TOPOLOGICAL DISTANCES BETWEEN DIRECTIONAL TRANSFORM REPRESENTATIONS OF GRAPHS

By

Elena Xinyi Wang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science, and Engineering—Doctor of Philosophy

2025

ABSTRACT

Shape analysis is important in fields like computational geometry, biology, and machine learning, where understanding differences in structure and tracking changes over time is useful. Topological Data Analysis (TDA) provides tools to study shape in a way that is resistant to noise and captures both fine and large-scale features. This dissertation focuses on directional transforms, a method that encodes shape by looking at its structure from different directions.

First, we introduce the Labeled Merge Tree Transform (LMTT), a new way to compare embedded graphs using merge trees and directional transforms. We test this method on real-world datasets and show that it works better than existing distance measures in some cases. Next, we develop a kinetic data structure (KDS) for bottleneck distance, which allows us to update shape comparisons efficiently when the data changes over time. We apply this method to the Persistent Homology Transform (PHT) and show that it reduces computation time while keeping accurate results. Finally, we explore a future direction that extends the kinetic data structure to the Wasserstein distance. These contributions improve the use of topology in studying dynamic shapes and open new research possibilities in both theory and practical applications.

Copyright by
ELENA XINYI WANG
2025

To my parents.

ACKNOWLEDGEMENTS

This dissertation is an accumulation of love and support I have received throughout the years; I would not have made it this far without the fortune of being surrounded by the most wonderful people.

First and foremost, I thank my parents. Mom, thank you for being my best friend and supporting my wildest ideas since I was a child. If it were not for your trust in my ability to take care of myself when I asked to move to the U.S. at the age of 14, none of this would have happened. Dad, thank you for being my rock for all this time. Your wisdom has guided me through obstacles and hardships in times when I felt most lost. To my partner, Wesley — thank you for your love, patience, and understanding throughout this journey. Your presence has been a source of comfort and joy, and I am endlessly grateful to have you by my side.

I am immensely grateful to my advisor, Liz Munch, for her invaluable mentorship, guidance, and encouragement. Her support has shaped both my research and my academic path. She believed in me even when I did not, and without her, I would not have grown into the person I am today. She has also introduced me to the most wonderful collaborators — Erin Chambers and Carola Wenk. I also want to express my sincere appreciation to my committee members, Teena Gerhardt and Longxiu Huang, for their insightful feedback and support. I am grateful to Vin de Silva and Sarah Percival, not only for their mathematical guidance but also for the friendships we have built along the way. I would also like to thank Dmitriy Morozov for a wonderful summer of learning from him at Berkeley. I am fortunate to have been part of MunchLab — thank you for the stimulating discussions, collaboration, and sense of community.

I owe a special debt of gratitude to my undergraduate mentors. I would like to thank John Little, Andy Hwang, and Gareth Roberts for showing me how fun mathematical research can be; Dave Damiano, for introducing me to the fascinating world of topological data analysis; and Cristina Ballentine, for her guidance. All of these mentors sparked my passion for mathematics and laid the foundation for my academic career.

To my friends, especially Kathy and Sheldon, thank you for your unwavering support, laughter,

and camaraderie. Your friendship has made this journey all the more meaningful. Who would have thought SAT camp would be the place to meet lifelong friends? A special thanks to SEVENTEEN and MAMAMOO, whose music has been a constant source of motivation and energy during long nights of research and writing. Finally, to my beloved pets, Shosty and Wolfie—thank you for your companionship, warmth, and the much-needed moments of joy and comfort.

This dissertation is a reflection of the collective support, kindness, and inspiration I have received from all of you. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TDA	Topological Data Analysis
KDS	Kinetic Data Structure
PHT	Persistence Homology Transform
ECC	Euler Characteristic Curve
ECT	Euler Characteristic Transform
LMTT	Labeled Merge Tree Transform
OP	Optimal Transport
SAP	Shortest Augmenting Path
ML	Machine Learning
MDS	Multidimensional Scaling
PCA	Principal Component Analysis

CHAPTER 1

INTRODUCTION

Shape plays a fundamental role in many areas of science, from understanding biological structures to analyzing complex geometric data in machine learning [36]. Accurately quantifying and comparing shapes is critical in applications such as image analysis, computer vision, neuroscience, and material science. However, traditional data analysis approaches often struggle to handle high-dimensional, noisy, or evolving data.

Topological Data Analysis (TDA) is a growing field that stands at the intersection of mathematics, computer science, and data analysis, seeking to combine mathematical rigor and computational efficiency to develop novel tools focused on using topological features [37, 32]. It provides a robust mathematical framework for quantifying shape and structure. By focusing on topological patterns, TDA methods are more resilient to noise and deformation, making them useful for real-world applications [6, 75, 13, 33]. One powerful technique in TDA is the directional transform, which encodes shape information by summarizing topological features along different directions [73, 28, 42].

While existing methods such as Persistent Homology Transform (PHT) and Euler Characteristic Transform (ECT) offer valuable insights, challenges remain in defining computable and effective shape comparisons, and efficiently handling time-varying data. Many existing TDA-based shape descriptors rely on static structures, meaning that even small continuous changes in shape require recomputing the distance metrics from scratch. This makes it computationally expensive to track evolving shapes over time. Additionally, while standard topological distances like the bottleneck distance and the Wasserstein distance are useful for comparing shapes, efficient methods for incremental updates in dynamic settings are yet to be developed.

This dissertation addresses these challenges by focusing on directional transforms, merge trees, and kinetic data structures (KDS) to develop more efficient ways of comparing shapes in both static and dynamic settings. Specifically, we introduce new metrics for comparing embedded graphs, which improve the efficiency of tracking topological changes over time, and explore potential applications in cellular biology and machine learning.

The primary goal of this dissertation is to enhance shape analysis using TDA, with a particular emphasis on efficient computation and dynamic tracking. The first objective focuses on the *Labeled Merge Tree Transform (LMTT)* for Shape Comparison, which introduces a novel metric for comparing embedded graphs by integrating directional transforms with merge trees. This method enhances sensitivity to structural differences and demonstrates superior performance over existing distance metrics when tested on real-world datasets. The second objective, *Kinetic Hourglass Data Structure* for dynamic shape tracking, involves the development of an efficient kinetic data structure for the bottleneck distance, allowing for faster updates as shapes evolve over time. This approach is specifically applied to the Persistent Homology Transform (PHT), reducing computational overhead. Finally, this dissertation explores broader applications and future research directions by extending kinetic data structure to compute the Wasserstein distance to further improve dynamic shape tracking.

The remainder of this dissertation is structured as follows. Chapter 2 provides background on persistent homology, merge trees, directional transforms, and kinetic data structures, forming the foundation for later contributions. Chapter 3 introduces the Labeled Merge Tree Transform (LMTT), a novel distance for comparing embedded graphs. In addition to showing and proving its theoretical properties, we validate its effectiveness empirically on real-world datasets and compare it with existing distances. Chapter 4 presents a kinetic data structure for computing the bottleneck distance in a dynamic setting. This approach enables efficient shape tracking in time-varying systems and reduces computational costs. Chapter 5 discusses future directions before concluding.

This dissertation introduces new methods for both static and dynamic shape analysis motivated by tools in TDA. These contributions improve the efficiency and interpretability of TDA-based methods, making them applicable to a wide range of real-world problems. Future work will further refine these techniques and extend their applications in real-life settings such as in cellular biology.

CHAPTER 2

BACKGROUND

In this chapter, we will introduce the ideas that inspired my work in chapters 3 and 4. First, we will discuss persistent homology and its extension, the persistent homology transform, along with its associated comparison metric. Then, we will provide the technical details and theoretical properties of merge trees. Finally, we will introduce the kinetic data structure. Additional background will be introduced in the relevant chapter.

2.1 Persistent Homology

Homology is a fundamental tool in algebraic topology for analyzing topological structures across various dimensions. Specifically, for a given space X, homology assigns a group $H_k(X)$ to each dimension $k = 0, 1, 2, \ldots$, capturing information about the structure at that dimension. For example, dimension 0 focuses on connected components, dimension 1 examines loops, dimension 2 explores voids, and higher dimensions investigate analogous higher-dimensional features.

Persistent homology is a method for measuring changes in homology of a filtration of simplicial complexes. An *n-simplex* is the convex hull of $n+1$ affinely independent points denoted $[v_0, v_1, \ldots, v_n]$. The 0-simplex $[v_0]$ is the vertex v_0, the $1-$simplex $[v_0, v_1]$ is the edge (v_0, v_1), and so on. A *simplicial complex*, $\mathcal{K} \in \mathbb{R}^d$, is a space built from simplices such that:

- the intersection of any two simplices in $\mathcal{K}$ is also a simplex in $\mathcal{K}$;
- all faces of a simplex in $\mathcal{K}$ are also simplices in $\mathcal{K}$.

A *geometric simplicial complex* $K \in \mathbb{R}^n$ is a collection of simplices in $\mathbb{R}^n$ such that:

1. If $\sigma \in K$, then every face of σ is also in K.
2. If $\sigma, \tau \in K$, then $\sigma \cap \tau$ is either empty or a face of both σ and τ.

Given a finite simplicial complex K, a *simplicial k-chain* is a formal linear combination of k-simplices in K. We often assume coefficients are in $\mathbb{Z}_2$, but any field can be used. The set of k-chains forms a vector space $C_k(K)$. Then the boundary map $\partial_k : C_k(K) \to C_{k-1}(K)$ is defined

as follows, which extends linearly:

$$\partial_k([v_0, v_1, \ldots, v_k]) = \sum_{j=0}^{k} (-1)^j [v_0, \ldots, v_{j-1}, v_{j+1}, \ldots, v_k].$$

We define the boundaries to be elements of $B_k(K) = \text{im}(\partial_{k+1})$ and cycles to be elements of $Z_k(K) = \ker(\partial_k)$. Since $\partial_{k+1} \circ \partial_k = 0$, $B_k(K) \subseteq Z_k(K)$. The *k-th homology group* of K is defined to be

$$H_k(K) \coloneqq Z_k(K)/B_k(K).$$

We define a *filtration* to be a nested set of simplicial complexes:

$$\varnothing = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n = K.$$

In this dissertation, we restrict ourselves to *scalar field*, which is a function that assigns a single real-valued measurement to each point in a domain, typically represented as $f : K \to \mathbb{R}$, where K is a geometric or topological space. In the context of filtrations, we consider sublevel set filtrations induced by such functions, defined as the collection of sublevel sets $\{K_\alpha\}_{\alpha \in \mathbb{R}}$, where:

$$K_\alpha = \{\sigma \mid f(\sigma) \leq \alpha\}.$$

We further define:

$$f(\sigma) = \sup_{x \in \sigma} f(x).$$

Additionally, we impose a *monotonicity* condition, meaning that if τ is a face of σ, then $f(\tau) \leq f(\sigma)$. This assumption simplifies the exposition, as it guarantees that all sublevel sets form a valid filtration. This is related to the *lower-star filtration* [37].

Definition 2.1.1 *Given a geometric simplicial complex K and a function $f : V(K) \to \mathbb{R}$ that assigns a scalar value to each vertex, the lower-star filtration of K is the nested sequence of subcomplexes $\{K_\alpha\}_{\alpha \in \mathbb{R}}$ defined by:*

$$K_\alpha = \{\sigma \in K \mid \max_{v \in \sigma} f(v) \leq \alpha\}.$$

That is, each simplex σ is included in K_α if and only if the maximum function value among its vertices is at most α.

The goal is to summarize the change in the topology of the filtration over time. For $a < b$, $\iota : K_a \to K_b$ is an inclusion map of simplicial complex. In general, $\iota^* : C_k(K_a) \to C_k(K_b)$. And it has the following induced inclusion maps: $\iota : B_k(K_a) \to B_k(K_b)$ and $\iota : Z_k(K_a) \to Z_k(K_b)$. This then induces homomorphisms

$$\iota_k^{a \to b} : H_k(K_a) \to H_k(K_b).$$

These statements are standard but non-trivial, see [50] for more detailed proofs.

We follow [37] for the definition of persistent homology groups $H_k(a, b)$:

$$H_k(a, b) \coloneqq Z_k(K_a)/(Z_k(K_a) \cup B_k(K_b)).$$

The cokernel of a function $f : X \to Y$ is $Y/\text{im}(F)$. A homology class $\alpha \in H_k(K_i)$ is said to be *born* at time a, denoted $\text{b}(\alpha)$, if it is in the cokernel of $\iota_k^{a' \to a}$ for any $a' < a$. We say α *dies* at time b, denoted $\text{d}(\alpha)$, if for all $a' < a < b' < b$, we have $\iota_k^{a \to b}(\alpha) \in \text{im}(\iota_k^{a' \to b})$ and $\iota_k^{a \to b'}(\alpha) \notin \text{im}(\iota_k^{a' \to b'})$. The homology class α has *persistence* $\text{d}(\alpha) - \text{b}(\alpha)$. If α never dies, then it is an *essential class* of K, in which case $\text{d}(\alpha) = \infty$.

Let $\mathbb{R}^{2+} \coloneqq \{(a, b) \in (\{\infty\} \cup \mathbb{R}) \times (\mathbb{R} \cup \{\infty\}) : a < b\}$. We denote the k-th persistence diagram corresponding to the filtration K as Dgm_k. It is a multi-set of points in $\mathbb{R}^{2+}$ with countably infinite copies of every point on the diagonal, $\Delta = \{(x, x) \in \mathbb{R}^{2+}\}$. The number of off-diagonal points in $[-\infty, a] \times [b, \infty]$ equals the dimension of $H_k(a, b)$. Each point (a, b) in $\mathbb{R}^{2+}$ corresponds to a k-dimensional homology class that is born at time a and dies at b. Throughout this dissertation, we assume all persistence diagrams have finitely many off-diagonal points.

2.1.1 Bottleneck Distance

The bottleneck distance is a standard measure used to compare two persistence diagrams [37]. To compare two persistence diagrams X and Y, we define a partial matching to be a bijection $\eta : X' \to Y'$ on a subset of the points $X' \subseteq X$ and $Y' \subseteq Y$. We denote this by $\eta : X \rightleftharpoons Y$.

Definition 2.1.2 *The* cost *of a partial matching is the maximum over the L_∞-norms of all pairs of matched points and the distance from the unmatched points to the diagonal:*

$$c(\eta) = \max\left(\{\|x - \eta(x)\|_\infty \mid x \in X'\} \cup \{\tfrac{1}{2}|z_2 - z_1| \mid (z_1, z_2) \in (X \setminus X') \cup (Y \setminus Y')\}\right).$$

The bottleneck distance *is defined as* $d_B(X, Y) = \inf_{\eta: X \rightleftharpoons Y} c(\eta)$.

The bottleneck distance can be formulated more generally as a graph-matching problem. Let X and Y be two arbitrary sets of n points. We construct the bipartite graph with vertex set $V = X \sqcup Y$ and edges between points whose weight $\mathrm{c} : E \to \mathbb{R}_{\geq 0}$ is at most λ:

$$G_\lambda = (X \sqcup Y, \{xy \mid \mathrm{c}(xy) \leq \lambda\}).$$

Definition 2.1.3 *The* optimal bottleneck cost *is the minimum λ such that G_λ has a perfect matching, denoted as* $d_B(G)$.

We now reduce finding the bottleneck distance between persistence diagrams to a problem of finding the bottleneck cost of a bipartite graph. Let X and Y be two persistence diagrams given as finite lists of off-diagonal points. Let $\overline{X}$ (resp. $\overline{Y}$) be the set of orthogonal projections of the points in X (resp. Y). Set $U = X \sqcup \overline{Y}$ and $V = Y \sqcup \overline{X}$. We define the complete bipartite graph $G = (U \sqcup V, U \times V, \mathrm{c})$, where for $u \in U$ and $v \in V$, the weight function c is given by

$$\mathrm{c}(uv) = \begin{cases} \|u - v\|_\infty & \text{if } u \in X \text{ or } v \in Y \\ 0 & \text{if } u \in \overline{X} \text{ and } v \in \overline{Y}. \end{cases}$$

An example of the bipartite graph construction is shown in Figure 2.1. This graph can be used to compute the bottleneck distance of the input diagrams because of the following lemma.

Lemma 2.1.4 (Reduction Lemma [37]) *For the above construction of G,* $d_B(G) = d_B(X, Y)$.

Naively, given a graph G of edge set size n, we can compute the bottleneck distance by sorting the edge weights and performing a binary search for the smallest λ such that G_λ has a perfect matching in $O(n^2 \log n)$ time by doing the following. Sorting the edge weights takes $O(n \log n)$ time, and binary search requires $O(\log n)$ iterations. In each interaction, we check for the existence of a perfect matching in G_λ, which can be done in $O(n^2)$ time. This results in an overall complexity of $O(n \log n + n^2 \log n) = O(n^2 \log n)$. Using the technique for efficient k-th distance selection for a bi-chromatic point set under the L_∞ distance introduced by Chew and Kedem, this can be reduced to $O(n^{1.5} \log n)$ [22]. Therefore, the overall complexity to compute the bottleneck distance of a static pair of persistence diagrams is $O(n^{1.5} \log n)$.

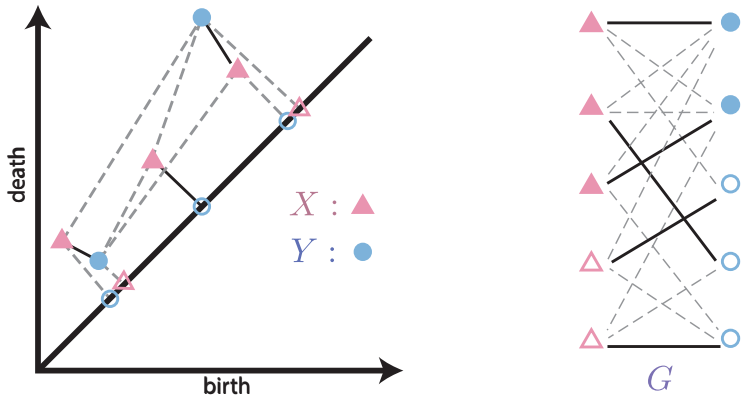


Figure 2.1 Construction of the bipartite graph G based on the persistence diagrams X and Y.

The bottleneck distance between two persistence diagrams is stable in the following sense:

Theorem 2.1.5 (Stability Theorem for Filtrations) *Let K be a simplicial complex and $f, g : K \to \mathbb{R}$ two monotonic functions. For each dimension p, the bottleneck distance between the diagrams $X = \mathrm{Dgm}_p(f)$ and $Y = \mathrm{Dgm}_p(g)$ is bounded from above by the L_∞-distance between the functions,*

$$d_B(X, Y) \leq \|f - g\|_\infty.$$

2.1.2 Computing the Bottleneck Distance

The *bipartite matching problem* is an optimization problem in graph theory that seeks to find a maximum matching in a bipartite graph, where a matching is a set of edges that do not share any vertices. Given a bipartite graph $G = (U \sqcup V, E)$, where U and V are two disjoint sets of vertices and E is the set of edges between them, the goal is to determine the largest possible subset of edges such that each vertex in U and V is incident to at most one edge in the matching.

Computing the bottleneck distance involves solving a bipartite matching problem to minimize the maximum displacement between paired persistence diagram points [24, 38]. Various algorithms exist that trade off exactness, efficiency, and scalability. Table 2.1 summarizes key algorithms for computing the bottleneck distance, highlighting their computational complexity and classification as exact or approximate.

Exact methods for computing the bottleneck distance rely on solving a sequence of maximum bipartite matching problems. The Hopcroft-Karp Algorithm, based on augmenting paths, achieves

Algorithm	Type	Complexity
Hopcroft-Karp Algorithm	Exact	$\mathcal{O}(n^{2.5})$
Push-Relabel Algorithm	Exact	$\mathcal{O}(n^2 \log n)$
Binary Search with Maximum Matching	Exact	$\mathcal{O}(n^3 \log n)$
Auction Algorithm (adapted)	Approximate	$\mathcal{O}(n^2 \log n)$
Greedy Matching	Approximate	$\mathcal{O}(n^2)$

Table 2.1 Comparison of algorithms for computing the bottleneck distance.

an $\mathcal{O}(n^{2.5})$ complexity and is widely used in practice [52]. The Push-Relabel Algorithm, commonly used for maximum flow problems, offers competitive empirical performance with a complexity of approximately $\mathcal{O}(n^2 \log n)$ [43]. Another approach employs a binary search over the bottleneck threshold, solving a sequence of maximum matching problems with an overall complexity of $\mathcal{O}(n^3 \log n)$ [37].

When exact computation is infeasible for large-scale datasets, approximation techniques provide efficient alternatives. A variant of the Auction Algorithm, adapted from optimal transport, offers a fast, auction-based approach with complexity $\mathcal{O}(n^2 \log n)$ [16]. Greedy matching heuristics provide even faster approximations at $\mathcal{O}(n^2)$ complexity but sacrifice solution quality in favor of efficiency [55]. These approximate methods enable the bottleneck distance to be computed on large persistence diagrams while maintaining reasonable accuracy.

2.1.3 Wasserstein Distance

Another measure commonly used to compare persistence diagrams is the Wasserstein distance. The bottleneck distance only takes into account the furthest pair of corresponding points and it is insensitive to the details of the bijection beyond that. To address this shortcoming, we introduce the *degree-q Wasserstein distance* between persistence diagrams where $q \in \mathbb{R}_+$. It takes the minimum over all bijections of the sum of q-th powers of the L_∞-distances between corresponding points. The bottleneck distance is, in fact, the limit of the Wasserstein distance as q goes to infinity.

Definition 2.1.6 *The* Wasserstein distance *between two persistence diagrams X and Y is*

$$W_q(X, Y) = \left[\inf_{\eta : X \rightleftharpoons Y} \sum_{x \in X} \|x - \eta(x)\|_\infty^q \right]^{\frac{1}{q}},$$

where $\eta : X \rightleftharpoons Y$ is a bijection that represents a partial matching between the points in X and Y.

A general stability result like the one for the bottleneck distance is out of reach. However, if we fix our underlying space to be a finite CW Complex, Skraba et al. [69] proved the stability results with respect to the Wasserstein distance. This improves on the previous results on the Lipschitz Wasserstein stability [25]. In order to state the theorem, we briefly introduce a finite *cell complex*, also known as a *CW complex*. It is a topological space X constructed as a union of cells satisfying the following properties:

- Start with a discrete set X^0 (0-cells).
- X^n is obtained from X^{n-1} by attaching n-dimensional disks (called n-cells) via continuous maps from their boundaries: $f_\alpha : S^{n-1} \to X^{n-1}$. Each n-cell is attached by gluing its boundary S^{n-1} to the existing X^{n-1}, forming $X^n = X^{n-1} \bigcup_\alpha e^n_\alpha$, where each e^n_α is an open n-disk.
- Each cell e^n_α is attached to a finite number of lower-dimensional cells. That is, each cell's boundary only intersects a finite number of other cells.

Given such an underlying space, the below stability results hold.

Theorem 2.1.7 (Cellular Wasserstein Stability Theorem [69]) *Let $f, g : K \to \mathbb{R}$ be monotone functions, and $X = \mathrm{Dgm}(f)$, $Y = \mathrm{Dgm}(g)$. Then*

$$W_q(X, Y) \leq \|f - g\|_q.$$

If we fix a homology dimension p, then

$$W_q(\mathrm{Dgm}_p(f), \mathrm{Dgm}_p(g))^q \leq \sum_{\dim(\sigma) \in \{k, k+1\}} |f(\sigma) - g(\sigma)|^q.$$

2.1.4 Computing the Wasserstein Distance

Computing the Wasserstein distance involves solving an optimal transport (OT) problem [67, 63, 76], which assigns probability mass from one distribution to another while minimizing transportation cost. Several algorithms exist that balance exactness, efficiency, and scalability. Table 2.2 summarizes key algorithms for computing the Wasserstein distance, highlighting their computational complexity and classification as exact or approximate.

