of points. Because the subsets to perform the clustering on are determined by the cover constructed in the Z-space, *mapper* can be thought of as a topological summarization of the data expressed through the range space of a mapping. Non-empty intersections of these vertices with other vertices form the higher dimensional simplexes in the resulting simplicial complex. Figure 1 illustrates, at a high level of abstraction, how the Mapper algorithm creates a simplicial complex from a given set of data. When we restrict ourselves to the 1-skeleton, Mapper essentially produces a graph describing topological information about a data set and a map defined on it. The degree of the summarization is expressed by the coarseness of the cover equipped on the codomain of the map. In the next section, we go more in-depth on how these covers are often constructed in practice.

2.2. Cover Parameterization. Theoretically, Mapper makes no assertions restricting the shape of the sets U_α used to construct the covering. For example, Figure 2 of [30] mentions two example covers of the parameter space in $\mathbb{R}^2$, one with rectangles and another with hexagons. Indeed, the Mapper construction is very general—there are numerous ways to construct a suitable covering. Consider the following options for $\mathcal{U} = \{U_\alpha\}$:

1. $U_\alpha = (-\infty, \alpha)$ *for* $\alpha \in \mathbb{R}$.
2. $U_\alpha = (\alpha - \epsilon, \alpha + \epsilon)$ *for* $\alpha \in \mathbb{R}$ *, for some fixed* $\epsilon > 0$.

The first option describes a cover comprised of sublevel sets, which collectively may be combined to represent a class of hierarchical structures sometimes referred to as *merge trees*, while the second partitions points of X into (ϵ-*thick*) level sets (also sometimes referred to as interval-level sets), which induce a relaxed notion of Reeb graphs. Because (ϵ-*thick*) sets represent intervals of length ϵ in the image of f, the cover of intervals of length ϵ is commonly denoted as $\mathcal{U}_\epsilon$. In this effort, we will focus on a few specific implementations of covers composed of intervals. We will always refer to the generalized intervals as intervals (represented by U_α), and the corresponding set of points in their preimage $f^{-1}(U_\alpha)$ as level sets, respectively.

2.3. Multiresolution Structure. Any *mapper* construction has a natural 'multiresolution' or 'multiscale' structure [30]. If one considers two covers $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ and $\mathcal{V} = \{V_\beta\}_{\beta \in B}$ of a space X, then one may define a *map of covers* from $\mathcal{U}$ to $\mathcal{V}$ by constructing a map $\xi : A \to B$ such that $U_\alpha \subseteq V_{\xi(\alpha)}$ for all $\alpha \in A$. Given such a map, there is an *induced* map of simplicial complexes $N(f) : N(\mathcal{U}) \to N(\mathcal{V})$ given on the vertices by the map ξ. If we have a family of covers $\mathfrak{U} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_n\}$ and a map of covers for each $i \in [1, n-1]$, $\xi_i : \mathcal{U}_i \to \mathcal{U}_{i+1}$, then one may construct a diagram of simplicial complexes connected by the induced simplicial maps between them:

$$N(\mathcal{U}_1) \overset{\xi_1}{\to} N(\mathcal{U}_2) \overset{\xi_2}{\to} \ldots \overset{\xi_{n-1}}{\to} N(\mathcal{U}_n)$$

This extends to the case of pullbacks as well: given a space X, a map $f : X \to Z$, and a family of covers $\mathfrak{U}$ equipped to Z, there is a corresponding family of pullback covers $f^*(\mathfrak{U})$ and a diagram of induced simplicial maps that follows. It can be shown that the induced simplicial maps associated with any two maps of covers are also *contiguous*, implying they induce identical maps at the homology level [8]. We do not address these in detail in our methodology, however see Section 7 for a small discussion on the matter.

3. Main Results Overview. The focus of this paper is to show an efficient computation of a *mapper* at multiple scales, focusing on reducing the complexity of computing the graph representation of Mapper. This amounts to improving the time to compute successive covers and their corresponding 0 and 1-skeletons. Our strategy is to compute the *mapper* once for a fixed number of non-overlapping intervals, then use the filtered points Z to compute an indexing structure capable of updating the graph for successive values of ϵ efficiently. The intuition is that, as ϵ changes, not all of the *point memberships* associated with the sets in the cover change. This observation is crucial, because if any of these point sets change, so

too will the clustering, implying all resulting simplices must be re-evaluated.[5] However, if the increase in ϵ is relatively small, not all sets in the cover must have their corresponding simplices recomputed: only the intervals whose corresponding level sets change (i.e. by the addition or removal of at least one point) ought to be considered. By ingraining the information of which intervals change into a data structure that can be easily queried, we can identify and minimize the number of simplices of a *mapper* which need to be recomputed between successive ϵ parameterizations. Since the number of intervals exponentially increases with the dimension of Z[6], the disparity between the total number of intervals and the set of intervals whose level sets need have changed also scales in a similar fashion.

We introduce the assumptions made about the family of covers used to compute the *mappers* of varying scale, and how we utilize each assumption in turn. Although our primary contribution is a structured methodology for updating *mapper* for *any* parameterization of the cover, we note that our methods may be used to enable other types of analysis, as we will demonstrate later. We introduce the concepts and terminology for the $d = 1$ case first, and then discuss generalizing to higher dimensions after in Section **??**.

4. Methodology. Our approach makes specific assumptions about the geometry of the family of covers used to construct the *mapper*. In what follows, we assume one already has defined a map $f : X \to Z$, wherein $X \subset \mathbb{R}^D$, $Z \subset \mathbb{R}^d$. We will generally assume $d << D$. We will consider the generic family of covers of generalized intervals, formed by the Cartesian product of tuples of the form:

$$\begin{aligned} \mathcal{U}[r, g] &= \{U_\alpha\}_{\alpha \in A} = \textstyle\prod_{i=1}^{d} (a_i, b_i) \\ r &= \ell(U_\alpha) \\ g &= \frac{\ell(U_\alpha \cap U_{\alpha+1})}{\ell(U_\alpha)} \end{aligned} \tag{1}$$

where $r \in \mathbb{R}$, $g \in [0, 1)$, the index set A consists of integers $\{1, 2, \ldots, k\}$, and ℓ is the Lebesgue measure on $\mathbb{R}$. This a 2-parameter family of coverings, wherein r conveys the length of the interval, and g determines the degree to which adjacent intervals overlap. It's common to limit the covering dimension by restricting $g < \frac{1}{2}$ in order to prevent non-trivial 3^d-fold intersections, which keeps the *mapper* from getting too densely connected. To connect the notation given in Section 2.2, r has length 2ϵ. Since these intervals are constructed on the range of f, their preimages decompose the data X into a (relaxed) notion of level sets called *interval-level sets*, defined as follows:

Interval-level set. *Given a map $f : X \to Z$ and a cover $\mathcal{U}$ equipped to Z, the* interval-level set *is defined as:*

$$\mathcal{L}_\alpha(\mathcal{U}, Z) = f^{-1}(U_\alpha) = \{x \in X \mid f(x) \in U_\alpha\}$$

We consider all covers in this family obeying the following properties:

[5] This implication may be relaxed if one makes further assumptions on the algorithm used to perform the clustering. We choose to remain agnostic to this choice, focusing instead on an approach that works for any clustering algorithm.

[6] This is assuming Z is equipped with a cover parameterized by a parameter k, where k represents the number of intervals to distribute along each dimension.

Assumptions about the form of the covers:

1. There is a *fixed* number of intervals along each dimension.
2. All subsets of $\mathcal{U}$ are in the shape of *iso-oriented rectangles*.[7]
3. The sets U_α have *reflective symmetry* about the center of the range of Z.

Note that we do not assume r is necessarily constant for each interval in the cover. Assumption (3) requires the lengths of the intervals reflect about the center (except for the central most interval, if the number of intervals is odd). That is, if one has the cover $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$ consisting of k intervals where k is even and each interval U_i has length r_i, then assumption (3) implies that $r_1 = r_k$, $r_2 = r_{k-1}$, and so on.

We first consider the one dimensional case where $f : X \to \mathbb{R}$ and all interval lengths are equal. We would like to represent any arbitrary such cover of $\mathcal{U}_r$ where r may be any length in the range

$$\bar{r} < r < \infty$$

where $\bar{r}$ is the smallest interval length such that $\mathcal{U}_{\bar{r}}$ covers Z. Intuitively, if each interval is exactly of length r, the smallest interval size that contiguously covers the full range of Z will have size $r = k^{-1}\text{range}(Z)$ (and $g = 0$). Denote this interval length as the *base interval length*. There are, of course, an infinite number of such cover parameterizations in the range $\bar{r} < r < \infty$. However, for a fixed data set X, observe that the *mapper* does not have infinite number of representations if we consider a specific covering scheme. For example, consider two *mappers* constructed from a fixed number of intervals k, one the cover $\mathcal{U}_r$ and one with $\mathcal{U}_{r'}$ where e.g. $r' = r + \epsilon$. If the data set X is considered fixed, and if ϵ is small enough, the level sets may not change between the two covers. Assuming the clustering algorithm is deterministic, if the the level sets do not change between two successive covers, there will be no new simplices introduced or removed between the two Mapper constructions: they will be identical. This motivates a notion of *distinction* between covers, and as a by-product, between *mapper* constructions.

Distinct Cover. *Given a map $f : X \to Z$ which produces a fixed set of point cloud data $Z \subset \mathbb{R}^d$ equipped with two covers $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ and $\mathcal{V} = \{V_\alpha\}_{\alpha \in A}$. We interpret two covers $\mathcal{U}$ and $\mathcal{V}$ over the image of f as distinct if and only if:*

$$\exists \alpha \in A \text{ s.t. } \mathcal{L}_\alpha(\mathcal{U}, Z) \neq \mathcal{L}_\alpha(\mathcal{V}, Z)$$

Thus, if one has two *mapper* instances $M(\mathcal{U}, f)$ and $M(\mathcal{V}, f)$ constructed with f, they are also considered *distinct* if $\mathcal{U}$ and $\mathcal{V}$ are distinct.

4.1. Computing cover parameters. Consider a cover $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ meeting the assumptions discussed in the previous section. Each of the k intervals in the cover has the form:

$$U_\alpha = [a, b], a < b, a, b \in \mathbb{R}$$

Definition 2 motivates considering which interval lengths $(r, r'), r < r'$ produce the sequence

$$\mathcal{U}_r \to \mathcal{U}_{r'}$$

[7] A set of rectangles on a plane are said to be *iso-oriented* or *iso-aligned* if their edges lie parallel to the coordinate axes. They have also been referred to as *axis-parallel* intervals.

such that there is no parameter r^* between $r < r^* < r'$ that yields a cover distinct from the covers $\mathcal{U}_r$ and $\mathcal{U}_{r'}$. To find this sequence, first consider a cover meeting our assumptions previously discussed, with the additional assumption that the center of each interval $\frac{a+b}{2}$ in the cover is *fixed*. Thus, each interval is parameterized by the same length $r = |a - b|$, and so as $r \to \infty$, each point $z_i \in Z$ will intersect at most $|\mathcal{U}_\infty| = k$ intervals. Since the centers of the intervals are fixed and they are all the same length, the order in which these intersections occur is fixed. Since every point intersects exactly one interval when r is at its smallest value $\bar{r}$, and there are k will intervals total, then as $r \to \infty$ there are $k-1$ intervals where any given point will intersect an interval it did intersect when $r = \bar{r}$. By the definition of distinction given by 2, if we have n distinct points in $\mathbb{R}$, there are $m = n(k-1)$ values of ϵ producing distinct constructions (excluding the case where $g = 0$).

