ANALYZING SCALAR FIELDS THROUGH TOPOLOGICAL SIMILARITY

BRIAN BOLLEN

Abstract. Reeb graphs have been used in a number of applications including analyzing the topological properties of unifaceted scalar fields. It is natural to consider the situation of analyzing multiple scalar fields arising from multifacted data via Reeb graphs. The interleaving distance on Reeb graphs has a rich, mathematical foundation and has been shown to provide a more sensitive metric than other common metrics such as bottleneck distance. Here, we provide the construction for interleaving distance as well as an overview of the complexities encountered when trying to compute it. We also provide several further directions for research which include using machine learning in various ways to either create similarity metrics between Reeb graphs or apply these machine learning techniques to aid in the computation of interleaving distance and other metrics.

.

1. Introduction

1.1. **What is a Scalar Field?** Suppose we want to measure the temperature of the surface of the Earth over a large region. We can think of this region as simply being a two-manifold ignoring where each point is its geographic coordinates. Denote this space as $\mathbb{X}$. We construct a function $f : \mathbb{X} \to \mathbb{R}$ which maps from this two-manifold to the real numbers by associating the coordinate pair with the corresponding temperature. The pair $(\mathbb{X}, f)$ is called a *scalar field*. We can often visualize well-behaved scalar fields as some sort of mountainous or hilly terrain. Just as we can use contour lines on a map to indicate regions that are the same elevation, we can use *level sets* to study the subset of $\mathbb{X}$ where all the coordinates have the same temperature. Formally, we have the following definitions:

Definition 1.1. Let $\mathbb{X}$ be a topological space and $f : \mathbb{X} \to \mathbb{R}$ be a scalar function. We call the pair $(\mathbb{X}, f)$ a *scalar field*.

Definition 1.2. Let $U \subseteq \mathbb{X}$ such that $f(x) = f(y) = a$ for each $x, y \in U$. We call the subset U a *level set* or *fiber* of $\mathbb{X}$. More specifically, we sometimes use the terms a-level set or a-fiber when the actual value a is significant.

When $\mathbb{X}$ is an n-dimensional space, we say that $(\mathbb{X}, f)$ is an n-dimensional scalar field. If $(\mathbb{X}, f)$ is a 1-dimensional scalar field and f is a differentiable function, then we say that $(\mathbb{X}, f)$ has a *critical point* if the first derivative is equal to 0. To generalize this to n-dimensional scalar fields, we have the following definition:

Definition 1.3. Let $(\mathbb{X}, f)$ be an n-dimensional scalar field and $a = (a_1, \ldots, a_n) \in \mathbb{X}$. If $\nabla f(a) = 0$, then we say that a is a *critical point* of $(\mathbb{X}, f)$. We call any fiber of $(\mathbb{X}, f)$ that contains a critical point a *critical fiber*. Similarly, a *non-critical fiber* is any fiber that does not contain any critical points.

Going back to the 1-dimensional case, we say that the critical point a is a *minimum* if the second derivative at the point is positive (convex). Analogously, a critical point is a *maximum* if the second derivative at the point is negative (concave).

Again, generalizing this situation to multi-dimensional scalar fields, we introduce the *Hessian Matrix*, denoted simply as $\mathbf{H} = \mathbf{H}(x)$ which is a $n \times n$ matrix where $a_{i,j} = \frac{\partial f^2}{\partial x_i \partial x_j}(x)$, where $a_{i,j}$ refers to the entry in the i^{th} row and j^{th} column. Suppose $a \in \mathbb{X}$ is a critical point of $(\mathbb{X}, f)$. That is, the gradient evaluated at a is equal to 0. Then, if all of the eigenvalues of $\mathbf{H}(a)$ are negative, then a is a maximum, while if all of the eigenvalues of $\mathbf{H}(a)$ are positive, then a is a minimum. Essentially, the signs of the eigenvalues are telling us in how many directions the function is concave (negative eigenvalues) or convex (positive eigenvalues).

However, this situation only arises when we are dealing with *non-degenerate* critical points. A non-degenerate critical point a is any critical point such that $\det(\mathbf{H}(a)) \neq 0$. That is, the Hessian matrix evaluated at a is non-singular. This situation occurs specifically when the neighborhood around the value a is dominated by quadratic terms.

Lemma 1.4 (Morse Lemma)**.** *Let u be a non-degenerate critical point $f : \mathbb{M} \to \mathbb{R}$. There are local coordinates $u = (0, 0, \ldots, 0)$ such that*

$$f(x) = f(u) - x_1^2 - \ldots x_q^2 + x_{q+1}^2 \ldots x_d^2$$

for every point $x = (x_1, x_2, \ldots, x_d)$ in a small neighborhood of u.

Definition 1.5. A *Morse Function* is a smooth function on a manifold $f : \mathbb{M} \to \mathbb{R}$ such that all critical points are non-degenerate and all critical points have distinct function values.

Due to critical points having distinct function values, this guarantees that a critical fiber does not contain infinitely many critical points (i.e. there are no "plateaus" of the scalar field).

Definition 1.6. Let $f := (\mathbb{X}, f)$ be an n-dimensional scalar field and let $a \in \mathbb{X}$ be a critical point of f. Let $m \leq n$ be the number of negative eigenvalues of $\mathbf{H}(a)$. Then we say that m is the *index* of the critical point a. Thus, if $m = n$, then a is a maximum and if $m = 0$, then a is a minimum. If $0 < m < n$, then we say that a is a *saddle.*

1.2. Topological Evolution of a Scalar Field.

When studying scalar fields, we are not only concerned with the size or magnitude of the scalar field, nor the particular positions of its critical points. Often we are concerned with its topological properties. Throughout the years, multiple different tools have been created for studying the topological properties of data such as persistence modules [32], persistence diagrams [9, 13], barcodes [16], the Morse-Smale complex [11, 17], and, in particular, the Reeb graph [5, 10] (as well as its descendant the contour tree [7]). The Reeb graph summarizes the topological evolution of the level sets of a scalar field. It particularly tracks what occurs at critical fibers, which we will call

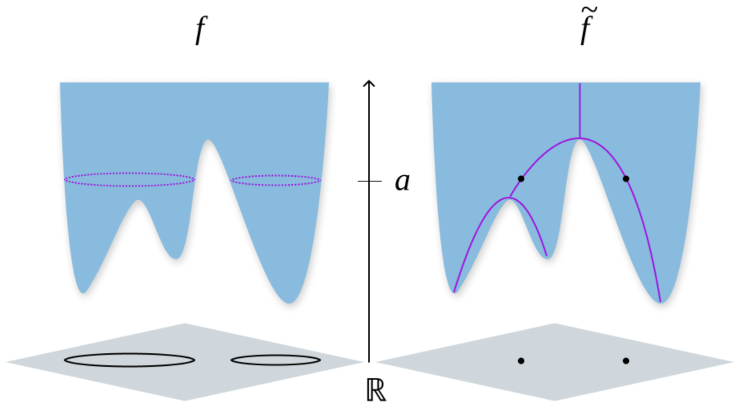

FIGURE 1. On the left is a two dimensional scalar field. Depicted on the scalar field is the a-fiber. On the right is the same scalar field with the Reeb graph superimposed on top. From the left picture, we see that the a-fiber has two path connected components. Thus, the two path connected components are displayed as two distinct points in the Reeb graph on the right.

"events".

Definition 1.7. Let $(\mathbb{X}, f)$ be a scalar field. We can define an equivalence relation on $\mathbb{X}$ by defining $x \sim y$ if $f(x) = f(y) = a$ (they belong to the same a-fiber) and if x, y are in the same path connected component of the a-fiber. Then, the quotient space $\mathbb{X}_f := \mathbb{X}/\sim_f$ is called the *Reeb graph* of the scalar field $(\mathbb{X}, f)$. We equip the Reeb graph with the function $\tilde{f}$ which is defined as $\tilde{f} \circ \rho = f$, where $\rho : \mathbb{X} \to \mathbb{X}_f$ is the quotient map defined by our equivalence relation.

The Reeb graph tracks the evolution of the levelset topology of the scalar field. It shows how the fibers of the scalar field split and merge. Figure 1 provides an example of a scalar field along with its Reeb graph.

It is not immediate that this definition of a Reeb graph actually creates a graph. But from Figure 1, we can see that we can construct a graph from the information we are given. If we label the critical points from the scalar field on the actual Reeb graph, we will see that all critical points occur where two components merge into one, or where a component begins or ends. It is easy to see that these critical points can then define the vertices of the Reeb graph, where edges correspond to connections between critical points based on the level sets. But more importantly, the edges correspond to "features" of the scalar field. They indicate where perhaps the base of a mountain is and where its peak is.

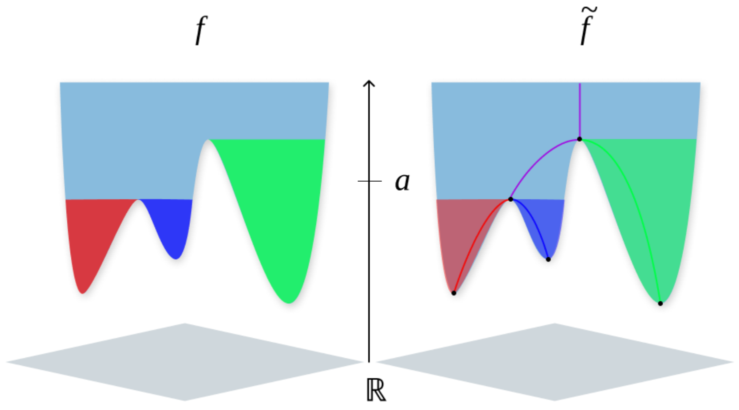

FIGURE 2. Depiction of a scalar field on the left with its features highlighted in various colors. On the right, we have the superimposed Reeb graph along highlighted versions of its edges. We can see that based on the lengths of these arcs, the green feature would be considered the most important, followed by the red feature and then the blue

If two edges meet at $a \in \mathbb{X}_f$, then a corresponds to a saddle point of the scalar field. If a is a leaf node of the Reeb graph, then a directly corresponds to a maximum or a minimum of the scalar field.

Since the critical points inherently have a value associated with them, we often associate the vertices of the Reeb graph with these function values. Let a, b be two vertices in the Reeb graph. We can assign a weight to the edge $[a, b]$ as $|\hat{f}(b) - \hat{f}(a)|$. The weights assigned to these edges help us characterize the "importance" of the features in the Reeb graph. The smaller the weight of the edge, the less important that feature is when compared to other features in the scalar field. This idea is closely related to the idea of *persistence* in the study of Persistent Homology. Figure 2 depcits the ranking of these arcs.

1.3. **Motivation.**

As the availability of computational power has increased, so has the complexity of the data that researchers are studying. In particular, researchers are often encountering situations involving **multifaceted data**. This includes data types such as a simulation run multiple times with varying parameters to determine variation and uncertainty (*multirun*), multivariate data which come from multiple sources (*multimodal*), and multiphysics simulations (*multimodel*). Here, we consider using Reeb graphs when trying to compare the scalar fields that arise from these multifaceted data types.

For example, consider the multirun situation where we have a set of parameters

θ which are used to adjust an output scalar field. In this situation, we would assume that the scalar fields all have the same domain. Because of this, it would be useful to have a metric with provides a single value measuring the similarity between these two scalar fields. If we are using Reeb graphs as adequate summaries of important features of a scalar field, the question then becomes "how do we compare two Reeb graphs"?

There has been several papers attempting to answer this question [3, 28, 2, 14]. In this document, we are reviewing a category theoretic approach to computing the *interleaving distance* between Reeb graphs [28]. While here we focus specifically on Reeb graphs, interleaving distance has also seen use for other topological structures such as persistence modules [8]. We will describe the mathematical foundation of this distance as well as the difficulties that arise in computing it. Lastly, we will provide several different future directions for this line of work, one of which includes utilizing contemporary machine learning techniques to construct similarity metrics on the space of Reeb graphs.

2. Categorifying Reeb Graphs and Scalar Fields

Category theory is a way of abstracting and generalizing objects and the relationships between these objects in a way that surpasses the usual definitions from fields such as set theory. Here, we will attempt to "categorify" the usual definitions of scalar fields and Reeb graphs to reveal important connections to the theory of pre-cosheafs. We can then define a distance between pre-cosheafs, known as the *interleaving distance*, which will ultimately provide us with a distance between Reeb graphs.

2.1. **Category Theory.**

Definition 2.1. A *category* is a collection $\mathbf{C}$ of *objects* with *morphisms* (also called *maps*) that map between these objects. The set of objects in $\mathbf{C}$ is denoted as $\mathsf{ob}(\mathbf{C})$ and the set of morphisms is denoted as $\mathsf{hom}(\mathbf{C})$. When referring to an object A in a category $\mathbf{C}$, we often denote this as $A \in \mathbf{C}$ instead of the more precise notation of $A \in \mathsf{ob}(\mathbf{C})$ when it is clear from the context that A is an object and not a morphism. Each morphism α has a source $A \in \mathbf{C}$ and a target $B \in \mathbf{C}$. Morphisms usually represent some sort of "structure" preserving idea, such as isomorphism when considering the category of vector spaces **Vect**, or homomorphisms in the category of groups **Grp**. Every object A in a category $\mathbf{C}$ has an identity morphsim $\mathbf{id}_A : A \to A$.

Definition 2.2. A *functor* F is a mapping between two categories $\mathbf{C}$ and $\mathbf{D}$ that satisfies the following properties:

- For each object $A \in \mathbf{C}$, there is a corresponding object $\mathsf{F}(A) \in \mathbf{D}$
- For each morphism $\alpha : A \to B$ in $\mathbf{C}$, there is a corresponding morphism $\mathsf{F}[\alpha] : \mathsf{F}(A) \to \mathsf{F}(B)$ in $\mathbf{D}$.
- The functors respect composition: $\mathsf{F}[\alpha \circ \beta] = \mathsf{F}[\alpha] \circ \mathsf{F}[\beta]$
- The functors respect identities: $\mathsf{F}[\mathbf{id}_A] = \mathbf{id}_{\mathsf{F}[A]}$

A functor can be thought of as a way to map between categories in the same way that functions map between spaces (vector spaces, groups, etc.)