Algorithm	Type	Complexity
Hungarian Algorithm	Exact	$\mathcal{O}(n^3)$
Network Simplex	Exact	$\mathcal{O}(n^2 \log n)$
Shortest Augmenting Path (Bipartite Graph)	Exact	$\mathcal{O}(n^3)$
Sinkhorn-Knopp	Approximate	$\mathcal{O}(n^2)$
Auction Algorithm	Approximate	$\mathcal{O}(n^2 \log n)$
Gaussian Approximation	Approximate	$\mathcal{O}(d^3)$ (closed-form)

Table 2.2 Comparison of algorithms for computing Wasserstein distance.

Exact methods for computing the Wasserstein distance solve the optimal transport (OT) problem as a linear program. The Hungarian algorithm is the classic approach with cubic complexity, suitable for small-scale problems [58]. The Network Simplex Method is often used for large-scale OT problems due to its efficiency in practice. However, its worst-case complexity can be as high as $O(2^n)$ in general cases [57], making it theoretically inefficient. For minimum-cost flow problems, which include OT as a special case, the worst-case complexity is typically $O(mn \log n)$ or $O(m \log C)$, where n is the number of nodes, m is the number of edges, and C is the largest cost coefficient [15]. Despite these worst-case bounds, the method remains widely used due to its empirical speedups in practical instances. Another widely used approach is the Shortest Augmenting Path Algorithm, which formulates the OT problem as a bipartite graph and iteratively finds the shortest paths in a residual graph to improve computational efficiency [5].

When exact computation is infeasible, approximation techniques offer scalable alternatives. The Sinkhorn-Knopp Algorithm introduces entropy regularization to the OT problem, enabling fast iterative updates and making it popular in deep learning applications [68]. The Auction Algorithm is a greedy, auction-based method designed for large-scale matching problems [14]. The Sliced Wasserstein Distance simplifies computation by projecting distributions onto one-dimensional subspaces, where the Wasserstein distance is computed efficiently before aggregating results [64]. Additionally, for Gaussian-distributed data, the Gaussian Approximation provides a closed-form solution, avoiding costly OT solvers [35]. These approximations balance speed and precision, making the Wasserstein distance practical for high-dimensional applications.

2.2 Directional Transform

The motivating idea of the directional transform is that given an object, such as a surface in $\mathbb{R}^3$ or a domain in $\mathbb{R}^2$, how can we faithfully represent it using topological descriptors? Computing topological summaries, such as persistent homology, along different directions produces distinct diagrams representing the same object. In this section, we introduce various types of topological summaries used in the context of directional transforms and discuss their theoretical properties.

2.2.1 Persistent Homology Transform

Given a finite geometric simplicial complex K as a subset of $\mathbb{R}^d$, for any unit vector $v \in \mathbb{S}^{d-1}$, we can define a *filtration* $K(v)$ of K parameterized by a height r where $K(v)_r = \{x \in K : \langle x, v\rangle \leq r\}$ is the subcomplex of K containing all the simplices below height r in the direction v. This is closely related to the lower-star filtration (Definition 2.1.1). The filtration $K(v)_r$ and $\{\sigma \in K : \langle x, v\rangle \leq r \text{ for all vertices } x \in \sigma\}$ have the same homology groups since they are homotopy equivalent. In this definition, partial simplices in $K(v)_r$ can be homotoped down to the subcomplex in the lower-star filtration h_ω. Homology is invariant under homotopy equivalences [50].

The k-dimensional persistence diagram $\mathrm{Dgm}_k(K, v)$ summarizes how the topology of the filtration $K(v)$ changes over r. We define $\mathcal{D}$ to be the space of all persistence diagrams with finitely many off-diagonal points in each diagram.

Definition 2.2.1 *A function $f : X \to Y$, where X, Y are metric spaces with metrics d_X and d_Y, is said to be* Lipschitz continuous *if there exists a constant $L \geq 0$ such that for all $x_1, x_2 \in X$,*

$$d_Y(f(x_1), f(x_2)) \leq L \cdot d_X(x_1, x_2).$$

The smallest such constant L is called the Lipschitz constant of f. If f satisfies this condition with $L < 1$, it is called a contraction.

Since $\mathcal{D}$ holds bottleneck distance stability, the following function is Lipschitz (hence continuous):

$$v \mapsto \mathrm{Dgm}_n(K, v).$$

Definition 2.2.2 *The* Persistent Homology Transform *of $K \subseteq \mathbb{R}^d$ is the function*

$$PHT(K) : \mathbb{S}^{d-1} \to \mathcal{D}^d$$
$$v \mapsto \left(\mathrm{Dgm}_0(K, v), \mathrm{Dgm}_1(K, v), \ldots, \mathrm{Dgm}_{d-1}(K, v)\right).$$

Theorem 2.2.3 ([73]) *The persistent homology transform is injective when the domain is K_d for $d = 2, 3$.*

Informally, we say a statistic is *sufficient* if no other statistic that can be calculated from the same data provides any additional information. For more formal definitions with respect to measure theory, see [49]. The main result of [73] is that since the PHT is injective for K_d for $d = 2, 3$, it is a sufficient statistic.

2.2.2 Euler Characteristic Transform

The Euler Characteristic Transform has a similar construction as the PHT, except that instead of persistent homology, the topology of the simplicial complex is captured by the Euler characteristic.

Definition 2.2.4 *The* Euler characteristic *for a simplicial complex K is an alternating sum of the number of simplices in each dimension,*

$$\chi(K) = \#(\textit{vertices}) - \#(\textit{edges}) + \#(\textit{faces}) - \cdots.$$

Standard yet surprisingly, following the above definition, the case for orientable examples is a corollary of Poincaré duality.

Definition 2.2.5 *The* Euler characteristic *of a simplicial complex K is an alternating sum of the number of simplices in each dimension:*

$$\chi(K) = \#(\textit{vertices}) - \#(\textit{edges}) + \#(\textit{faces}) - \cdots.$$

More formally, letting f_k denote the number of k-dimensional simplices in K, the Euler characteristic is given by:

$$\chi(K) = \sum_{k=0}^{\dim K} (-1)^k f_k.$$

The Euler characteristic is a fundamental topological invariant that provides insight into the combinatorial structure of a simplicial complex. However, while it gives a global measure of

the space, it does not directly capture more refined topological features, such as the number of connected components, loops, or voids. These are better described using homology theory and its associated Betti numbers.

Definition 2.2.6 *The* Betti numbers β_k *of a simplicial complex* K *quantify the number of independent* k*-dimensional holes in* K*. Formally, they are defined as the ranks of the homology groups:*

$$\beta_k = \textit{rank } H_k(K).$$

Here, $H_k(K)$ *represents the* k*-th homology group, which encodes information about cycles and boundaries at dimension* k*. For example,* β_0 *counts the number of connected components,* β_1 *counts the number of independent loops, and* β_2 *counts the number of voids.*

The Betti numbers provide a finer classification of a topological space than the Euler characteristic alone. However, there exists a strong relationship between these two concepts, which is encapsulated in the following theorem.

Theorem 2.2.7 *The Euler characteristic of a simplicial complex* K *can also be expressed in terms of its Betti numbers and torsion coefficients:*

$$\chi(K) = \sum_{k=0}^{\dim K} (-1)^k \beta_k.$$

If the homology groups are free abelian (i.e., there is no torsion), this formula holds exactly.

We can construct a variation of the PHT using the Euler characteristic.

Recall the height filtration $K(v)_r = \{\sigma \in K : \langle x, v\rangle \leq r \text{ for all vertices } x \in \sigma\}$. The Euler characteristic curve (ECC) $\chi(K, v) : \mathbb{R} \to \mathbb{Z}$ is a function giving the Euler characteristic for each sublevel set at values r, $\chi(K, v)(r) = \chi(K(v)_r)$.

Definition 2.2.8 *The* Euler characteristic transform (ECT) *of a geometric simplicial complex* $K \subseteq \mathbb{R}^d$ *is*

$$\begin{aligned} \textit{ECT}(K) : \mathbb{S}^{d-1} &\to \mathbb{Z}^{\mathbb{R}} \\ v &\mapsto (\chi(K, v)) \,. \end{aligned}$$

The ECT is also both injective and a sufficient statistic for K_d for $d = 2, 3$. This construction has demonstrated its utility in many settings, such as analyzing barley seed morphology [6], see [7] for a more detailed survey on the ECT.

2.3 Merge Tree

Intuitively, a *merge tree* is a 1-dimensional topological invariant that tracks when components appear and subsequently merge in sublevel sets of a scalar function. It is closely related to persistent homology, as the branches of merge trees track the same 0-dimensional features that appear in the persistence diagram.

Our input data will be a topological graph with an $\mathbb{R}$-valued function. First, we treat a graph $G = (V, E)$ as a 1-dimensional stratified space, so that the edges are homeomorphic to copies of the interval $[0, 1]$. We will additionally be given a continuous map $f : G \to \mathbb{R}$ where we will assume that the function is monotone on the vertices. This additional assumption means we can combinatorially represent the function by specifying the function on the vertex set $f : V(G) \to \mathbb{R}$ and then extend linearly to the edges. We also note that whenever needed, we can subdivide an edge $e = (u, w)$ with a vertex v with $f(u) \leq f(v) \leq f(w)$ to get an equivalent representation of the structure.

While it can be defined for more general inputs, our next step will be to understand the merge tree of a given graph with a function $f : G \to \mathbb{R}$. A merge tree is defined as follows.

Definition 2.3.1 *A* merge tree *is a pair (T, f) with $f : T \to \mathbb{R} \cup \{\infty\}$ where T is a rooted, finite, acyclic, connected graph (i.e. a finite tree in the graph-theoretic sense with a distinguished root vertex, r), together with a real-valued function $f : V \to \mathbb{R} \cup \{\infty\}$. This function has the property that every non-root vertex has exactly one neighbor with a higher function value, and $f(v) = \infty$ if and only if v is the root r.*

We write $V(T)$ for all non-root vertices of the tree T. We call the non-root vertices with degree 1 *leaves* and denote the set of leaves $L(T)$. Every vertex has an up- and a down-degree, given by the number of neighbors with higher (respectively lower) function values. A vertex w is regular if its up- and down-degree are both equal to 1. In this case, denoting the neighbors of w as u and

v, we can remove the vertex w and add in the edge u, v to get an equivalent representation of the merge tree in a topological sense. So we say two merge trees are *combinatorially equivalent* if T_1 and T_2 are isomorphic as combinatorial trees up to this subdivision equivalence, ignoring the function value information. We use $\mathcal{MT}$ to denote the space of all merge trees where two merge trees are considered the same if they are both combinatorial equivalent and have the same function value.

The merge tree of a given graph with a function $f : G \to \mathbb{R}$ is defined as follows, following [60]. For graph (G, f), define the epigraph of the function as $\text{epi}(f) = \{(x, y) \in G \times \mathbb{R} \mid y \geq f(x)\}$. Define a function $\bar{f} : \text{epi}(f) \to \mathbb{R}$ by $(x, y) \mapsto y$. Say that two points $(x, y) \sim_{\bar{f}} (x', y')$ are equivalent if and only if $y = y'$ (i.e. $\bar{f}(x, y) = \bar{f}(x', y')$) and they are in the same connected component of the level set $\bar{f}^{-1}(a)$. Then the merge tree of (G, f), denoted $T(G, f)$, is the quotient space $\text{epi}(f)/\sim_{\bar{f}}$. Note that the points of $T(G, f)$ inherit the function f since we can set $f([x]) = f(x)$ which is well defined by construction of $\sim_f$. See Figure 2.2 for an example.

Note that this definition is subtly different from the one used in the visualization literature (see e.g. [79]). In that setting, efine an equivalence class on the points of G by setting $x \sim y$ if and only if $f(x) = f(y)$, and x and y are in the same connected component of $f^{-1}(-\infty, f(x))$. Then the alternative merge tree is the quotient space $X/\sim$. While the result is nearly identical, the difference is the root vertex will have a function value equal to the global maximum of f; in particular in the case of compact X this results in a finite function value for the root which goes against Definition 2.3.1. However, this causes issues in the case of some distance computations, specifically the interleaving distance [60, 61, 40], where an infinite tail (i.e. $f(r) = \infty$) is needed for technical reasons. In this paper, we will use the infinite tail definition due to our focus on the use of distances. However, we note that we could use the first definition with some modification as in [60]; or alternatively, we could attach a root vertex r to the global maximum vertex in the alternative merge tree and set $f(r) = \infty$ to resolve the issue.

We will next introduce labeled merge trees [61].

Definition 2.3.2 *Fix an integer $n \in \mathbb{Z}_{\geq 0}$ let $[n] = \{1, \ldots, n\}$. An* n-labeled merge tree *is a merge*

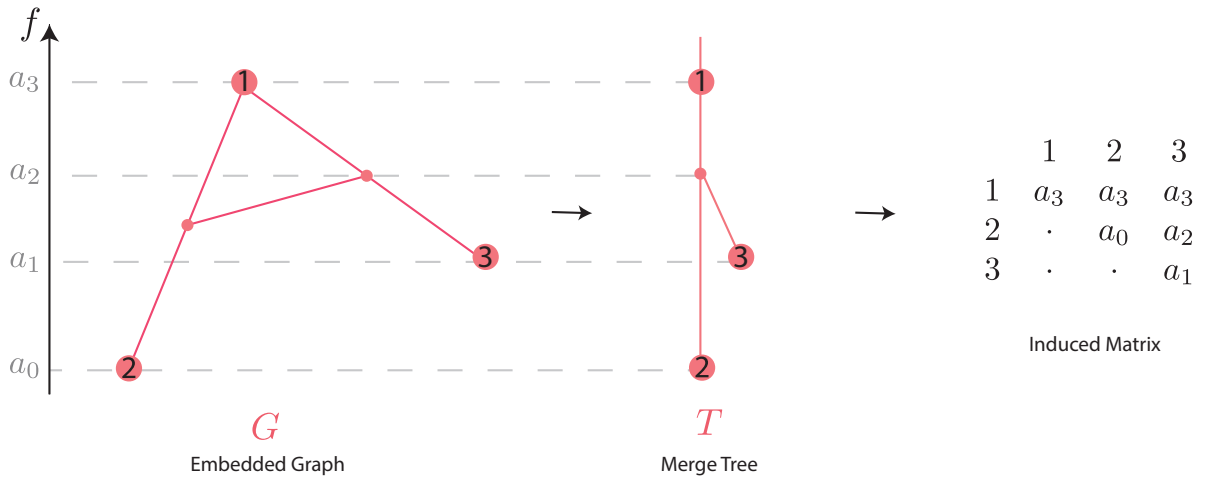


Figure 2.2 Left: A graph embedded in $\mathbb{R}^2$, along with a height function $f : V(G) \to \mathbb{R}$. Middle: The merge tree of (G, f), with labels coming from a labeled vertex of G. Right: The matrix $\mathcal{M}(T, f, \pi)$, as defined in Section 2.3.1.

tree (T, f) *together with a map to the non-root vertices* $\pi : [n] \to V(T)$ *that is surjective on the set of leaves. We denote a labeled merge tree with the triple* (T, f, π)*, and the space of all* n*-labeled merge trees by* $\mathcal{LMT}_n$*.*

Note that this requirement has a label on all leaves, but might involve having labels on interior vertices, including those resulting from the subdivision of an edge. See Fig. 2.2 for an example.

Finally, assume we are given a labeling (no longer requiring surjectivity in any form) on an input graph $\pi : [n] \to V(G)$, where we recall that we can subdivide an edge as needed to have this labeling be defined on the vertex set. Then we can use this to induce a labeling $\bar{\pi} : [n] \to V(T)$ on the merge tree $T = T(G, f)$ by setting $\bar{\pi}(i)$ to be the equivalence class of $\pi(i)$ under the quotient space construction $[\pi(i) \in T]$. Note that this induced labeling does not have the required surjectivity on the leaves of T without additional assumptions on π; later in Chapter 3, we will provide assumptions to ensure this is satisfied.

2.3.1 Labeled Merge Tree Distance

Finding metrics to compare different merge trees has recently been an active focus of study, given their wide use as a simple topological summary of scalar field data. In this dissertation, we focus on the interleaving distance, which has been broadly used to compare persistence diagrams,

with a range of theoretical and practical guarantees in various settings. Computing the interleaving distance on merge trees is NP-hard [2, 17], although there has been work done on heuristic methods for certain types of data [80]. The reason we are interested in labeled merge tree data here is that we can utilize the L_∞-cophenetic metric [61, 40], also known as the labeled interleaving distance, since computation becomes tractable in this setting.

In order to state the interleaving distance, we define a natural conversion of a merge tree into a matrix as follows. Let $\text{LCA}(v_1, v_2) \in T$ denote the *lowest common ancestor* of vertices v_1 and $v_2 \in V(T)$, that is, the vertex with the lowest function value $f(v)$ that has both v_1 and v_2 as descendants, where a vertex u is a *descendant* of a vertex x if x lies on the path from u to the root of T. We define each node to be a descendant of itself. The *induced matrix of the labeled merge tree* (T, f, π) is $\mathcal{M}(T, f, \pi) \in \mathbb{R}^{n\times n}$, where $\mathcal{M}(T, f, \pi)_{ij} = f(\text{LCA}(\pi(i), \pi(j)))$. See Fig. 2.2 for an example. We can now compute the distance between two labeled merge trees with the same set of n labels by taking the L_∞ norm of the difference between the two induced matrices; see Fig. 2.3. More formally, given a matrix $A = (a_{ij}) \in \mathbb{R}^{n_1\times n_2}$, recall that the L_∞ norm of A is $\|A\|_\infty = \max_{i,j} |a_{ij}|$. Then the distance is defined as follows.

Definition 2.3.3 *For a fixed $n > 0$ and two labeled merge trees in $\mathcal{LMT}_n$, the* labeled interleaving distance *is*

$$d((T_1, f_1, \pi_1), (T_2, f_2, \pi_2)) := \|\mathcal{M}(T_1, f_1, \pi_1) - \mathcal{M}(T_2, f_2, \pi_2)\|_\infty.$$

It is worth noting that this yields a metric on the space of merge trees with a common label set [61, 40], and further is an interleaving distance [31] for a particular choice of category.

2.4 Convexity Analysis

Our work relies heavily on convexity, a fundamental concept in geometry and optimization. A set is said to be *convex* if, for any two points within the set, the entire line segment connecting them is also contained within the set. Many properties of convex sets, such as their ability to be enclosed by a minimal convex region, play a key role in our analysis. One essential notion is the convex hull, which captures the smallest convex set containing a given set of points.

Definition 2.4.1 *The convex hull of a set of points $S \subseteq \mathbb{R}^d$, denoted as $\text{conv}(S)$, is the smallest*

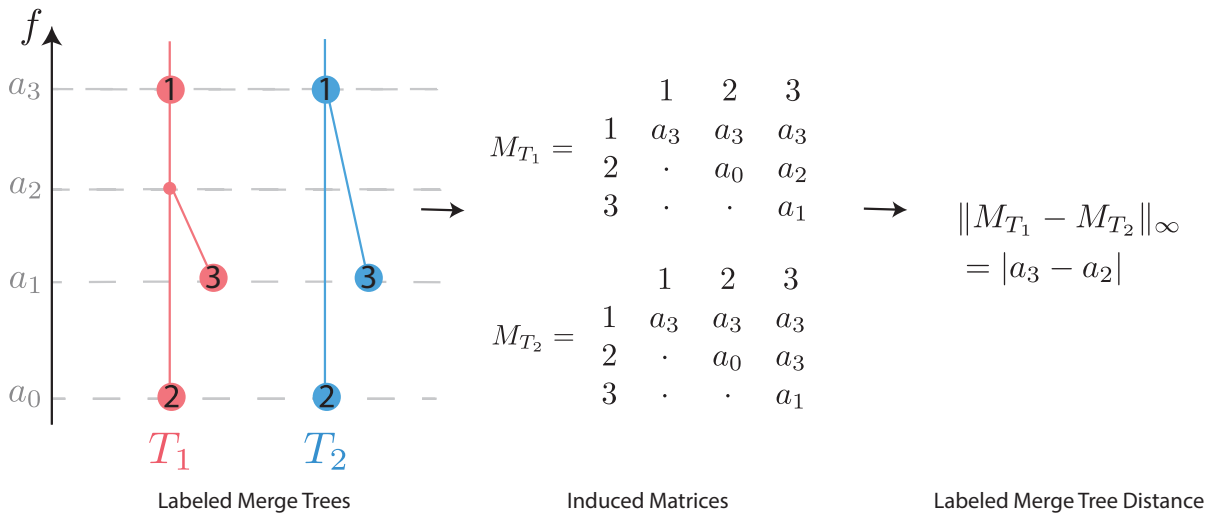


Figure 2.3 Two merge trees with a common set of labels, along with their induced matrices and the labeled merge tree distance between them.

convex set that contains all points of S. Formally, it is given by

$$\text{conv}(S) = \left\{ \sum_{i=1}^{k} \lambda_i p_i \mid p_i \in S, \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1, k \in \mathbb{N} \right\}.$$

A key property of convex sets is that disjoint convex sets can be separated by hyperplanes. The following theorem guarantees the existence of such a separating hyperplane under certain conditions.

Theorem 2.4.2 (Hyperplane Separation Theorem [66]) *Let A, B be two non-empty, disjoint, convex subsets of $\mathbb{R}^d$.*

- *(**Strict Separation**) If both A and B are open, and at least one of them is compact, then there exists a hyperplane that strictly separates them, i.e., there exists a nonzero vector $\mathbf{w} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$ such that*

$$\mathbf{w} \cdot \mathbf{x} < b \quad \forall \mathbf{x} \in A, \quad \textit{and} \quad \mathbf{w} \cdot \mathbf{y} > b \quad \forall \mathbf{y} \in B.$$

- *(**Weak Separation**) If A and B are both compact, or if they are merely convex and disjoint, then there exists a hyperplane that weakly separates them, i.e., there exists $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that*

$$\mathbf{w} \cdot \mathbf{x} \leq b \quad \forall \mathbf{x} \in A, \quad \textit{and} \quad \mathbf{w} \cdot \mathbf{y} \geq b \quad \forall \mathbf{y} \in B.$$

2.5 Distance between Graphs

There are several existing approaches to quantify a comparison between embedded graphs, which we will define shortly. Buchin et al. surveyed the most commonly used measures between two embedded graphs in [20], which we briefly discuss below. Each distance captures different elements of graphs; we wish to understand these trade-offs in order to develop a new distance.

We evaluate the metric properties of each distance by checking the following.

Definition 2.5.1 *Let $\mathbb{X}$ be a set with a given function $\delta : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, where $\mathbb{R}_{\geq 0}$ is the non-negative extended real line. We define the following properties:*

1. *Finiteness: for all $x, y \in \mathbb{X}$, $\delta(x, y) < \infty$.*
2. *Identity: for all $x \in \mathbb{R}$, $\delta(x, x) = 0$.*
3. *Symmetry: for all $x, y \in \mathbb{X}$, $d(x, y) = \delta(y, x)$.*
4. *Separability: for all $x, y \in \mathbb{X}$, $\delta(x, y) = 0$ implies $x = y$.*
5. *Subadditivity (Triangle Inequality): for all $x, y, z \in \mathbb{X}$, $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$.*

If δ satisfies all five properties, then it is a *metric*; it is a *semi-metric* if it satisfies all but subadditivity. There are less strict notions of dissimilarity, such as the following: d is a *pseudo-metric* if it satisfies all but separability; it is an *extended metric* if it satisfies all but finiteness; it is a *semi-metric* if it satisfies all but subadditivity, and it is a *quasi-metric* if it satisfies all but symmetry.

Let (M, δ) be a metric space and $G = (V, E)$ be an abstract graph, which can be viewed as a stratified one-dimensional simplicial complex. We require our graphs to be non-empty, containing at least one vertex. For $x \in G$, it can either be a vertex or a point interior to an edge. The open neighborhood of x is denoted as n_x. If a continuous map $\phi : G \to M$ is homeomorphic onto its image for all small enough n_x of $x \in G$, then we call ϕ an *immersion* of G into M. If an immersion of G is homeomorphic onto the image $\phi(G)$, then it is an *embedding*. Intuively, immersions allow edge crossings, and embeddings do not. We denote the collection of finite immersed graphs in M as $\mathcal{G}_M$ and embedded graphs $\widetilde{\mathcal{G}}_M$.