We would like to to determine these covering parameters *a priori* for a given data set X, before computing any *mappers*, and so we discuss a general strategy for handling rectangular covers of any dimension that take the form given by equation 1. Determining these values requires explicit analytical expressions for the covers—we discuss two popular covering strategies that fit our assumptions in the appendix B. The rectangular cover is generated by first finding the range of the function f restricted to the set of given points Z. The range vector is the component-wise difference between the extrema:

$$\bar{z} = \max(Z) - \min(Z)$$

Given a user-supplied input for either the interval length and overlap (r, g) or the number of intervals and overlap (k, g), a cover $\mathcal{U}$ consisting of overlapping intervals may be constructed. Notice that if k is fixed and $g = 0$, r is completely determined by the range the data, and vice versa. Recall that setting $g = 0$ yields the *base interval length*:

$$\bar{r} = k^{-1}\bar{z} \tag{2}$$

When the intervals are length $\bar{r}$, every point $z_i \in Z$ intersects exactly one interval, and there are no non-empty intersections between intervals. It's clear that the interval length must satisfy $r \geq \bar{r}$, otherwise the intervals would not contiguously cover the range of Z, which also implies $g \geq 0$. Now consider a cover composed of k intervals, $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$, and denote the map associating each point to its corresponding interval in the cover as $\phi : Z \to A$, where $A = \{1, \ldots, k\}$ such that if a point z lies in U_j, where $1 < j < k$, then $\phi(z) = j$. Suppose we wish to use these corresponding indices to compute the distance from every point z to its corresponding $(k-1)$ intervals. This distance inevitably depends on which halfspace the point lies in. If z_i lies in U_j, where $1 < j < k$, then $\phi(z_i) = j$. If $z_i < a_j + \frac{r_j}{2}$, then its distance to U_l for all $l \in [1, j-1]$ is $|b_l - z_i|$, whereas its distance to U_h for all $h \in [j+1, k]$ is $|a_h - z_i|$.

Define a distance matrix for all such points and sets as $\mathcal{D} \subset \mathbb{R}^{n \times (k-1)}$. Our observation is that a number of popular covers used in practice may have all possible distinct values of $\bar{r} < r < \infty$ expressed as a function of these distances $\mathcal{D}$ (see Appendix B). We denote this discrete set of parameters $\Sigma = \{\epsilon_1, \ldots, \epsilon_m\}$. That is, for a fixed k and Z, Σ represents all interval length parameters that produce distinct covers for a given family of covers, and a result, distinct *mapper* constructions.

Having the set Σ precomputed (without creating any *mapper* constructions) motivates the creation of an indexable function ζ which produces the equivalent *mapper* for any parameter $\bar{r} < r < \infty$ where $r \in \mathbb{R}^+$. Since all covers in this sequence are distinct, their *mappers* are distinct, implying that although we have the finite sequence

$$\mathcal{U}_{\epsilon_1} \to \mathcal{U}_{\epsilon_2} \to \cdots \to \mathcal{U}_{\epsilon_m}$$

we are able to index an arbitrary interval length r to some $\epsilon_i \in \Sigma$. That is, since Σ is comprehensive, the sequence may defined over all of $\mathbb{R}^+$. To demonstrate this, we may declare $\mathcal{U}_{\zeta(r)}$ where $r \in \mathbb{R}^+$, and ζ as

$$\zeta(r) = \max\{p \in \mathbb{Z}^+ \mid \epsilon_p \leq r, \epsilon_p \in \Sigma\}$$

The assumption that the centers of the intervals are fixed for all covers in $\zeta(r)$ can be relaxed, as we show in the next section, if the covers have reflective symmetry about the center.

4.2. Computing the level sets. We would like to use the information Σ provides to improve the efficiency of generating any cover indexed by ζ. The computational problem here is to quickly compute the *level sets* of these indexed covers in the sequence:

$$\forall\, r \in \mathbb{R}^+ \left(\mathcal{L}(\mathcal{U}_{\zeta(r)}, Z) = \mathcal{L}(\mathcal{U}_{\epsilon_1}, Z) \to \mathcal{L}(\mathcal{U}_{\epsilon_2}, Z) \cdots \to \mathcal{L}(\mathcal{U}_{\epsilon_m}, Z)\right) \tag{3}$$

where, when the data set Z is fixed in the sense that no points are to be added or removed,

$$\forall\, r \in \mathbb{R}^+ \left(\mathcal{L}(\mathcal{U}_{\zeta(r)}, Z) \Leftrightarrow \mathcal{L}(\mathcal{U}_r, Z)\right) \tag{4}$$

Conceptually, each cover $\mathcal{U}_{\epsilon_*}$ is associated with a collection of k interval-level sets. Obviously, precomputing and storing the entire sequence is feckless. An alternative but naïve way to enable the construction of any cover in this sequence is to create a set of k lists corresponding to the level sets $\mathcal{L}(\mathcal{U}_{\bar{r}}, Z)$, and then update each of the lists accordingly as $r \to \infty$. When $r = \bar{r}$, there is no overlap between the subsets of the cover, and thus the storage requirement for the level sets is $O(n)$. However, the storage becomes more redundant for each level set as r increases, requiring upwards of $\approx O(nk)$ as $r \to \infty$. Although much of the discussion that follows is from the perspective where Z is one-dimensional (we discuss how it extends to higher dimensions 4.5) it's worth noting that this naïve strategy becomes particularly inefficient for higher dimensions, in the worst case requiring $O(n\kappa)$ memory when $g > 0$, where κ represents the highest order k-fold intersection in the cover. For covers of the type shown by equation 1, κ is on the order of k^d, where d is the dimension of the filter space.

We can take advantage of the assumptions we've made about the family covers to reduce this storage requirement. Consider two covers, each composed of k intervals, where the length of every interval in the first cover is r and in the second r', where $r' > r$. Denoted the two covers as $\mathcal{U}_r$ and $\mathcal{U}_{r'}$ respectively. Since $r' > r$, assuming the intervals are always positioned equally distance from each other, the intervals $U_\alpha \in \mathcal{U}_{r'}$ properly contain the intervals in $\mathcal{U}_r$, and a map of covers may be formed to obtain the multiresolution structure discussed in Section 2.3. This structure shares many characteristics with the principle of *decomposability* often used in many spatial indexing methods. A problem is said to *decomposable* if its solution

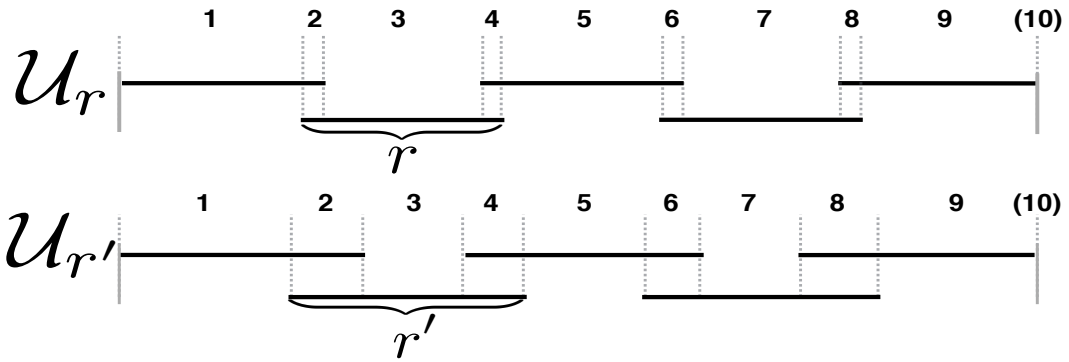

Figure 2: Example of the notion of standard ranges and their corresponding canonical covers between two covers $\mathcal{U} = \mathcal{U}_r$ and $\mathcal{V} = \mathcal{U}_{r'}$ of some space Z. In this figure, each cover consists of $k = 5$ intervals, and thus $2k$ segments, with the last segment always being trivial. The indices corresponding to segments are shown above each standard range, delimited by the dashed gray vertical lines. Note how any given pair U_α and $V_{\xi(\alpha)}$ satisfying $U_\alpha \subseteq V_{\xi(\alpha)}$ may be expressed using the same canonical cover, i.e. as the union of the same set of segments. For example, the second interval of both covers may be expressed as the union of segments $\{S_2, S_3, S_4\}$.

can be efficiently calculated by breaking the problem input into small, overlapping subsets, computing the partial solutions for these subsets, and then recombining to form the final solution. In what follows, we show how one may decompose $\mathcal{L}(\mathcal{U}_{\zeta(r)}, Z)$ using principles inspired by geometric range searching.

In order to decompose $\mathcal{L}(\mathcal{U}_{\zeta(r)}, Z)$, for now let $Z \subset \mathbb{R}$ (i.e. $d = 1$). Each cover interval U_α has an associated set of endpoints $[a, b], a, b \in \mathbb{R}, a \leq b$. If there are k such intervals, there are $2k$ endpoints. Now consider the sorted array of non-decreasing endpoints $\mathcal{E}$ composing $\mathcal{U}$. Since $\mathcal{E}$ is fixed, the covers in $\mathcal{U}$ may be indexed by an integer range $[s, t], s < t$. We use the following definition:

Canonical Covering. *The interval cover $U_\alpha = [a, b]$ is in the* canonical covering *of the range $[s, t]$, where $s, t \in \mathbb{Z}^+$ if $[a, b] \subseteq [\,\mathcal{E}[s], \mathcal{E}[t]\,]$.*

It can be shown that any range $[s, t], 1 \leq s < t \leq 2k$ may be decomposed into at most $\lceil \log_2(t - s) \rceil + \lfloor \log_2(t - s) \rfloor - 2$ subintervals [18]. Now consider any two covers $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ and $\mathcal{V} = \{V_\beta\}_{\beta \in B}$ of a space $Z \subset \mathbb{R}$ connected by a map of covers $\xi : A \to B$ from $\mathcal{U}$ to $\mathcal{V}$ such that $U_\alpha \subseteq V_{\xi(\alpha)}$ for all $\alpha \in A$. Assume both covers $\mathcal{U}$ and $\mathcal{V}$ satisfy $0 < g < \frac{1}{2}$, i.e. the interval lengths of each cover are restricted such that no non-trivial threefold intersections are observed, and are parameterized by interval lengths r and r', where $r' > r$, denoted as $\mathcal{U}_r$ and $\mathcal{V}'_r$, respectively. Then a pair of intervals from $\mathcal{U}_r$ and $\mathcal{V}'_r$ mapped by ξ have equivalent canonical covers (see a proof in appendix 1). Figure 2 illustrates this concept.

The equivalency of the canonical covers for varying interval covers enable a decomposition of the covers into indexed canonical covers, which allows the entire $\mathcal{L}(\mathcal{U}_{\zeta(r)}, Z)$ to be expressed more compactly. Going forward, we'll refer to the disjoint ranges in the canonical cover of $\mathcal{U}$

as *segments*, indexed by F, i.e.

$$\mathcal{S} = \{S_i\}_{i \in F} = \left\{ \left[\mathcal{E}[s], \mathcal{E}[t] \right) \mid s, t \in F \right\}$$

We can represent any range in a given cover $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ as a union of disjoint segments by a map between the index set of the cover $A = \{1, 2, \ldots, k\}$ and the index set of the segments $F = \{0, 1, 2, \ldots, 2k-1\}$ as $\omega : A \to F$, where $\omega(\alpha) = \{i \subseteq F \mid S_i \cap U_\alpha \neq \emptyset\}$. The level sets can be dynamically expressed via ω:

$$\mathcal{L}_\alpha(U_\alpha, Z) = \left\{ \bigcup_{i \in \omega(\alpha)} S_i \subseteq \mathcal{S} \right\}$$

In summary, instead of creating lists to represent the level sets for each cover in the sequence, we simply express the level sets as a union its corresponding segments. Since these segments are disjoint, and since each point into exactly one segment, there is no redundancy in this representation of the level sets.

Since the level sets are no longer stored explicitly, each level set must be dynamically computed via ω. This requires a bit of index tracking: we represent $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ with an integer indexed set U_i, $i \in \mathbb{Z}^+$, and then map that index to a pair of ordered segment indices $(\omega_l(i), \omega_h(i))$ representing the lower and upper endpoints that enclose the canonical cover of U_i. Due to the reflective symmetry assumption we make regarding the cover, there are only k such maps, for all $g \in [0, 1)$. To see this, observe that when $g = 0$, the canonical cover of any interval U is simply the union of two immediately adjacent segments, i.e. the two endpoints the interval corresponds too. As shown previously, this mapping will also be constant for $0 < g < \frac{1}{2}$.