Definition 2.3. Let $\mathsf{F}, \mathsf{G} : \mathbf{U} \to \mathbf{V}$ be two functors. Let $\alpha : a \to a'$ and $\beta : b \to b'$ be two morphisms in $\mathbf{U}$ and then let $\eta : \mathsf{F}(a) \to \mathsf{G}(b)$ and $\theta : \mathsf{F}(a') \to \mathsf{G}(b')$ be two morphisms in $\mathbf{V}$. Then we say that η and θ are *natural with respect to α and β* if the following diagram commutes:

$$\begin{tikzcd}
\mathsf{F}(a) \arrow[r, "\mathsf{F}[\alpha]"] \arrow[d, "\eta"'] & \mathsf{F}(a') \arrow[d, "\theta"] \\
\mathsf{G}(b) \arrow[r, "\mathsf{G}[\beta]"'] & \mathsf{G}(b')
\end{tikzcd}$$

Definition 2.4. A *natural transformation* is a map $\eta : \mathsf{F} \Rightarrow \mathsf{G}$ between two functors, $\mathsf{F}, \mathsf{G} : \mathbf{U} \to \mathbf{V}$. It consists of a family of morphisms $\eta_a : \mathsf{F}(a) \to \mathsf{G}(a)$, one for each object $a \in \mathbf{U}$, such that for each morphism $\alpha : a \to a'$ in $\mathbf{U}$, the following diagram commutes:

$$\begin{tikzcd}
\mathsf{F}(a) \arrow[r, "\mathsf{F}[\alpha]"] \arrow[d, "\eta_a"'] & \mathsf{F}(a') \arrow[d, "\eta_{a'}"] \\
\mathsf{G}(a) \arrow[r, "\mathsf{G}[\alpha]"'] & \mathsf{G}(a')
\end{tikzcd}$$

In other words, we say that the family of maps η is natural with respect to every morphism α in $\mathbf{U}$.

Definition 2.5. We say that two functors $\mathsf{F}, \mathsf{G} : \mathbf{U} \to \mathbf{V}$ are *isomorphic* to each other if there exists a pair of natural transformations $\eta := \{\eta_a\}_{a \in \mathbf{U}}$ and $\theta := \{\theta_a\}_{a \in \mathbf{U}}$ where $\eta_a : \mathsf{F}(a) \to \mathsf{G}(a)$ and $\theta_a : \mathsf{G}(a) \to \mathsf{F}(a)$, such that $\eta_a \circ \theta_a = G[\mathbf{id}_a]$ and $\theta_a \circ \eta_a = F[\mathbf{id}_a]$. In other words, the maps η_a and θ_a are inverses of each other for each $a \in \mathbf{U}$.

Definition 2.6. Let $\mathbf{C}$ be a category. A *subcategory* $\mathbf{S}$ of $\mathbf{C}$ is a category consisting of a subcollection of objects of $\mathbf{C}$, denoted as $\mathsf{ob}(\mathbf{S})$ and a subcollection of morphisms of C, denoted as $\mathsf{hom}(\mathbf{S})$, such that

- for every X in $\mathsf{ob}(\mathbf{S})$, the identity morphism $\mathbf{id}_X$ is in $\mathsf{hom}(\mathbf{S})$

- for every morphism $\alpha : X \to Y$ in $\mathsf{hom}(\mathbf{S})$, both the source X and the target Y are in $\mathsf{ob}(\mathbf{S})$
- for every pair of morphisms α and β in $\mathsf{hom}(\mathbf{S})$, the composite $\alpha \circ \beta$ is in $\mathsf{hom}(\mathbf{S})$ (whenever the composition is defined).

2.2. **Category of Scalar Fields.**

Definition 2.7. We can construct the *category of scalar fields* (in related work also called the category of $\mathbb{R}$-spaces), denoted as $\mathbb{R}$-**Top**, by stating that each object is a scalar field $(\mathbb{X}, f)$ and each morphism $\alpha : (\mathbb{X}, f) \to (\mathbb{Y}, g)$ is a continuous, *function-preserving* map $\alpha : \mathbb{X} \to \mathbb{Y}$ between topological spaces. A function preserving map means that $\alpha \circ g = f$. In other words, the following diagram commutes:

$$\begin{tikzcd} \mathbb{X} \arrow[rr, "\alpha"] \arrow[dr, "f"'] & & \mathbb{Y} \arrow[dl, "g"] \\ & \mathbb{R} & \end{tikzcd}$$

Definition 2.8. We call the point pre-images $f^{-1}(a)$ of $\mathbb{X}$ the *a-levelset* or *a-fiber* of $\mathbb{X}$. In this way, we can say that the morphisms in the category of scalar fields preserve a-levelsets or a-fibers.

To make sure that we are dealing with well-behaved scalar fields, we define a new subcategory called the category of *constructible* scalar fields, denoted as $\mathbb{R}$-$\mathbf{Top}^{\mathbf{C}}$.

Let $S = \{a_1, \dots, a_n\}$ be a finite set of critical points. For each critical point, we have an associated critical fiber $\mathbb{V}_i$. Likewise, we define a single non-ciritcal fiber denoted as $\mathbb{E}_i$ for each $0 \leq i \leq n-1$. While in theory there are infinitely many non-critical fibers, here we are introducing only a single "representative" non-critical fiber for each section between critical points. Furthermore, we require that each critical and non-critical fiber is a be a locally path-connected, compact space.

Now we define continuous attaching maps $\mathbb{l}_i : \mathbb{E}_i \to \mathbb{V}_i$ and $\mathbb{r}_i : \mathbb{E}_i \to \mathbb{V}_{i-1}$ for each $0 \leq i \leq n-1$. Finally, we define $\hat{\mathbb{X}}$ to be the quotient space obtained by the disjoint union of $\mathbb{V}_i \times \{a_i\}$ and $\mathbb{E}_i \times [a_i, a_{i+1}]$ by making the identifications $(\mathbb{l}_i(x), a_i) \sim (x, a_i)$ and $(\mathbb{r}_i(x), a_{i+1}) \sim (x, a_{i+1})$ for all i and all $x \in \mathbb{E}_i$. Now, define a function f such that $f(x, a) = a$. We can see that the pair $(\hat{\mathbb{X}}, f)$ defines a scalar field since $\hat{\mathbb{X}}$ is a topological space and $\hat{f}$ is simply a projection onto the second factor, which is a continuous function.

Definition 2.9. If $\mathbb{X}$ is isomorphic to $\hat{\mathbb{X}}$, then $\mathbb{X}$ is a *constructible scalar field* and we say that $\hat{\mathbb{X}}$ is its *constructible base*. The intuition here is that the scalar fields are "well-behaved" if they have a finite set of critical points and if between critical points, their shape is homotopic to a cylinder. Then at the critical fibers, we have places where the cylindrical structures are able to "merge" or "split". Note that these attaching maps are always mapping from the non-critical fibers. So if our non-critical fiber has a cylinder that split into two cylinders, there must be a critical fiber which is a figure eight, exactly at the point where they start to split. We can always map a single cylinder to a figure eight or two cylinders to the figure eight. We never have to worry about actually "splitting" the figure eight into two cylinders (which is discontinuous) because the maps only go towards the critical-fibers.

2.3. **Category of Reeb Graphs and the Reeb Functor.**

Proposition 2.10. *The set of Reeb graphs created from scalar fields defines a subcategory of* $\mathbb{R}$-***Top***.

To prove this, we need to show that the Reeb graphs are scalar fields and that there exists a subcollection of morphisms of $\mathsf{hom}(\mathbb{R}\text{-}\mathbf{Top})$ where the source and target are always Reeb graphs. To help us with the following proofs, we refer to the lemma below:

Lemma 2.11. *(from Munkres [24], Theorem 22.2) Let $\rho : X \to Y$ be a quotient map. Let Z be a space and let $f : X \to Z$ be a map that is constant on each set $\rho^{-1}(\{y\})$, for $y \in Y$. Then f induces a map $g : Y \to Z$ such that $g \circ \rho = f$. The induced map g is continuous if and only if f is continuous.*

Lemma 2.12. *Let $(\mathbb{X}, f)$ be a scalar field and $(\mathbb{X}_f, \tilde{f})$ denote its Reeb graph. Then $(\mathbb{X}_f, \tilde{f})$ is a scalar field.*

Proof. By definition $\mathbb{X}_f$ is a topological space. What is left to show is that $\tilde{f}$ is a continuous function. Let ρ be the quotient map from $\mathbb{X}$ to $\mathbb{X}_f$. Since f is constant on point pre-images of $\mathbb{X}_f$, Munkres Theorem 22.2 tells us that there exists a continuous function $g \circ \rho = f$. By definition of a Reeb graph, we have that $\tilde{f} \circ \rho = f$, so $\tilde{f} = g$ and $(\mathbb{X}_f, \tilde{f})$ is a scalar field. □

Lemma 2.13. *Let $(\mathbb{X}, f)$ and $(\mathbb{Y}, g)$ be scalar fields, let $(\mathbb{X}_f, \tilde{f})$ and $(\mathbb{Y}_g, \tilde{g})$ be their Reeb graphs resepectively, and let ρ_f and ρ_g be the quotient maps from the scalar fields to the Reeb graphs. If α is a morphism between $(\mathbb{X}, f)$ and $(\mathbb{Y}, g)$, then the map $\tilde{\alpha}$ defined by $\rho_g \circ \alpha = \tilde{\alpha} \circ \rho_f$ is a morphism in* $\mathbb{R}$-***Top***.

Proof. Consider the composition of maps $(\rho_g \circ \alpha) : \mathbb{X} \to \mathbb{Y}_g$. This map is continuous since both ρ_g and α are continuous. Now, consider a point pre-image $\rho_f^{-1}(\{y\}) \in \mathbb{X}$ where $\tilde{f}(y) = a$. Then this pre-image is the a-fiber of $\mathbb{X}$. Since $\alpha \in \mathsf{hom}(\mathbb{R}\text{-}\mathbf{Top})$, it preserves level sets, so $\alpha(\rho_f^{-1}(\{y\}))$ is the a-fiber of $\mathbb{Y}$. Then, since ρ_g is the quotient map turning all a-fibers into a single contracted point in $\mathbb{Y}_g$, the composition of maps $(\rho_g \circ \alpha)$ is constant on pre-images of ρ_f. Using Munkres Theorem 22.2 again, we have that the induced map $\tilde{\alpha}$ in the equation $\rho_g \circ \alpha = \tilde{\alpha} \circ \rho_f$ must be continuous, since $\rho_g \circ \alpha$ is continuous and constant on the point pre-images of the quotient map ρ_f.

To show that $\tilde{\alpha}$ is a function preserving map, we need to show that $\tilde{f} = \tilde{g} \circ \tilde{\alpha}$. We begin with the fact that $f = g \circ \alpha$:

$$\begin{aligned} f = g \circ \alpha \quad &\Rightarrow (\tilde{f} \circ \rho_f) = (\tilde{g} \circ \rho_g) \circ \alpha \\ &\Rightarrow (\tilde{f} \circ \rho_f) = \tilde{g} \circ (\rho_g \circ \alpha) \\ &\Rightarrow (\tilde{f} \circ \rho_f) = \tilde{g} \circ (\tilde{\alpha} \circ \rho_f) \\ &\Rightarrow \tilde{f} = \tilde{g} \circ \tilde{\alpha} \end{aligned}$$

□

Proof of Proposition 2.10 . Note that the set of Reeb graphs is a collection of constructible scalar fields and that the maps we defined between Reeb graphs are morphisms in $\mathbb{R}$-**Top**. Thus, the collection of Reeb graphs constructed from scalar fields is a subcategory of $\mathbb{R}$-**Top** □

Definition 2.14. We define the *Reeb Functor* as the functor $\mathcal{R} : \mathbb{R}\text{-}\mathbf{Top} \to \mathbb{R}\text{-}\mathbf{Top}$ by the formulas $\mathcal{R}(\mathbb{X}, f) := (\mathbb{X}_f, \tilde{f})$ and $\mathcal{R}[\alpha] = \tilde{\alpha}$.

Proposition 2.15. *If $(\mathbb{X}, f)$ is constructible, then $\mathcal{R}(\mathbb{X}, f)$ is constructible as well. That is, when restricted to constructible scalar fields, the Reeb Functor maps to constructible scalar fields.*

Proof. Let $(\mathbb{X}, f) \in \mathbb{R}\text{-}\mathbf{Top}^{\mathbf{C}}$ and let $(\mathbb{X}_f, \tilde{f})$ be its Reeb graph. Let $\hat{\mathbb{X}}$ be a constructible base of $(\mathbb{X}, f)$ with critical points $\mathcal{S} = \{a_0, \ldots, a_n\}$ and critical and non-critical fibers, $\mathbb{V}_i$ and $\mathbb{E}_i$. We create a new constructible scalar field $\mathbb{G}$ made up of critical fibers V_i and non-critical fibers E_i by using the following definitions:

$$V_i := \pi_0(\mathbb{V}_i) \qquad E_i := \pi_0(\mathbb{E}_i) \qquad l_i := \pi_0(\mathbb{l}_i) \qquad r_i := \pi_0(\mathbb{r}_i),$$

where π_0 denotes the set of path connected components. $\mathbb{G}$ is then defined as the quotient space of the disjoint union of spaces $V_i \times \{a_i\}$, for all $0 \le i \le n$ and $E_i \times [a_i, a_{i+1}]$ for all $0 \le i \le n-1$ using the identifications $(l_i(x), a_i) \sim (x, a_i)$ and $(r_i(x), a_{i+1}) \sim (x, a_{i+1})$. Note that this is exactly the same construction as in the definition of constructible scalar fields, except the critical and non-critical fibers here are 0-dimensional. Our goal is to show that $\mathbb{G}$ is a valid constructible base for $\mathbb{X}_f$.