2.5.1 Hausdorff Distance

Let (M, δ) be a metric space and let 2^M denote the collection of all subsets of M. Then, the *directed Hausdorff distance* $\overrightarrow{\delta}_H \colon 2^M \times 2^M \to \mathbb{R}_{\geq 0}$ in (M, δ) is defined by

$$\overrightarrow{\delta}_H(A, B) \coloneqq \sup_{a \in A} \delta(a, B) = \sup_{a \in A} \inf_{b \in B} \delta(a, b),$$

where the supremum over an empty set is defined to be zero and the infimum to be ∞. We define the *Hausdorff distance* $\delta_H : 2^M \times 2^M \to \mathbb{R}_{\geq 0}$ to be the following:

$$\delta_H(A, B) \coloneqq \max\{ \overrightarrow{\delta_H}(A, B), \overrightarrow{\delta_H}(B, A)\}.$$

The above distances are defined in the general setting, but now we consider $\overrightarrow{\delta_H}$ restricted to $\mathcal{G}_M$. The Hausdorff distance can be extended to the set of immersed graphs $\mathcal{G}_M$ in M. The *directed Hausdorff distance between immersed graphs* $\overrightarrow{d_H} : \mathcal{G}_M \times \mathcal{G}_M \to \overline{\mathbb{R}}_{\geq 0}$ for $(G_1, \phi_1), (G_2, \phi_2) \in \mathcal{G}_M$ is defined by

$$\overrightarrow{d_H}(G_1, G_2) = \overrightarrow{\delta_H}(\phi_1(G_1), \phi_2(G_2)),$$

and the *(undirected) Hausdorff distance* $d_H : \mathcal{G}_M \times \mathcal{G}_M \to \overline{\mathbb{R}}_{\geq 0}$ is

$$\begin{aligned} d_H(G_1, G_2) &= \delta_H(\phi_1(G_1), \phi_2(G_2)) \\ &= \max\{\overrightarrow{d_H}(G_1, G_2), \overrightarrow{d_H}(G_2, G_1)\}. \end{aligned}$$

The various Hausdorff distances satisfy the below metric properties.

Theorem 2.5.2 *(Metric Properties of Hausdorff Distances Between Immersed Graphs) Let (M, δ) be a metric space. The following statements hold:*

1. *The directed Hausdorff distance d_H on $\mathcal{G}_M$ satisfies finiteness, identity, and subadditivity, but not symmetry and separability (i.e., it is a directed pseudo-metric).*
2. *The Hausdorff distance d_H on $\mathcal{G}_M$ satisfies finiteness, identity, symmetry, and subadditivity, but does not satisfy separability (i.e., the Hausdorff distance on immersed graphs is a pseudo-metric).*
3. *The Hausdorff distance d_H restricted to embedded graphs in $\widetilde{\mathcal{G}}_M$ is a metric.*

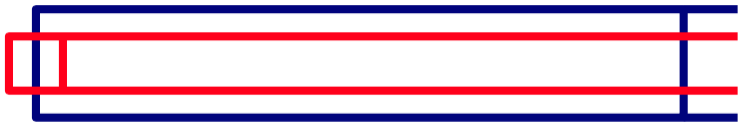


Figure 2.4 An example of graphs with large Fréchet distance but small Hausdorff distance [20].

2.5.2 Fréchet Distance

For two polynomial curves $\gamma_1 : [1, n] \to \mathbb{R}^2$ and $\gamma_2 : [1, m] \to \mathbb{R}^2$, the Fréchet distance, δ_F, is the infimum of the maximum distance between the points on γ_1 and γ_2 achieved by any reparameterization of the unit interval $I = [0, 1]$. Mathematically, it can be written as:

$$\delta_F(\gamma_1, \gamma_2) = \inf_{\varphi} \max_{t \in [0,1]} \delta(\gamma_1(t), \gamma_2(\varphi(t))),$$

where φ ranges over all reparametrizations of I. The Fréchet distance is also known as the dog-walking distance. Intuitively, this represents the shortest leash length needed for a woman and her dog to walk their respective curves from start to end.

The *Fréchet distance between immersed graphs* can be expressed as $d_F : \mathcal{G}_M \times \mathcal{G}_M \to \overline{\mathbb{R}}_{\geq 0}$, where the setting is restricted to the Fréchet distance between paths and orientation-preserving homeomorphisms are avoided. For $(G_1, \phi_1), (G_2, \phi_2) \in \mathcal{G}_M$, the Fréchet distance is defined as

$$d_F(G_1, G_2) = \min_{\alpha} \max_{e \in E_1} \delta_F(\phi_1(e), \phi_2(\alpha(e)))$$

with the condition that G_1 and G_2 are homeomorphic; otherwise, the distance is ∞. Here, α ranges over all edge mappings corresponding to the isomorphisms of G_1 and G_2.

As graphs are compared based on an isomorphism, it is possible for the Fréchet distance to be significantly greater than the Hausdorff distance. See Figure 2.4. In particular, the Hausdorff distance is relatively small because every point on the red graph (inner rectangle) has a nearby corresponding point on the blue graph (outer rectangle). The maximum distance between the closest points is not large. On the other hand, the Fréchet distance is larger because when moving continuously along the red and blue graphs, there are sections such as the right-most vertical edge, where the corresponding points must maintain a large separation. This example illustrates the different information that specific distance metrics can capture. While the Hausdorff distance

captures only the worst-case closest point distance, the Fréchet distance is much more sensitive to the overall shape and traversal.

2.5.3 PHT Distance

Given two persistence diagrams, their distance can be measured by the bottleneck distance or the Wasserstein distance. Theoretical properties of the PHT (Definition 2.2.2) ensure that the points move continuously across the whole family of persistence diagrams. Therefore, the distance between two PHTs can be measured by the distance of choice at all angles and integrated over the whole domain.

We first define the distance in terms of the bottleneck distance.

Definition 2.5.3 *For simplicial complexes K_1 and K_2 the integrated bottleneck distance between K_1 and K_2 is defined as*

$$d_B^{\mathrm{PHT}}(K_1, K_2) = \int_0^{2\pi} d_B\left(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})\right) d\omega.$$

The stability of PHT with respect to the bottleneck distance given next follows directly from Theorem 2.1.5.

Theorem 2.5.4 *Assume that we have two geometric simplicial complexes K_1 and K_2 with the same underlying abstract complex. Let $f_1, f_2 : K \to \mathbb{R}^d$ be different geometric vertex embeddings of K with vertex set V. The distance between the diagrams is bounded by*

$$d_B^{\mathrm{PHT}}(f(K), g(K)) \leq \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty$$

for all $\omega \in \mathbb{S}^1$. Integrating this inequality immediately gives the following result for the integrated bottleneck distance.

We can also define a more general p-PHT distance.

Definition 2.5.5 *For $p \in [0, \infty)$, and simplicial comlexes $K_1, K_2 \subseteq \mathbb{R}^d$, we can define a p-PHT distance between K_1 and K_2 by*

$$d_p^{PHT}(K_1, K_2) = \left(\int_{S^{d-1}} W_p(\boldsymbol{Dgm}(h_\omega^{K_1}), \boldsymbol{Dgm}(h_\omega^{K_2}))^p d\omega\right)^{\frac{1}{p}}.$$

Skraba et al. proved that the PHTs of different vertex embeddings of the same arbitrary simplicial complex are stable with respect to the Wasserstein distance.

Theorem 2.5.6 (Theorem 2.1.7 [69]) *Fox a simplicial complex K with vertex set V. Let*

$$C_{p,d} = 2\mu_{d-2} \int_0^{\frac{\pi}{2}} \cos^p(\theta) \sin^{d-2}(\theta) d\theta$$

where μ_{d-2} is the area of the unit sphere S^{d-2} and

$$C_K = \max_{v \in K} |\{\sigma \in K | v \in \bar{\sigma}\}|.$$

Let $f_1, f_2 : K \to \mathbb{R}^d$ be different geometric vertex embeddings of K. Then

$$d_p^{PHT}(f(K), g(K)) \leq \left(C_K C_{p,d} \sum_{v \in V} \|f(v) - g(v)\|_2^p \right)^{\frac{1}{p}}.$$

2.5.4 ECT Distance

Similar to the PHT distance, ECT distance is also stable with respect to the Wasserstein distance. We first define the L_1 distance between Euler Characteristic Curves. Let $\mathrm{ECC}(K, r)$ denote the Euler Characteristics Curve of the simplicial complex K, where $r \in \mathbb{R}$ is each filtration level.

Definition 2.5.7 *For simplicial complexes K_1 and K_2, the L_1 distance between their Euler Characteristic Curves is given by*

$$\| \mathrm{ECC}(K_1, r) - \mathrm{ECC}(K_2, r)\|_1 = \int_{\mathbb{R}} | \mathrm{ECC}(K_1, r) - \mathrm{ECC}(K_2, r)| dr$$

Dłotko & Gurnari showed that this is stable with respect to the Wasserstein distance in [34]

Theorem 2.5.8 ([34] Proposition 2) *Given geometric simplicial complexes K_1, K_2, with filtrations $\mathcal{F}$ and $\mathcal{G}$ respectively, then the L_1 distance of the Euler Characteristic Curves between them is bounded by the Wasserstein distance between the filtrations.*

$$\| \mathrm{ECC}_{\mathcal{F}}(K_1) - \mathrm{ECC}_{\mathcal{G}}(K_2)\|_1 \leq 2W_{1,\infty}(\mathrm{Dgm}(\mathcal{F}(K_1)), \mathrm{Dgm}(\mathcal{G}(K_2))$$

Assume we have two different geometric vertex embeddings of a simplicial complex K, $f, g : K \to \mathbb{R}^d$, we can restrict the above result further:

$$\| \mathrm{ECC}_{\omega}(f(K)) - \mathrm{ECC}_{\omega}(g(K))\|_1 \leq 2W_{1,\infty}(\mathrm{Dgm}(h_{\omega}^{f(K)}), \mathrm{Dgm}(h_{\omega}^{g(K)})).$$

We can now define the ECT distance.

Definition 2.5.9 *Given two embeddings of the same abstract simplicial complex K, the distance between the Euler Characteristic Transform of each is defined as*

$$d_{\mathrm{ECT}}(ECT(f(K)), ECT(g(K))) := \int_{\mathbb{S}^{d-1}} \| \mathrm{ECC}_\omega(f(K)) - \mathrm{ECC}_\omega(g(K))\|_1 \, d\omega.$$

George et al. showed that this distance is stable with respect to the Wasserstein distance inspired by the proof of Theorem 2.5.6 [41].

Theorem 2.5.10 *Given an abstract simplicial complex K with vertex set V. Let*

$$C_d = 2\mu_{d-2} \int_0^{\frac{\pi}{2}} \cos(\theta) \sin^{d-2}(\theta) d\theta$$

where μ_{d-2} is the area of the unit sphere S^{d-2} and

$$C_K = \max_{v \in V(K)} |\{\sigma \in K | v \in \bar{\sigma}\}|.$$

Let $f, fg : K \to \mathbb{R}^d$ be different geometric vertex embeddings of K. Then

$$d_{\mathrm{ECT}}\left(ECT(f(K)), ECT(g(K))\right) \leq 2C_K C_d \sum_{v \in V(K)} ||f(v) - g(v)||_2.$$

2.6 Shape Comparison and Dimension Reduction

Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) are two classical methods for dimensionality reduction and data visualization. Both techniques aim to represent high-dimensional data in a lower-dimensional space while preserving essential structure.

2.6.1 Principal Component Analysis

PCA identifies an orthogonal basis in which the variance of the data is maximized. Given a centered dataset $X \in \mathbb{R}^{n \times d}$, PCA finds the principal components by solving the eigenvalue problem for the covariance matrix:

$$C = \frac{1}{n} X^T X.$$

The eigenvectors corresponding to the largest eigenvalues define a new coordinate system that best captures the variance. To obtain a lower-dimensional representation, the dataset X is projected

onto the subspace spanned by the top k eigenvectors. Formally, let $V_k \in \mathbb{R}^{d\times k}$ be the matrix whose columns are the top k eigenvectors of C. The projection of X is then given by

$$X_k = XV_k,$$

where $X_k \in \mathbb{R}^{n\times k}$ contains the coordinates of the original data points in the reduced k-dimensional space. This transformation preserves as much variance as possible while reducing dimensionality, making it useful for data compression, visualization, and noise reduction.

2.6.2 Multidimensional Scaling

MDS seeks a configuration of points in a lower-dimensional space such that pairwise distances approximate a given dissimilarity matrix [18, 59]. In *classical* MDS, the goal is to find $Y \in \mathbb{R}^{n\times k}$ such that the Euclidean distances between points in Y best match the input dissimilarities. Given a squared distance matrix D, classical MDS is computed by double-centering the matrix

$$B = -\frac{1}{2}HDH, \quad H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T,$$

where H is the centering matrix, and $\mathbf{1}$ denotes the n-dimensional column vector of all ones.

The embedding is obtained from the top k eigenvectors of B, similar to PCA. Specifically, let $\lambda_1, \lambda_2, \ldots, \lambda_k$ be the largest k eigenvalues of B, and let $v_1, v_2, \ldots, v_k$ be the corresponding eigenvectors. The low-dimensional embedding Y is then constructed as

$$Y = \left[\sqrt{\lambda_1}v_1, \sqrt{\lambda_2}v_2, \ldots, \sqrt{\lambda_k}v_k\right]^T \in \mathbb{R}^{n\times k}.$$

This ensures that the squared Euclidean distances in the embedded space optimally approximate the input dissimilarities, preserving as much variance as possible in the lower-dimensional representation. The resulting configuration of points can be interpreted as a projection into the principal subspace determined by the dissimilarity structure of the data.

When applied to Euclidean distances, classical MDS and PCA produce equivalent embeddings up to scaling and rotation [18, 59]. This equivalence arises because both methods ultimately rely on eigenvalue decomposition to obtain a low-dimensional representation. However, MDS can also be applied to non-Euclidean dissimilarities, making it more flexible for general distance-preserving

embeddings. Unlike PCA, which explicitly assumes an Euclidean metric in the original space, MDS can handle arbitrary dissimilarity measures, such as geodesic distances, correlation-based distances, or edit distances. This allows MDS to be used in various domains where the notion of distance is not strictly Euclidean, such as shape analysis, gene expression data, and linguistic similarity studies. In such cases, the eigenvalue decomposition of B still provides an embedding, but the resulting configuration may not correspond to an exact Euclidean space, instead reflecting the inherent structure of the dissimilarity measure used [18, 27, 72].

2.7 Kinetic Data Structure

A kinetic data structure (KDS) [45, 9, 46] maintains a system of objects v that move along a known continuous *flight plan* as a function of time, denoted as $v = f(v)$. *Certificates* are conditions under which the data structure is accurate, and *events* track when the certificate fails and what corresponding *updates* need to be made. The certificates of the KDS form a proof of correctness of the current configuration function at all times. Updates are stored in a priority queue, keyed by event time. Finally, to advance to time t, we process all the updates keyed at times before t and pop them from the queue after updating. We continue until t is smaller than the first event time in the queue.

A KDS is evaluated by four measures. We say a quantity is *small* if it is a polylogarithmic function of n, or is $O(n^{\varepsilon})$ for arbitrarily small ε, and n is the number of objects. A KDS is considered *responsive* if the worst-case time required to fix the data structure and augment a failed certificate is small. *Locality* refers to the maximum number of certificates in which any one value is involved. A KDS is local if this number is small. A *compact* KDS is one where the maximum number of certificates used to augment the data structure at any time is of complexity $O(n \text{ polylog } n)$, or $O(n^{1+\varepsilon})$. Finally, we distinguish between *external events*, i.e., those affecting the configuration function (e.g., maximum or minimum values) we are maintaining, and *internal events*, i.e., those that do not affect the outcome. We consider the ratio between the worst-case total number of events that can occur when the structure is advanced to $t = \infty$ and the worst-case number of external to the data structure. The second number is problem-dependent. A KDS is *efficient* if this ratio is

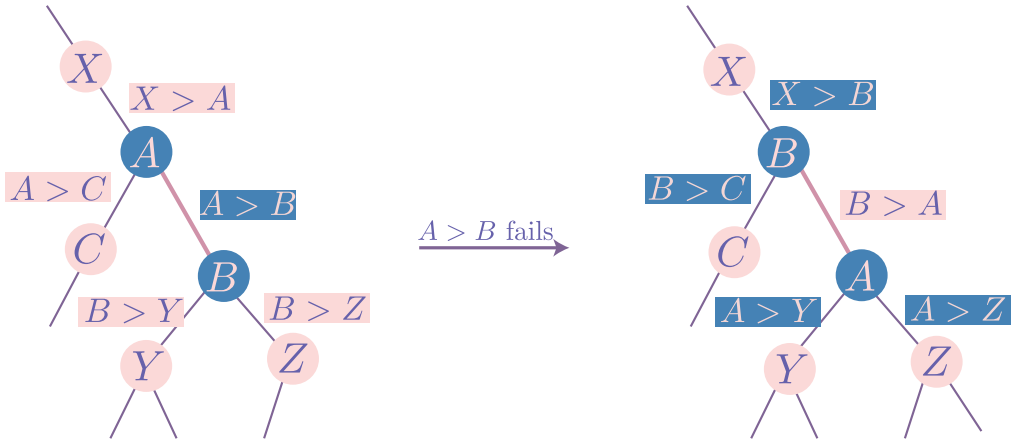


Figure 2.5 Kinetic heap event updates.

small.

We next explore kinetic solutions to upper- and lower-envelope problems. The deterministic algorithm uses a kinetic heap, whereas the more efficient kinetic hanger adds randomness.

2.7.1 Kinetic Heap

The kinetization of a heap results in a *kinetic heap*, which maintains the priority of a set of objects as they change continuously. Both maximum and minimum are examples of priorities. A kinetic heap follows the properties of a static heap such that objects are stored as a balanced tree. If v is a child node of u, then u has a higher priority than v. In the example of a max heap, that means $u > v$. In a kinetic max heap, the value of a node is stored as a function of time $f_X(t)$. The data structure augments a certificate $[A > B]$ for every pair of parent-child nodes A and B. It is only valid when $f_A(t) > f_B(t)$. Thus, the failure times of the certificate are scheduled in the event queue at time t such that $f_A(t) = f_B(t)$. When a certificate $[A > B]$ fails, we swap A and B in the heap and make 5 parent-child updates. This is the total number of certificates in which any pair of A and B are present. The kinetic max heap supports operations similar to a static heap, such as create-heap, find-max, insert, and delete. Insertion and deletion are performed in $O(\log n)$ time.

This KDS satisfies the responsiveness criteria because, in the worst case, each certificate failure only leads to $O(1)$ updates. Locality, compactness, and efficiency all depend on the behavior of $f_X(t)$. The analysis below assumes the case of affine motion where $f_X(t) = at + b$. In the worst

Scenario	Kinetic Heap	Kinetic Hanger
Lines	$O(n \log^2 n)$	$O(n \log^2 n)$
Line segments	$O(n\sqrt{m} \log^{3/2} m)$	$O(n\alpha(m) \log^2 m)$
s-intersecting curves	$O(n^2 \log^2 n)$	$O(\lambda_s(n) \log^2 n)$
s-intersecting curve segments	$O(mn \log^2 m)$	$O(n/m\lambda_{s+2}(m) \log^2 m)$

Table 2.3 Deterministic complexity of the kinetic heap and expected complexity of the kinetic hanger. Here, n is the total number of elements, m denotes the maximum number of elements stored at a given time, s is a constant.

case, each node is present in only three certificates, one with its parent and two with children, resulting in 3 events per node. For compactness, the total number of scheduled events is $n - 1$ for n nodes in the kinetic heap. It is also efficient, where the maximum number of events processed is $O(n \log n)$ for a tree of height $O(\log n)$. The total time complexity of the linear case is $O(n \log^2 n)$ [10]. The results for locality and compactness hold for the more general setting of s-intersecting curves, where each pair of curves intersect at most s times, $s \in \mathbb{Z}$. The efficiency is unknown in this case and the time complexity is $O(n^2 \log^2 n)$ [11].

When the functions are only defined within an interval of time, we call them segments. We perform an insertion when a new segment appears and a deletion when a segment disappears. In the scenario of affine motion segments, the complexity of the kinetic heap is $O(n\sqrt{m} \log^{3/2} m)$, n is the total number of elements and m is the maximum number of elements stored at a given time. In the case of s-intersecting curve segments, the complexity of the kinetic heap is $O(mn \log^2 m)$.

2.7.2 Kinetic Hanger

More efficient solutions to this problem have been developed, such as the kinetic *heater* and *tournament* [11, 9]. While both improve some aspects of the heap, they each have downsides. The kinetic heater increases the space complexity and is difficult to implement, and the tournament has a high locality bound. We instead use the *kinetic hanger*, introduced by da Fopnseca et al. in [30]. This modifies the kinetic heap by incorporating randomization to balance the tree, meaning that all complexity results are expected rather than deterministic.

Given an initial set of elements S, sorted by decreasing priorities, we construct a hanger $hanger(S)$ by so-called *hanging* each element at the root. To hang an element e at node v, if no

element is assigned to v, we assign e to v and return. Otherwise, we use a random seed r to choose a child c_r of v and recursively hang e at c_r. Insertion is similar to hanging - it only requires the additional step of comparing the priorities of e and e', where e' is the element already assigned to v. Deletion can be done by removing the desired element and going down the tree, replacing the current element with its child with the highest priority.

As shown in [30], the kinetic hanger has $O(1)$ locality. The number of expected events in the affine motion case is $O(n^2 \log n)$ and $O(\lambda_s(n) \log n)$ for n s-intersecting curves. The expected runtime is then obtained by multiplying by $O(\log n)$ for each event. We use $\alpha(\cdot)$ to denote the inverse Ackerman function and λ_s is the length bound for the Davenport-Schinzel Sequence; see [4] for details. When the functions are partially defined, the complexities are $O(n\alpha(m) \log^2 m)$ for line segments and $O(n/m\lambda_{s+2}(m) \log^2 m)$ for curve segments.

CHAPTER 3

LABELED MERGE TREE TRANSFORM DISTANCE

This chapter is motivated by the question of how to define and efficiently compute the distance between any two embedded graphs in a way that encodes information about the embedding itself. We will develop a distance that summarizes the outer shape of graphs and reduces the computation complexity by eliminating their inner structures. To preserve as much useful information as possible from the original data, we introduce a new distance by setting the merge trees in the context of the directional transform, see Section 2.5. Given a pair of embedded graphs, we represent them as a family of labeled merge trees using a surjective multi-labeling scheme and then compute the distance between two families of representative matrices. We show theoretically desirable qualities and provide two methods of computation: approximation via sampling and exact distance using a kinetic data structure, both in polynomial time. We demonstrate the utility of the distance by implementing it on two datasets as a proof of concept: one consisting of leaf morphologies and the other of handwritten letters.

3.1 Directional Transform on Embedded Graphs

We take inspiration from the directional transform (Def. 2.2.2) and use the labeled interleaving distance (Def. 2.3.3) on merge trees, inspired by previous work that successfully used a merge tree transform to encode data, but used a different distance for merge trees [12].