Furthermore, observe that the left-and-right-most endpoints $\mathcal{E}$ maintain their positions in the ordering—all other endpoints swap their relative positions by 1. To keep the mapping constant for any g, we need only track the order the endpoints follow for varying g. As $g \to 1$, this relative ordering of $\mathcal{E}$ changes just k times, implying these indices may be precomputed and stored in a $(k \times 2k)$ lookup table, based on the value of g. As a result, up to a reindexing of the order of the segments, the map $A \to F$ is given by $[2i, 2i+1)$ where $i \in [0, k-1]$ can be fixed, allowing retrieval of which segment indices comprise each level set in constant time. Here is an example:

Endpoints Ordering Example

Consider the case where there are $k = 3$ intervals $\{U_1 = [a_1, b_1], U_1 = [a_2, b_2], U_1 = [a_3, b_3]\}$, and $d = 1$. When $g = 0$, the endpoints of the intervals follow the order $\{a_1, b_1, a_2, b_2, a_3, b_3\}$. When $g \in (0, 1/2)$, they follow the order $\{a_1, a_2, b_1, a_3, b_2, b_3\}$, and when $g \in [1/2, 1)$, they follow $\{a_1, a_2, a_3, b_1, b_2, b_3\}$. Since the intervals have reflective symmetry, internal endpoints always 'swap' in ordering with the interval that is immediately adjacent. Since there are always $2k$ endpoints and each 'swap' corresponds to 2 endpoints changing positions, there are only k ranges of g where the ordering changes. Table 1 illustrates this process in-depth.

In summary, by decomposing the cover into disjoint segments, each of the n points of Z is associated with exactly one of $2k$ segments, and tracking which segments map to the

Overlap range	$\mathcal{S}$ ordering						U_1		U_2		U_3	
	S_1	S_2	S_3	S_4	S_5	S_6	$\omega_l(1)$	$\omega_h(1)$	$\omega_l(2)$	$\omega_h(2)$	$\omega_l(3)$	$\omega_h(3)$
$g = 0$	0	1	2	3	4	5	0	1	2	3	4	5
$g \in (0, 1/2)$	0	2	1	4	3	5	0	2	1	4	3	5
$g \in [1/2, 1)$	0	2	4	1	3	5	0	3	1	4	2	5

Table 1: Supplementing the example below, two $(k \times 2k)$ tables depict the order the segments and their corresponding maps follow when $k = 3$. The first table shows which range of g the rows correspond too. The second shows the order the segments follow, based on their initial ordered assignment when $g = 0$. The third shows the relative order the segments from the previous table follow, which allow the mapping $\omega(i) = [2i, 2i + 1)$ to stay intact. Note the second table is for illustrative purposes and does not need to be directly computed.

appropriate interval in the cover requires looking up the appropriate row in a precomputed $(k \times 2k)$ table, whose values depend on g. Thus, the storage required to represent the cover $\mathcal{U}_r$ for any $r \in [\bar{r}, \infty)$ as a union of segments reduces to $O(n + 2k + 2k^2) = O(n)$ since k is constant, assuming that the amount of data n exceeds the number of segments.

4.3. Computing Point Assignments as r Changes. The segment each *point* is assigned to depends on the interval length of the cover, which is expected to vary when computing *mappers* at multiple scales. We discussed how we may compute the discrete set of interval length values Σ wherein the *mapper* changes in Section 4.1, and that if we compute all such values, then $|\Sigma| = m = n(k-1)$. We also discussed how we may use Σ to create an indexable function ζ which can produce any distinct cover $\mathcal{U}_{\zeta(r)}$ for $r \in \mathbb{R}^+$. With a slight abuse of notation, denote the cover at index p as $\mathcal{U}_p = \mathcal{U}_{\zeta(\epsilon_p)}$, where $\epsilon_p \in \Sigma$, and $p \in [1, m]$. Each ϵ_p carries with it specific information: it represents the minimal interval length wherein between $\epsilon_o < \epsilon_p < \epsilon_q$, $\forall\ o, p, q \in [1, m]$, each cover $\mathcal{U}_o, \mathcal{U}_p, \mathcal{U}_q$ is distinct, and if $p = o+1 = q-1$, then the level sets of each successive cover differ by exactly one point. Since these values are minimal, intuitively they represent the critical values wherein a given point $z \in Z$ lies exactly on one of the boundaries, or endpoints, of two overlapping intervals at parameterization ϵ_p. That is, if one has the points assigned to segments corresponding to $\mathcal{U}_p$ computed and wishes to compute level sets $\mathcal{U}_q$, where $q = p + 1$, then the only modification that needs to be performed is the reassignment of a single point to a new segment.

In what follows we assume one has a cover already computed at some index p, and is concerned with computing the cover at some target index q, again where $p, q \in [1, m]$. Denote with $\alpha_p^+, \alpha_p^- \in A$ the adjacent interval indices the point z lies on at index p, where $\alpha_p^+ > \alpha_p^-$. For all values where $p < q$, the critical values of r always occur at the boundary of the intervals composing $\mathcal{U}$, and the segments associated with either α_p^+ or α_p^- can be inferred from the table discussed in the previous section. To store these changes, observe that between $r_p = 0$ and $r_q = \infty$, each point will either merge to or be removed from at most $k - 1$ sets in the cover. We refer to this length $k - 1$ ordered sequence of (interval) indices a given point $z_i \in Z$ adheres to its *indexed path*, and denote it as $\mathcal{P}_{ij}$, where the first index i denotes the points index ($i \in [1, n]$), and the second index j denote the points current index into its $k - 1$-length path. Observe the number of unique index paths followed by

any given point will be relatively small, depending on k. If a point $z \in Z$ intersecting the interval U_k will intersect the upper endpoint of the interval U_{k-1} at the length r_p, then by our reflective symmetry 4 assumption about the sizes of the intervals in the cover, the point must intersect U_{k+1} as r increases before it intersects U_{k-2}. Since there are only 2 possible paths any given point intersecting an internal interval will follow, and there is just 1 ordered path followed by points in the outside intervals, then if the cover consists of k intervals, there are only $2(k-1)$ unique index paths. Furthermore, due to assumption (3) of the cover, the path can be calculated *a priori* based on which halfspace of the set each point lies in when $g = 0$, without computing the sequence indexed by ζ explicitly. An example is shown below:

Path example

Consider the case where one has a cover consisting of $k = 3$ intervals, i.e. $\mathcal{U} = \{U_1, U_2, U_3\}$, where each interval U_p has the range $[a_p, b_p]$, and $a_p < a_{p+1} \leq b_p < b_{p+1}$. If at $g = 0$ a given point z lies in U_1, and if each interval is uniformly parameterized by the same length r, then as $g \to 1$ the point must intersect U_2 first, and then U_3. The same situation occurs in reverse when z lies in U_3. If z lies in U_2, the path may be U_1, U_3 or U_3, U_1, depending on whether z lies to the left or to the right of the center of U_2. In total, there are 4 unique indexed paths that any point $z \in Z$ could follow: $\{\{1,2,3\},\{3,2,1\},\{2,1,3\},\{2,3,1\}\}$.

Thus, with a cover over $\mathbb{R}$, there are at most $2(k-1)$ unique paths taken by any point starting at $r = \bar{r}$, and as $r \to \infty$. For each point, we only need to store the following three pieces of information: (1) which of the $2(k-1)$ unique paths a the point follows; (2) an index into the points path representing its current position; and (3) the segment index the point is assigned to. By storing this information, we're able to track how two covers $\mathcal{U}_p \to \mathcal{U}_q$ differ. The third piece of information is slightly redundant in the sense that each point is already associated with a segment, but is useful for caching purposes to prevent an $O(n+k)$ point-segment correspondence lookup, implying the above point information requires $O(3n)$ storage. It's worth noting that these three pieces of indexing information may be stored contiguously, which is of practical importance for runtime concerns. Informally, we use this preprocessed information recording which interval(s) each point intersects at any given interval length value to minimize the number of operations needed to 'update' the *mapper* computed at a given index p to a *mapper* at a given q, discussed in the next section. Intuitively, the smaller $|p-q|$ is, the fewer number of operations are required to update the *mapper* object.

4.4. Computing the simplicial complexes. We use the indexing structures discussed above to move fluidly between the *mapper* at index p to the *mapper* at index q, i.e. to compute the transformation

$$N(f^*(\mathcal{U}_p)) \to N(f^*(\mathcal{U}_q))$$

More exactly, we would like to compute the transformation *efficiently*, such that the number of operation to obtain $N(f^*(\mathcal{U}_q))$ given $N(f^*(\mathcal{U}_p))$ is smaller than the number of operations to compute $N(f^*(\mathcal{U}_q))$ from scratch.

Consider one has a *mapper* built with a cover parameterized by r_p, and wants to compute a new *mapper* with a cover parameterized by r_q, where $p, q \in [1, m]$. As discussed in Section 4.1, if we treat Z as fixed, we can compute and store the values $\Sigma = \{r_1, r_2, \ldots, r_m\}$ ahead of time,

where at each index $i \in \{1, 2, \ldots, m\}$, exactly one point is changing segments (and as a result, only one level set is changing per index). If we process this sequence $i \in [p, p+1, \ldots, q], p < q$ in order, there are at most $c = |p-q|$ point-to-segment reassignments. The converse argument holds when $q > p$. Each index $\alpha \in A$ whose corresponding level set is modified in the sequence $[p, q]$ implies an update to the simplicial complex. Specifically each index $i \in [p, q]$ implies updating a k-simplex if the intersection of the intervals (α_i^+, α_i^-) corresponds to non-empty $(k+1)$-fold intersection. Between the indices $[p, q]$, we populate which indices have had their level set representation modified, denoted as α_{pq}. Since we are agnostic to the clustering algorithm used, between p and q at most $|\alpha_{pq}|$ indices will require their corresponding simplices to be updated, where $|\alpha_{pq}|$ is at most $|A| = k$. When the output is a graph $G(V, E)$, this essentially involves tracking which vertices and edges need to be recomputed or modified. To track the vertices that need to be updated, given the graph of a *mapper* at index p, $G_p(V, E)$, we maintain a $O(k + |V|)$-sized surjective map $\nu : A \to V$ matching which vertices belong to each index $\alpha \in A$ of the cover, and update $\nu(\alpha_{pq})$ accordingly. To track the edges, we record the pairwise combinations of indices $(\alpha_{pq}, \alpha'_{pq})$ that have overlapping intervals, as well as any additional pairs (α_i, α_j) connected by pre-existing edges in $G_p(V, E)$ associated with the vertices in changed by α_{pq}. Denote the edges changed in the sequence from $p \to q$ as $\varepsilon(\alpha_{pq})$. Since there are $\binom{k}{2}$ combinations, so $|\varepsilon(\alpha_{pq})|$ is (loosely) at most $O(|\nu(\alpha_{pq})|^2 + |E_p|)$ when $|r_p - r_q|$ is large, where E_p corresponds to the edges of $G_p(V, E)$, but in practice this is typically much smaller when the overlap is small. The complexity of these update operations depends on the structure used to store G as we discuss in Section 6.1.

4.5. Generalizing to higher dimensions. To generalize to higher dimensions, we take advantage of our assumption that the cover is composed of axis-parallel rectangles. As a result, with a few exceptions, we may treat each dimension independently. The only modification we need of the above are the complexities involving k. When the filter space $Z \subset \mathbb{R}^d$ for $d > 1$, the parameters of the cover are vector-valued. For example, the number of sets in the cover is parameterized by a vector $\boldsymbol{k} = (k_1, k_2, \ldots, k_d)$ representing the number of intervals to distribute along each dimension. In this case, there are $\tilde{k} = \prod_{i=1}^{d} k_i$ sets in the cover.