Firstly, note that there is a bijection between $\mathbb{X}_f$ and $\mathbb{G}$ because $\mathbb{G}$ is exactly the path-connected components of the levelsets of f. Now, consider the map, say α, from $\hat{\mathbb{X}}$ to $\mathbb{G}$ which maps locally path-connected spaces $\mathbb{V}_i$ and $\mathbb{E}_i$ to their path components V_i and E_i, respectively. By definition, this map is continuous since the path components of a locally path-connected space are open. Now consider the maps $\beta : \hat{\mathbb{X}} \to \mathbb{X}$ and $\rho : \mathbb{X} \to \mathbb{X}_f$. The former is a bijection while the latter is a quotient map. Applying Lemma 2.11, we have that there exists a map g such that $(\rho \circ \beta) \circ g = \alpha$. Since α is continuous, then g must also be continuous. Thus, the map $g : \mathbb{X}_f \to \mathbb{G}$ is a continuous bijection. We finish by noting that a continuous bijection from a compact space $\mathbb{X}_f$ to a Hausdorff space $\mathbb{G}$ is a homeomorphism.

□

2.4. **Pre-Cosheafs.**

Pre-cosheafs allow us to abstract the data that a Reeb graph provides us. In this section, we define what a pre-cosheaf is and show how we can describe the pre-cosheaf of a Reeb graph.

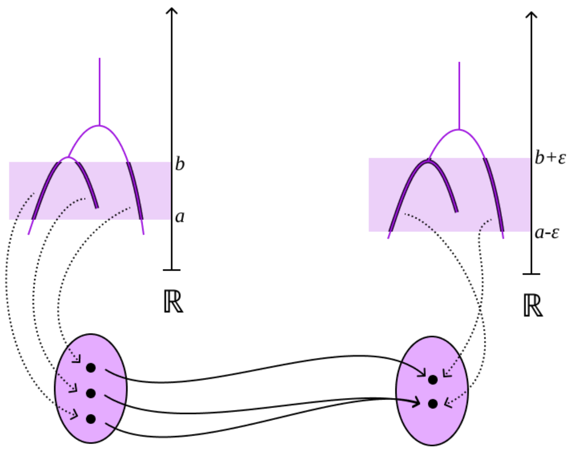

FIGURE 3. On the left, we have the representation of a Reeb graph along with its pre-cosheaf on the interval $I = (a, b)$. On the right, we have the same Reeb graph with an expanded interval $I^\varepsilon = (a - \varepsilon, b + \varepsilon)$. Below, we show how the pre-cosheaf of the left-hand diagram maps into the pre-cosheaf of the right diagram based on inclusion of the interval $I \subseteq I^\varepsilon$

Definition 2.16. Let $\mathbf{Open}(\mathcal{X})$ be the category of open sets on the topological space $\mathcal{X}$ where $I \to J$ if $I \subseteq J$. Then a *pre-cosheaf* is a functor $\mathsf{F} : \mathbf{Open}(\mathcal{X}) \to \mathbf{D}$, where $\mathbf{D}$ is some category.

In other words, **a pre-cosheaf is an assignment of data to open sets of a topological space $\mathcal{X}$.**

In this work, our pre-cosheaves will exclusively have the category of intervals on the real line **Int** as the domain space and the category of sets **Set** as the range space. Note that the morphisms in **Set** are functions between sets.

Definition 2.17. We define the *category of pre-cosheaves*, denoted as **Pre**, to be the set of functors from **Int** to **Set**. The morphisms in **Pre** are then natural transformations.

Definition 2.18. Let $f = (\mathbb{X}, f)$ be a Reeb graph. The *Reeb cosheaf functor* $\mathcal{C}$ is defined by the formulas

$$\mathsf{F}(I) = \pi_0(f^{-1}(I)), \qquad \mathsf{F}[I \subseteq J] = \pi_0[f^{-1}(I) \subseteq f^{-1}(J)].$$

This functor converts a Reeb graph to its pre-cosheaf. Thus, $\mathcal{C} : \mathbf{Reeb} \to \mathbf{Pre}$. On the left-hand side of Figure 3, we have a depiction of the pre-cosheaf of a Reeb graph on the interval $I = (a, b)$.

3. Interleaving Distance

In this section, we begin the category-theoretic definition of *interleaving distance* based on pre-cosheafs. We will leverage the relationship between Reeb graphs and pre-cosheafs to show that this distance in fact can be realized geometrically. Finally, we re-define interleaving distance solely in the context of Reeb graphs.

3.1. **Smoothing Functors and Interleaving of Pre-Cosheafs.**

Recall that an isomorphism between two functors F, G is defined as two natural transformations $\varphi = \{\varphi_I\}_{I \in \mathbf{C}}$ and $\psi = \{\psi_I\}_{I \in \mathbf{C}}$ such that $\varphi_I : \mathsf{F}(I) \to \mathsf{G}(I)$, $\psi_I : \mathsf{G}(I) \to \mathsf{F}(I)$ and φ_I, ψ_I are inverses of each other for each $I \in \mathbf{C}$. If we are not able to construct an isomorphism between two functors, we somehow want to quantify how far away we are from an isomorphism.

Suppose we have two pre-cosheafs F, G. Suppose that G is an exact copy of F except that G has additional noise to it. By noise, we are saying that G assigns data to very small intervals that F does not assign data too. If the intervals we are viewing are sufficiently large, then the noise is undetectable. To essentially remove the noise from G, we can pass G through a "filter". If noise is detectable up to intervals of length ε, then the filter can simply expand each interval by ε in both directions before the assignment of data.

Earlier, we discussed how the lengths of the arcs of the Reeb graph correspond to the importance of the feature displayed. Recall that to change a Reeb graph to a pre-cosheaf, we indentify every interval of the real line with the set of path connected components of the pre-image on that interval. Now, suppose that the we have a leaf of the Reeb graph such that is begins at the height a and ends (merges with another component) at the height b. If we consider an interval $I = (a-\varepsilon, b+\varepsilon)$ where $\varepsilon > 0$, then the pre-cosheaf associated with this interval, defined as $\pi_0(f^{-1}(I))$, would not show this arc as a separate path connected component because the arc is too small compared to the interval. By increasing <u>all</u> intervals by a length of $\varepsilon > 0$, then no arc that has length equal to or less than ε is detectable. This operation of smoothing these intervals is essentially removing this topological noise.

The Interleaving distance quantifies the distance between pre-cosheafs by finding the minimum filter size ε that we need to pass one pre-cosheaf through in order to make it isomorphic to another pre-cosheaf.

Definition 3.1. Let $I = (a, b) \subseteq \mathbb{R}$ and $I^\varepsilon = (a - \varepsilon, b + \varepsilon)$. The *$\varepsilon$-expansion functor*, Ω_ε, where $\varepsilon > 0$, is defined as

$$\Omega_\varepsilon : \mathbf{Int} \to \mathbf{Int}; \quad I \mapsto I^\varepsilon$$

This is a well-defined functor since when $I \subseteq J$ we have $I^\varepsilon \subseteq J^\varepsilon$. We say that an pre-cosheaf can be "ε-smoothed" to obtain a new pre-cosheaf by pre-composition of the expansion functor:

$$\mathsf{F}\Omega_\varepsilon : \mathbf{Int} \to \mathbf{Set}; \quad \mathsf{F}\Omega_\varepsilon(I) = \mathsf{F}(I^\varepsilon)$$

Note that since $I \subseteq I^\varepsilon$ for all I, we can construct a natural transformation $\omega^\varepsilon : 1_{\mathbf{Int}} \Rightarrow \Omega_\varepsilon$. That is, the following diagram commutes:

$$\begin{tikzcd}
1_{\mathbf{Int}}(I) \arrow[r, "1_{\mathbf{Int}}[I \subseteq J]"] \arrow[d, "\omega^{\varepsilon}_I"'] & 1_{\mathbf{Int}}(J) \arrow[d, "\omega^{\varepsilon}_J"] \\
\Omega_\varepsilon(I) \arrow[r, "\Omega_\varepsilon[I \subseteq J]"'] & \Omega_\varepsilon(J)
\end{tikzcd}$$

Because of this, we can compose ω^ε with the pre-cosheaf F to get $\sigma^\varepsilon_{\mathsf{F}} = \mathsf{F}\omega^\varepsilon : \mathsf{F} \Rightarrow \mathsf{F}\Omega_\varepsilon$ defined explicitly by $(\sigma^\varepsilon_{\mathsf{F}})_I = \mathsf{F}[I \subseteq I^\varepsilon] : \mathsf{F}(I) \to \mathsf{F}(I^\varepsilon)$. Figure 3 shows how this map $(\sigma^\varepsilon_{\mathsf{F}})_I$ operates.

Remark 3.2. The functor Ω_ε is an example of a *pointed endofunctor*. An *endofunctor* is any functor that has the same source and target space. The term *pointed* refers to the fact that it induces this natural transformation ω^ε from the identity functor to itself.

Definition 3.3. We say that two pre-cosheafs F, G are *ε-interleaved* if there exists a pair of natural transformations $\varphi : \mathsf{F} \Rightarrow \mathsf{G}\Omega_\varepsilon$ and $\psi : \mathsf{G} \Rightarrow \mathsf{F}\Omega_\varepsilon$ such that the following diagrams commute:

$$\begin{tikzcd}
\mathsf{F} \arrow[dr, Rightarrow, "\varphi"] \arrow[dd, Rightarrow, "\sigma^{2\varepsilon}_{\mathsf{F}}"'] & \\
& \mathsf{G}\Omega_\varepsilon \arrow[dl, Rightarrow, "\psi\Omega_\varepsilon"] \\
\mathsf{F}\Omega_{2\varepsilon} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathsf{G} \arrow[dl, Rightarrow, "\psi"'] \arrow[dd, Rightarrow, "\sigma^{2\varepsilon}_{\mathsf{G}}"] \\
\mathsf{F}\Omega_\varepsilon \arrow[dr, Rightarrow, "\varphi\Omega_\varepsilon"'] & \\
& \mathsf{G}\Omega_{2\varepsilon}
\end{tikzcd}$$

If $\varepsilon = 0$, then this is exactly the definition of an isomorphism between F and G. When two pre-cosheafs are ε-interleaved, we say that there exists an *ε-interleaving* between them.

Definition 3.4. The *interleaving distance* between two pre-cosheafs F and G is defined as

$$d_I(\mathsf{F}, \mathsf{G}) = \inf\{\varepsilon \geq 0 | \text{ there exists an } \varepsilon\text{-interleaving between } \mathsf{F}, \mathsf{G}\}$$

To find the interleaving distance between Reeb graphs, we can convert them into pre-cosheafs and then find the interleaving distance between the cosheafs.

Definition 3.5. Let $f = (\mathbb{X}, f)$ and $g = (\mathbb{Y}, g)$ be two Reeb graphs. Since we have a notion of distance between two pre-cosheafs, we can define the *interleaving distance between* f, g simply by

$$d_I(f, g) := d_I(\mathcal{C}(f), \mathcal{C}(g))$$

Definition 3.6. Let $\mathcal{S}_\varepsilon$ be defined by the formula $\mathcal{S}_\varepsilon(\mathsf{F}) := \mathsf{F}\Omega_\varepsilon$ and $\mathcal{S}_\varepsilon[\alpha] := \alpha\Omega_\varepsilon : \mathsf{F}\Omega_\varepsilon \Rightarrow \mathsf{G}\Omega_\varepsilon$ for all $\alpha \in \mathsf{hom}(\mathbf{Pre})$. We call $\mathcal{S}_\varepsilon$ the *smoothing functor.*

Definition 3.7. Let $f = (\mathbb{X}, f)$ and $g = (\mathbb{Y}, g)$ be two Reeb graphs. We say that f, g are ε-interleaved if there exists a pair of natural transformations $\varphi : \mathcal{C}(f) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(g)$ and $\psi : \mathcal{C}(g) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(f)$ such that the following diagrams commute:

$$\begin{tikzcd}
\mathcal{C}(f) \arrow[dr, Rightarrow, "\varphi"] \arrow[dd, Rightarrow, "\sigma^{2\varepsilon}_{\mathcal{C}(f)}"'] & \\
 & \mathcal{S}_\varepsilon\mathcal{C}(g) \arrow[dl, Rightarrow, "\mathcal{S}_\varepsilon[\psi]"] \\
\mathcal{S}_{2\varepsilon}\mathcal{C}(f) &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
 & \mathcal{C}(g) \arrow[dl, Rightarrow, "\psi"'] \arrow[dd, Rightarrow, "\sigma^{2\varepsilon}_{\mathcal{C}(g)}"] \\
\mathcal{S}_\varepsilon\mathcal{C}(f) \arrow[dr, Rightarrow, "\mathcal{S}_\varepsilon[\varphi]"'] & \\
 & \mathcal{S}_{2\varepsilon}\mathcal{C}(g)
\end{tikzcd}$$

FIGURE 4. Interleaving Between Two Pre-cosheafs

Remark 3.8. The natural transformation σ^ε was defined before as $\sigma^\varepsilon_{\mathsf{F}} : \mathsf{F} \Rightarrow \mathsf{F}\Omega_\varepsilon$. In the above diagram, the definition is the same except with a different functor $\mathcal{C}(f)$ and the target space has been rewritten. That is,

$$\sigma^\varepsilon_{\mathcal{C}(f)} : \mathcal{C}(f) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(f)$$

In the following sections, we will leverage the above definition to show that interleaving distance can be defined without having to convert to pre-cosheafs at all.

3.2. Thickening Functors.

Here we describe a *thickening functor* on Reeb graphs that operates in parallel to the smoothing functors described previously. What we will show is that it is equivalent to work with thickened Reeb graphs or smoothed pre-cosheafs. Because of this equivalence, we will be able to utilize this thickening procedure (which will be shown to have a very nice, geometric realization) instead of using the more abstract smoothing functor approach.