Our input data will be embedded graphs. As in Sec. 2.3.1, we treat a graph $G = (V, E)$ as a 1-dimensional stratified space, so that the edges are homeomorphic to copies of the interval $[0, 1]$. We say that a graph $G = (V, E)$ is *embedded* in $\mathbb{R}^d$ if it has a continuous function $\phi : G \to \mathbb{R}^d$ for $d \geq 2$, that is homeomorphic onto its image. This means that each vertex in V is mapped to a point, and each edge in E is mapped to a curve in $\mathbb{R}^d$ so that the graph structure is maintained. For ease of notation, we often denote this information as a pair (G, ϕ). In this chapter, we restrict ourselves to a graph embedded in $\mathbb{R}^2$ where each edge is a straight line between its endpoints, which we refer to as a *geometric graph*.*

*Note that the term geometric graph can have different definitions in the literature; our definition matches the usage

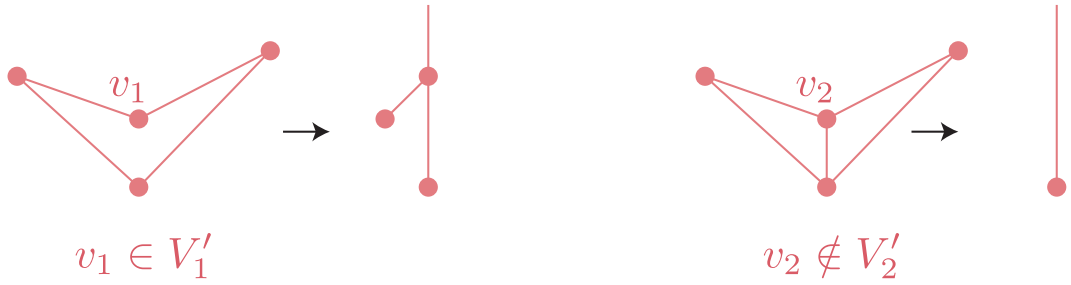


Figure 3.1 Examples of an extremal vertex, v_1, and a non-extremal vertex v_2.

Fix a unit vector direction $\omega \in \mathbb{S}^1$ as in Section 2.2. We can define the height function for direction ω, $f_\omega : G \to \mathbb{R}$, on the embedded graph (G, ϕ) by setting $f_\omega(x) = \langle \phi(x), \omega \rangle$. For a fixed $a \in \mathbb{R}$, the sublevel set at a is homeomorphic to the subgraph G_a induced by the vertices $\{v \mid f_\omega(v) \leq a\}$. Then the merge tree for direction ω, denoted $T^\omega(G)$, is the merge tree of (G, f_ω) (Definition 2.3.1). The merge tree transform is the function

$$\begin{aligned} MTT(G): \ \mathbb{S}^1 &\longrightarrow \mathcal{MT} \\ \omega &\longmapsto T^\omega(G) \end{aligned}$$

Definition 3.1.1 *Let $G = (V, E)$ be a graph with an embedding in $\mathbb{R}^d$. A vertex $v \in V(G)$ is called an extremal vertex if its embedding does* not *belong to the convex hull of its neighborhood $nbr(v) = \{w \in V(G) \mid vw \in E(G)\}$. The collection of extremal vertices, denoted as $V'(G) \subseteq V(G)$, is called the extremal vertex set.*

See Figure 3.1. These vertices completely characterize the collection of vertices that result in a leaf in the merge tree for some direction, as shown in the following lemma.

Lemma 3.1.2 *Fix a straight-line embedded graph (G, ϕ). A vertex $v \in V(G)$ gives birth to a leaf in a merge tree $T^\omega(G)$ for some ω if and only if $v \in V'(G)$ (i.e. v is an extremal vertex).*

Proof. For a straight-line embedded graph, the leaves of the merge tree can be characterized by the following property. A vertex $v \in V(G)$ gives birth to a leaf in a merge tree for direction ω if and only if there is no neighbor u of v with function value $f_\omega(u) \leq f_\omega(v)$.

For the forward direction, assume v gives birth to a leaf in a merge tree for some direction $\omega \in \mathbb{S}^1$. This means that for all $u \in nbr(v)$, $f_\omega(u) > f_\omega(v)$. Let u' be the neighbor with the

in [44] with the additional restriction that the edges cannot cross.

lowest function value among them. Then there is a line normal to ω, which is the level set at $\{x \in \mathbb{R}^2 \mid f_\omega(x) = \frac{1}{2}(f_\omega(u') - f_\omega(v))\}$. This line separates the point v from $nbr(v)$, so it must also separate the convex hulls. Thus v is not in the convex hull of its neighbors, so $v \in V'$.

For the other direction, assume we have an extremal vertex $v \in V'$. By the hyperplane separation theorem (Theorem 2.4.2) and because v is its own convex hull, there exists a line that separates v and the convex hull of $nbr(v)$. Choose ω to be the normal to this line such that $f_\omega(v)$ is less than all its neighbors (else take the opposite normal). Then for the function f_ω, all neighbors have function values greater than v, and so v gives birth to a leaf in direction ω as required. □

Note that this result immediately provides the assumptions discussed earlier to induce a labeling on the merge tree from a labeling on the graph.

Corollary 3.1.3 *Given a map $\pi : [n] \to V(G)$ which is surjective on the extremal vertex set, the induced labeling $\bar{\pi} : [n] \to V(T^\omega(G))$ is surjective on the leaves for any $\omega \in \mathbb{S}^1$.*

3.2 Distance between Embedded Graphs

In this section, we give the definition of a distance between embedded graphs via the labeled interleaving distance (Definition 2.3.3). At a high level, given two embedded graphs G_1, G_2, we will compute their merge trees (T_1, f_1) and (T_2, f_2) for height functions f_ω over all $\omega \in \mathbb{S}^1$. We then introduce a method to label the merge trees with a common label set using the geometric embedding in order to compute the labeled interleaving distance. We then define a distance between graphs by integrating the distance over all $\omega \in \mathbb{S}^1$. We conclude the section with a discussion of properties of the distance.

3.2.1 Surjective Labeling Scheme

Given embedded graphs $\phi_1 : G_1 \to \mathbb{R}^2$ and $\phi_2 : G_2 \to \mathbb{R}^2$, let V_1 and V_2 denote the vertex sets respectively, where their extremal vertex sets are $|V_1'| = n_1$ and $|V_2'| = n_2$. We start by constructing a labeling on the embedded graphs with the same label set $[n_1 + n_2]$, which then induces a labeling on the merge trees for any fixed ω.

Recall that a vertex is extremal if it is not in the convex hull of its neighbors in the graph, and define the subsets $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ to be the respective extremal vertex sets. Enumerate these

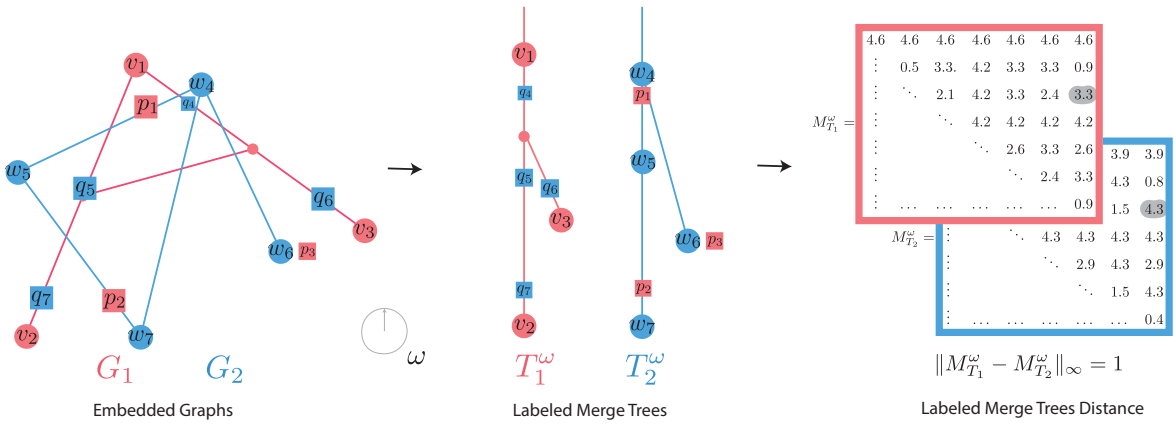


Figure 3.2 Given G_1, G_2, and a direction ω as shown, we perform the surjective labeling scheme on $v \in V_1' \cup V_2'$. Here, any named vertex is extremal, and so, generates a label in the merge trees.

vertices by setting $V_1' = \{v_1, \cdots, v_{n_1}\}$ and $V_2' = \{w_{n_1+1}, \cdots, w_{n_1+n_2}\}$. Next, for each $v_i \in V_1'$, let p_i be a point (choosing arbitrarily if non-unique) which is closest in the embedding of G_2 via the Euclidean distance; so

$$\|f_1(v_i) - f_2(p_i)\| = \min_{x \in G} \|f_1(v_i) - f_2(x)\|.$$

Note that p_i can be a vertex of graph G_2 or an interior point on an edge, in which case we will subdivide the edge and add p_i to the vertex set V_2. Similarly, find a point $q_j \in G_1$ which is closest to $w_j \in V_2'$, adding a q_j to V_1 if necessary. This gives a labeling from the set $[n_1 + n_2]$ to each geometric graph, which we denote as π_1 and π_2, given by

$$\pi_1(i) = \begin{cases} v_i & i \in [1, n_1] \\ q_i & i \in [n_1 + 1, n_1 + n_2] \end{cases} \qquad \pi_2(i) = \begin{cases} p_i & i \in [1, n_1] \\ w_i & i \in [n_1 + 1, n_1 + n_2]. \end{cases}$$

This labeling can be pushed forward to the merge tree for any direction, so we abuse notation and write $\pi_1 : [n_1 + n_2] \to V(T_1)$ and $\pi_2 : [n_1 + n_2] \to V(T_2)$ for the resulting labelings. Note that by Cor. 3.1.3, because the original labeling was defined for all extremal vertices, the resulting labeling is surjective on the leaves as is required by the interleaving distance definition. The induced matrices of T_1^ω and T_2^ω therefore have size $(n_1 + n_2) \times (n_1 + n_2)$. See Fig. 3.2 for an example of the resulting labeled merge trees and their matrices.

3.2.2 The Labeled Merge Tree Transform Distance

For a given pair of graphs, we fix the labels from the previous section, and use this to induce a labeling on the merge tree for each direction $\omega \in \mathbb{S}^1$ resulting in a pair of labeled merge trees T_1^ω and T_2^ω. Denote the induced matrices of T_1^ω and T_2^ω as $M_{T_1}^\omega$ and $M_{T_2}^\omega$ respectively. As seen in Defn. 2.3.3, we have a distance between the labeled trees for every direction, $d(T_1^\omega, T_2^\omega) = \|M_{T_1}^\omega - M_{T_2}^\omega\|_\infty$. To get a distance between the two collections of trees, we integrate over this distance as follows:

Definition 3.2.1 *The* labeled merge tree transform (LMTT) distance *between G_1 and G_2 is defined as*

$$D(G_1, G_2) = \frac{1}{2\pi} \int_0^{2\pi} d(T_1^\omega, T_2^\omega)\, d\omega \; = \frac{1}{2\pi} \int_0^{2\pi} \|M_{T_1}^\omega - M_{T_2}^\omega\|_\infty \, d\omega$$

where the labeling of G_1 and G_2 is done as in Sec. 3.2.1.

We use the term distance for this comparison measure, as we have not yet shown that this definition satisfies all the axioms of a metric. Recall the following properties from Definition 2.5.1.

Definition 3.2.2 *Let S be a set with a given function $\delta : S \times S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, where $\mathbb{R}_{\geq 0}$ is the non-negative extended real line. If δ satisfies finiteness, identity, and symmetry, it is called a* dissimilarity function *[54]. If δ satisfies all five properties, then it is a* metric*; it is a* semi-metric *if it satisfies all but subadditivity.*

Theorem 3.2.3 *The LMTT distance D is a dissimilarity function. That is, it satisfies finiteness, identity, and symmetry.*

Proof. For the proof, we assume we have inputs (G_1, f_1) and (G_2, f_2) with labels $\pi_1 : [n+m] \to V(G_1)$ and $\pi_2 : [n+m] \to V(G_2)$ given by the previous section throughout.

Finiteness: Because we assume that the input graphs are finite, the image $f_2(G_1) \cup f_2(G_2)$ is contained in a finite bounding box. This in turn means that there is a global bound B such that $|f_\omega(x) - f_\omega(y)| \leq B$ for any $x, y \in f_2(G_1) \cup f_2(G_2)$ and $\omega \in \mathbb{S}^1$. Then the difference between any entries in $M_{T_1}^\omega$ and $M_{T_2}^\omega$ is also bounded by B, so $d(T_1^\omega, T_2^\omega) \leq B$ as well. This implies that the integral of the function is also bounded, so D is finite.

Identity: When computing $D(G, G)$, we make the tautological observation that the closest point of G to $x \in G$ is, of course, x. Thus, the extremal vertices will each have a pair of labels: i and

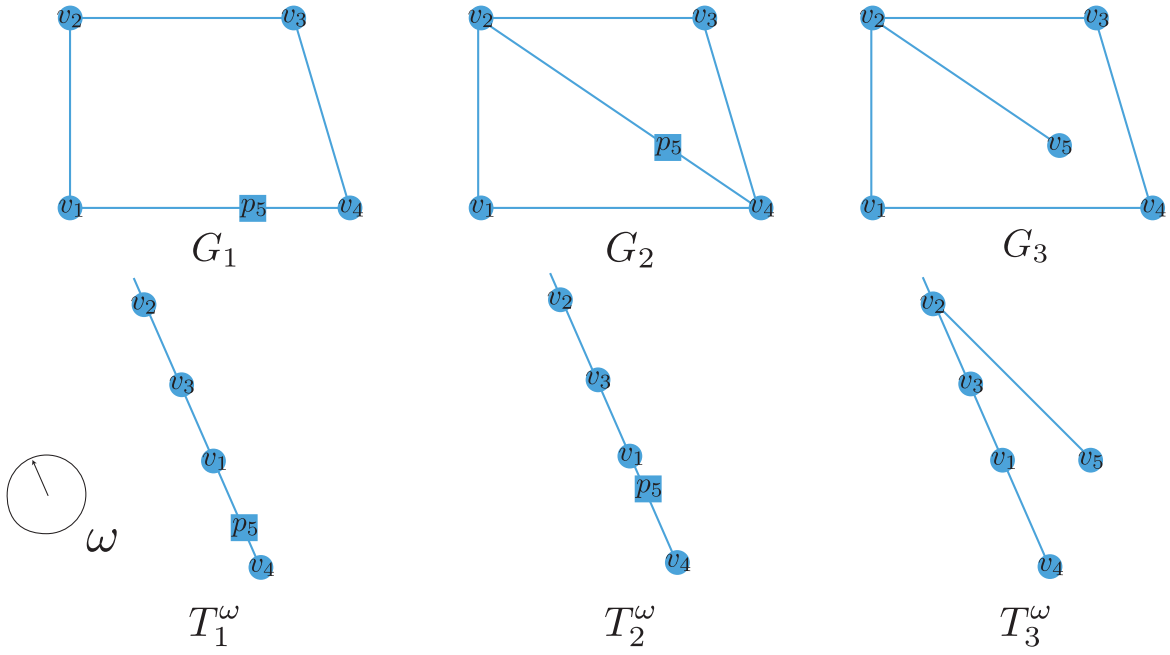


Figure 3.3 A counterexample for both separability and the triangle inequality for the LMTT distance.

$i + n$ where $n = |V'(G)|$ and this labeling will be the same on both graphs. Further, this means the resulting matrices and merge trees will be the same for each, so the distance for any direction is 0, and thus the integral is also 0.

Symmetry: This is immediate from both the symmetry of the labeling and the symmetry of the L_∞ norm of matrices. □

The LMTT distance is not a metric since it does not satisfy the separability or subadditivity properties. Both can be seen in the counterexample of Figure 3.3. Here, G_1 is a simple trapezoid shape with vertices $v_1, \ldots, v_4$. These vertices are embedded identically for G_2 and G_3, but with the addition of one more edge from v_2 to v_4 in G_2, and an extra edge and vertex in G_3 that is a subset of the diagonal edge in G_2. The merge tree of both G_1 and G_2 from any direction is a single edge, so the LMTT cannot hope to detect the difference between them, and separability fails. For the triangle inequality, note that $d(G_1, G_2) = 0$ but that $d(G_1, G_3) > d(G_2, G_3)$ in any direction ω, since v_5 can project with distance 0 to p_5 on G_2, but must project on G_1 to a point at positive distance. Hence, $d(G_1, G_3) > d(G_2, G_3) + d(G_1, G_2)$.

3.2.3 Continuity

In this section, we study continuity of the distance with respect to the varying parameter ω. In the example of Figure 3.4, we see that varying the direction leads to different merge tree structures for the same input graph. When computing the distance between the embedded graphs, this means that we need to separate regions of $\mathbb{S}^1$ for which the underlying graph of the merge tree is the same. To find these regions, we first define a collection of *critical angles*.

Definition 3.2.4 *Given a graph $G = (V, E)$, for all $e \in E$, we define* critical angles *to be the set of all normal vectors for each e_i. Specifically, if we denote $(v_i, v_j) = e$, then*

$$\mathit{Crit}(G) := \left\{ \theta \in \mathbb{S}^1 \;\middle|\; \vec{\theta} \cdot (\vec{v_i} - \vec{v_j}) = 0 \text{ for some } (v_i, v_j) = e \in E \right\}. \tag{3.1}$$

Since the number of edges of a finite graph is finite, we have a finite number of critical angles. Recall that two merge trees are combinatorially the same if the underlying trees are the same up to merging/subdivision of monotone edges and ignoring anything about the function values. A crucial property of critical angles is that they define where the combinatorial merge tree could change given a continuous direction input from $\mathbb{S}^1$.

Lemma 3.2.5 *Denote $\mathit{Crit}(G) = \{\theta_1, \theta_2, \cdots, \theta_n = \theta_0\}$ ordered counterclockwise around the circle. This forms a subdivision of $\mathbb{S}^1$ such that for any $\omega_1, \omega_2 \in (\theta_i, \theta_{i+1})$, T^{ω_1} and T^{ω_2} are combinatorially equivalent.*

Proof. We will show that the set of vertices of G giving birth to a connected component in direction ω_1 and ω_2 are the same. We will do the same for the set of vertices causing a merging. This implies that even if the function values on the merge tree are different, the combinatorial structure of the trees will be the same.

Assume we have a birth causing vertex $v \in V(G)$ for direction ω_1. By Lemma 3.1.2, this implies that v is an extremal vertex, and so it is not in the convex hull of its neighbors. As in the proof of Lemma 3.1.2, v causes a birth in direction ω_1 if and only if all neighbors lie above the hyperplane perpendicular to ω_1 through v. Because passing from ω_1 to ω_2 does not cross any critical angles, the neighbors of v must also lie above the hyperplane perpendicular to ω_2 through

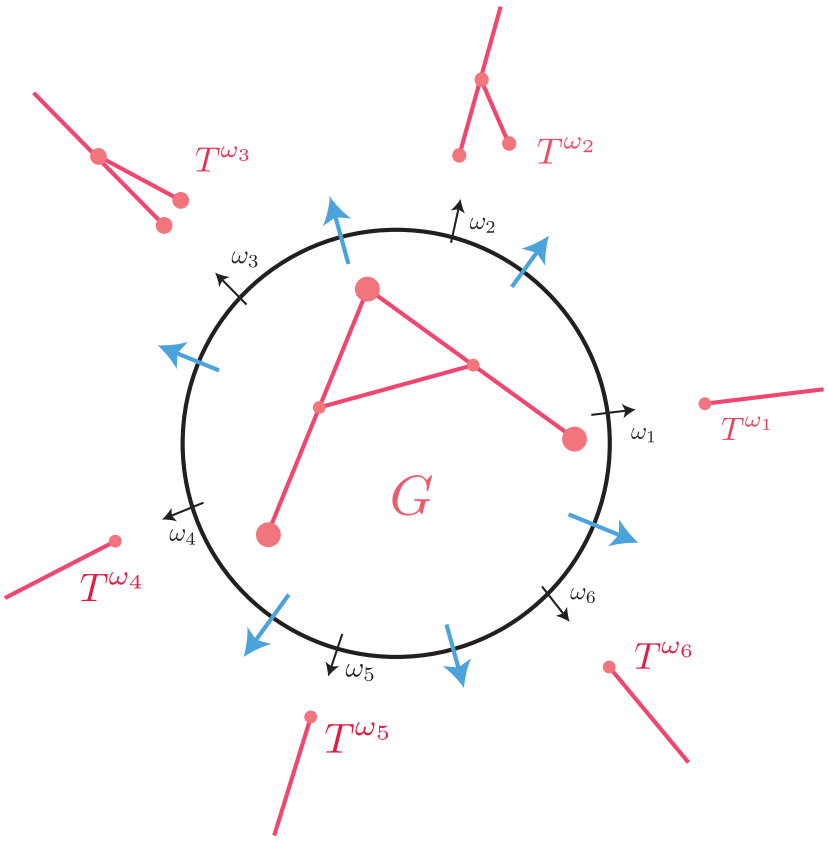


Figure 3.4 For a given graph G, the critical angles are marked in blue. An example merge tree for each region of $\mathbb{S}^1$ between critical values is given.

v. Thus v is still a birth causing vertex for direction ω_2. This implies that the set of birth causing vertices are the same.

Then, assume that v merges two connected components in the direction ω_1. This occurs if and only if there are two connected components in the sublevel set $f_{\omega_1}^{-1}(f_{\omega_1}(v) - \delta)$ for any small $\varepsilon > 0$. This sublevel set must be the same as $f_{\omega_2}^{-1}(f_{\omega_2}(v) - \varepsilon)$, otherwise there would be a critical angle between ω_1 and ω_2. Thus v is a merge vertex for ω_1 if and only if it is also a merge vertex for ω_2. □

For an example of this lemma, see Figure 3.4, where the critical angles are shown in blue. Within the regions separated by the critical angles, the tree structure remains constant. However, passing across a critical angle, such as from ω_1 to ω_2, the tree structure can change. Note that we could still have critical θ_i across which the combinatorial structure does not change, such as from

ω_6 to ω_1, thus the lemma implies that the set of potential changes for the merge tree structure is a subset of Crit(G). With this in hand, we see that the distance function is piecewise continuous with discontinuities given by the set of critical angles.

Theorem 3.2.6 *Fix two geometric graphs $\phi_1 : G_1 \to \mathbb{R}^2$, $\phi_2 : G_2 \to \mathbb{R}^2$, and define $F(\omega)$ to be the L_∞ norm between the two matrices obtained from the merge trees computed using direction; that is, $\omega \in \mathbb{S}^1$*

$$\begin{aligned} F: \ \mathbb{S}^1 &\to \mathbb{R} \\ \omega &\mapsto \|M_1^\omega - M_2^\omega\|_\infty. \end{aligned}$$

Then, $F(\omega)$ is piecewise continuous with the set of discontinuities given by a subset of Crit(G).

Proof. Let $\pi_1 : [n] \to G_1$ and $\pi_2 : [n] \to G_2$ be the labelings on the geometric graphs as defined in Sec. 3.2.1. Denote $v_i = \pi_1(i) \in G_1$ and $w_i = \pi_2(i) \in V(G_2)$ for all i. Now fix a pair i, j and we will first show that between critical angles, every entry in $M_1^\omega - M_2^\omega$ is continuous.

First, we will show that the entry of $M_1^\omega[i, j]$ is continuous with respect to ω. Between critical angles, by Lem. 3.2.5, we know that the combinatorial tree remains the same. This means that the vertex in the merge tree which is the LCA of v_i and v_j remains the same. Further, the vertex $v \in V(G)$ which causes this merge point is also the same. Thus, the entry $M_1^\omega[i, j] = f_\omega(v)$, which is continuous. The same argument holds for $M_1^\omega[i, j]$, thus the difference between the matrices is continuous as required. Finally, since the L_∞ norm of a matrix is continuous for continuously varying entries, we have that F is continuous between critical angles. □

3.3 Implementation

Here, we discuss our implementation of the labeled merge tree transform distance computation, starting with two embedded graphs as input and returning either an exact value or an approximation of the final integrated L_∞ norm. The open source code is provided in a Github repository at github.com/elenawang93/Labeled-Merge-Tree-Transform. The main idea of the procedure is as follows.

1: Compute Crit$(G) =$ Crit$(G_1) \cup$ Crit(G_2). Retain a list of *key angles*, i.e., midpoints between each pair of adjacent critical angles.