Although $\tilde{k}$ is exponential in d, because we consider a simple covering strategy where all the sets in the cover are axis-parallel, the complexities of most of the substructures our approach scales multiplicatively with d. For example, since the r parameterizations given by Σ are all based on orthogonal distances, $|\Sigma|$ need only be $\sum_{i=1}^{d} n(k_i - 1)$ rather than $n(\tilde{k} - 1)$.[8] We also require an indexing vector $\mathcal{I} = \text{order}(\Sigma)$ computed before Σ is sorted to track which points are involved with each $r \in \Sigma$ of the same size as Σ. When $n >> \tilde{k}$, these are the largest structures needed. To illustrate this more clearly, when k is constant across dimensions $|\Sigma|$ need be on the order of $O(nkd)$, as opposed to $O(nk^d)$. A similar argument can be made for most of the above structures. The exception to this rule is any stated complexity involving A (since $|A| = \tilde{k}$), and in the dynamic expression of $\mathcal{L}(\mathcal{U}, Z)$ via ω. The index set of the segments (F) is on the order of $|F| = O(2\tilde{k})$, which is a requirement to decouple the explicit storage of the level sets index by ζ, otherwise the equivalent naïve method discussed in section 4 would

[8] If the cover is guaranteed to be even simpler, e.g. there exists a rank-y decomposition of the $k-1$ interval parameterizations for each $z \in Z$ where $y < k - 1$, this can obviously be reduced further. We present the general case here.

require $O(n\tilde{k})$, which is obviously intractable even for relatively small values of n and d. This has implications in tracking which segments are modified between two index parameters p, q. Specifically, determining which segment index each point is is assigned to between two indices p and q must be delayed until the path indexing information $\mathcal{P}$ has been updated for each point and in each dimension.

5. Algorithms. We now translate the formulations above into algorithms that construct *mappers* for varying r values. The first step is to produce the set of auxiliary data structures containing the appropriate information.

Algorithm 1 Preprocess data X into a filtered map of covers

Require: Filtered points Z, number of intervals k, cover function f
Require: $\omega(i) := [2i, 2i+1)$
1: **procedure** ENABLE MULTISCALE
2: $\quad \boldsymbol{A}_0 \leftarrow \phi_{\bar{r}}(Z)$
3: $\quad \hat{\boldsymbol{R}} \leftarrow f(\mathcal{D}, \bar{r}, \dots)$
4: $\quad (\Sigma, \mathcal{I}) \leftarrow (\hat{\boldsymbol{R}}_{\mathcal{I}}, \text{order}(\hat{\boldsymbol{R}}))$ $\quad O(2nk)$
5: $\quad \mathcal{S} \leftarrow \{\, \mathcal{L}_{\omega(\alpha)}(\mathcal{U}_{\bar{r}}, Z) \;\forall\; \alpha \in A \,\}$ $\quad O(n+2k)$
6: $\quad \mathcal{T} \leftarrow \text{precomputeSegmentTable}(\mathcal{S}_0,\, k)$ $\quad O(2k^2)$
7: $\quad \mathcal{P}_{\text{uniq}} \leftarrow \text{computeUniquePaths}(k)$ $\quad O(2(k-1))$
8: $\quad \mathcal{P} \leftarrow \text{computeIndexPaths}(\boldsymbol{A}_0,\, Z)$ $\quad O(3n)$
9: **end procedure**

Denote the map between the ranges of g to the row of $\mathcal{T}$ as $\tau : g \to Q$, $Q = \{1, 2, \dots, k\}$. These data structures enable the following algorithm:

Algorithm 2 Computes the *mapper* at a new length/overlap value

Require: $G_p(V, E) \leftarrow N(f^*(\mathcal{U}_p))$
Ensure: $G_p(V, E) \to G_q(V, E)$
1: **function** UPDATE MAPPER(g) ▷ User-supplied g
2: $\quad$ TODO
3: **end function**

6. Experimental Results. We demonstrate the performance of the indexing structures and algorithms by building *mappers* at multiple scales over three data sets whose sizes vary by roughly an order of magnitude each. As discussed in Section 1, Mapper is a general framework in that it can be computed with any continuous filter function. Common choices for the filter include height functions applied to specific dimensions of the data, eigenfunctions of various linear operators (e.g. PCA, laplacian eigenmaps, or other rotations in the classical factor analysis model), eccentricity functions, density or regression estimators, or other Morse-type functions that provide useful geometric information about the point cloud data. Generally speaking, as the *mapper* construction is highly dependent on the filter function used, the choice of filter depends on the goal of the analysis. Before illustrating the utility of our approach,

we discuss the data sets we performed experiments and some brief background on meaningful filter functions that were chosen.

Miller-Reaven Diebetes dataset. The Miller-Reaven diabetes data set consists of 145 records of patients who volunteered to participate in a study at the Stanford Clinical Research Center conducted to further understand the etiology of diabetes. The patients were classified into one of three classes based on the type of diabetes: *overt*, *chemical*, or *normal*. For each patient, there were five measurements taken: (1) relative weight, (2) fasting plasma glucose, (3) area under the plasma glucose curve for the three hour glucose tolerance test (OGTT), (4) area under the plasma insulin curve for the OGTT, and (5) steady state plasma glucose response. The goal of the study was to investigate what connections could be discovered to relate the observed variables to the patients classification. In Millers discussion [22], the dataset was explored using *projection pursuit* until a specific 3-dimensional projection was found that reasonably separated the classes naturally into two "flares" radiating from the normal population, one for the those in the overt stage of diabetes, and another for those in the latent (or chemical) stage. A picture of the artistic rendering of the data set, including the aforementioned flares, is given in Appendix **??**.

This particular data set has been previously analyzed with Mapper in [30] and [6] using *density estimation* and *eccentricity* functions. The eccentricity function is defined as:

$$\mathrm{Ecc}(x) = \left(\frac{\sum_{x_i \in X}^{n} d(x, x_i)^p}{n} \right)^{\frac{1}{p}} \tag{5}$$

for $1 \leq p \leq \infty$. The eccentricity function can capture simple geometric information about the data set. One of the often-mentioned benefits of the function is that one need not have a center (or centers) explicitly available, as the function is defined purely in terms of average distances (with the norm depending on p). Like many functions chosen as filters, the eccentricity of a data set is coordinate-independent, implying it is invariant under data rotations or translations.

The kernel density estimate (KDE) of a sample $X_1, X_2, \ldots, X_n$ of size n from a random variable the density f is defined as:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h} \right) \tag{6}$$

where the kernel function K satisfies $\int K(x)dx = 1$, and h is often referred to as the *bandwidth*, to which the resulting estimate is highly dependent on. Typical choices for K include Gaussian, Epanechnikov, exponential, cubic, or other kernels. The choice of bandwidth has been studied for decades, see [29] for a brief overview. As a interesting sidenote, if one chooses a cover consisting of sublevel sets as mentioned as option (1) in Section 2.2, it can be shown that the *mapper* construction encompasses a structure known as the *cluster tree* $\mathcal{C}(X, f)$ of a density f (where f is not necessarily estimated via KDE), which has also recently seen developments aided by tools from computational topology (see [16]). The cluster tree $\mathcal{C}(X, f)$ is defined succinctly as:

$$\mathcal{C}(X, f) := \{x : f(x) \geq \lambda\} \text{ for some } \lambda > 0 \tag{7}$$

where λ is a density threshold level used to partition the data. This speaks to the generality of Mapper framework.

World Values Survey dataset. The World Values Survey (WVS) is an international research project aimed at documenting and studying the sociological factors that influence peoples values and beliefs across the world. The study is conducted in longitudinal waves in around 100 countries which collectively make up approximately 90% of the worlds population. The study involves a common survey questionnaire conducted anonymously made up of over 250+ survey questions. For more information, the reader is referred to the organizations website.[9] We analyzed Wave 6 of the survey (conducted in 2012) for all $2,232$ U.S. respondents with Mapper. Within the U.S. respondent pool, we choose 28 interesting dimensions to look at with Mapper, including questions involving happiness and self content, political leanings, religious and other organization associations, gender, quality of health and healthcare, married and family status, wealth and income status, etc. The dimensions were chosen based on the interpretability of their correlation structure, their cultural significance, and out of the authors own personal interests.

We consider two eigenfunctions as our filters for the WVS data, each representing a rotation in the exploratory factor analysis (EFA) model. The factor analysis model is a statistical technique for exploring ways of modeling the variation observed in a given data set by a linear combination of unobserved or latent variables. Formally, the classical factor analysis forms a latent variable representation of a set of random variables $X = \{X_1, X_2, \ldots, X_p\}$ of the form:

$$X = \boldsymbol{L}S + \epsilon \tag{8}$$

where the p observed, correlated random variables X are expressed as a linear expansion of q uncorrelated, unit variance latent variables or "factors" S (where $q < p$). $\boldsymbol{L}$ is a $(p \times q)$ matrix of coefficients whose columns are referred to as the *factor loadings*, and ϵ are the uncorrelated error terms unique to each $X_i \in X$. In applied factor analysis, the factor loadings are often used to name and interpret the latent variables. Note that X can be expressed by any orthogonal $p \times p$ rotation matrix $\boldsymbol{R}$, i.e.

$$X = \boldsymbol{L}\boldsymbol{R}^\intercal \boldsymbol{R}S \tag{9}$$

when $\boldsymbol{R}\mathrm{Cov}(S)\boldsymbol{R}^\intercal = \boldsymbol{I}$, and so there are many such decompositions. Because of this, EFA is sometimes considered a subjective art, resulting in an significant amount of effort put forth in attempt to find rotations capturing models that are more meaningful or objective, according to some given criterion. For example, *Principle Components Analysis* (PCA) may be interpreted in the latent variable sense as a model that finds the sequence of "best" linear approximations of $\boldsymbol{X}$; that is, the PCA solution provides a rank $q < p$ truncated decomposition of 8 that approximates $\boldsymbol{X}$ in an optimal way. Another popular technique is *Independent Components Analysis* (ICA), which seeks to reconstruct the loadings $\mathcal{L}$ such that

$$I(\boldsymbol{A}^\intercal \boldsymbol{X}) \tag{10}$$

[9] See http://www.worldvaluessurvey.org.

is minimized, where I is the mutual information between the components of the latent variables,

$$I(S) = \sum_{j=1}^{p} H(S_j) - H(X) \tag{11}$$

In summary, for our purposes, these filter functions may be interpreted simply as rotations in the EFA model. PCA and ICA methodically find components that (1) maximize the largest sample variance among all linear combinations of the columns of X, called the *principle components*, or (2) minimize the mutual information between the estimated components such that the the components are [ideally] statistically independent, called the *independent components*. The reader is referred to Section 14.7 of [31] for more details on exploratory factor analysis. The WVS data has been analyzed several times with factor analysis methods, see e.g. [23] and references therein for an exemplary overview.

Torus dataset. One of the use cases for Mapper is its ability to represent a simplified representation of the data by encoding the structure of the filter f through the lens of Z. By interpreting the data X as samples from a low-dimensional manifold, observed in some high dimensional topological space, Mapper provides a natural tool for exploratory data analysis. For example, in the previous section, f was always expressed as a linear manifold capable of approximating the data "well", where the quality of the decomposition was measured by the interpretability and amount of variation explained by the underlying latent variables. In the context of data analysis, the natural workflow is to treat the observed higher dimensional data set as fixed, and then infer the low-dimensional representation. Here, in an experiment similar in spirit as was conducted in the original Mapper paper (see section 5.2 in [30]), we construct a synthetic experiment to show the converse situation where we know the shape of the low dimensional manifold (and the manifold is non-linear), and we seek to embed it in a higher dimensional space. We then test our ability to recover the shape of the low-dimensional manifold with Mapper.

Consider the task of of recovering the shape of a compact 2-dimensional manifold in $\mathbb{R}^3$. As a simple example, we generated a synthetic data set creating by drawing samples from a probability distribution on a simple 2-dimensional torus, defined as:

$$\mathcal{M} = \{[(R + r\cos(\theta))\cos(\psi), (R + r\cos(\theta))\sin(\psi), r\sin(\theta)]\} \tag{12}$$

where $0 \leq \theta, \psi \in [0, 2\pi]$ and $0 < r < R$. If our goal is inference, for theoretical reasons its advantageous to assume the data are uniformly distributed on $\mathcal{M}$, see e.g. [2]. It's tempting to use equation 12 to obtain points on $\mathcal{M}$ directly by sampling uniformly from θ and ψ, however this will *not* result in a uniform sampling on the manifold itself. Intuitively, such a sampling scheme would result in a higher density of points on the areas of the torus with relatively high curvature, while points sampled on the "outside" of the torus will haver a lower density. To adjust for this, Diaconis et. al [10] use tools from geometric measure theory to derive a general technique that reparameterizes [the density of] a given sampling scheme via a change-of-variables transformation in such a way that properly adjusts for the surface area over $\mathcal{M}$. The result is a simple rejection-sampling scheme which can be used to sample points

uniformly on $\mathcal{M}$. We use this approach to synthetically generate $10,000$ points uniformly distributed along the surface of a 2D torus (represented in $\mathbb{R}^3$). The torus had an inner radius of 6 (distance from the outside of tube to the center) and an inner tube radius of 3.