Definition 3.9. For $\varepsilon \geq 0$, we define the thickening functor $\mathcal{T}_\varepsilon : \mathbb{R}\text{-}\mathbf{Top} \to \mathbb{R}\text{-}\mathbf{Top}$ as follows:

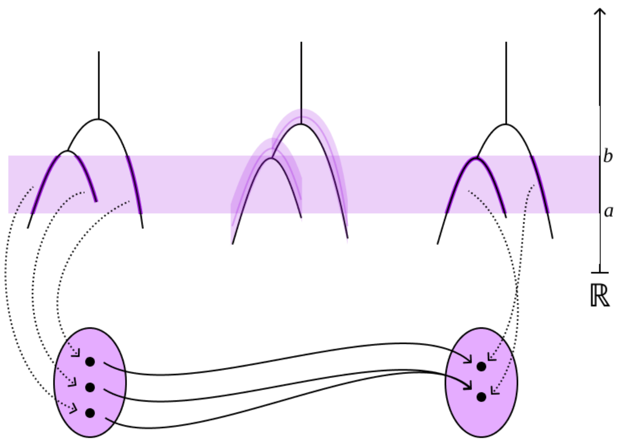

FIGURE 5. On the left we have a Reeb graph whose connected components based on the interval $I = (a, b)$ are highlighted. Below it we display the three connected components as a set – the pre-cosheaf of this Reeb graph on the interval I. The center diagram shows a thickening of that by ε, with the new Reeb graph superimposed on top of it. The right picture shows that the pre-cosheaf on the interval I for this new Reeb graph only has two components.

- Let $(\mathbb{X}, f)$ be a scalar field, let $\mathbb{X}_\varepsilon = \mathbb{X} \times [-\varepsilon, \varepsilon]$, and let $f_\varepsilon(x, t) = f(x) + t$. Then $\mathcal{T}_\varepsilon(\mathbb{X}, f) = (\mathbb{X}_\varepsilon, f_\varepsilon)$
- Let $\alpha : (\mathbb{X}, f) \to (\mathbb{Y}, g)$ be a morphism in $\mathbb{R}$-**Top**. Then we define $\mathcal{T}_\varepsilon : (\mathbb{X}_\varepsilon, f_\varepsilon) \to (\mathbb{Y}_\varepsilon, g_\varepsilon)$ by $(x, t) \mapsto (\alpha(x), t)$.

Note that when $(\mathbb{X}, f)$ is a Reeb graph, $\mathbb{X}_\varepsilon$ is a two dimensional space, so it is no longer a Reeb graph. Often, we immediately use the Reeb Functor after thickening to keep everything in the category of Reeb graphs. We can see an example of Thickening in Figure 5. The middle diagram depicts the thickened scalar field as well as the Reeb graph of this thickened field superimposed on top of it.

Theorem 3.10. *The functors $\mathcal{CRT}_\varepsilon$ and $\mathcal{S}_\varepsilon\mathcal{C}$ are naturally isomorphic.*

That is, looking at the cosheafs of an ε-thickened Reeb graph is the same as ε-smoothing the pre-cosheafs. Note that on the left side, before finding the pre-cosheafs, we need to project $\mathcal{T}_\varepsilon$ back into **Reeb**. To prove this, we will use the following lemma:

Lemma 3.11. *Let $p : \mathbb{X}_\varepsilon \to \mathbb{X}$ note restriction on the first factor. That is, $(x, t) \mapsto x$. The map p restricts to a homotopy equivalence $f_\varepsilon^{-1}(I) \xrightarrow{\sim} f^{-1}(I^\varepsilon)$ for each interval I.*

Proof of Theorem 3.10. We will define a natrual transformation $\rho : \mathcal{CT}_\varepsilon \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}$ and show that ρ_f is an isomorphism for all $f \in \mathbb{R}$-**Top**. First, let $\rho_f : \mathcal{CT}_\varepsilon(f) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(f)$ be defined by the formula $(\rho_f)_I = \pi_0[p_I]$, where p_I is the map defined in Lemma 3.11. Since p_I is a homotopy equivalence, $(\rho_f)_I$ must be an isomorphism. We begin by showing that the family of maps $(\rho_f)_I$ is natural with respect to the inclusions $I \subseteq J$. Consider the following two diagrams:

$$\begin{tikzcd}
f_\varepsilon^{-1}(I) \arrow[r] \arrow[d, "p_I"'] & f_\varepsilon^{-1}(J) \arrow[d, "p_J"] \\
f^{-1}(I^\varepsilon) \arrow[r] & f^{-1}(J^\varepsilon)
\end{tikzcd}
\qquad
\begin{tikzcd}
\pi_0\left(f_\varepsilon^{-1}(I)\right) \arrow[r] \arrow[d, "\pi_0[p_I]"'] & \pi_0\left(f_\varepsilon^{-1}(J)\right) \arrow[d, "\pi_0[p_J]"] \\
\pi_0\left(f^{-1}(I^\varepsilon)\right) \arrow[r] & \pi_0\left(f^{-1}(J^\varepsilon)\right)
\end{tikzcd}$$

Since the left square commutes, the right square must commute as well. Thus, ρ_f is a natural transformation. Furthermore, since each $(\rho_f)_I$ is an isomorphism, ρ_f defines an isomorphism between $\mathcal{C}\mathcal{T}_\varepsilon$ and $\mathcal{S}_\varepsilon\mathcal{C}(f)$.

Now we need so show that the family of maps $\rho = (\rho_f)$ is natural with repsect to the morphisms in $\mathbb{R}$-**Top**. Again, consider the following diagrams

$$\begin{tikzcd}
f_\varepsilon^{-1}(I) \arrow[r, "\alpha \times \mathbb{1}"] \arrow[d, "p_I^f"'] & g_\varepsilon^{-1}(I) \arrow[d, "p_I^g"] \\
f^{-1}(I^\varepsilon) \arrow[r, "\alpha"'] & g^{-1}(I^\varepsilon)
\end{tikzcd}
\qquad
\begin{tikzcd}
\pi_0\left(f_\varepsilon^{-1}(I)\right) \arrow[r, "\pi_0[\alpha \times \mathbb{1}]"] \arrow[d, "\pi_0[p_I]"'] & \pi_0\left(g_\varepsilon^{-1}(I)\right) \arrow[d, "\pi_0\left[p_I^g\right]"] \\
\pi_0\left(f^{-1}(I^\varepsilon)\right) \arrow[r, "\pi_0[\alpha]"'] & \pi_0\left(g^{-1}(I^\varepsilon)\right)
\end{tikzcd}$$

Here, $\alpha \in \mathsf{hom}(\mathbb{R}\text{-}\mathbf{Top})$. The morphism $\alpha \times \mathbb{1}$ is defined as $(x, t) \mapsto (\alpha(x), t)$. From this, we can see that ρ satisfies the naturality condition for each I. Thus, ρ defines an isomorphism between $\mathcal{C}\mathcal{R}\mathcal{T}_\varepsilon$ and $\mathcal{S}_\varepsilon\mathcal{C}$. □

Without proof, we state another proposition concerning constructibility of the thickened Reeb graphs:

Proposition 3.12. *If* $(\mathbb{X}, f) \in \mathbf{Reeb}$ *or* $(\mathbb{X}, f) \in \mathbb{R}\text{-}\boldsymbol{Top}^{\boldsymbol{C}}$*, then* $\mathcal{T}_\varepsilon(\mathbb{X}, f) \in \mathbb{R}\text{-}\boldsymbol{Top}^{\boldsymbol{C}}$.

Because of this, we do not need to worry about our thickened scalar fields introducing complexity that we are not already equipped to deal with.

3.3. **Topological Smoothing of Reeb Graphs.**

At this point, we are ready to give a geometric procedure for defining the interleaving distance.

Definition 3.13. Let $(\mathbb{X}, f)$ be a Reeb graph. We define the *Reeb Smoothing Functor* $\mathcal{U}_\varepsilon$ as $\mathcal{U}_\varepsilon := \mathcal{R}\mathcal{T}_\varepsilon$.

Remark 3.14. Each $\mathcal{U}_\varepsilon$ is a pointed endofunctor of **Reeb**. There exists a natural transformation $\zeta^\varepsilon : 1_{\mathbf{Reeb}} \Rightarrow \mathcal{U}_\varepsilon$ where $\zeta^\varepsilon_f : f \to \mathcal{U}_\varepsilon(f)$. We define ζ^ε_f as follows:

- Let $\tau^\varepsilon_f : \mathbb{X} \to \mathbb{X}_\varepsilon$ be inclusion map from the Reeb graph $\mathbb{X}$ into $\mathbb{X}_\varepsilon = \mathbb{X} \times [-\varepsilon, \varepsilon]$ defined by $x \mapsto (x, 0)$.
- Let $\rho_{f_\varepsilon} : \mathbb{X}_\varepsilon \to \mathbb{X}_\varepsilon / \sim$ be the quotient map from this new scalar field $\mathbb{X}_\varepsilon$ to its Reeb graph

We define $\zeta^\varepsilon_f := \rho_{f_\varepsilon} \circ \tau^\varepsilon_f$.

Theorem 3.15. *If $f, g \in$ **Reeb**, then f, g are ε-interleaved if there exists maps*

$$\alpha : f \to \mathcal{U}_\varepsilon(g) \quad \textit{and} \quad \beta : g \to \mathcal{U}_\varepsilon(f)$$

such that the following diagrams commute:

$$\begin{tikzcd}
f \arrow[dr, "\alpha"] \arrow[dd, "\zeta^{2\varepsilon}_f"'] & \\
& \mathcal{U}_\varepsilon(g) \arrow[dl, "\beta^\varepsilon_{2\varepsilon}"] \\
\mathcal{U}_{2\varepsilon}(f) &
\end{tikzcd}
\quad \textit{and} \quad
\begin{tikzcd}
& g \arrow[dl, "\beta"'] \arrow[dd, "\zeta^{2\varepsilon}_g"] \\
\mathcal{U}_\varepsilon(f) \arrow[dr, "\alpha^\varepsilon_{2\varepsilon}"'] & \\
& \mathcal{U}_{2\varepsilon}(g)
\end{tikzcd}$$

FIGURE 6. Interleaving between two Reeb graphs.

This proposition tell us that we can find ε-interleavings by direclty relating the Reeb graphs instead of converting them into pre-cosheafs – a useful result as it gives us a great geometric picture for computing the interleaving distance. To prove this, we start by stating and proving the following Lemma:

Lemma 3.16. *For $f \in$ **Reeb** and $\varepsilon \geq 0$, the diagram*

$$\begin{tikzcd}
\mathcal{C}(f) \arrow[d, Rightarrow, "\mathcal{C}[\zeta]"'] \arrow[dr, Rightarrow, "\sigma"] & \\
\mathcal{C}\mathcal{U}_\varepsilon(f) \arrow[r, Rightarrow, "\rho_f"'] & \mathcal{S}_\varepsilon\mathcal{C}(f)
\end{tikzcd}$$

commutes.

Proof. We need to show that this diagram commutes when evaluated on an arbitrary interval I. If we follow the definition of the Reeb Cosheaf functor, we get diagram on the right, below. On the left, we have the same diagram before applying the connected component function π_0:

$$\begin{tikzcd}
f^{-1}(I) \arrow[d, "p_I^*"'] \arrow[dr, "f^{-1}(I) \subseteq f^{-1}(I^\varepsilon)"] & & {} \arrow[r, "\pi_0"] & {} & \pi_0\left(f^{-1}(I)\right) \arrow[d, "\mathcal{C}[\zeta]_I"'] \arrow[dr, "\sigma_I"] & \\
f_\varepsilon^{-1}(I) \arrow[r, "p_I"'] & f^{-1}\left(I^\varepsilon\right) & & & \pi_0\left(f_\varepsilon^{-1}(I)\right) \arrow[r, "\rho_I"'] & \pi_0\left(f^{-1}\left(I^\varepsilon\right)\right)
\end{tikzcd}$$

In the left square, the map p_I^* is defined as $x \mapsto (x, 0)$. Thus, based on the definition of p_I as defined in 3.11, the composite map $p_I \circ p_I^* : f^{-1}(I) \to f^{-1}(I^\varepsilon)$ maps x to x. On the other hand, the map defined by the inclusion $f^{-1}(I) \subseteq f^{-1}(I^\varepsilon$ works the same way: A point $x \in f^{-1}(I)$ gets mapped to itself in $f^{-1}(I^\varepsilon)$. Since this diagram commutes, the right diagram commutes as well. □

Proof of Proposition 3.15. Suppose we have α, β such that the diagrams above commute. We can obtain a new diagram by applying the Reeb Cosheaf functor $\mathcal{C}$ to each of the objects and morphisms in the diagrams:

$$\begin{tikzcd}
\mathcal{C}(f) \arrow[dd, Rightarrow, "\mathcal{C}\left[\zeta_f^{2\varepsilon}\right]"'] \arrow[dr, Rightarrow, "\alpha"] & & & & \mathcal{C}(g) \arrow[dl, Rightarrow, "\beta"'] \arrow[dd, Rightarrow, "\mathcal{C}\left[\zeta_g^{2\varepsilon}\right]"] \\
& \mathcal{C}\mathcal{U}_\varepsilon(g) \arrow[dl, Rightarrow, "\mathcal{C}\mathcal{U}_\varepsilon[\beta]"] & & \mathcal{C}\mathcal{U}_\varepsilon(f) \arrow[dr, Rightarrow, "\mathcal{C}\mathcal{U}_\varepsilon[\alpha]"'] & \\
\mathcal{C}\mathcal{U}_{2\varepsilon}(f) & & & & \mathcal{C}\mathcal{U}_{2\varepsilon}(g)
\end{tikzcd}$$