2: Create the labeling scheme of Sec. 3.2.1 for a pair of input graphs G_1 and G_2 by determining the collection of extremal vertices of each along with projection to the other graph.

3: Build the merge tree for each key angle and for each vertex of the tree, associate the vertex of G which determines its function value.

4: Use this structure to compute the piecewise-continuous function for every entry in the matrix $\omega \mapsto |M_1^\omega - M_2^\omega|$.

5: For any ω, find the maximum of these entries and integrate the resulting L_∞ norm distance output to obtain the LMTT distance.

We highlight the key algorithms in more detail while providing a run-time analysis of these sections with some commentary on decisions and future work. To fix notation, we assume we have graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, with $n_i = |V_i|$ and $m_i = |E_i|$ for $i = 1, 2$, and let $n = n_1 + n_2$ and $m = m_1 + m_2$. As these are embedded graphs in $\mathbb{R}^2$, recall that Euler's formula implies $m = O(n)$.

Step 1 takes $O(m)$ time since we only need to keep a list of normal vectors for each edge. Specifics of this can be found in Algorithm 3.2. Next, we analyze the runtime of Step 2's computation by breaking it into two pieces: finding the extremal vertices and then using these to construct the labeling scheme. We use the *PointPolygonTest* in OpenCV [19] to find the sets of extremal vertices in each graph. OpenCV uses an optimized implementation of a ray-casting method to determine whether a point is inside a polygon, determining whether a vertex v is extremal in $O(\deg(v))$, where $\deg(v)$ is the degree of the vertex. This is done by leveraging the Jordan Curve Theorem, which states that a point is inside a closed curve if and only if a ray extended from that point to infinity crosses the curve an odd number of times. By checking the point with a ray extended WLOG horizontally to the right for computational efficiency, OpenCV computes the cardinality of the number of edge-crossings, where the domain of the intersection is within the line segment of the polygon, as a valid crossing. Thus, determining whether each vertex is extremal is computed in $O(\sum_{v \in V_1 \cup V_1} \deg(v)) = O(m)$ time. Let n' be the total number of extremal vertices in the two graphs, so that the label set is $[n']$. Note that $n' \leq n$.

For the labeling scheme, we take each extremal vertex in $v \in V_1$ and compute its distance to each edge $e \in E_2$. If the nearest point is on an edge, we subdivide the edge by adding a vertex with corresponding labels. If it is a vertex, we add a label to the existing vertex. We repeat this process for each extremal $v \in V_2$ to find their nearest points on G_1. This requires up to $(n_1(n_2 + m_2) + n_2(n_1 + m_1)) = O(mn)$ distance calculations. In total, this means that Step 2 takes time $O(m + mn) = O(mn)$. At most, we add $n' \leq n$ additional vertices and edges by adding a degree two node that is not already a vertex.

For Step 3, we use a union-find data structure and follow a similar approach to [70], which takes $O(m_i \log n_i)$ steps in the worst case on a graph with n_i vertices and m_i edges. Our implementation produces a merge tree as a digraph represented in NetworkX. We leverage the union-find to track the updating state of known connected components as we sweep along the given angle. This requires extra control over the naming of each union-find group, as we track the connected groups from the known top to incrementally build the merge tree upward. One way to handle this is to require a non-commutative `union()` operation (i.e., `union(`n`, `m`) ≠ union(`m`, `n`)`). The second option which we have chosen is to extend a generic union-find with a `reroot(`n`)` method in our implementation that sets a specified element as the representative member of its group. Going forward, we will use `union(`n`,`m`)` to represent unioning m to n and rerooting the component as m.

Standard implementations of union find will employ path compression for computational efficiency, which we also maintain, allowing us to benefit from the amortized constant time property of `find()`. Also note that the projected nodes of the labeling scheme 3.2.1 do not have to be leaves or merge points in the merge tree but still need to be represented as part of the merge tree output, which will differ from a generic merge tree construction algorithm. Finally, as a further optimization as we sweep along the angle, we pass on a list of "group candidates" to currently unvisited nodes. Put another way, when each node is being processed, we only need to consider connected components from already visited nodes.

This merge tree construction is performed at each of the $|\text{Crit}(G_i)| = 2m_i$ directions for

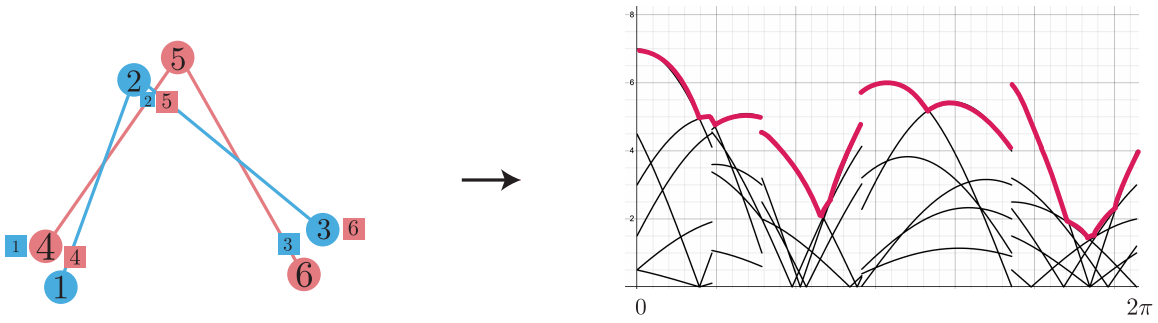


Figure 3.5 Illustration of KDS maintaining the maximum value for computing the LMTT distance. Each curve represents how values change in each matrix entry.

$i = 1, 2$, resulting in a total complexity of $O(m^2 \log n)$ for Step 3. See Algorithm 3.3, where for conciseness, a union() operation also implies adding a node and drawing an edge on the merge tree digraph. Processing the merge trees to find the LCA matrix uses Tarjan's off-line algorithm [71], as implemented in NetworkX [47]. This method leverages a depth-first search combined with the union-find data structure, resulting in a complexity of $O(n^2)$ to compute the LCAs of all node pairs in a graph given a specific direction. With the $2m$ directions, the whole of Step 4 has a complexity of $O(mn^2)$.

Putting this together, Step 1 through Step 4 take $O(m + mn + m^2 \log n + mn^2)$ time; using Euler's formula to see that $O(m) = O(n)$, this yields a total running time of $O(n^3)$. See Fig. 3.5 for an example of the output after these steps. In this figure, at right we have a collection of functions giving the entries for each pair of labels; i.e. for each entry in the difference matrix.

The goal of Step 5 is to determine the maximum value of these functions over ω. We provide two computation methods for this final step: an exact but slower computation using kinetic data structures and a faster approximation-based approach.

3.3.1 Exact Distance

To compute the exact distance, we implemented a kinetic data structure (KDS), see Section 2.7. When computing the maximum entry of the difference matrix $\widetilde{M}^{\omega} := |M_1^{\omega} - M_2^{\omega}|$, we can construct the problem as the kinetic maximum maintenance problem. The state-of-the-art algorithm incorporates a heap structure [10, 29]. Each entry in the matrix can be considered as an object

Algorithm 3.1 Tarjan's Offline LCA Algorithm for Full Matrix

Require: Graph G with n nodes
Ensure: Matrix LCA storing the LCA for every pair (i, j)
  Initialize union-find structure for G
  Initialize an array $ancestor[]$ to store the ancestor of each node
  Initialize matrix $LCA[n][n]$
  function TarjanOLCA(u)
    $ancestor$[Find(u)] $\leftarrow u$
    for each child v of u in G **do**
      TarjanOLCA(v)
      Union(u, v)
      $ancestor$[Find(u)] $\leftarrow u$
    Mark u as visited
    for each visited node v **do**
      if $v \neq u$ **then**
        $LCA[u][v] \leftarrow LCA[v][u] \leftarrow ancestor$[Find($v$)]
  Choose an arbitrary node as the root and call TarjanOLCA(root)

Algorithm 3.2 Computation of Graph Distance at Key Angles

Require: Embedded Graphs, G_0, G_1
  Project vertices from G_0 and G_1 onto each other
  Calculate critical angles C from each edge
  Compute midpoints M between critical angles
  for all ω_i in C, M **do**
    Build merge trees T_0 and T_1 at ω_i
    Use nodes of T_1 to update G_0 edges (and vice versa) to get G_0', G_1'
    Build new merge trees for G_0' and G_1'
    Compute LCA matrix (see Algorithm 3.1) per graph and save by ω_i
Ensure: LCA matrices keyed by ω_i

moving along a flight plan with a parametrization of the form $y(\omega) = a \sin \omega + b \cos \omega$, where a and b are constants. The certificate is a maximum heap that accurately represents the relations between the values of the entries in the matrix. The certificate fails when two curves cross each other, and the failure times are the angles at which the crossings happen. Since all of our objects have the same frequency of 2π, each pair of curves crosses exactly twice in $[0, 2\pi]$. Explicitly, for $v_1 = a_1 \sin \omega + b_1 \cos \omega$ and $v_2 = a_2 \sin \omega + b_2 \cos \omega$, they cross at $\omega_1 = \arctan\left(\frac{a_1 - a_2}{b_1 - b_2}\right)$ and $\omega_2 \equiv \omega_1 \pmod{\pi}$. The ω for which any crossing happens is the key of the event priority queue and the two intersecting objects are the values. We perform the update by swapping the locations of the two objects in the maximum heap. Finally, by maintaining the maximum object, we find the

Algorithm 3.3 Merge Tree Construction via Union-Find

Require: Embedded Graph G as one connected component

Require: Angle ω for filtration

1: Compute normal line to filtration angle F given ω
2: Compute heights from F to each node in G, and store in sorted set n
3: Initialize a set c_n for each node in G to track known connected components during iteration 14
4: **for all** node n in n' **do**
5: Track n as visited
6: **if** c_n is empty **then**
7: Draw n as leaf in merge tree M
8: **else**
9: **if** c_n has length 1 **then**
10: `union(`g`,` n`)`
11: **else**
12: **for all** c_{ni} of c_n **do**
13: `union(`c_{n_0}`,` c_{n_i}`)`
14: **for all** node n'_i connected to n **do**
15: **if** n'_i is not visited **then**
16: Add `find(`n`)` to $c_{n'_i}$
17: Update set $c_{n'_i}$ with `find(`$c_{n'_i}$`)` per element
18: Add infinity node to M

Ensure: Merge tree M as digraph

distance function and integrate it to obtain the final distance. See Figure 3.5 for an example.

This approach is implemented in our code by maintaining a kinetic heap of n^2 objects between each pair of $2m$ critical angles. Consequently, the complexity of this computation is

$$O(2m \cdot n^4 \log n^2) = O(n^5 \log n)$$

[9]. This results in an overall complexity for steps 1-5 of

$$O(m + mn + m^2 \log n + mn^2 + n^5 \log n),$$

which simplifies to $O(n^5 \log n)$ for the deterministic algorithm.

This can be further optimized to

$$O(2m \cdot \lambda_2(n^2) \log n^2) = O(n^3 \log^2 n)$$

by using a kinetic hanger [4], which introduces randomness to the algorithm. Here, λ_2 is the length bound of a Davenport–Schinzel sequence, where the subscript 2 indicates the objects are

2-intersecting curves. The final expected running time for steps 1-5 is then of $O(n^3 \log^2 n)$ using the kinetic hanger data structure.

3.3.2 Approximated Distance

Although the previous section gives a computation for the exact distance, it leads to a practical slowdown. While there is code for the exact method in our repository, we use the following sampling method in practice, as it performs significantly better on our data sets.

After computing the LCA matrix at each key angle, we check for the maximum entry in the difference matrix, which takes $O(n^2)$ time. We sample to get ω in K directions and use the trapezoid rule to integrate; see Algorithm 3.4. While the worst running time in this approach is of $O(m + mn + m^2 \log n + mn^2 + n^2 K) = O(n^3 + n^2 K)$, shaving off only a $\log^2 n$ factor in the worst case, it runs significantly faster in practice compared to the implemented kinetic heap.

We quantify the quality of this approximation by the trapezoidal rule error analysis. Given a function F, sampled at K evenly spaced points over the interval $[A, B]$, the error E_T can be approximated by

$$|E_T| \leq \frac{(B-A)^3}{12K^2} \max(F'').$$

Since the LMTT distance function $F(\omega) : \mathbb{S}^1 \to \mathbb{R}$ is piecewise continuous and is composed of functions in the form of $y(\omega) = a \sin \omega + b \cos \omega$, we have $\max(y''(\omega)) = \sqrt{a^2 + b^2}$. Let $a = v_1 - u_1$ and $b = v_2 - u_2$ for any two vertices $v = (v_1, v_2) \in G_1$ and $u = (u_1, u_2) \in G_2$, and let R be the radius of the bounding circle that contains both G_1 and G_2. This gives $\sqrt{a^2 + b^2} \leq 2R$. Therefore, we can bound the second derivative by $\max F''(\omega) \leq 2R$. Hence, we have a coarse bound given by

$$|E_T| \leq \frac{\max(F'')(B-A)^3}{12K^2} = \frac{2R(2\pi)^3}{12K^2} = \frac{4}{3} \cdot \frac{R\pi^3}{K^2}.$$

3.4 Applications

To test the meaning of our distance in practice and its implementation, we use it to visualize the structural differences in two different data sets.

Algorithm 3.4 Compute Graph Distance for Any Angle

Require: Precomputed LCA matrix of graphs, L_0, L_1 for every key angle ω_i

Require: Angle ω

if ω is a critical angle **then**
    Look up difference matrix D
    $d = \|X\|_\infty$
else
    Compute midpoint of critical angle interval containing angle and lookup corresponding LCA matrix
    Get unit vector v given ω
    $X_0 \leftarrow X_{0|i,j} = \langle v * L_{0|i,j} \rangle$
    $X_1 \leftarrow X_{1|i,j} = \langle v * L_{1|i,j} \rangle$
    $d = \|X_1 - X_0\|_\infty$

Ensure: d

3.4.1 Isomorphic graphs: *Passiflora* leaves

The first data set we examine consists of *Passiflora* leaves [23]. This dataset contains 3319 leaves from 226 plants across 40 species which were classified into seven morphotypes based on traditional morphometric techniques. Each entry represents a leaf, with 15 Procrustes-aligned (x, y)-coordinate pairs representing the locations of 15 landmarks. We construct a graph for each leaf by connecting adjacent landmarks, creating an outline that approximates the shape of the leaf. The resulting graphs consist solely of extremal vertices, making this data set an efficient use case of our metric. See Figure 3.6 for a comparison of each species' canonical leaf shape with the average graph. Examples of the seven morphotypes for classification can be seen in the center of Fig. 3.7.

Biologically, *Passiflora* leaves are of great interest to biologists due to their shape diversity while being in the same genus. Also of interest is the relationship between leaf shape and maturity. As leaves develop along the vine, their shape changes dramatically from juvenile leaves to mature leaves to aged leaves; this phenomenon is known as *heteroblasty*. Young leaves from plants of different species often look more similar to each other than to mature leaves from the same plant. In order to select leaves that are the most representative of their species, we sample one leaf from each of the 226 plants that represents the mature shape, with a heteroblasty number of around 0.5.

The results of this analysis are shown in Figure 3.7. First, on the right of Fig. 3.7 we have reproduced the analysis of the leaves from [23]. Treating each data point as a 30-dimensional

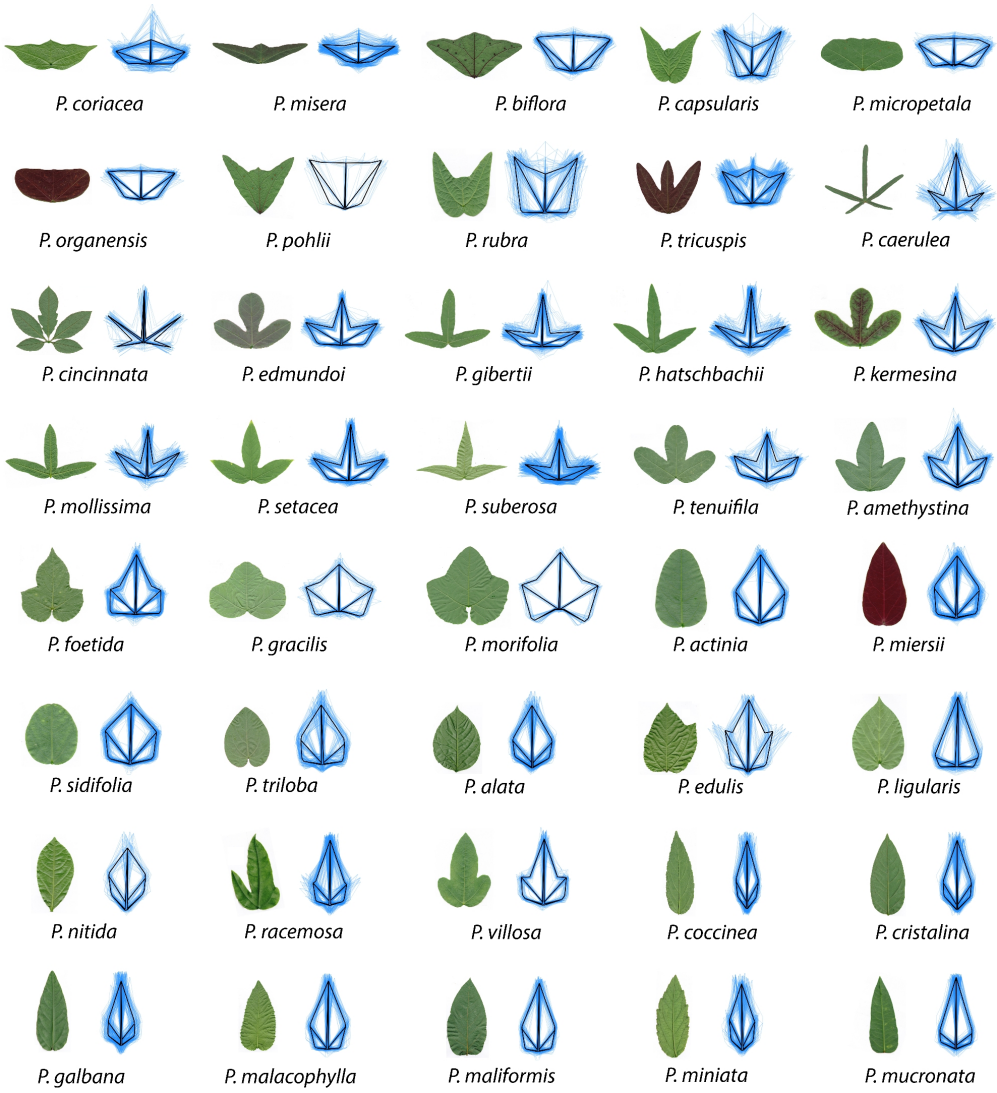


Figure 3.6 The shapes of *Passiflora* leaves measured using landmarks from [23].

vector (which is only possible because of the landmark labels), the authors of [23] used Principal Component Analysis as the dimension reduction algorithm. While this plot shows some separation between morphotypes, the overlapping points make the visualization difficult to interpret.

We then compute pairwise LMTT distances between all the leaves (note that this does not utilize the landmark labels in any way) and construct a 226×226 distance matrix. In order to to visualize the relationships given by this distance, we use Multi-Dimensional Scaling (MDS) [27] which is shown in the left of Fig. 3.7.

Comparing the LMTT to the PCA, we see a similar global structure but with better separation

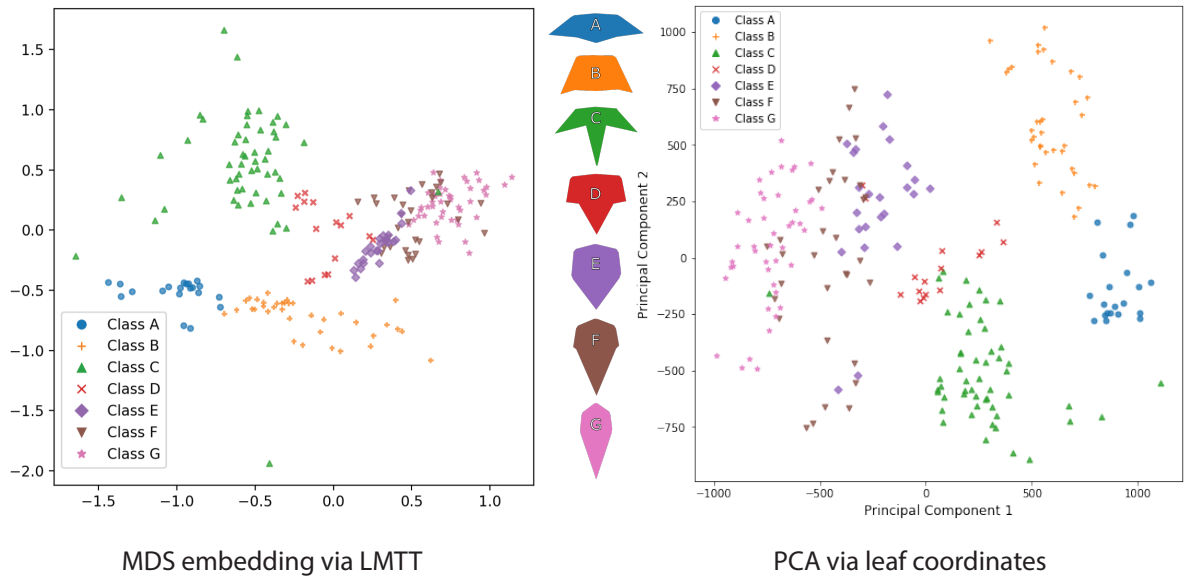


Figure 3.7 MDS and PCA plots with points colored by morphotype.

amongst points. However, we note in particular that the LMTT comparison is done with no memory of the landmark structure, meaning we have similar results with less input information. We note that most of the overlapping in the MDS plot occurs amongst the leaves of morphotypes E, F, and G, which have very similar outlines as seen in the middle examples of Fig. 3.7. Amongst leaves with sufficiently different outlines, such as type A, B, C, and D, we see the distinction reflected by the embedding. This leads us to believe that the LMTT is a good tool for use in distinguishing shapes, so long as they are sufficiently distinct.