To embed the point cloud in a higher dimensional space, one may use the QR decomposition to represent an orthogonal basis for the column space of the point cloud. Keeping with notation, denote the uniformly distributed points along $\mathcal{M}$ as $\boldsymbol{Z} \subset \mathbb{R}^3$. We embedded $\boldsymbol{Z}$ in $\boldsymbol{X} \subset \mathbb{R}^{30}$ by padding dimensions $4-30$ of $\boldsymbol{Z}$ with 0s, creating a new data set $\hat{\boldsymbol{X}}$, and then applying a rotation via:

$$\boldsymbol{X} = \hat{\boldsymbol{X}}\boldsymbol{Q}\boldsymbol{R} \tag{13}$$

where $\boldsymbol{QR}$ represents the QR decomposition of a (30×30) matrix of normally distributed random deviates. Given $\boldsymbol{X}$, we may now properly utilize a number of projection techniques which depend a uniform distribution over $\mathcal{M}$ to test Mappers ability to recover the torus. The two high-efficiency algorithms we choose to use explicitly make this assumption on $\mathcal{M}$: Laplacian Eigenmaps [2] and the Uniform Manifold Approximation and Projection (UMAP) [20].

The Laplacian Eigenmap (LE) algorithm builds an approximation to the eigenmaps of the Laplace-Beltrami operator on manifolds using a (potentially weighted) Laplacian of a graph G with weights W_{ij} given by a kernel (e.g. a gaussian kernel $\exp(-\|x_i - x_j\|/\epsilon)$). G is typically constructed by considering adjacencies formed by (1) the intersection graph of ϵ-neighborhood of $\boldsymbol{X}$, or (2) the k-nearest neighbor graph of $\boldsymbol{X}$, with the former depending more on the geometry of $\boldsymbol{X}$ and sometimes difficult to parameterize, and the latter less dependent on the geometry but easier to specify. LE then finds an embedding $\boldsymbol{Z}$ by minimizing the objective:

$$\underset{\boldsymbol{Z}^\intercal \boldsymbol{D} \boldsymbol{Z} = \boldsymbol{I}}{\arg\min}\ \mathrm{tr}(\boldsymbol{Z}^\intercal \boldsymbol{L} \boldsymbol{Z}) \tag{14}$$

In essence, this minimizes $\sum_{i,j}(z_i - z_j)^2 W_{ij}$. The motivation is that if any two data points x_i and x_j are close in the ambient space, then the lower-dimensional points they map to should also be close.

The Uniform Manifold Approximation and Projection (UMAP) algorithm [20] is a high-efficiency manifold learning technique that also makes use of the uniformity assumption of the points on $\mathcal{M}$. Briefly, the central premise of the UMAP strategy for projection is that local neighborhood manifold approximations may be 'patched' together to create a topological representation of the ambient space. This topological representation may then be used to create a [topologically] equivalent lower dimensional projection under the enforced assumption that the data are uniformly distributed on the manifold.[10] UMAP then optimizes the layout of the resulting embedding by cross-entropy functional between each spaces fuzzy simplicial set representation. For more details, the reader is referred to the paper [20].

6.1. Experiments. Using the data sets and filters above, we ran various experiments using the indexing structure and algorithms described above to assess the computational savings

[10]Technically, UMAP does not assume a Riemannian metric is inherited from the ambient space, but rather seeks to find a metric that is uniformly distributed on the manifold by creating a discrete metric space for each point $x_i \in X$. See [20] for more details.

we achieve in computing multiple *mappers*. A table of various statistics related to the data sets and their corresponding Mapper parameter settings is given in Table 2. We implemented our indexing structure mentioned above in Rcpp [11], incorporated into an R package.[11]. To maintain an updated version of the simplicial complex, we store the adjacency relations in a Simplex Tree [4].

Table 2: Details on the filter functions and covering methods used

Data set	[n/D/d]	Filter	Num. Intervals	% non-empty
MR	145/6/1	Density estimate	4	100%
MR	145/6/1	Eccentricity	4	100%
WVS	2232/28/3	Principle Components	5^3	70.4%
WVS	2232/28/3	Independent Components	5^3	78.4%
Torus	≈10k/30/2	Laplacian Eigenmaps	20^2	98.6%
Torus	≈10k/30/2	UMAP	15^2	86.2 %

To test our indexed approach, we consider the scenario where one wishes to compute distinct *mappers* successively between overlap percentages. In practice, one is rarely concerned with *mappers* computed with $g > 1/2$, and in the interest of reporting fair and practical results we also constrain ourselves to this scenario.[12] In what follows, we test the efficiency of computing *mappers* for n equally spaced values of $g = 0$ and $g = 1/2$ (for varying n), as well the efficiency of computing the *exact mapper* sequences, i.e. all distinct mappers in range $g \in [0, 1/2]$. When $d = 1$, the "exact" approach corresponds to all distinct *mappers* in Σ. When $d > 1$, the number of distinct *mappers* is combinatorial in $\boldsymbol{g}$, and of course $\boldsymbol{g}$ is not well-ordered. To have a similar comparison as the 1-dimensional case, we construct a well-ordered sequence of parameters $\boldsymbol{G} = [\boldsymbol{g}_i, \boldsymbol{g}_{i+1}, \ldots, \boldsymbol{g}_k]$ (obtained from Σ) such that

$$\|\boldsymbol{g}_j\|_1 \leq \|\boldsymbol{g}_{j+1}\|_1 \; \forall \; j \in [i, k] \tag{15}$$

Equation 15 creates a well-ordered set of parameters $\boldsymbol{G}$ by ensuring each successive component between $\boldsymbol{g}_j$ and $\boldsymbol{g}_{j+1}$ is nondecreasing. As such, the "exact" sequence is always of size $O(md)$, where $m = |\Sigma|$.

Table 2 shows the parameters corresponding to the performed experiments. We include the size and dimensions of the data sets and filter values, respectively, as well as the total number of intervals for each method. For each filter, we set the number of interval to distribute as constant along every dimension. For example, we set $k = 5$ for both covers of the filtered WVS data, which results in a cover with 125 intervals total since the data is 3-dimensional. In fairness, we report the percentage of intervals which are non-empty when $g = 0$. It's well known that as the embedding dimension of a fixed data set increases, most of the space becomes "empty" in the sense that the data become increasingly sparse. One may easily devise pathological examples where one has a given even a single outlier that is so far from mass of the data that the majority of the data appears to be within a single interval, increasing the likelihood that lower values of g do not produce distinct *mappers*. In the interest of

[11] See: https://github.com/peekxc/Mapper

[12] The reader is welcome to test values for $g > 1/2$ using the provided source code. The code to generate the plots is also publicly available, and fully reproducible.

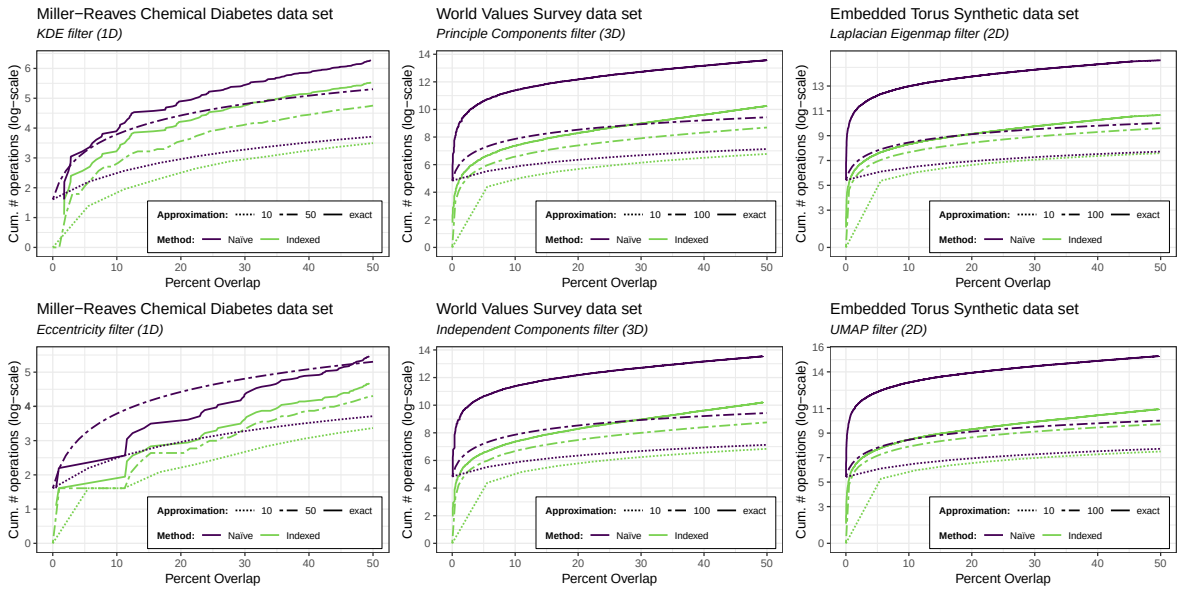

Figure 3: Cumulative number of number of 0-simplices that are recomputed when the clustering function is trivial, as a function of the percent overlap in the cover. The vertical axis is in log-scale. The colors denote the strategy used, and the line type denotes the number of successive *mappers* computed in sequence.

transparency, additional data-centric analysis has been performed and is available in the supplementary material.

The results of computing the discrete and exact *mappers* are given in Figures 3 and 3 for the data sets and their corresponding filters. In both figures, from left to right, the columns show the results for the two filter functions applied to the (1) Miller-Reaven Diabetes data set, (2) World Values Survey data set, and (3) the Synthetic Torus data set. In each plot, the horizontal axis depicts the overlap percentage[13], and the vertical axis tracks the cumulative number 'operations' needed provide to create the successive *mappers* in a log-scale. For Figure 3, the 'operations' are the cumulative number of indices $\alpha \in A$ which need to be recomputed (via partial clustering) to recover the *mapper*. In practice, the corresponding vertices also need to be replaced in the resulting simplicial complex, however the number of such vertices depends on the clustering algorithm (which we are agnostic to). As a result, we find it more informative to report the number of indices directly. Conceptually, this count is equal to the number of vertices that need to be recomputed when the clustering function is trivial in the sense that it always returns as a result a single connected component for each index $\alpha \in A$. We follow a similar scheme for Figure 4 by reporting the cumulative number of indexed *pairs* on the vertical axis (log-scale) that need to be recomputed. Conceptually, this count is equal to the number of edges that need to be recomputed when the clustering algorithm is trivial. Note that although the empirical counts seem similar and are indeed related, computationally recomputing a 'vertex' and an 'edge' are fundamentally two very

[13]Or, in the case where $d > 1$, the average of the the parameter vector $\boldsymbol{g}$.