From Theorem 3.10 we know that ρ defines a natural transformation from $\mathcal{C}\mathcal{U}_\varepsilon$ to $\mathcal{S}_\varepsilon\mathcal{C}$. We can define $\varphi = \rho_g \circ \mathcal{C}[\alpha]$ and $\psi = \rho_f \circ \mathcal{C}[\beta]$. Then, using Lemma 3.16, we add new pieces to the diagram above, constructing the following diagram:

$$\begin{tikzcd}
 & & \mathcal{C}(f) \arrow[dll, Rightarrow, "\sigma"'] \arrow[dd, Rightarrow, "\mathcal{C}\left[\zeta_f^{2\varepsilon}\right]"'] \arrow[dr, Rightarrow, "\mathcal{C}[\alpha]"] \arrow[drr, Rightarrow, "\varphi"] & & \\
S_{2\varepsilon}\mathcal{C}(f) \arrow[drr, Leftrightarrow, "\mathcal{S}_\varepsilon[\rho_f]"'] & & & \mathcal{CU}_\varepsilon(g) \arrow[r, Leftrightarrow, "\rho_g"'] \arrow[dl, Rightarrow, "\mathcal{CU}_\varepsilon[\beta]"] & \mathcal{S}_\varepsilon\mathcal{C}(g) \arrow[dll, Rightarrow, "\psi^*"] \\
 & & \mathcal{CU}_{2\varepsilon}(f) & &
\end{tikzcd}$$

Our ultimate goal is to show that this diagram commuting implies that the diagram in Figure 4 commutes. What needs to be shown is that $\mathcal{S}_\varepsilon[\rho_f] \circ \psi^* = \mathcal{S}_\varepsilon[\psi] = \mathcal{S}_\varepsilon\left[\rho_f \circ \mathcal{C}[\beta]\right]$. By definition of a functor, we have that $\mathcal{S}_\varepsilon\left[\rho_f \circ \mathcal{C}[\beta]\right] = \mathcal{S}_\varepsilon[\rho_f] \circ \mathcal{S}_\varepsilon\mathcal{C}[\beta]$. Thus, we need to show that $\mathcal{S}_\varepsilon\mathcal{C}[\beta] = \psi^* = \mathcal{CU}_\varepsilon[\beta] \circ \rho_g^{-1}$. By definition, $\rho_g : \mathcal{CU}_\varepsilon(g) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(g)$ is a family of isomorphisms. Thus, $\mathcal{S}_\varepsilon\mathcal{C}[\beta] \circ \rho_g = \mathcal{CU}_\varepsilon[\beta]$, which implies $\mathcal{S}_\varepsilon\mathcal{C}[\beta] = \mathcal{CU}_\varepsilon[\beta] \circ \rho_g^{-1}$.

Finally, we have that Figure 6 commuting implies that the diagram above commutes, which further implies that there exists natural transformations $\varphi : \mathcal{C}(f) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(g)$,$\psi : \mathcal{C}(g) \Rightarrow \mathcal{S}_\varepsilon\mathcal{C}(f)$ such that the diagram in Figure 4 commutes. Thus, we have an ε-interleaving between f and g.

$\square$

4. Computational Methods

To fully understand the complexities of computing interleaving distance (as well as other metrics) on Reeb graphs, it is important to first understand the computation behind Reeb graphs and how we work with a computational representation of scalar fields in general. For the following section, we will only consider the discrete representation of 2-manifolds, just as we have tried to remain in the 2-dimensional case in the preceeding sections.

4.1. **How data is represented.**

To create a computational representation of a scalar field $(\mathbb{X}, f)$, we need to be able to discretize both the space $\mathbb{X}$ and the function f. The discretized version of $\mathbb{X}$ should still contain all the topological information of $\mathbb{X}$ while retaining as much geometric information as needed. To do this, we introduce the notion of a *triangulation*, which stores $f(v)$ for each v in a sample set $V \subset \mathbb{X}$ and then provide a way to interpolate between these points to provide an approximation of the remaining unsampled points. Then, the discretized function is designed to be exact on the sample set V while being an approximation to the true value on points outside of V. We can expect that the larger the sample set V is, the more accurate our approximations will be. However, this comes with the trade off of having to store more information, increasing the computational complexity of any tasks using the triangulation.

Definition 4.1. An *n-simplex* is the convex hull of $n+1$ geometrically independent (affinely independent) points embedded in $\mathbb{R}^n$. A 2-simplex is a triangle, a 1-simplex is a line segment (or edge), and a 0-simplex is a single point (or vertex). The value n is known as the *dimension* of the simplex.

Definition 4.2. An *m-face* τ of an n-simplex σ is a m-simplex of lower or equal dimension that is contained within σ. For example, if σ is a 2-simplex, then the 0-faces of σ are the vertices of the triangle and the 1-faces are the edges. We denote that τ is a face of σ with the notation $\tau \leq \sigma$. If τ is strictly of less dimension than σ, then we say that $\tau < \sigma$ and call τ a *proper face.*

Definition 4.3. A *finite simplicial complex* $\mathcal{K}$ is a finite family of simplices σ that satisfies two properties: (1) If $\tau \leq \sigma$, then $\tau \in \mathcal{K}$ and (2) if $\sigma_1, \sigma_2 \in \mathcal{K}$, then $\sigma_1 \cap \sigma_2 = \sigma_3 \in \mathcal{K}$ or $\sigma_1 \cap \sigma_2 = \emptyset$. In this work, we solely work with finite simplicial complexes, so we will drop the term "finite" from now on.

Definition 4.4. The *underlying space* $|\mathcal{K}|$ of a simplicial complex $\mathcal{K}$ is the union of all the simplices in $\mathcal{K}$. That is, $|\mathcal{K}| = \cup_{\sigma \in \mathcal{K}} \sigma$.

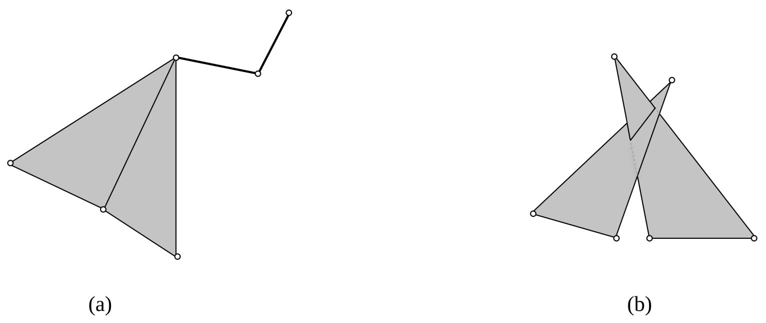

FIGURE 7. (a) A valid simplicial complex. Each simplex that insersects another simplex does so along either an edge or a vertex. (b) Not a valid simplicial complex, since the two 2-simplices intersect within the center of the face, not along another simplex.

The criteria that the intersection of two simplices is also a simplex guarantees that when we "glue" together simplices to form a complex, they must be glued along edges or vertices. Figure 7 shows an example and non-example of a simplicial complex.

Definition 4.5. Let $\mathbb{X}$ be a toplogical space. A simplical complex $\mathcal{K}$ such that the underlying space $|\mathcal{K}|$ is homeomorphic to $\mathbb{X}$ is known as a *triangulation* of the topological space $\mathbb{X}$, and is denoted as $\mathcal{T}\mathbb{X}$.

To construct a triangulation of a topological space $\mathbb{X}$, we first need to choose a finite sample set $V \subset \mathbb{X}$. This set V is the vertex set of the simplicial complex. Then we assign connections between the vertices to create faces and edges, ultimately creating the triangulation. The goal is that the triangulation would preserve all topological information needed in order to perform necessary computations on. Of course, the amount of information available is partly due to the resolution (or size) of the vertex set V. The smaller the set, the higher the chances that we are removing important toplogical information from the simplicial complex, thus making the underlying space <u>not</u> homeomorphic to the actual space $\mathbb{X}$. Furthermore, the smaller the vertex set is, the more approximations we are performing when attaining values between the vertices. On the other hand, when we increase the size of the vertex set, we also increase the runtime for any computations that are run on the simplicial complex.

Next, we need to create a function $\hat{f}$ which is the discrete analog to f. First, we define $\hat{f}$ to agree with f on all $x \in V$. Then, we use interpolation to assign values to points within the face. There are many different types of interpolations that we can use, however here we only consider the standard method of *linear interpolation.*

Definition 4.6. Suppose we have an n-simplex with vertices $\{v_0, \ldots, v_n\}$. Each point x in the interior of the n-simplex can be described as the weighted sum $x = \sum_{i=0}^{n} \alpha_i v_i$ where all coefficients $\{\alpha_0, \ldots, \alpha_n\}$ are non-negative and $\sum_{i=0}^{n} \alpha_i = 1$. These coefficients $\{\alpha_0, \ldots, \alpha_n\}$ are known as the *barycentric coordinates* of x.

Definition 4.7. Let $(\mathbb{X}, f)$ be a scalar field and $\mathcal{T}\mathbb{X}$ be the triangulation of the topological space $\mathbb{X}$ with vertex set $\{v_0, \ldots, v_n\}$. We construct a new function $\hat{f}$ by the following rule:

$$\hat{f}(x) = \sum_{i=0}^{n} \alpha_i f(v_i),$$

where $\{\alpha_0, \ldots, \alpha_n\}$ are the barycentric coordinates of x. Note that if the vertex v_i is not a vertex of the smallest simplex that contains x, then α_i is 0. The pair $(\mathcal{T}\mathbb{X}, \hat{f})$ is the discretized version of $(\mathbb{X}, f)$ and is known as the *piecewise-linear scalar field.* The term "piecewise-linear" directly refers to the fact that we are using first order interpolation on the values within the simplices.

Since each piecewise-linear scalar field is representing a continuous scalar field, we must introduce a discrete analog to a *neighborhood* of a point x. This is defined as the star of x.

Definition 4.8. Let $\mathcal{K}$ be a simplicial complex and τ be a simplex in $\mathcal{K}$. The *star* of τ is defined as all the simplices in $\mathcal{K}$ in which τ is a face. That is,

$$\mathrm{St}\ \tau = \{\sigma \in \mathcal{K} | \tau \leq \sigma\}$$

In general, the star of a simplex is not necessarily a sub-complex. For example, the star of a 0-simplex may contain a 2-simplex while still not contain all the 1-simplices that make up that 2-face. This is because there is no need for the other 1-simplices in this 2-simplex to connect directly to the original 0-simplex. To alleviate this issue, we introduce the idea of the *closed star*, denoted as $\bar{St}$, which is defined as the smallest sub complex that contains the star.

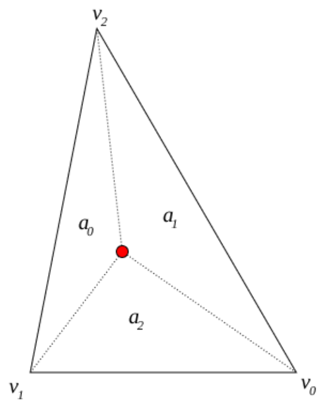

Figure 8. The barycentric coordinates of the red vertex can be found by taking the ratio of area made up by each triangle a_0, a_1 and a_2 when compared to the ratio of the triangle in total. That is, if T is the total area of the triangle, then $x = \frac{a_0}{T}v_0 + \frac{a_1}{T}v_1 + \frac{a_2}{T}v_2$.

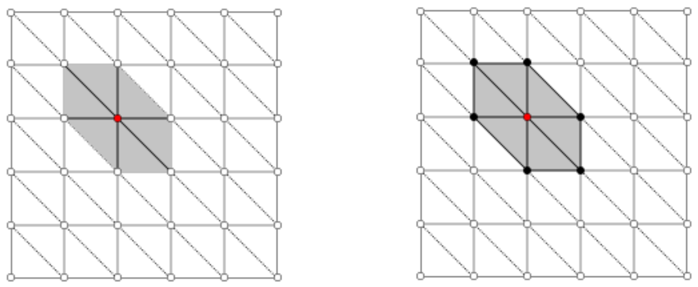

Figure 9. On the left, we have a simplicial complex with the star of a vertex (red) highlighted. The grey represents the 2-simplices that are contained within the star, and the thickened lines represent the 1-simplices that contained in the star. On the right we have the closed star of the same vertex. Note the additional 1-simplices contained in this subcomplex as well as all of the outer vertices that were absent in the star.

4.2. **Determining Critical Points.**

Piecewise-linearity provides us with easy ways to determine where critical points of the discretized scalar field are. While higher-order interpolations are useful in many situations, first order interpolation guarantees that each critical point of this triangulation must lie on one of its vertices.

To see this, suppose we have vertices $\{v_0, \ldots, v_n\}$ with corresponding function values $\{f(v_0), \ldots, f(v_n)\}$. We want to show that

$$\min\{f(v_0), \ldots, f(v_n)\} \leq \alpha_0 f(v_0) + \ldots + \alpha_n f(v_n) \leq \max\{f(v_0), \ldots, f(v_n)\}.$$

Without loss of generality, suppose that $f(v_0) = \min\{f(v_0), \ldots, f(v_n)\}$ and $f(v_n) = \max\{f(v_0), \ldots, f(v_n)\}$. Then, we have that

$$(\alpha_0 + \ldots + \alpha_n) f(v_0) \leq \sum_{i=0}^{n} \alpha_i f(v_i) \leq (\alpha_0 + \ldots + \alpha_n) f(v_n).$$

It follows that $f(v_0) \leq \sum_{i=0}^{n} \alpha_i f(v_i) \leq f(v_n)$ since the sum of the barycentric coordinates is 1. Thus, no interior point within the simplex can have higher (or lower) function value than its maximum (or minimum). This implies that that critical points can only occur on the actual vertices of the triangulation.