3.4.2 Non-isomorphic graphs: letters

The second data set we examine is a set of different letters from the IAM graph database [65]. There are example letter graphs in this data set which are quite distorted; therefore, we restricted the data set to the graphs with only one connected component. Further, the dataset consists of some unrecognizable data points since they are generated from hand-written letters. Therefore, we created a template graph for each letter to remove these outliers from the data before running the experiment. We removed data points if they did not have the same number of vertices and the same edge list as their letter's template. The data set we end up with has 450 data points: 15 different letters, and 30 graphs per letter. Each letter graph has 2 to 7 vertices and 1 to 6 edges. We follow

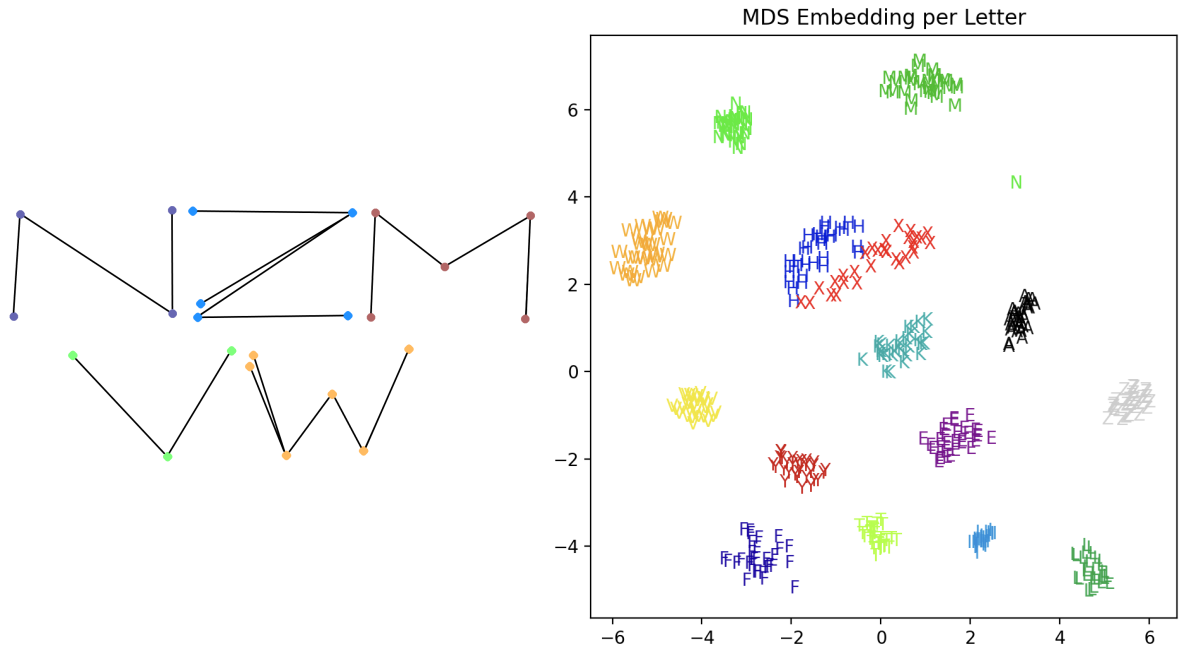


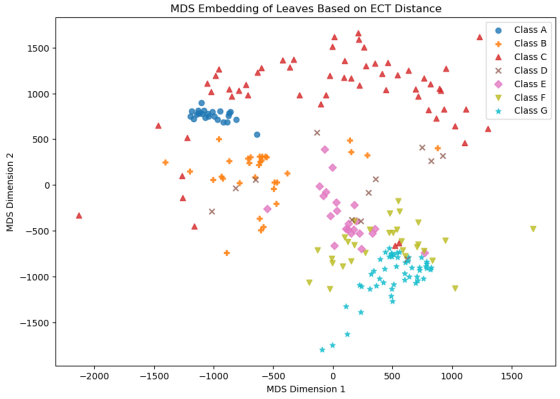
Figure 3.8 At left, an example of letter graphs from the IAM data set. At right, the MDS plot labeled by letter computed using the LMTT distance.

the same process as in the leaf shapes by computing all pairwise LMTT distances and constructing a 450×450 distance matrix.

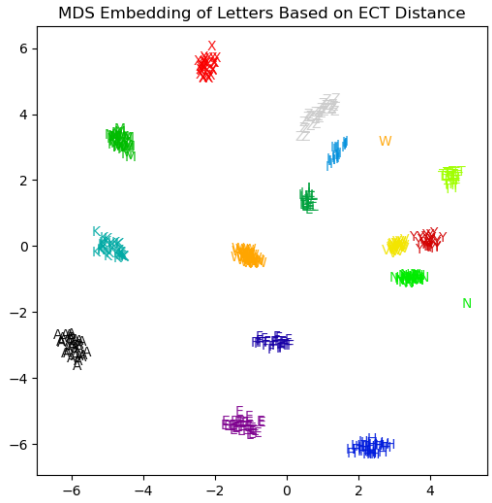
The relationship between the letters is then visualized via MDS; see Fig 3.8. We observe clear separation of clusters amongst the different letters. The nearby clusters also follow what we expect, given the choice of merge tree as our topological signature. For example, the letters "H" and "X" can generate similar merge trees in certain directions, and the MDS plot shows those clusters close together. The graphs in this data set differ from the leaf example in that they have different numbers of vertices and edges; hence, they are non-isomorphic. This shows the utility and meaningfulness of our distance on two datasets that have different isomorphism types.

3.5 Comparison with Other Distances

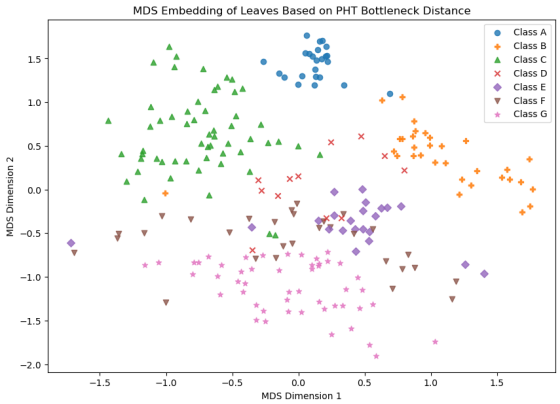
In this section, we compare the MDS embedding computed via four other distances to gain empirical knowledge of what the LMTT is capturing. The results below show the visualization of dissimilarity matrices of the same sampled datasets via the Hausdorff, Fréchet, ECT, and PHT distances.



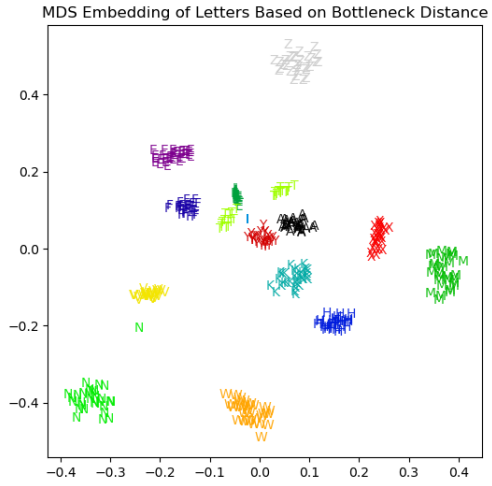
(a) Leaves - ECT distance



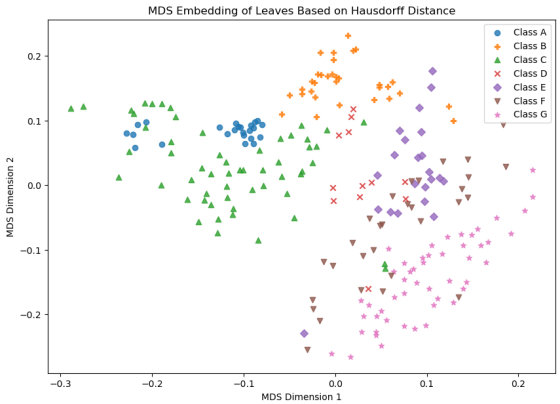
(b) Letters - ECT distance



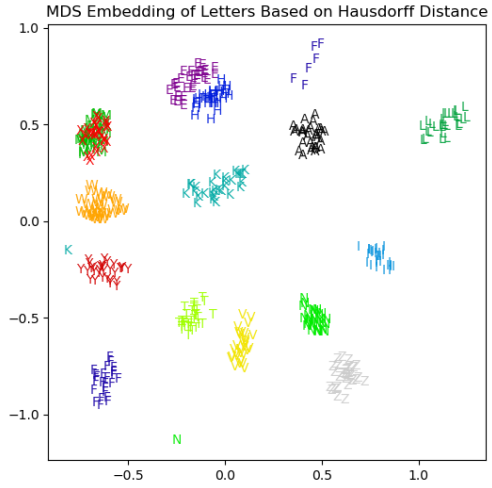
(c) Leaves - Bottleneck distance



(d) Letters - Bottleneck distance



(e) Leaves - Hausdorff distance



(f) Letters - Hausdorff distance

Figure 3.9 Comparison of MDS embeddings of leaves (left column) and letters (right column) using different topological distances.

Intuitively, the Hausdorff and Fréchet distances capture more geometry of the data than the topology, whereas the ECT and PHT capture the topology, like the LMTT. While the ECT, PHT, and LMTT all aim to capture the 0-dimensional topology, the PHT contains strictly more information than the ECT. The LMTT tracks additional merging information, which PHT does not. In terms of computation time, the ECT is the fastest to compute, the PHT the second, and the LMTT trades additional information for computational time.

Figure 3.9 presents a comparison of MDS embeddings for the above two datasets. The left column contains the embeddings for the Passiflora leaf dataset, while the right column contains the embeddings for the letter dataset. Each row corresponds to a different distance measure used to compute pairwise dissimilarities before applying MDS.

- Top row (Figures 3.9a, 3.9b): The MDS embeddings based on the Euler Characteristic Transform (ECT) distance. In both datasets, distinct clusters emerge, corresponding to different classes of shapes. For the leaf dataset, the clusters are less separated than compared to the MDS embedding based on the LMTT.
- Middle row (Figures 3.9c, 3.9d): The embeddings derived from the PHT bottleneck distance. Compared to ECT, the bottleneck distance separates the classes more clearly. However, there is no clear distinction between each of them.
- Bottom row (Figures 3.9e, 3.9f): The embeddings computed using the Hausdorff distance. This metric considers worst-case deviations, leading to a different arrangement of clusters compared to the other two distances.

Overall, the difference in the results of using these distances is more visible on the leaf dataset. We see that the LMTT distance is capable of abstracting both the topological and geometric properties of the dataset. For the letter dataset, while each class is more clearly separated using different distances, the LMTT distance still presents the clearest distinction and presents the least number of outliers. We see only one ‘N’ outlier in the LMTT embedding, while the clearest ECT embedding among the other three distances presents two outliers —‘N’ and ‘W’.

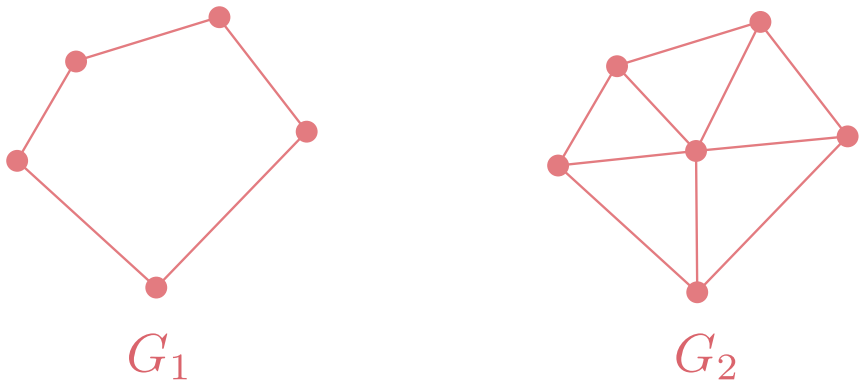


Figure 3.10 An example of graphs with LMTT distance 0.

3.6 Conclusion and Discussion

In this chapter, we have given a new distance between embedded graphs by constructing a labeling procedure and computing distance based on the labeled merge tree interleaving distance. We proved the theoretical properties of the labeled merge tree transform distance and used them for optimization in its implementation. We showed its utility by using it to discriminate among leaves in the *Passiflora* data set and between letter graphs in the IAM data set.

Our method comes with its own strengths and limitations. As a strength, it is easy to compute compared to many embedded graph distances; see [20] for further details on other options. In the future, we aim to optimize the algorithm and implementation to allow for applications to larger data sets. However, while it summarizes the outer shape of the data and differentiates pairs of data points that are similar, we note that it cannot distinguish the internal structures of data. For example, the distance between a hexagon and hexagon with a central vertex as in Fig.3.10 is 0 since the merge trees are the same for the two graphs in any direction. Another limitation of the method is that it requires the data to be pre-aligned, a difficult problem in its own right. Taken in another light, however, the distance is based on the embedding information so we would expect that alignment of the data is taken into account when computing this distance.

We also note that there is an instability inherent to the labeling procedure described in Section 3.2.1. See the example of Fig. 3.11 where a slight edit to the location of the vertex w_5 from G_2 to G_2' results in the label jumping from one edge of G_1 to the other. This occurs because the

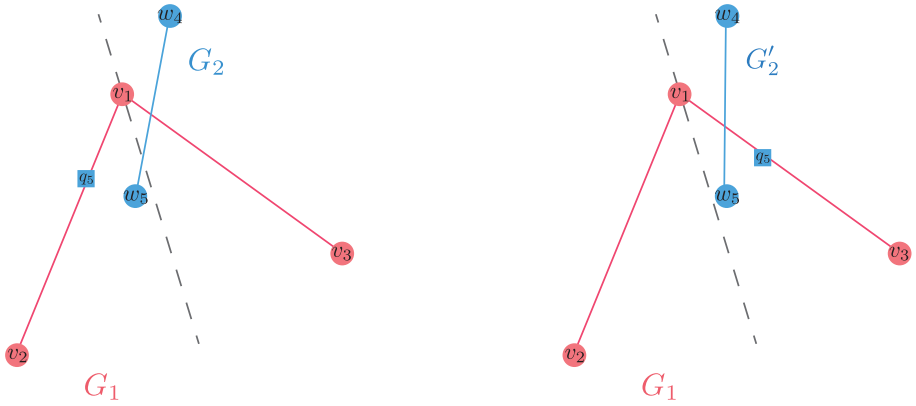


Figure 3.11 An example of a small change in a graph resulting in a large change in labeling locations.

vertex is moved past the bisector of the angle at v_1. For this reason, it would be interesting to see if there are other, more stable methods for labeling which can be used in the procedure while still resulting in similar distance properties. While this example shows that stability for the entire collection of embedded graphs is unlikely, it would be interesting to see if this distance is stable for restricted sets. These restricted settings may also allow for stronger properties of the distance such as separatbiliy and the triangle inequality.

Finally, in theory this definition should be applicable to graphs and/or simplicial complexes embedded in higher dimensions $\mathbb{R}^d$. The labeled merge tree transform would result in collections of matrices parameterized by $\mathbb{S}^{d-1}$ for higher dimensional spheres, so further care would need to be taken to understand how to track the maximum value of the distance for the integral definition.

CHAPTER 4

THE KINETIC HOURGLASS DATA STRUCTURE

The *kinetic data structure* framework has been applied to many geometric problems since it was introduced, including but not limited to finding the convex hull of a set of moving points in the plane [9], the closest pair of such a set [9], a point in the center region [1], kinetic medians and kd-trees [3], and range searching, see Section 2.7 for an introduction. In this chapter, we extend the framework to the geometric matching problem. Specifically, we are interested in the min-cost matching of a weighted graph with continuously changing weights on the edges.

We have seen in previous chapters that one of the most common comparisons between persistence diagrams is to compute the bottleneck distance, which can be formulated as a min-cost matching problem in graphs. Improving the algorithm to compute the bottleneck distance has been studied extensively, both theoretically and practically [33, 52, 39, 55].

Vineyards are continuous families of persistence diagrams that represent persistence for time-series of continuous functions [26, 77, 56]. PHT is a special case of vineyards. It possesses desirable properties such as continuity, injectivity, and stability, see Chapter 2. However, an efficient method for comparing two PHTs has not been thoroughly investigated. In particular, existing approaches to compute the bottleneck distance between two PHTs of shapes in $\mathbb{R}^2$ rely on sampling various directions, which only provides approximate results. For example, the results in Section 3.5 are computed at sampled directions. Prior to the work of this dissertation, there is no known solution for computing the exact bottleneck distance. This chapter presents the first method to address this challenge, providing a precise and efficient solution.

We construct a new kinetic data structure that maintains a min-cost matching and the bottleneck distance of a weighted bipartite graph with continuously changing weight functions. We evaluate the data structure based on the four standard metrics in the KDS framework. We provide a specific case study where the weighted graph is computed from two persistent homology transforms.

This chapter is organized as follows: we cover the bottleneck matching problem's background in Section 4.1. In Section 4.2, we iterate the events and updates necessary to maintain the KDS and

the overall complexity analysis. Finally, we focus in Section 4.3 on our case study of the persistent homology transform for a special case of embedded objects, called star-shaped objects, as defined in [8].

4.1 Geometric Matching Problem

Broadly, we are interested in a geometric matching problem. Here we give an abstraction of the construction discussed in Section 2.1.1. Given an undirected graph $G = (V, E)$, a *matching* M of G is a subset of the edges E such that no vertex in V is incident to more than one edge in M. A vertex v is said to be *matched* if there is an edge $e \in M$ that is incident to v. A matching is *maximal* if it is not properly contained in any other matching. A *maximum* matching M is the matching with the largest cardinality; i.e., for any other matching M', $|M| \geq |M'|$. A maximum matching is always maximal; the reverse is not true.

For a graph $G = (X \sqcup Y, E)$ where $|X| = |Y| = n$ and $|E| = m$, a maximum matching is a *perfect* matching if every $v \in X \sqcup Y$ is matched, and $|M| = n$. This can be expressed as a bijection $\eta : X \to Y$. For a subset $W \subseteq X$, let $N(W)$ denote the neighborhood of W in G, the set of vertices in Y that are adjacent to at least one vertex of W. Hall's marriage theorem provides a necessary condition for a bipartite graph to have a perfect matching [48].

Theorem 4.1.1 (Hall's Marriage Theorem) *A bipartite graph $G = (X \sqcup Y, E)$ has a perfect matching if and only if for every subset W of X: $|W| \leq |N(W)|$.*

Building on Hall's Marriage Theorem, we extend our focus to weighted bipartite graphs. Given such a graph, a fundamental optimization problem is to identify matchings that minimize the maximum edge weight, known as the *bottleneck cost*.

Definition 4.1.2 *A weighted graph $\mathcal{G} = (G, c)$ is a graph G together with a weight function $c : E \to \mathbb{R}_+$. The* bottleneck cost *of a matching M for such a $\mathcal{G}$ is $\max\{c(e) \mid e \in M\}$. The* bottleneck edge *is the highest weighted edge in M, assuming this is unique. A perfect matching is* optimal *if its cost is minimal among all perfect matchings. An optimal matching is also called a* min-cost *matching.*

To find a maximum matching of a graph, we use augmenting paths.

Definition 4.1.3 *For a graph G and matching M, a path P is an* augmenting path *for M if:*

1. *the two end points of P are unmatched in M, and*
2. *the edges of P alternate between edges $e \in M$ and $e \notin M$.*

Berge's Theorem provides the necessary and sufficient conditions for a matching to be maximum in terms of the existence of an augmenting path.

Theorem 4.1.4 (Berge's Theorem) *A matching M in a graph G is a maximum matching if and only if G contains no M-augmenting path.*

The existing algorithms that compute the bottleneck cost are derived from the Hopcroft-Karp maximum matching algorithm [52], which we briefly review. Given a graph $G = (X \sqcup Y, E)$ where $|E| = n$, and an initial matching M, the algorithm iteratively searches for augmenting paths P. Each phase begins with a breadth-first search from all unmatched vertices in X, creating a layered subgraph of G by alternating between $e \in M$ and $e \notin M$. It stops when an unmatched vertex in Y is reached. From this layered graph, we can find a maximal set of vertex-disjoint augmenting paths of the shortest length. For each P, we augment M by replacing $M \cap P$ with $P \backslash M$. We denote this process as $\mathrm{Aug}(M, P) = M \setminus (M \cap P) \cup (P \setminus M)$. Note that $|\mathrm{Aug}(M, P)| = |M| + 1$, so we can repeat the above process until no more augmenting paths can be found. By Theorem 4.1.4, the resulting M is maximum. The algorithm terminates in $O(\sqrt{n})$ rounds, resulting in a total complexity of $O(n^{2.5})$. This algorithm was later improved to $O(n^{1.5} \log n)$ for geometric graphs by Efrat et al. by constructing the layered graph via a nearest-neighbor search data structure [39].

As discussed in Section 2.1.1, the bottleneck distance between persistence diagrams can be reduced to the above bottleneck cost problem.

4.2 Kinetic Hourglass

In this section, we introduce a new kinetic data structure that keeps track of the optimal bottleneck cost of a weighted graph $\mathcal{G} = (G, \mathrm{c})$ where c changes continuously with respect to time t. We recall that the key elements of a KDS are certificates, events, and updates, see Section 2.7. The *kinetic hourglass* data structure is composed of two kinetic heaps; in Section 4.2.3 we will give the details for replacing these with kinetic hangers. One heap maintains minimum priority, and the

other maintains maximum. Assume we are given a connected bipartite graph $G = (V, E)$ with the vertex set $V = X \sqcup Y$, where $|X| = |Y| = n$; and edge set E, where $|E| = m$. If G is a complete bipartite graph, then $m = n^2$. The weight of the edges at time t is given by $\mathrm{c}^t : E \to \mathbb{R}_+$. Denote the weighted bipartite graph by $\mathcal{G}^t = (G, \mathrm{c}^t)$. The weights of those m edges are the objects we keep track of in our *kinetic hourglass*. We assume that these weights, called flight plans in the kinetic data structure setting, are given for all times $t \in [0, T]$.

Let $G^t_\delta \subseteq G$ be the portion of the complete bipartite graph with all edges with weight at most δ at time t; i.e., $V(G^t_\delta) = V$, and $E(G^t_\delta) = \{e \in E \mid \mathrm{c}^t(e) \leq \delta\}$. If the bottleneck distance $\hat{\delta}^t = d_B(\mathcal{G}^t)$ is known, we are focused on the bipartite graph $G^t_{\hat{\delta}}$, which we will denote by $\hat{G}^t$ for brevity. By definition, we know that there is a perfect matching in $\hat{G}^t$, which we denote as M^t, although we note that this is not unique. Further, there is an edge $\hat{e}^t \in M^t$ with $\mathrm{c}(\hat{e}^t) = \hat{\delta}^t$, which we call the *bottleneck edge*. This edge is unique as long as all edges have unique weights. We separate the remaining edges into the sets

$$L^t = E(\hat{G}^t) \setminus E(M^t), \text{ and}$$
$$U^t = E \setminus E(\hat{G}^t) = \{e \in E \mid \mathrm{c}^t(e) > \hat{\delta}^t\}$$

so that $E = L^t \sqcup M^t \sqcup U^t$, see Figure 4.1 for an example.

The kinetic hourglass consists of the following kinetic max heap and kinetic min heap. The *lower heap* $\mathcal{H}_L$ is the max heap containing $L^t \sqcup M^t = E(\hat{G}^t)$. The *upper heap* $\mathcal{H}_U$ is the min heap containing $U^t \cup \{\hat{e}\}$. Note that $\mathrm{c}^t(\hat{e}) = \max\{\mathrm{c}^t(e) \mid e \in L^t \sqcup M^t\}$ and $\mathrm{c}^t(\hat{e}) < \min\{\mathrm{c}^t(e) \mid e \in U^t\}$, meaning that $\hat{e}$ is the root for both heaps.

4.2.1 Certificates

The certificates for the kinetic hourglass (i.e., properties held by the data structure for all $t \in [0, T]$) are

1. All max-heap certificates for $\mathcal{H}_L$ and min-heap certificates for $\mathcal{H}_U$.
2. Both heaps have the same root, denoted r^t.
3. The edge r^t is the edge with bottleneck cost; i.e., $r^t = \hat{e}^t$ where $c^t(\hat{e}^t) = \hat{\delta}^t$.

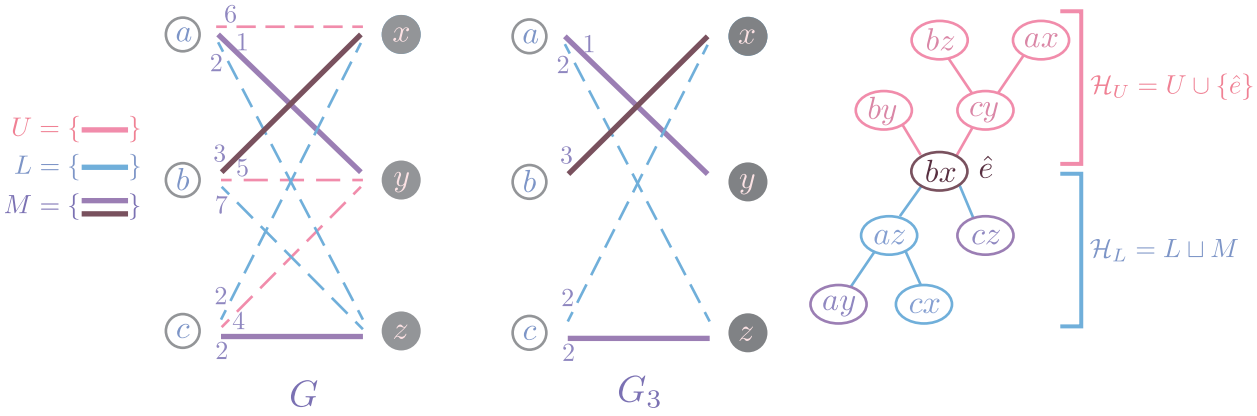


Figure 4.1 Illustration of construction of the kinetic hourglass.