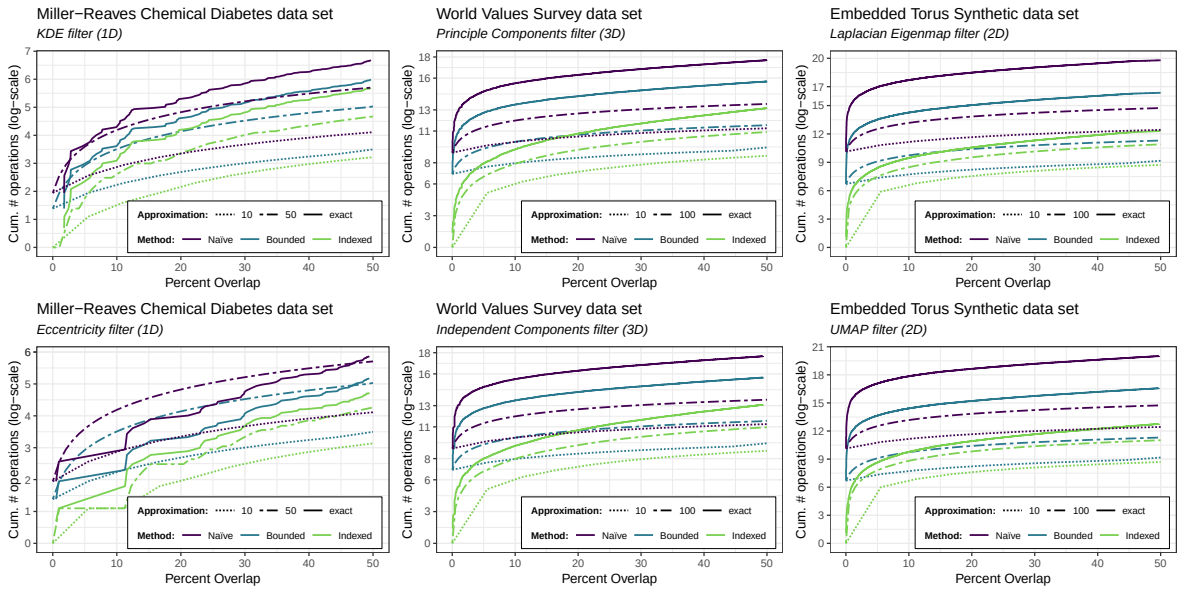

Figure 4: Cumulative number of number of 1-simplices that are recomputed when the clustering function is trivial, as a function of the percent overlap in the cover. The vertical axis is in log-scale. The colors denote the strategy used, and the line type denotes the number of successive *mappers* computed in sequence.

different operations: the former involves running a clustering algorithm on points in the data space (which often involves expensive distance calculations), and the latter involves checking for non-empty intersection between the vertices produced by the clustering (which scales exponentially with the intrinsic dimensionality of the space).

The varying types of lines in both figures depict the type sequence of *mappers* that was computed, where the 'type' is either a sequence of n successive *mappers* computed for equally spaced g values, and the 'exact' strategy is as previously discussed. The color of the lines depict the strategy that was used to compute the sequence of *mappers*. For Figure 3, there just two strategies: the traditional strategy we deem naïve, and our 'indexed' strategy. The naïve strategy involves recomputing the vertices at every interval in the cover to compute each successive *mapper* (for each overlap value g). This is conceptually equivalent to running Mapper independently at each value of a predetermined sequence of values $\boldsymbol{G}$. The indexed strategy uses the algorithms discussed in Section 5 to minimize the number of indices to reapply the partial clustering too. For Figure 4, there are three strategies: again the traditional or naïve strategy, the 'bounded' approach, and our indexed approach. The naïve approach involves checking for a non-empty intersection between all $\binom{k}{2}$ pairwise combinations of vertices, an approach that albeit is naïve in this context, is easy to implement and guaranteed to work for any type of cover. The 'bounded' approach denotes the approach that only checks for non-empty intersections between vertices that immediately adjacent. Intuitively, when $0 < g < 1/2$, each interval only intersects its immediate neighbors, and thus the pairs of these neighboring intervals are the only pairs that need to be considered. Only unique combinations of pairs are counted in Figure 4—there was no double-counting. The indexed approach

corresponds to our approach.

We can see from the results in Figures 3 and 4 that our approach compares favorably with the other approaches tested for data sets of varying size and dimension. As expected, the indexed approach requires less operations that the standard approaches mentioned, for both discrete sequences of size n and exact sequences. Some interesting artifacts to point out include the exact sequence requiring less pairwise index comparisons than the $n = 50$ sequence for the MR data set with the eccentricity filter for the naïve and bounded sequences, but requiring more (as expected) for the indexed approach. Since $d = 1$, the exact sequence would at most consists of 145 *mappers*. However, the eccentricity filter placed much of the data on the lower and upper halfspaces of the lowest and highest intervals, respectively. Because they lie on the boundary halfspaces of the cover, these points are exceptional cases in that they will not intersect any other intervals for $0 < g < 1/2$. As a result, the number of *mappers* for particularly low values of g (e.g. $g < 1/4$) is remarkably small, and since $k = 4$, only a few pairwise checks are needed to produce valid *mappers* relative to the $\binom{4}{2}$ checks that are needed for each g in the naïve approach. This artifact highlights an important aspect of our indexed approach though—the indexed approach will never require operations the other more standard approaches mentioned. At worst, the number of pairs of indices compares will match the bounded approach. This also becomes noticeable when the intervals are densely and uniformly packed, when the data set is large, when only a few *mappers* are requested. This is illustrated by considering the dashed $n = 10$ sequence computed over the Torus data set (for either 3 or 4). Since n is relatively large, the data are uniformly distributed, and the number of intervals is relatively small, computing a small number of mappers in a sequence (e.g. $g \in \{5\%, 10\%, \dots\}$ is comparable to simply running Mapper independently at each overlap value. However, the indexed approach still requires significantly less operations when computing the exact sequences. We mention in passing that increasing k and adding dimensions to the filter space will only exacerbate the performance for the naïve and bounded approaches, since the number of operations they both require is solely dependent on aspects of the cover itself, whereas the indexed approach depends more on aspects of the data.

7. Related Work.

Inverse Range Searching. The algorithmic problem of computing the level sets reminiscent of a class of *geometric range searching* problems well-known to computational geometry. If one has experience with these structures, one may wonder why they are not used instead, or why they are not discussed in detail here. We demonstrate that these class of range searching data structures, although inspirational in design, are fundamentally different than the one proposed here. Consider the generic geometric range problem, given as follows:

Design an efficient algorithm which, for a given *range* $R \in \mathcal{R}$, determines the set $X \cap R$.

where X is some set of points of interest, and $\mathcal{R}$ is a set of *range queries*. The point set of interest in our case are the filtered points, Z. Since we consider the point set Z as fixed, we may create a suitable data structure that stores auxiliary information about the space. The structure may then be repeatedly given ranges of the form:

$$[a_l^1, b_r^1] \times [a_l^2, b_r^2] \times \cdots \times [a_l^d, b_r^d] \tag{16}$$

where each range $[x_l^i, x_r^i], x_l^i, x_r^i \in \mathbb{R}, x_l^i \leq x_r^i$ for all $i = 1, 2, \ldots, d$ denotes an (generalized) interval. Collections of these intervals are often referred to as *range queries*, or simply *queries*. The goal is to solve the geometric range problem listed above as efficiently as possible for a given set of queries. For our application, this translates directly to the task of populating the level sets $\mathcal{L}(\mathcal{U}, Z)$.

When $d = 1$ the problem is easily solved in logarithmic time via a binary search on a sorted array. For $d > 1$, binary search trees augmented with some auxiliary information are often used. While tree-like data structures specialized for range searching (e.g. a kD trees, range trees, morton codes, etc.) are by now well-studied and highly accessible, and although dynamitization schemes do exist for these structures, they are primarily static structures designed to give efficient results at a specific localization of searching. An example of this would a preprocessed Z-order curve, or *morton code*. At the desired level of locality, such structures have proven quite efficient, however their efficiency degrades the more the locality of the queries differ from the locality of the space-filling design, an effect that only exaggerated as d grows.

A more appropriate data structure would be one that solves what some have called the *inverse range search* problem [27]: given a set $\mathcal{R}$ of (possibly) intersecting d-dimensional rectangles and a point $z_i \in Z$, determine all elements of $\mathcal{R}$ that intersect (or enclose) the point. Examples of data structures designed for this problem include the segment tree, the interval tree, etc. (see [18] for an overview). The generic inverse range problem is designed to address the task where the point queries may be unknown, however in our case, we have additional information: the "queries" are always correspond to the upper and lower bounds of the intervals composing the cover, and since we consider Z as fixed, such queries may be computed algebraically. Although conceptually the problem of computing the covers for varying r is similar to the dynamitized inverse range search approach, our approach is fundamentally a deterministic structure.

Mapper Parameter Selection. There have been recent efforts to alleviate the difficulty of properly parameterizing Mapper. Carriére et al. [6] investigated how to parameterize the Mapper by testing Mappers ability to recover the Reeb graph from a set of points sampled from a generative, stochastic process. Specifically, Carriére explored the problem of using Mapper to infer the Reeb graph, viewed from the statistical setting where the observations are assumed to be sampled from a generative process, i.e. the points $X = x_1, x_2, \ldots x_n$ are assumed to be i.i.d sampled from probability distribution on $\mathbb{R}^D$, where $X \subset \mathbb{R}^D$, and the distribution satisfies

$$\mathbb{P}(B(x,t)) \geq \min(1, at^b)$$

for some constants $a > 0, b \geq D$, where $B(x,t)$ is the euclidean ball centered at x with radius t. The cover they consider is what we refer to in the appendix as the "fixed" rectangular cover, and they specifically consider the connected components of the Rips complex for parameter δ when considering the connected components induced by the preimage of f. Under the perspective that the data come from a generative process, and with a few additional assumptions on the smoothness of the density, Carriére et al. studied how well Mapper captures the Reeb graph under different situations (exact Morse-type filter is specified *a priori* with known generative model, exact filter is specified with an unknown generative model, and an inferred filter

is used with an unknown generative model). He then proceeds to analyze Mappers ability to recover the Reeb graph for $g \in (\frac{1}{3}, \frac{1}{2})$ for varying resolutions (r), and they derive confidence sets for extended persistence diagrams generated from the Mappers using bootstrap. Their results suggest that Mapper is a minimax optimal estimator of the Reeb graph, under certain conditions on the regularity of the filter and the parameters of the generative model.

We see this probabilistic approach to computing a stable set of parameters for Mapper as an vital contribution toward simplifying the usability of the Mapper method, complementary to our own. Like their effort, we acknowledge the instability of the *mapper* construction with respect to its parameters. Whereas Carriére et al. avoid the intractability of computing multiple *mappers* by developing an algorithm that makes various assumptions on the filter function and the distribution of the data for selecting a viable set of (ideally) stable parameters, we make specific assumptions about the form of the cover to enable a much more tractable means of computing the *mappers* directly, independent of the filter used.

Multiscale Mapper. The primary contribution of this effort is an efficient way of computing distinct *mappers* for varying values of overlap. One of the applications of this is a means of computing discrete approximations to the so-called Multiscale Mapper studied recently by Dey et al. [8]. Before discussing this, we recall a number of definitions from their work.

Suppose we are given a map of covers, as defined previously in Section 2.3. Note that, just as we have a pullback cover of X induced by $\mathcal{U}$ via f with Mapper at a single resolution, given a map of covers ξ, there is a corresponding map of pullback covers of X, $f^*(\xi) : f^*(\mathcal{U}) \to f^*(\mathcal{V})$. Given such a map, there is an *induced* simplicial map $N(\xi) : N(\mathcal{U}) \to N(\mathcal{V})$, given on the vertices by the map ξ. It can be shown that induced maps are *contiguous*, and that they induce identical maps at the homology level, however we will not discuss this in detail, see Dey et. al [8] for more details.

Using these two notions, Dey et. al introduce the notion of a *tower* as a collection of objects connected by such maps, defined as follows:

Definition 7.1 (Tower). *A tower $\mathfrak{W}$ with resolution $r \in \mathbb{R}$ is any collection $\mathfrak{W} = \{\mathcal{W}_\epsilon\}_{\epsilon \geq r}$ of objects $\mathcal{W}_\epsilon$ together with maps $w_{\epsilon,\epsilon'} : \mathcal{W}_\epsilon \to \mathcal{W}_{\epsilon'}$ so that $w_{\epsilon,\epsilon} = \mathrm{id}$ and $w_{\epsilon',\epsilon''} \circ w_{\epsilon,\epsilon'} = w_{\epsilon,\epsilon''}$ for all $r \leq \epsilon \leq \epsilon' \leq \epsilon''$. Given such a tower $\mathfrak{W}$, $\mathrm{res}(\mathfrak{W})$ refers to its resolution.*

When $\mathfrak{W}$ is a collection of finite covers equipped with maps of covers between them, we call it a tower of covers. *When $\mathfrak{W}$ is a collection of finite simplicial complexes equipped with simplicial maps between them, we call it a* tower of simplicial complexes.