Definition 4.9. Suppose we have a piecewise-linear scalar field $(\mathcal{T}\mathbb{M}, \hat{f})$. We define the *lower star* of a vertex v_i, denoted as $\mathrm{St}_{-}v_i$, to be the simplices in the star of v_i such that all points within these simplices have lower function value than v_i. That is,

$$\mathrm{St}_{-}v_i = \{\sigma \in \mathrm{St}\ v_i | x \in \sigma \Rightarrow f(x) \leq f(v_i)\}.$$

Similarly, the *upper star* of v_i, denoted as $\mathrm{St}_{+}v_i$, is defined as

$$\mathrm{St}_{+}v_i = \{\sigma \in \mathrm{St}\ v_i | x \in \sigma \Rightarrow f(x) \geq f(v_i)\}.$$

Based on the structure of either the lower star or upper star of a vertex v_i, we can determine whether or not we have a maximum, minimum, saddle, or just a regular vertex. Figure 10 shows us how we can classify these points.

4.3. **Computing the Reeb Graph.**

To compute the Reeb graph, we need some sort of information about how the topology is changing when we pass critical points. A perhaps naive way to do this would be to construct a level set between each critical point (using something like the triangular variant of the Marching Cubes algorithm [21]) and then investigate how the level set changes as we move past the critical point. In practice, we would like to avoid the computationally intensive task of computing multiple different level sets. Instead, here we will describe a technique in which we begin with a contour and track how that fiber must be changed based on the critical points that we pass [10]. Instead of computing each fiber individually, we are simply making adjustments to the previous one. On top this, we will construct the Reeb graph as we go, as certain critical types contribute very specific structure to the Reeb graph.

To begin, we assume that we are working on a two-dimensional piecewise-linear manifold $(\mathcal{T}\mathbb{M}, \hat{f})$. Just as with the continuous case, this makes each level set homeomorphic to 1-spheres. To represent these 1-spheres in the discrete case, we can simply track the triangles that the 1-sphere passes through. Then, each level set is represented as a cyclic list of triangles from the triangulation. This list must also be able to keep track of the connections between triangles in the fiber as these connections may need to be altered as we change the contours.

We begin by sorting the vertices in order of increasing function value. Without loss of generality, we say that $\hat{f}(v_i) < \hat{f}(v_j)$ for all $i < j$. We traverse the vertices in this order and track how the level set changes as we pass each vertex. Note that in this discrete case, the contour is represented the same for each value s_i between $\hat{f}(v_i)$ and $\hat{f}(v_j)$.

There are four different types of vertices v_i and each contribute different operations to algorithm:

Case 1: v_i **is a minimum**. This means that we initialize a new level set which consists of all triangles in the star of the vertex v_i. In addition to this, we had a degree 1 node to the Reeb graph which begins a new arc associated with this level set.

Case 2: v_i **is a maximum**. We remove all triangles in the star (or equivalently the lower star) from the list triangles and end the arc associated with this level set by placing a degree 1 node to the Reeb graph.

Case 3: v_i **is a regular vertex**. In this case, we make no adjustments to the Reeb graph, but we still need to worry about the alterations to the fiber. To adjust for passing from below the vertex to above the vertex, we take the lower star triangles in the list and replace them with the upper star triangles.

Case 4: v_i **is a saddle**. A simple example of a saddle point would be two hills that are sufficiently close to each other. If we look at the fibers, moving upward in function value, we see that we begin with one level set which then splits into two separate level sets. Conversely, another example would be two troughs that are sufficiently close to each other. In this case, we begin with two level sets which then merge into one as we pass the saddle point. These two situations can be boiled down into one question: do we begin with one fiber or two separate fibers? In the discrete case, we can ask whether or not the triangles in the lower star of the saddle point are contained in the same level set or two different level sets. If two level sets merge into one, we add a *down fork*, which is a degree three node with two arcs merging upwards towards one. If one level sets split into two, we add an *up fork*, which is a degree three node with one arc splitting into two. Just as in **Case 3**, we replace the lower star triangles with the corresponding upper star triangles in the list of triangles.

For example in Figure 10 we have a saddle which has two lower star sections consisting of two consecutive triangles each. If these two sets of triangles are in the same fiber (meaning they are in the same list of triangles), then we know that the upper star triangles are in two different fibers. Thus, we place an up fork in the Reeb graph and split the existing list of triangles into two separate lists. To split this list accordingly, we remove the lower star triangles and replace them with the corresponding upper star triangles. An example of this can be seen in Figure **??**.

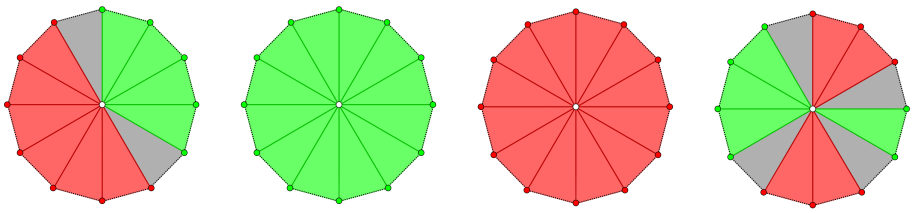

FIGURE 10. Above we have various types of vertices shown. Each vertex is connected to 12 other vertices. Red vertices indicate that the function value is below the center vertex, while the green indicates that it is above the center vertex. Simplices highlighted red are in the lower star of the vertex, while simplices highlighted green are in the upper star. The grey simplices are not classified as either due to them including points that are both above and below the center vertex. From left to right, we have a regular vertex, a minimum, a maximum, and a saddle.

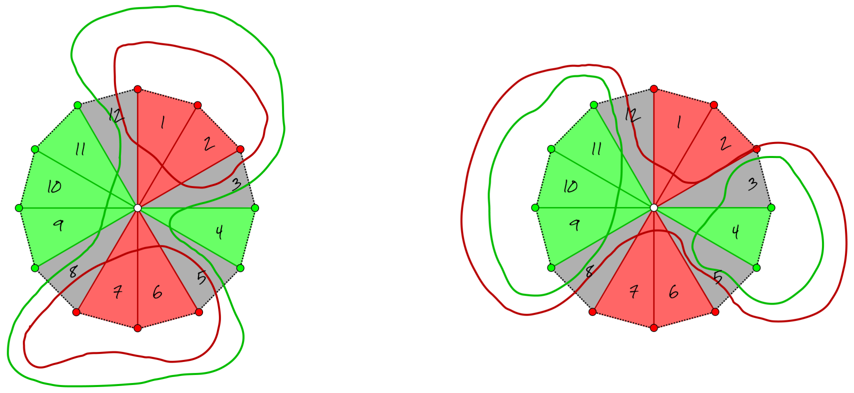

FIGURE 11. Above we have two saddle points with the same star structure, but with the two different cases of fibers. In each, the green lines represent the contours above the saddle point and the red contours represent the fibers below the saddle point. On the left hand side, there are two lower contours and one upper contour. We can see that as we pass the saddle point, we need to join the two lists of triangles (which include the triangles {12,1,2,3} and {5,6,7,8}) into one single list of triangles (which includes the triangles {3,4,5,8,9,10,11,12}). Similarly, the saddle point on the right has one lower contour and two upper contours, which means we need to split the list of triangles into two separate lists.

4.4. **How interleaving distance is computed.** To compute interleaving distance, we would need to be able to (1) compute a ε-smoothed Reeb graph, (2) verify if an ε-interleaving exists between two Reeb graphs, and (3) find the optimal ε which creates an interleaving. It turns out that even with a smoothing algorithm which is computed in linear time, (2) can be shown to be in **NP** [28]. Despite the challenges, here we discuss how we compute a ε-smoothed Reeb graph from the original.

One way to construct a smoothed Reeb graph might be to take the Reeb graph $\mathbb{X}$, then create the smoothed space $\mathbb{X} \times [-\varepsilon, \varepsilon]$, and finally recompute the Reeb graph on the new thickened space. However, the triangulation of the new space actually follows a very specific pattern: each vertex of the Reeb graph would be replaced with two new vertices and one new edge connecting these two vertices, and then each edge would be replaced with three new edges and two 2-simplices. For example, a single 1-simplex would be changed to a simplicial complex with 4 vertices in a rectangular fashion, with an edge two of the vertices diagonally. Because of this increase in complexity, if the original graph has m edges and n vertices, the new complex has $O(m+n)$ total simplices.

Here, we take advantage of the predictable nature of the smoothing algorithm. The key piece to note is that given a Reeb graph $(\mathbb{X}, f)$ and its ε-smoothed counterpart $(\mathbb{X}_\varepsilon, f_\varepsilon)$, we have that $f_\varepsilon^{-1}(I)$ is homotopic to $f^{-1}(I^\varepsilon)$. So, the a-fiber of the smoothed Reeb graph has the same number of path connected components as the pre-image of the interval $[a-\varepsilon, a+\varepsilon]$.

This smoothing algorithm works similarly to how the actual constrcution of the Reeb graph works: we begin building the smoothed Reeb graph by traversing upwards on the original Reeb graph. As we pass each critical point, depending on the local structure of the critical point, we will add arcs and nodes to the smoothed Reeb graph. This process continues until we have done a full sweep of the original Reeb graph.

Let $\mathcal{S} = \{v_1, \dots, v_n\}$ be the set of vertices of the Reeb graph, i.e. the critical points of the original scalar field, such that $f(v_1) < f(v_2) < \dots < f(v_n)$. Furthermore, to simplify this explanation, suppose the vertices are such that $f(v_i) \pm \varepsilon \neq f(v_j) \pm \varepsilon$, for all $i \neq j$. Then, the set of possible critical values for the thickened space would be $\{f(v_i) \pm \varepsilon\}$. To see this, suppose we have a critical point v_i with critical value $f(v_i)$. Suppose also that we have an interval $(a-\varepsilon, a+\varepsilon)$. If $a + \varepsilon = f(v_i) - \delta$ for some $\delta > 0$, then the interval is directly below the critical value $f(v_i)$. As δ approaches 0 (and the interval moves upward), we get closer to passing over $f(v_i)$. Since $f(v_i)$ is a critical value of the original Reeb graph, it is possible that the pre-image $f^{-1}(a-\varepsilon, a+\varepsilon)$ will introduce or delete connected components when $f(v_i)$ enters the interval. This happens exactly when $a = f(v_i) - \varepsilon$ (or similarly when $a = f(v_i) + \varepsilon$.

Let $b_1 < b_2 < \dots < b_k$ be the sorted values of $\{f(v_i) \pm \varepsilon\}$ and denote the interval $[a-\varepsilon, a+\varepsilon]$ as $[a]^\varepsilon$ for ease of notation. Now, suppose that $b_i = f(v_j) - \varepsilon$. As the center of our interval a increases in value and we eventually pass b_i, our interval now includes the vertex v_j. Similarly, if $b_i = f(v_j) + \varepsilon$, then as we pass b_i, the vertex v_j now exits the interval. These are the only situations where the connected components of the Reeb graph can change (by definition of critical point). Instead of thinking of this interval smoothly increasing in value, we need only worry about the changes that might or might not occur at the values $\{b_1, \dots, b_k\}$. We define the *pre-components* to be the components before the pass over b_i, and the *post-components* as the components after thw pass over b_i.

To keep track of the changes to connected components as we iterate through these possible critical values, we introduce a data structure which we will call H^a. When viewing the preimage of the thickened interval $[a]^\varepsilon$, depending on where a is, it is possible that we have partial edges where one or more endpoints of the edge are outside the interval. Instead of storing this pre-image how it is, we introduce a new "augmented" graph, denoted as H^a. Each vertex of $f^{-1}([a]^\varepsilon)$ has an associated vertex in H^a, each edge (partial or full) in $f^{-1}([a]^\varepsilon)$ has an associated vertex in H^a, and the incidence between vertex and edge in $f^{-1}([a]^\varepsilon)$ is represented as an edge in H^a. This makes H^a a well-defined graph.

Once we know what the pre-components and post-components are, we are ready to update the new Reeb graph. If the lower components and the upper components are the same, then we don't need to update the Reeb graph at all since this implies that no connected components have changes. If they differ, then we first add a vertex β_i whose function value is equal to b_i. Then, for every pre-component, we assign β_i as the end vertex, and for every pre-component, have have an edge that starts at β_i.

In our Full Sweep Algorithm, we initialize our graph H to be an empty graph. We then iterate through each of the potential critical points b_i in increasing order. We then call the methods to find the pre-components, then update the graph H, and then find the post components. Afterwards, we update the Reeb graph based on the values of `Pre` and `Post` that are returned.

Algorithm 1 Full Sweep Algorithm

```
Set $H$ to be an empty graph
for $i = 1, \dots, k$ where $b_i = f(v_j) \pm \varepsilon$ do
    Pre = PreComps($v_j, b_i$)
    UpdateH($v_j, b_i$)
    Post = PostComps($v_j, b_i$)
    UpdateReebGraph(Pre,Post)
```

The UpdateH(v,b) decides how to update our fiber representation based on the whether b_i is above or below the corresponding v_j. If it's below, then we know that we are going to introduce the vertex v_j into the subgraph H. If it is above v_j, then we know that we need to remove it and the corresponding edges.

Algorithm 2 UpdateH(v,b)

```
if $b < f(v)$ then
  Add a vertex $H$ for $v$
  for $e$ ending at $v$ in $\mathbb{X}$ do
    H.insert($e$,$v$)
  for $e$ starting at $v$ in $\mathbb{X}$ do
    Add a vertex to $H$ for $e$
    H.insert($v$,$e$)
else
  for $e$ ending at $v$ in $\mathbb{X}$ do
    H.delete($e$,$v$)
    Remove vertex $e \in V(H)$
  for $e$ starting at $v$ in $\mathbb{X}$ do
    H.delete($v$,$e$)
  Remove vertex $v \in V(H)$
```

The PreComps(v,b) and PostComps(v,b) methods are very similar to one another. If $b < f(v)$, to find the pre-components, we want to find all the edges that are supposed to be connected to the vertex that we introduce. For every edge we find, we find its corresponding connected component. Then, the list of connected components are stored in `Pre`. If we are above $f(v)$, then instead we just find the single connected component for this vertex. Similarly, if $b > f(v)$, to find the post components, we search for all edges which start at this vertex v and store it in `Post`. If $b < f(v)$, then the upper components are just the root of the current H. Note that in between these two methods we have actually updated H, so these are in no way acting on the same structure.