Assuming the certificates are maintained, r^t and $\hat{e}^t$ are the same edge. However, in the course of proofs, bottleneck edge of the matching is denoted by $\hat{e}^t$, while we use r^t for the edge stored in the root of the two heaps (or $r^t(\mathcal{H}_U)$ and $r^t(\mathcal{H}_L)$ when a distinction is needed).

4.2.2 Events

For a particular event time t, we denote the moment of time just before an event by $t^- = t - \varepsilon$ and the moment of time just after by $t^+ = t + \varepsilon$. For two edges, we write $a \preccurlyeq_t b$ to mean that $\mathrm{c}^t(a) \leq \mathrm{c}^t(b)$. The two heaps have their own certificates, events (both internal and external), and updates as in the standard setting. Our main task of this section is to determine which events in the heaps lead to an external event in the hourglass. We define the external events for the hourglass as those events in $\mathcal{H}_U$ and $\mathcal{H}_L$ which lead to changes of the root, r^t. Thus, internal events are those which do not affect the roots; i.e., $r^{t^-}(\mathcal{H}_U) = r^{t^+}(\mathcal{H}_U)$ and $r^{t^-}(\mathcal{H}_L) = r^{t^+}(\mathcal{H}_L)$.

We first show that an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ is an internal event of the hourglass.

Lemma 4.2.1 *If the event at time t is an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ and the kinetic hourglass satisfies all certificates at time t^-, then the edge giving the bottleneck distance for times t^- and t^+ is the same; that is, $\hat{e}^{t^-} = \hat{e}^{t^+}$.*

Proof. Following the previous notation, we have bottleneck distances before and after given by $\hat{\delta}^{t^-} = \mathrm{c}^{t^-}(\hat{e}^{t^-})$ and $\hat{\delta}^{t^+} = \mathrm{c}^{t^+}(\hat{e}^{t^+})$. Because we start with a correct hourglass, we know that $\hat{e}^{t^-} = r^{t^-}(\mathcal{H}_U) = r^{t^-}(\mathcal{H}_L) = r^{t^-}$. By definition, an internal event in either heap is a swap of two elements with a parent-child relationship but for which neither is the root so the roots remain

unchanged; that is, $r^{t^-} = r^{t^+}$ so we denote it by r for brevity. This additionally means that no edge moves from one heap to the other, so the set of elements in each heap does not change and thus $G_{\mathrm{c}^{t^-}(r)} = G_{\mathrm{c}^{t^+}(r)}$. Again for brevity, we write this subgraph as Γ.

We need to show that this edge r is the one giving the bottleneck distance at t^+, i.e. $\hat{\delta}^{t^+} = \mathrm{c}^{t^+}(r)$ or equivalently that $r = \hat{e}^{t^+}$. All edges of the perfect matching from t^-, M^{t^-}, are contained in Γ; thus M^{t^-} is still a perfect matching at time t^+. We show that any minimal cost perfect matching for t^+ must contain r, and thus M^{t^+} is a *minimal* cost perfect matching. Since r is $\hat{e}^{t^-}$, by removing r, $\Gamma \backslash \{r\}$ will cease to have a perfect matching, else contracting the minimality of the perfect matching at t^-. But as the order is unchanged, this further means that for time t^+, any lower threshold for the subgraph $\Gamma = G_{\mathrm{c}^{t^+}(r)}$ results in removing the edge r, but $\Gamma \backslash \{r\}$ will not have a perfect matching, finishing the proof. □

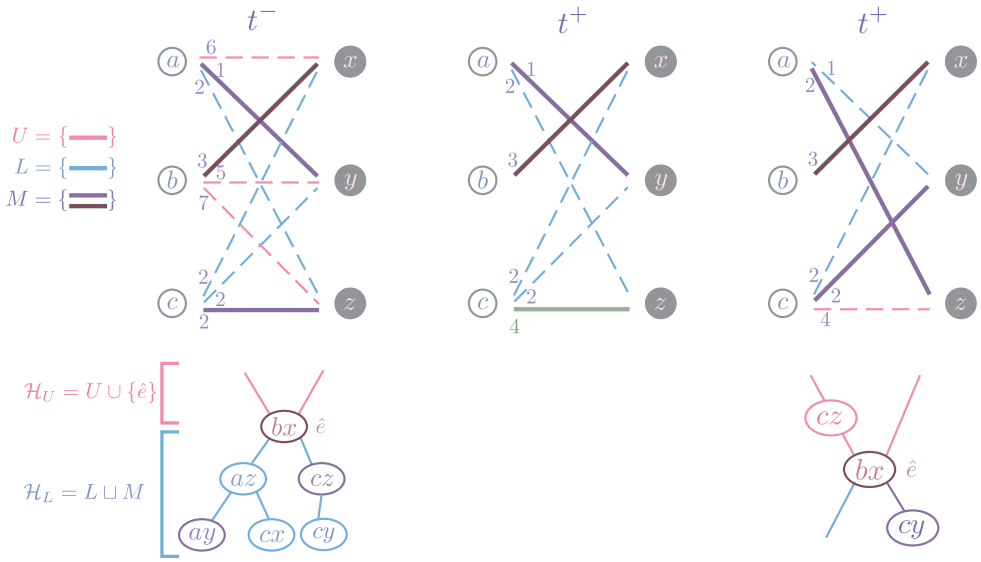
The remaining cases to consider are external events of $\mathcal{H}_U$ and $\mathcal{H}_L$, when a certificate in one of the heaps involving the root fails. This can be summarized in the three cases below. In each case, denote the root at time t^- as $r = r^{t^-}$.

1. (L-Event) Swap priority of r and an $e \in L^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.
2. (M-Event) Swap priority of r and an $e \in M^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.
3. (U-Event) Swap priority of r and an $e \in U^t$ in $\mathcal{H}_U$; i.e. $r \preccurlyeq_{t^-} e$ and $e \preccurlyeq_{t^+} r$.

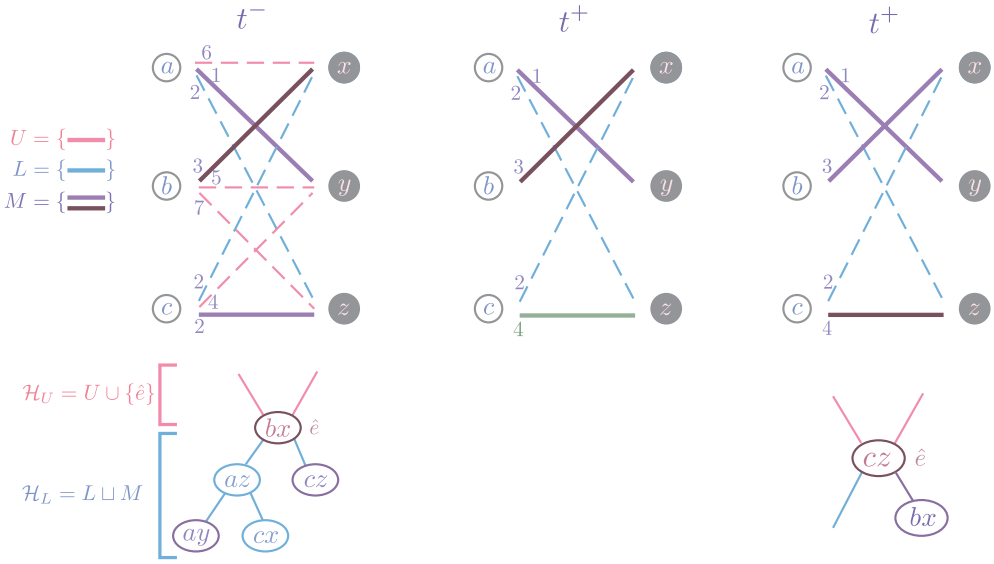
In the remainder of this section, we consider each of those events and provide the necessary updates. The simplest update comes from the first event in the list, since we will show that no additional checks are needed.

Lemma 4.2.2 (L-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in L^{t^-}$ at t; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$. Then e moves from $\mathcal{H}_L$ to $\mathcal{H}_U$, and r remains the root and is the edge with the bottleneck cost for t^+, i.e., $r^{t^+} = \hat{e}^{t^+}$.*

Proof. Denote $M = M^{t^-}$. In this case, e is an edge in the lower heap but $e \notin M$. Because $r \preccurlyeq_{t^+} e$, in order to maintain the heap certificates, either e needs to be inserted into the upper heap with r remaining as the root; or if e remains in the lower heap, it needs to become the new root. Note that although they swap orders, the graph thresholded at the cost of r at t^- and the graph thresholded



(a) Illustration of Scenario 1 of an M-Event



(b) Illustration of Scenario 2 of an M-Event

Figure 4.2 Illustrations of two scenarios of an M-Event; see Lemma 4.2.3.

at the cost of e at t^+ are the same; i.e. $G_{c^{t^-}(r)} = G_{c^{t^+}(e)}$. In addition, $G_{c^{t^+}(r)} = G_{c^{t^+}(e)} \setminus \{e\}$. However, since $e \notin M$, M is still a perfect matching in $G_{c^{t^+}(r)}$. If there exists a perfect matching in $G_{c^{t^+}(r)} \setminus \{r\}$, this would constitute a perfect matching at time t^- which has lower cost than M, contradicting the assumption that M is a minimal cost matching for that time. Thus M is a minimal cost matching for time t^+. □

Lemma 4.2.3 (M-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in M^{t^-}$ at t; i.e. $e \preccurlyeq_{t^+} r$*

and $r \preccurlyeq_{t^-} e$. *Let* $G' = \hat{G}^{t^-} \setminus \{e\}$ *be the graph with* $e = (u, v)$ *removed. Exactly one of the following scenarios happens.*

1. *There exists an augmenting path* P *in* G' *from* u *to* v. *Then* e *moves into the upper heap (*$e \in U^{t^+}$*), the root remains the same (*$\hat{e}^{t^+} = r^{t^+} = r$*), and the matching is updated with the augmenting path, specifically* $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.
2. *There is no such augmenting path. Then* $M^{t^+} = M^{t^-}$ *and* $\hat{e}^{t^+} = r^{t^+} = e$*; i.e. the only update is that* r *and* e *switch places in the lower heap.*

Proof. Let $M' = M^{t^-} \setminus \{e\}$. Then M' is a matching of size $n - 1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

If there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1 and thus M'' is a perfect matching. Note that if $r \in P$, then M'' is also a perfect matching in t^- with strictly cheaper cost than M^{t^-}; but this contradicts the assumption of r giving the bottleneck distance. Thus we can assume that $r \in M''$, and all edges in the lower heap have cost at most that of r. This means that r is still the bottleneck edge of the matching, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4.1.4, however it is not perfect. Note that because $G' = \hat{G}^{t^-} \setminus e$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(r)}$. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(r)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(e)}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M^{t^+}\}$. Therefore, $e = \hat{e}^{t^+} = r^{t^+}$ becomes the root, and r becomes a child of the root e in $\mathcal{H}_L$. □

As observed in Figure 4.2, in Scenario 1, when the edge involved in the event is removed, and an augmenting path ceases to exist, the bottleneck edge remains the same; in Scenario 2, if an augmenting path is found, the bottleneck edge is updated accordingly. The U-Event is processed similarly.

Lemma 4.2.4 (U-Event) *Assume we swap priority of* $r = r^{t^-}$ *and an* $e \in U^{t^-}$ *at* t*; i.e.* $r \preccurlyeq_{t^-} e$ *and* $e \preccurlyeq_{t^+} r$. *Let* $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ *be the graph with* $r = (u, v)$ *removed and* e *included. Exactly*

one of the following events happens.

1. *There exists an augmenting path P from u to v. Then r moves into the upper heap ($r \in U^{t^+}$), e becomes the root ($\hat{e}^{t^+} = r^{t^+} = e$), and the matching is updated with the augmenting path, specifically $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.*
2. *There is no such augmenting path. Then $M^{t^+} = M^{t^-}$ and e moves into the lower heap ($e \in L^{t^+}$), and the root remains the same ($\hat{e}^{t^+} = r^{t^+} = r$).*

Proof. Let $M' = M^{t^-} \setminus \{r\}$, and again M' is a matching of size $n-1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

Similar to the M-Event, if there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1; thus, M'' is a perfect matching. Further, because $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. Therefore $e = \hat{e}^{t^+} = r^{t^+}$, and r gets moved down to $\mathcal{H}_U$ and becomes a child of e.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4.1.4, however it is not perfect. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(e)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(r)} \setminus \{e\}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(r)}$. This is thus an internal event; we move e to $\mathcal{H}_L$ as a child of r, and r remains the bottleneck edge, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. □

4.2.3 Complexity and Performance Evaluation

The kinetic hourglass's complexity analysis largely follows that of the kinetic heap and kinetic hanger [30]. Recall the time complexities of the structures summarized in Table 2.3 are obtained by multiplying the number of events by the time to process each event, $O(\log n)$ for n total object. This is the key difference between those data structures and the kinetic hourglass.

In the kinetic hourglass structure, we maintain two heaps or hangers at the same time. For a bipartite graph G of size $|E| = m$, we maintain the m edges in the kinetic hourglass. The number of edges is also the worst-case maximum number of elements in the max-heap $\mathcal{H}_L$ and the min-heap $\mathcal{H}_U$. We therefore use $m = n$ in our analysis. Within a single heap, we can think of this procedure

Scenario	Kinetic Heap Hourglass	Kinetic Hanger Hourglass
Line segments	$O(m^{5/2}\log^{1/2} m)$	$O(m^2\alpha(m)\log m)$
s-intersecting curve segments	$O(m^3\log m)$	$O(m\lambda_{s+2}(m)\log m)$

Table 4.1 Deterministic complexity of the kinetic heap hourglass and expected complexity of the kinetic hanger hourglass for $|E| = m$.

as focusing on the cost of an edge at time t as $c^t(e)$ only given for its time inside the heap. This means that when we evaluate the runtime for the heap (or hanger) structures, we can think of the function on each edge as being a curve segment, rather than defined for all time.

The time complexities for the kinetic hourglass are summarized in Table 4.1 and we describe them further here. The main change in time complexity results from the augmenting path search required at every external event. Note that in both the L- and M-events from Lemma 4.2.3 and 4.2.4, exactly only one iteration of augmenting path search is required, so this takes $O(|E|) = O(m)$ per iteration. Therefore, we replace a $\log n$ factor with m. When $\mathrm{c}(e)$ is linear, the complexity of the kinetic hourglass is $O(m^2\sqrt{m}\log^{3/2} m)$ using heaps, and is $O(m^2\alpha(m)\log m)$ using hangers, since the arrangement of m lines has a complexity of $O(m^2)$. If the weight function is non-linear, we are in the s-intersecting curve segments scenario. This results in a complexity of $O(m^3\log m)$ and $O(m\lambda_{s+2}(m)\log m)$, when utilizing heaps and hangers respectively. The constant s is determined by the behavior of the weight function.

While this data structure is responsive, local, and compact, following directly from the analysis of kinetic heap and hanger, it fails to be efficient due to the linear time complexity required by each external event of the heap. We conjecture that this can be improved via amortization analysis, which involves investigating the ratio between the number of internal and external events. However, this is a non-trivial problem and has yet to be addressed in existing literature.

4.3 Kinetic Hourglass for Persistent Homology Transform

In the following section, we apply our kinetic hourglass to compute the bottleneck distance between two persistent homology transforms [73, 28, 42], an aforementioned specific case of persistence vineyard [26]. The structure of the PHT can be quite complex, even for simple embedded graph inputs [8]. Thus, we first follow the discussion of general PHT in Section 2.2.1,

but then following [8], focus here on a simple case of input structures where we have additional knowledge of the PHT vineyard.

4.3.1 The Persistent Homology Transform

Recall from Section 2.2.1 the definition of the Persistent Homology Transform given an embedded graph K as input.

Definition 4.3.1 *The* persistent homology transform *of $K \subseteq \mathbb{R}^2$ is the function*

$$\begin{aligned} \mathrm{PHT}(K): \ \mathbb{S}^1 &\to \mathcal{D} \\ \omega &\mapsto \mathrm{Dgm}(h_\omega^K) \end{aligned}$$

where $h_\omega^K : K \to \mathbb{R}$, $h_\omega^K(x) = \langle x, \omega \rangle$ is the height function on K in direction ω, and $\mathcal{D}$ is the space of persistence diagrams.

To compare two complexes, K_1 and K_2, we can take the integrated bottleneck distance between $\mathrm{PHT}(K_1)$ and $\mathrm{PHT}(K_2)$:

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) = \int_0^{2\pi} d_B\left(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})\right) d\omega.$$

For a fixed direction ω, the bottleneck distance between the two persistence diagrams can be formulated as a geometric matching problem as described in Section 2.1.1. Moreover, it is stable in that

$$d_B(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})) \leq \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty$$

for all $\omega \in \mathbb{S}^1$, see Section 2.2.1 for details. Integrating this inequality immediately gives the following result for the integrated bottleneck distance.

Proposition 4.3.2 *For two finite, connected geometric simplicial complexes with the same underlying abstract complex,*

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) \leq 2\pi \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty.$$

A choice of total order on the simplices of K, the persistence induces a pairing on the simplices of K, where for a pair (τ, σ), $\dim(\sigma) = \dim(\tau) + 1$, and a $\dim(\tau)$-dimensional homology class was born with the inclusion of τ and dies with the inclusion of σ. Because the lower

star filtration, the function values of these simplices are given by the unique maximum vertex $v_\sigma = \{v \in \sigma \mid f_\omega(v) \geq f_\omega(u)\ \forall u \in \sigma)\}$ and so the function value when this simplex was added is given by $f_\omega(v_\sigma)$, and v_σ does not change for nearby σ that do not pass through one of the non-generic directions. For this reason, we can associate each finite point (b, d) in the persistence diagram $\mathrm{Dgm}_k(h^K_\omega)$ to a vertex pair (v_b, v_d) with $f_\omega(v_b) = b$ and $f_\omega(v_d) = d$. This pair is well-defined in the sense that given a point in the diagram, there is exactly one such pair. Further, because of the pairing on the full set of simplicies, every birth vertex appears in exactly one such pair, although here a death vertex could be associated to multiple points in the diagram. For any infinite points in the diagram of the form (b, ∞), we likewise have a unique vertex v_b again with $f_\omega(v_b) = b$.

Write a given filtration direction as $\omega = (\cos(\theta), \sin(\theta)) \in \mathbb{S}^1$ and fix a point $x \in \mathrm{Dgm}(h^K_\omega)$ with birth vertex $v_a \in K$ located at coordinates $(a_1, a_2) \in \mathbb{R}^2$ and death vertex $v_b \in K$ at coordinates $(b_1, b_2) \in \mathbb{R}^2$. This means the corresponding point in $x \in \mathrm{Dgm}(h^K_\omega)$ has coordinates which we denote as

$$x(\omega) = (a_1 \cos\theta + a_2 \sin v, b_1 \cos v + b_2 \sin\theta).$$

Notice that for some interval of directions $I_x \subseteq \mathbb{S}^1$, the vertices v_a and v_b will remain paired, and so for that region, we will have the point $x(\omega')$ in the diagram for all directions $\omega' \in I_x$. Viewing this as a function of angle θ, we observe that $x(\omega)$ is a parametrization of an ellipse, meaning that the point $x(\omega)$ in the diagram will trace out a portion of an ellipse in the persistence diagram plane. Further, the projection of this point to the diagonal has the form

$$x'(\omega) = \left(\tfrac{a_1+b_1}{2} \cos\theta + \tfrac{a_2+b_2}{2} \sin\theta, \tfrac{a_1+b_1}{2} \cos\theta + \tfrac{a_2+b_2}{2} \sin\theta\right),$$

which is also a parameterization of an ellipse, albeit a degenerate one.

4.3.2 Kinetic Hourglass for PHT

In this section, we show how the kinetic hourglass data structure investigated in Sec. 4.2 can be applied to compute the exact distance between the 0-dimensional PHTs of two geometric simplicial complexes in $\mathbb{R}^2$, and provide an exact runtime analysis of the kinetic hourglass by explicitly

determining the number of curve crossings in the flight plans for the edges in the bipartite graph. Here, we work in the restricted case of *star-shaped* simplicial complexes to avoid the potential for complicated monodromy in the PHT [51].

Following [8], an embedded simplicial complex K is called *star-shaped* if there is a point $c \in |K|$ such that for every $x \in |K|$, the line segment between c and x is contained in $|K|$. In this context, we call c a (non-unique) *center* of M. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. We say that a function $F : \mathbb{S}^1 \to \mathcal{D}$ has *trivial geometric monodromy* if there exist continuous maps $\{\gamma_i : \mathbb{S}^1 \to \overline{\mathbb{R}}^2\}$ (called vines) such that the off-diagonal points of $PHT(\omega)$ are exactly the set $\Gamma(\omega) = \{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$.

Theorem 4.3.3 ([8]) *Let K be a star-shaped simplicial complex whose vertices are in general position. Then the 0-dimensional persistent homology transform $PHT(K)$ has trivial geometric monodromy.*

Thus, assume we have the vines $\{\gamma_i\}_{i=1}^N$ for the PHT of a star-shaped K. We assume that these vines are written so that they either never touch the diagonal $\Delta = \{(x, x) \in \mathbb{R}\}$, or they each correspond to exactly one entrance and exit from Δ and so are off-diagonal for some interval I_{γ_i}. Further, assume that K is connected so that exactly one of these vines, say γ_1, is the infinite vine of the form $(\tilde{\gamma}(\omega), \infty)$ for $\tilde{\gamma} : \mathbb{S}^1 \to \mathbb{R}$. For a geometric simplicial complex K, a vertex $x \in K$ is called *extremal* if it gives birth to a 0-dimensional class for some direction ω. The set of extremal vertices is denoted as $\left|\text{ext}^K(V)\right|$.

To bound the number of vines N, we call a vertex v in a geometric simplicial complex K an *extremal vertex* if it is not in the convex hull of its neighbors following Definition 3.1.1. These are the vertex that gives birth for at least one direction. Write $\text{ext}(V)$ for the set of vertices of K which are extremal. It is an immediate corollary of the merge tree version Lemma 3.1.2 that a vertex in K gives birth to a 0-dimensional class for some direction ω if and only if it is an extremal vertex. We define the *external edges* of an extremal vertex v to be the one or two edges that are incident to v and are part of the boundary of the convex hull formed by v and its neighbors. The remaining edges incident to v are called *internal edges*. For an extremal vertex, we can find the interval of directions I_v for which it gives birth for all $\omega \in I_v$. We note that this is a connected interval in the

circle, so we have the starting and ending direction ω_1 and ω_2 for which the point exits and enters the diagonal with $I_v = [\omega_1, \omega_2] \subseteq \mathbb{S}^1$. The largest angle among the edges incident to v is formed by the normal vectors of the edges, ω_1 and ω_2. We can write a continuous, piecewise-ellipsoidal path associated with the vertex v as $\mu_v : I_v \to \mathbb{R}^2$; $\omega \mapsto x(\omega)$ for the point $x(\omega)$ in the diagram with birth vertex v, and note that each piece of this path is a portion of an ellipse.

Lemma 4.3.4 *For an extremal vertex v, if $\mu_v(\omega) \in \Delta$ then $\omega = \omega_1$ or $\omega = \omega_2$.*

Proof. Let ω_1, ω_2 be the normal vectors of the external edges (v, u_1), (v, u_2) respectively, and $\mu_v(\omega) = (h_\omega(v), h_\omega(v'))$ where v is the birth-causing vertex and v' is the death causing vertex. Since v is birth-causing, all other neighbors of v have higher height values along direction ω: $\{h_\omega(u) > h_\omega(v) \mid u \in N_K(v), u \neq v'\}$. Because $\mu_v(\omega) \in \Delta$, then $h_\omega(v) = h_\omega(v')$, equivalently $\langle v, \omega \rangle = \langle v', \omega \rangle$. This implies that ω is a normal vector to the edge defined by v and v'. Furthermore, no neighbor of v lies in the half plane defined by (v, v') and $-\omega$. Therefore, (v, v') must be part of the convex hull formed by v and $N_K(v)$. We thus conclude that v' is either u_1 or u_2, and $\omega = \omega_1$ or $\omega = \omega_2$. □

Because the birth of every point in the diagram $\mathrm{PHT}(\omega)$ at any fixed direction is uniquely associated with an extremal vertex, we can see that the off-diagonal points of the PHT can also be written as $M(\omega) = \{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$, and so $M(\omega) = \Gamma(\omega)$. Thus there is an injective map from the set of finite vines γ_i to the vertex giving birth to the corresponding points at the time it comes out of the diagonal. This means that the number of vines is at most the number of extremal vertices, i.e. $N \leq |\operatorname{ext}(V)|$.