In the above definition, $\circ$ is used to denote a composition of maps. The interpretation of ϵ as a "courseness" parameter becomes clear from this definition. Letting X and Z be topological spaces and $f : X \to Z$ be a continuous map, given a tower of covers $\mathfrak{U}$, the *multiscale mapper* is defined to be the tower of simplicial complexes defined by the nerve of the pullback:

$$MM(\mathfrak{U}, f) := N(f^*(\mathfrak{U}))$$

Representing a sequence of simplicial complexes connected by simplicial maps. Passing these complexes to the homology functor $H_k(\cdot)$, $k = 0, 1, 2, \ldots$ with coefficients in the field leads to *persistence modules*, which may be summarized through the variety of tools being developed to summarize such maps, i.e. their associated persistence diagrams. Notably, one of the results

from [8] include, under mild assumptions on the map f, the ability to compute the exact persistence diagram from only the 1-skeleton of the complexes.

8. Conclusion and Future Work. In this paper, we've derived solutions to computing the discrete set of overlap parameters values which yield distinct Mapper constructions, and we introduced an algorithmic solution that uses these values to enables efficient computation successive *mappers*. We demonstrated the efficiency of the structure empirically, and we also introduced a way of representing *mappers* in a much more memory efficient way, for a fixed data set. We showed experiments with real-world data sets that illustrate the benefits of our approach in the exploratory/confirmatory analysis setting, and we also highlighted relationships between our work with recent theoretical work involving Mapper related to persistence and stability. We included a small discussion of how the solution provided enables further development in the application of topological data analysis to various theoretical and application areas. Finally, in the spirit of transparency and reproducibility, we publicly made available the scripts[14] and source code of our experiments in the form of a R package.[15]

There is much in the way of future work to explore, primarily with regard to the strength of our assumptions, and in exploring its theoretical connections to persistence more extensively. Perhaps the most critical assumption of our approach is that we always assume the number of intervals, k, is fixed prior to making the structure. A useful extension to this work would be a mean of adapting the structure to adjust for varying k without significantly compromising the storage or runtime complexities. Intuitively, all one needs a compressed way of representing, for varying k, the minimal settings wherein any arbitrary point intersects a new set in the cover. One idea that Carlsson pointed out in his seminal paper [5] (see page 35), one may use a map of integers $k \to \lfloor \frac{k}{2} \rfloor$ to define a map of coverings $\mathcal{U} \to \mathcal{V}$ wherein every interval in $\mathcal{V}$ contains two intervals from $\mathcal{U}$. This strategy could also be amended to wok with our structure for k varying by powers of two.

Another effort to look into is whether there exists a way to use the fact that we assume the cover is composed of axis-parallel sets to reduce the storage complexity even further. We make no conjecture that either the storage or computational complexities of our approach are optimal; we set out to create a solution specific to Mapper that was relatively efficient, generic enough to work with the most often used covering strategies in practice, and simple enough to easily extend to the multidimensional case. Although we emphasize the particular problem we address here is fundamentally different from the generic class of geometric range searching problems in the sense that the 'queries' are deterministic, calculated *a priori*, and form an inherent part of the structure, its possible that more efficient, dimension-specific adaptations of our approach may be derived from e.g. [19] to provide lower complexity bounds.

Appendix.

A. Proofs. In what follows we prove the canonical representation of the cover is equivalent for all covers $\mathcal{U}_r^*$ with interval lengths $r \leq r^* \leq r'$, where $r > \bar{r}$ and r' is the maximum interval length such that $g < \frac{1}{2}$. Before discussing the proof, recall that a partial order is an *interval*

[14] See https://github.com/peekxc/IndexedMapper

[15] The package is CRAN-pending, but available for installation via github: https://github.com/peekxc/Mapper

order if its elements can be assigned intervals on the real line so that $x < y$ if and only if the interval assigned to x is completely to the left of the interval assigned to y, and a partial order is a *semiorder* if its elements can be assigned numbers so that $x < y$ if and only if y's assignment exceeds x's assignment by no more than 1. Our proof summarizes an equivalent result involving the representation of intervals as interval graphs [3].

Proof. Assume $0 < g < \frac{1}{2}$. Construct an *interval graph* $G_{\epsilon'}(V, E)$ of $\mathcal{U}'_\epsilon$ that assigns to each vertex $v_\alpha \in V$ the range of U_α for all $\alpha \in A, A \subset \mathbb{Z}^+$. Since $r < \frac{1}{2}$, $\ell(U_\alpha) + \ell(U_\alpha \cap U_{\alpha+1}) < \frac{1}{2}\ell(U_\alpha)$ for all $\alpha \in A$, and there is no pair $v_x, v_y \in V$ such that (1) $U_y \subset U_x$ and (2) U_x intersects intervals to the left and right U_y that do not intersect U_y. Therefore no interval is properly contained in another, the intervals indexed by A follow a semiorder. If A is semiordered, there is no induced subgraph isomorphic to the bipartite 'claw graph' $K_{1,3}$.

This yields a *proper interval representation* of G, wherein the right endpoints have the same order as the left endpoints. As a result, the intervals in $\mathcal{U}$ may be reduced to unit length. Processing from left to right, at each step let $U_x = [a, b]$ be the unadjusted interval that has the leftmost left endpoint, and let $\alpha = a$ unless U_x contains the right endpoint of some other interval, in which case let α be the largest such right endpoint. Such an endpoint would belong to an interval that has already been adjusted to have length 1; thus $\alpha < \min\{a + 1, b\}$. Now, adjust the portion of the interval in $[\alpha, \infty)$ by shrinking or expanding $[\alpha, b]$ to $[\alpha, a + 1]$ and translating $[b, \infty)$ to $[a + 1, \infty)$. The order of endpoints does not change, intervals processed before U_x must have length 1, and U_x now also has length 1. Iterating this operation produces the unit interval representation. We conclude that since every proper interval graph is a unit interval graph, the canonical representation of the intervals $0 < g < \frac{1}{2}$ are equivalent. ∎

We extend this result to covers with $g \in [0, 1)$ (or $\bar{r} \leq r < \infty$) using our reflective symmetry assumption to track the order of interval endpoints for varying ranges of g. We recompute their corresponding canonical representations, and record them in a table, e.g. 1. See section 4.2 for more details.

B. Derivations. The hyper-rectangular covering method denoted by equation **??** requires as input the number of rectangles k to distribute along each dimension, and an overlap percentage g neighboring rectangles should intersect (where $g \in [0, 1)$). where each generalized interval has side lengths given by r, and the degree to which overlap is proportional to g. The precise way one indexes these parameters into a cover can vary. Below, we outline two ways often used in practice.

B.1. Restrained cover. One family hyper-rectangular covers requires as input the number of rectangles k to distribute along each dimension, and an overlap percentage g neighboring rectangles should intersect (where $g \in [0, 1)$). The cover method uses the range of Z to reparameterizing (k, g) into parameter (r, e), where r is the length of each interval and e is the step size between intervals, which is inversely proportional to g. We refer to this family as the *restrained* cover because the intervals at the ends of the cover are always restrained by the range of Z. As a result, although when $g = 0$ the intervals are equally spaced and equal length, as $g \to 1$ the centers of each intervals moves towards the center of the range of Z. In what follows, we derive how to reconstruct the discrete set of interval values r which produce distinct covers. Without loss in generality, we should the case where $d = 1$ to simplify the

notation.

One proceeds constructing the cover by first finding the range of the function f restricted to the set of given points Z. The range vector is the component-wise difference between them:

$$\bar{z} = z_{max} - z_{min}$$

Given a user-supplied input for g and k, a cover $\mathcal{U}$ is constructed by computing the *interval length* and *step size* parameters, respectively:

$$\begin{aligned} r &= \bar{z}^{-1}(k - g(k-1)) \\ e &= r(1-g) \end{aligned} \tag{17}$$

Notice that if k is fixed and $g = 0$, r is completely determined by the range the data, yielding the *base interval length*:

$$\bar{r} = k^{-1}\bar{z} \tag{18}$$

We refer to this length as the *base interval length*, and denote it with $\bar{r}$. It's clear that when $g > 0$, $r > \bar{r}$, otherwise the intervals would not contiguously cover the range of Z. Now since

$$r = \bar{r} + \epsilon$$

where $\epsilon \in \mathbb{R}$, our simple observation is that we may compute all possible distinct values of r by computing the distance $\mathcal{D}$ from each point to their corresponding $k-1$ intervals, and then adding this distance to the base interval length:

$$\hat{R} = \bar{r} + 2\mathcal{D} \tag{19}$$

Setting $\Sigma = \hat{R}$ gives all the parameterizations that produce distinct covers for this family of covers for a fixed k and Z, and a result, distinct *mapper* constructions.

One exploratory use case of this is as follows. Suppose a user, for example, wishes to compute the *mappers* for an ordered set $\boldsymbol{g} = \{g_1, g_2, \ldots, g_m\}$. Since e is dependent on g, it's clear that the only non-fixed variable $\boldsymbol{g}$ depends on is $\boldsymbol{r}$, i.e. $\boldsymbol{g}$ is computable if $\boldsymbol{r}$ is known, shown below:

$$\begin{aligned} r(k - g(k-1)) &= \bar{z} \\ rk - rgk + rg &= \bar{z} \\ g(-rk + r) &= \bar{z} - rk \\ g &= \frac{\bar{z} - rk}{r(-k+1)} \end{aligned} \tag{20}$$

We may convert the values of $\boldsymbol{g}$ to their corresponding intervals lengths $\boldsymbol{r}$ using 17, and then use the index map $\zeta(\boldsymbol{r})$ to compute the *mappers* incrementally using the indexed structure.

B.2. Fixed-centroid cover. Another family of hyper-rectangular covers that also parameterized by (k, g) is the fixed-center (or centroid) family of intervals. This method follows more closely with the second cover defined in Section 2.2, where $\mathcal{U}$ is decomposed into a collection of intervals with centers α, each of length 2ϵ. We refer to this family as the *fixed-centroid* cover because the centers of the intervals are fixed using the range of Z, and the only free parameter is ϵ. As before, we derive how to reconstruct the set which produces distinct covers in the $d = 1$ case.

Given parameters (k, g) and a set of points Z, calculate the base interval length as before:

$$\bar{r} = k^{-1}\bar{z}$$

In general, the centers of the intervals may be placed arbitrarily. We consider an equidistant placement of the centers α

$$\alpha = a_i + \frac{|b_i - a_i|}{2} \; \forall \; i \in 1, 2, \ldots, k$$

where k is fixed, and $[a_i, b_i]$ are given by the endpoints of the intervals constructed from the intervals in $\mathcal{U}_{\bar{r}}$. Once these centers are determined, ϵ is computed using g as follows:

$$\epsilon = 2^{-1}\left(\bar{r} + \frac{\bar{r}g}{1-g}\right)$$

To construct the cover, one iterates through through the k centers $\alpha \in A$ and constructs the resulting intervals as

$$U_\alpha = (\alpha - \epsilon, \alpha + \epsilon) \; \forall \; \alpha \in A$$

All possible distinct values of r are computed by adding this distance $\mathcal{D}$ from each point to their corresponding $k - 1$ intervals to the base interval length:

$$\hat{R} = \bar{r} + 2\mathcal{D} \tag{21}$$

Setting $\Sigma = \bar{R}$ gives all the parameterizations that produce distinct covers for this family of covers for a fixed k and Z, and a result, distinct *mapper* constructions. As before, the exploratory case is handled via a conversion from these lengths to the overlap parameters:

$$g = 1 - \frac{\bar{r}}{\hat{R}}$$

Acknowledgement. This work was supported by an appointment to the Internship/Research Participation Program at the U.S. Air Force Research Laboratory (AFRL), administered by the Oak Ridge Institute for Science and Education through an interagency agreement between the U.S. Department of Energy and Wright-Patterson Air Force Base.