Algorithm 3 PreComps(v,b)

```
if $b < f(v)$ then
  for $e$ ending at $v$ in $\mathbb{X}$ do
    c = $H$.find($e$)
    if c is not marked then
      Pre.add(c)
      Mark c as listed
else
  Pre = $H$.find($v$)
return Pre
```

Algorithm 4 PostComps(v,b)

```
if $b > f(v)$ then
  for $e$ starting at $v$ in $\mathbb{X}$ do
    c = $H$.find($e$)
    if c is not marked then
      Post.add(c)
      Mark c as listed
else
  Post = $H$.find($v$)
return Post
```

Figure 12 shows an example of how the smoothing algorithm would work on a small example. We show the first nine iterations of the algorithm, showing what the values of `Pre`, `Post`, and H are along the way. Generally speaking, to find the pre-components, we always need to look at the value of H in the previous iteration. So, in iteration 2, we are using H from iteration 1 to find the pre-components. In iteration 2, the pre-components is simply just v_1 since this is the root of the only tree whose edge ends in the new vertex, v_2. Then, we update H based on the fact that $b_2 < f(v_2)$. Afterwards, we find the post components by using the current H that we just updated. In this case, it just finds the root of the tree which is v_1. Since the pre and post components are the same, this means that no component was introduced or deleted, so we do not alter the Reeb graph. The next iteration would then use the H in iteration 2 when we need to determine the pre-components, and the process continues.

In the end, the smoothed Reeb graph has four critical points, $\beta_1, \beta_2, \beta_3, \beta_4$ whose function values are equivalent to b_1, b_4, b_8, b_{12}, respectively.

In [28], they show that the question "Does there exist an ε-interleaving between $(\mathbb{X}, f)$ and $(\mathbb{Y}, g)$?" is a problem in **NP**. So, of course, the problem of finding the optimal ε becomes an extremely complex problem.

5. Future Direction and Proposed Work

5.1. **Metrics for Topological Structures Survey/Empirical Study.** There are several reasons that we believe that topological analysis will be useful for multi-faceted data such as the continued use of topological analysis in unifaceted data as well as topology's natural robustness to small perturbations of position and magnitude of features.

Interleaving distance has provided a metric on Reeb graphs which has a strong mathematical foundation, yet is incredibly difficult to compute. Similarly, other metrics defined on Reeb graphs such as functional distortion distance [4] and the Reeb graph edit distance [14] suffer from the same computational complexity but are rooted in a strong theoretical background.Due to the difficulty implementing current distances on Reeb graphs, it's difficult to understand how useful these distances are when using them from a data analysis perspective.

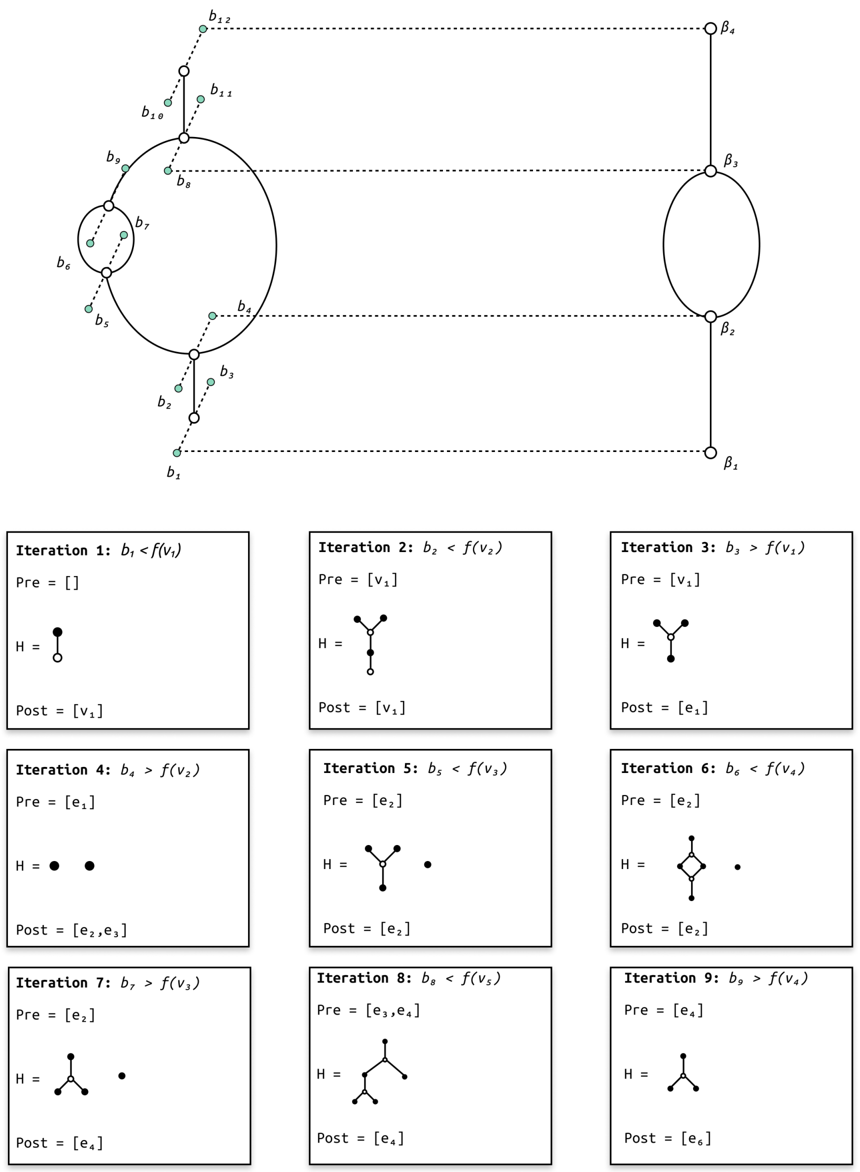

FIGURE 12. Above we have displayed the first twelve iterations of the smoothing algorithm. The original Reeb graph is on the left-hand side, accompanied by additional teal vertices which are placed ε above or below the original critical point. This set of vertices contains all possible critical points for the smoothed Reeb graph. In the iterations below, we show what values are in the pre and post components, as well as how the tree **H** looks during that iteration.

In contrast, there has been success in defining distances on other topological structures such as the persistence diagram. Both the bottleneck distance and wasserstein distance [12] are common tools when discussing distance between these diagrams. However, there has been work showing that interleaving distance on merge trees is bounded below by bottleneck distance [23], implying that interleaving distance is no less sensitive to perturbations than the bottleneck distance is.

Our first major goal for future work would be to **create a comprehensive survey/empirical study of the various metrics that are used to measure similarity between topological structures and evaluate their use in multifaceted data analysis.** We seek to find the relationships between these metrics based on where each metric performs well and where each falter. In addition, we plan to identify the computational complexities in implementing each of these metrics in detail. A short, incomplete summary of distances we plan on investigating is below:

Metric	**Topological Structure**	**Notes**	**Related Work**
Interleaving Distance	Reeb Graphs	Fine-grained, NP-hard	[28],[23]
Interleaving Distance	1-D Barcodes	Equivalent to Bottleneck Distance	[20]
Functional Distortion Distance	Reeb Graphs	Fine-grained, NP-hard, Strong equiv. to interleaving dist	[4]
Bottleneck Distance	Persistence Diagrams	Bounded by interleaving for Merge Trees	[23]
Wasserstein Distance	Persistence Diagrams	————	[12]

For this project to be successful, we would like to be able to build a comprehensive paper that is useful for those deciding which metrics are important when analyzing data. We would like to support our claims for how well these metrics work with empiracal evidence when we are able to. Even though certain metrics for Reeb graphs are difficult to compute for large graphs, we think that it is possible to still provide key examples with small graphs that will illustrate their properties. We believe that this survey will be published by March, 2021.

We denote this project as **Milestone 0** in Figure 13.

5.2. **Applying Machine Learning Techniques to Metrics on Topological Structures.** As stated before, we know that there are several metrics that are mathematically sound but difficult to implement. Our second major goal is to **utilize contemporary machine learning techniques to aid in the computation of and/or approximate these metrics**. There are several different techniques we have considered for applying machine learning to these areas:

(A) Utilize deep learning to speed up computation of modules that are difficult to compute with traditional methods
(B) Utilize Graph Neural Networks and other deep learning approaches to learn similarity metrics on topological structures in a *supervised* manor
(C) Utilize Graph Neural Networks and other deep learning approaches to learn similarity metrics on topological structures in an *unsupervised* manor

Here we provide a overview of the techniques described above. We give an introduction to Graph Convolutional Neural Networks, which have recently seen an increase in utilization to classify and label graphs is various settings [29].

5.2.1. *Background.* A *Graph Convolutional Neural Networks*, or GCNN, utilizes a similar structure to that of a Convolutional Neural Network but is used to classify local and global behavior of graphs specifically. There are two main types of GCNN's: Spatial and Spectral. The Spectral GCNN's have a rich mathematical foundation based in signal processing [27],[6]. While the spectral method is useful in many situations, if the graphs in question have varying topologies (as opposed to just varying node information), the computational comnplexity increases drastically since the key component of this method involves computing the discrete Laplacian (which varies as the topology of the graph varies) [29]. On the other hand, Spatial GCNN' determines the updated state for a target node solely by utilizing the features of the nodes in its neighborhood [29], which makes it more flexible for ensembles of graphs with varying topologies.

The input of a GCNN is a graph $G = (V, E)$ along with a set of node attributes $X = \{x_1, x_2, \ldots, x_n\}$, where x_i is the *feature vector* for node v_i. In [25], researchers also take edge features into account, denoted as X_e. Node feature vectors consist of attributes that we determine are important to the learning of the network. For example, a graph modeling a social network where each node would be an individual, the feature vector might consist of some demographic information. The GCNN updates each feature vector at each hidden layer and then outputs a set of *hidden feature vectors* $\{h_1, h_2, \ldots, h_n\}$. We can think of these hidden feature vectors as being latent variables that are embedding the node information into lower-dimensional space. If we want to classify the graph as a whole, a readout layer is added to the end of the neural network which pools the individual hidden feature vectors together to create a *hidden graph feature vector* h_G. The pooling can be as simple as summation across the hidden node feature vectors. If the input also consists of edge attributes, then we would incorporate the resulting hidden edge feature vectors as well.

Recently, GCNN's have been coupled with a *siamese architecture* to learn a similarity metric between graphs constructed from brain networks [18]. A siamese architecture takes two CNNs with identical weights and applies additional readout layers which determine a distance between the two readouts of the CNNs. The CNNs are then trained to minimize the distance between similar (positive) pairs and then maximize the distance between dissimilar (negative) pairs. In [18], they used Spectral GCNNs as opposed to Spatial GCNNs.

5.2.2. *Supervised Similarity Metric Learning via Siamese, Spatial GCNNs.* We propose using a siamese architecture where the two neural networks are Spatial GC-NNs, similar to those utilized in [25] and [15]. In both these these architectures, a convolution is achieved by constructing a 1-D (or 2-D) vector space from the neighborhood of the target node which creates a well-ordered list of nodes. This makes the receptive field similar to a receptive field used in regular CNNs.

To perform supervised training for this siamese network, we need to have pairs of inputs which are labeled as being positive or negative pairs, meaning we need to already have some notion of closeness between the Reeb graphs. We propose three different ways of computing similarity between these scalar fields for training purposes: (1) interleaving distance of the Reeb graphs, (2) wasserstein/bottlneck distance of persistence diagrams constructed from the scalar fields, and (3) any contemporary geometric way of measuring similarity between scalar fields directly.

There are several challenges that we will need to overcome to be successful with this project:

(1) Determining necessary node and edge feature information
(2) Creating a ranking and labeling algorithm that creates adequate receptive fields
(3) Computational complexities of creating training sets

To solve challenge **(1)**, we need to determine what features of these nodes and edges are used to describe their role in the Reeb graph. Some example attributes we can use are function value, critical type (using the standard index numbering system), and position of critical point in the scalar field. As for edges, we think it might be worthy to assign the number of voxels that a particular edge passes through to move from one critical point to the next. We believe that geometric information that we can encode into the Reeb graph might be useful for finding this metric.

The choices we make for the node and edge feature information directly affect the solution to challenge **(2)**. Different information encoded in the nodes and edges implies different ranking algorithms for neighborhoods. A ranking can be something as simple as an ordering based on function values, or something more complex such as a combination of function value, critical type, position of the critical point in the scalar field, and volume passed through to traverse the edge in the scalar field.

The difficulty of challenge **(3)** directly correlates to the metric that we are using to train the data. We think that an implementation of something like interleaving distance might be extremely difficult, while other metrics such as Wasserstein distance and bottleneck distance are known to be computationally reasonable [12].

Success in this context would involve a working architecture that can
We denote this project as **Milestone A** in Figure 13

5.2.3. *Deep Learning for solving optimization and graph matching problems.* To determine whether two Reeb graphs $(\mathbb{X}, f)$ and $(\mathbb{Y}, g)$ are 0-interleaved is a question of asking whether or not the Reeb graphs are isomoprhic. A simple test to determine if they are not isomoprhic is the common WL test described in [19] which states that, based on a recursive labeling algorithm, if two graphs end with different labels then they cannot be isomorphic. The converse in this situation is not necessarily true. Later, an extension to this WL-test was introduced, coined as the k-WL test [1]. It can be shown that there exist graphs such that the k-WL test can show to be non-isomorphic that the $(k-1)$-WL test cannot.

With the introduction of GCNNs, researchers have tried to search for the answer to the question "How powerful are GCNNs?". One paper [30] attempts to quantify this by comparing the ability for a GCNN to discern between non-isomorphic graphs just as the 1-WL test can. They set a list of criteria for the various layers of the GCNN so that it will be just as powerful as the 1-WL test. Later, researchers developed criteria for making GCNNs just as powerful as the k-WL test.