Given two star-shaped complexes K_1 and K_2, say the two PHTs are given by the vines $\{\gamma_{1,i}\}_{i=1}^{n_1}$ and $\{\gamma_{2,j}\}_{j=1}^{n_2}$ respectively and write the projection of the respective vine to the diagonal as $\gamma_{k,i}^{\Delta}$. We then construct the complete bipartite graph G with vertex set $U \cup V$ where U and V each have $n_1 + n_2$ vertices, each associated to one of the vines from either vineyard. However, in U we think of these as being the vines $\{\gamma_{1,i}\}_i$ followed by the projections of each vine $\{\gamma_{2,j}^{\Delta}\}_j$ to the diagonal; the reverse is assumed for V. The basic idea is that the cost of an edge between two vertices is based on the distance between the location of the paths (or projection of the paths for the diagonal

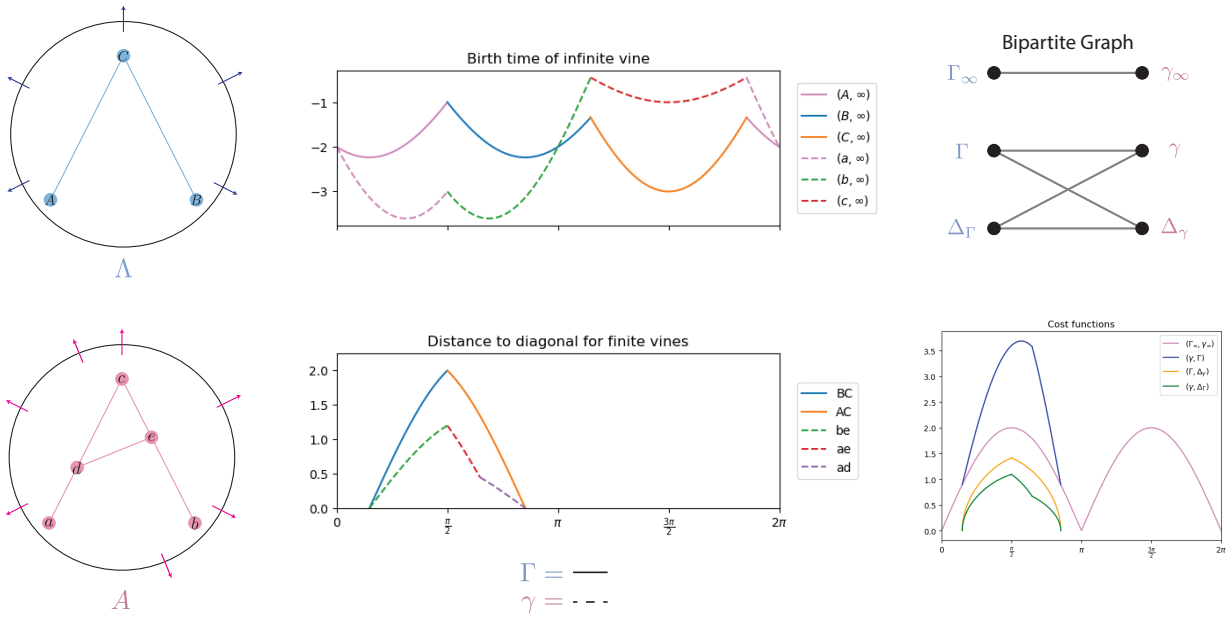


Figure 4.3 An example of the weight function c. The two figures in the middle visualize the behavior of the vines. On the top right is the bipartite graph representation, and on the bottom are the cost functions.

representatives) so that at any fixed ω, this graph is the bipartite graph described in Section 2.1.1. Specifically, writing γ_u for the vine associated to vertex u, the edge weights are given by

$$c(u,v) = \begin{cases} \|\gamma_u(\omega) - \gamma_v(\omega)\|_\infty & \text{if } \gamma_u(\omega) \notin \Delta \text{ or } \gamma_v(\omega) \notin \Delta \\ \|\gamma_u(\omega) - \Delta\|_\infty & \text{if } \gamma_v(\omega) \in \Delta \\ \|\gamma_v(\omega) - \Delta\|_\infty & \text{if } \gamma_u(\omega) \in \Delta \\ 0 & \text{else.} \end{cases}$$

See Figure 4.3 for an example of the weight functions.

Given the above construction, let $n = \max\{\left|\text{ext}^{K_1}(V)\right|, \left|\text{ext}^{K_2}(V)\right|\}$, we have at most $2n$ nodes in each vertex set U and V and thus at most $4n^2$ edges in G. Therefore, the kinetic heap hourglass maintains $4n^2$ elements, resulting in a complexity of $O((4n^2)^3 \log(4n^2)) = O(n^6 \log n)$.

4.4 Conclusions

In this chapter, we introduced the *kinetic hourglass*, a novel kinetic data structure designed for maintaining the bottleneck distance for graphs with continuously changing edge weights. The

structure incorporates two kinetic priority queues, which can be either the deterministic kinetic heap or the randomized kinetic hanger. Both versions are straightforward to implement.

In the future, we hope to improve the runtime for this data structure. In particular, the augmenting path search requires $O(n)$ time, falling short of the efficiency goals in the kinetic data structure (KDS) framework. Moreover, when comparing PHTs with n vertices, the kinetic hourglass holds n^2 elements, which can be computationally expensive. This method can immediately be extended to study the extended persistent homology transform (XPHT) to compare objects that have different underlying topologies [74], however there is much to be understood for the structure of the vines in the PHT and how this particular structure can deal with monodromy. Further, the nature of this data structure means that it is not immediately extendable to the Wasserstein distance case, however, we would like to build a modified version that will work for that case, see discussion in Section 5.1. Finally, since the kinetic hourglass data structure also has the potential to compare more general vineyards that extend beyond the PHT, it will be interesting in the future to find further applications.

CHAPTER 5

FUTURE WORK AND CONCLUSION

5.1 KDS for the Wasserstein Distance

In the previous chapters, we discussed how tools from Topological Data Analysis can be used for shape comparison measures, both in terms of developing new distances and improving the computation of existing ones. This chapter explores future directions building on these results.

The bottleneck and Wasserstein distances are both used to compare persistence diagrams, but the Wasserstein distance is often preferred in practical applications [21]. The bottleneck distance captures the largest difference between matched points, making it highly sensitive to outliers. A single noisy point can dominate the distance, even if the rest of the diagram remains similar. In contrast, the Wasserstein distance takes into account the overall distribution of points, leading to a more stable and informative measure.

Another key advantage of the Wasserstein distance is its use in optimization. Many machine learning and statistical applications require differentiable loss functions, but the bottleneck distance is not differentiable. The Wasserstein distance, particularly with entropic regularization, is smooth and can be efficiently approximated with techniques such as Sinkhorn iterations [68]. This makes it useful in deep learning, optimal transport, and statistical inference.

However, the Wasserstein distance is computationally more expensive, requiring solving an optimal transport problem. While exact computation is costly, recent advances in approximation methods make it feasible for large-scale applications [62, 78, 53]. Its ability to capture small changes in topology while being more robust to noise makes it a strong choice for dynamic and high-dimensional settings.

In the context of time-evolving persistence diagrams, the Wasserstein distance provides a continuous notion of change. This motivates the need for an efficient way to maintain the Wasserstein distance dynamically, leading to the idea of extending the kinetic hourglass to a new kinetic data structure.

5.1.1 Shortest Augmenting Path

The *Shortest Augmenting Path (SAP)* algorithm is an efficient approach for solving the *minimum-cost perfect matching problem* in bipartite graphs, which is central to computing the Wasserstein distance. Given two discrete point sets representing probability measures, SAP finds an optimal transport plan by incrementally constructing augmenting paths in a residual network.

Let $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{n} b_j \delta_{y_j}$ be two discrete probability measures supported on finite point sets $\{x_i\}_{i=1}^{n}$ and $\{y_j\}_{j=1}^{n}$ with weights $a_i, b_j \geq 0$ satisfying $\sum_i a_i = \sum_j b_j = 1$. The *Wasserstein distance* of order 1 is given by the solution to the linear program:

$$W_1(\mu, \nu) = \min_{\gamma \in \Pi(\mu,\nu)} \sum_{i,j} c_{ij} \gamma_{ij}$$

where: $\Pi(\mu, \nu)$ is the set of transport plans satisfying the marginal constraints:

$$\sum_j \gamma_{ij} = a_i, \quad \sum_i \gamma_{ij} = b_j, \quad \gamma_{ij} \geq 0$$

and $c_{ij} = d(x_i, y_j)$ is the cost of transporting mass from x_i to y_j.

This problem can be formulated as a minimum-cost flow problem on a bipartite graph $G = (X \cup Y, E)$, where:

- Each node in X corresponds to a source location x_i with supply a_i.
- Each node in Y corresponds to a target location y_j with demand b_j.
- Edges $(i, j) \in E$ have weights c_{ij} representing the transport cost.

The SAP algorithm solves this problem by maintaining a residual network and iteratively finding the shortest augmenting path to improve the matching. It follows these steps:

1. Initialize the Bipartite Graph: Construct the graph $G = (X \cup Y, E)$ where each node in X is initially unmatched, and edges represent feasible transport assignments with cost c_{ij}.
2. Construct the Residual Graph: Define a directed residual graph G_r where:
    - Each edge (i, j) represents an available transport assignment with residual capacity.
    - Backward edges (j, i) track adjustments in the transport flow.
3. Find the Shortest Augmenting Path:

- Use Dijkstra's algorithm or Bellman-Ford to find the shortest path from an unmatched node in X to an unmatched node in Y.
- The cost function is defined as $c_{ij} + \pi_i - \pi_j$, where π_i, π_j are dual potentials maintaining reduced costs.

4. Update the Matching: Augment flow along the shortest path, modifying the residual network.
5. Repeat Until an Optimal Matching is Found:** Continue finding and augmenting paths until all nodes in X are matched to nodes in Y.

The algorithm runs in $\mathcal{O}(n^3)$ time in the worst case but performs efficiently in practice.

5.1.2 Adapting SAP to a Kinetic Data Structure

We discuss ideas of designing a new KDS that maintains an evolving transport plan as point sets $\{x_i(t)\}$ and $\{y_j(t)\}$ move over time. Several key modifications are needed to make SAP kinetic:

- **Handling Continuous Motion:** As points move, edge weights $c_{ij}(t) = d(x_i(t), y_j(t))$ change, affecting the optimal transport plan.
- **Event-Driven Updates:** Instead of recomputing the full transport plan at every time step, update the residual graph only when key events occur:
  - *Edge cost crossing a threshold* (e.g., a shorter transport path appears).
  - *New optimal match forming* (e.g., a better assignment becomes viable).
  - *Node insertion/deletion* (e.g., a point appears or disappears from the dataset).
- **Maintaining an Efficient Residual Graph:** Track augmenting paths incrementally instead of recomputing shortest paths from scratch.
- **Approximate Kinetic Updates:** Introduce tolerance thresholds for minor transport cost changes, avoiding unnecessary recomputation.

By incorporating these modifications, a kinetic version of SAP can maintain Wasserstein distance efficiently in time-varying settings.

5.2 Conclusion

This dissertation explored the use of topological data analysis (TDA) methods for shape analysis, with a particular focus on directional transforms as a tool for encoding geometric and topological

structures. Through the development of novel metrics and computational frameworks, this work contributes to the growing intersection of computational topology, geometry, and data science, addressing challenges in both static and dynamic settings.

The first two chapters provided the necessary foundation: covering homology, persistent homology, merge trees, and kinetic data structures, with a special emphasis on directional transforms, which offer a structured way to analyze shape variations across different orientations. These fundamental concepts set the stage for the core methodological contributions of this dissertation.

In Chapter 3, we introduced the Labeled Merge Tree Transform (LMTT), a novel distance metric that leverages directional transforms and merge trees to compare embedded graphs. The LMTT provides a robust, topology-aware measure of similarity that captures both local and global structural differences. We validated its effectiveness on two real-world datasets and demonstrated its advantages over existing metrics in terms of both stability and discriminative power.

In Chapter 4, we extended the study of dynamic topological structures by developing a kinetic data structure (KDS) for efficiently updating bottleneck distances in time-varying systems. This was applied to the Persistent Homology Transform (PHT), allowing for a more computationally efficient approach to tracking topological changes over time. By reducing redundant computations and leveraging event-driven updates, the proposed KDS framework significantly improves the feasibility of persistence-based shape analysis in evolving datasets.

Finally, in this chapter, we outlined a promising future direction that builds on the foundation laid in this work: extending kinetic approaches beyond the bottleneck distance to the Wasserstein distance, allowing for more flexible and informative dynamic topological comparisons.

The contributions of this dissertation provide both theoretical advancements and practical computational tools for shape analysis using TDA. The methods developed here offer efficient and interpretable ways to compare shapes across different domains and open new avenues for applying topology-driven insights to diverse scientific disciplines. Future work will continue to refine these techniques, explore their computational efficiency, and extend their applicability to dynamic and high-dimensional data settings.

BIBLIOGRAPHY

[1] Pankaj K Agarwal, Mark De Berg, Jie Gao, Leonidas J Guibas, and Sariel Har-Peled. Staying in the middle: Exact and approximate medians in R^1 and R^2 for moving points. In *CCCG*, pages 43–46, 2005.

[2] Pankaj K. Agarwal, Emily Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff distance for metric trees. *ACM Trans. Algorithms*, 14(2), apr 2018.

[3] Pankaj K. Agarwal, Jie Gao, and Leonidas J. Guibas. Kinetic medians and kd-trees. In Rolf Möhring and Rajeev Raman, editors, *Algorithms — ESA 2002*, pages 5–17, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[4] Pankaj K. Agarwal and Micha Sharir. Davenport–schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. North-Holland, Amsterdam, 2000.

[5] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., 1993.

[6] Erik J Amézquita, Michelle Y Quigley, Tim Ophelders, Jacob B Landis, Daniel Koenig, Elizabeth Munch, and Daniel H Chitwood. Measuring hidden phenotype: Quantifying the shape of barley seeds using the Euler characteristic transform. *in silico Plants*, 4(1), 2022.

[7] Elizabeth Munch and. An invitation to the Euler characteristic transform. *The American Mathematical Monthly*, 132(1):15–25, 2025.

[8] Shreya Arya, Barbara Giunti, Abigail Hickok, Lida Kanari, Sarah McGuire, and Katharine Turner. Decomposing the persistent homology transform of star-shaped objects, 2024.

[9] Julien Basch, Leonidas J Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[10] Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, pages 469–471, New York, NY, USA, 1997. Association for Computing Machinery.

[11] Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Reporting red—blue intersections between two sets of connected line segments. *Algorithmica*, 35(1):1–20, Jan 2003.

[12] Levent Batakci, Abigail Branson, Bryan Castillo, Candace Todd, Erin Wolf Chambers, and Elizabeth Munch. Comparing embedded graphs using average branching distance. *Involve, a Journal of Mathematics*, 16(3):365–388, August 2023.

[13] Katherine Benjamin et al. Multiscale topology classifies cells in subcellular spatial transcriptomics. *Nature*, 630(8018):943–949, Jun 2024.

[14] Dimitri P Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper*, 1, 1979.

[15] Dimitri P Bertsekas. Linear network optimization: algorithms and codes. *MIT Press, Cambridge, MA*, 1991.

[16] Dimitri P Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66, 1992.

[17] Håvard Bakke Bjerkevik and Magnus Bakke Botnan. Computational Complexity of the Interleaving Distance. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, New York, 2005.

[19] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[20] Maike Buchin, Erin Chambers, Pan Fang, Brittany Terese Fasy, Ellen Gasparovic, Elizabeth Munch, and Carola Wenk. Distances between immersed graphs: Metric properties. *La Matematica*, Jan 2023.

[21] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012.

[22] L. Paul Chew and Klara Kedem. Improvements on geometric pattern matching problems. In Otto Nurmi and Esko Ukkonen, editors, *Algorithm Theory — SWAT '92*, pages 318–325, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[23] Daniel H Chitwood and Wagner C Otoni. Morphometric analysis of Passiflora leaves: the relationship between landmarks of the vasculature and elliptical fourier descriptors of the blade. *Gigasciendce*, 6(1):1–13, January 2017.

[24] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, Jan 2007.

[25] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz functions have lp-stable persistence. *Foundations of Computational Mathematics*, 10(2):127–139, Apr 2010.

[26] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 119–126, New York, NY, USA, 2006. Association for Computing Machinery.

[27] Michael A. A. Cox and Trevor F. Cox. *Multidimensional Scaling*, pages 315–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] Justin Curry, Sayan Mukherjee, and Katharine Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 2018.

[29] Guilherme D. da Fonseca and Celina M.H. de Figueiredo. Kinetic heap-ordered trees: Tight analysis and improved algorithms. *Information Processing Letters*, 85(3):165–169, 2003.

[30] Guilherme D. da Fonseca, Celina M.H. de Figueiredo, and Paulo C.P. Carvalho. Kinetic hanger. *Information Processing Letters*, 89(3):151–157, 2004.

[31] Vin de Silva, Elizabeth Munch, and Anastasios Stefanou. Theory of interleavings on categories with a flow. *Theory and Applications of Categories*, 33(21):583–607, 2018.

[32] Tamal K. Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, Cambridge, England, 2022.

[33] Tamal K. Dey and Cheng Xin. Computing Bottleneck Distance for 2-D Interval Decomposable Modules. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[34] Paweł Dłotko and Davide Gurnari. Euler characteristic curves and profiles: a stable shape invariant for big data problems. *GigaScience*, 12, 2023.

[35] D C Dowson and B V Landau. The Fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, 1982.

[36] Ian L. Dryden and Kanti V. Mardia. *Statistical Shape Analysis, with Applications in R*. Wiley, September 2016.

[37] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction.* American Mathematical Society, 2010.

[38] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[39] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, Sep 2001.

[40] Ellen Gasparovic, Elizabeth Munch, Steve Oudot, Katharine Turner, Bei Wang, and Yusu Wang. Intrinsic interleaving distance for merge trees. *La Matematica*, Dec 2024.

[41] Jamine George, Oscar Lledo Osborn, Elizabeth Munch, Messiah Ridgley, and Elena Xinyi Wang. On the stability of the Euler characteristic transform. *In preparation*, 2025.

[42] Robert Ghrist, Rachel Levanger, and Huy Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, Oct 2018.

[43] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[44] Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Toth, editors. *Handbook of discrete and computational geometry*. Discrete mathematics and its applications. CRC Press, Boca Raton, third edition edition, 2018.

[45] Leonidas J. Guibas. Kinetic data structures: a state of the art report. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, page 191–209, USA, 1998. A. K. Peters, Ltd.

[46] Leonidas J Guibas and Marcel Roeloffzen. Modeling motion. In Csba D. Toth, Joseph O'Rourke, and Jacob E. Goodman, editors, *Handbook of Discrete and Computational Geometry (3rd ed.)*, chapter 57, pages 1117 – 1134. Chapman and Hall/CRC, 2017.

[47] Aric Hagberg, Pieter Swart, and Daniel Chult. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, 06 2008.

[48] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.

[49] Paul R. Halmos and L. J. Savage. Application of the Radon-Nikodym Theorem to the Theory of Sufficient Statistics. *The Annals of Mathematical Statistics*, 20(2):225–241, 1949.

[50] Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.

[51] Abigail Hickok. Persistence diagram bundles: A multidimensional generalization of vineyards, 2023.

[52] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, Dec 1973.

[53] Guillaume Houry, Han Bao, Han Zhao, and Makoto Yamada. Fast 1-Wasserstein distance approximations using greedy strategies. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, pages 325–333, 2024.

[54] Nicholas Jardine and Robin Sibson. *Mathematical Taxonomy*. Wiley series in probability and mathematical statistics. Wiley, 1971.

[55] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. *ACM J. Exp. Algorithmics*, 22, Sep 2017.

[56] Woojin Kim and Facundo Mémoli. Spatiotemporal persistent homology for dynamic metric spaces. *Discrete & Computational Geometry*, 66(3):831–875, Oct 2021.

[57] Victor Klee and George Minty. How good is the simplex algorithm? *Inequalities*, 3:159–175, 1972.

[58] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[59] Kanti V. Mardia, John T. Kent, and John M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[60] Dmitriy Morozov, Kenes Beketayev, and Gunther Weber. Interleaving distance between merge trees. In *Proceedings of TopoInVis*, 2013.

[61] Elizabeth Munch and Anastasios Stefanou. The l_∞-cophenetic metric for phylogenetic trees as an interleaving distance. In Ellen Gasparovic and Carlotta Domeniconi, editors, *Research in Data Science*, pages 109–127. Springer International Publishing, Cham, 2019.

[62] Sloan Nietert, Ziv Goldfeld, and Kengo Kato. Smooth p-Wasserstein distance: Structure, empirical approximation, and statistical applications. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8172–8183, 2021.

[63] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[64] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 435–446. Springer, 2011.

[65] Kaspar Riesen and Horst Bunke. *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning*, pages 287–297. Springer Berlin Heidelberg, 2008.

[66] R. Tyrrell Rockafellar. *Convex Analysis*, volume 28 of *Princeton Mathematical Series*. Princeton University Press, 1970.

[67] Filippo Santambrogio. *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*, volume 87 of *Progress in Nonlinear Differential Equations and Their Applications*. Birkhäuser, 2015.

[68] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[69] Primoz Skraba and Katharine Turner. Wasserstein stability for persistence diagrams, 2023.

[70] Dmitriy Smirnov and Dmitriy Morozov. Triplet merge trees. In Hamish Carr, Issei Fujishiro, Filip Sadlo, and Shigeo Takahashi, editors, *Topological Methods in Data Analysis and Visualization V*, pages 19–36, Cham, 2020. Springer International Publishing.

[71] Robert Endre Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, October 1979.

[72] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[73] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, Dec 2014.

[74] Katharine Turner, Vanessa Robins, and James Morgan. The extended persistent homology transform of manifolds with boundary. *Journal of Applied and Computational Topology*, May 2024.

[75] Sarah Tymochko, Elizabeth Munch, Jason Dunion, Kristen Corbosiero, and Ryan Torn. Using Persistent Homology to Quantify a Diurnal Cycle in Hurricanes. *Pattern Recognition Letters*, 133:137–143, 2020.

[76] Cédric Villani. *Optimal Transport: Old and New*, volume 338 of *Grundlehren der mathematischen Wissenschaften*. Springer, 2008.

[77] Lu Xian, Henry Adams, Chad M. Topaz, and Lori Ziegelmeier. Capturing dynamics of time-varying data via topology. *Foundations of Data Science*, 4(1):1–36, 2022.

[78] Makoto Yamada, Yuki Takezawa, Ryoma Sato, Han Bao, Zornitsa Kozareva, and Sujith Ravi. Approximating 1-Wasserstein distance with trees. *arXiv preprint arXiv:2206.12116*, 2022.

[79] Lin Yan, Talha Bin Masood, Raghavendra Sridharamurthy, Farhan Rasheed, Vijay Natarajan, Ingrid Hotz, and Bei Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Computer Graphics Forum*, 40(3):599–633, Jun 2021.

[80] Lin Yan, Yusu Wang, Elizabeth Munch, Ellen Gasparovic, and Bei Wang. A structural average of labeled merge trees for uncertainty visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):832–842, January 2020.