REFERENCES

[1]
[2] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural computation, 15 (2003), pp. 1373–1396.
[3] K. P. Bogart and D. B. West, *A short proof that "proper= unit"*, Discrete Mathematics, 201 (1999), pp. 21–23.
[4] J.-D. Boissonnat and C. Maria, *The simplex tree: An efficient data structure for general simplicial complexes*, Algorithmica, 70 (2014), pp. 406–427.
[5] G. Carlsson, *Topology and data*, Bulletin of the American Mathematical Society, 46 (2009), pp. 255–308.
[6] M. Carriere, B. Michel, and S. Oudot, *Statistical analysis and parameter selection for mapper*, The Journal of Machine Learning Research, 19 (2018), pp. 478–516.
[7] V. De Silva and R. Ghrist, *Homological sensor networks*, Notices of the American mathematical society, 54 (2007).
[8] T. K. Dey, F. Mémoli, and Y. Wang, *Multiscale mapper: Topological summarization via codomain covers*, in Proceedings of the twenty-seventh annual acm-siam symposium on discrete algorithms, SIAM, 2016, pp. 997–1013.
[9] T. K. Dey, F. Mémoli, and Y. Wang, *Topological analysis of nerves, reeb spaces, mappers, and multiscale mappers*, arXiv preprint arXiv:1703.07387, (2017).
[10] P. Diaconis, S. Holmes, M. Shahshahani, et al., *Sampling from a manifold*, in Advances in Modern Statistical Theory and Applications: A Festschrift in honor of Morris L. Eaton, Institute of Mathematical Statistics, 2013, pp. 102–125.
[11] D. Eddelbuettel and R. François, *Rcpp: Seamless R and C++ integration*, Journal of Statistical Software, 40 (2011), pp. 1–18, https://doi.org/10.18637/jss.v040.i08, http://www.jstatsoft.org/v40/i08/.
[12] K. J. Emmett and R. Rabadan, *Characterizing scales of genetic recombination and antibiotic resistance in pathogenic bacteria using topological data analysis*, in International Conference on Brain Informatics and Health, Springer, 2014, pp. 540–551.
[13] R. Ghrist, *Barcodes: the persistent topology of data*, Bulletin of the American Mathematical Society, 45 (2008), pp. 61–75.
[14] R. W. Ghrist, *Elementary applied topology*, Createspace, 2014.
[15] C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth, *A survey of topology-based methods in visualization*, in Computer Graphics Forum, vol. 35, Wiley Online Library, 2016, pp. 643–667.
[16] K. Jisu, Y.-C. Chen, S. Balakrishnan, A. Rinaldo, and L. Wasserman, *Statistical inference for cluster trees*, in Advances in Neural Information Processing Systems, 2016, pp. 1839–1847.
[17] M. Kramar, A. Goullet, L. Kondic, and K. Mischaikow, *Persistence of force networks in compressed granular media*, Physical Review E, 87 (2013), p. 042207.
[18] D. Lee, *Interval, segment, range and priority search trees*, 2005.
[19] J. Matoušek, *Geometric range searching*, ACM Computing Surveys (CSUR), 26 (1994), pp. 422–461.
[20] L. McInnes, J. Healy, N. Saul, and L. Grossberger, *Umap: uniform manifold approximation and projection*, The Journal of Open Source Software, 3 (2018), p. 861.
[21] S. MERKULOV, *Hatcher, a. algebraic topology (cambridge university press, 2002)*, Proceedings of the Edinburgh Mathematical Society, 46 (2003), pp. 511–512.
[22] R. G. Miller, *Discussion: Projection pursuit*, The Annals of Statistics, 13 (1985), pp. 510–513.
[23] M. Minkov and G. Hofstede, *Hofstedes fifth dimension: New evidence from the world values survey*, Journal of cross-cultural psychology, 43 (2012), pp. 3–14.
[24] E. Munch, *A users guide to topological data analysis*, Journal of Learning Analytics, 4 (2017), pp. 47–61.
[25] E. Munch and B. Wang, *Convergence between categorical representations of reeb space and mapper*, arXiv preprint arXiv:1512.04108, (2015).
[26] J. R. Munkres, *Topology*, Prentice Hall, 2000.
[27] M. H. Overmars, *The design of dynamic data structures*, vol. 156, Springer Science & Business Media, 1987.
[28] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical

Computing, Vienna, Austria, 2013, http://www.R-project.org/.
[29] S. J. Sheather, *Density estimation*, Statistical science, (2004), pp. 588–597.
[30] G. Singh, F. Mémoli, and G. E. Carlsson, *Topological methods for the analysis of high dimensional data sets and 3d object recognition.*, in SPBG, 2007, pp. 91–100.
[31] H. Trevor, T. Robert, and F. JH, *The elements of statistical learning: data mining, inference, and prediction*, 2009.
[32] Y. Yao, J. Sun, X. Huang, G. R. Bowman, G. Singh, M. Lesnick, L. J. Guibas, V. S. Pande, and G. Carlsson, *Topological methods for exploring low-density states in biomolecular folding pathways*, The Journal of chemical physics, 130 (2009), p. 04B614.

Supplementary material.

Mapper Ring Data Example. To further illustrate `Mapper`, consider the example given by Singh *et. al* in [30], shown in Figure 5. The left figure displays the ring data X in $\mathbb{R}^2$. The filter function f used computes the distance of every point $x \in X$ to the left-most point p, resulting in a filter space in $\mathbb{R}$. This filter space is shown in the center figure, from which a covering is constructed of 5 sets, each with 20% overlap (the points lie in the filter space are not shown). The right figure shows the resulting 1-skeleton of the constructed simplicial complex, represented as a graph G.

Torus Example. Torus explanation.

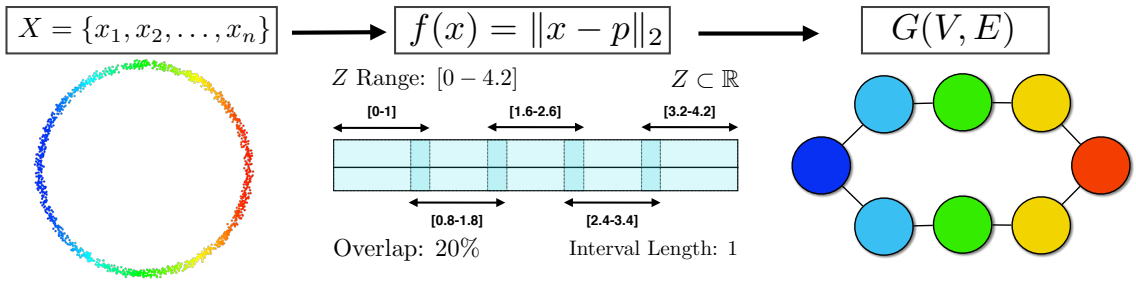

Figure 5: Mapper ring data example discussed in Example 1.1. In this example, $Z \in \mathbb{R}$ represents the distance from each point $x_i \in X$ to the point with the smallest coordinate in $X^{(1)}$, i.e. the "left-most" point. Note how the three inner intervals resulted in two clusters each, which connect at each end point to a single cluster in the first and last intervals, forming a ring.

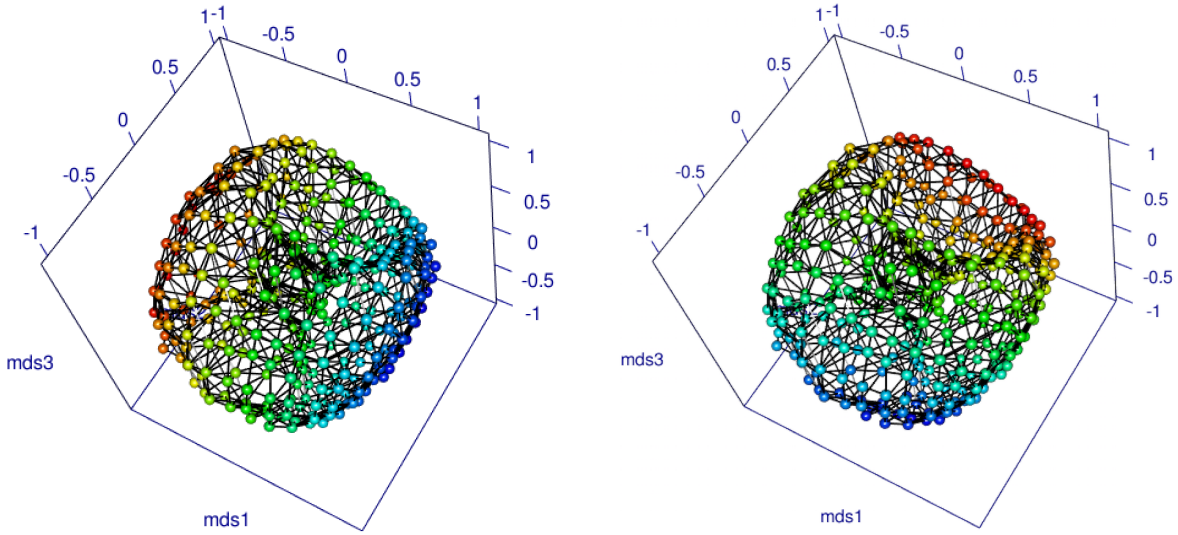

Figure 6: The mapper was built by projecting the torus data $X \subset \mathbb{R}^{30}$ to $Z \subset \mathbb{R}^2$ with UMAP. Node positions were estimated using multidimensional scaling on a smooth approximation of haussdorff distance. The colors of the nodes reflect first and second dimensions of the filter, respectively.

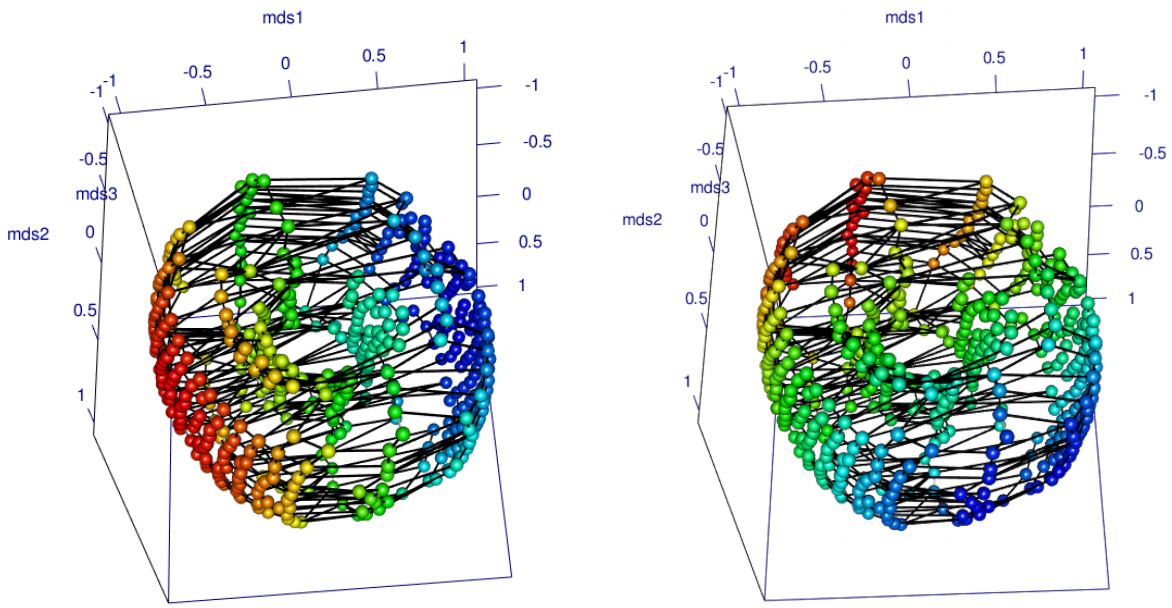

Figure 7: The mapper was built by projecting the torus data $X \subset \mathbb{R}^{30}$ to $Z \subset \mathbb{R}^2$ with Laplacian eigenmaps. Node positions were estimated using multidimensional scaling on a smooth approximation of haussdorff distance. The colors of the nodes reflect first and second dimensions of the filter, respectively.