We believe that due to the applications of GCNNs to determine if graphs are non-isomorphic, that it is possible to apply these types of GCNNs to the interleaving distance problem. Furthermore, since machine learning is heavily used in optimization problem, it is interesting to us to investigate the possibility of utilizing these techniques for the optimality question as well.

Continuing with this, we are interested in understanding which metrics that are difficult to compute, such as interleaving distance and functional distortion distance, can be aided with machine learning techniques. Instead of approximating these distances like in **Milestone A**, we will specifically search for modules within the standard pipeline for computing these metrics which can be simplified using machine learning.

If we can make significant contributions to computing interleaving distance or other metrics through applying machine learning, we would consider this project a success.

We denote this project as **Milestone B** in Figure 13.

5.2.4. *Unsupervised Similarity Metric Learning via Graph Autoencoders.* Recently, there has been work in adversarial, unsupervised learning using graph autoencoders [26]. A graph autoencoder is a type of deep learning approach to take embed a graph G along with its node features X into a lower-dimensional, latent space. We can call this latent representation Z. Afterwards, a decoder is used to attempt to reproduce G from Z. This approach tries to minimize the discrepancy between the original G and the reconstructed version G', stating that this implies that Z is a good, lower-dimensional representation of G. The autoencoders can be a GCNN like we have described above, or from some other set of deep learning approaches. Once the graphs are projected into this latent space, we can use more standard metrics to measure the distance between graphs, such as the L_2 norm.

To measure the success, we would first like to test how well the neural network discriminates between two non-isomoprhic Reeb graphs. Furthermore, we would like to see how well it compares to other metrics, such as interleaving distance, when analyzing multifaceted data.

We denote this project as **Milestone C** in Figure 13.

5.3. **Research Timeline.** Our first Milestone, denoted as Milestone 0 is intended to be completed and submitted for publication by March of 2021. After this milestone, we have the choice of going into any of the three Milestones A, B, and C. However, Milestone A is heavily dependent on the success of Milestone 0. We would need to make sure that we have an adequate understand of how each metric studies similarities in Reeb graphs (and other structures) as well as how difficult they are to implement. For the metrics we are able to implement, we can use to develop training sets for the similarity learning. Milestone B does not rely on the success of Milestone 0 as much. Of course, having a deep understanding of the computational challenges we face for each metric would be ideal when deciding how best to apply deep learning to aid in these challenges, we will still be able to focus on specific metrics (such as interleaving distance) for our study of how machine learning can be applied. In other words, we believe that the application of deep learning specifically to interleaving distance is significant in and of itself. Lastly, Milestone C can be completed without any success from Milestone 0, since unsupervised learning does not require us to use these metrics for training sets.

We have designed this research plan to be flexible, attainable, and robust to difficulties that we might see in the future. We believe that significant contributions can be made to this area in terms of each of these milestones which will lead to multiple publications and eventually a PhD dissertation. A summary graphic of our research plan is shown below in Figure 13:

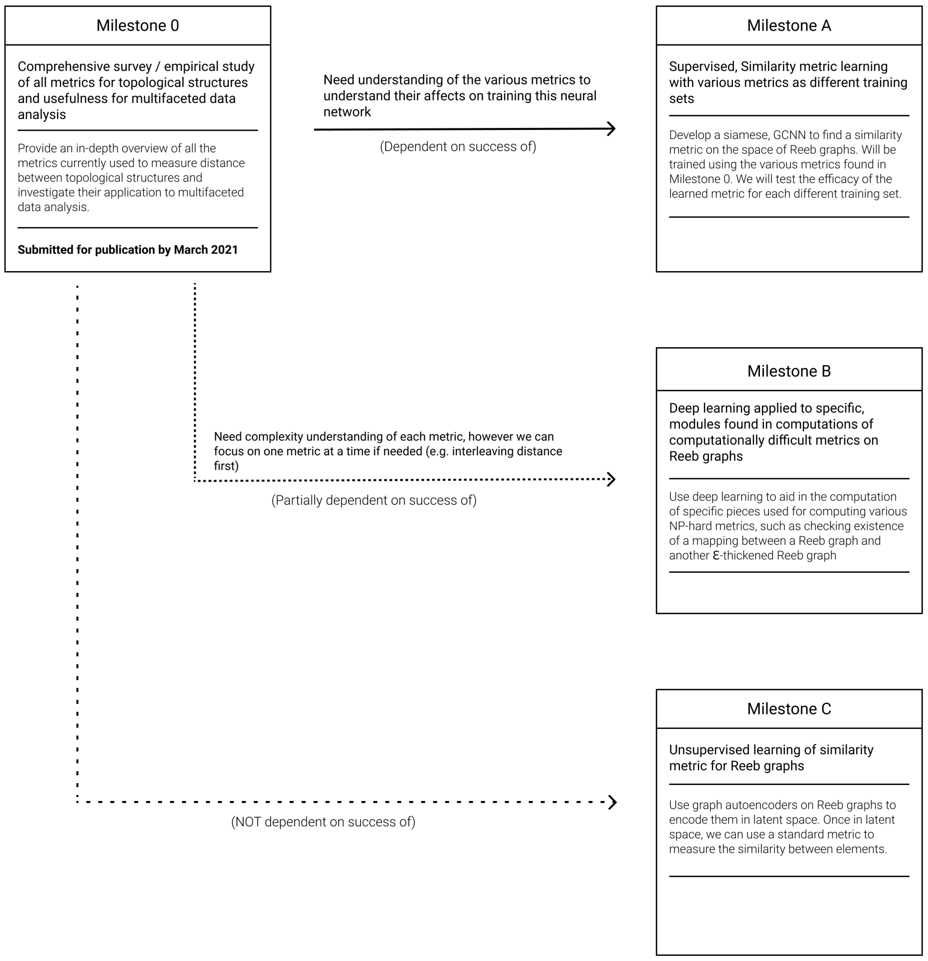
Milestone 0
Comprehensive survey / empirical study of all metrics for topological structures and usefulness for multifaceted data analysis
Provide an in-depth overview of all the metrics currently used to measure distance between topological structures and investigate their application to multifaceted data analysis.
Submitted for publication by March 2021
Need understanding of the various metrics to understand their affects on training this neural network
(Dependent on success of)
Milestone A
Supervised, Similarity metric learning with various metrics as different training sets
Develop a siamese, GCNN to find a similarity metric on the space of Reeb graphs. Will be trained using the various metrics found in Milestone 0. We will test the efficacy of the learned metric for each different training set.
Need complexity understanding of each metric, however we can focus on one metric at a time if needed (e.g. interleaving distance first)
(Partially dependent on success of)
Milestone B
Deep learning applied to specific, modules found in computations of computationally difficult metrics on Reeb graphs
Use deep learning to aid in the computation of specific pieces used for computing various NP-hard metrics, such as checking existence of a mapping between a Reeb graph and another ε-thickened Reeb graph
(NOT dependent on success of)
Milestone C
Unsupervised learning of similarity metric for Reeb graphs
Use graph autoencoders on Reeb graphs to encode them in latent space. Once in latent space, we can use a standard metric to measure the similarity between elements.

FIGURE 13. Graphic of the milestones outlined in the Research Timeline section.

References

[1] László Babai, Paul Erdös, and Stanley Selkow. "Random Graph Isomorphism". In: *SIAM J. Comput.* 9 (Aug. 1980), pp. 628–635. DOI: 10.1137/0209047.

[2] U. Bauer, B. D. Fabio, and C. Landi. "An Edit Distance for Reeb Graphs". In: *3DOR@Eurographics*. 2016.

[3] Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. "Measuring Distance between Reeb Graphs". In: *CoRR* abs/1307.2839 (2013). arXiv: 1307.2839. URL: http://arxiv.org/abs/1307.2839.

[4] Ulrich Bauer, Elizabeth Munch, and Yusu Wang. *Strong Equivalence of the Interleaving and Functional Distortion Metrics for Reeb Graphs*. 2014. arXiv: 1412.6646 [math.AT].

[5] Silvia Biasotti et al. "Reeb graphs for shape analysis and applications". In: *Theoretical Computer Science* 392 (Feb. 2008), pp. 5–22. DOI: 10.1016/j.tcs.2007.10.018.

[6] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *CoRR* abs/1312.6203 (2014). arXiv: 1312.6203. URL: https://arxiv.org/abs/1312.6203.

[7] Hamish Carr, Jack Snoeyink, and Ulrike Axen. "Computing Contour Trees in All Dimensions". In: *Computational Geometry* 24 (Aug. 2002). DOI: 10.1145/338219.338659.

[8] Frédéric Chazal et al. "Proximity of Persistence Modules and Their Diagrams". In: *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*. SCG '09. Aarhus, Denmark: Association for Computing Machinery, 2009, pp. 237–246. ISBN: 9781605585017. DOI: 10.1145/1542362.1542407. URL: https://doi.org/10.1145/1542362.1542407.

[9] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of Persistence Diagrams". In: *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*. SCG '05. Pisa, Italy: Association for Computing Machinery, 2005, pp. 263–271. ISBN: 1581139918. DOI: 10.1145/1064092.1064133. URL: https://doi.org/10.1145/1064092.1064133.

[10] Kree Cole-McLaughlin et al. "Loops in Reeb Graphs of 2-Manifolds". In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG '03. San Diego, California, USA: Association for Computing Machinery, 2003, pp. 344–350. ISBN: 1581136633. DOI: 10.1145/777792.777844. URL: https://doi.org/10.1145/777792.777844.

[11] Leila De Floriani et al. "Morse complexes for shape segmentation and homological analysis: discrete models and algorithms". In: *Computer Graphics Forum* 34.2 (2015), pp. 761–785. DOI: 10.1111/cgf.12596. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12596. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12596.

[12] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Applied Mathematics. American Mathematical Society, 2010. ISBN: 9780821849255. URL: https://books.google.com/books?id=MDXa6gFRZuIC.

[13] H. Edelsbrunner, D. Letscher, and A. Zomorodian. "Topological persistence and simplification". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 2000, pp. 454–463.

[14] Barbara Di Fabio and Claudia Landi. “The edit distance for Reeb graphs of surfaces”. In: *CoRR* abs/1411.1544 (2014). arXiv: `1411.1544`. URL: `http://arxiv.org/abs/1411.1544`.

[15] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. “Large-Scale Learnable Graph Convolutional Networks”. In: *CoRR* abs/1808.03965 (2018). arXiv: `1808.03965`. URL: `http://arxiv.org/abs/1808.03965`.

[16] Robert Ghrist. “Barcodes: The persistent topology of data”. In: *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY* 45 (Feb. 2008). DOI: `10.1090/S0273-0979-07-01191-3`.

[17] A. Gyulassy et al. “A Practical Approach to Morse-Smale Complex Computation: Scalability and Generality”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1619–1626.

[18] Sofia Ira Ktena et al. “Distance Metric Learning using Graph Convolutional Networks: Application to Functional Brain Networks”. In: *CoRR* abs/1703.02161 (2017). arXiv: `1703.02161`. URL: `http://arxiv.org/abs/1703.02161`.

[19] A. Leman. “THE REDUCTION OF A GRAPH TO CANONICAL FORM AND THE ALGEBRA WHICH APPEARS THEREIN”. In: 2018.

[20] Michael Lesnick. “The Optimality of the Interleaving Distance on Multidimensional Persistence Modules”. In: *CoRR* abs/1106.5305 (2011). arXiv: `1106.5305`. URL: `http://arxiv.org/abs/1106.5305`.

[21] William E. Lorensen and Harvey E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. ISBN: 0897912276. DOI: `10.1145/37401.37422`. URL: `https://doi.org/10.1145/37401.37422`.

[22] William E. Lorensen and Harvey E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: `10.1145/37402.37422`. URL: `https://doi.org/10.1145/37402.37422`.

[23] D. Morozov, Kenes Beketayev, and G. Weber. “Interleaving Distance between Merge Trees”. In: 2013.

[24] J.R. Munkres. *Topology*. Featured Titles for Topology. Prentice Hall, Incorporated, 2000. ISBN: 9780131816299. URL: `https://books.google.com/books?id=XjoZAQAAIAAJ`.

[25] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. “Learning Convolutional Neural Networks for Graphs”. In: *CoRR* abs/1605.05273 (2016). arXiv: `1605.05273`. URL: `http://arxiv.org/abs/1605.05273`.

[26] Shirui Pan et al. “Adversarially Regularized Graph Autoencoder”. In: *CoRR* abs/1802.04407 (2018). arXiv: `1802.04407`. URL: `http://arxiv.org/abs/1802.04407`.

[27] David I. Shuman et al. “Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Data Domains”. In: *CoRR* abs/1211.0053 (2012). arXiv: `1211.0053`. URL: `http://arxiv.org/abs/1211.0053`.

[28] Vin de Silva, Elizabeth Munch, and Amit Patel. “Categorified Reeb Graphs”. In: *CoRR* abs/1501.04147 (2015). arXiv: `1501.04147`. URL: `http://arxiv.org/abs/1501.04147`.

[29] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596. URL: http://arxiv.org/abs/1901.00596.

[30] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *CoRR* abs/1810.00826 (2018). arXiv: 1810.00826. URL: http://arxiv.org/abs/1810.00826.

[31] Lin Yan et al. "A Structural Average of Labeled Merge Trees for Uncertainty Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 832–842. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934242. URL: http://dx.doi.org/10.1109/TVCG.2019.2934242.

[32] Afra Zomorodian and Gunnar Carlsson. "Computing Persistent Homology". In: *Discrete and Computational Geometry* 33 (Feb. 2005), pp. 249–274. DOI: 10.1007/s00454-004-1146-y.

Current address: Department of Mathematics, University of Arizona, Tucson, Arizona 85719
E-mail address: bbollen23@math.arizona.edu